Evaluierung web-basierter 3D-Technologien

Masterarbeit

vorgelegt von

Markus Bader

Institut für Wissensmanagement
Technische Universität Graz
A-8010 Graz

Betreuer: Dipl.-Ing. Dr. tech. Michael Granitzer
Begutachter: Univ.-Prof. Dr. rer. nat. Klaus Tochtermann

Mai 2010

Abstract

People are confronted with an ever increasing amount of information each day, therefore meaningful information visualisations become more and more important. High quality 3D contents can be found on the web more often, due to faster computer hardware and higher internet bandwidth. So information visualisations on the web provide an opportunity to reach a larger audience. A third dimension in visualisations can help to structure the information in a better way. Creating 3D visualisations and publishing them on the web need formats that can satisfy the high technological requirements of information visualisations. Therefore in this thesis current available 3D web formats are listed and evaluated to meet the criteria for such visualisations. The high penetration of Flash is the decisive factor for implementing information visualisations with this format. Hence there is more than one 3D engine available for Flash, it is necessary to choose one and therefore a detailed comparison between them is made. This thesis shows that implementing information visualisations can be done in Flash, however, you have to compromise speed and imprecise depth calculations due to weak hardware acceleration support.

Kurzfassung

Aufgrund der wachsenden Anzahl von Informationen, die ständig zu verarbeiten sind, werden aussagekräftige Visualisierungen von Informationen immer wichtiger. Gleichzeitig finden auch durch die ständig schneller werdende Computerhardware und die größer werdende Bandbreite bei Internetzugängen anspruchsvolle dreidimensionale Inhalte im Web immer stärkere Verbreitung. Die Darstellung von Informationsvisualisierungen im Web ist somit eine gute Möglichkeit, um viele Nutzer zu erreichen. Die Erweiterung von zweidimensionalen Visualisierungen um eine weitere Dimension kann hierbei zur besseren Strukturierung der Informationen genutzt werden. Die dreidimensionale Darstellung von Informationen im Web verlangt aber auch nach entsprechenden Technologien, die diese Aufgabe erfüllen können. Somit werden in dieser Arbeit derzeit verfügbare web-basierte 3D-Formate ermittelt, diese anhand von Kriterien verglichen und es wird auf Grund der hohen Verbreitung Flash für eine prototypische Umsetzung einer Informationsvisualisierung ausgewählt. Da es in Flash mehrere 3D-Engines gibt, werden diese einer genaueren Untersuchung unterzogen, um für die Informationsvisualisierung die passende auswählen zu können. Die Arbeit zeigt, dass es mit Flash möglich ist eine Informationsvisualisierung umzusetzen, auch wenn dabei aufgrund der geringen Hardwareunterstützung oft Kompromisse bei der Geschwindigkeit und der ungenauen Tiefenberechnung einzugehen sind.

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..

(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..

date (signature)

Danksagung

Hiermit möchte ich mich bei allen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben. Ein besonderer Dank geht an Herrn Dr. Michael Granitzer, der durch seine Ratschläge wesentlich zum Gelingen dieser Arbeit beigetragen hat, sowie an Herrn Wolfgang Kienreich, der für mich während dieser Zeit ein wichtiger Ansprechpartner war.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Die Menge an Informationen, mit der heutzutage jeder konfrontiert wird, ist in letzter Zeit immer weiter gestiegen. Dabei ist es nicht einfach die Vielzahl an Informationen zu verarbeiten. Übersichtliche und leicht verständliche Visualisierungen der Informationen können hier helfen. Um eine große Anzahl an Nutzern erreichen zu können, bietet sich das Web an. Dabei gibt es die Möglichkeit zwei- und dreidimensionale Darstellungen zu verwenden, wobei eine zusätzliche dritte Dimension sich gut zur besseren Strukturierung eignet. Dreidimensionale Darstellungen in entsprechender Qualität verlangen jedoch nach leistungsfähigen Technologien. Aufgrund der hohen Ansprüche und der Vielzahl an Formaten, die derzeit für das Web angeboten werden, besteht die Notwendigkeit diese zu evaluieren, um das richtige Format für die Erstellung einer Informationsvisualisierung auszuwählen.

1.1. Von der realen zur virtuellen Welt

Um dreidimensionale Inhalte betrachten zu können, müssen diese erst erstellt werden. Dabei wird, meist abgeleitet von Bestandteilen der realen Welt, durch Modellierung eine dreidimensionale virtuelle Welt erzeugt. In Abbildung 1 wird der Entstehungsprozess einer virtuellen Welt und deren Wechselwirkungen mit der realen Welt skizziert.

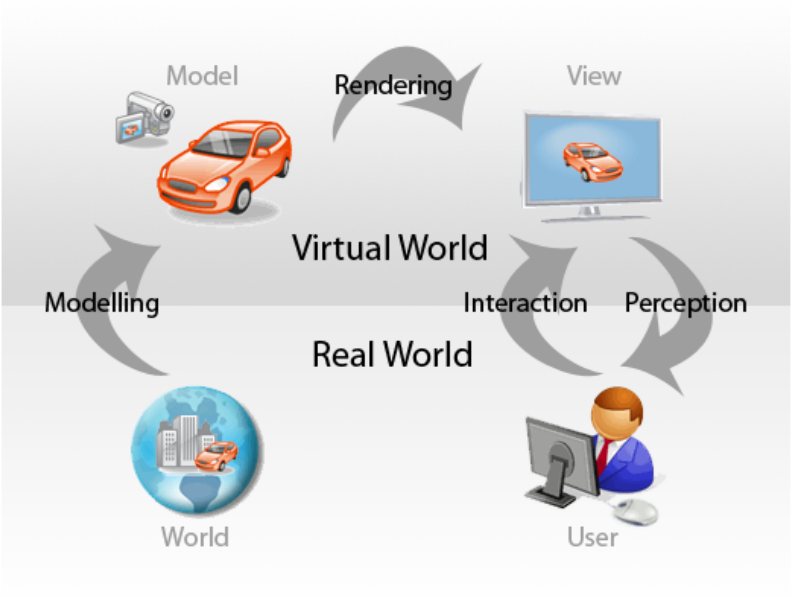


Abbildung 1: Rendering

Mit Hilfe von zB CAD-Anwendungen werden reale Elemente in eine virtuelle Welt transformiert, wo sie als Modelle oder Objekte bezeichnet werden. Modelle setzen sich aus Punkten, Linien und Dreiecken oder auch aus anderen Objekten zusammen. Eine Ansammlung von Objekten wird als Szene bezeichnet. Sie enthält neben den Modellen zB auch Materialien, die den Objekten zugeordnet werden sowie Lichtquellen und Kameras, durch die die Szene betrachtet werden kann (vgl. [AKHAHO08], S. 8).

Das Ziel des Renderns ist es, die Modelle auf einem Ausgabegerät, zB einem Bildschirm, anzuzeigen. Ein User in der realen Welt kann mit Eingabegeräten, wie der Maus, mit den virtuellen Modellen interagieren. Als Reaktion wird die Veränderung, die bewirkt wurde, auf dem Ausgabegerät angezeigt. Diese Veränderung wird in Echtzeit durchgeführt, sodass der User vom Prozess des Renderns selbst nichts merkt.

Die „Rendering Pipeline“ untergliedert das Rendern in weitere Bereiche, welche im Kapitel „3D-Grundlagen“ behandelt werden.

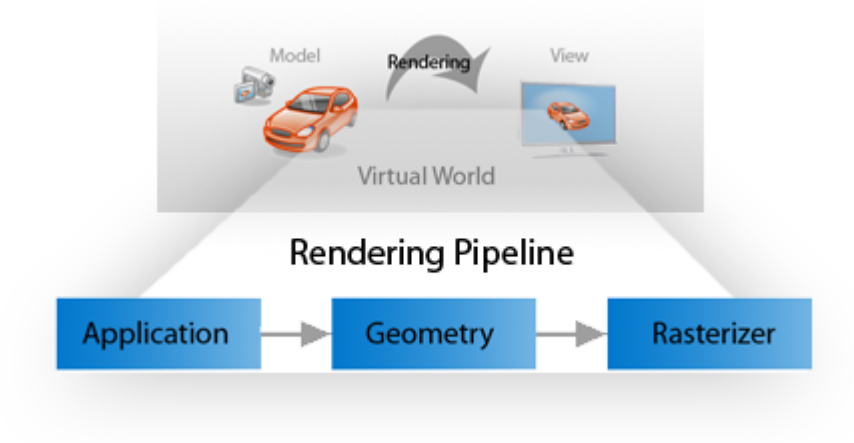


Abbildung 2: Rendering Pipeline

Mit „3D“ werden mehrere Begriffe in Verbindung gebracht, die auch nicht immer eindeutig voneinander unterschieden werden können. Darunter fallen zB die nachstehenden, in dieser Arbeit angepassten, die Raimund Dachselt in seiner Dissertation identifiziert hat (vgl. [DACHSELT06], S. 9ff):

Virtual Reality

Darunter wird eine Computersimulation verstanden, die ein Bild einer Welt kreiert, welches die gleichen Sinneseindrücke erweckt, wie die reale Welt. Sie erfasst die Position und Aktionen einer Person und gibt entsprechendes Feedback. Da dies alles in Echtzeit zu erfolgen hat, werden große Ansprüche an die Hardware gestellt (vgl. [CRSHWI09], S. 1, S. 10).

Augmented Reality

Dabei wird dem User eine modifizierte Sicht auf die reale Welt geboten. Es werden Zusatzinformationen für bestimmte Sinne gewährt (vgl. [CRSHWI09], S. 2f).

Desktop-VR

Bei Desktop-VR erfolgt die Ausführung auf Standardcomputern. Spezielle Ein- und Ausgabegeräte sind hierbei nicht notwendig, aber auch nicht ausgeschlossen. Ein Eintauchen in die virtuelle Realität durch die Stimulation von unterschiedlichen Sinnen ist damit nur in geringem Umfang, zB durch die Verwendung von 3D-Brillen, möglich. Desktop-VR ist eher für einen allgemeinen Einsatz gedacht.

Mobile3D

Unter diese Kategorie fallen 3D-Anwendungen, die auf mobilen Endgeräten ausgeführt werden können. Die Anforderungen an die bei diesen Geräten oft nicht so leistungsfähige Hardware müssen eher gering sein. Auch hier werden keine besonderen Ein- und Ausgabegeräte benötigt.

Web3D

Das Web selbst ist ein Synonym für „World Wide Web“ und stellt laut dem „WordNet®“ von Princeton ein Computernetzwerk dar, welches eine Ansammlung von Internetseiten bestehend aus Text, Grafiken und anderen multimedialen Elementen über das Hypertext Transferprotokoll (HTTP) zur Verfügung stellt. Das Hypertext Transferprotokoll selbst wird in WordNet® als ein Protokoll zur Übertragung von Anfragen und Informationen zwischen Servern und Browsern beschrieben. Die Abgrenzung zum Internet besteht darin, dass das Internet ein weltweites Computernetzwerk ist, welches es ermöglicht Daten, darunter auch Inhalte des Webs, zwischen den einzelnen Computern auszutauschen (vgl. [WORDNET09]).

In dieser Arbeit werden nur 3D-Technologien untersucht, die in einem (Web-) Browser, zB über Browser-Plugins, dargestellt werden können. Diese Arbeit grenzt sich von anderen Möglichkeiten ab, bei denen es nötig ist, sich eine eigene Anwendung herunter zu laden und zu installieren, um 3D-Inhalte betrachten zu können, wozu kein (Web-) Browser nötig ist. Die Grenzen der oben genannten Begriffe können dabei aber nicht immer genau gezogen werden, da Web3D auch auf mobilen Endgeräten, auf denen ein Webbrowser läuft, ausgeführt werden kann.

Im Rahmen dieser Arbeit wird der Begriff Technologie generell als Überbegriff für Formate und Engines sowie ebenfalls im Zusammenhang mit verfügbarer Hardware verwendet. Als Format im Speziellen werden Spezifikationen für die Darstellung dreidimensionaler Inhalte verstanden und als Engines werden die unterschiedlichen softwaretechnischen Implementierungen dieser Spezifikationen bezeichnet.

Verschiedene Einsatzbereiche von Web3D sind zB (vgl. [WABO01], S. 15ff):

- **Produkt- und Datenvisualisierung**
  Web3D bietet neue Möglichkeiten ein Produkt darzustellen wie es zweidimensionale Bilder und Text alleine nicht schaffen. Das Produkt kann aus allen Blickwinkeln betrachtet sowie vergrößert und verkleinert werden. Durch das Hinzufügen von Interaktivität ist es denkbar, die Funktionsweise des Produkts zu simulieren. Die dritte Dimension bietet auch im Bereich der Datenvisualisierung neue Möglichkeiten um komplexe und umfangreiche Daten übersichtlicher darzustellen.

- **E-Commerce und Geschäftsanwendungen**
  Die Darstellung und Präsentation von Produkten oder Konzepten in dreidimensionaler Form kann im Geschäftsbereich den Ausschlag geben, ob sie angenommen werden oder nicht.

- **Entertainment**
  Ein großer Teil des Web3D‘s findet sich im Bereich der Unterhaltung wieder. Darunter fallen Spiele, Comics und auch virtuelle Touren.

- **Bereicherung für Webseiten**
  Im Bereich Design können Darstellungen in 3D die Webseite erheblich aufwerten. Interaktionen mit der dreidimensionalen Darstellung werden möglich.

- **Nachrichten und Werbung**
  Bei Nachrichten können komplexe Sachverhalte einfacher dargestellt werden und dreidimensionale Visualisierungen können auch Werbung für ein Produkt erleichtern.

1.2. Motivation und Ziele

Ein wichtiger der oben aufgezählten Bereiche ist die Daten- oder auch Informationsvisualisierung. Der Bedarf an Informationsvisualisierungen nimmt durch die steigende Menge an Informationen, mit der heutzutage jeder konfrontiert wird, ständig zu. Waren früher noch eine kleine Auswahl von Quellen zur Befriedigung des individuellen Informationsbedürfnisses ausreichend, kann dies heute durch die wachsende Informationsvielfalt nicht mehr so einfach garantiert werden. Auch die Komplexität der Informationen wächst und es wird immer schwerer sie zu erfassen. Zur Verarbeitung der Vielzahl an Informationen existieren bereits automatisierte Methoden, wobei jedoch die abstrakten Resultate dieser für viele noch immer schwer zu interpretieren sind. Somit besteht die Notwendigkeit solche Ergebnisse übersichtlich und leicht verständlich aufzubereiten. Dabei kommt die Wichtigkeit von Informationsvisualisierungen zum Ausdruck, da Visualisierungen generell das allgemeine Verständnis unterstützen können. Hier können zweidimensionale oder auch dreidimensionale Visualisierungen verwendet werden, wobei vor allem durch die Zuhilfenahme einer dritten Dimension eine bessere Strukturierung der Information möglich ist.

Gleichzeitig haben immer mehr Anwender Zugang zum Web, wo derzeit zahlreiche verschiedene 3D-Technologien existieren. So gibt es für Entwickler eine große Auswahl an Formaten, die in Frage kommen, um dreidimensionale Inhalte darzustellen.

Viele Formate brauchen dazu ein eigenes Browser-Plugin, welches, wenn es nicht bereits vorhanden ist, vorher heruntergeladen und installiert werden muss. Für die Darstellung und das Rendern komplexer dreidimensionaler Inhalte in Echtzeit wird auch entsprechende Hardware, wie zB eine 3D-fähige Grafikkarte, benötigt. Dies kann bei den meisten Anwendern nicht standardmäßig vorausgesetzt werden, obwohl durch den technologischen Fortschritt die Verbreitung dieser Hardware zunimmt.

Somit stellt sich die Frage, ob und wie das Web für die dreidimensionale Darstellung von Informationsvisualisierungen geeignet ist. Das führt zu den Zielen dieser Arbeit, welche nachstehend aufgelistet sind:

1. Evaluierung derzeit verfügbarer web-basierter 3D-Technologien und Auswahl einer für die Erstellung einer dreidimensionalen Informationsvisualisierung
2. Implementierung einer Informationsvisualisierung und Analyse der Ergebnisse

Durch den Vergleich von 3D-Technologien wird ein Überblick geboten, welcher die Auswahl eines passenden Formats in Hinblick auf die Erstellung einer Informationsvisualisierung erleichtert. Dieses Format muss die für diesen Bereich wichtigen Kriterien, wie einen entsprechenden Funktionsumfang und eine möglichst große Zielgruppe, erfüllen. Für Informationsvisualisierungen ist zB eine realitätsnahe Darstellung im Allgemeinen nicht von entscheidender Bedeutung. Um dreidimensionale Visualisierungen im Browser in guter Qualität darstellen zu können, sind entsprechend leistungsfähige Technologien notwendig, welche zB die Grafikkarte zur Beschleunigung von 3D-Inhalten nutzen. Davon profitiert natürlich auch die Darstellung von zweidimensionalen Inhalten, da 2D im Prinzip nur ein spezieller Fall von 3D ist und somit 2D-Inhalte auch durch 3D-Engines dargestellt werden können.

Da es für ein ausgewähltes Format mehrere Möglichkeiten der Umsetzung, wie zB unterschiedliche Engines, geben kann, müssen in diesem Fall die vorhandenen Engines ermittelt und wiederum mittels bestimmter Kriterien verglichen werden. Dadurch soll hauptsächlich auf funktionaler Ebene ein detaillierter Überblick über diese Engines gegeben werden. Aufgrund des Vergleichs wird für die Umsetzung einer Informationsvisualisierung eine geeignete Engine ausgewählt, wobei vor allem Wert auf die benötigten Funktionen und die Einfachheit der Erstellung eines Prototyps gelegt wird.

Ein weiteres Ziel ist die prototypische Implementierung einer Informationsvisualisierung mit der ausgewählten Engine und die Analyse der Ergebnisse. Es soll dafür die Vorgehensweise bei der Umsetzung beschrieben und auf wichtige Faktoren, die dabei zu beachten sind, hingewiesen werden. So wird ein Überblick über die Probleme, aber auch über die Vorteile, welche das ausgewählte Format bzw. die verwendete Engine bietet, gegeben. Das Ergebnis wird analysiert und darauf basierend ein Vergleich zu den anderen nicht ausgewählten Formaten bzw. Engines gezogen. Dabei wird ebenso beleuchtet, ob eine andere Auswahl dieselben Vor- und Nachteile ergeben hätte.

1.3. Nutzen

Die vorliegende Arbeit gibt unter anderem Entwicklern im Bereich der Informationsvisualisierungen einen Überblick über wichtige, derzeit verfügbare webbasierte 3D-Technologien. Durch die Integration von dreidimensionalen Informationsvisualisierungen in das Web kann eine größere Anzahl von Benutzern erreicht werden, als mit Offline-Lösungen. Dabei gibt es jedoch auch einiges zu beachten. Durch die beispielhafte Umsetzung einer dreidimensionalen Informationsvisualisierung wird Entwicklern ein Eindruck vermittelt, wofür sich derzeit verfügbare Formate und Engines eignen und welche Probleme dabei auftreten können.

1.4. Gliederung

In Kapitel 2 wird anhand der Rendering Pipeline auf die Grundlagen des Real-Time Renderings eingegangen. Dabei wird vorerst nicht zwischen Online- und Offline- Rendering unterschieden. Essentielle Begriffe und Verfahren des Echtzeit-Renderings werden erläutert.

Darauf folgend wird in Kapitel 3 ein Überblick über die geschichtliche Entwicklung im Gebiet web-basierter 3D-Technologien gegeben. Der Überblick gliedert sich in die drei Bereiche Technologie, Formate und Wirtschaft, wobei sich hier der Bereich Technologie auf die Entwicklung der Grafikkarten mit 3D-Unterstützung konzentriert. Die geschichtliche Entwicklung der Formate zeigt, wann das erste web-basierte 3D-Format verfügbar war und welche Formate bis zum heutigen Zeitpunkt entstanden sind. Der wirtschaftliche Bereich stellt dar, ob die Formate bereits wirtschaftlichen Erfolg verzeichnen konnten oder woran dieser scheiterte.

In Kapitel 4 werden wichtige derzeit verfügbare web-basierte Formate anhand von verschiedenen Kriterien miteinander verglichen und ein Format für die Entwicklung eines Prototyps im Bereich der Informationsvisualisierung ausgewählt.

Kapitel 5 beschreibt das ausgewählte Format genauer und analysiert die für dieses Format verfügbaren Engines anhand von weiteren Kriterien.

Nach der Auswahl einer der untersuchten Engines wird in Kapitel 6 die Implementierung des Prototyps mit dieser Engine beschrieben. Dabei wird auf die Probleme und spezifischen Charakteristika dieser Engine eingegangen.

Eine Zusammenfassung und ein Ausblick in Kapitel 7 beschließen die vorliegende Arbeit.

2. 3D-Grundlagen

Dieses Kapitel beschreibt die Grundlagen des Real-Time Renderings. Dabei werden die wichtigsten Verfahren und Begriffe, die sowohl in der Offline- als auch in der Onlinevisualisierung Gültigkeit haben, vorgestellt. Die Grundlagen werden benötigt, da die Evaluierung der Technologien auf den hier vorgestellten Begriffen und Verfahren basiert. Um eine Auswahl für ein entsprechendes Format finden zu können und mit diesem Format einen Prototyp im Bereich der Informationsvisualisierung zu erstellen, ist die Kenntnis dieser Begriffe und Verfahren bis zu dem nachfolgend beschriebenen Grad Voraussetzung.

Real-Time Rendering bietet, im Gegensatz zu bereits vorgerenderten Szenen, die Möglichkeit mit der Szene zu interagieren und sie je nach Interaktion in Echtzeit zu beeinflussen. Diese Interaktivität ist vor allem für den Bereich der Informationsvisualisierung wichtig. Der Einsatz von vorgerenderten Bildern und das Hinzufügen von simulierter Interaktivität durch Webschnittstellen ist zwar möglich, wie die Verwendung des Appear-Formats vom Know-Center (siehe Kapitel 6.3) zeigt, aber sie nutzen nicht die meist bereits vorhandene, für 3D ausgelegte Hardware und die dafür verfügbaren Engines. Vor allem im Bereich der Informationsvisualisierung ist es nicht unbedingt nötig auf Fotorealismus und somit auf die neuesten Techniken, um diesen zu erreichen, zu setzen. Außerdem sollen Informationsvisualisierungen für viele Nutzer zur Verfügung stehen. Dabei ist jedoch meist nicht festzustellen, welche Hardware sie zur Verfügung haben und somit kann die korrekte Darstellung von aufwändigen Szenen nicht vorausgesetzt werden.

Die Kernkomponente des Real-Time Renderings ist die Rendering Pipeline. Sie besteht aus drei Stufen (Application, Geometry, Rasterizer) und ihre Hauptaufgabe ist es, ein zweidimensionales Bild aus den gesamten, in einer dreidimensionalen Szene vorhandenen Elementen zu erzeugen. Nachstehend werden die einzelnen Stufen und die darin enthaltenen Schritte beschrieben (vgl. [AKHAHO08], S.11ff).

2.1. Application-Phase

In dieser Phase werden alle Berechnungen auf der CPU ausgeführt (Software basierend), was bedeutet, dass der Entwickler die volle Kontrolle über diese Phase hat. Um Geschwindigkeitszuwächse erzielen zu können, sind die Aufgaben auf mehrere Prozessorkerne aufteilbar.

Die Erzeugung der einzelnen Primitive, welche sich aus den Punkten, Linien und Dreiecken zusammensetzen, fällt in diese Phase. Sie werden anschließend zur weiteren Verarbeitung an die Geometry-Phase übergeben. Zu den Aufgaben der Application-Phase gehört auch die Verarbeitung des Inputs von verschiedenen Eingabegeräten, wie der Maus oder der Tastatur. In dieser Phase können zB ebenso Kollisionserkennung, Animationen und physikalische Simulationen durchgeführt werden.

2.1.1. Animationen und Morphing

Beim Morphing wird die Form eines Objekts innerhalb einer bestimmten Zeitspanne verändert. Animationen verändern zB die Position von Objekten in einem gewissen Zeitraum.

2.1.2. Kollisionsüberprüfung

Bei der Kollisionsüberprüfung wird getestet, ob sich zwei oder mehrere Objekte überschneiden. Es existieren unterschiedliche Varianten wie dies festgestellt werden kann. Eine einfache Art der Überprüfung ist es, ein Objekt durch Linien zu approximieren und dann nur diese Linien mit der restlichen Umgebung zu überprüfen. Dies ist für komplexe Szenen meist nicht ausreichend, daher gibt es weitere Algorithmen die zB auf BSP-Bäume aufbauen (vgl. [AKHAHO08], S. 793ff).

2.1.3. Culling-Techniken

Culling entfernt Objekte, die in den weiteren Phasen des Renderns nicht berücksichtigt werden sollen, weil sie zB von anderen Objekten verdeckt werden. Im Folgenden werden unterschiedliche Culling-Techniken vorgestellt (vgl. [AKHAHO08], S. 660ff).

Backface Culling

Bei dieser Art von Culling wird die Rückseite der einzelnen Objekte entfernt, da diese im Normalfall nicht sichtbar ist. Eine weitere Verarbeitung ist somit in den nächsten Phasen wenig sinnvoll.

View Frustum Culling

Der sichtbare Bereich wird durch den pyramidenförmigen View Frustum begrenzt. Der Frustum setzt sich aus sechs Ebenen zusammen. Der vordere Bereich des Frustums wird durch die Near Plane und der hintere Bereich durch die Far Plane begrenzt. Alle Objekte außerhalb dieses Bereichs können entfernt werden. Sich mit dem Frustum überschneidende oder sich innerhalb des Frustums befindliche Objekte werden an die Geometry-Phase weitergeleitet.

Occlusion Culling

Wenn sich in der Szene verschiedene Objekte gegenseitig überdecken, können die nichtsichtbaren Objekte gelöscht werden. Dieses Verfahren wird Occlusion Culling genannt. Backface Culling stellt eine einfache Art des Occlusion Cullings dar.

2.1.4. Level of Detail (LOD)

Die Idee hinter LOD ist, dass Objekte, die sich weit von der Kamera entfernt befinden, nicht mehr mit allen Details dargestellt werden müssen, da es nicht auffällt. Es werden unterschiedliche Detailstufen eines Models generiert und je nach Entfernung dargestellt die entsprechende Stufe (vgl. [AKHAHO08], S. 680f).

2.1.5. Depth of Field

Auf Fotos sind Objekte, welche sich im Fokus befinden, scharf und Objekte außerhalb dieses Bereiches unscharf dargestellt. Dieser Effekt kann auch beim 3D-Rendering angewandt werden (vgl. [AKHAHO08], S. 486f).

2.1.6. Motion Blur

In Filmen gibt es den Effekt, dass schnell bewegende Objekte leicht verschwommen erscheinen. Dies hängt mit den Verschlusszeiten einer realen Kamera zusammen. Da sich viele Betrachter bereits an diesen Effekt gewöhnt haben, wird Motion Blur auch im 3D-Bereich erwartet (vgl. [AKHAHO08], S. 490).

2.1.7. Nebel

Nebel kann in der Szene aus verschiedenen Gründen eingesetzt werden. Zum einen erhöht es den Realismus und zum anderen hilft es dem Betrachter zu bestimmen, wie weit das einzelne Objekt entfernt ist. Nebel kann auch im Zusammenhang mit Culling verwendet werden. Ist ein Objekt nahe der Far Plane und auf Grund des Nebels nicht mehr sichtbar, kann dieser benutzt werden, um die Objekte langsam auszublenden. Ohne Nebel würde deutlich ein Schnitt des Objekts an der Far Plane sichtbar sein (vgl. [AKHAHO08], S. 496).

2.1.8. Billboarding

Wenn ein Polygon immer in Blickrichtung ausgerichtet ist, wird dies Billboarding genannt. Das Polygon zeigt dabei immer in die Kamera. Ändert sich dich Blickrichtung, so ändert sich auch die Orientierung des Polygons. Beispiele für Billboarding können Gras oder Wolken sein (vgl. [AKHAHO08], S. 446f).

2.1.9. Szenengraphen

Um alle Elemente einer Szene userorientiert zu strukturieren, werden Szenengraphen verwendet. Die Strukturierung erfolgt in Form eines Baumes, der Objekte, Texturen, Lichtquellen, Kameras, Koordinaten und weitere in einer Szene vorkommenden Elemente enthält. Er ist ein gerichteter azyklischer Graph, wobei gerichtet bedeutet, dass zwei Knoten durch eine Kante in einer bestimmten Richtung verbunden sind und azyklisch heißt, dass der Baum keine Schleifen enthält. Unterschiedliche Knoten können dabei jedoch auf den gleichen Unterknoten zeigen und darauf zB unterschiedliche Transformationen anwenden, um Mehrfachdefinitionen des Unterknotens zu vermeiden (vgl. [AKHAHO08], S. 658ff).

2.1.10. Transformationen

Um Objekte zu positionieren, zu drehen oder zu skalieren, besitzt jedes Objekt eine 4x4-Matrix, mit Hilfe derer solche Operationen durchgeführt werden können. Mit einer 3x3-Matrix können nur lineare Transformationen, wie Rotationen oder Skalierungen, ausgeführt werden. Für Translationen zum Positionieren des Objekts wird eine 4x4-Matrix benötigt. Dies wird „homogene Notation“ genannt (vgl. [AKHAHO08], S. 905f).

2.2. Geometry-Phase

In dieser Phase werden ein Großteil der Polygon- und Vertex-Berechnungen durchgeführt. Sie wird in fünf weitere Phasen unterteilt:

1. Model- und Viewtransform
2. Vertex Shading
3. Projection
4. Clipping
5. Screen Mapping

2.2.1. Model- und Viewtransform

Bei der Modelltransformation werden die Vertices und Normalen des Modells von Modellkoordinaten in Weltkoordinaten umgerechnet, sodass alle Modelle denselben Koordinatenraum verwenden. Dabei werden nur Modelle, die von der Kamera erfasst werden, berücksichtigt. Die Kamera selbst wird durch Weltkoordinaten und eine Richtung, in die sie blickt, definiert. Die View-Transformation setzt die Kamera an den Koordinatenursprung zB mit Blickrichtung entlang der negativen Z-Achse, wobei die Y-Achse nach oben und die X-Achse nach rechts zeigt. Es werden natürlich auch die einzelnen Modelle entsprechend transformiert. Dies hat den Sinn, die Berechnungen für die nächsten Phasen zu vereinfachen und zu beschleunigen.

Dreidimensionale Kartesische Koordinatensysteme

Eine Unterscheidung der Systeme erfolgt durch die Handregel (vgl. [HEBA04], S. 789f)

- **Rechtshändiges System**
  Wenn mit der rechten Hand die Z-Achse gegriffen wird und die Hand dabei aus Richtung positiver X-Achse kommt, zeigt der Daumen in die positive Z-Richtung.

- **Linkshändiges System**
  Zeigt der Daumen in die positive Z-Richtung, wenn die Z-Achse mit der linken Hand gegriffen wird, handelt es sich um ein linkshändiges System.

2.2.2. Vertex Shading

Da es nicht nur wichtig ist, die Position und die Form der Objekte zu rendern, werden hier die Auswirkungen des Lichts auf das Material des Modells berücksichtigt. Dieser Prozess wird Shading genannt.

Die Berechnungen in der Geometry-Phase beziehen sich nur auf die einzelnen Vertices des Objekts. Für jeden Vertex werden die Daten, die für das Shading nötig sind, wie zB die Position, die Normale oder die Farbe, gespeichert. Das Ergebnis der Berechnung, welches Farben, Vektoren oder Texturkoordinaten sein können, wird an die Rasterizer-Phase weitergegeben. Die Shading-Berechnungen werden meist in Weltkoordinaten durchgeführt.

Lichtquellen
Es gibt unterschiedliche Arten von Lichtquellen (vgl. [AKHA02], S. 67ff):

- **Direktes Licht**
  Direktes Licht simuliert eine Lichtquelle, die im Prinzip unendlich weit entfernt ist. Als Beispiel kann die Sonne gesehen werden. Die Lichtstrahlen haben somit eine parallele Ausrichtung, wenn sie auf das Objekt treffen.

- **Punktlicht**
  Punktlichter haben eine Position im Raum und senden ihre Lichtstrahlen gleichmäßig in jede Richtung von diesem Punkt aus.

- **Spotlicht**
  Sie sind den Punktlichtern ähnlich, aber senden die Strahlen nicht in alle Richtungen, sondern nur innerhalb eines kegelförmigen Bereichs aus. Somit benötigen sie eine Richtungsangabe sowie die Definition eines Winkels, der die Kegelform bestimmt.

Für alle Lichtquellen können normalerweise die Lichtfarbe und bestimmte Intensitätsparameter definiert werden. Die Intensitätsparameter für Licht sind:

- **Ambient**
  Da das Licht, welches ein Objekt trifft, nicht nur direkt von Lichtquellen, sondern auch indirekt durch Reflektion von anderen Gegenständen kommen kann, wird zur Simulation dieses Lichts die Ambient-Komponente verwendet.

- **Diffuse**
  Mit dieser Komponente wird dem Verhalten von matten Oberflächen entsprochen.

- **Specular**
  Der Zweck der Specular-Komponente ist es, die Oberfläche glänzend erscheinen zu lassen. Dies geschieht durch die Darstellung von sehr hellen Bereichen auf der Oberfläche, wodurch auf die Krümmung der Oberfläche sowie auf den Ort der Lichtquellen geschlossen werden kann.

Material

Jedes Material hat ebenso bestimmte Parameter (vgl. [AKHA02], S. 69f):

- Ambient
- Diffuse
- Specular
- Shininess
- Emissive

Die Farbe der Oberfläche, der ein Material zugewiesen ist, wird durch diese Parameter, den Parametern der Lichtquelle, die auf diese Oberfläche scheint, und durch ein Beleuchtungsmodell bestimmt.

Shading

Es wird zwischen drei Shading-Haupttypen unterschieden (vgl. [AKHA02], S. 70ff):

- **Flat Shading**

  Hier wird die Farbe für ein komplettes Dreieck berechnet und die Oberfläche des Dreiecks wird mit der entsprechenden Farbe gefüllt. Die Berechnungen für Flat Shading sind einfach zu implementieren und laufen schnell ab. Flat Shading hat aber den Nachteil, dass es nicht unbedingt real aussieht.

- **Gouraud Shading**

  Beim Gouraud Shading werden die Berechnungen für jeden Vertex eines Dreiecks durchgeführt und die berechneten Farben werden dann über die Fläche des Dreiecks interpoliert. Gouraud Shading ist schnell und bietet eine bessere Qualität als Flat Shading, weshalb es auch in den meisten Grafikkarten implementiert ist. Die Qualität ist jedoch sehr vom Detailgrad der Objekte abhängig.

- **Phong Shading**

  Dabei werden die Shading-Normalen, die bei den Vertices gespeichert sind verwendet, um eine Shading-Normale für jeden Pixel der Dreiecksoberfläche zu interpolieren. Diese Normale wird benutzt, um die Farbe für den bestimmten Pixel zu ermitteln. Die Probleme des Gouraud Shadings wie fehlende Glanzpunkte werden hier vermieden, aber die Berechnungen sind aufwändiger.

Schatten

Schatten stellen in der Szene wichtige Elemente dar, um den Betrachter Informationen über die Lage von Objekten zu geben. Sie werden im Zusammenwirken mit einer Lichtquelle, die den Schatten eines Objekts auf eine Oberfläche wirft, verwendet. Es werden dabei harte und weiche Schatten unterschieden. Punktlichter zB erzeugen harte Schatten, welche nur aus einer totalen Schattenregion bestehen. Weiche Schatten werden durch Volumenlichter erzeugt und setzen sich aus einer totalen Schattenregion (umbra) und einer partiellen Schattenregion (penumbra) zusammen (vgl. [AKHAHO08], S. 331ff). Um Schatten in Echtzeit zu erzeugen gibt es verschiedene Methoden, auf die nachstehend kurz eingegangen wird (vgl. [AKHAHO08], S. 333ff).

Planar Shadows

Hier wird der Schattenwurf eines Objekts auf ebene Flächen berechnet. Die einfachste Art ist dabei ein Schatten, der auf eine Fläche mit Y=0 geworfen wird. Die Projektionsmatrix, mit der die Transformationsmatrix des Objektes, welches den Schatten bilden soll, multipliziert werden muss, sieht folgendermaßen aus, wobei „*l*“ die entsprechende Koordinate des Lichts darstellt:

$$\begin{pmatrix} ly & -lx & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -lz & ly & 0 \\ 0 & -1 & 0 & ly \end{pmatrix}$$

Das projizierte Objekt muss nun mit einer dunklen Farbe gerendert werden. Zu beachten sind auch die Tiefensortierung des Schattens mit der Ebene und die Transparenz, die ein Schatten aufweist. Das Objekt sollte sich dabei zwischen der Lichtquelle und der Ebene befinden, damit die Schatten auf die Ebene geworfen werden können. Auch das Projizieren eines Schattens auf eine schräge Fläche fällt unter diese Kategorie.

Shadows on Curved Surfaces

Um Schatten auf unebene Flächen zu projizieren, können Schattentexturen verwendet werden. Dabei wird der Schatten als Textur dargestellt, die dann auf die unebene Fläche angewandt wird.

Weitere Möglichkeiten Schatten darzustellen sind zB Shadow Volumes und Shadow Maps (vgl. [AKHAHO08], S. 340ff).

2.2.3. Projection

In dieser Phase wird der sichtbare Bereich, die View Volume, in einen Unit Cube mit den Extrempunkten an den Stellen (X: -1,Y: -1,Z: -1) und (X:1, Y:1, Z:1) transformiert. Dafür gibt es zwei Projektionsmethoden, welche üblicherweise verwendet werden:

- **Ortographic** (parallel)
  Dabei bleiben parallele Linien auch nach der Projektion parallel. Die Transformation selbst ist eine Kombination aus Positions- und Größenveränderungen.

- **Perspective**
  Hier werden Objekte, die weiter von der Kamera entfernt liegen, nach der Projektion kleiner dargestellt. Parallele Linien näheren sich am Horizont an.

Nach dieser Phase werden keine Z-Koordinaten im generierten Bild mehr gespeichert. Es erfolgt eine Reduktion von drei auf zwei Dimensionen.

2.2.4. Clipping

Nur die Objekte, die sich auch wirklich im View Volume befinden, werden an die Rasterizer-Phase weitergeleitet. Bei Objekten, die zur Gänze innerhalb des sichtbaren Bereichs positioniert sind, ist in dieser Phase keine Berechnung nötig. Sie werden einfach weitergegeben und Objekte ganz außerhalb der View Volume werden entfernt, da sie nicht sichtbar sind.

Bei Modellen, die sich teilweise innerhalb und außerhalb des View Volumes befinden, müssen die draußen liegenden Vertices durch neue ersetzt werden. Wenn sich zB ein Punkt einer Linie außerhalb befindet, wird dieser durch einen neuen Punkt beim Schnittpunkt von Linie und View Volume ersetzt.

2.2.5. Screen Mapping

Als nächstes werden die X- und Y-Koordinaten jedes Objekts in Screen-Koordinaten umgewandelt. Screen-Koordinaten zusammen mit den Z-Koordinaten werden als Window-Koordinaten bezeichnet. Das Screen Mapping ist im Prinzip eine Größentransformation, die die Koordinaten des Unit Cubes auf die Koordinaten des Ausgabefensters umrechnet.

2.3. Rasterizer-Phase

Das Ziel in der Rasterizer-Phase ist es, die Farben für die einzelnen Pixel des Objekts zu berechnen. Die zweidimensionalen Screen-Koordinaten (jeweils mit dem Z-Wert als Tiefeninformation) und die Shading-Daten für die einzelnen Vertices werden hier in Pixel auf dem Bildschirm umgewandelt.

Die Rasterizer-Phase gliedert sich in weitere Phasen:

- Triangle Setup
- Triangle Traversal
- Pixel Shading
- Merging

2.3.1. Triangle Setup

Beim Triangle Setup werden Daten für die Dreiecksoberflächen berechnet. Diese werden für den nächsten Schritt, die Scan Conversion, und für die Interpolation von Shading-Daten aus der Geometry-Phase verwendet.

2.3.2. Triangle Traversal

Es wird überprüft, welche Pixel sich komplett innerhalb der Dreiecke befinden und welche die Dreiecke schneiden. Für den Teil des Pixels, der ein Dreieck überlappt, wird ein Fragment gebildet. Das Auffinden der Pixel innerhalb eines Dreiecks wird Triangle Traversal oder Scan Conversion genannt.

2.3.3. Pixel Shading

Der Input in dieser Phase sind die interpolierten Shading-Daten und das Resultat sind ein oder mehrere Farben. Triangle Setup und Triangle Traversal werden durch fixe Operationen auf der Hardware durchgeführt, das Pixel Shading selbst jedoch durch die programmierbare GPU.

2.3.4. Texturing

Durch das Verwenden von Texturen kann dem Objekt in der Szene ein realistischeres Aussehen gegeben werden. Texturen können zB Bilder sein, die auf die Oberfläche eines Objekts angewandt werden. Der Texturierungsprozess besteht wiederum aus mehreren Phasen (vgl. [AKHAHO08], S. 147ff, vgl. [AKHA02], S. 117ff).

Projector Function

Als Input wird die Position der Oberfläche im lokalen Objektraum benutzt. Darauf wird nun eine Projector-Funktion angewandt, welche aus den dreidimensionalen Koordinaten zweidimensionale berechnet. Die (X,Y,Z) Werte eines Punkts auf dem Objekt werden dadurch in die Texturkoordinaten (U,V), welche im Wertebereich zwischen 0 und 1 liegen, geändert.

Corresponder Function

Diese Funktion verwendet die (U,V) Koordinaten, auch Parameter-Space-Koordinaten genannt, und konvertiert sie zu Textur-Space-Koordinaten. Sie werden mit der Auflösung des Bildes multipliziert, um für die gesuchte Position auf dem Objekt die richtigen Koordinaten auf dem Bild zu finden.

Danach werden mit diesen Koordinaten die Texturwerte ermittelt. Bei Bildern werden sie als Texel bezeichnet. Sie bestehen zB aus RGB Farbwerten, welche die Farben auf dem Objekt ersetzen oder modifizieren. Das Ergebnis wird als Diffuse Color der Oberfläche beim Shading verwendet.

Image Texturing

Ist die Oberfläche des Objekts größer (Magnification) oder kleiner (Minification) als die zu verwendende Textur, muss umgerechnet werden.

Magnification

Sollte zB für eine Oberfläche, welche aus 100x100 Pixel besteht, ein Bild von 50x50 Texel benutzt werden, muss die Textur vergrößert werden. Die gebräuchlichsten Algorithmen dafür sind Nearest Neighbor und Bilinear Interpolation. Bei Nearest Neighbor wird der dem Pixel am nächsten liegende Texel verwendet. Dadurch wirkt die Textur jedoch oft sehr verpixelt. Der Algorithmus der bilinearen Interpolation wiederum nutzt die vier, dem Pixel am nächsten liegenden Texel und interpoliert diese.

Minification

Ist die Textur zu groß, kommen verschiedene Texel für einen Pixel in Frage. Auch hier können Nearest Neighbor und Bilinear Interpolation verwendet werden. Bei Nearest Neighbor wird der der Pixelmitte am nächsten liegende Texel verwendet. Die Bilineare Interpolation verwendet statt einem vier Texel und interpoliert sie. Ein Problem dieser beiden Algorithmen ist, dass durch die Auswahl von nur einem oder eben auch vier bestimmten Texel Aliasing auftreten kann. Um Aliasing zu vermeiden, gibt es einige Methoden. Eine der wichtigsten ist das Mipmapping. Bei diesem Verfahren werden bereits vor dem Rendern kleinere Versionen der originalen Textur gespeichert.

2.3.5. Texturierungsmethoden

Alpha Mapping

Um bestimmte Bereiche der Textur als durchsichtig zu definieren, kann einem Texel ein Alphawert von 0 zugewiesen werden (vgl. [AKHAHO08], S. 181f).

Bump Mapping

Durch Bump Mapping werden Unebenheiten auf eigentlich glatten Flächen erzeugt, um ihnen ein realistischeres Aussehen zu geben. Bei dieser Methode wird für den Shading-Prozess anstatt der echten eine leicht veränderte Oberflächennormale verwendet. Anstatt eine Textur zu verwenden, um einen Farbwert anzupassen, wird sie benutzt, um die Oberflächennormalen zu ändern. Die eigentliche Geometrie des Objekts ändert sich dabei jedoch nicht.

Es gibt unterschiedliche Arten wie eine Textur verwendet werden kann, um die Normale der Oberfläche zu beeinflussen.

Bei einer Offset Map werden in der Textur zwei Werte gespeichert, die senkrecht zur Oberflächennormalen stehen. Sie bestimmen, in U- und V-Richtung der Textur, die Änderung der Normalen. Ein Hightfield ist ein Graustufenbild, wo Weiß einen hohen und Schwarz einen niedrigen Bereich bedeutet. Auch hier werden aus den entstehenden Neigungen die entsprechenden Abweichungen in U-(Spalten) und V-(Reihen)Richtung ermittelt.

In Hardwareimplementierungen wird eine sogenannte „Normal Map“, auch als „dot product bump map“ bezeichnet, verwendet. Hier wird die veränderte Normale direkt gespeichert und nicht mehr durch den Offset, der sich aus zwei Werten ergibt, ermittelt. Jeder Farbkanal stellt dabei direkt eine Oberflächenkoordinate der Normalen dar. Der rote Kanal ist die X-, der grüne Kanal die Y- und der blaue Kanal die Z-Richtung (vgl. [AKHAHO08], S. 183ff).

Environment Mapping

Da die Berechnungen, um echte Reflektionen zu erzeugen, sehr komplex und somit auch langsam sind, werden einfachere Methoden bevorzugt. Eine Environment Map ist ein Abbild der Umgebung, welches auf ein Objekt gemappt wird, um den Eindruck zu erzeugen, dass es sich um Reflexionen der Umgebung handelt.

Es wird unter anderem zwischen Sphere Mapping, wo die Textur, die die Umgebung darstellt, die Form einer Kugel hat und Cubic Environment Mapping, bei dem die Umgebung so abgebildet wird, dass die resultierende Textur als Würfel dargestellt werden kann, unterschieden (vgl. [AKHA02], S. 153ff).

Light Maps

Da die Berechnungen von Beleuchtungseffekten in Echtzeit sehr aufwändig sind, besteht die Möglichkeit, diese bereits vorher zu berechnen und in der Textur zu berücksichtigen. Solche Texturen werden auch Light Maps genannt. Diese haben jedoch die Einschränkung, dass die Szene sowie die Lichtquellen statisch sein müssen (vgl. [AKHAHO08], S. 417f).

2.3.6. Merging

Die Farbinformation jedes Pixels wird als Array von Farben (Rot, Grün, Blau) im Color Buffer gespeichert. Durch das Merging wird die Fragmentfarbe, welche in der Shading-Phase berechnet wird, mit der Farbe, die sich derzeit im Buffer befindet, kombiniert.

Es wird auch die Sichtbarkeit der einzelnen Objekte überprüft. So sollten sich nach dem Rendern alle Farben der Objekte, die derzeit sichtbar sind, im Color Buffer befinden. Für diesen Vorgang ist der Z-Buffer verantwortlich. Er hat dieselbe Größe und Form wie der Color Buffer und speichert für jeden Pixel den Z-Wert, von der Kamera zu dem der Kamera am nächsten liegenden Objekts. Zusammen mit dem Color Buffer wird der Alpha Kanal gespeichert, er enthält für jeden Pixel die Transparenz.

Der Stencil Buffer speichert die Positionen der gerenderten Primitive. Der Inhalt des Stencil Buffers kann dann verwendet werden, um das weitere Rendern in den Color- und Z-Buffer zu steuern. Er dient hauptsächlich der Erzeugung von Spezialeffekten. Das heißt, wenn der Stencil Buffer zB einen gefüllten Kreis enthält, kann dieser dazu genutzt werden, um nachfolgende Objekte nur dort in den Color Buffer zu schreiben, wo sich der Kreis befindet. Der Frame Buffer bezeichnet eigentlich alle Buffer auf einem System, wird aber manchmal auch nur als Begriff zusammen für den Color- und Z-Buffer verwendet. Im Accumulation Buffer werden Bilder mit verschiedenen Operationen akkumuliert, so kann zB durch Verwendung von mehreren Bildern eines sich bewegenden Objekts ein Motion-Blur-Effekt erzeugt werden.

Wenn nun alle Phasen durchlaufen sind, wird am Ende der Inhalt des Color Buffers am Bildschirm angezeigt. Um zu vermeiden, dass der Betrachter das Rendern der Objekte am Bildschirm sieht, wird Double Buffering angewandt. Eine Szene wird im Hintergrund in einen Back Buffer gerendert. Dieser wird, sobald die Szene gerendert wurde, mit dem Front Buffer, der am Bildschirm angezeigt wird, getauscht.

Visible-Surface-Algorithmen

Um zu bestimmen welches Objekt sichtbar ist, gibt es verschiedene Methoden, auch Hidden-Surface- oder Visible-Surface-Algorithmen genannt. Sie werden in „Image Precision“-, welche auf Pixel basieren, „Object Precision“-, welche mit den Objekten direkt arbeiten, und „List Priority“-Algorithmen, die eine Zwischenform der beiden anderen darstellen, unterteilt. Folgend werden einige „Image Precision“- sowie „List Priority“-Algorithmen beschrieben (vgl. [AGOS05], S. 264ff):

„Image Precision“-Algorithmen

Z-Buffer-Algorithmus

Beim Z-Buffer-Algorithmus wird, wenn ein Objekt zu einem Pixel gerendert wird, der Tiefenwert für das Objekt und den entsprechenden Pixel ermittelt und mit dem derzeit im Z-Buffer gespeicherten Wert verglichen. Ist nun der neue Wert kleiner als der Wert, der sich im Z-Buffer befindet, ist das neue Objekt näher an der Kamera und der Tiefenwert sowie der Farbwert werden durch den jeweils neuen ersetzt. Im anderen Fall werden die alten Werte beibehalten (vgl. [AKHAHO08], S. 23f).

Scanline-Algorithmus

Hier wird mit sogenannten Scanlines gearbeitet, die entlang der X-Achse verlaufen. Für jeden Wert der Y-Achse existiert eine Scanline. Entlang dieser Scanline werden Tiefenberechnungen angestellt. Dabei werden die Kanteninformationen der Polygone ausgewertet. Wenn eine Kante in einer Scanline erreicht wird, bedeutet dies, dass jetzt eine Oberfläche beginnt. Wird keine andere Kante eines anderen Polygons bis zur schließenden Kante dieser Oberfläche erreicht, ist diese Oberfläche sichtbar. Sind jedoch gleichzeitig zwei oder mehr Anfangskanten durchschritten worden ohne eine schließende Kante erreicht zu haben, muss eine Tiefenberechnung erfolgen (vgl. [HEBA04], S. 535ff).

“List Priority”-Algorithmen

Newell-Newell-Sancha Depth Sorting

Hier werden die Polygone nach ihrer Tiefe geordnet und danach mit Hilfe des Painter’s Algorithmus gezeichnet. Die Reihenfolge bestimmt dabei den Zeitpunkt der Darstellung. Die Polygone werden von hinten nach vorne „gezeichnet“, wodurch kein hinteres Polygon ein vorderes überdecken kann. Probleme kann es mit sich überlappenden Polygonen geben, wo die Reihenfolge nicht eindeutig bestimmt werden kann. Hier besteht eine Lösung darin, die Polygone zu teilen (vgl. [AGOS05], S. 269f, vgl. [DEBERG93] S. 5ff).

BSP-Algorithmus

Beim „Binary Space Partitioning“-Algorithmus werden die Polygone mit Hilfe eines Binary-Space-Baums geordnet. Dabei wird eine Ebene verwendet, die die Polygone so teilt, dass keines der Polygone der einen Seite eines der Polygone auf der anderen Seite überdecken kann. Dies wird rekursiv mit den beiden entstehenden Hälften fortgeführt, bis alle Polygone erfasst sind. Es ist ebenfalls erlaubt Polygone aufzutrennen. Zur Darstellung genügt es dann, diesen Baum entsprechend zu durchlaufen (vgl. [AGOS05], S. 270ff).

QuadTree-Algorithmus

Dabei wird der Anzeigebereich in immer kleinere Gebiete entlang der X- und Y-Achsen unterteilt. Der Trennpunkt ist jedes Mal die Mitte der Achse, wodurch vier gleich große Bereiche gebildet werden. Dies wird solange wiederholt, bis jeder Bereich entweder leer ist oder nur mehr ein einzelnes Objekt bzw. Dreieck enthält (vgl. [AKHAHO08], S. 654f).

3. Geschichtliche Entwicklung

Es folgt ein Überblick über die geschichtliche Entwicklung der Technologien und Formate im 3D-Bereich. Ebenso wird eine kurze Übersicht über die wirtschaftliche Entwicklung gegeben.

3.1. Technologien

Dieser Abschnitt beschreibt die historische Entwicklung von 3D-Grafikkarten (vgl. [SPILLE08]). Feng Liu teilt diese Entwicklung in seiner Dissertation in mehrere Generationen ein, welche in dieser Arbeit ergänzt werden. Dies zeigt vor allem die schnelle technologische Entwicklung im Bereich der Grafikkarten (vgl. [LUI05], S. 8ff).

3.1.1. Erste Generation

Zur ersten Generation zählen Grafikkarten, welche bis zum Jahr 1998 auf den Markt kamen. Sie konnten erstmals Berechnungen unabhängig von der CPU durchführen. Darunter fielen das Rastern von vortransformierten Dreiecken und das Anwenden von ein bis zwei Texturen. Die verfügbaren Operationen auf der GPU, dem Prozessor auf der Grafikkarte, waren sehr gering, so mussten Vertex-Transformationen noch immer von der CPU durchgeführt werden. Die Hauptaufgabe lag darin Texturfarben zu kombinieren, um die Farbe für die Pixel zu berechnen.

Die erste Grafikkarte dieser Art war im Jahr 1996 die Voodoo Graphics von 3dfx. Sie bestand aus zwei Chips, der Pixel- und der Texel-FX, und verfügte bereits über wichtige 3D-Verfahren, wie bilineares Texture Mapping mit Mipmap-Unterstützung, einen 16 Bit Z-Buffer, Perspektivenkorrektur, Kantenglättung und Texturkompression. Der Verfall der EDO-RAM Preise im Jahr 1996 ermöglichte den Siegeszug von 3dfx. Computerspiele, wie Tomb Raider, welches zB schon bilineares Filtering bot, waren die Vorreiter im 3D-Bereich.

3.1.2. Zweite Generation

Der Zeitraum der zweiten Generation erstreckte sich von 1999 bis 2000. Es war nun möglich T&L (Transform and Lighting) von der GPU durchführen zu lassen.

Das Verlagern der Vertex-Transformationen auf die Grafikkarte brachte eine deutliche Entlastung der CPU. Grafikkarten dieser Generation waren zB die NVidia Geforce 256 und die ATI Radeon 7500. Die Programmierschnittstellen DirectX 7 und OpenGL erlaubten das Nutzen dieser Funktionen auf der Grafikkarte.

3.1.3. Dritte Generation

Im Jahr 2001 wurde es möglich die GPU zu programmieren. Vertex-Berechnungen konnten nun durch das Angeben von einfachen Instruktionen auf die GPU verlagert werden. Zu dieser Phase zählten Grafikkarten, wie die NVidia Geforce 3 und die ATI 8500. DirectX 8 und OpenGL-Erweiterungen gaben dem Programmierer die Fähigkeit den Vertex Shader zu nutzen. Dazu wurde innerhalb von DirectX das Konzept eines Shader-Models eingeführt, um Hardware mit unterschiedlichen Shader Möglichkeiten zu unterscheiden. Zu dieser Zeit war das Shader-Model 1.1 aktuell. Der Vertex Shader musste dazu in einer Assembly ähnlichen Sprache programmiert werden (vgl. [AKHAHO08], S. 34f).

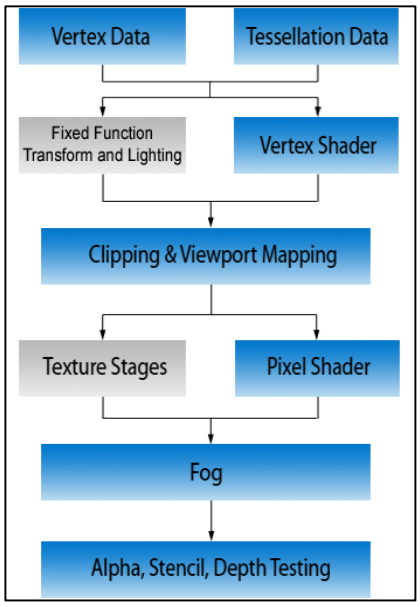


Abbildung 3: DirectX 8 Pipeline ([AKHA02], S. 214)

3.1.4. Vierte Generation

In den Jahren 2002 und 2003 waren auf der Grafikkarte nicht mehr nur Berechnungen auf Vertex-, sondern auch auf Pixel-Ebene durchführbar. Das Shader-Model 2.0 ermöglichte es, neben Programmen für den Vertex-, auch Programme für den Pixel Shader zu schreiben. Es wurde eine neue Shader-Programmiersprache namens HLSL für DirectX (High Level Shading Language) eingeführt, welche sich von der Assembly-Ebene abhob und C-ähnliche Programmierung zuließ. OpenGL verwendete für die Shader-Programmierung GLSL (vgl. [AKHAHO08], S. 35). Die NVidia Geforce FX und die ATI Radeon 9700 mit der Möglichkeit DirectX 9 auszuführen, sind Grafikkarten dieser Generation.

3.1.5. Fünfte Generation

Das Shader-Model 3.0 wurde im Jahr 2004 eingeführt und enthielt einige Verbesserungen. Andere Sprachen (zB Sh, zum Programmieren von Shadern) wurden vorgestellt. Sh erlaubte es Shader mit C++ Bibliotheken zu schreiben (vgl. [AKHAHO08], S. 35).

Mit der Einführung des Shader-Models 4.0 im Jahr 2007 wurde nach dem Vertex Shader ein weiterer, nämlich der Geometry Shader eingeführt. Der Geometry Shader erhält als Input ein Objekt mit den dazugehörigen Vertices und kann darauf Berechnungen durchführen. Der resultierende Output kann von keinem bis zu mehreren Objekten reichen. DirectX 10 und entsprechende Erweiterungen in OpenGL ermöglichten es diesen zu nutzen. Gleichzeitig gab es nun für alle drei verfügbaren Shader ein einheitliches Programmiermodel. Grafikkarten dieser Kategorie waren zB die NVidia Geforce 8800 GTX oder die Radeon HD 2900 XT von AMD, welche die Firma ATI gekauft hatte. Da die Shader nun frei programmierbar waren, konnten sie auch für nicht grafische Anwendungen, wie Kollisionserkennung, verwendet werden. Der Ansatz wird (General Purpose GPU - „GPGPU“) genannt (vgl. [AKHAHO08], S. 36ff, vgl. [AKHAHO08], S. 841).

3.1.6. Sechste Generation

Ende August 2008 wurde DirectX 11 und das damit verbundene Shader-Model 5.0 angekündigt (vgl. [GOLEM08]). Es wurden weitere Shader wie der Compute Shader eingeführt, hauptsächlich um allgemeine Berechnungen auf der GPU zu ermöglichen. Die Shader konnten nun auch objektorientiert programmiert werden. Die Tesselation, dh die Zerlegung von Flächen in Dreiecke, konnte somit auch auf der GPU durchgeführt werden, um feinere Oberflächen mit mehr Details zu generieren (vgl. [MSDN09], vgl. [MSDN10]). Die Pipeline hat sich gegenüber DirectX 10 etwas verändert. Es wurde zwischen Vertex- und Geometry Shader die Hull Shader Stufe, der Tesselator und die Domain Shader Stufe eingeführt (vgl. [MSDN10]).

Die erste DirectX 11 Grafikkarte war die Radeon HD 5870 von AMD (vgl. [SPILLE09]). NVidia bietet mit der GeForce GTX 480 ebenfalls eine Grafikkarte mit DirectX 11 Unterstützung (vgl. [NVIDIA10]).

3.2. Formate

In diesem Abschnitt wird auf die Entwicklung der Formate, die 3D im Web möglich machen, eingegangen. Es wird eine Übersicht der Entwicklung von den ersten Formaten bis zu den heute verfügbaren gegeben.

Im Jahr 1989 wurde bei Silicon Graphics ein neues Projekt namens Scenario gestartet, welches das Ziel hatte eine Infrastruktur für interaktive 3D-Grafikanwendungen zu designen und zu erstellen. Nach drei Jahren wurde das „Iris Inventor 3D Toolkit“ als erste Anwendung dieses Projektes veröffentlicht. Dieses Toolkit verwendete bereits viele Semantiken, die später auch in VRML zum Einsatz kamen. Ein Hauptaugenmerk wurde auf das Design des verwendeten Dateiformats gelegt, es sollte einfach zu verwenden sein. 1994 wurde die zweite Version von Inventor mit dem Namen „Open Inventor“ herausgebracht, welches auf mehreren Plattformen lief, da es auf OpenGL von Silicon Graphics aufbaute. Das Referenz Manual, das die verwendeten Objekte und das Dateiformat im Open Inventor Toolkit beschreibt, wurde letztendlich von Gavin Bell für den ersten Entwurf der VRML 1.0 Spezifikation verwendet.

Noch im Jahr 1994 wurde von Mark Pesce und Tony Parisi ein erster Prototyp eines 3D-Browsers entwickelt, der Labyrinth genannt wurde. Später im Jahr 1994 wurde die VRML Mailing List „www-vrml“ gegründet, die zur Einreichung von Vorschlägen für eine formale Spezifikation, um 3D im Web darstellen zu können, aufrief. Gavin Bell sah die Möglichkeit für Open Inventor und passte das Format an das World Wide Web an. Nach einer anregenden Debatte in der Mailing List wurde der Vorschlag von Gavin Bell angenommen und im Oktober 1994 wurde die erste Spezifikation von VRML 1.0 vorgestellt. In den nächsten Jahren wurde über eine Nachfolge von VRML 1.0 diskutiert, da viele Schlüsselfunktionen wie Animation und Interaktion fehlten.

Im August 1996 wurde die erste Version von VRML 2.0 veröffentlicht. Basis von VRML 2.0 war der Vorschlag „Moving Worlds“ von Silicon Graphics in Zusammenarbeit mit Mitra und Sony. Da VRML 2.0 1997 als Committee Draft 14722 veröffentlich wurde, ist diese Version auch als VRML97 bekannt (vgl. [CARBEL97]).

Das VRML Consortium wurde im Jahr 1996 gegründet und, um einen wirtschaftlich relevanten Nachfolger für VRML zu finden, wurde es 1999 in das WEB3D Consortium umgruppiert. Im Jahr 2000 wurde die X3D Working Group geformt (vgl. [FESTA02]) und im Juli 2002 wurde X3D veröffentlicht (vgl. [X3DDRAFT02]).

VRML war also das erste Format, mit dem 3D im Web möglich war. Ab dem Jahr 1996 kamen weitere Formate auf den Markt. Nachstehend wird eine Übersicht über die Formate von damals bis heute gegeben.

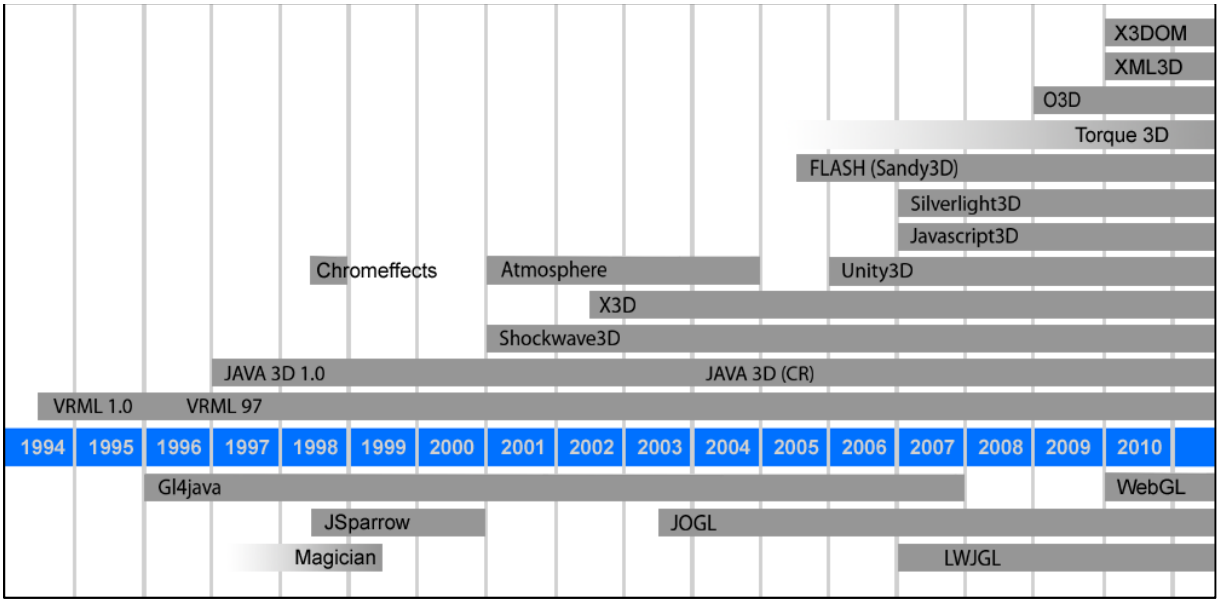


Abbildung 4: Geschichtliche Entwicklung von 3D-Formaten im Web

3.3. Wirtschaft

Wie bei der zeitlichen Abfolge der Formate gesehen werden kann, gibt es seit 1994 immer wieder kontinuierlich Versuche neue Formate, mit denen die dreidimensionale Darstellung von Inhalten im Web möglich ist, einzuführen. Daran allein kann die wirtschaftliche Entwicklung jedoch nicht festgemacht werden, da hier wichtige Kriterien, wie die Verbreitung und der Erfolg der Formate nicht ersichtlich sind. Diese Kriterien sind kaum zu eruieren. Viele Formate, die vor einigen Jahren veröffentlicht wurden, existieren zwar noch, aber ob sie auch noch verwendet werden, kann nicht gesagt werden. Teilweise sind zB die Webseiten der Formate seit Jahren nicht mehr aktualisiert worden. Es können daher oft nur subjektive Eindrücke verwendet werden, um eine Einschätzung über die wirtschaftliche Entwicklung vorzunehmen.

Da dreidimensionale Darstellung sehr viel Performance benötigt, müssen einerseits entsprechend leistungsstarke Rechner beim Betrachter vorhanden sein und andererseits muss eine schnelle Internetverbindung verfügbar sein, um die zum Teil großen Datenmengen zu übertragen. Alan Hudson vom Web3D Consortium meinte in einem Interview bei der Siggraph 2008, dass VRML nach einem Hype in den 90ern daran scheiterte, dass die Hardware und die Netzwerkgeschwindigkeit für Web 3D noch zu langsam war. Außerdem war die Kompatibilität der einzelnen Browser in Bezug auf VRML unzureichend. Es konnte sein, dass Inhalte in einem Browser anders aussahen als in einem anderen oder überhaupt nicht funktionierten (vgl. [W3DHUD]).

Im Bereich des dreidimensionalen Internets zB, welches nicht von einem Browser abhängig ist, gab es im Jahr 2008 für virtuelle Welten wie Second Life, von Analysten des Marktforschungsinstituts Forrester eine Vorhersage. Sie meinten, dass das 3D-Internet, von damals aus gesehen in fünf Jahren, also im Jahr 2013, eine Bedeutung erlangen kann wie das Web heute. Dreidimensionale Internetauftritte und Avatare werden vor allem in der Geschäftswelt an Gewicht gewinnen und einige große Firmen wie IBM und Intel investieren bereits in solche Technologien (vgl. [3DZU08]).

4. Vergleich aktueller Formate

In diesem Kapitel werden wichtige, derzeit aktuell verfügbare Formate anhand der nachstehenden Kriterien:

- Preis
- Verbreitung
- Plugin-Größe
- Browserkompatibilität
- Hardwareunterstützung
- Einsatzgebiete
- Funktionsumfang

Es gibt standardisierte und proprietäre Formate. Wichtig dabei ist, dass für standardisierte Formate, auf Grund unterschiedlicher Plugins mehrere Möglichkeiten vorhanden sind diese darzustellen, wobei es bei proprietären Formaten meist nur ein vom Hersteller bereitgestelltes Plugin gibt, auf das diese Formate aufsetzen. Es werden somit die einzelnen Formate sowie auch die für das einzelne Format verfügbaren Plugins gegenübergestellt. Die Plugin-Technologie war entscheidend für den Beginn von 3D im Web. Der Browser „Netscape Navigator“ war der erste, der diese Technologie verwendet hat. Dadurch können Inhalte direkt im Browser angezeigt werden, ohne weitere Anwendungen starten zu müssen (vgl. [STAPPLER97]).

Anzumerken ist, dass beim Vergleich nur Bezug auf Systeme mit dem Betriebssystem Windows von Microsoft genommen wird, was bedeutet, dass zB die Plugin-Größe bei anderen Betriebssystemen variieren kann. Die Daten für den Vergleich wurden jeweils der entsprechenden Webseite des Anbieters entnommen. Andere Quellen werden gesondert angegeben.

4.1. Standards

VRML und X3D sind Standards und somit gibt es hier nicht nur ein Plugin, sondern mehrere. Nachstehend wird eine Auswahl von derzeit verfügbaren Plugins aufgelistet.

4.1.1. VRML

VRML setzt auf einen Szenengraphen auf, welcher in einem eigenen Dateiformat mit der Endung „*.wrl“ definiert wird. Als Einsatzgebiete werden in der Spezifikation zB wissenschaftliche Visualisierungen, Multimedia Präsentationen, Unterhaltung, Bildung, Internet Seiten und virtuelle Welten angegeben. Dabei gehören Primitive, Kamera, Licht, Text, Material, Shading, Interaktion und Animation zum Funktionsumfang von VRML (vgl. [VRMLSPEC03]).

Plugins

Cortona3D Viewer 6.0	
Webseite:	http://www.cortona3d.com/Products/Cortona-3D-Viewer.aspx
Downloadgröße (Windows):	Ca. 4,5 MB
Preis:	Kostenlos
Verbreitung:	Keine Information verfügbar
Browserkompatibilität:	Microsoft Internet Explorer 6.0+, Netscape Navigator 8.0+, Mozilla Firefox 1.5+, Google Chrome 1.0, Opera 8.5+
Hardwareunterstützung:	Ja, für OpenGL und DirectX
Funktionalität (Version des Standards):	VRML97

Tabelle 1: VRML - Cortona3D Viewer 6.0

Cosmo Player	
Webseite:	http://cic.nist.gov/vrml/cosmoplayer.html
Downloadgröße (Windows):	Ca. 2,4 MB
Preis:	Kostenlos
Verbreitung:	Keine Information verfügbar
Browserkompatibilität:	Microsoft Internet Explorer, Netscape Navigator, Mozilla Firefox 2.0, Google Chrome
Hardwareunterstützung:	Ja, für OpenGL und DirectX
Funktionsumfang (Version des Standards):	VRML97 (laut [COSMO98] erfüllt der Cosmo Player in der Version 2.1.1 die VRML 97 Spezifikation)

Tabelle 2: VRML - Cosmo Player

Octaga Player 3.0	
Webseite:	http://www.octaga.com/index.php?option=com_content&task=view&id=17&Itemid=40
Downloadgröße (Windows):	Ca. 5,6 MB
Preis:	Kostenlos (für nicht kommerziellen Einsatz)
Verbreitung:	Keine Information verfügbar
Browserkompatibilität:	Microsoft Internet Explorer, Mozilla Firefox, andere Mozilla basierte Browser (vgl. [OCTAGA10])
Hardwareunterstützung:	Ja, zumindest OpenGL (vgl. [OCTAGA10])
Funktionsumfang (Version des Standards):	VRML 97

Tabelle 3: VRML - Octaga Player 3.0

4.1.2. X3D

X3D ist der Nachfolger von VRML und setzt ebenso auf einen Szenengraphen auf. Neben neuen Funktionen wird die Datei mit der Endung „*.x3d“ nun durch eine XML-Struktur definiert. Als Einsatzgebiete werden dieselben wie bei VRML angegeben. Primitive, Kamera, Licht, Text, Material, Shading, Interaktion, Partikelsystem, Animation und Physik sind im Funktionsumfang von X3D inkludiert (vgl. [X3DSPEC08]).

Plugins

BS Contact 7.2 (Testversion)	
Webseite:	http://www.bitmanagement.de/en/products/interactive-3d-clients/bs-contact
Downloadgröße (Windows):	Ca. 5,2 MB
Preis:	Kostenlos (mit Wasserzeichen)
Verbreitung:	Keine Information verfügbar
Browserkompatibilität:	Microsoft Internet Explorer 6.0+, Mozilla Firefox 2.0+, Google Chrome, Opera (vgl. [BSCONTACT09])
Hardwareunterstützung:	Ja, OpenGL und DirectX (vgl. [BSCONTACT09])

Tabelle 4: X3D - BS Contact 7.2

Vivaty Player	
Webseite:	http://www.vivaty.com/index.php (Anmerkung: Diese Seite war beim Abschluss der Arbeit nicht verfügbar.)
Downloadgröße (Windows):	Ca. 4 MB
Preis:	Kostenlos
Verbreitung:	Keine Information verfügbar
Browserkompatibilität:	Internet Explorer 6+, Mozilla Firefox 3.0+, Safari
Hardwareunterstützung:	Ja, DirectX (vgl. [VIVATY09])

Tabelle 5: X3D - Vivaty Player

Octaga Player 3.0
Der Octaga Player 3.0 unterstützt auch X3D. Für Details siehe Octaga Player 3.0

4.2. Proprietäre Formate

4.2.1. Shockwave 3D

Um Shockwave-3D-Inhalte zu erzeugen, wird das Programm „Director“ von Adobe benötigt. Director selbst ist kostenpflichtig (vgl. [SWPR10]). Seit der Version 8.5 wird Echzeit-3D Grafik unterstützt. Die erzeugten Inhalte können durch ein Plugin auch in einem Webbrowser angezeigt werden. Die zugehörige Programmiersprache nennt sich Lingo (vgl. [GRGR02], S. 1f).

Shockwave3D	
Start:	April 2001 (vgl. [SHOCK3D07])
Ende:	-
Plugin:	Adobe Shockwave Player 11.5
Webseite:	http://get.adobe.com/de/shockwave
Downloadgröße (Windows):	Ca. 4 MB
Preis:	Kostenlos
Verbreitung:	Der Shockwave Player in der Version 11 hatte im März 2010 in den Mature Markets (USA, Kanada, Großbritannien, Deutschland, Frankreich, Japan, Australien und Neuseeland) eine Verbreitung von 43,1 % (vgl. [SHWPENE10]).
Browserkompatibilität:	Microsoft Internet Explorer 6.0+, Mozilla Firefox 2.0+ (vgl. [SWBROWSER10])
Hardwareunterstützung:	Ja, OpenGL und DirectX (vgl. [GRGR02], S. 58)
Einsatzgebiete:	Entertainment (Spiele), Demos und Prototypen, Simulationen, E-Learning Kurse (vgl. [SWEG10])
Funktionsumfang (vgl. [GRGR02])	Primitive Kamera Licht 3D-Text Material Interaktion Shading Partikelsystem Animation Physik (vgl. [SWHW10])

Tabelle 6: Shockwave 3D

4.2.2. Flash 3D

Die Autorenumgebung „Flash“ ist ein kostenpflichtiges Programm von Adobe und wird verwendet, um interaktive Inhalte vor allem für das Web zu generieren (vgl. [FLFAQ]). Flashprogramme können jedoch mit Hilfe des frei verfügbaren Flex SDKs auch ohne die Autorenumgebung von Adobe erstellt werden (vgl. [FLEX10]).

Die Programmiersprache, zur Erzeugung von Flashinhalten, nennt sich ActionScript und ist derzeit in der Version 3 verfügbar (vgl. [ASTC10]). Flash bietet standardmäßig nur sehr geringe Unterstützung für 3D-Echtzeitgrafik (vgl. [FLASHP10]) und ist hier auf Bibliotheken von Drittanbietern angewiesen, wovon es jedoch bereits einige, wie zB Sandy 3D oder Papervision3D, gibt. Papervision3D zB ist Szenengraphen basiert (vgl. [PV3D10]).

Flash 3D	
Start:	Oktober 2005 (Sandy 3D) (vgl. [SANDY3D09])
Ende:	-
Plugin:	Adobe Flash Player 10.0
Webseite:	http://get.adobe.com/de/flashplayer
Downloadgröße (Windows):	Ca. 1,8 MB
Preis:	Kostenlos
Verbreitung:	Die Version Flashplayer 10.0 hatte im März 2010 eine Verbreitung von 97,2 % in den Mature Markets, wobei die Tendenz über die letzten Monate steigend ist. Der Flashplayer 9 hatte in den Mature Markets im März 2010 bereits eine Verbreitung von 99,1 %. Die Mature Markets umfassen USA, Kanada, Großbritannien, Deutschland, Frankreich, Japan, Australien und Neuseeland (vgl. [FPPENE10]).
Browserkompatibilität:	Für Microsoft Windows XP werden folgende Browser unterstützt: Microsoft Internet Explorer 6.0+, Firefox 2.x+, AOL 9, Opera 9.5+, Chrome 2.0+ (vgl. [FLBROWSER10]).
Hardwareunterstützung:	Ab dem Flashplayer in der Version 10 ist Hardwareunterstützung eingeschränkt möglich (vgl. [URO08]).
Einsatzgebiete:	ZB Entertainment (Spiele), Werbung, Bereicherung für Webseiten (ersichtlich im Blog von Papervision3D)

Funktionsumfang: **zB. Papervision3D**	Primitive Laden von externen Modellen Kamera Licht 3D-Text Material Shading Interaktion Partikelsystem Animation Physik

Tabelle 7: Flash 3D

4.2.3. Unity

Unity ist vor allem für die einfache Erzeugung von 3D-Spielen gedacht und wird von Unity Technologies angeboten. Um Inhalte zu erzeugen, wird ein Editor benötigt. Die erzeugten Inhalte können dann über ein Plugin im Browser betrachtet werden. Es gibt eine freie Variante des Editors, die jedoch nicht alle Features der professionellen Variante bietet. Die Programmiersprachen, die verwendet werden können, sind JavaScript, C# und ein Python-Dialekt namens Boo (vgl. [UNITY3D10]).

Unity	
Start:	2005 - 2006 (vgl. [UNITY3D09]). Es kann nicht genau festgestellt werden, ab wann ein Plugin für Windows-basierte Browser verfügbar war.
Ende:	-
Plugin:	Unity 2.6.1
Webseite:	http://unity3d.com/webplayer
Downloadgröße (Windows):	Ca. 3,1 MB
Preis:	Kostenlos
Verbreitung:	Laut einem Blogeintrag von David Helgason hatte das Unity3D Plugin im März 2008 eine Verbreitung von ca. 1 % (vgl. [HELG08]).

Browserkompatibilität:	Microsoft Internet Explorer, Mozilla Firefox, Safari, Google Chrome, Opera, Mozilla, Netscape (vgl. [UNITYBROWSER10])
Hardwareunterstützung:	Ja, OpenGL und DirectX (vgl. [UNITYHW10])
Einsatzgebiete:	Entertainment (Spiele)
Funktionsumfang: **(vgl. [UNITYMAN10])**	Primitive Laden von externen Modellen Kamera Licht 3D-Text Material Shading Interaktion Partikelsystem Animation Physik

Tabelle 8: Unity

4.2.4. O3D

Google bietet mit O3D eine Open Source Web API, um interaktive 3D-Webapplikationen zu erzeugen. Es ist ein Vorschlag von Google für einen offenen Webstandard (vgl. [O3DFAQ09]). Anwendungen in O3D werden mit JavaScript geschrieben (vgl. [O3DPRG10]). O3D setzt dabei auf einen Szenengraphen auf.

O3D	
Start:	April 2009 (vgl. [O3DBLOG09])
Ende:	-
Plugin:	O3D 0.1.42
Webseite:	http://code.google.com/intl/de-DE/apis/o3d
Downloadgröße (Windows):	ca. 1,4 MB bis 1,7 MB (vgl. [O3DFAQ09])
Preis:	Kostenlos
Verbreitung:	Es wurden keine genauen Angaben gefunden, aber da das Plugin erst seit April 2009 existiert, wird von einer sehr geringen Verbreitung ausgegangen.

Browserkompatibilität:	Microsoft Internet Explorer 7.0+ (nur x86), Mozilla Firefox 2+, Google Chrome (vgl. [O3DINSTR10]).
Hardwareunterstützung:	Ja, O3D unterstützt DirectX unter Microsoft Windows und OpenGL für MAC OS und Linux (vgl. [O3DHW09]).
Einsatzgebiete:	Entertainment (Spiele), Werbung, Produktdemonstrationen, Virtuelle Welten (vgl. [O3DPRG10])
Funktionsumfang (vgl. [O3DDEVG10]):	Primitive Laden von externen Modellen Kamera Licht Material Shading Interaktion Partikelsystem Animation

Tabelle 9: O3D

4.2.5. Torque 3D

Torque 3D ist eine Plattform, die zur einfachen Erzeugung von Multiplattformspielen gedacht ist. Dazu wird ein kostenpflichtiger Editor benötigt. Die Programmiersprache, die bei Torque verwendet wird, nennt sich TorqueScript und ist an C++ angelehnt (vgl. [TORQUE10]).

Torque 3D	
Start:	2001 (Torque). Ob es zu diesem Zeitpunkt auch schon möglich war 3D-Inhalte im Browser darzustellen, war nicht feststellbar.
Ende:	-
Webseite:	http://www.torquepowered.com/products/torque-3d
Plugin:	Für jedes Projekt muss ein eigenes Plugin erstellt werden (vgl. [TORQUEDOC10]).
Downloadgröße (Windows):	-
Preis:	-

Verbreitung:	Da jedes Mal ein eigenes Plugin erstellt werden muss, kann hier nichts über die Verbreitung gesagt werden.
Browserkompatibilität:	Zumindest: Microsoft Internet Explorer 7.0, Mozilla Firefox 3.0+, Google Chrome
Hardwareunterstützung:	Ja, die Minimalvoraussetzung für Windows ist DirectX 9.0c+.
Einsatzgebiete:	Entertainment (Spiele)
Funktionsumfang (vgl. [TORQUEDOCB10]):	Laden von externen Modellen Kamera Licht Material Shading Interaktion Partikelsystem Animation Physik

Tabelle 10: Torque 3D

4.2.6. Silverlight 3D

Silverlight ist von Microsoft und dient unter anderem dazu, interaktive Webapplikationen zu entwickeln. Um in Silverlight Inhalte zu erstellen, kann zB das Microsoft® Silverlight™ 3 SDK verwendet werden, welches kostenlos zum Download angeboten wird (vgl. [SILVERSDK09]). Silverlight bietet selbst keine 3D-Unterstützung und ist auf externe Bibliotheken wie Balder (http://balder.codeplex.com) angewiesen (vgl. [SILVER3D07]).

Silverlight 3D	
Start:	Mai 2007 (Balder) (vgl. [BALDER07])
Ende:	-
Plugin:	Microsoft Silverlight 3.0
Webseite:	http://silverlight.net
Downloadgröße (Windows):	Ca. 4,8 MB
Preis:	Kostenlos (vgl. [SILVERBROWSER10])

Verbreitung:	Laut den Rich Internet Application Statistics haben Silverlight 2 und Silverlight 3 zusammen derzeit eine Verbreitung von ca. 55 % (vgl. [RIASTAT10]).
Browserkompatibilität:	Microsoft Internet Explorer 6.0+, Mozilla Firefox 2.0+, Google Chrome (vgl. [SILVERBROWSER10])
Hardwareunterstützung:	Ja, aber nur sehr eingeschränkt. Derzeit können nur bereits gerenderte Bilder im Grafikspeicher behalten und wiederverwendet werden (vgl. [SILVERHW09]).
Einsatzgebiete:	Für den Bereich 3D ist keine Information verfügbar.
Funktionsumfang (vgl. [BALDER10]):	Primitive Laden von externen Modellen Kamera Licht Material Shading Interaktion Animation

Tabelle 11: Silverlight 3D

4.2.7. Java

Bei Java gibt es mehrere Ansätze 3D-Inhalte in Browsern darzustellen. Dazu ist das Java Plugin von Sun Microsystems nötig.

Java	
Webseite:	http://java.sun.com/javase/downloads/index.jsp
Plugin:	Java SE Runtime Environment 6u20
Downloadgröße (Windows):	Ca. 15.9 MB
Preis:	Kostenlos
Verbreitung:	Laut einer von Adobe in Auftrag gegebenen Studie lag die Verbreitung des Java Plugins in den Mature Markets (USA, Kanada, Großbritannien, Deutschland, Frankreich, Japan, Australien, Neuseeland) im März 2010 bei 77 % (vgl. [JAVAPEN10]).

Browserkompatibilität:	Für Microsoft Windows XP werden die folgenden Browser unterstützt: Microsoft Internet Explorer 6 SP1+, Mozilla Firefox 2+ (vgl. [JAVABROWSER10]).
Hardwareunterstützung:	Ja (vgl. [JAVAHW10])

Tabelle 12: Java

Die Möglichkeiten, die Java im Bereich 3D bietet, werden im Anschluss näher besprochen. Es existieren hierfür APIs, die auf Szenengraphen aufsetzen und jene, die OpenGL Bindings darstellen.

Szenengraphen

Java 3D

Derzeit ist Java 3D in der Version 1.5.2 verfügbar. Ab der Version 1.3.2 im Jahr 2004 wurde Java 3D nicht mehr von Sun weiterentwickelt, sondern als „Community Source“ Projekt freigegeben, wodurch sich jeder an der Entwicklung beteiligen konnte. Im Jahr 2008 wurde es mit dem Namen „3D Graphics API for the Java Platform“ unter der GPL veröffentlicht (vgl. [JAVA3D08], vgl. [JAVA3D09]).

Java 3D	
Start:	1997 (vgl. [WABO01], S. 529)
Ende:	-
Webseite:	https://java3d.dev.java.net
Einsatzgebiete:	Entertainment (Spiele), Visualisierungen, Simulationen (vgl. [JAVA3DEG10])
Funktionsumfang (vgl. [JAVA3D00]):	Primitive Laden von externen Modellen Kamera Licht 3D-Text Material Shading Interaktion Animation

Tabelle 13: Java 3D

Weitere Szenengraph-Engines für Java sind Xith3D (http://xith.org), jME (http://www.jmonkeyengine.com) und Ardor3D (http://www.ardor3d.com).

OpenGL Bindings

Da die folgenden Bibliotheken OpenGL-Bindings sind und sie somit die gleichen Befehle, wie OpenGL verwenden, haben sie auch denselben Funktionsumfang.

LWJGL

LWJGL steht für „Lightweight Java Game Library“ und wird angeboten, um hochwertige Spiele mit Java schreiben zu können. Es ist unter der BSD Lizenz erhältlich und somit frei verwendbar (vgl. [LWJGL10]).

LWJGL	
Start:	Februar 2007 (vgl. [LWJGL07])
Ende:	-
Webseite:	http://lwjgl.org

Tabelle 14: LWJGL

JOGL

JOGL ist ein Java Binding für die OpenGL API. Es bietet Unterstützung für die OpenGL-Versionen 1.0 - 3.0, größer gleich 3.1 sowie ES 1.X und ES 2.X inklusive fast aller Erweiterungen. Es ist für AWT und SWING verfügbar (vgl. [JOGLWIKI09]) und unter der BSD Lizenz veröffentlicht und somit frei verwendbar (vgl. [JOGLLIZ09]).

JOGL	
Start:	August 2003 (vgl. [JOGL09])
Ende:	-
Webseite:	http://kenai.com/projects/jogl/pages/Home

Tabelle 15: JOGL

Weitere OpenGL-Bindings, die jedoch nicht mehr weiterentwickelt werden, sind nachstehend aufgelistet.

Gl4Java	
Start:	Jänner 1996 (vgl. [GL4JAVA97])
Ende:	September 2007 (vgl. [GL4JAVA07])

Tabelle 16: Gl4Java

JSparrow	
Start:	September 1998 (vgl. [JSPARROW00])
Ende:	September 2000 (vgl. [JSPARROW00])

Tabelle 17: JSparrow

Magician	
Start:	?
Ende:	März 1999 (vgl. [DAY99])

Tabelle 18: Magician

4.2.8. JavaScript 3D

Da JavaScript in sogut wie jedem Browser vorhanden sein dürfte, gibt es auch hier Ansätze, 3D im Web durch diese Skriptsprache zu realisieren. Der große Vorteil dabei ist, dass kein zusätzliches Plugin installiert werden muss, um 3D-Inhalte betrachten zu können. Es existieren Bibliotheken, die entsprechende 3D-Funktionalitäten zur Verfügung stellen. Eine sehr einfache ist zB JS3D (vgl. [JS3D07]).

Im Jahr 2006 hat Mozilla begonnen, ein Add-On namens „Canvas 3D“ für den Webbrowser Mozilla Firefox zu entwickeln, welches OpenGL 3D-Inhalte anzeigen kann (vgl. [VUKI07]). Es setzt auf das in HTML5 spezifizierte Element Canvas auf (vgl. [VUKI09]). Das Canvas-Element ist eine Zeichenfläche, die zum Rendern von Graphen, Spielegrafiken oder auch Bildern verwendet werden kann (vgl. [HIHY09]). Es existieren JavaScript Bibliotheken, wie C3DL, die Canvas 3D verwenden und die das Schreiben von Anwendungen durch Zur-Verfügung-Stellen von zB mathematischen Funktionen erleichtern (vgl. [C3DL09]).

4.2.9. WebGL

Im März 2009 wurde von der Khronos Group aufgrund eines Vorschlags von Mozilla eine Initiative gegründet, um einen offenen, gebührenfreien Standard für 3D-Beschleunigung im Web zu kreieren (vgl. [KHRONPR09]). Mitglieder der daraus entstandenen Arbeitsgruppe sind Browseranbieter wie Apple, Google, Mozilla und Opera. Der Name des Standards ist „WebGL“ und er lehnt sich sehr an die OpenGL ES 2.0 Spezifikation an. Geschrieben werden die Programme in JavaScript und sie setzen ebenso auf das in HTML5 vorhandene Element Canvas auf. Über die OpenGL-Shader-Sprache GLSL können auch eigene Shader geschrieben werden. Der große Vorteil ist, dass dabei Hardwarebeschleunigung geboten wird (vgl. [KHRON09]). Der Standard befindet sich derzeit noch in Entwicklung und mit einer Veröffentlichung wird in der ersten Hälfte dieses Jahres gerechnet (vgl. [KHRONPRA09]). Gleichzeitig gibt es auch Bemühungen Bibliotheken zu entwickeln, welche die Verwendung von WebGL vereinfachen sollen, zB GLGE (vgl. [GLGE10]) und SceneJS (vgl. [SCENEJS10]). Beide Entwicklungen stehen dabei noch am Anfang. Der Nachfolger von C3DL „C3DL 2.0“ setzt nun ebenso auf WebGL.

Ein Nachteil ist, dass Microsoft nicht zu dieser Arbeitsgruppe gehört und somit der Internet Explorer wahrscheinlich keine Unterstützung für WebGL bieten wird. Es wird jedoch erwartet, dass Drittanbieter WebGL für den Internet Explorer möglich machen (vgl. [WEBGLINT09]). Aber es vergeht sicher noch einige Zeit bis die Verbreitung der Browser, die WebGL unterstützen, ausreichend groß ist.

4.2.10. X3DOM

Am Fraunhofer-Institut wird derzeit an einem experimentellen Open Source Framework namens X3DOM gearbeitet, welches X3D in den HTML DOM integriert, ohne dass dafür ein spezielles Plugin benötigt werden soll. Ziel ist, dass X3DOM Teil des HTML5 Standards wird. Die derzeit auf der Webseite verfügbaren Demos verwenden zur Darstellung WebGL (vgl. [X3DOM09]).

4.2.11. XML3D

Am Computer Graphics Lab der Universität Saarland wird XML3D entwickelt, welches sich wie X3DOM in den HTML DOM integrieren lässt und somit ohne Browserplugin betrachtet werden kann. XML3D befindet sich noch in Entwicklung und es wird derzeit eine modifizierte Version eines Browsers benötigt. Das Besondere ist, dass zum Rendern der Szene neben Rasterisierung auch Raytracing verwendet werden kann (vgl. [XML3D10], vgl. [GOLXML3D10]).

4.2.12. Weitere Formate

Es existieren noch weitere Möglichkeiten 3D im Web darzustellen, einige davon werden nachstehend nur kurz erwähnt, da es dazu nur wenige Informationen gibt, sie nicht mehr verfügbar sind, kaum noch verwendet werden oder im Funktionsumfang nicht an die oben genannten heranreichen.

- Adobe Atmosphere (Start: 2001 (vgl. [ADOBE05]), Ende: Dezember 2004 (vgl. [ADOBE04]))
- Microsoft Chromeffects (Start: August 1998 (vgl. [MSCHRA98]), Ende: November 1998 (vgl. [MSCHRB98]))
- Virtools (3DVIA) (Start: Juni 2007 (vgl. [3DVIA07]), Ende: -)
- MPEG 4 (Start: 2000 (vgl. [WABO01], S. 58), Ende: -)
- Shout3D (vgl. [SHOUT3D])
- 3DAnywhere (vgl. [3DANYW01])
- Anfy 3D (vgl. [Anfy3D02])

4.3. Ergebnis

Der Vergleich der Formate zeigt, dass es derzeit keines gibt, welches uneingeschränkt für den Einsatz von Informationsvisualisierungen zu empfehlen ist. Ein großer Nachteil ist, dass bei den meisten Formaten ein Plugin installiert werden muss, um die Inhalte betrachten zu können. Dies stellt aus eigener Erfahrung meist ein großes Hindernis dar, vor allem wenn das Plugin nur zum Betrachten eines bestimmten Inhalts für kurze Zeit benötigt wird. Sollte ein anderer Inhalt in einem anderen Format erstellt worden sein, müsste dafür wiederum ein anderes Plugin installiert werden.

Preis

Es existiert für alle Formate ein kostenloses Plugin, mit dem die Inhalte betrachtet werden können. Die Programme zum Erstellen der Inhalte sind bei einigen, vor allem proprietären Anbietern wie Shockwave3D, Unity und Torque 3D jedoch kostenpflichtig.

Verbreitung

Da im Bereich der Informationsvisualisierung eine breite Nutzergruppe angesprochen werden soll, ist eine hohe Verbreitung des Plugins Voraussetzung. Die Ermittlung der Verbreitung ist bei den meisten Formaten jedoch oft nur schwer möglich, da entsprechende Zahlen nicht existieren. Adobe zB lässt die Verbreitung ihrer eigenen Plugins und teilweise auch die Verbreitung anderer Plugins wie zB die des Java-Plugins in regelmäßigen Abständen ermitteln. Laut diesen Zahlen bietet Flash mit dem Flashplayer in der Version 9 eine Unterstützung von ca. 99 %, was wahrscheinlich kein anderes der verglichenen Formate (außer JavaScript) erreichen kann. Java hat eine Verbreitung von ca. 77 % und Shockwave 3D von ca. 43 %. Da JavaScript, ohne ein Plugin installieren zu müssen, für jeden Browser verfügbar ist, kann eine Verbreitung von an die 100 % angenommen werden.

Funktionsumfang

Der Funktionsumfang ist bei allen Formaten ungefähr derselbe. Die Unterschiede sind hier in Detailbereichen zu finden. Für die Umsetzung einer Informationsvisualisierung sollte der Umfang bei allen ausreichend sein. Nur bei JavaScript ist der Funktionsumfang, der derzeit ohne zusätzliche Plugins verfügbar ist, nicht für eine Informationsvisualisierung geeignet. Viele Formate bieten hier Szenengraphen, die eine Erstellung von 3D-Inhalten vereinfachen. Die Java-OpenGL-Bindings und WebGL setzen auf OpenGL.

Hardwareunterstützung

Shockwave 3D, Unity, Torque 3D, Java und JavaScript durch WebGL oder O3D bieten Hardwareunterstützung. Flash 3D und Silverlight 3D unterstützen die Hardware nur eingeschränkt.

Entscheidung

Da aufgrund der großen Verbreitung des Flashplayers 9 von ca. 99 % so gut wie jeder die Möglichkeit hat Flashinhalte zu betrachten und ein Download des Plugins nur in Ausnahmefällen nötig ist, wird für die Erstellung einer Informationsvisualisierung Flash ausgewählt. Bei Flash gibt es außerdem mehrere 3D-Engines, die zur Auswahl stehen und je nach Anforderung kann davon eine ausgewählt werden. Der Nachteil, dass Flash derzeit kaum Hardwareunterstützung bietet, wird zu Gunsten der großen Verbreitung in Kauf genommen. Die Überwindung sich ein Plugin zu installieren ist wahrscheinlich höher, als eventuell auftretende Geschwindigkeitsnachteile hinzunehmen.

WebGL wäre auf Grund der Tatsache, dass kein Plugin installiert werden muss und trotzdem Hardwarebeschleunigung geboten wird, eine Alternative zu Flash. WebGL und die entsprechenden Bibliotheken, die auf WebGL aufbauen, sind derzeit aber noch in der Entwicklung und es wird sicher noch dauern, bis die Browserversionen, die dies unterstützen, eine hohe Verbreitung haben.

5. Vergleich von Flash 3D-Engines

Nach einer allgemeinen Beschreibung zu Flash und einer Vorstellung wichtiger verfügbarer 3D-Engines folgt ein Vergleich dieser aufgrund ausgewählter Kriterien.

5.1. Allgemeines zu Flash

Flash ist eine Autorenumgebung von Adobe, um interaktive, plattformübergreifende Anwendungen zu schreiben (vgl. [FLALLG10]). Wie aus der Hilfe von Flash entnommen werden kann, sind die wichtigsten Bereiche der Autorenumgebung die Bühne, auf der alle Elemente platziert werden, die Zeitleiste, mit der die Elemente zeitlich gesteuert werden können, sowie die Bibliothek, die alle Elemente welche im Zeitablauf benötigt werden, enthält. Die Erstellung von Animationen kann in Flash ohne jegliche Programmierung erfolgen. Für komplexere Anwendungen steht jedoch eine objektorientierte Programmiersprache zur Verfügung. Diese heißt in der aktuellen Version ActionScript 3.0.

Über ActionScript 3.0 besteht Zugriff auf alle notwendigen Elemente. Es können auch externe Ressourcen wie Bilder oder Audios zur Laufzeit geladen werden. Genauso ist eine Kombination zwischen der Verwendung der Autorenumgebung und dem Einsatz von ActionScript möglich. Für den Prototyp dieser Arbeit wird nur ActionScript 3.0 verwendet. Allein die Veröffentlichung der Anwendung erfolgt über die Autorenumgebung Adobe Flash CS3 Professional. Es existiert aber zB auch die Möglichkeit, Editoren von Drittanbietern zu nutzen und den ActionScript-Code zB mit Adobe® Flex® 4 SDK zu kompilieren. Um die Anwendung im Browser abspielen zu können, muss diese zu einer „SWF“-Datei veröffentlicht und in einer HTML-Datei eingebettet werden. Abgespielt wird die SWF-Datei durch den Flashplayer, welcher als Plugin im Webbrowser vorhanden sein muss. ActionScript 3.0 basiert auf dem Entwurf des ECMAScript (ECMA-262) in der Version 4. Es inkludiert ebenso eine XML-API, die die E4X-Spezifikation (ECMAScript for XML, ECMA-357 Version 2) erfüllt.

Alle in Flash erstellten Anwendungen bestehen aus einer Hierarchie von sichtbaren Objekten. Diese Hierarchie wird Anzeigeliste genannt. Die Bühne bildet dabei die Spitze dieser Anzeigeliste und dient als Container für alle andern sichtbaren Objekte. Solche Objekte können zB Sprites oder MovieClips, aber auch Textfelder und Vektorformen sein. Als Sprites werden Container verstanden, die neben einfachen sichtbaren Objekten auch andere Container enthalten können. Die Bühne selbst ist ein Container vom Typ Sprite. MovieClips sind ebenso Container, haben aber im Gegensatz zu Sprites auch noch eine eigene Zeitleiste, die durch eine Abfolge von Bildern definiert ist (vgl. [FLHELP]).

5.2. 3D-Features

5.2.1. Neue 3D-Funktionen

Ab dem Flashplayer in der Version 10 ist für jedes sichtbare Objekt neben den Eigenschaften „X“ und „Y“ auch die Eigenschaft „Z“ verfügbar, welche als Tiefeninformation dient. Dazu gibt es Methoden, um Drehungen oder Größenänderungen im 3D-Raum durchführen zu können. Gleichzeitig wurden entsprechende Klassen wie Vector3D, um 3D-Punkte managen zu können, oder Matrix3D, um komplexere 3D-Transformationen und -Translationen vornehmen zu können, hinzugefügt (vgl. [FLASHP10]).

5.2.2. Hardwarebeschleunigung

Ebenfalls mit dem Flashplayer in der Version 10 wird erstmals GPU Unterstützung für Flash angeboten. Die GPU Unterstützung wird über den Parameter „wmode“ bei der Einbettung von Flash in eine HTML Seite gesteuert. Es gibt dafür neben den Werten „normal“, „transparent“ und „opaque“ zwei neue Werte, nämlich „direct“ und „gpu“ (vgl. [URO08]).

Direct

Hier wird versucht die Daten so schnell wie möglich auf den Bildschirm zu bringen, ohne dabei auf den Browser Rücksicht zu nehmen. Überlappende HTML-Menüs könnten dabei zB nicht mehr funktionieren. Ein typischer Anwendungsfall für diesen Modus ist die Wiedergabe von Videos.

Gpu

Dabei werden die Elemente von Flash noch per Software gerendert, aber die Zusammensetzung erfolgt bereits mit Hilfe der GPU. ZB ist das Skalieren von Videos, wenn möglich, bereits hardwareseitig. In weiteren Flashplayerversionen werden wahrscheinlich immer mehr Funktionen von der Software auf die Hardware verlagert. Bei der Verwendung dieser Modi ist jedoch einiges zu beachten (vgl. [URO08]):

- Es kann nicht immer ein Geschwindigkeitsgewinn erzielt werden, da der Software-Renderer von Flash vieles besser optimiert, als es die GPU kann. Der Entwickler muss sich dabei immer bewusst sein, was mit der GPU möglich ist und was nicht. In den meisten Fällen wird der Inhalt sogar langsamer.

- Um die GPU von Flash aus ansprechen zu können, wird mindestens eine DirectX 9 fähige Grafikkarte benötigt.

- Die Pixelgenauigkeit ist beim „gpu“ Modus nicht garantiert. Das bedeutet, dass Inhalte auf verschiedenen Rechnern unterschiedlich aussehen können und auch die Farbe muss nicht immer exakt übereinstimmen.

- Die Bildwiederholungsrate ist auf die des Bildschirms beschränkt. Sie sollte in Flash in beiden Modi 50 bis 55 nicht überschreiten, um Frameverluste, die aus unterschiedlichen Gründen auftreten können, zu vermeiden.

- GPU-basierter Inhalt im Webbrowser ist immer sehr rechenintensiv und sollte daher auf einen Flashinhalt pro HTML-Seite beschränkt werden.

5.2.3. Pixel Bender

Pixel Bender ist eine Engine zur Grafikverarbeitung, unter anderem für den Flashplayer in der Version 10. Die Programmiersprache basiert auf Shader-Sprachen wie zB GLSL für OpenGL. In Flash kann sie verwendet werden, um Filter, Blendeffekte und Füllungen in Echtzeit zu berechnen.

Die Programme werden „Kernels“ genannt und mit dem „Pixel Bender Toolkit“ von Adobe erzeugt. Der produzierte Bytecode wird als „*.pbj“ Datei gespeichert und mit Hilfe von ActionScript zur Verwendung in eine SWF-Datei geladen (vgl. [PBFF08]).

Der Shader wird jedoch nicht von der GPU ausgeführt, sondern ist softwarebasiert. Als Grund werden Einschränkungen im GPU-Rendering angeführt, sodass in diesen Fällen sowieso nur eine softwarebasierte Ausführung möglich wäre. Außerdem wird erwähnt, dass User über entsprechende Hardware verfügen müsste und im Moment viele diese nicht hätten. Ein weiterer Grund ist, dass der Flashplayer in der Downloadgröße um zwei bis drei Megabyte wachsen würde. Als Vorteil kann jedoch gesehen werden, dass Pixel Bender in einem eigenen Thread abläuft und auch von mehrkernigen CPUs Gebrauch macht (vgl. [BPFP08]).

5.3. Flash Engines

Da es für Flash nicht nur eine 3D-Engine gibt, werden im Weiteren wichtige verfügbare Engines kurz vorgestellt. Folgende wurden durch Recherchen im Internet ermittelt:

- Sandy 3D (http://www.flashsandy.org)
- Papervision3D (http://www.papervision3d.org, http://blog.papervision3d.org)
- Away3D (http://away3d.com)
- Alternativa3D (http://alternativaplatform.com/en)

Vor diesen gab es bereits mehrere Versuche 3D-Flash-Engines zu realisieren. Einige dieser sind zB Five3D (vgl. [FIVE3D10]), 9elements, Illogicz, flashworx V4 oder 3DFS (vgl. [3DENG06]). Die meisten erreichen jedoch nicht den Umfang der oben genannten oder werden nicht mehr weiterentwickelt.

5.3.1. Sandy 3D

Sandy 3D wurde im Jahr 2005 durch Thomas Pfeiffer ins Leben gerufen und derzeit arbeiten vier Entwickler an diesem Projekt (vgl. [SANDY3D10], vgl. [SANDYTEAM10]). Es wurde aufgrund der geringen Möglichkeiten 3D in Flash auszuführen gegründet (vgl. [SANDYWHY10]). Sandy 3D ist Open Source und kann sowohl für private als auch für kommerzielle Projekte frei genutzt werden (vgl. [SANDYOS10]). Derzeit sind die Versionen 3.1.2 für ActionScript 3 und haXe sowie die Version 1.2 für ActionScript 2 verfügbar (vgl. [SANDY3D10]). Im Folgenden wird nur die Version 3.1.2 für ActionScript 3 untersucht.

5.3.2. Papervision3D

Die erste Public Beta von Papervision3D wurde im Juli 2007 veröffentlicht. Papervision3D fällt unter die MIT Lizenz und kann daher auch in kommerziellen Projekten frei verwendet werden (vgl. [PV3DFAQ08]). Dem Kernteam von Papervision3D gehören derzeit vier Mitglieder an (vgl. [PV3D10]). Es ist momentan in der Version 2.1 für ActionScript 3 und den Flashplayer 9 zum Download verfügbar (vgl. [PV3DDL10]). Die neuen Funktionen des Flashplayer 10 werden erst in der jetzt noch nicht offiziell verfügbaren Version 3.0 unterstützt (vgl. [PV3DV09]).

5.3.3. Away3D

Away3D ist aus Papervision3D hervorgegangen. Es entstand nicht aus einer Konkurrenzsituation heraus, sondern um eine Engine zu schaffen, die besser an die Bedürfnisse der Away3D Entwickler Alexander Zadorozhny und Rob Bateman angepasst ist. Dabei werden auch neue Features von Away3D wieder von Papervsion3D übernommen (vgl. [AWAY3D07]). Bereits im März 2007 wurde Away3D von Papervision3D abgespalten (vgl. [AWAY3DF09]). Neben den beiden Gründern arbeiten derzeit noch 12 weitere Mitarbeiter an Away3D (vgl. [AW3DTEAM10]).

Die verfügbaren Versionen sind 2.5 für den Flashplayer 9 (ActionScript 3) und 3.5 für den Flashplayer 10 (ActionScript 3). Da die Versionen 2.5 und 3.5 erst kurz vor dem Abschluss dieser Arbeit veröffentlicht wurden, werden nur die Version 2.4 und 3.4 berücksichtigt.

Außerdem ist eine vereinfachte Version namens Away3D Lite in der Version 1.0 vorhanden (vgl. [AW3DDL09], vgl. [AWAY3D10]). Away3D ist unter der Apache License 2.0 veröffentlicht und kann kostenlos verwendet werden (vgl. [AW3DLIC10]).

5.3.4. Alternativa3D

Die erste Version von Alternativa3D war im Juni 2008 verfügbar (vgl. [A3DSTART08]). Alternativa3D ist eine Engine, welche im Gegensatz zu den anderen Engines nicht kostenlos für kommerzielle Projekte eingesetzt werden kann. Allein für nicht kommerzielle Projekte ist sie frei verfügbar (vgl. [A3DLIC09]). Derzeit sind 21 Mitarbeiter bei Alternativa beschäftigt, wobei jedoch möglicherweise nicht alle an Alternativa3D arbeiten (vgl. [A3DABOUT10]). Aktuell ist die Version 5.6 für ActionScript 3 und Flash 10 erhältlich (vgl. [A3D10]).

5.4. Kriterien

Um die verschiedenen Engines zu vergleichen, werden unterschiedliche Kriterien, welche nach den Phasen in der Rendering Pipeline gegliedert sind, berücksichtigt. Die Informationen stammen entweder direkt von der Webseite der Engines, aus der API-Dokumentation, sowie aus eigenen Tests. Andere Quellen werden gesondert angegeben.

5.4.1. Allgemein

Dokumentation und Support

Dokumentation und Support				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
API-Dokumentation	Sehr gut	Gut	Gut	Gut
Tutorials	Sehr gut	Befriedigend	Gut	Befriedigend
Forum	Ja	Ja	Ja	Ja
Chat	Nein	Ja	Nein	Nein
Bücher	Nein	Ja	Nein	Nein
Schulungen	Nein	Ja	Ja	Nein

Tabelle 19: Allgemein - Dokumentation und Support

Begründungen und Anmerkungen

Die Beurteilung der Kriterien „API-Dokumentation“ und „Tutorials“ erfolgt nach dem österreichischen Schulnotensystem mit den Noten von „Sehr gut“ bis „Nicht genügend“.

Bei Sandy 3D sowie Alternativa3D sind die Klassen, Methoden und Attribute in der API-Dokumentation sehr gut beschrieben, wobei bei Away3D und Papervision3D ein paar Lücken vorhanden sind.

Die zahlreichen Tutorials von Sandy 3D befinden sich direkt auf der Webseite und sind sehr gut strukturiert. Away3D hat eine nach Themen geordnete Linkliste zu externen Tutorials. Papervision3D bietet dagegen nur Verlinkungen auf externe Tutorial-Seiten. Alternativa3D verfügt auf der Webseite selbst nur über wenige Tutorials, es gibt jedoch ein paar externe. Anzumerken ist, dass bei Alternativa3D auch Informationen auf Russisch vorhanden sind, die in dieser Arbeit nicht berücksichtigt werden.

Bei Papervision3D ist hervorzuheben, dass es für diese Engine einen Chat (http://pv3dchat.flashbookmarks.com) gibt. Außerdem sind für Papervision3D die folgenden zwei Bücher verfügbar:

- „Papervision3d Essentials“ von Paul Tondeur und Jeff Winder
- „Professional Papervision3D“ von Michael Lively

Allgemeine Eigenschaften

Allgemeine Eigenschaften				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Primitive	Box, Cone, Cylinder, Geodesic Sphere, Hedra, Line3D, Plane3D, SkyBox, Sphere, Torus	Arrow, Cone, Cube, Cylinder, PaperPlane, Plane, Sphere	BezierPatch, Cone, Cube, Cylinder, GeodesicSphere, GridPlane, LineSegment, Plane, ReflectivePlane, RegularPolygon, RoundedCube, SeaTurtle, Skybox, Skybox6, Sphere, TextField3D, Torus, Triangle, Trident, WireCircle, WireCone, WireCube, WireCylinder, WirePlane, WireSphere, WireTorus	Box, Cone, GeoPlane, GeoSphere, Plane, Sphere
Eigene Objekte (Vertices)	Ja	Ja	Ja	Ja
Eigene Objekte (Extrude)	Ja	Nein	Ja	Nein
3D-Text	Nein	Ja	Ja	Nein
Importierbare Formate	ASE, Collada, MD2, 3DS	ASE, Collada, MD2, 3DS, KMZ, DAE, Sketchup Collada	ASE, Collada, MD2, 3DS, KMZ, MD2still, OBJ, SWF	3DS, OBJ
Billboarding	Ja	Nein	Ja	Ja
Partikelsysteme	Ja	Ja	Ja	Nein

Tabelle 20: Allgemein - Allgemeine Eigenschaften

Begründungen und Anmerkungen

Primitive

Alle Engines bieten eine gewisse Anzahl von vordefinierten Primitiven.

Eigene Objekte

Unter eigenen Objekten werden solche verstanden, die von den vorhandenen Primitiven abweichen. Sie können zB durch das Extrudieren von Pfaden oder das Zusammenbauen von Oberflächen, durch einzelne Dreiecke und das Definieren von Vertices entstehen.

Hier bieten alle Engines entsprechende Möglichkeiten. Bei Sandy 3D können sowohl eigene Objekte durch das Extrudieren von Pfaden, als auch durch die Definition von Punkten und Dreiecken (vgl. [SANDYF10]) erstellt werden. Bei Papervision3D können Objekte nur durch Definition von Dreiecken und Punkten erstellt werden, die Funktionalität zum Extrudieren von Pfaden konnte bei Papervision3D nicht gefunden werden. Away3D bietet laut API-Dokumentation beide Möglichkeiten und bei Alternativa3D können komplexe Objekte durch die Definition von Punkten erstellt werden.

3D-Text

Papervision3D und Away3D sind die einzigen Engines, die dreidimensionalen Text unterstützen.

Importierbare Formate

Es können bei allen Engines 3D-Modelle in den entsprechenden Formaten geladen werden. Dadurch ist es nicht nötig komplexere Modelle im Code selbst zu erstellen.

Billboarding und Partikelsysteme

Es existiert eine externe Bibliothek namens „Flint“ (http://flintparticles.org), welche Partikelsysteme anbietet. Papervision3D unterstützt neben der eigenen auch diese Bibliothek (vgl. [TOWI09], S. 312). Ebenso ist es in Away3D und Sandy 3D möglich Flint zu verwenden (vgl. [AW3DPA08]). Wie in einem Forumsbeitrag auf der Webseite zu lesen ist, gab es zumindest im Februar 2009 noch keine Unterstützung für Alternativa3D (vgl. [AT3DFL09]). In Papervision3D ist selbst kein Billboarding vorhanden, das Partikelsystem kann jedoch dafür verwendet werden.

Interaktivität

Interaktivität				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Maus	Ja	Ja	Ja	Ja
Tastatur	Ja	Ja	Ja	Ja
Objekte	Ja	Ja	Ja	Ja
Material	Ja	Ja	Ja	Ja
3D-Positonserken-nung	Ja	Ja	Ja	Ja
UV-Positionserken-nung	Ja	Ja	Ja	Ja

Tabelle 21: Allgemein - Interaktivität

Begründungen und Anmerkungen

Interaktionen mit Maus und Tastatur sind in allen Engines vorhanden und können entweder auf Objekte selbst oder auf Materialien und Bereiche von Materialien, welche auf die Objekte gelegt wurden, angewendet werden.

Mit allen Engines ist es möglich den Punkt im dreidimensionalen Raum, auf den geklickt wurde, zu ermitteln. Mit Sandy 3D, Away3D und Alternativa3D und Papervision3D (vgl. [PV3UV10]) können ebenso die UV-Koordinaten dieses Punkts auf dem Material ausgegeben werden.

Kamera

Kamera				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Kameratypen	Camera3D, Mode7, Spring	Target, Free, Debug, Spring	Camera3D, Target, Hover, Spring	Camera3D
Kameraeigen-schaften	near plane, far plane, fov, focal length	near plane, far plane, fov, focus, zoom	near plane, far plane, fov, focus, zoom	near plane, far plane, fov, focal length, zoom

Tabelle 22: Allgemein - Kamera

Begründungen und Anmerkungen

Es gibt bei fast allen Engines mehrere Arten von Kameras, die für unterschiedliche Aufgaben verwendet werden können. Eine „Target-Kamera“ kann auf ein Objekt in der Szene fixiert werden und behält dieses, im Gegensatz zu einer freien Kamera, immer im Bild.

Die „Hover-Kamera“ schwebt um ein Zielobjekt und eine „Spring-Kamera“ verfolgt das Zielobjekt bei seinen Bewegungen. Die „Debug-Kamera“ besitzt Funktionen, mit denen ihre Einstellungen über Tastatur und Maus geändert werden können und gibt dabei Informationen über die Kamera selbst aus.

Die Kamera der Alternativa3D Engine besitzt Controller, die zB ebenso Funktionen bieten, mit denen die Kamera bewegt, rotiert und auf ein bestimmtes Ziel gerichtet werden kann. Der Mode7, welcher in Sandy 3D ausgewählt werden kann, hat ein paar Beschränkungen gegenüber der Standardkamera, so sind mit ihm zB nur Rotationen um die Y-Achse möglich. Insgesamt bieten alle Engines ca. dieselben Einstellungsmöglichkeiten für Kameras.

5.4.2. Application-Phase

Allgemein

Allgemein				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Physik und Kollisionsüber-prüfung	Ja	Ja	Ja	Nein
Animationen	Ja	Ja	Ja	Ja
Morphing	Ja	Ja	Ja	Ja

Tabelle 23: Application-Phase - Allgemein

Begründungen und Anmerkungen

Physik und Kollisionsüberprüfung

Die WOW Engine (http://seraf.mediabox.fr/wow-engine/as3-3d-physics-engine-wow-engine) simuliert physikalische Effekte und kann zusammen mit Sandy 3D, Papervision3D und Away3D eingesetzt werden. Für Alternativa3D konnte kein Beispiel mit WOW gefunden werden und laut einem Forumsbeitrag beinhaltet Alternativa3D auch keine eigene Physik-Engine (vgl. [AT3DKO08]). Zur Zeit des Schreibens dieser Arbeit war die Webseite der WOW Engine nicht mehr erreichbar.

Animationen

Es ist möglich bereits vorhandene Funktionen von ActionScript zu nutzen. Es gibt jedoch auch externe Bibliotheken, die für Animationen verwendet werden können. Eine dieser Bibliotheken nennt sich „Tweener“ (http://code.google.com/p/tweener). Sie kann mit Papervision3D (vgl. [TOWI09], S. 180), Away3D (vgl. [AW3DAN08]), Alternativa3D (vgl. [AT3DAN09]) und Sandy 3D verwendet werden.

Morphing

Auch für Morphing können externe Bibliotheken benutzt werden. Eine davon ist „AS3Dmod“ (http://code.google.com/p/as3dmod). Wie auf der Webseite von „AS3Dmod“ erwähnt wird, funktioniert sie für alle hier untersuchten Engines.

Culling

Culling				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Hierarchical View Frustum Culling	Ja	Ja	Ja	Nein
Backface Culling	Ja	Ja	Ja	Ja
Occlusion Culling	Nein	Nein	Ja	Nein

Tabelle 24: Application-Phase - Culling

Begründungen und Anmerkungen

Alle Engines bis auf Alternativa3D (vgl. [AT3DFR08]) beinhalten einen Frustum und erlauben somit das Entfernen von Objekten, die außerhalb des Frustums liegen. Backface Culling wird von allen Engines geboten, bei Alternativa3D ist Backface Culling immer aktiviert (vgl. [AT3DFOR10]). Away3D bietet Occlusion Culling (vgl. [AWAYOCC07]). Von Papervision3D (vgl. [TOWI09], S. 151) und Sandy 3D (vgl. [SANDYF10]) wird dies nicht unterstützt. Alternativa3D bietet Occlusion Culling erst in Version 7 (vgl. [AT3DFOR10]).

Bildbasierende Effekte

Bildbasierende Effekte				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
LOD	Nein	Ja	Ja	Nein
Depth of Field	Nein	Nein	Ja	-
Motion Blur	Nein	Nein	Nein	Nein

Tabelle 25: Application-Phase - Bildbasierende Effekte

Begründungen und Anmerkungen

LOD

Sandy 3D unterstützt LOD nicht, es werden allein Objekte, die hinter der Far-Plane liegen, nicht mehr gerendert. Bei Papervision3D können mehrere Objekte, zwischen denen gewechselt werden soll, sowie deren Verwendungszeitpunkt angegeben werden (vgl. [PV3DLOD08]). Away3D war die erste Flash-Engine die LOD unterstützt hat (vgl. [AW3DLOD08]). Alternativa3D wird LOD erst mit der Version 7.0 unterstützen (vgl. [AT3DLOD09]).

Depth of Field

Papervision3D und Sandy 3D bieten Depth of Field nur durch die Nutzung von in ActionScript 3.0 bereits vorhandenen Filtern (Blur). Away3D bietet hingegen eine eigene Klasse, um diesen Effekt zu erzeugen. Er basiert auf Billboards, wobei unterschiedliche Schärfestufen von einem Billboard gespeichert und entsprechend angezeigt werden (vgl. [AW3DDOF09]). Für Alternativa3D kann keine Aussage über Depth of Field gemacht werden.

Motion Blur

Motion Blur kann in allen Engines nur durch das Verwenden von in ActionScript 3.0 vorhandenen Filtern erzeugt werden.

5.4.3. Geometry-Phase

Koordinatensystem

Koordinatensystem				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Koordinatensys-tem	linkshändig	linkshändig	linkshändig	rechtshändig

Tabelle 26: Geometry-Phase - Koordinatensystem

Begründungen und Anmerkungen

Alle Engines benutzen das linkshändige Koordinatensystem, allein Alternativa3D verwendet das rechtshändige.

Lichtarten

Lichtarten				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Direktes Licht	Ja	Nein	Ja	Nein
Punktlicht	Nein	Ja	Ja	Nein
Spotlicht	Nein	Nein	Nein	Nein
Umgebungslicht	Nein	Nein	Ja	Nein
Mehrere Lichtquellen	Nein	Ja	Ja	Nein

Tabelle 27: Geometry-Phase - Lichtarten

Lichtparameter

Lichtparameter				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Farbe	Ja	Nein	Ja	Nein
Position	Nein	Ja	Ja	Nein
Richtung	Ja	Nein	Nein	Nein
Intensität	Ja	Nein	Ja	Nein

Tabelle 28: Geometry-Phase - Lichtparameter

Begründungen und Anmerkungen

Bei Sandy 3D gibt es nur eine Lichtquellenart, nämlich direktes Licht. Es kann keine Position angegeben werden, sondern nur eine Richtung, in die das Licht scheinen soll. Dabei können sowohl die Farbe wie auch die Intensität bestimmt werden.

Papervision3D hingegen hat nur die Möglichkeit Punktlichter zu definieren. Dabei kann die Position des Lichts angegeben werden. Die Farbe des Lichts wird über die Eigenschaften des Shading-Materials definiert. Es ist ebenso möglich mehrere Lichtquellen zu verwenden, wobei jedoch nur eine für das Shading benutzt werden kann (vgl. [TOWI09], S. 219).

Away3D bietet die größte Auswahl an Lichtquellen. Hier können auch mehrere Lichtquellen definiert werden. Mit Away3D kann direkt bei den Lichteigenschaften die Intensität für die Komponenten Ambient, Diffuse und Specular eingestellt werden und muss nicht bei jedem Material extra definiert werden. Die Eigenschaften des Lichts sind bei Away3D im Gegensatz zu Sandy 3D und Papervision3D vom Material getrennt. Beim Punktlicht nimmt bei Away3D die Intensität mit der Entfernung zur Lichtquelle ab (vgl. [AW3DLIGHT08]).

Alternativa3D erlaubt es keine eigenen Lichtquellen zu definieren. Anstatt dessen können allein Light Maps verwendet werden (vgl. [AT3DLIGHT07]).

Shading

Shading				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Flat	Ja	Ja	Ja	Nein
Gouraud	Ja	Ja	-	Nein
Phong	Ja	Ja	Ja	Nein
Cell	Ja	Ja	Nein	Nein

Tabelle 29: Geometry-Phase - Shading

Material

Material				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Farbe	Ja	Ja	Ja	Ja
Bilder	Ja	Ja	Ja	Ja
MovieClips	Ja	Ja	Ja	Ja
Videos	Ja	Ja	Ja	Nein
Wireframe	Ja	Ja	Ja	Ja

Tabelle 30: Geometry-Phase - Material

Materialparameter

Materialparameter				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Ambient	Ja	Ja	Ja	Nein
Diffuse	Ja	Nein	Ja	Nein
Specular	Ja	Ja	Ja	Nein
Gloss	Ja	Nein	-	Nein

Tabelle 31: Geometry-Phase - Materialparameter

Begründungen und Anmerkungen

Shading

Bei allen Engines, außer bei Alternativa3D, können verschiedene Shading-Algorithmen genutzt werden. Bei Away3D konnte nicht ermittelt werden, ob Gouraud-Shading möglich ist.

Material

Farben, Bilder, MovieClips und Wireframe können bei allen Engines als Material definiert werden. Videos können nur bei Alternativa3D nicht als Material verwenden werden (vgl. [AT3DFLV09]).

Materialeigenschaften

Bei Sandy 3D werden diese Eigenschaften in eigenen Klassen definiert und dann dem Material zugewiesen. Dabei können auch Attribute von mehreren Shadern kombiniert werden, zB die Diffuse Komponente vom Phong-Shader und die anderen Komponenten vom Gouraud-Shader.

In Papervision3D werden die möglichen Eigenschaften direkt beim Material definiert. Gleichzeitig muss einem Shading-Material auch eine Lichtquelle zugewiesen werden. Das Zuweisen von mehreren Lichtquellen ist nicht möglich. Bei Away3D werden „ambient“, „diffuse“ und „specular“ beim Licht angegeben. Ob bei Away3D der Parameter „gloss“ definiert werden kann, konnte nicht ermittelt werden.

Allgemein

Allgemein				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Schatten	Nein	Nein	Ja	-
Nebel	Ja	Ja	Ja	Nein

Tabelle 32: Geometry-Phase – Allgemein

Begründungen und Anmerkungen

Schatten

In keiner der Engines werden Echtzeit-Schatten in Abhängigkeit der Lichtquelle unterstützt, wobei es einige Versuche gibt Schatten zu implementieren, zB von Li, einem Teammitglied von Away3D (http://www.lidev.com.ar/?p=79) oder von Andy Zupko, einem der Mitglieder von Papervision3D (http://blog.zupko.info/?p=146). Die Versuche dazu wurden bisher aber noch nicht in die Engines übernommen.

In Away3D gibt es zumindest die Möglichkeit einfache Schatten zu definieren. Dabei werden die Schatten von Objekten jedoch nur in die Textur einer Ebene integriert, ohne sie wirklich auf eine Ebene, in Abhängigkeit einer Lichtquelle, zu projizieren. Für Alternativa3D konnte nicht ermittelt werden, ob zumindest die Darstellung von einfachen Schatten möglich ist.

Nebel

Die Möglichkeit Nebel darzustellen beherrschen bis auf Alternativa3D (vgl. [AT3DFOR10]) alle Engines.

Projektion

Projektion				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Orthographic	Nein	Ja	Ja	Ja
Perspective	Ja	Ja	Ja	Ja

Tabelle 33: Geometry-Phase - Projektion

Begründungen und Anmerkungen

Bei Papervision3D, Away3D und Alternativa3D ist es möglich sowohl eine orthotgrafische Projektion als auch eine perspektivische Projektion zu verwenden. Allein Sandy 3D bietet hier nur eine perspektivische Projektion (vgl. [SANDYF10]).

Clipping

Clipping				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Frustum Clipping	Ja	Ja	Ja	Ja
Nearfield Clipping	Ja	Ja	Ja	Ja

Tabelle 34: Geometry-Phase - Clipping

Begründungen und Anmerkungen

Sandy 3D ermöglicht es Objekte gegen alle Ebenen des Frustums oder auch nur gegen die Near Plane abschneiden zu lassen. Bei Papervision3D kann angegeben werden, gegen welche Ebene des Frustums die Objekte beschnitten werden sollen. Clipping kann aber auch auf den gesamten Frustum angewandt werden. Away3D ermöglicht es sowohl gegen den Frustum als auch nur gegen die Near Plane zu clippen.

Alternativa3D hat keinen Frustum, bietet aber trotzdem View-, Near- und Far-Clipping. Wobei die Entfernungen für das Near- und Far-Clipping bei den Eigenschaften der Kamera definiert werden können. View Clipping wird gegen die pyramidenförmige Sicht der Kamera durchgeführt.

5.4.4. Rasterizer-Phase

Texturierung

Texturierung				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Mipmapping	Ja	Ja	Ja	-
Bitmap Tiling	Ja	Ja	Ja	Nein

Tabelle 35: Rasterizer-Phase - Texturierung

Begründungen und Anmerkungen

Mit der Version 9 (Update 3) des Flashplayers wurde Mipmapping eingeführt. Der Flashplayer generiert automatisch kleinere Versionen der Originalbilder und wendet diese bei verkleinerten Darstellungen an. Wichtig zu erwähnen ist, dass nur Mipmaps von Bildern mit gerader Höhe und Breite generiert werden. Sobald bei einer der Verkleinerungen eine ungerade Zahl vorkommt, wird das Mipmapping abgebrochen. So ist es ratsam, nur Höhen und Breiten mit Größen von $2^{\wedge n}$ zu verwenden, zB 512x256 Pixel (vgl. [FLMM07]). Alternativa3D hat eine eigene Mipmapping-Methode entwickelt, welche auch eine ungerade Anzahl von Pixeln in Höhe und Breite von Bildern erlaubt. Es ist aber unklar, ob diese Methode in der aktuellen Version von Alternativa3D bereits verfügbar ist (vgl. [AT3DMM09]).

Bitmap Tiling wird von Sandy 3D, Papervision3D und Away3D unterstützt. Bei Alternativa3D kann Bitmap Tiling nicht explizit eingestellt werden (vgl. [AT3DFOR10]).

Texturierungsmethoden

Texturierungsmethoden				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Alpha Mapping	Ja	Ja	Ja	Ja
Environment Mapping	Ja	Ja	Ja	Nein
Bump Mapping	Nein	Ja	Ja	Nein

Tabelle 36: Rasterizer-Phase - Texturierungsmethoden

Begründungen und Anmerkungen

Alpha Mapping

Alle Engines erlauben das Verwenden einer Textur mit transparenten Bereichen (zB MovieClips) und somit die Möglichkeit für Alpha Mapping.

Environment Mapping

Bis auf Alternativa3D (vgl. [AT3DFOR10]) wird von allen Engines Environment Mapping geboten.

Bump Mapping

In Sandy 3D ist Bump Mapping vorgesehen, wird aber derzeit noch nicht unterstützt (vgl. [SANDYBM10]). Bei Papervision3D wird ein Heightfield verwendet. Dies kann nicht nur ein Graustufen-, sondern auch ein Farbbild sein (vgl. [TOWI09], S. 228). Away3D bietet Bump Mapping auf Basis von Normal Maps (vgl. [AW3DBM09]). Alternativa3D hat kein Bump Mapping (vgl. [AT3DBUMP09]).

Visible-Surface-Algorithmen

Visible-Surface-Algorithmen				
Kriterien	**Sandy 3D**	**Papervision3D**	**Away3D**	**Alternativa3D**
Depth Sorting	Ja	Ja	Ja	Nein
Quad-Tree	Nein	Ja	Ja	Nein
BSP-Tree	Nein	Nein	Nein	Ja

Tabelle 37: Rasterizer-Phase - Visible-Surface-Algorithmen

Begründungen und Anmerkungen

Da Flash die GPU nur in sehr geringem Ausmaß unterstützt, besteht keine Möglichkeit auf den Z-Buffer der GPU zuzugreifen. Die gesamte Berechnung muss softwarebasiert erfolgen.

Sandy 3D

Sandy 3D verwendet, wie auf der Webseite zu lesen ist, zum Sortieren einen Mean-Algorithmus in Zusammenhang mit dem Painter's Algorithmus. Es kann zwischen Objekt- und Polygon-Sortierung gewählt werden. Die Objekt-Sortierung ist schneller, aber sie kann nicht mit sich überschneidenden Objekten umgehen und dadurch kann die Sortierung fehlerhaft sein. Der Algorithmus auf Polygonbasis hingegen funktioniert auch mit sich überschneidenden Objekten, ist dafür aber langsamer.

Es wird bei beiden Varianten für jedes Polygon oder Objekt ein transformierter Mittelwert der Tiefe „Z" errechnet und die Polygone bzw. Objekte werden auf Basis dieses Mittelwertes sortiert. Die entsprechende Konsequenz daraus ist, dass große Objekte oder Polygone keinen repräsentativen Mittelwert besitzen und es zu Darstellungsfehlern kommen kann. In der aktuellen Version wird experimentell auch BSP-Sortierung angeboten.

Papervision3D

Bei dieser Engine gibt es die Möglichkeit verschiedene Algorithmen zu verwenden. Standardmäßig wird bei Papervision3D der Painter's Algorithmus (BasicRenderEngine) auf der Basis von Dreiecken verwendet (vgl. [TOWI09], S. 272ff), aber es gibt auch die Option, „QuadTrees" (QuadrantRenderEngine) für die Sortierung einzusetzen. Papervision3D hat „QuadTrees" von Away3D übernommen und diese an ihre eigene Engine angepasst (vgl. [PV3DQT08]).

Es gibt unterschiedliche Optionen (vgl. [PV3DUQ08]):

- **CORRECT_Z_FILTER**
  Hier werden die Dreiecke nach Möglichkeit richtig sortiert, aber sich überschneidende Bereiche werden nicht korrekt behandelt.

- **QUAD_SPLIT_FILTER**
  Diese Option funktioniert speziell mit sich überschneidenden Dreiecken und unterteilt diese wenn nötig. Danach ist jedoch noch immer eine Sortierung der neu entstandenen Dreiecke nötig.

- **ALL_FILTERS**
  Dies ist eine Kombination aus beiden oben genannten Filtern. Die Dreiecke werden richtig sortiert, egal ob sie sich überschneiden oder nicht.

Da die Berechnungen im Zusammenhang mit „QuadTrees“ die Performance sehr beeinträchtigen, kann für jedes Objekt die Überprüfung deaktiviert werden. Andy Zupko erwähnt in seinem Blog auch, dass die CPU durch die Verwendung dieses Algorithmus sogar zerstört werden kann.

Away3D

Da Papervision3D den „QuadTree“-Algorithmus von Away3D übernommen hat, gibt es logischerweise auch bei Away3D die Option, neben dem Painter’s Algorithmus (vgl. [AWAYZS10]) für die Sortierung den „QuadTree“-Algorithmus einzusetzen (vgl. [PV3DQT08]).

Folgende Möglichkeiten sind beim „QuadTree“-Algorithmus vorhanden:

- **CORRECT_Z_ORDER**
  Nach dem Sortieren der Dreiecke werden diese nochmals neu geordnet, um das korrekte Rendern zu garantieren.

- **INTERSECTING_OBJECTS**
  Bei sich überschneidenden Dreiecken werden diese aufgeteilt, um eine richtige Darstellung zu ermöglichen.

Alternativa3D

Alternativa3D verwendet laut Beschreibung auf der Webseite zur Sortierung einen BSP-Baum. Es werden einige Parameter angeboten, die der Entwickler selbst beeinflussen kann, je nachdem, ob Genauigkeit oder Schnelligkeit wichtiger ist. ZB kann für ein Objekt bestimmt werden, ob es näher oder weiter vom Root-Knoten des Baums entfernt sein soll. Es ist auch möglich die Anzahl der Teilungen, um den Baum aufzubauen, zu reduzieren. Andere Algorithmen sind nicht implementiert (vgl. [AT3DFOR10]).

Andere Engines

Interessant zu erwähnen ist, dass es bereits 3D-Engines mit anderen Ansätzen gab (vgl. [SWFZ08]). Bei der SWFZ Engine wird ein „Image Precision“-Algorithmus nach dem Scanline-Prinzip verwendet. Aufgrund der hohen Performance-Anforderungen ist die Auflösung auf 320x240 Pixel beschränkt. Die Webseite der Engine ist jedoch nicht mehr erreichbar.

6. Implementierung

Nachstehend wird das Einsatzgebiet des Prototyps, der Bereich der Informationsvisualisierung umrissen, um die Anforderungen an den Prototyp zu ermitteln. Diese Anforderungen sind mitbestimmend für die Auswahl der Flash-Engine, mit der der Prototyp umgesetzt wird. Danach wird die Implementierung des Prototyps beschrieben und es werden die dabei auftretenden Probleme genannt und untersucht, ob bei den anderen Engines bzw. Formaten diese auch vorkommen.

6.1. Voraussetzungen

Wolfgang Kienreich und Michael Granitzer beschreiben in ihrem Artikel „Visualising Knowledge Webs for Encyclopedia Articles“ eine Art der Darstellung von Daten im dreidimensionalen Raum (vgl. [KIGR05]).

Große Ansammlungen von Daten sind zB in Enzyklopädien zu finden. Sie beinhalten unzählige Artikel aus den unterschiedlichsten Gebieten, welche meist alphabetisch geordnet sind. Das Finden der passenden Inhalte erfolgt durch eine textbasierte Suche, aber die benutzerfreundliche Strukturierung der Daten gestaltet sich aufgrund der allumfassenden Inhalte einer Enzyklopädie als eher schwierig. Da jedoch Artikel in Enzyklopädien oft entsprechende Metadaten, welche den Artikel selbst genauer beschreiben, besitzen, ist es möglich mit Hilfe dieser eine Strukturierung zu schaffen.

In der Enzyklopädie von Brockhaus gibt es erste Bemühungen, eine Struktur in die Organisation der Inhalte zu bringen. So existiert zu beinahe jedem Artikel ein Knowledge Web, welches einen Artikel in Beziehung zu ähnlichen Artikeln bringt und eine Quantifizierung des Grades dieser Beziehung bietet. Dadurch können ähnliche Artikel schnell identifiziert und mit Hilfe einer Navigation angesteuert werden, anstatt die Suchbegriffe textuell verfeinern zu müssen. In der Brockhaus-Enzyklopädie gibt es zu jedem Artikel ca. 25 verwandte Artikel. Die Knowledge Webs werden dabei halbautomatisch erzeugt. Dies bedeutet, dass sie nach der automatischen Generierung manuell editiert werden, um eine gute Qualität der Knowledge Webs sicherzustellen. Damit eine geeignete Darstellungsform für Knowledge Webs gefunden werden kann, sind weitere Überlegungen nötig.

Grundsätzlich können, zur Visualisierung von Daten bzw. Informationen im Web, bestimmte Methoden verwendet werden. Technologische Überlegungen spielen aufgrund der Nähe von Informationsvisualisierungen zu Informationstechnologien dabei meist eine große Rolle. Vor der Auswahl des Formats ist es jedoch wichtig zu entscheiden, welche Repräsentationsform gewählt wird. Dabei werden verschiedene Formen unterschieden:

- **Formalismen**
  Visuelle Formalismen verwenden diagrammartige Repräsentationen, um Informationen auf eine abstrakte Art darzustellen. Dabei wird die Art der Darstellung vom Designer auf Grund der darzustellenden Informationen selbst gewählt. Da sie auf nichts real Existierendes aufbaut, kann sie auf jegliche Information angewandt werden, mit dem Nachteil, dass die User diese Formalismen erst verstehen lernen müssen.

- **Methapern**
  Visuelle Methapern bauen bei der Darstellung auf etwas real Vorhandenes auf. Die verwendete Semantik wird durch das Äquivalent in der realen Welt bestimmt. So impliziert zB bei der Verwendung einer Landschaft als Darstellungsform, eine örtliche Nähe auch eine gewisse Verbundenheit der visualisierten Information. Dadurch, dass die User nicht erst die verwendete Semantik erlernen müssen, ist ein intuitives Verständnis möglich.

- **Modelle**
  Bei visuellen Modellen beruht die Information, die dargestellt werden soll, selbst auf ein in der realen Welt vorkommendes Äquivalent. Geospatiale Information kann zB gut durch ein dreidimensionales Modell visualisiert werden, welches durch die Information vorgegeben ist und vom User leicht verstanden wird. Oft ist aber gerade auch bei Informationen, die auf einem realen Äquivalent beruhen, wie zB bei historischen Gebäuden, die Darstellung nicht so einfach, da viele weitere Informationen, wie zB die geschichtliche Entwicklung, gezeigt werden müssen.

- **Diversität**
  Bei den meisten Informationen kann nicht nur eine visuelle Repräsentationsform gewählt werden. Eine Mischung aus Formalismen, Metaphern und Modellen ist dann durchaus sinnvoll. Dabei gibt es eine Hauptrepräsentationsform, die die meisten Informationen visualisiert. Weitere Informationen werden durch eine der anderen Formen ergänzt.

Um Knowledge Webs darzustellen, eignet sich am besten eine Metapher als Repräsentationsform, da es kein in der realen Welt vorkommendes Äquivalent zu den Beziehungen der Artikel gibt und außerdem die geringe Anzahl an darzustellenden Objekten keinen komplexen Formalismus benötigt (vgl. [KIEN06]).

Daher wurde vom Know-Center eine dreidimensionale drehbare Scheibe entwickelt, welche aus verschiedenfarbigen, nach Inhalten gegliederten Sektoren besteht. Diese aus dem Forschungsbereich stammende Visualisierung eignet sich sehr gut zur Verwendung für den Prototyp. Sie wird bereits als Produkt in der Enzyklopädie von Brockhaus eingesetzt und bietet daher eine gewisse Userbasis und auch Verbreitung. Es wird versucht diese mit der noch auszuwählenden Engine umzusetzen. Dabei werden die auftretenden Probleme und auch Vorteile entsprechend dokumentiert.

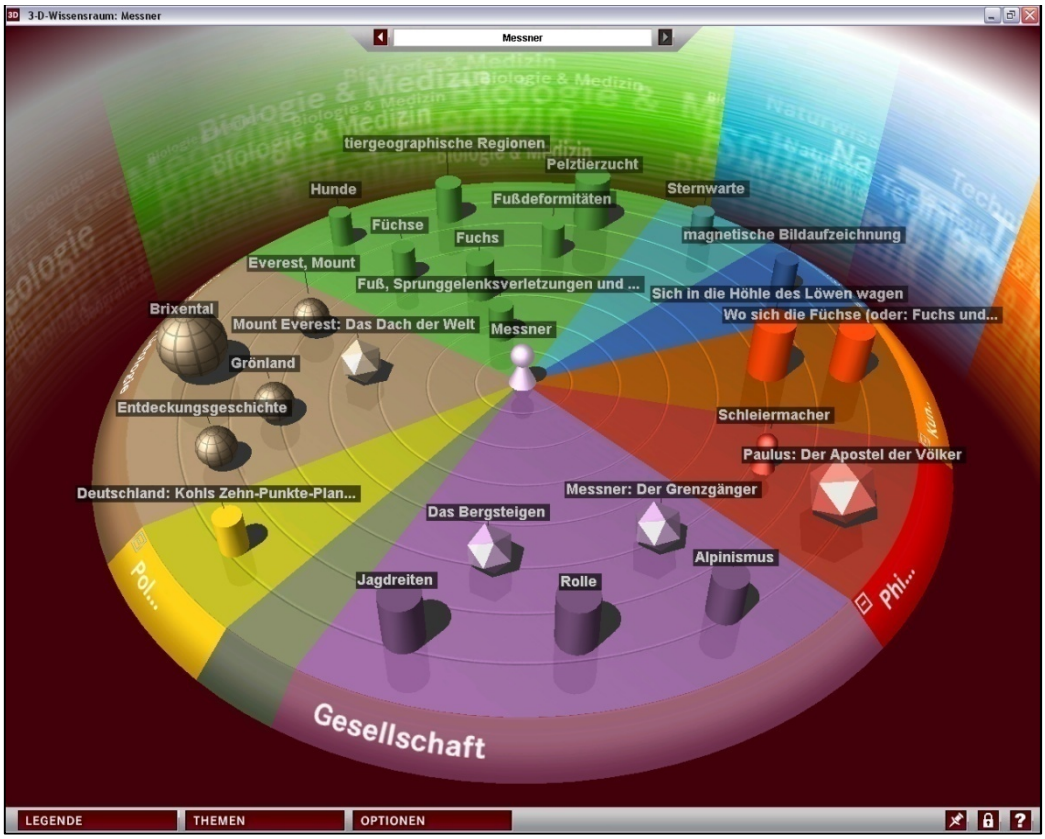


Abbildung 5: Wissensscheibe von Brockhaus [KIKRA08]

Wie Abbildung 5 zeigt werden die unterschiedlichen Artikel durch verschiedene geometrische Formen repräsentiert. Der gesuchte Artikel wird dabei in die Mitte der Scheibe gesetzt und alle weiteren werden, geordnet nach abnehmendem Relationsgrad, weiter von der Mitte entfernt positioniert. Gleichzeitig werden sie in den zugehörigen Themensektor platziert. Die Distanz der Artikel untereinander hat keine weitere Bedeutung. Um die einzelnen Formen zu identifizieren, wird der Artikelname über dem Objekt angezeigt. Zur Verbesserung der Darstellung wird eine Lichtquelle eingesetzt, welche Schatten der Objekte auf die Scheibe wirft. Gleichzeitig kann die Scheibe gedreht werden und es ist möglich sie zu vergrößern und zu verkleinern.

6.2. Anforderungen an den Prototypen

Aus dieser Beschreibung leiten sich die Anforderungen an den Prototypen ab, der das Ergebnis dieser Arbeit darstellen soll. Die Kriterien für den Prototyp sind:

- **Darstellen von einfachen Objekten**
  Es müssen zumindest Zylinder, Kugeln, Würfel und Kegel verwendet werden können. Weitere Primitive sind zur Darstellung von neuen Artikelarten von Vorteil.

- **Zuweisen von Material zu Objekten**
  Den Objekten muss eine Farbe zugewiesen werden können und das Anwenden eines Bildes auf die Wissensscheibe ist nötig.

- **Darstellen von Licht und Schatten**
  Es existiert eine fixe Lichtquelle, die die Schatten der Objekte auf die Wissensscheibe wirft.

- **Text**
  Um die Namen der Artikel über den Objekten darstellen zu können, wird die Darstellung von Text benötigt. Dieser Text soll beim Drehen der Scheibe immer zum User zeigen, damit er lesbar bleibt. Daher wird hier idealerweise Billboarding vorausgesetzt.

- **Importieren von 3D-Formaten**
  Die Wissensscheibe liegt in einem externen 3D-Format vor zB *.3ds, welches geladen und dargestellt werden können muss. Die Anforderung ist kein Ausschlusskriterium, da die Wissensscheibe auch ein Zylinder sein kann.

- **Interaktion**
  Es soll möglich sein, die Wissensscheibe mit Hilfe der Maus drehen und zoomen zu können.

- **Events**
  Beim Anklicken der einzelnen Objekte mit der Maus muss sich der dazugehörige Artikel in einem eigenständigen Fenster öffnen. Es ist nötig, dass eine Schnittstelle zu externen Inhalten vorhanden ist.

- **Animation**
  Die Scheibe dreht sich, wenn sie nicht vom User bewegt wird, automatisch langsam in eine Richtung. Daraus ergibt sich die Anforderung an eine zeitbasierte Animation.

- **Darstellungsqualität und Geschwindigkeit**
  Damit das Knowledge Web für einen Echteinsatz verwendbar ist, muss die Darstellungsqualität ansprechend sein, das heißt, es dürfen keine gravierenden Renderfehler auftreten. Ebenso ist die Geschwindigkeit von großer Bedeutung. Der Prototyp soll in guter Qualität flüssig ablaufen. Verzögerungen bei Interaktionen mit dem Prototyp sollen größtenteils vermieden werden. Dafür werden mindestens 20 FPS benötigt, wobei beim Fernsehen 25 Bilder pro Sekunde (PAL) verwendet werden (vgl. [MAHE05] S. 90).

Der Einfachheit halber wird auf die Sektoren, welche die Themen beschreiben, verzichtet. Ebenso gibt es keine Menüs, welche im Vordergrund die Auswahl von Themen, Optionen und einer Legende ermöglichen.

6.3. Darstellungsformat

Die Umsetzung des Knowledge Webs für die Brockhaus Enzyklopädie erfolgte mit Java und dem OpenGL Binding JOGL. Am Know-Center wurde eine Bibliothek namens „Appear“ entwickelt, um zweidimensionale Grafiken aus dreidimensionalen Szenen zu generieren. Sie basiert auf dem Szenengraphen-Prinzip und beinhaltet ein szenenbeschreibendes XML-Format, welches auch als Basis für die Umsetzung des Knowledge Webs mit Flash verwendet werden soll (vgl. [KIEN07]). Für den Prototyp in dieser Arbeit wird direkt die dreidimensionale Darstellung benutzt, ohne sie vorher zu einer zweidimensionalen Grafik zu rendern.

Ausgehend von einem Root-Knoten werden alle in der Szene vorkommenden Elemente in Form eines Graphen hierarchisch gegliedert. Der Graph besteht aus Transform Nodes, welche die Eigenschaften wie die Position, der darin vorkommenden Elemente, relativ zu den übergeordneten Knoten beschreiben. Standardmäßig werden Knoten vom Typ „scene“ verwendet.

Es gibt jedoch noch zwei spezielle weitere Typen. Zum einen sind das Knoten vom Typ „board“. Alle Elemente, die ihnen zugewiesen werden, zeigen immer dieselbe Seite in die Kamera, so auch bei Animationen. Zum anderen gibt es Knoten der Art „layer“. Sie agieren wie zweidimensionale Objekte. Wenn ein solcher Knoten sich innerhalb eines Knotens des Typs „scene“ befindet, ist er immer an der Stelle im Raum, an der sich auch der Knoten des Typs „scene“ befinden würde, wenn er zu einem zweidimensionalen Bild gerendert würde.

Die folgenden Elemente können den Transform Nodes zugewiesen werden und erben somit auch ihre Eigenschaften, wie Position und Skalierung:

Actors

Alle sichtbaren Elemente werden Actors genannt. Sie werden entweder in einem globalen Bereich definiert, der von allen Szenen aus zugänglich ist, oder sie werden in einem lokalen Bereich extra für jede Szene definiert und existieren auch nur in dieser. Dieser Bereich wird Actor Store genannt.

Visual

Ein Visual ist eine besondere Art eines Actors, welches ein dreidimensionales Objekt zusammen mit einem ihm zugewiesenen Material darstellt.

Light

Dies wird benötigt, um entsprechende Lichteffekte wie zB Schatten zu erzeugen. Objekte und die ihnen zugewiesenen Materialen reagieren entsprechend auf den Lichteinfall. Lichter werden wie Visuals den Transform Nodes zugewiesen und haben ebenso eine bestimmte Position im Raum.

Camera

Mit Hilfe der Kamera wird bestimmt, was in der Szene sichtbar ist. Von der angegebenen Position aus definiert sie ein entsprechendes Blickfeld. Es ist zur gleichen Zeit immer nur eine Kamera aktiv.

Media

Visuals bestehen aus Objekten und Materialien, welche Medien darstellen. Aber auch Farben für Materialien und Lichter sowie Bilder und Schriftarten zählen zu den Medien. Sie können ebenso global und lokal in einem sogenannten Media Store definiert werden.

Decorations

Sie werden verwendet, um die Darstellung einer Szene zu verbessern. Zu ihnen gehören zB Schatten und Reflektionen. Wenn Decorations einem Knoten zugeordnet werden, benötigen sie eine Referenz zu einem anderen Knoten, dem Source Node, welcher zB den Schatten auf die Visuals in dem Knoten, zu dem das Decor gehört, wirft.

Parameter

Sie stellen Platzhalter für Werte dar, die während der Darstellung der Szene geändert werden können. Es gibt numerische, textuelle und logische Typen, die in einem so genannten Param Store definiert werden. Abhängig vom Platz ihrer Definition gelten sie global oder lokal.

Events

Sie stellen in der Appear Bibliothek die Bereiche im resultierenden zweidimensionalen Bild dar, welche auf eine Interaktion von außen reagieren können und eine definierte Aktion auslösen.

6.4. Formatauswahl

6.4.1. Papervision3D

Der Prototyp wird mit Papervision3D umgesetzt. Ein wichtiger Grund dafür ist die ausführliche Hilfe. Es gibt einige Tutorials und darüber hinaus zwei Bücher, die angefangen von grundlegendem Basiswissen auch tiefgehende Themen erläutern. Den Funktionsumfang, den der Prototyp benötigt, kann Papervision3D zum größten Teil direkt erfüllen. Schatten werden nicht unterstützt, aber diese Funktionalität beherrscht auch keine der anderen Engines. Die Schatten, die Away3D bietet, reichen für die Anforderung an den Prototyp aufgrund der Nichtberücksichtigung der Lichtquelle nicht aus.

Somit müssen für den Prototyp planare Schatten selbst implementiert werden. Damit es bei der Darstellung der Schatten zu keinem Z-Fighting mit der Scheibe kommt, können bei Papervision3D „Layer“ verwendet werden, deren Verwendung im Buch „Papervision3D Essentials“ sehr gut beschrieben ist. Billboarding ist bei Papervision3D nur über Partikelsysteme möglich, welche für die Textdarstellung verwendet werden müssen. Es gibt ebenso mehrere Arten von Renderern, die für den Prototyp in Frage kommen können.

6.4.2. Away3D und Sandy 3D

Hier wurde aufgrund der etwas geringer vorhandenen Hilfe gegen die Umsetzung mit einer dieser Engines entschieden, obwohl sie die Anforderungen auch erfüllen. Es war im Prinzip eine subjektive Entscheidung, die den Ausschlag für Papervision3D gegeben hat.

6.4.3. Alternativa3D

Alternativa3D verwendet zur Tiefensortierung zwar einen BSP-Baum, was in der Darstellung sicherlich Vorteile hat. Hauptgründe für die Entscheidung gegen Alternativa3D sind jedoch die fehlenden Lichtquellen, die für die Darstellung von Schatten benötigt werden. Gleichzeitig ist diese Engine für einen eventuellen kommerziellen Einsatz nicht kostenlos. Die kaum vorhandene Hilfe und die wenigen Tutorials erschweren die Umsetzung ebenfalls.

6.5. Prototyp

Nachstehend werden Aufbau und Ablauf des Prototyps beschrieben. Dazu werden zur besseren Veranschaulichung immer auch beispielhaft wichtige Teile des Programmcodes und der zugehörigen XML-Datei präsentiert. Der Prototyp sieht wie folgt aus:

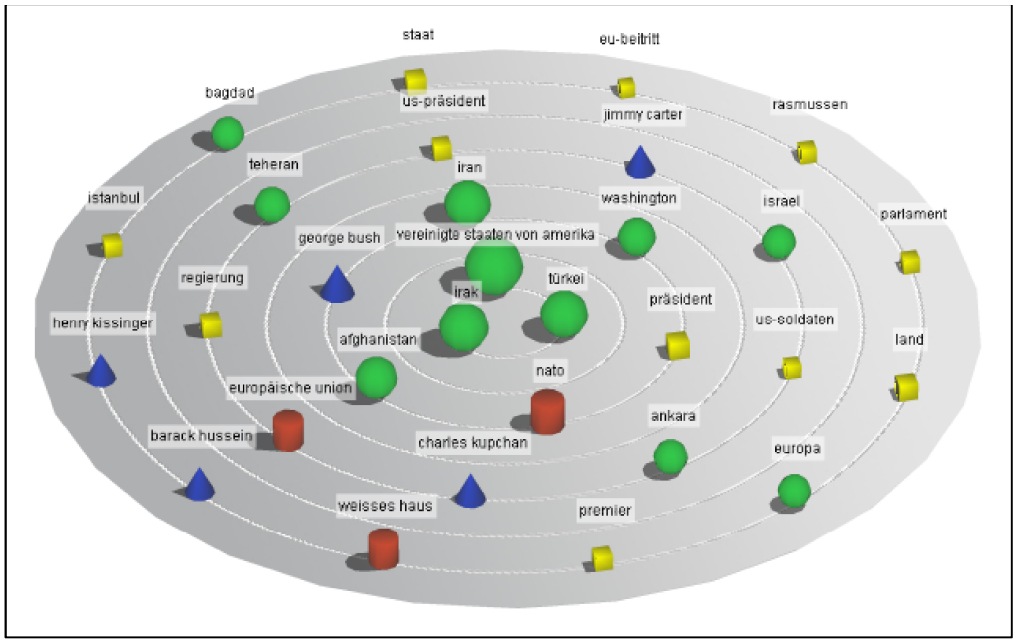


Abbildung 6: Artikelmenge zum Thema „Irak“

Zur Darstellung wird eine Artikelmenge zum Thema „Irak“ verwendet. Die zugehörige XML-Datei befindet sich in Anhang A.

6.5.1. Ablauf

Über den Parameter „appearscene“ in der HTML-Datei, welche die Flash-Datei einbettet, wird der Pfad zur XML-Datei, die das Appear-Format enthält, an Flash übergeben.

```
'FlashVars', 'appearscene=./xml_files/platform.xml'
```

Nach dem Laden der XML-Datei wird diese geparst und für die einzelnen darin enthaltenen Tags, werden Instanzen der entsprechenden Klassen gebildet. Für jeden XML-Tag existiert eine eigene Klasse mit demselben Namen, die gleichzeitig auch das Parsen der Unterelemente ihres XML-Knotens übernimmt. Beim Parsen eines Knotens wird eine Instanz der entsprechenden Klasse erzeugt und in dieser ein Verweis zum Elternobjekt sowie Verweise zu den Kindobjekten gespeichert. Wenn ein Objekt alle Unterobjekte fertig geparst hat, wird dies dem Elternobjekt mitgeteilt, wodurch festgestellt werden kann, wann das gesamte Parsen abgeschlossen ist. Da das Appear-Format Szenengraphen-basierend ist, werden die einzelnen Objekte in einer Hierarchie angelegt.

Die Klassen, die aufgrund von im Appear-Format enthaltenen Tags definiert werden, sind von Basisklassen abgeleitet, welche die gemeinsamen Eigenschaften und Funktionen dieser Klassen definieren. Nach dem Parsen werden die Basiselemente, die für Papervision3D nötig sind, erzeugt. Für jeden Node-Tag im Appear-Format wird ein „DisplayObject3D“ erstellt, welches weitere Nodes oder auch direkt Visuals enthalten kann. Die Transformationen werden immer relativ auf den übergeordneten Node bezogen. Nachdem die gesamte Szene geparst wurde, wird ein Timer gestartet, der für zeitgesteuerte Aktionen, zB die automatische Drehung der Wissensscheibe, verwendet wird.

6.5.2. Basis

Um eine Anwendung mit Papervision3D zu erstellen, werden unabhängig, von dem, den Inhalt beschreibenden XML-File bestimmte Objekte benötigt. Dazu gehören (vgl. [TOWI09], S. 38ff):

- **Scene**
  Zur Szene werden alle erzeugten Objekte, wie zB die Primitive, hinzugefügt.

- **Camera**
  Durch die Kamera wird die Szene von einem bestimmten Punkt aus betrachtet.

- **Viewport**
  Der Viewport ist ein Sprite und stellt das dar, was durch die Kamera betrachtet wird.

- **Render Engine**
  Dem Renderer werden die vorhergenannten Objekte übergeben. Es können dabei, wie bereits beschrieben, unterschiedliche Arten verwendet werden.

Code

```
this.viewport = new Viewport3D (800, 600, false, true);
this.stageMC.addChild (this.viewport);
this.renderer = new BasicRenderEngine ();
this.scene = new Scene3D ();
this.camera = new Camera3D ();
this.stageMC.addEventListener (Event.ENTER_FRAME, onRenderFrame);


this.renderer.renderScene (this.scene, this.camera, this.viewport);
```

Das Objekt „stageMC“ ist dabei der zugrundeliegende MovieClip, dem der Viewport hinzugefügt wird. Bei jedem Aufruf eines neuen Bildes dieses MovieClips wird die Szene neu gerendert. Die Bildrate wurde in der Autorenumgebung auf 50 Bilder pro Sekunde eingestellt. Die Größe des Viewports beträgt 800 Pixel in der Breite und 600 Pixel in der Höhe. Bei der Erzeugung des Viewports muss die Eigenschaft „interactive“ auf „true“ gesetzt werden, damit Interaktionen mit den Objekten möglich sind. Dies geschieht im oberen Codesegment bereits im Konstruktor.

Für den Prototyp wurde die „BasicRenderEngine“ ausgewählt, da sie die höchste Performance bietet. Allein sich überschneidende Objekte können damit nicht korrekt dargestellt werden, daher ist bei der Platzierung der Objekte inklusive ihrer Schatten, zu beachten dass sie sich nicht berühren. Ein Versuch zeigte, dass die „QuadrantRenderEngine“, welche den „QuadTree“-Algorithmus verwendet, ebenfalls nicht zu einer korrekten Darstellung führt, da das Z-Fighting zwischen den Schatten und der Wissensscheibe nicht behoben wird, wie Abbildung 7 zeigt. Bei sich überschneidenden Objekten tritt, wie Abbildung 8 andeutet, das Z-Fighting zwar nicht auf, jedoch wird das Material nicht mehr korrekt dargestellt.

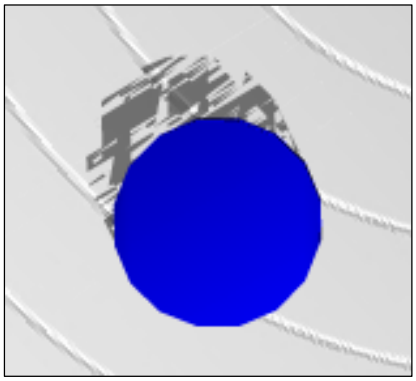


Abbildung 7: Z-Fighting - Schatten auf Wissensscheibe

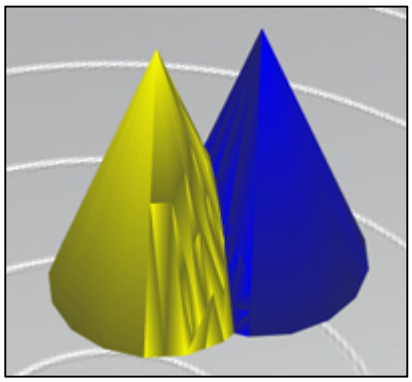


Abbildung 8: Sich überschneidende Kegel

6.5.3. Primitive

XML

```
<shape type="cone" radius="1" height="3"></shape>
```

Es werden vier Arten von Primitiven unterstützt. Diese sind Würfel, Kugel, Zylinder und Kegel. Nachstehend ist beispielhaft die Erstellung eines Kegels beschrieben. Die Erzeugung der anderen Primitive erfolgt auf ähnliche Weise. Für die Erstellung eines Kegels wird eine neue Instanz der Klasse „Cone“ (this.dispObject) erzeugt und diese an die Position (X:0, Z:0) gesetzt, wobei das Primitiv in der Höhe automatisch so platziert wird, dass die Unterseite auf (Y:0) steht. Gleichzeitig wird dem Primitiv das entsprechende Material zugeordnet und das Objekt dem Elternobjekt hinzugefügt.

Code

```
this.dispObject = new Cone (null, this.shape.radius, this.shape.height3D,
this.shape.edges, 1);
this.dispObject.x = 0;
this.dispObject.y = this.shape.height3D / 2;
this.dispObject.z = 0;
this.dispObject.material = this.material.material3D;
this.tfNode.addChild (this.dispObject);
```

6.5.4. Kamera

XML

```
<camera name="cam" width="800" height="600" near="0.1" far="100"
field="25">
    <source x="0" y="20" z="-30"/>
    <direction x="0" y="0" z="0"/>
</camera>
```

Nach dem Erstellen der Kamera werden ihr Position und Ziel zugeordnet. Die Near- und Far-Plane werden bei der Kamera definiert, indem ihr Abstand angegeben wird. Das View of Field wird durch die Eigenschaft „fov“ bestimmt. Die Kamera wird dann, der bereits bei den Basiselementen definierten Kamera, zugewiesen.

Code

```
this.camera = new Camera3D ();
this.camera.x = this.source.x;
this.camera.y = this.source.y;
this.camera.z = this.source.z;
this.camera.near = this.near;
this.camera.far = this.far;
this.camera.fov = this.field;
var directionpoint : DisplayObject3D = new DisplayObject3D ();
directionpoint.x = this.direction.x;
directionpoint.y = this.direction.y;
directionpoint.z = this.direction.z;
this.tfNode.addChild (directionpoint);
this.camera.target = directionpoint;
this.getSceneRoot ().camera = this.camera;
```

6.5.5. Licht

XML

```
<light type="point">
    <source x="10" y="20" z="-10"/>
</light>
```

Da es in Papervision3D nur eine Lichtart, nämlich Punktlicht gibt, wird bei jeder Art von Licht, die im Appear-Format vorkommt, immer ein Punktlicht an der angegebenen Position erstellt.

Code

```
this.light = new PointLight3D ();
this.light.x = this.source.x;
this.light.y = this.source.y;
this.light.z = this.source.z;
this.getSceneRoot ().light = this.light;
```

6.5.6. Events

XML

```
<event name="e1" type="mouseclick" action="calljs" param="nodename"/>
```

Im Appear-Format können Aktionen definiert werden, die durch Mausereignisse ausgeführt werden. Eine Aktion kann bei Mausklick, wenn die Maus über das Objekt bewegt wird und wenn die Maus das Objekt wieder verlässt, ausgelöst werden. Damit ein Primitiv auf ein Mausereignis reagieren kann, muss ein Event bei dem entsprechenden Primitiv registriert werden.

```
this.dispObject.addEventListener (InteractiveScene3DEvent.OBJECT_RELEASE,
mouseUp);
this.dispObject.addEventListener (InteractiveScene3DEvent.OBJECT_OVER,
mouseOver);
```

Bei Auftreten des Events werden die Parameter an die HTML-Seite, welche die Flash-Datei einbindet, weitergeleitet. In diesem Fall ist das der Knotenname, zu dem das Objekt gehört, auf das geklickt wird. Dazu wird ein „ExternalInterface“ benötigt, welches es erlaubt eine JavaScript-Funktion, die sich in der HTML-Seite befindet, auszuführen.

```
ExternalInterface.call ("CallMethod", method, param);
```

Derzeit ist nur die Aktion „Öffne einen Link in einem neuen Fenster“ definiert.

Ohne sie in der XML-Datei definieren zu müssen sind die Aktionen

- Drehen (Mausklick),
- Kippen (Mausklick) und
- Vergrößern/Verkleinern (Mausrad)

für die gesamte Szene integriert. Dabei werden nicht alle vorhandenen Objekte, sondern die Kamera bewegt.

6.5.7. Material

Es können drei Arten von Materialien verwendet werden:

ColorMaterial

XML

```
<material type="color">
    <color r="1.0" g="0.0" b="0.0"/>
</material>
```

Es dient zum Darstellen von Farben, ohne dass dabei auf die Lichtposition Rücksicht genommen wird.

Code

```
this.material3D = new ColorMaterial (Number ("0x" +
this.color3D.hexValue));
this.material3D.interactive = true;
```

Bei der Erstellung eines Farbmaterials wird nur die Farbe angegeben. Es ist wichtig, dass beim Material die Eigenschaft „interactive“ auf „true“ gesetzt wird, damit Interaktionen mit dem Objekt, welches das Material verwendet, funktionieren.

GouraudMaterial

XML

```
<material type="shaded">
    <ambient r="0.2" g="0.2" b="0.2"/>
    <color r="1.0" g="0.0" b="0.0"/>
</material>
```

Der Gouraud-Shading-Algorithmus wird dem von Phong vorgezogen, da durch Ausprobieren herausgefunden wurde, dass die Darstellungsqualität besser ist. Beim Phong Shader wurde festgestellt, dass harte Kanten zwischen der Umgebungsfarbe und der Farbe des Objekts selbst auftreten. Bei beiden Abbildungen wurde der Parameter „Ambient“ auf Schwarz und der Parameter „Specular“ auf 10 gesetzt.

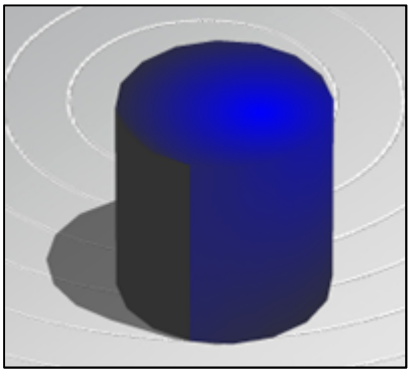


Abbildung 9: Papervision3D - Phong Shader

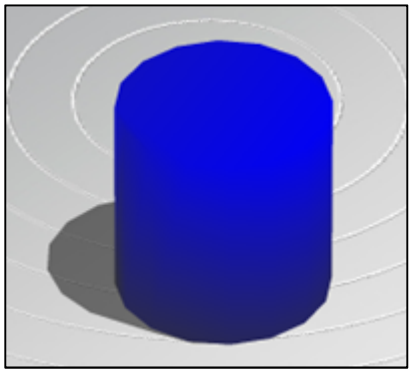


Abbildung 10: Papervision3D - Gouraud Shader

Code

```
this.material3D = new GouraudMaterial (this.getSceneRoot ().light, Number
("0x" + this.color3D.hexValue) , Number ("0x" +
this.ambientColor3D.hexValue) , 10);
this.material3D.interactive = true;
```

Ebenso werden nur die Eigenschaften „Ambient“ und „Specular“ berücksichtigt, wobei bei „Specular“ die Intensität zwischen 0 und 255 eingestellt werden kann. Die Definition von „Specular“ bedeutet laut dem Buch „Papervision3D Essential“ bei Flat Shading und im Weiteren anzunehmen auch bei den anderen Shading-Varianten, dass mit steigendem Wert immer mehr Licht absorbiert wird (vgl. [TOWI09], S. 219). Beim Prototyp wurde der Wert auf 10 gesetzt.

BitmapFileMaterial

Mit dem BitmapFileMaterial können Texturen geladen und dargestellt werden. Dabei werden wie in der Hilfe von Adobe Flash CS3 zu lesen ist, nur die Bildformate JPEG (progressiv, nicht progressiv), PNG (transparent, nicht transparent) und GIF (transparent, nicht transparent, erstes Bild eines animierten GIFs) unterstützt (vgl. [FLHELP]).

XML

```
<image name="ibase" source="material/base.jpg"/>
<material name="mbase" type="image" image="ibase"/>
```

Die obenstehende Definition im XML-File befindet sich innerhalb eines Media Stores. Die Angabe der URL, wo sich das Bild befindet, erfolgt im Tag „image“ und wird dann beim Material durch den Namen referenziert.

Code

```
this.material3D = new BitmapFileMaterial (this.image, true);
this.material3D.interactive = true;
```

Die Erstellung erfolgt durch die Angabe der URL.

6.5.8. Schatten

XML

```
<decor type="shadow" source="nitem" caster="sun"/>
```

Schatten können auf die Primitive Kugel, Würfel, Zylinder und Kegel angewandt werden, wobei in der XML-Datei der Name eines Knotens als „source“ angegeben wird und alle Kindobjekte des Knotens einen Schatten erhalten, sofern sie zu den Primitiven gehören, für die Schatten erzeugt werden können. Zur Darstellung von Schatten wird das Primitiv dupliziert, transformiert und mit einem halbtransparenten schwarzen ColorMaterial versehen.

Um die Schatten zu erzeugen, wird die Transformationsmatrix des Primitivs mit der Projektionsmatrix für planare Schatten multipliziert. Dabei wurde festgestellt, dass homogene Koordinaten in Papervision3D nicht richtig funktionieren. Somit wird die Transformationsmatrix noch mit der nachfolgenden Matrix multipliziert, um den Schatten zu skalieren und die richtigen Größenverhältnisse wieder herzustellen.

$$\begin{pmatrix} 1/ly & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/ly & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Code

Nachdem die Szene fertig geparst wurde, kann für die Objekte ein Schatten erstellt werden. Dazu wird zuerst ein ColorMaterial mit der Farbe Schwarz und einer Transparenz von 50 % erzeugt.

```
var shadowmaterial : ColorMaterial = new ColorMaterial (0x000000, 0.5);
```

Als nächstes wird das Primitiv, welches den Schatten erhält, dupliziert und diesem Objekt wird das Material für den Schatten zugewiesen.

```
this.shadowObject = this.dispObject.clone ();
this.shadowObject.material = shadowmaterial;
```

Für die Schattenmatrix wird die Position des Lichts benötigt und wie folgt ermittelt. Es werden jeweils die Positionen der Knoten, welche die schattenwerfenden Primitive enthalten, von der Position des Lichts subtrahiert, da sonst die Lichtposition als lokal für jedes Primitiv betrachtet wird. Das heißt, dass in diesem Fall, wenn das Licht die Position (X: 0, Y: 50, Z:0) hat, kein Objekt einen Schatten wirft, da die Lichtquelle für die Berechnung des Schattens für jedes Objekt direkt über diesem gesehen wird.

```
var l : Number3D = new Number3D (this.getSceneRoot ().light.x -
this.tfNode.sceneX, this.getSceneRoot ().light.y - this.tfNode.sceneY,
this.getSceneRoot ().light.z - this.tfNode.sceneZ);
```

Um die Projektionsmatrix zu erhalten, wird zuerst ein Array erstellt, welches danach in eine Matrix umgewandelt wird.

```
var shadowarray : Array = new Array (
l.y, - l.x, 0, 0,
0, 0, 0, 0,
0, - l.z, l.y, 0,
0, - 1, 0, l.y);
var shadowmatrix : Matrix3D = new Matrix3D (shadowarray);
```

Da homogene Koordinaten nicht korrekt verwendet werden können und durch diese eigentlich die Größe des Schattens wieder auf Objektgröße reduziert werden soll, wird die Größe des Schattens durch eine weitere Matrix zurückgesetzt. Die Definition dieser Matrix beschreiben die nächsten Codezeilen.

```
var correctionarray : Array = new Array (
1 / l.y, 0, 0, 0,
0, 0, 0, 0,
0, 0, 1 / l.y, 0,
0, 0, 0, 1);
var correctionmatrix : Matrix3D = new Matrix3D (correctionarray);
```

Die Transformationsmatrix des Schattenobjekts wird nun mit den oben definierten Matrizen multipliziert.

```
var objectmatrix : Matrix3D = this.shadowObject.transform;
objectmatrix.calculateMultiply (shadowmatrix, objectmatrix);
objectmatrix.calculateMultiply (correctionmatrix, objectmatrix);
this.shadowObject.copyTransform (objectmatrix);
```

Da nun der Schatten jedoch noch falsch, nämlich an den Szenenkoordinaten (X:0, Y:0, Z:0) positioniert wird, muss mit den folgenden Zeilen (exemplarisch für eine Kugel) der Schatten erst zum Primitiv versetzt werden.

```
this.shadowObject.x = this.tfNode.sceneX +
(this.shadowObject.transform.n12 * (this.shape.radius));
this.shadowObject.z = this.tfNode.sceneZ +
(this.shadowObject.transform.n32 * (this.shape.radius));
```

Gleichzeitig wird der Schatten beim Objekt selbst mit seinem Mittelpunkt an den Mittelpunkt des Objekts gesetzt (siehe Abbildung 11). Dies muss ebenfalls korrigiert werden.

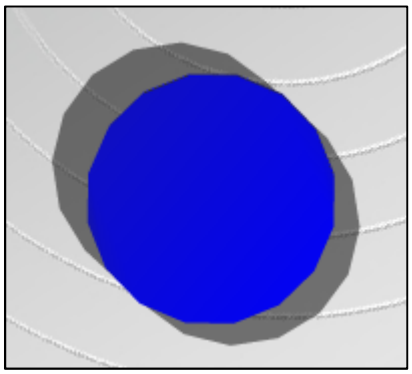


Abbildung 11: Falsche Schatten-position

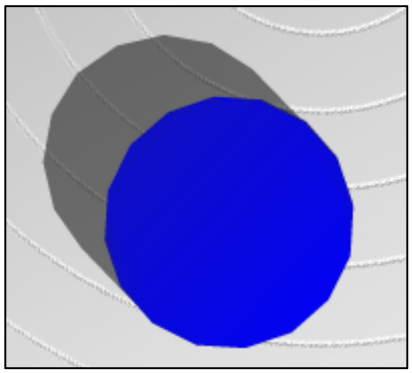


Abbildung 12: Korrekte Schatten-position

Nach der Positionierung des Schattens wird dieser direkt der Szene hinzugefügt.

```
this.getSceneRoot ().scene.addChild (this.shadowObject);
```

6.5.9. Viewport Layer

Bei der Darstellung von Schatten kommt es bei der Verwendung der BasicRenderEngine zu Z-Fighting Problemen mit der Wissensscheibe, da sie direkt auf der Oberfläche der Wissensscheibe positioniert werden. Um die erzeugten Schattenobjekte in der Tiefe richtig positionieren zu können, werden Viewport Layer verwendet. Sie stellen eigene Container dar, bei denen es zu keinen Tiefenüberschneidungen zwischen den einzelnen Containern kommt. Es gibt zwei Möglichkeiten Viewport Layer zu verwenden (vgl. [TOWI09], S. 278ff).

Zum einen kann die Eigenschaft „useOwnContainer“, die bei jedem DisplayObject3D in Papervision3D verfügbar ist, auf „true“ gesetzt werden, wodurch automatisch ein Layer erzeugt wird. Diese Option benötigt aber sehr viele CPU-Ressourcen und kann je nach Blickwinkel noch immer zu Problemen bei der Tiefensortierung führen. Zum anderen können auch selbst Viewport Layer über die Klasse „ViewportLayer“ erstellt werden. Diese Technik wird für den Prototyp verwendet.

Für jedes „Visual“ wird ein eigener Viewport Layer erzeugt, der dem „Viewport“ hinzugefügt wird. Innerhalb dieses Viewport Layers werden die Primitive, die ein Visual darstellen kann, wiederum in einzelne Layer gegliedert. Dazu gibt es für die Viewport Layer unterschiedliche Sortierungsmodi (vgl. [TOWI09], S. 282ff):

- **Z-Sort**
  Dabei werden die Layer anhand der gemittelten Z-Werte aller Vertices in dem Layer sortiert.

- **Origin-Sort**
  Hier werden sie aufgrund des Registrierungspunktes, der die Position des Objektes im Raum bestimmt, geordnet.

- **Index-Sort**
  Bei dieser Art von Sortierung kann die Reihenfolge selbst mit Hilfe einer Indexangabe bestimmt werden.

Die Verwendung von Viewport Layern ist notwendig, damit der Schatten sich immer unter bzw. hinter dem Primitiv und die Objektbeschriftung sich immer vor bzw. über dem Primitiv befinden. Die Layer innerhalb des Viewport Layers für das Visual werden mit Hilfe eines Index sortiert. Der Schatten erhält den Index „1“, das Objekt selbst den Index „2“ und der Text den Index „3“. Die Viewport Layer der Visuals werden untereinander mit Hilfe des „Z-Sort“-Modus automatisch sortiert.

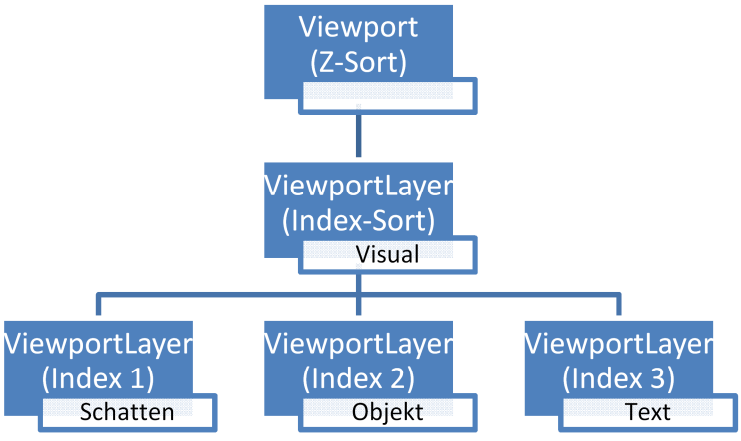


Abbildung 13: Anordnung der Viewport Layer

Code

Es wird ein globaler Viewport erstellt, auf den die Viewports der Visuals platziert werden und dieser wird dem „stageMC“ hinzugefügt.

```
this.viewport.containerSprite.sortMode = ViewportLayerSortMode.Z_SORT;
this.stageMC.addChild (this.viewport);
```

Bei der Erstellung der Visuals wird der Viewport Layer für die Primitive wie folgt erstellt.

```
this.viewpLayer = new ViewportLayer (this.getSceneRoot ().viewport,
null);
this.getSceneRoot ().viewport.containerSprite.addLayer (this.viewpLayer);
this.viewpLayerObject = new ViewportLayer (this.getSceneRoot ().viewport,
null);
this.viewpLayer.addLayer (viewpLayerObject);
this.viewpLayer.sortMode = ViewportLayerSortMode.INDEX_SORT;
this.viewpLayerObject.addDisplayObject3D (this.dispObject, true);
this.viewpLayerObject.layerIndex = 2;
if (this.backgroundLayer)
{
    this.viewpLayer.forceDepth = true;
    this.viewpLayer.screenDepth = 1000;
}
```

Das Erzeugen der Viewport Layer für den Text wie auch für die Schatten erfolgt mit dem entsprechend angepassten Index ähnlich. Als Parameter bei der Erstellung eines Viewport Layers wird dabei immer der Viewport mitgegeben und erst dann wird der neue Viewport Layer dem übergeordneten Viewport Layer zugeordnet. Wenn im XML-File das Attribut „background“ des Visuals auf „true“ gesetzt wurde, wird für diesen Viewport Layer eine bestimmte Tiefe erzwungen, sodass er immer im Hintergrund bleibt.

6.5.10. Text

XML

```
<visual>
    <shape type="text" font="ffont" text="Text"/>
    <material type="color">
          <color r="1.0" g="0.0" b="0.0"/>
    </material>
</visual>
```

Bei Papervision3D gibt es die Möglichkeit 3D-Text zu erstellen, jedoch nicht in der Form, dass der Text automatisch immer in Richtung Kamera zeigt, sodass er gelesen werden kann. Es kann zwar zB über die Funktion „lookAt()“ für den Text angegeben werden, dass er die Kamera fixiert, aber eigene Versuche haben gezeigt, dass dies nicht immer richtig funktioniert. So werden zur Darstellung von Text die vorhandenen Partikelsysteme verwendet. Sie bieten von vornherein Billboarding, haben aber den Nachteil, dass nur Bilder dargestellt werden können.

Der erzeugte Text muss somit vorher in Echtzeit zu einem Bild gerendert werden, um dann als Partikel angezeigt zu werden. Damit der Text trotzdem lesbar bleibt, ist es notwendig die Schriftgröße und die Größe des Partikels aufeinander abzustimmen. Um den Text zentriert über dem Objekt zu positionieren, wird ein Offset bei der Materialposition für den Partikel angegeben. Der Hintergrund des Textfeldes wird leicht transparent angezeigt.

Es werden zur Darstellung Geräteschriftarten verwendet. Die ausgewählte Schriftart sollte am Computer des Benutzers vorhanden sein. Ist sie nicht vorhanden, wird laut der Hilfe von Adobe Flash CS3 eine ähnliche ausgewählt (vgl. [FLHELP]). Eine weitere Möglichkeit wäre es, Schriftarten einzubetten, wodurch die genaue Darstellung der Schrift garantiert werden kann. Hier würde auch Anti-Aliasing für die Schrift verwendet werden. Da aber nicht genau gesagt werden kann, welche Schriftart beim Appear-Format angegeben wird, wird auf das Einbetten der Schriftart verzichtet. Außerdem würde mit jeder eingebetteten Schriftart die kompilierte SWF-Datei in der Größe anwachsen.

Code

```
var txt : TextField = new TextField ();
txt.autoSize = TextFieldAutoSize.LEFT;
txt.selectable = false;
var format : TextFormat = new TextFormat ();
format.font = this.shape.fontName;
format.color = this.material.material3D.fillColor;
format.size = 12;
format.align = TextFormatAlign.CENTER;
txt.defaultTextFormat = format;
txt.text = this.shape.text;

var particles : Particles = new Particles ();
this.tfNode.addChild (particles);

var bitmapData : BitmapData = new BitmapData (txt.textWidth + 5,
txt.textHeight + 5, true, 0x88FFFFFF);
bitmapData.draw (txt);

var xoffset : Number = (txt.textWidth / 2) * ( - 1);
var yoffset : Number = (txt.textHeight / 2) * ( - 1);
var material : BitmapParticleMaterial = new BitmapParticleMaterial (bit-
mapData, 1, xoffset, yoffset);
material.smooth = true;

var particle : Particle = new Particle (material, 0.05, 0, 0, 0);
particles.addParticle (particle);
this.viewpLayerText = new ViewportLayer (this.getSceneRoot ().viewport,
null);
this.getSceneRoot ().viewport.containerSprite.addLayer
(this.viewpLayerText);
this.viewpLayerText.addDisplayObject3D (particles, true);
this.viewpLayerText.layerIndex = 3;
```

6.5.11. Animation

XML

```
<node type="scene" name="nbase" active="true" autoRotation="true">
```

Die automatische Drehung der Wissensscheibe erfolgt, indem beim Tag „node“, welches die Wissensscheibe definiert, das Attribut „autoRotation“ auf „true“ gesetzt wird. Dadurch wird die Scheibe pro Timeraufruf alle 100 Millisekunden um ein Grad im Uhrzeigersinn um die Y-Achse rotiert.

Code

```
this.tfNode.rotationY = this.tfNode.rotationY + 1;
```

6.5.12. Media Stores

XML

```
<mediastore>
    <image name="base" source="material/base.jpg"/>
    <material name="basemat" type="image" image="base"/>
    <font name="ffont" face="arial" size="12"/>
</mediastore>
```

Wenn ein Media Store vorkommt, wird dieser in seinem Elternobjekt gespeichert. Benötigt ein Element einen Inhalt vom Media Store, wird die Suche beim Elternobjekt des Elements begonnen. Sollte das gesuchte Objekt dort vorkommen, wird die Suche beendet, sonst wird bis zum obersten Element weitergesucht. Die Suche wird immer nur nach oben fortgesetzt, womit gewährleistet wird, dass Medien in lokalen Media Stores Medien in globalen Media Stores überschreiben.

6.5.13. Parameter Stores

XML

```
<paramstore>
    <param type="numeric" name="cam_dir_x" value="0"/>
    <param type="numeric" name="cam_dir_y" value="0"/>
    <param type="numeric" name="cam_dir_z" value="0"/>
</paramstore>
```

```
<direction x="=cam_dir_x" y="=cam_dir_y" z="=cam_dir_z"/>
```

Für die Attributwerte der einzelnen Knoten können Variablen verwendet werden. Die Definition dieser erfolgt mit einem „=“ vor dem Namen der Variablen.

Die Parameter Stores werden ebenso wie die Media Stores bei ihrem Elternobjekt gespeichert. Wird beim Parsen eine Variable gefunden, erfolgt die Suche nach dem Wert analog zum Suchen in Media Stores. Es wird immer nur nach oben gesucht und lokale Parameter überschreiben globale Parameter.

6.5.14. Geschwindigkeit

Die Darstellungsgeschwindigkeit sinkt bei Papervision3D auch bei der Verwendung der BasicRenderEngine mit der Anzahl der Objekte schnell. Nachfolgend ist die Wissensscheibe in Abbildung 14 mit 30 und in Abbildung 15 mit 13 Objekten dargestellt. Um die Anzahl der Bilder pro Sekunde zu ermitteln, wird das in Papervision3D inkludierte „StatsView“ verwendet (vgl. [TOWI09], S. 380f). In dieser Anzeige werden neben den Bildern pro Sekunde (FPS) auch die Anzahl der Dreiecke (Tri), die Anzahl der mit einem Shader versehenen Dreiecke (Sha) und die verwendeten Partikel (Par) angezeigt. Genauso wird die Menge der derzeit entfernten Objekte (COb) und Dreiecke (CTr) sowie Partikel (CPa) ermittelt.

Die Anzahl der Bilder sinkt bei der Darstellung von 30 Objekten auf 14 FPS, wobei bei 13 dargestellten Objekten die FPS noch bei 23 liegen. Zu berücksichtigen ist, dass Schatten jeweils eigene Objekte darstellen, da sie von den eigentlichen Primitiven dupliziert wurden. Die ungefähre Anzahl an Dreiecken, die der Flashplayer laut dem Buch „Papervision 3D Essentials“ noch handhaben kann, liegt bei ca. 3.000 (vgl. [TOWI09], S. 241). Bei 30 Objekten ist die Anzahl, wie in der untenstehenden Abbildung zu sehen ist, bereits bei ca. 2.500 Dreiecken, was den Schluss zulässt, dass dies bereits, wie auch an den FPS zu sehen ist, schon an der Grenze des Machbaren liegt. Wird statt der BasicRenderEngine die QuadrantRenderEngine mit der Option „ALL_FILTERS“ verwendet, sinken wie in Abbildung 16 zu sehen ist sogar schon bei einem darzustellenden Objekt die FPS auf 0 bis 1 und der Browser ist bereits nach kurzer Zeit nicht mehr bedienbar. Die Anzahl der Dreiecke steigt hier auf über 3.000, was jedoch hauptsächlich auf die Scheibe im Hintergrund zurückzuführen ist. Es ist anzunehmen, dass die oben genannten Werte bei unterschiedlichen Hardwarekonfigurationen variieren, aber sie zeigen zumindest eine Tendenz, wie sich die Anzahl der Bilder pro Sekunde bei unterschiedlichen Einstellungen und unterschiedlicher Objektanzahl verhält.

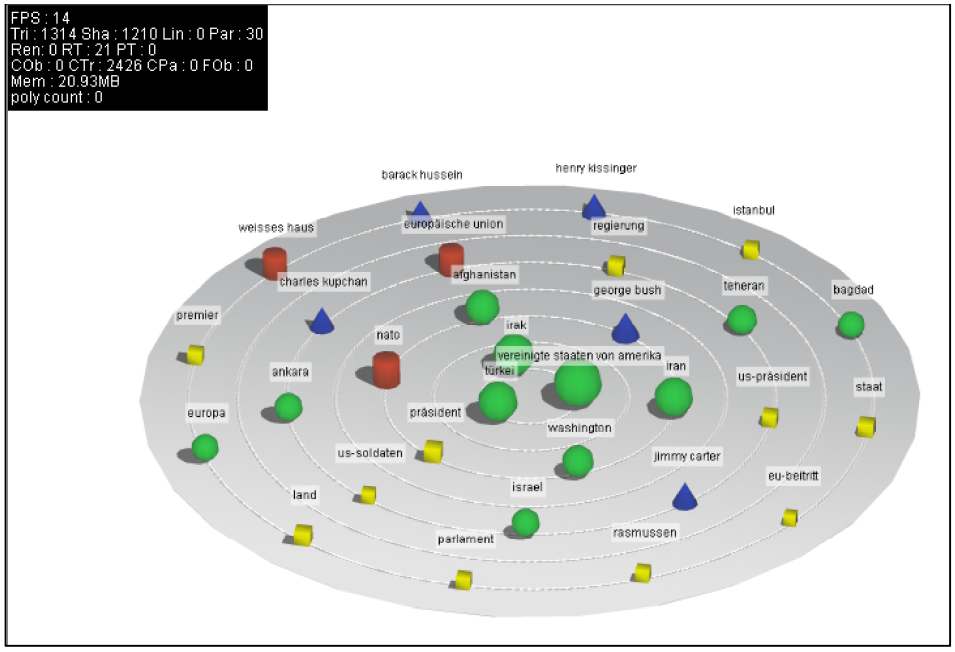


Abbildung 14: BasicRenderEngine mit 30 Objekten

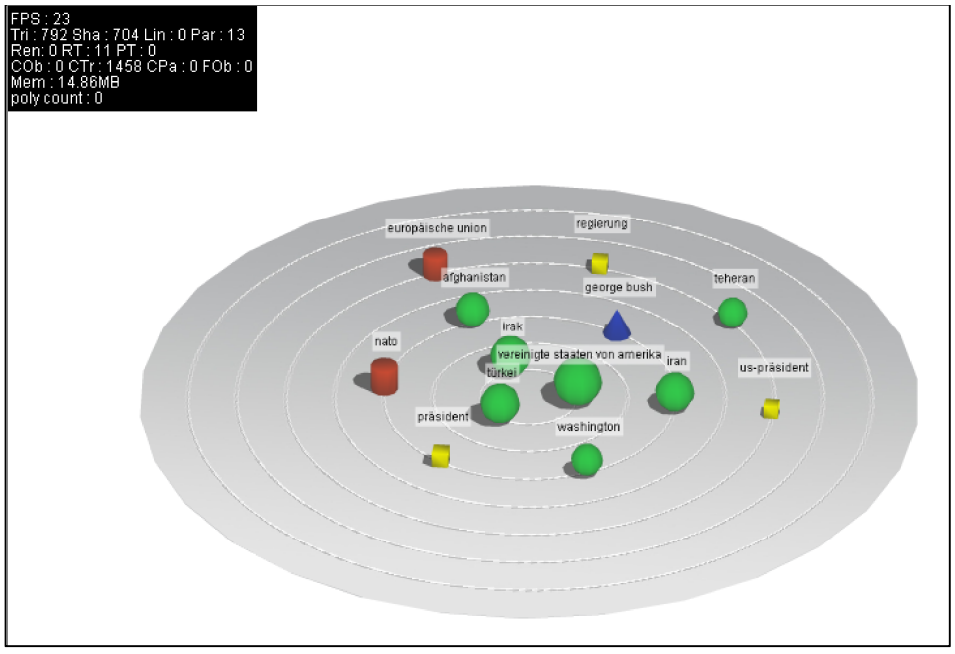


Abbildung 15: BasicRenderEngine mit 13 Objekten

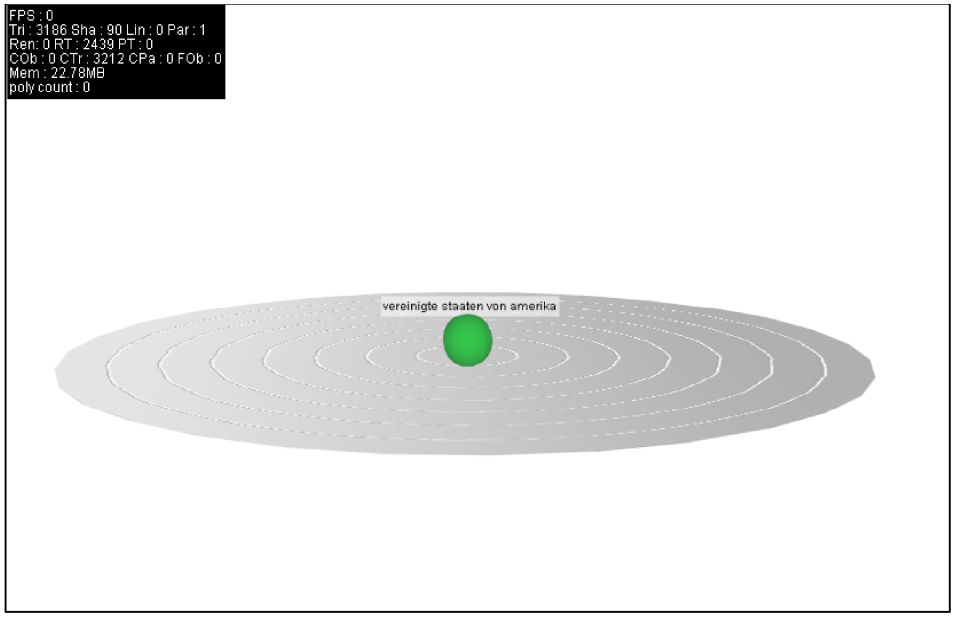


Abbildung 16: QuadrantRenderEngine mit einem Objekt

6.6. Fazit

Generell ist zu sagen, dass zur Erstellung von Informationsvisualisierungen Flash verwendet werden kann, wenn darauf geachtet wird, dass die Gesamtanzahl der Dreiecke in einer Szene eine bestimmte Menge nicht übersteigt. Die Probleme, die bei der Erstellung des Prototyps aufgetreten sind, beziehen sich hauptsächlich auf die kaum vorhandene Hardwareunterstützung des Flashplayers.

6.6.1. Geschwindigkeit

Ohne Hardwareunterstützung kann zur Tiefensortierung nicht auf die Grafikkarte und den „Z-Buffer“ zurückgegriffen werden und der Einsatz von softwarebasierten Algorithmen ist nötig. Dabei verlangsamen diese die Geschwindigkeit auf ein nicht mehr verwendbares Ausmaß, wie in Papervision3D bei der Verwendung der QuadrantRenderEngine gesehen werden kann.

6.6.2. Z-Fighting

Ein weiteres Problem ist das des Z-Fightings, welches bei sich überschneidenden Objekten auftritt. Wenn die BasicRenderEngine eingesetzt wird, die den Painter’s Algorithmus verwendet, ist dies deutlich sichtbar. Die Verwendung der QuadrantRenderEngine kann dies auch nicht zufriedenstellend lösen. Durch Verwendung von Viewport Layern, welche bei Papervision3D gut dokumentiert sind, kann dieses Problem jedoch umgangen werden, was vor allem bei Schatten von großer Bedeutung ist.

6.6.3. Sonstiges

Es ist bei der Umsetzung nicht einfach festzustellen, ob eine eventuell falsche Verwendung der angegebenen Methoden zu einem nicht korrekten Verhalten führt oder ob hier noch ein Fehler in der Engine vorliegt. Somit ist hier die genaue Beschreibung der Funktionalität durch einige Tutorials und Bücher von großem Vorteil. Das Finden eines Workarounds für einzelne, dem Anschein nach nicht funktionierende Methoden, war bei der Entwicklung des Prototyps immer möglich.

6.6.4. Vergleich mit anderen Flash Engines

Away3D

Hier konnten in einfachen Tests ähnliche Probleme wie bei Papervision3D beobachtet werden. Die Geschwindigkeit sinkt bei der Verwendung des Renderers, der den „QuadTree“-Algorithmus benutzt, vor allem bei Bewegungen ebenfalls sehr stark. Die Verwendung des „QuadTree“-Algorithmus ist jedoch notwendig, da es sonst auch hier zu Z-Fighting bei sich überlappenden Objekten kommt.

Abbildung 17 zeigt, dass bei sich zwei überschneidenden Kegeln, bei Verwendung des „QuadTree“-Renderers in der Option „INTERSECTING_OBJECTS“ vor allem bei Bewegungen die dargestellten Bilder pro Sekunde auf eine sehr niedrige Anzahl sinken, wobei jedoch das Z-Fighting nicht mehr sichtbar auftritt.

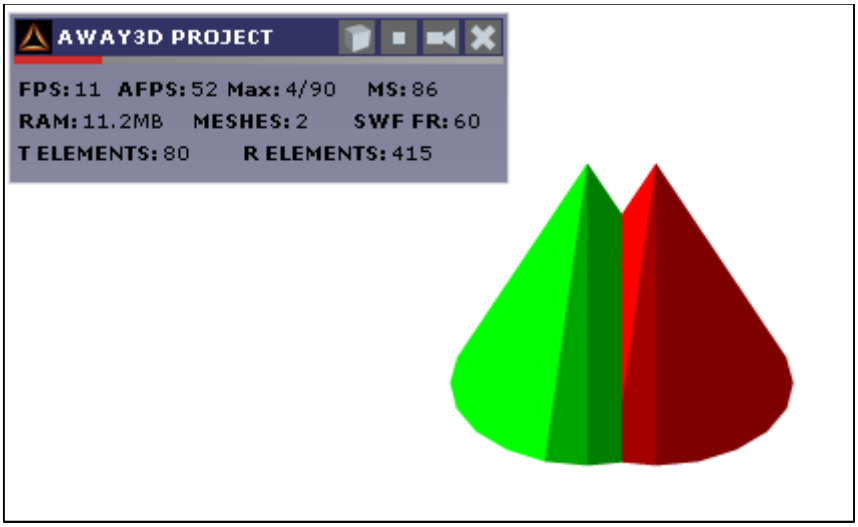


Abbildung 17: Away3D - QuadTree-Renderer

Die Abbildung 18 zeigt im Gegensatz zur oberen, dass bei der Verwendung des Renderers, der auf den Painter’s Algorithmus setzt, die Anzahl der Bilder pro Sekunde sehr hoch ist, aber Z-Fighting-Probleme auftreten.

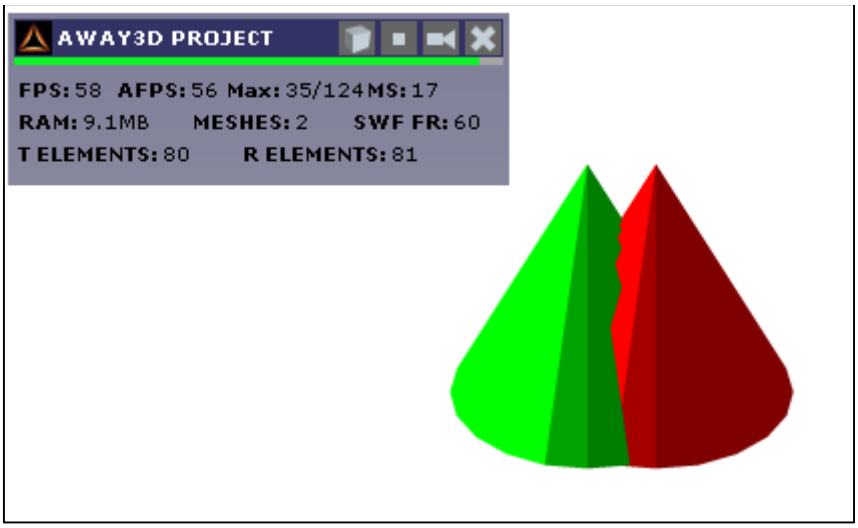


Abbildung 18: Away3D - BasicRenderer

Eventuell beeinflussen andere Einstellungen die Geschwindigkeit und das Z-Fighting, aber die Beispiele zeigen dennoch deutlich die auftretenden Probleme.

Sandy 3D

Da Sandy 3D ebenfalls auf den Painter's Algorithmus setzt, treten hier genauso „Z-Fighting"-Probleme auf. Der Einsatz der experimentell bereits vorhandenen BSP-Sortierung kann hier möglicherweise in Zukunft Verbesserungen bringen. Abbildung 19 zeigt die Sortierung auf Objektbasis, was Überschneidungen nicht zulässt.

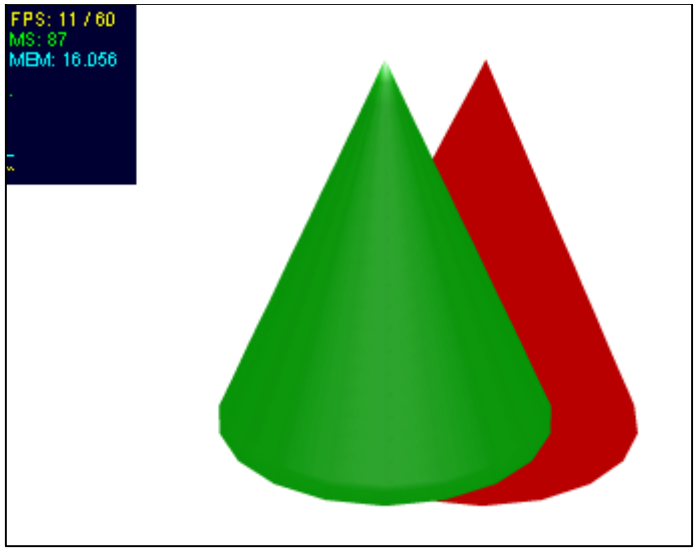


Abbildung 19: Sandy 3D - objektbasierte Sortierung

Die Abbildung 20 stellt die Verwendung der polygonbasierten Sortierung dar, wobei das Z-Fighting deutlich sichtbar wird.

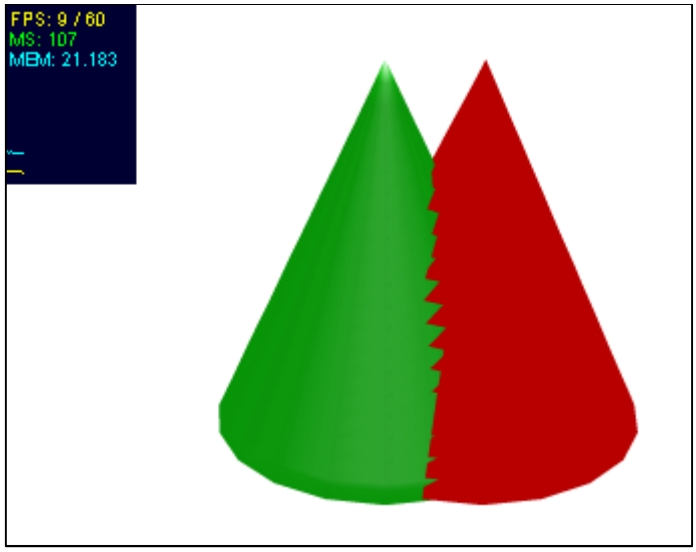


Abbildung 20: Sandy 3D - polygonbasierte Sortierung

Wie an den FPS gesehen werden kann, ist die Geschwindigkeit bei beiden Sortierungsarten nicht besonders hoch.

Alternativa3D

Durch die Verwendung einer BSP-Sortierung gibt es bei sich überschneidenden Objekten kaum Probleme. Hier hat sich bei Tests jedoch schnell gezeigt, dass die wenigen verfügbaren Tutorials die Erstellung eines Prototyps erschwert hätten. Außerdem sind hier durch die fehlenden Lichtquellen nicht alle Voraussetzungen für den Prototyp gegeben. Wie in der nachfolgenden Abbildung 21 gesehen werden kann, sind bei Alternativa3D auch sich überschneidende halbtransparente Kegel noch ohne Z-Fighting mit einer hohen Anzahl von Bildern pro Sekunde von ca. 60 (dies entspricht den in Flash eingestellten Frames pro Sekunde) darstellbar. Auch bei Bewegungen sinkt die Geschwindigkeit kaum.

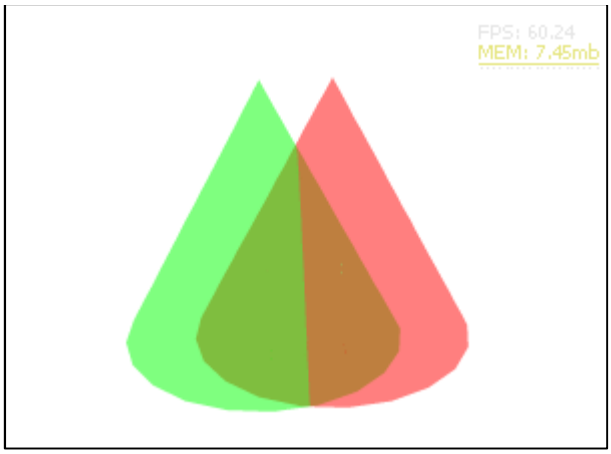


Abbildung 21: Alternativa3D – BSP-Sortierung

6.6.5. Vergleich zu anderen Formaten

Die Geschwindigkeitsprobleme und das Auftreten des Z-Fightings sind zumindest bei den Formaten, die Hardwareunterstützung bieten, nicht zu erwarten, da dort die Verwendung der GPU und eines „Z-Buffers“ möglich sind.

7. Zusammenfassung und Ausblick

In dieser Arbeit wurden aufbauend auf die 3D-Grundlagen die wichtigsten derzeit verfügbaren 3D-Formate im Web ermittelt und eines dieser Formate aufgrund von verschiedenen untersuchten Kriterien ausgewählt. Die Fokussierung dieser Arbeit auf Informationsvisualisierungen beeinflusst die Kriterien in der Art und Weise, dass keine realitätstreue Darstellung benötigt wird, sondern die möglichst gute Erreichbarkeit der Nutzergruppe im Vordergrund steht. Somit wurde aus den verfügbaren Formaten aufgrund der hohen Verbreitung des Flashplayers Flash ausgewählt. Der Nachteil, dass Flash derzeit nur eingeschränkte Hardwareunterstützung bietet, wurde in Kauf genommen, da Hardwareunterstützung für einfache Informationsvisualisierungen, wie die Wissensscheibe, eher vernachlässigbar sein sollte. Weil es für Flash wiederum unterschiedliche 3D-Engines, nämlich Away3D, Papervision3D, Sandy 3D und Alternativa3D gibt, wurden diese ebenfalls einer genaueren Untersuchung unterzogen. Dass die Auswahl auf Papervision3D gefallen ist, liegt vor allem am umfangreichen Hilfsangebot dieser Engine. Bei der Implementierung des Prototyps erwiesen sich die genau beschriebenen Funktionen als sehr nützlich. Es gibt bei den untersuchten Flash-Engines in Hinblick auf die benötigte Funktionalität für den Prototyp kaum Unterschiede. Die Umsetzung der Wissensscheibe von Brockhaus hat jedoch trotzdem die Grenzen der Flash-Engines aufgezeigt. Aufgrund der fehlenden Hardwareunterstützung müssen alle Algorithmen, wie zB die Tiefenberechnung, softwarebasiert durchgeführt werden. Dabei kommt es zu Darstellungsfehlern und die Geschwindigkeit bei komplexeren Algorithmen, wie dem „QuadTree“-Algorithmus, wird sehr langsam.

Die vielen Formate und unterschiedlichen Plugins erschweren eine Verbreitung von 3D-Inhalten im Web. Derzeit gibt es jedoch eine Bestrebung namens „WebGL“, die hardwarebasierte Unterstützung von 3D bereits im Browser zu verwirklichen, ohne dass ein Plugin installiert werden muss. Zumindest einige der heutigen Browseranbieter folgen dieser Richtung. Die Zukunft wird zeigen, ob durch die immer schneller werdende Hardware und die immer schneller werdenden Internetzugänge sowie standardmäßige 3D-Unterstützung, sich 3D im Web auch wirtschaftlich durchsetzen kann.

8. Literaturverzeichnis

[3DANYW01] Interview mit Gilad Khen bei 3D-Test, 2001, Online abrufbar: http://www.3d-test.com/interviews/3danywhere_1.htm (02.01.2010)

[3DENG06] Others 3D engine (Blog), 2006, Online abrufbar: http://www.flashsandy.org/blog/page/11 (02.02.2010)

[3DVIA07] Dassault Systèmes Unveils 3DVIA to Imagine, Play, and Experience Life in 3D Online, 2007, Online abrufbar: http://www.3ds.com/fileadmin/newsevents/finance/3DVIA_260607_EN.pdf (02.01.2010)

[3DZU08] 3D-Internet: In fünf Jahren Alltag, 2008, Online abrufbar: http://www.pressetext.de/news/080826016/3d-internet-in-fuenf-jahren-alltag (14.02.2010)

[A3D10] Alternativa3D - browser 3D-engine based on Adobe Flash, 2010, Online abrufbar: http://alternativaplatform.com/en/alternativa3d (03.01.2010)

[A3DABOUT10] Alternativa - About us, 2010, Online abrufbar: http://alternativaplatform.com/en/about (03.01.2010)

[A3DLIC09] Alternativa3D licensing details, 2009, Online abrufbar: http://alternativaplatform.com/en/alternativa3d/licenseInfo.html (04.01.2010)

[A3DSTART08] Alternativa3D engine is ready to use (Blog), 2008, Online abrufbar: http://www.flashsandy.org/blog/category/must-see (05.01.2010)

[ADOBE04] Adobe Atmosphere 1.0 End of Life, Frequently Asked Questions, 2004, Online abrufbar: http://www.adobe.com/products/atmosphere/pdfs/atmosphere_faq.pdf (30.12.2009)

[ADOBE05] Adobe History, 2005, Online abrufbar: http://www.adobe.com/aboutadobe/pressroom/pdfs/timeline.pdf (30.12.2009)

[AGOS05] M. K. Agoston: Computer graphics and geometric modeling: implementation and algorithms, 2005, Springer Verlag, London

[AKHA02] T. Akenine-Möller, E. Haines: Real-Time Rendering, Second Edition, 2002, A K Peters, Natick, Massachusetts

[AKHAHO08] T. Akenine-Möller, E. Haines, Naty Hoffmann: Real-Time Rendering, Third Edition, 2008, A K Peters, Wellesley, Massachusetts

[Anfy3D02] Anfy3D API support page, 2002, Online abrufbar: http://www.anfyteam.com/dev/a3d (02.01.2010)

[ASTC10] ActionScript Technology Center, 2010, Online abrufbar: http://www.adobe.com/devnet/actionscript (14.02.2010)

[AT3DAN09] Alternativa 3D Series – Tutorial 5 – Creating Interactivity, 2009, Online abrufbar: http://www.thetechlabs.com/tutorials/3d/alternativa3d-creating-interactivity (14.02.2010)

[AT3DBUMP09] How to use bump mapping on alternativa? (Forum), 2009, Online abrufbar: http://forum.alternativaplatform.com/posts/list/430.page#2334 (30.04.2010)

[AT3DFL09] Alternativa Support for FLINT Particles (Forum), 2009, Online abrufbar: http://flintparticles.org/forum/comments.php?DiscussionID=172&page=1#Item_0 (21.04.2010)

[AT3DFLV09] Alternativa3D: Can flv video be used as texture of object? (Forum), 2009, Online abrufbar: http://forum.alternativaplatform.com/posts/list/445.page#2441 (27.03.2010)

[AT3DFOR10] Alternativa3D: general questions (Forum), 2010, Online abrufbar http://forum.alternativaplatform.com/posts/list/2911.page (21.04.2010)

[AT3DFR08] Alternativa3D: Re:Next few quastions :) (Forum), 2008, Online abrufbar: http://forum.alternativaplatform.com/posts/list/71.page (14.02.2010)

[AT3DKO08] Alternativa3D: Re: After the Tutorials (Forum), 2008, Online abrufbar: http://forum.alternativaplatform.com/posts/list/1152.page#9962 (14.02.2010)

[AT3DLIGHT07] The Right shading – Kommentar von Mikhail Fominykh (Blog), 2007, Online abrufbar: http://blog.alternativaplatform.com/en/2007/07/26/pravilnyj-shejding (07.02.2010)

[AT3DLOD09] Alternativa3D 7.0: first impressions (Blog), 2009, Online abrufbar: http://makc3d.wordpress.com/2009/10/05/alternativa3d-7-first-impressions (06.02.2010)

[AT3DMM09] Mipmapping (Blog), 2009, Online abrufbar: http://blog.alternativaplatform.com/en/2009/12/25/mipmapping (07.02.2010)

[AW3DAN08] First steps in Away3D: Part 4 - Scene interaction (Blog), 2008, Online abrufbar: http://blog.tartiflop.com/2008/11/first-steps-in-away3d-part-4-scene-interaction (14.02.2010)

[AW3DBM09] Away3D Programming Tutorial - Bump Mapping, 2009, Online abrufbar: http://www.brighthub.com/hubfolio/matthew-casperson/articles/47627.aspx (07.02.2010)

[AW3DDL09] Away3D - Downloads, 2009, Online abrufbar: http://away3d.com/downloads (03.01.2010)

[AW3DDOF09] Away3D Programming Tutorials - Depth of Field, 2009, Online abrufbar: http://www.brighthub.com/hubfolio/matthew-casperson/articles/48046.aspx (07.02.2010)

[AW3DLIC10] AWAY3D - Realtime 3D engine for Flash, 2010, Online abrufbar: http://code.google.com/p/away3d (03.01.2010)

[AW3DLOD08] AWAY3D vs PAPERVISION (Forum), 2008, Online abrufbar: http://groups.google.com/group/away3d-dev/browse_thread/thread/d9c029574b9e16d0 (30.04.2010)

[AW3DPA08] Flint 2.0 released (Blog), 2008, Online abrufbar: http://www.richardlord.net/blog/flint-20-released (21.04.2010)

[AW3DTEAM10] Away3D – Team, 2010, Online abrufbar: http://away3d.com/team (03.01.2010)

[AWAY3D07] Papervision3D to merge Away3D features (Blog), 2007, Online abrufbar: http://blog.papervision3d.org/2007/05/16/papervision3d-to-merge-away3d-features (03.01.2010)

[AWAY3D10] AWAY3D - Realtime 3D engine for Flash, 2010, Online abrufbar: http://away3d.com (03.01.2010)

[AWAY3DF09] Away3D - Features, 2009, Online abrufbar: http://away3d.com/features (03.01.2010)

[AWAYZS10] Antwort auf: How do I improve Z sorting for Away3DLite?, 2010, Online abrufbar: http://www.mail-archive.com/away3d-dev@googlegroups.com/msg09609.html (12.01.2010)

[AWAYOCC07] Away3D flash engine, 2007, Online abrufbar: http://away.kiev.ua/away3d (15.04.2010)

[AW3DLIGHT08] First steps in Away3D: Part 5 - Lighting and shading (Blog), 2008, Online abrufbar: http://blog.tartiflop.com/2008/11/first-steps-in-away-3d-part-5-lighting-and-shading (15.04.2010)

[BALDER07] 3D in Silverlight 1.1 Alpha (Blog), 2007, Online abrufbar: http://www.dolittle.com/blogs/einar/archive/2007/05/19/3d-in-silverlight-1-1-alpha.aspx (02.01.2010)

[BALDER10] Balder - Project Description, 2010, Online abrufbar: http://balder.codeplex.com (30.04.2010)

[BPFP08] Adobe Pixel Bender in Flash Player 10 Beta (Blog), 2008, Online abrufbar: http://www.kaourantin.net/2008/05/adobe-pixel-bender-in-flash-player-10.html (07.02.2010)

[BSCONTACT09] BS Contact 7.2 - Release Notes - Document version 1.3, 2009, Online abrufbar: http://www.bitmanagement.de/developer (07.01.2010)

[C3DL09] What is C3DL?, 2009, Online abrufbar: http://www.c3dl.org (30.12.2009)

[CARBEL97] R. Carey, G. Bell: The Annotated VRML 97 Reference Manual, 1997, Online abrufbar: http://accad.osu.edu/~pgerstma/class/vnv/resources/info/AnnotatedVrmlRef/ch1.htm#1.3 (1.12.2009)

[CORTONA09] Cortona3D Viewer 6.0, 2009, Online abrufbar: http://www.cortona3d.com/cortona3d/media/downloads/Viewer/win.html (07.01.2010)

[COSMO09] Download and Install the Cosmo Player VRML Plugin, 2009, Online abrufbar: http://cic.nist.gov/vrml/cosmoplayer.html (07.01.2010)

[COSMO98] Cosmo Player 2.1.1 -Release Notes for Windows 95/98 and Windows NT, 1998, Online abrufbar: http://www.3dezine.com/www/CosmoPlayer/Doc/frames-relnotes.html (10.04.2010)

[CRSHWI09] A. B. Craig, W. R. Sherman, J. D. Will: Developing Virtual Reality Applications – Foundations of Effective Design, 2009, Elsevier, Burlington

[DACHSELT06] R. Dachselt: Eine deklarative Komponentenarchitektur und Interaktionsbausteine für dreidimensionale multimediale Anwendungen - Kapitel 2: 3D-Echtzeitgrafik im World Wide Web, 2006

[DAY99] B. Day: 3D graphics programming in Java, Part 3: OpenGL, 1999, Online abrufbar: http://www.javaworld.com/jw-05-1999/jw-05-media.html?page=2 (03.01.2010)

[DEBERG93] M. de Berg: Ray Shooting, Depth Orders and Hidden Surface Removal, 1993, Springer-Verlag, Berlin

[FESTA02] P. Festa: Bringing 3D to the Web, 2002, Online abrufbar: http://news.cnet.com/2100-1023-844985.html (1.12.2009)

[FIVE3D10] Five3D – Flash Interactive Vector-based 3D, 2010, Online abrufbar: http://five3d.mathieu-badimon.com (03.01.2010)

[FLALLG10] ADOBE FLASH CS4 PROFESSIONAL, 2010, Online abrufbar: http://www.adobe.com/products/flash/?promoid=BPDEE (27.03.2010)

[FLASHP10] Understanding the 3D features of Flash Player and the AIR runtime, 2010, Online abrufbar: http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5467498E-BCF8-454f-8607-A51AD392CC07.html (03.01.2010)

[FLBROWSER10] Flash Player 10 – Systemanforderungen, 2010, Online abrufbar: http://www.adobe.com/de/products/flashplayer/systemreqs (07.01.2010)

[FLEX10] Flex 3 SDK Downloads, 2010, Online abrufbar: http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+3 (14.02.2010)

[FLFAQ] Adobe CS4 Professional - FAQ, 2010, Online abrufbar: http://www.adobe.com/products/flash/faq/?promoid=DRHWS (14.02.2010)

[FLHELP] Adobe Flash CS3 Hilfe

[FLMM07] Mip map what? (Blog), 2007, Online abrufbar: http://www.kaourantin.net/2007/06/mip-map-what.html (07.02.2010)

[FPPENE10] Flash Player Version Penetration, 2010, Online abrufbar: http://www.adobe.com/products/player_census/flashplayer/version_penetration.html (30.04.2010)

[GL4JAVA07] OpenGL[tm] for Java[tm], 2007, Online abrufbar: http://sourceforge.net/projects/gl4java (03.01.2010)

[GL4JAVA97] OpenGL for Java/Overview/Historie, 1997, Online abrufbar: http://gl4java.sourceforge.net/docs/overview/history.html (03.01.2010)

[GLGE10] GLGE: WebGL for the lazy, 2010, Online abrufbar: http://www.glge.org (05.01.2010)

[GOLEM08] Nvision: Erste Beta von DirectX-11 noch 2008, 2008, Online abrufbar: http://www.golem.de/0808/62007.html (26.12.2009)

[GOLXML3D10] XML3D - Browser lernen Echtzeit-Raytracing, 2010, Online abrufbar: http://www.golem.de/1003/73645.html (12.04.2010)

[GRGR02] P. Gross, M. Gross: Macromedia Director 8.5 Shockwave Studio für 3D, Das offizielle Trainingsbuch, 2002, Markt+Technik Verlag, München

[HEBA04] D. Hearn, M. Baker: Computer Graphics with OpenGL, Third Edition, 2004, Pearson Prentice Hall, New York

[HELG08] D. Helgason: Thoughts On Browser Plugin Penetration (Blog), 2008, Online abrufbar: http://blogs.unity3d.com/2008/03/31/thoughts-on-browser-plugin-penetration (07.01.2010)

[HIHY09] I. Hickson, D. Hyatt: HTML5, A vocabulary and associated APIs for HTML and XHTML, 2009, Online abrufbar: http://dev.w3.org/html5/spec/Overview.html#the-canvas-element (30.12.2009)

[JAVA3D00] Java 3D API Tutorial, 2000, Online abrufbar: http://java.sun.com/developer/onlineTraining/java3d/index.html (30.04.2010)

[JAVA3D08] Java 3D unter der GPL veröffentlicht, 2008, Online abrufbar: http://www.golem.de/0803/58091.html (02.01.2010)

[JAVA3D09] Java3DRoadmap, 2009, Online abrufbar: http://wiki.java.net/bin/view/Javadesktop/Java3DRoadmap (02.01.2010)

[JAVA3DEG10] List of Java 3D Users & Applications, 2010 Online abrufbar: http://wiki.java.net/bin/view/Javadesktop/Java3DUsers (30.04.2010)

[JAVABROWSER10] Java™ SE 6 Release Notes Supported System Configurations, 2010, Online abrufbar: http://java.sun.com/javase/6/webnotes/install/system-configurations.html (12.01.2010)

[JAVAHW10] Welche Verbesserungen bietet die aktuelle Version Java 6 Update10 (6u10)?, 2010, Online abrufbar: http://java.com/de/download/help/features_java6update10.xml (12.01.2010)

[JAVAPEN10] Flash Player penetration, Millward Brown survey, 2010, Online abrufbar: http://www.adobe.com/products/player_census/flashplayer (22.04.2010)

[JOGL09] Über JOGL - Entstehung, 2009, Online abrufbar: http://www.jogl.info/entstehung.htm (03.01.2010)

[JOGLLIZ09] jogl-git/LICENSE.txt, 2009, Online abrufbar: http://kenai.com/projects/jogl/sources/jogl-git/content/LICENSE.txt (03.03.2010)

[JOGLWIKI09] Java™ Binding for the OpenGL® API Wiki, 2009, Online abrufbar: http://kenai.com/projects/jogl/pages/Home (14.02.2010)

[JS3D07] JS3D (alpha): The 3d Javascript Graphics Layer, 2007, Online abrufbar: http://www.wxs.ca/js3d (30.12.2009)

[JSPARROW00] JSparrow - An Implementation of Java binding for OpenGL, 2000, Online abrufbar: http://home.att.ne.jp/red/aruga/jsparrow/index.html (03.01.2010)

[KHRON09] Khronos WebGL Wiki, 2009, Online abrufbar: http://khronos.org/webgl/wiki/Getting_Started (01.01.2010)

[KHRONPR09] Khronos Launches Initiative to Create Open Royalty Free Standard for Accelerated 3D on the Web, 2009, Online abrufbar: http://www.khronos.org/news/press/releases/khronos-launches-initiative-for-free-standard-for-accelerated-3d-on-web (21.04.2010)

[KHRONPRA09] Khronos Details WebGL Initiative to Bring Hardware-Accelerated 3D Graphics to the Internet, 2009, Online abrufbar: http://www.khronos.org/news/press/releases/khronos-webgl-initiative-hardware-accelerated-3d-graphics-internet (01.01.2010)

[KIEN07] W. Kienreich: APPEAR - 3D Visualization Library, User's Guide, 2007

[KIEN06] W. Kienreich: Information and Knowledge Visualisation – An Oblique View, 2006

[KIKRA08] W. Kienreich, P. Kraker: Comparative Evaluation of Two Systems for the Visual Navigation of Encyclopedia Knowledge Spaces, 2008

[KIGR05] W. Kienreich, M. Granitzer: Visualising Knowledge Webs for Encyclopedia Articles, 2005

[LUI05] F. Lui: Platform independent real-time X3D shaders and their applications in bioinformatics visualization, 2005, Online abrufbar: http://etd.gsu.edu/theses/available/etd-11232005-114921/unrestricted/Feng_Liu_2005_PhD.pdf (10.12.2009)

[LWJGL07] LWJGL 1.0 Released, 2007, Online abrufbar: http://lwjgl.org/forum/index.php/topic,2202 (03.01.2010)

[LWJGL10] LWJGL - Lightweight Java Game Library: Introduction, 2010, Online abrufbar: http://lwjgl.org (03.03.2010)

[MAHE05] C. MacGillivray, A. Head: 3D for the Web: Interactive 3D Animation Using 3ds Max, Flash and Director, 2005, Elsevier, Burlington

[MSCHRA98] Microsoft debuts Chromeffects, 1998, Online abrufbar: http://news.cnet.com/2100-1001-213566.html (14.02.2010)

[MSCHRB98] Microsoft shelves Chromeffects, 1998, Online abrufbar: http://news.cnet.com/Microsoft-shelves-Chromeffects/2100-1023_3-217885.html?tag=st.rc.targ_mb (14.02.2010)

[MSDN09] Direct3D 11 Features, 2009, Online abrufbar: http://msdn.microsoft.com/en-us/library/ff476342%28v=VS.85%29.aspx (21.04.2010)

[MSDN10] Tessellation Overview, 2010, Online abrufbar: http://msdn.microsoft.com/en-us/library/ff476340%28v=VS.85%29.aspx (21.04.2010)

[NVIDIA10] Produkt – Überblick: NVIDIA GeForce GTX 480, 2010, Online abrufbar: http://www.nvidia.de/object/product_geforce_gtx_480_de.html (07.04.2010)

[O3DBLOG09] Introducing O3D (Blog), 2009, Online abrufbar: http://o3d.blogspot.com/2009/04/toward-open-web-standard-for-3d.html (01.01.2010)

[O3DDEVG10] Developer's Guide, 2010, Online abrufbar: http://code.google.com/intl/de-DE/apis/o3d/docs/devguideintro.html (22.04.2010)

[O3DFAQ09] O3D - FAQ, 2009, Online abrufbar: http://code.google.com/p/o3d/wiki/FAQs (14.02.2010)

[O3DHW09] Designing a 3D API for the browser, 2009, Online abrufbar: http://o3d.blogspot.com/2009/04/designing-3d-api-for-browser.html (12.01.2010)

[O3DINSTR10] O3D - Installation Instructions, 2010, Online abrufbar: http://code.google.com/intl/de-DE/apis/o3d/docs/gettingstarted.html (12.01.2010)

[O3DPRG10] O3D - Program Structure, 2010, Online abrufbar: http://code.google.com/intl/de-DE/apis/o3d/docs/devguidechap01.html (14.02.2010)

[OCTAGA10] Reference Manual - Octaga Player - Version 3.0, 2010, Online abrufbar: http://www.octaga.com/freedownloads/OctagaPlayer/3.0/userManual.pdf (30.04.2010)

[PBFF08] Pixel Bender basics for Flash, 2008, Online abrufbar: http://www.adobe.com/devnet/flash/articles/pixel_bender_basics.html (07.02.2010)

[PV3D10] Papervision3D (Blog), 2010, Online abrufbar: http://blog.papervision3d.org (03.01.2010)

[PV3DDL10] Papervision3D - Downloads, 2010, Online abrufbar: http://code.google.com/p/papervision3d/downloads/list (03.01.2010)

[PV3DFAQ08] Papervision3D - Getting_Started_FAQ, 2008, Online abrufbar: http://code.google.com/p/papervision3d/wiki/Getting_Started_FAQ (03.01.2010)

[PV3DLOD08] Papervision LOD – SimpleLevelOfDetail (Blog), 2008, Online abrufbar: http://blog.zupko.info/?p=137 (06.02.2010)

[PV3DQT08] Papervision QuadTree Support (Blog), 2008, Online abrufbar: http://blog.papervision3d.org/2008/10/14/papervision-quadtree-support (04.02.2010)

[PV3DUQ08] Using QuadTrees In Papervision3D (Blog), 2008, Online abrufbar: http://blog.zupko.info/?p=177 (04.02.2010)

[PV3UV10] Papervision3D: uv-coordinates (Forum), 2010, Online abrufbar: http://forum.papervision3d.org/viewtopic.php?f=13&t=2031 (21.04.2010)

[PV3DV09] Papervision3D is Shifting Gears (Blog), 2009, Online abrufbar: http://blog.papervision3d.org/2009/10/13/papervision3d-is-shifting-gears (14.02.2010)

[RIASTAT10] Rich Internet Application Statistics, 2010, Online abrufbar: http://riastats.com (30.04.2010)

[SANDY3D09] Sandy 3D Engine, 2009, Online abrufbar: http://www.flashsandy.org (30.12.2009)

[SANDY3D10] Sandy 3D Engine, 2010, Online abrufbar: http://www.flashsandy.org (03.01.2010)

[SANDYBM10] Technical notes about Sandy3D engine, 2010, Online abrufbar: http://www.flashsandy.org/technotes (07.02.2010)

[SANDYF10] Sandy 3D: vertices... (Forum), 2010, Online abrufbar: http://www.flashsandy.org/forum/viewtopic.php?f=5&t=2087 (21.04.2010)

[SANDYOS10] Can I use Sandy in a commercial project?, 2010, Online abrufbar: http://www.flashsandy.org/faq/can_i_use_sandy_in_commercial_projects (03.01.2010)

[SANDYTEAM10] Sandy development team, 2010, Online abrufbar: http://www.flashsandy.org/developers/team (03.01.2010)

[SANDYWHY10] Why this project ?, 2010, Online abrufbar: http://www.flashsandy.org/faq/why_this_project (03.01.2010)

[SCENEJS10] SceneJS, 2010, Online abrufbar: http://www.scenejs.com (05.01.2010)

[SHOCK3D07] Shockwave Player version history, 2007, Online abrufbar: http://kb2.adobe.com/cps/148/tn_14820.html (30.12.2009)

[SHOUT3D] Shout3D®, It Just Works!®, Online abrufbar: http://shout3d.net (15.04.2010)

[SHWPENE10] Shockwave Player Version Penetration, 2010, Online abrufbar: http://www.adobe.com/products/player_census/shockwaveplayer/version_penetration.html (22.04.2010)

[SILVER3D07] Silverlight and 3D (Forum), 2007, Online abrufbar: http://forums.silverlight.net/forums/t/1134.aspx (14.02.2010)

[SILVERBROWSER10] Silverlight - FAQ, 2010, Online abrufbar: http://www.microsoft.com/silverlight/resources/faq/default.aspx#general (30.04.2010)

[SILVERHW09] Silverlight Tip of the Day #105 – How to Enable GPU Acceleration (Blog), 2009, Online abrufbar: http://blogs.silverlight.net/blogs/msnow/archive/2009/04/01/silverlight-tip-of-the-day-104-how-to-enable-gpu-acceleration.aspx (14.02.2010)

[SILVERSDK09] Microsoft® Silverlight™ 3 SDK, 2009, Online abrufbar: http://www.microsoft.com/downloads/details.aspx?FamilyID=1ea49236-0de7-41b1-81c8-a126ff39975b&displaylang=en (30.04.2010)

[SPILLE08] C. Spille: PCGH Rückblick: Entwicklung der 3D-Grafik (Teil 1 - 6), 2008, Online abrufbar (Teil 1): http://www.pcgameshardware.de/aid,627809/PCGH-Rueckblick-Entwicklung-der-3D-Grafik-Teil-1/Grafikkarte/Wissen (10.12.2009)

[SPILLE09] C. Spille: Evolution der Grafikkarten: Ati/AMD, 2009, Online abrufbar: http://www.pcgameshardware.de/aid,668614/Die-wichtigsten-Ati-Grafikkarten-Von-Rage-bis-Radeon-HD-5870/Grafikkarte/Wissen (26.12.2009)

[STAPPLER97] H. Stappler: Sonderausstattung für Ihren WWW-Browser, 1997, Online abrufbar: http://comment.univie.ac.at/97-3/36 (01.12.2009)

[SWBROWSER10] Shockwave Player – Systemanforderungen, 2010 ,Online abrufbar: http://www.adobe.com/de/products/shockwaveplayer/productinfo/systemreqs (07.01.2010)

[SWEG10] Adobe Director 11.5 - Explore new dimensions in rich multimedia authoring, 2010, Online abrufbar: http://www.adobe.com/products/director (14.02.2010)

[SWFZ08] AS3 Flash 3D Engine SWFZ Source Code Goes Open Source (Blog), 2008, Online abrufbar: http://drawlogic.com/2008/04/11/as3-swfz-source-code-goes-open-source (05.02.2010)

[SWHW10] 3D game development, 2010, Online abrufbar: http://www.adobe.com/products/director/3d_game_programming (14.02.2010)

[SWPR10] Adobe Store – Österreich: Adobe Director 11.5, 2010, Online abrufbar: https://store2.adobe.com/cfusion/store/html/index.cfm?store=OLS-AT&event=displayProduct&categoryPath=/Applications/Director&distributionMethod=FULL (14.02.2010)

[TORQUE10] Torque 3D, 2010, Online abrufbar: http://www.torquepowered.com/products/torque-3d (09.03.2010)

[TORQUEDOC10] Torque 3D documentation, 2010, Online abrufbar: http://docs.torquepowered.com/torque-3d/official/content/documentation/Beginners%20Guide/Getting%20Started/ToolBox.html (07.01.2010)

[TORQUEDOCB10] Torque 3D documentation, 2010, Online abrufbar: http://docs.torquepowered.com/torque-3d/official (30.04.2010)

[TOWI09] P. Tondeur, J. Winder: Papervision3D Essentials, 2009, Packt Publishing, Birmingham

[UNITY3D10] Unity Overview, 2010, Online abrufbar: http://unity3d.com/unity (2010)

[UNITY3D09] Unity News, 2009, Online abrufbar: http://unity3d.com/company/news.html (30.12.2009)

[UNITYBROWSER10] Unity Web Player, 2010, Online abrufbar: http://unity3d.com/webplayer (12.01.2010)

[UNITYMAN10] Unity Manual, 2010, Online abrufbar: http://unity3d.com/support/documentation/Manual/index.html (22.04.2010)

[UNITYHW10] Graphical Fidelity, 2010, Online abrufbar: http://unity3d.com/unity/features/graphics.html (12.01.2010)

[URO08] T. Uro: What does GPU acceleration mean? (Blog), 2008, Online abrufbar: http://www.kaourantin.net/2008/05/what-does-gpu-acceleration-mean.html (03.01.2010)

[VIVATY09] Does Vivaty require Direct3D (like Flux) or will it run with OpenGL?, 2009, Online abrufbar: http://getsatisfaction.com/vivaty/topics/does_vivaty_require_direct3d_like_flux_or_will_it_run_with_opengl# (12.04.2010)

[VRMLSPEC03] The Virtual Reality Modeling Language, 2003, Online verfügbar: http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97 (15.04.2010)

[VUKI07] V. Vukićević: Canvas 3D - GL power, web-style (Blog), 2007, Online abrufbar: http://blog.vlad1.com/2007/11/26/canvas-3d-gl-power-web-style (30.12.2009)

[VUKI09] V. Vukićević: WebGL in Firefox Nightly Builds (Blog), 2009, Online abrufbar: http://blog.vlad1.com/2009/09/18/webgl-in-firefox-nightly-builds (30.12.2009)

[W3DHUD] Interview mit Alan Hudson vom Web3D Consortium (Video), 2008, Online abrufbar: http://vids.myspace.com/index.cfm?fuseaction=vids.individual&VideoID=41597834 (14.02.2010)

[WABO01] A. E. Walsh, M. Bourges-Sevenier: Core Web 3D, 2001, Prentice Hall PTR, New York

[WEBGLINT09] Interview mit Neil Trevett, Vice President Embedded Content, NVIDIA - President, Khronos Group, 2009, Online abrufbar: http://www.3d-test.com/interviews/khronos_2.htm (14.02.2010)

[WORDNET09] WordNet®, Princeton University, 2009, Online abrufbar: http://wordnet.princeton.edu (27.11.2009)

[X3DDRAFT02] Web3D Consortium Releases X3D Final Working Draft, 2002, Online abrufbar: http://www.web3d.org/press/detail/web3d_consortium_releases_x3d_final_working_draft (29.12.2009)

[X3DOM09] X3DOM – Instant 3D the HTML way, 2010, Online abrufbar: http://www.x3dom.org (14.02.2010)

[X3DSPEC08] Extensible 3D (X3D), Part 1: Architecture and base components, ISO/IEC 19775-1:2008, 2008, Online abrufbar: http://www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification (15.04.2010)

[XML3D10] XML3D, 2010, Online abrufbar: http://graphics.cs.uni-sb.de/489 (12.04.2010)

Anhang A (Appear XML-File)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<appear>
  <paramstore>
    <param type="numeric" name="cam_pos_x" value="0"/>
    <param type="numeric" name="cam_pos_y" value="20"/>
    <param type="numeric" name="cam_pos_z" value="-30"/>
    <param type="numeric" name="cam_dir_x" value="0"/>
    <param type="numeric" name="cam_dir_y" value="0"/>
    <param type="numeric" name="cam_dir_z" value="0"/>
  </paramstore>
  <mediastore>
    <shape type="cylinder" name="sbase" radius="8" height="0.01" edges="30" />
    <shape type="cone" name="sname" radius="0.25" height="0.5" edges="10"/>
    <shape type="cylinder" name="sterm" radius="0.25" height="0.5" edges="10"/>
    <shape type="sphere" name="sarea" radius="0.25" slices="10" stacks="10"/>
    <shape type="cube" name="snoun" width="0.25" height="0.25" depth="0.25"/>
    <color name="fcolor" r="0" g="0" b="0"/>
    <image name="ibase" source="material/base.jpg"/>
    <font name="ffont" face="Arial" size="8"/>
    <material name="mbase" type="image" image="ibase"/>
    <material name="mname" type="shaded" transparency="0.0">
      <ambient r="0.2" g="0.2" b="0.2"/>
      <color r="0.2" g="0.3" b="0.8"/>
    </material>
    <material name="marea" type="shaded" transparency="0.0">
      <ambient r="0.2" g="0.2" b="0.2"/>
      <color r="0.2" g="0.8" b="0.3"/>
    </material>
    <material name="mterm" type="shaded" transparency="0.0">
      <ambient r="0.2" g="0.2" b="0.2"/>
      <color r="0.8" g="0.3" b="0.2"/>
    </material>
    <material name="mnoun" type="shaded" transparency="0.0">
      <ambient r="0.2" g="0.2" b="0.2"/>
      <color r="1.0" g="1.0" b="0.0"/>
    </material>
  </mediastore>
  <scene name="disc">
    <node>
      <node type="scene">
        <camera name="cam" width="800" height="600" near="0.1" far="100" field="25">
          <source x="=cam_pos_x" y="=cam_pos_y" z="=cam_pos_z"/>
          <direction x="=cam_dir_x" y="=cam_dir_y" z="=cam_dir_z"/>
        </camera>
        <node type="scene" name="sun">
          <light type="point">
            <source x="10" y="20" z="-10"/>
            <color r="1.0" g="1.0" b="1.0"/>
          </light>
        </node>
        <node type="scene" name="nbase" active="true" autoRotation="true">
          <decor type="shadow" source="nitem" caster="sun" hierarchical="true"/>
          <visual shape="sbase" material="mbase" background="true"/>
        </node>
      </node>
      <node type="scene" name="nitem" active="true" autoRotation="true">
        <node type="scene" name="n0" active="true">
          <transform>
            <translation x="+1.000" y="0" z="+0.000"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="2.000" y="2.000" z="2.000"/>
            </transform>
            <visual shape="sarea" material="marea"/>
            <event name="e1" type="mouseclick" action="calljs" param="nodename"/>
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="vereinigte staaten von amerika"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
```

```
      </node>
    </node>
    <node type="scene" name="n1" active="true">
      <transform>
        <translation x="-0.500" y="0" z="+0.866"/>
      </transform>
      <node type="scene" active="true">
        <transform>
          <scaling x="1.693" y="1.693" z="1.693"/>
        </transform>
        <visual shape="sarea" material="marea"/>
        <event name="e1" type="mouseclick" action="calljs" param="nodename" />
      </node>
      <node type="layer" active="true">
        <offset x="0" y="1.2"/>
        <visual>
          <shape type="text" font="ffont" text="irak"/>
          <material type="color" color="fcolor" transparency="0.0"/>
        </visual>
      </node>
    </node>
    <node type="scene" name="n2" active="true">
      <transform>
        <translation x="-0.500" y="0" z="-0.866"/>
      </transform>
      <node type="scene" active="true">
        <transform>
          <scaling x="1.634" y="1.634" z="1.634"/>
        </transform>
        <visual shape="sarea" material="marea"/>
        <event name="e2" type="mouseclick" action="calljs" param="nodename" />
      </node>
      <node type="layer" active="true">
        <offset x="0" y="1.2"/>
        <visual>
          <shape type="text" font="ffont" text="türkei"/>
          <material type="color" color="fcolor" transparency="0.0"/>
        </visual>
      </node>
    </node>
    <node type="scene" name="n3" active="true">
      <transform>
        <translation x="+3.000" y="0" z="+0.000"/>
      </transform>
      <node type="scene" active="true">
        <transform>
          <scaling x="1.601" y="1.601" z="1.601"/>
        </transform>
        <visual shape="sarea" material="marea"/>
        <event name="e3" type="mouseclick" action="calljs" param="nodename" />
      </node>
      <node type="layer" active="true">
        <offset x="0" y="1.2"/>
        <visual>
          <shape type="text" font="ffont" text="iran"/>
          <material type="color" color="fcolor" transparency="0.0"/>
        </visual>
      </node>
    </node>
    <node type="scene" name="n4" active="true">
      <transform>
        <translation x="+1.500" y="0" z="+2.598"/>
      </transform>
      <node type="scene" active="true">
        <transform>
          <scaling x="1.196" y="1.196" z="1.196"/>
        </transform>
        <visual shape="sname" material="mname"/>
        <event name="e4" type="mouseclick" action="calljs" param="nodename" />
      </node>
      <node type="layer" active="true">
        <offset x="0" y="1.2"/>
        <visual>
          <shape type="text" font="ffont" text="george bush"/>
          <material type="color" color="fcolor" transparency="0.0"/>
        </visual>
      </node>
```

```
        </node>
        <node type="scene" name="n5" active="true">
          <transform>
            <translation x="-1.500" y="0" z="+2.598"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.431" y="1.431" z="1.431"/>
            </transform>
            <visual shape="sarea" material="marea"/>
            <event name="e5" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="afghanistan"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n6" active="true">
          <transform>
            <translation x="-3.000" y="0" z="+0.000"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.137" y="1.137" z="1.137"/>
            </transform>
            <visual shape="sterm" material="mterm"/>
            <event name="e6" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="nato"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n7" active="true">
          <transform>
            <translation x="-1.500" y="0" z="-2.598"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.327" y="1.327" z="1.327"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e7" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="präsident"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n8" active="true">
          <transform>
            <translation x="+1.500" y="0" z="-2.598"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.307" y="1.307" z="1.307"/>
            </transform>
            <visual shape="sarea" material="marea"/>
            <event name="e8" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="washington"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
```

```
        <node type="scene" name="n9" active="true">
          <transform>
            <translation x="+5.000" y="0" z="+0.000"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.163" y="1.163" z="1.163"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e9" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="us-präsident"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n10" active="true">
          <transform>
            <translation x="+3.830" y="0" z="+3.214"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.242" y="1.242" z="1.242"/>
            </transform>
            <visual shape="sarea" material="marea"/>
            <event name="e10" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="teheran"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n11" active="true">
          <transform>
            <translation x="+0.868" y="0" z="+4.924"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.242" y="1.242" z="1.242"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e11" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="regierung"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n12" active="true">
          <transform>
            <translation x="-2.500" y="0" z="+4.330"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.039" y="1.039" z="1.039"/>
            </transform>
            <visual shape="sterm" material="mterm"/>
            <event name="e12" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="europäische union"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n13" active="true">
```

```
          <transform>
            <translation x="-4.698" y="0" z="+1.710"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.052" y="1.052" z="1.052"/>
            </transform>
            <visual shape="sname" material="mname"/>
            <event name="e13" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="charles kupchan"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n14" active="true">
          <transform>
            <translation x="-4.698" y="0" z="-1.710"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.163" y="1.163" z="1.163"/>
            </transform>
            <visual shape="sarea" material="marea"/>
            <event name="e14" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="ankara"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n15" active="true">
          <transform>
            <translation x="-2.500" y="0" z="-4.330"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.052" y="1.052" z="1.052"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e15" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="us-soldaten"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n16" active="true">
          <transform>
            <translation x="+0.868" y="0" z="-4.924"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.157" y="1.157" z="1.157"/>
            </transform>
            <visual shape="sarea" material="marea"/>
            <event name="e16" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="israel"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n17" active="true">
          <transform>
```

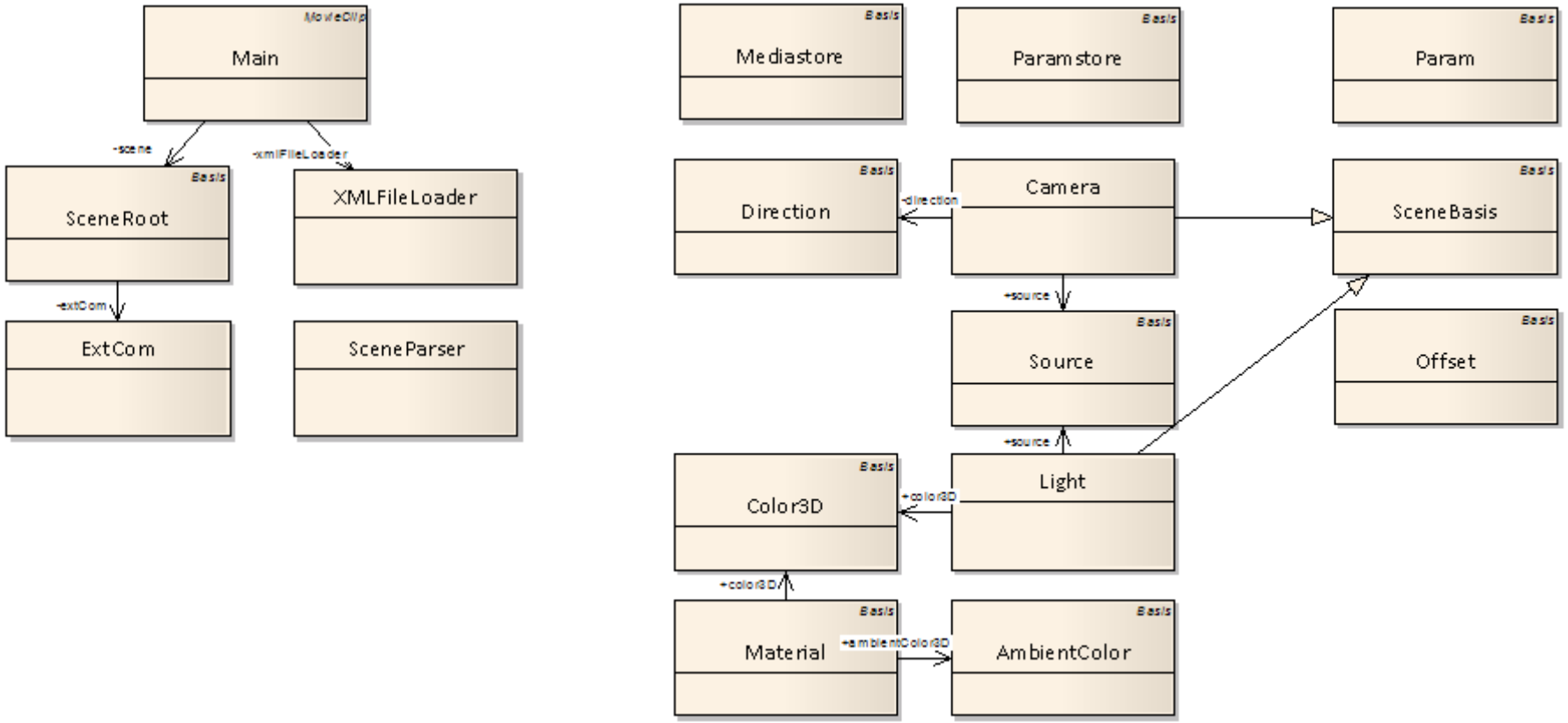
```
            <translation x="+3.830" y="0" z="-3.214"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.046" y="1.046" z="1.046"/>
            </transform>
            <visual shape="sname" material="mname"/>
            <event name="e17" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="jimmy carter"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n18" active="true">
          <transform>
            <translation x="+7.000" y="0" z="+0.000"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.275" y="1.275" z="1.275"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e18" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="staat"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n19" active="true">
          <transform>
            <translation x="+6.062" y="0" z="+3.500"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.131" y="1.131" z="1.131"/>
            </transform>
            <visual shape="sarea" material="marea"/>
            <event name="e19" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="bagdad"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n20" active="true">
          <transform>
            <translation x="+3.500" y="0" z="+6.062"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.150" y="1.150" z="1.150"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e20" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="istanbul"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n21" active="true">
          <transform>
            <translation x="+0.000" y="0" z="+7.000"/>
```

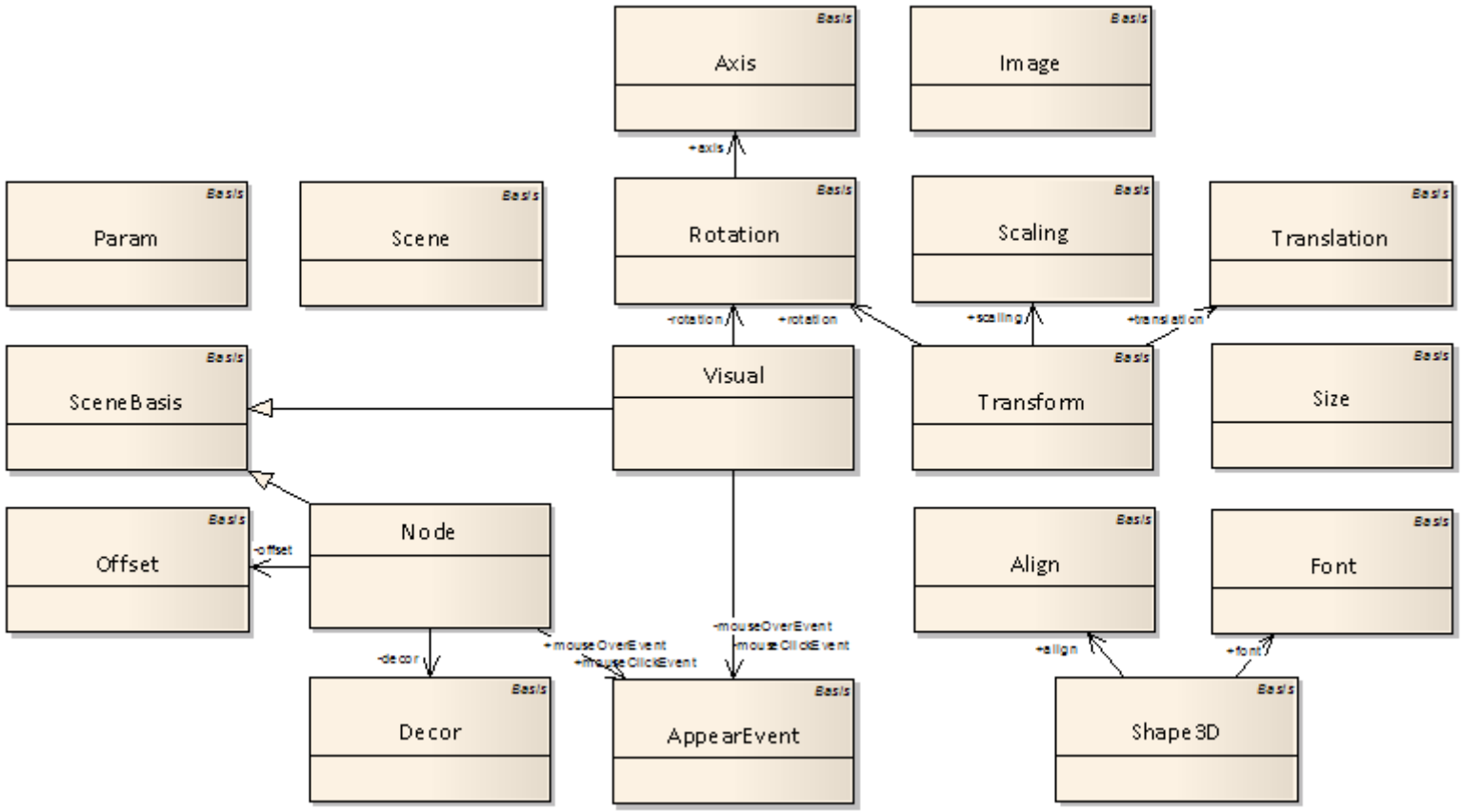
```
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.033" y="1.033" z="1.033"/>
            </transform>
            <visual shape="sname" material="mname"/>
            <event name="e21" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="henry kissinger"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n22" active="true">
          <transform>
            <translation x="-3.500" y="0" z="+6.062"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.033" y="1.033" z="1.033"/>
            </transform>
            <visual shape="sname" material="mname"/>
            <event name="e22" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="barack hussein"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n23" active="true">
          <transform>
            <translation x="-6.062" y="0" z="+3.500"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.013" y="1.013" z="1.013"/>
            </transform>
            <visual shape="sterm" material="mterm"/>
            <event name="e23" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="weisses haus"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n24" active="true">
          <transform>
            <translation x="-7.000" y="0" z="+0.000"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.163" y="1.163" z="1.163"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e24" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="premier"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n25" active="true">
          <transform>
            <translation x="-6.062" y="0" z="-3.500"/>
          </transform>
```

```
          <node type="scene" active="true">
            <transform>
              <scaling x="1.118" y="1.118" z="1.118"/>
            </transform>
            <visual shape="sarea" material="marea"/>
            <event name="e25" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="europa"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n26" active="true">
          <transform>
            <translation x="-3.500" y="0" z="-6.062"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.288" y="1.288" z="1.288"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e26" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="land"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n27" active="true">
          <transform>
            <translation x="-0.000" y="0" z="-7.000"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.111" y="1.111" z="1.111"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e27" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="parlament"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n28" active="true">
          <transform>
            <translation x="+3.500" y="0" z="-6.062"/>
          </transform>
          <node type="scene" active="true">
            <transform>
              <scaling x="1.105" y="1.105" z="1.105"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e28" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="rasmussen"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
        <node type="scene" name="n29" active="true">
          <transform>
            <translation x="+6.062" y="0" z="-3.500"/>
          </transform>
          <node type="scene" active="true">
```

```
            <transform>
              <scaling x="1.000" y="1.000" z="1.000"/>
            </transform>
            <visual shape="snoun" material="mnoun"/>
            <event name="e29" type="mouseclick" action="calljs" param="nodename" />
          </node>
          <node type="layer" active="true">
            <offset x="0" y="1.2"/>
            <visual>
              <shape type="text" font="ffont" text="eu-beitritt"/>
              <material type="color" color="fcolor" transparency="0.0"/>
            </visual>
          </node>
        </node>
      </node>
    </node>
  </scene>
</appear>
```

Anhang B (Klassendiagramm)




Basis
Axis
Basis
Image
+axis
Basis
Param
Basis
Scene
Basis
Rotation
Basis
Scaling
Basis
Translation
+rotation
+rotation
+scaling
+translation
Basis
SceneBasis
Visual
Basis
Transform
Basis
Size
Basis
Offset
+offset
Node
Basis
Align
Basis
Font
+mouseOverEvent
+mouseClickEvent
+mouseOverEvent
+mouseClickEvent
+decor
+align
+font
Basis
Decor
Basis
AppearEvent
Basis
Shape3D