3D für das Web

PEARSON Education

ADDISON-WESLEY

Frank Lamprecht

3D für das Web

Grundlagen und Beispiele mit Flash, Director, DHTML und Java

ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist bei der Deutschen Bibliothek erhältlich

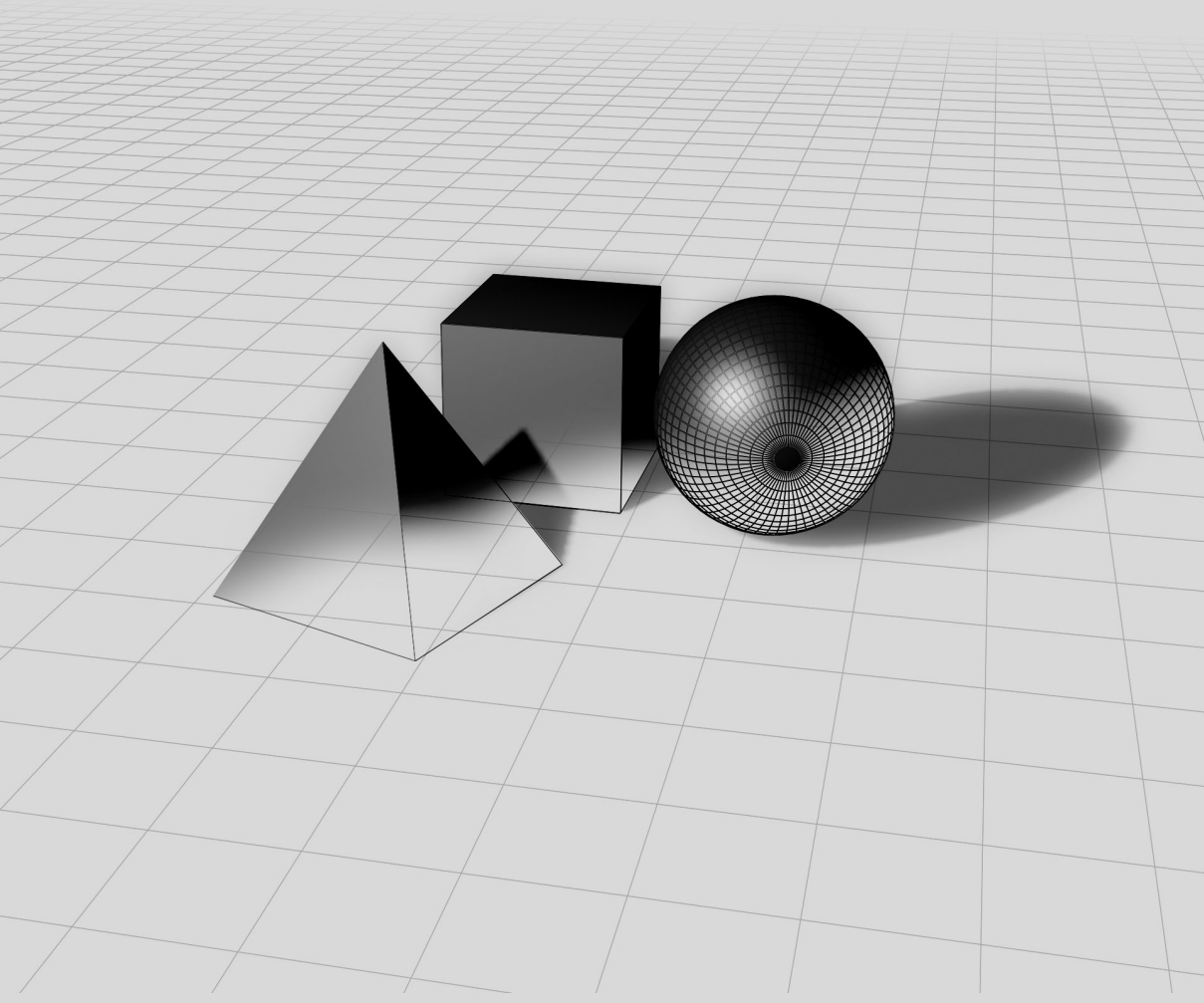
10 9 8 7 6 5 4 3 2 1

05 04 03 02

ISBN 3-8273-1970-6

Einbandgestaltung:	Marco Lindenbeck, Parsdorf bei München
Lektorat:	Klaus Hofmann, khofmann@pearson.de
Korrektorat:	Simone Meisner, Fürstenfeldbruck
Herstellung:	Anna Plenk, aplenk@pearson.de
CD-Mastering:	Gregor Kopietz, gkopietz@pearson.de
Satz:	mediaService – Siegen (www.media-service.tv)
Druck und Verarbeitung:	Kösel, Kempten (www.KoeselBuch.de)

Printed in Germany



Inhaltsverzeichnis

Einleitung

3D fürs Web. Während man früher damit zumeist zweifelnde Blicke erntete, ist die Zeit heute reif dafür.

Es ist so weit

Inzwischen steckt in fast jedem zum Internet-Surfen verwendeten PC eine Grafikkarte, die zumindest grundlegende Unterstützung für 3D-Beschleunigung bietet. Aber nicht nur die Grafikkarten, sondern auch die PCs selbst sind inzwischen leistungsfähig genug, um die aufwändigen Berechnungen für 3D-Animationen in akzeptabler Zeit durchzuführen. Außerdem haben sich die Standards OpenGL und DirectX zur standardisierten Programmierung der Hardware durchgesetzt.

Auf der Software-Seite sieht die Lage allerdings schlechter aus. Die Standardisierungsbemühungen des W3C hinken dieser Entwicklung etwas hinterher. Deshalb gibt es heute leider noch keinen etablierten Standard für das Format von 3D-Darstellungen, welcher in die Standardbrowser integriert sein könnte.

Aus diesem Grund halten Sie ein Buch in den Händen, das nicht VRML in allen Details beschreibt oder Schritt für Schritt in Macromedia Director einführt. Dieses Buch beschreibt einen möglichst großen Teil aller aktuellen Technologien, die es erlauben, 3D-Darstellungen auf Webseiten zu bringen. Einen möglichst großen Teil deshalb, weil es viele verschiedene Einsatzgebiete, Zielgruppen und Randbedingungen dafür gibt, welche Technik man für ein Projekt tatsächlich verwenden sollte.

Die Praxis-Kapitel beschreiben neben den Methoden für real berechnete 3D-Grafik auch Methoden zur Simulation des 3D-Eindrucks. Sie werden Schritt-für-Schritt-Anleitungen für mehrere verschiedene Technologien finden. Und die Technologien, die nicht näher erläutert werden, sind in einer großen Übersicht nebeneinander gestellt, um einen Ansatzpunkt für weitere Erkundungszüge zu bieten.

Wofür 3D?

Warum und wofür sollte man 3D-Grafiken auf Webseiten einsetzen? Die Gründe dafür sind sehr vielfältig. Die folgende Liste vermittelt einen Eindruck davon:

- Produktpräsentationen: Ein Hinderungsgrund für viele Personen, im Web einzukaufen, ist immer noch, dass sie das Produkt nicht anfassen und wirklich von allen Seiten betrachten können. 3D-Darstellungen lösen nicht wirklich das Problem des kinästhetischen Erlebens, aber das Betrachten von allen Seiten wird möglich. Es gibt Beispiele, bei denen man etwa ein Handy drehen und auch zerlegen kann, um zu sehen, aus welchen Teilen es zusammengesetzt ist. Oder man kann bei einem

Auto die Türen öffnen und sich virtuell hinters Lenkrad setzen. Die Funktionsweise eines technischen Gerätes wird verständlich, wenn man es in Aktion sieht und womöglich auch noch sieht, was im Inneren passiert.

- Virtuelle Welten: Chatrooms und Diskussionsforen sind sehr beliebt bei vielen Surfern. 3D-Welten erlauben es den Benutzern, noch tiefer einzusteigen. Sie können auch ein natürlicheres Gesellschaftsverhalten simulieren, indem man das Gesicht des Gesprächspartners sehen kann oder indem eine optische Bildung von Gesprächsgruppen ermöglicht wird.
- Geringerer Speicherplatz: Dieser Grund mag zunächst verblüffen. Normalerweise assoziiert man mit 3D-Grafik lange Ladezeiten. Das liegt allerdings eher an der falschen oder eingeschränkten Verwendung der Möglichkeiten. Auf den meisten Webseiten werden Grafiken für abgerundete Buttons, Flächen, die einen Schatten werfen, Metalleffekte und dergleichen verwendet. Das sind alles Simulationen von Dreidimensionalität. Würde man statt dieser mehrere Kilobyte großen Grafiken direkt einen 3D-Renderer verwenden, fielen beispielsweise für eine Fläche, die einen Schatten wirft, nur die Angaben für die vier Eckpunkte der Fläche, die Farbe der Fläche und die Position einer Lichtquelle an. Das ist natürlich etwas vereinfacht dargestellt, aber das Prinzip wird klar. Noch deutlicher wird der Unterschied, wenn Animationen verwendet werden. Normalerweise wird dies mit Animated-GIFs realisiert. Mit einem 3D-Renderer sähe es allerdings besser aus und würde weniger Speicherplatz benötigen.
- Kollaboration: Verteiltes Arbeiten an dreidimensionalen Konstruktionen wird erleichtert. Beispielsweise kann ein Architekt seinen Kunden auch zu Hause einen Rundgang durch das zukünftige Eigenheim ermöglichen.
- Avatare: Auf manchen Websites wird heute schon eine persönlichere Ansprache versucht, indem man dem Besucher einen Avatar, einen simulierten Gastgeber zur Seite stellt.
- Eye-Catcher: Manchmal ist es schlicht und einfach auch nur notwendig, aufzufallen. Normale Flasheffekte reißen heute keinen mehr vom Hocker; gut gemachte 3D-Animationen schon.

Wie oben schon erwähnt, ist zwar die Hardware bereit und größtenteils vorhanden, auf der Softwareseite sieht das allerdings schon anders aus. Aus diesem Grund ist praktisch für alle 3D-Darstellungen auf Webseiten ein Plugin notwendig.

Neben den speziellen 3D-Plugins werden allerdings zwei Kapitel dieses Buches zeigen, wie man auch mit Flash oder gar DHTML 3D-Animationen erzeugen kann.

Verwendung des Buches

Dieses Buch müssen Sie nicht von vorne bis hinten Seite für Seite durchlesen. Je nachdem, warum Sie dieses Buch gekauft haben, interessieren Sie sich womöglich nur für einige der Kapitel. Deshalb sollen die folgenden Abschnitte erläutern, was Sie wo finden.

Das Grundlagen-Kapitel beschreibt die allgemeinen Dinge, die Sie für die Verwendung und Erstellung von 3D-Animationen kennen müssen. Dies umfasst neben Begriffsklärungen und Beschreibung der prinzipiellen Techniken auch die mathematischen Grundlagen, die man für eine eigenständige Weiterentwicklung der Beispiele benötigt.

In Kapitel 2 werden die verschiedenen Technologien, die es gibt, um 3D-Darstellungen auf Webseiten zu ermöglichen, aufgelistet und miteinander verglichen.

Das Kapitel 3 beschäftigt sich mit den Möglichkeiten, die man ohne jegliches Plugin hat. Nun bestehen Webseiten in der Regel ja aus HTML, gewürzt mit ein wenig CSS (Cascading Style Sheets) und mehr oder weniger JavaScript. HTML ist dabei nur eine Auszeichnungssprache zur Darstellung von formatierten Dokumenten. Durch die Erweiterung um CSS ab den Browsern der Version 4 (Netscape und MS Internet Explorer) haben HTML-Dokumente erstmals die Möglichkeit bekommen zumindest quasi-3D-Darstellungen zu erlauben – indem man HTML-Elemente übereinander schichten konnte. Das alles klingt erst mal nicht wirklich nach dem, was Sie beim Kauf dieses Buches vermutlich erwartet haben. Aber dies sind die Grundlagen, die Bausteine, mit denen ich beginnen möchte, ein dreidimensionales Erlebnis für den Besucher einer Website zu schaffen.

In den darauf folgenden Kapiteln finden Sie einzelne Technologien wie Flash, Director Shockwave und Java näher beschrieben. Die Kapitel enthalten zahlreiche Beispiele, mit denen Sie die Technologien direkt ausprobieren können. Die Beispiele sind genau erklärt, erfordern aber ein Grundverständnis in der jeweiligen Programmiersprache. Wenn Sie sich aber generell mit irgendeiner Programmiersprache gut auskennen, wird es Ihnen nicht schwer fallen, die Beispiele zumindest zu verstehen.

Da manche der Bilder nur in Farbe ihren Reiz entfalten und alle Informationen zeigen, finden Sie in diesem Buch einen Farbteil, der ausgewählte Bilder der einzelnen Kapitel noch mal in Farbe zeigt.

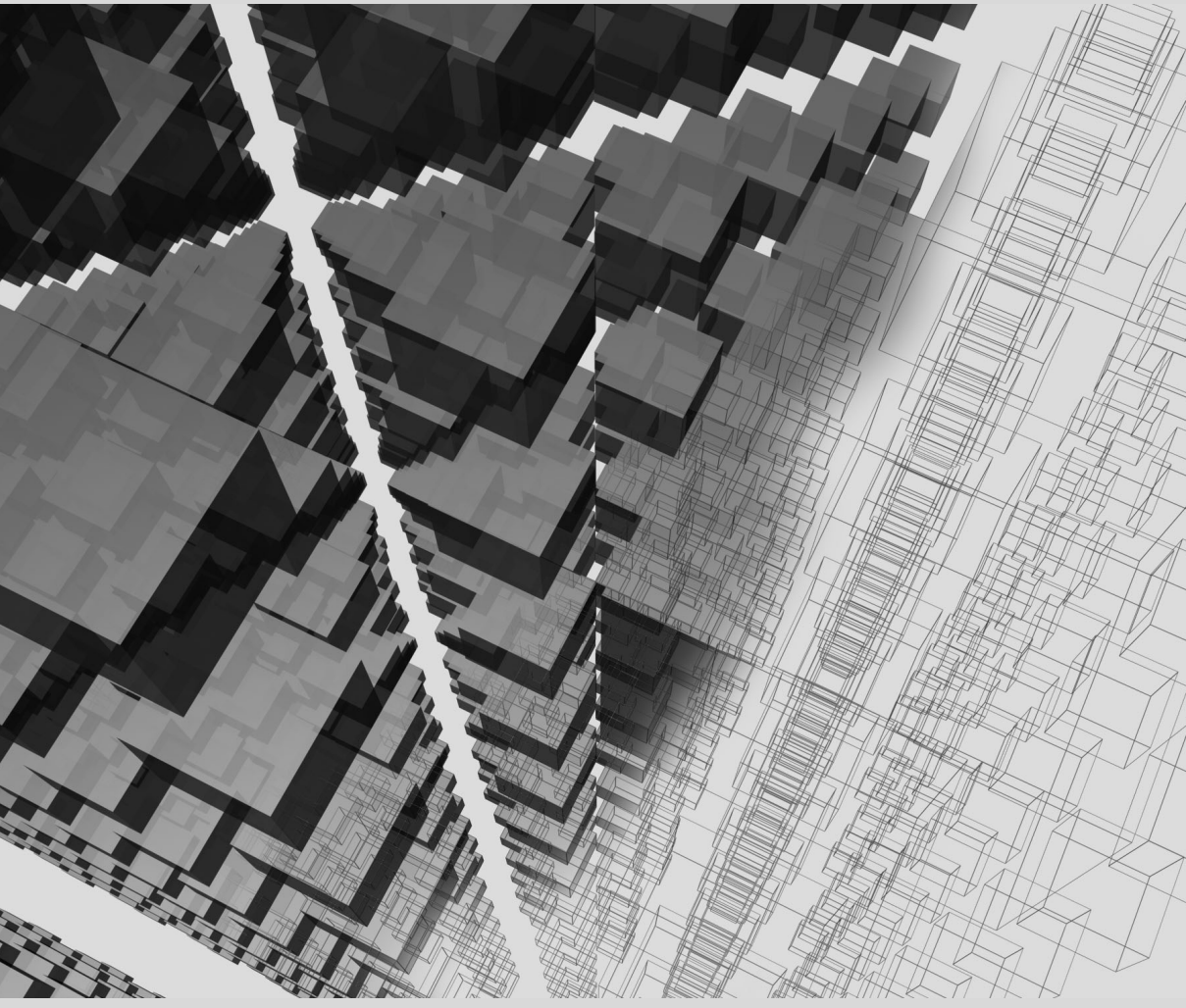
CD-Inhalt Auf der Begleit-CD finden Sie jeweils den Quellcode zu den Beispielen, damit Sie nicht alles abzutippen brauchen. Der Inhalt ist wie folgt gegliedert:

- Beispiele: Dieses Verzeichnis enthält die Beispielprogramme zu den verschiedenen Kapiteln. Zum Betrachten der fertigen Beispiele auf den HTML-Seiten benötigen Sie nur das entsprechende Plugin. Für die Quelldateien ist in der Regel aber das entsprechende Softwarepaket notwendig.
  - DHTML: Da bei den DHTML-Beispielen mehrere Dateien zusammengehören, befinden sich diese jeweils in einem Unterverzeichnis.
  - Flash: Hier sind alle Beispiele in einem Verzeichnis zusammengefasst. Die Quelldatei, Filmdatei und HTML-Datei haben bis auf die Endung jeweils den gleichen Namen.
  - Director: Hier sind alle Beispiele in einem Verzeichnis zusammengefasst. Die Quelldatei, Filmdatei und HTML-Datei haben bis auf die Endung jeweils den gleichen Namen. Beispiele zu Director Shockwave 8 haben den Präfix **dir8_** und Beispiele zu Director Shockwave 8.5 den Präfix **dir85_.**
  - Java: Dieses Verzeichnis enthält zwei Unterverzeichnisse für die Java3D- und Shout3D-Beispiele.
  - X3D: In diesem Verzeichnis finden Sie die X3D-Beispiele.
- Demos: Hier finden Sie Demoversionen von ausgewählter Software, die im Buch beschrieben wird.

Die Beispiele zum Ansehen und viele nützliche Links finden Sie auch auf meiner Website zu dem Buch *3d.franklamprecht.de*. Selbstverständlich sind Anregungen und Kommentare an *3d-buch@franklamprecht.de* oder *frank.lamprecht@addison-wesley.de* herzlich willkommen.

Und nun tauchen Sie mit ein in die Welt der dreidimensionalen Möglichkeiten.

Frank Lamprecht
April 2002



Grundlagen

Bevor ich die Benutzung von DHTML, Flash oder anderen Technologien für die Darstellung von dreidimensionalen Grafiken auf Webseiten erläutere, möchte ich einige grundlegende Prinzipien erklären, die für viele der später verwendeten Technologien gleich sein werden.

Dieses Kapitel ist nicht zwingend notwendig, um die Beispiele der kommenden Kapitel zu benutzen. Wenn Sie jedoch die dahinter liegenden Prinzipien verstehen wollen, um selbst Erweiterungen vorzunehmen und eigene Ideen umzusetzen, ist es sehr ratsam, die folgenden Grundlagen zu kennen.

1.1 Wie entsteht der dreidimensionale Eindruck?

Webseiten sind, da sie auf dem Computerbildschirm dargestellt werden prinzipiell erst mal zweidimensional, d.h. sie haben eine Breite und Höhe, aber keine Tiefe. Man kann aber trotzdem den Eindruck von Tiefe – von einer dritten Dimension – mit verschiedenen Mitteln erzeugen.

Der Mensch nimmt den Tiefeneindruck über unterschiedliche Methoden wahr. Die offensichtlichste Methode, nämlich dadurch, dass man mit den beiden Augen von unterschiedlichen Winkeln auf einen Gegenstand blickt und auf diese Weise einen Eindruck der Entfernung gewinnt, ist allerdings ohne erheblichen technischen Aufwand nicht zu bewerkstelligen für Webseiten. Deshalb müssen wir die anderen Methoden benutzen, die aber trotzdem den größten Teil bei der Wahrnehmung ausmachen:

Verdeckung, Licht und Schatten

Dadurch, dass ein Objekt ein anderes Objekt ganz oder teilweise verdeckt, kann man erkennen, welches näher am Betrachter ist als das andere.

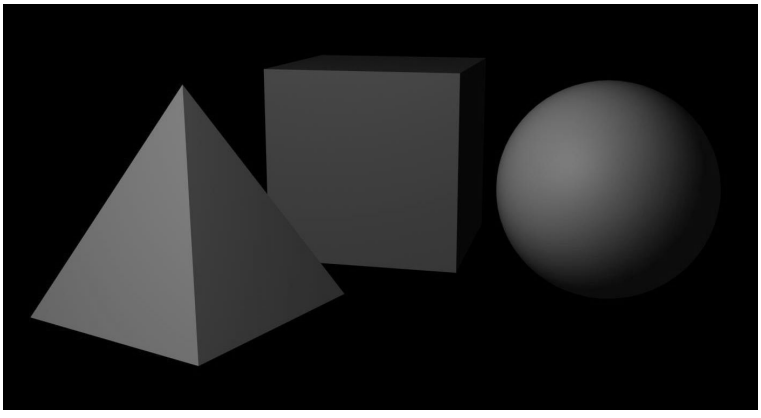


Abbildung 1.1: Einfache 3D-Szene

Sie sehen, dass man dieses recht einfache Mittel der dreidimensionalen Darstellung sogar in einem Buch verwenden kann und auch für Webseiten ist das natürlich sehr einfach möglich:

```
<img src="super3Dbild.jpg">
```

Das ist selbstverständlich nur ein einfaches Image-Tag, welches ein Bild in HTML einbindet. Aber so einfach wollen wir es uns ja nicht machen. Denn eines der wichtigsten Mittel zur dreidimensionalen Darstellung, um welches es auch im Weiteren gehen soll, ist die:

Bewegung im 3D-Raum

Erst, wenn sich Objekte bewegen, wird die Dreidimensionalität richtig sichtbar und wirkungsvoll. Das bezieht sich nicht nur auf Drehungen, bei denen sichtbar wird, dass ein Objekt auch eine Tiefe besitzt. Vielmehr wird die Dreidimensionalität schon durch die perspektivische Verzerrung sichtbar bei einer einfachen Verschiebung des Objektes.

Um eine Bewegung nun auf einem zweidimensionalen Medium wie dem Bildschirm zu simulieren, gibt es im Wesentlichen drei Prinzipien, die die nächsten Abschnitte erklären werden.

▶ Das Filmprinzip

Dieses Prinzip kennt jeder vom Daumenkino: Das schnelle Austauschen von einzelnen Bilder, so dass das träge menschliche Auge den Austausch nicht mitbekommt und stattdessen eine flüssige Bewegung wahrnimmt.

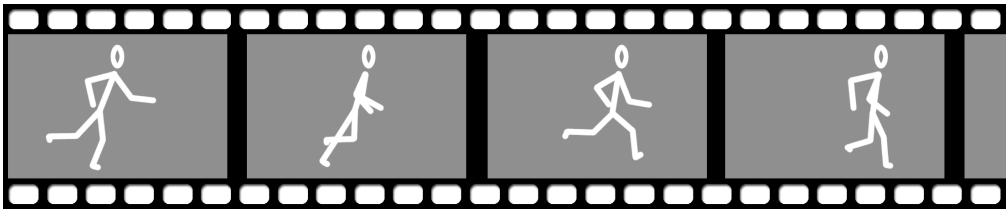


Abbildung 1.2: Eine „klassische" Filmsequenz

Dieses Prinzip wurde schon sehr früh in der Filmtechnik ausgenutzt und funktioniert auch sehr gut. Als Faustregel gilt, dass ab 24 Bildern in der Sekunde nicht mehr die Einzelbilder wahrgenommen werden, sondern eine flüssige Bewegung. In der Tricktechnik werden teilweise aber auch geringere Bildraten verwendet.

Anwendung des Film-Prinzips

Für unsere Zwecke werden wir das Prinzip ähnlich benutzen: Wir werden einzelne Bilder mit einer 3D-Grafiksoftware vorberechnen oder mit einer Kamera aufnehmen und diese Bilder an Ort und Stelle austauschen. Die Interaktion mit dem Benutzer bestimmt, wie die Bilder ausgetauscht werden, so dass die Sequenz von Bildern vorwärts oder rückwärts laufen kann. Auch kann eine Sequenz von Bildern verzweigen oder komplett ausgetauscht werden. Als Beispiel können Sie sich vorstellen, dass wir ein Auto präsentieren möchten. Wir berechnen nun im Voraus eine Sequenz, in der sich das Auto einmal um die eigene Achse dreht, und eine weitere, in der sich eine Tür öffnet. Der Benutzer kann nun bestimmen, aus welchem Winkel er das Auto betrachten möchte, oder er kann sich auch darum herumbewegen, indem die komplette Drehsequenz abgespielt wird. Außerdem kann er die Tür öffnen und für das Schließen wird die Sequenz für das Öffnen rückwärts abgespielt.

Für die Erzeugung der einzelnen Bilder für eine Sequenz kann ein Animationsprogramm wie 3D Studio Max oder auch einfach eine Video- bzw. Digitalkamera benutzt werden. Für eines der Beispiele in Kapitel 3 hatte ich etwa eine normale Digitalkamera verwendet und damit eine Sequenz von 16 Bildern einzeln aufgenommen.

Das Sprite-Prinzip

Als Sprites bezeichnet man einzelne Grafikobjekte, die sich praktisch autonom bewegen. Die Sprites sind in der Regel auch vorberechnete Bilder, die dann je nach Position im 3D-Raum auf dem Bildschirm platziert werden. Abhängig von der Position im 3D-Raum kann das Bild des Sprites auch ausgewechselt werden. Zum Beispiel muss die Größe des Bildes je nach Tiefe im Raum angepasst werden. Sollte das dargestellte Objekt nicht immer die gleiche Form haben aus verschiedenen Perspektiven, muss das Bild auch dafür angepasst werden (siehe Abbildung 1.3). Kugeln sind aus diesem Grund ein recht dankbares Objekt für eine allgemeine winkelunabhängige Darstellung.

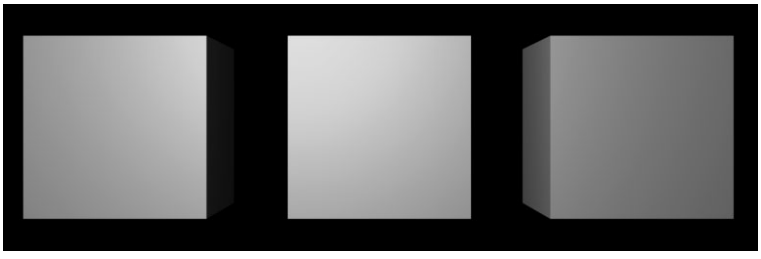


Abbildung 1.3: Unterschiedliche Ansichten eines Würfels an verschiedenen Positionen

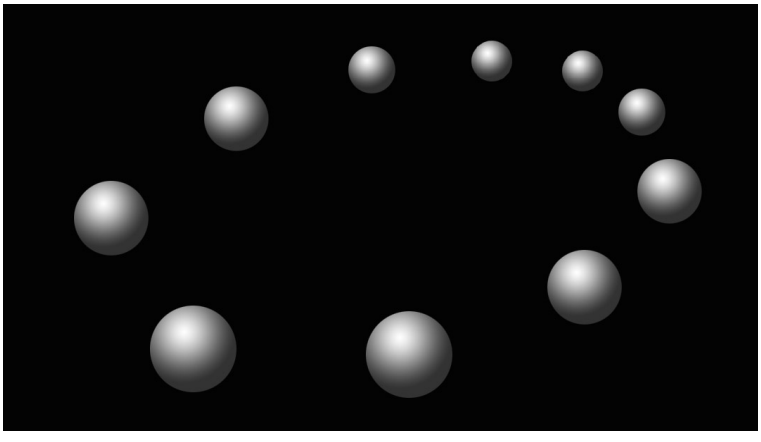


Abbildung 1.4: Kugelförmige Sprites

Wie genau die Transformation der Position im 3D-Raum in die Bildschirmfläche funktioniert, erläutert der nächste Abschnitt.

Teilweise werden Sie dieser Technik auch unter dem Namen „Billboard“ begegnen, da das Bild des Sprites immer dem Betrachter zugewandt ist und sich nie mitdreht, sondern nur Größe und Position verändert.

Eine Besonderheit, die man hierbei allerdings beachten muss, ist, dass die Sprites gegebenenfalls der Tiefe nach sortiert und die am weitesten weg liegenden zuerst gezeichnet werden.

▸ „The real thing“: Echte 3D-Darstellung

Die beiden vorhergehenden Prinzipien beruhen jeweils auf teilweise vorberechneten Bildelementen. Viel mehr ist zum Beispiel mit DHTML auch nicht möglich. Mit der Hilfe eines Plugins jedoch kann eine 3D-Szene inklusive Animation auch komplett zur Laufzeit berechnet werden. Dies ist z.B. möglich mit Java oder Shockwave und natürlich den spezialisierten Plugins zur 3D-Darstellung.

Dafür ist es dann notwendig, dass man die Szenerie in einer entsprechenden Form beschreibt. Die Darstellung erfordert allerdings eine Vielzahl an Berechnungen und Transformationen. Auch wenn ein Großteil der Berechnungen in der Regel von dem jeweiligen Plugin übernommen wird, ist es jedoch sinnvoll, zu wissen, was da „tief drinnen“ vor sich geht. Dieses Wissen kann helfen, Optimierungen vorzunehmen, die aufgrund beschränkter Übertragungs- und Rechnerleistungen notwendig sind.

Die Beschreibung der Szenerie umfasst in der Regel die Definition der Objektoberflächen, angegeben durch Quadrate und Dreiecke und die Eckpunkte, durch die sie bestimmt sind. Weiterhin muss dann noch angegeben sein, wie die Oberflächen aussehen und sich bei Lichteinfall verhalten sollen, ob sie spiegelnd sind, glänzend, matt oder welche Grundfarbe sie haben.

Die Software, die nun all diese Angaben benutzt, um damit ein Bild auf dem Bildschirm zu berechnen, wird Renderer genannt. Wie so ein Renderer arbeitet, zeigen die nächsten beiden Abschnitte.

1.2 Die Mathematik

Viele Leser werden vermutlich schon vom Titel dieses Abschnittes abgeschreckt sein. Ich möchte die folgenden Seiten auch gar nicht beschönigen. Wenn Sie mit Mathematik schon zu Schulzeiten auf Kriegsfuß standen, wird das Folgende kein schöner Anblick werden. Doch auch wenn es oft als „trockene“ Mathematik bezeichnet wird, bildet es doch die anschauliche Grundlage für faszinierende Bilder.

Wagen Sie es also ruhig, mit einzusteigen, und versuchen Sie zu verstehen, was es mit der ganzen Sache auf sich hat und wo die kleinen Vektoren herkommen.

An wichtiger Mathematik für den Bereich 3D-Grafik gibt es vier Themen, die hier kurz besprochen werden sollen:

- Vektoren und Matrizen: Das sind die grundlegenden Arbeitsmittel für alles weitere. Dreidimensionale Objekte werden in der Regel über markante Eckpunkte und durch diese Eckpunkte definierte Flächen beschrieben. Runde Oberflächen werden damit einfach angenähert. Vektoren dienen nun dazu, die Eckpunkte zu beschreiben, und Matrizen dazu, die Eckpunkte zu verändern.
- Projektion: Wenn wir irgendwelche Objekte im 3D-Raum beschreiben, müssen diese irgendwann auf unserem flachen Bildschirm landen.
- Transformationen: Da wir ja Bewegung haben wollen im 3D-Raum, müssen wir wissen, wie man solche Transformationen durchführt.
- Realismus: Das Ganze soll ja nicht wirklich nach Vektoren und Matrizen aussehen, deshalb müssen wir uns damit beschäftigen, wie es realistischer wirken kann.

Vektoren und Matrizen

Vektoren dienen der Positionsbestimmung

Vektoren sind nichts weiter als eine Liste von Zahlen. Für den dreidimensionalen Raum benutzt man dreidimensionale Vektoren, die dann die Koordinaten eines Punktes im Raum enthalten. In Abbildung 1.5 ist ein Beispielvektor zu sehen, der folgendermaßen geschrieben würde:

$$V = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

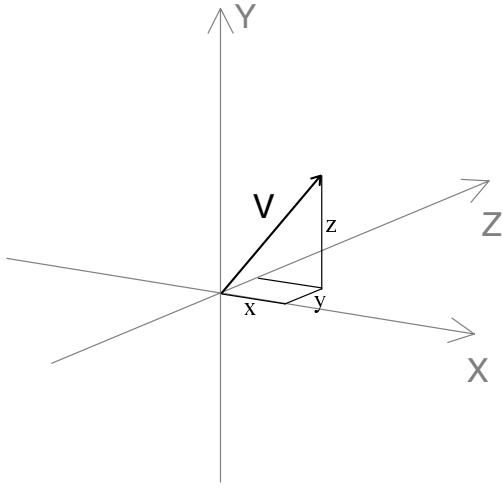


Abbildung 1.5: Ein Vektor im 3D-Raum

Rechnen mit Vektoren

Mit Vektoren kann man auch rechnen. Als Grundlage sollen zwei Vektoren dienen, Vektor

$$a = \begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \text{ und Vektor } b = \begin{pmatrix} bx \\ by \\ bz \end{pmatrix}$$

- Addition: Man kann zwei Vektoren a und b addieren, indem man die einzelnen korrespondierenden Werte miteinander addiert. Das Ergebnis kann man sich veranschaulichen, indem man den zu addierenden Vektor b einfach mit seinem Anfang an die Spitze des ersten Vektors a verschiebt. Der resultierende Vektor vom Anfang von a bis zur Spitze von b ist das Ergebnis.

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

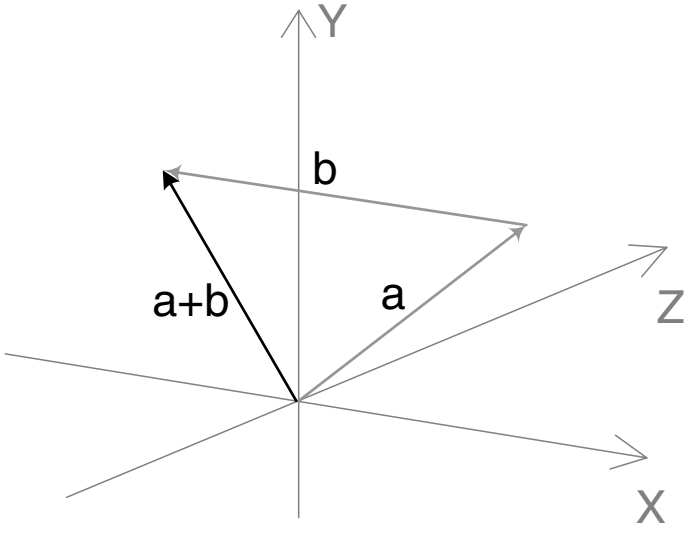


Abbildung 1.6: Vektoraddition

Die Subtraktion a-b ist genauso einfach. Allerdings denkt man sie sich besser als a+(-b). Das bedeutet, man dreht den Vektor b um und addiert ihn dann mit a.

- Skalierung: Man kann einen Vektor verkürzen oder verlängern, indem man die einzelnen Komponenten mit einem Wert s multipliziert.

$$s * \begin{pmatrix} ax \\ ay \\ az \end{pmatrix} = \begin{pmatrix} s * ax \\ s * ay \\ s * az \end{pmatrix}$$

  Ein Wert von 1 für s lässt den Vektor unverändert, ein Wert kleiner 1 verkürzt ihn und ein Wert größer 1 verlängert ihn.
- Betrag/Länge: Die Länge eines Vektors a bestimmt man mit der Formel

$$\left| \begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \right| = \sqrt{ax * ax + ay * ay + az * az}$$

  Mit dem Wissen, wie lang ein Vektor ist, kann man diesen auch **normalisieren**. Das bedeutet, man skaliert den Vektor so, dass er die Länge 1 hat. Dieser Normalenvektor ist wichtig, damit manche Berechnungen, bei denen es nur auf die Richtung des Vektors ankommt, korrekt durchgeführt werden können.
- Multiplikation: Man kann zwei Vektoren a und b miteinander multiplizieren, indem man die einzelnen Komponenten miteinander multipliziert.

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

  Das Besondere an dieser Multiplikation ist, dass das Multiplikationsergebnis gleich

$$\left| \begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \right| * \left| \begin{pmatrix} bx \\ by \\ bz \end{pmatrix} \right| \cos(\alpha)$$

  ist, wobei Alpha der Winkel zwischen den beiden Vektoren ist. Man kann damit also den Winkel zwischen zwei Vektoren bestimmen.

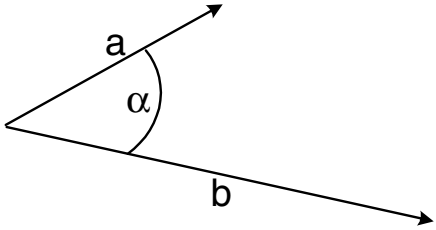


Abbildung 1.7: Der Winkel zwischen zwei Vektoren

- Vektorkreuzprodukt: Diese besondere Art zwei Vektoren miteinander zu multiplizieren erzeugt einen Vektor, der senkrecht auf der von a und b aufgespannten Ebene steht.

$$\begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \times \begin{pmatrix} bx \\ by \\ bz \end{pmatrix} = \begin{pmatrix} ay * bz - az * by \\ az * bx - ax * bz \\ ax * by - ay * bx \end{pmatrix}$$

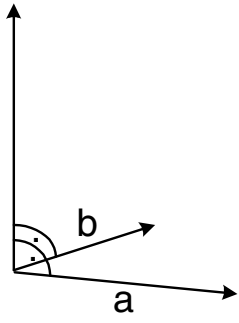


Abbildung 1.8: Vektorkreuzprodukt

Matrizen verändern Vektoren

Nachdem wir nun die Vektoren betrachtet haben, wenden wir uns den Matrizen zu. Matrizen sind dazu da, um Vektoren zu verändern. Mit einer passenden Matrix kann man einen Vektor verschieben, drehen oder skalieren. Bevor wir allerdings dazu kommen, wie derartige Matrizen aussehen, müssen wir unsere Vektoren noch etwas erweitern, und zwar um eine vierte Komponente. Diese vierte Komponente ist vereinfacht gesagt nur eine Hilfskonstruktion, um mit den existierenden Regeln der Mathematik die oben genannten Transformationen durch Matrizen darstellen zu können. Die um die vierte Komponente erweiterte Darstellung nennt man **homogene Koordinaten**. In der Regel wird diese vierte Komponente immer auf 1 gesetzt sein und hat auch für die Anschauung keine weitere Bedeutung.

Wie funktioniert nun die Veränderung eines Vektors durch eine Matrix? Die Multiplikation einer Matrix M mit einem Vektor V ergibt wieder einen Vektor, der folgendermaßen aussieht:

$$\begin{pmatrix} a1 & b1 & c1 & d1 \\ a2 & b2 & c2 & d2 \\ a3 & b3 & c3 & d3 \\ a4 & b4 & c4 & d4 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} a1*x+b1*y+c1*z+d1*w \\ a2*x+b2*y+c2*z+d2*w \\ a3*x+b3*y+c3*z+d3*w \\ a4*x+b4*y+c4*z+d4*w \end{pmatrix}$$

Hier kann man jetzt auch sehen, dass aufgrund dieser Rechenvorschrift die homogenen Koordinaten zum Beispiel für eine Verschiebung notwendig sind. Möchte man eine Matrix bilden, die einen Vektor um fünf Einheiten auf der x-Achse verschiebt, geht dies nur über diese vierte Komponente, die eine 1 ist.

$$\begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 4 \\ 1 \\ 7 \\ 1 \end{pmatrix} = \begin{pmatrix} 9 \\ 1 \\ 7 \\ 1 \end{pmatrix}$$

Wie die entsprechenden Matrizen für die verschiedenen Transformationen genau aussehen, folgt weiter unten.

Matrizen sind wie Bausteine, die man zusammensetzen kann.

Matrizen sind ja schön und gut, aber wozu muss man sich das Ganze so kompliziert machen und ändert die Koordinaten nicht einfach direkt über die einzelnen Berechnungen? Matrizen haben noch einen weiteren Vorteil: Man kann sie kombinieren, bzw. miteinander multiplizieren. Man kann also eine Drehungsmatrix mit einer Verschiebungsmatrix kombinieren und erhält wieder eine einzelne Matrix, die angewendet auf einen Vektor diesen zuerst dreht und dann verschiebt. Da in der Regel Objekte im 3D-Raum aus einer Vielzahl von Punkten definiert sind, muss die gleiche Folge von Berechnungen (z.B. erst eine Drehung, dann eine Verschiebung und schließlich eine Projektion) auf eine große Anzahl von Vektoren angewendet werden. Wenn man die Transformationen zuerst zu einer einzigen Matrix zusammenfasst, spart man sich unter Umständen viel Rechenzeit.

Viele Grafikbibliotheken (wie zum Beispiel OpenGL oder Java3D) basieren auf Transformationsmatrizen. Es ist also durchaus sinnvoll, dieses Prinzip zu verstehen.

Transformationen (Drehungen und mehr)

Nachdem wir gesehen haben, warum Matrizen so nützlich sind, kommen wir nun zum konkreten Aussehen der gebräuchlichsten Matrizen.

Einheitsmatrix (Identität): Wenn man wie oben beschrieben Transformationsketten bildet, tut man dies, indem man zu der bereits bestehenden Transformationsmatrix die gewünschte zusätzliche Transformation dazumultipliziert. Nur muss man natürlich mit einer Matrix anfangen, die idealerweise noch gar keine Transformation bewirkt. Genau dafür ist die Einheitsmatrix da. Wenn man diese mit einer anderen Matrix oder einem Vektor multipliziert, wird die Matrix oder der Vektor nicht geändert.

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

Verschiebung (Translation): Man kann nicht wirklich einen Vektor verschieben, da ein Vektor immer im Nullpunkt startet und zu dem Punkt an den angegebenen Koordinaten geht. Die Verschiebung bezieht sich also nur auf den Punkt an den Koordinaten und ist im Wesentlichen das Gleiche wie eine Vektoraddition.

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

Skalierung: Eine Skalierung verschiebt im Prinzip den durch den Vektor definierten Punkt auf der Geraden, die durch den Punkt und den Koordinatenursprung geht.

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

Drehung (Rotation): Eine Drehung ist immer eine Drehung eines Punktes um einen bestimmten Winkel um den Ursprung, d.h. eine Rotationsmatrix beschreibt auch immer nur die Drehung um einen bestimmten Winkel. In den unten gezeigten Matrizen sind noch Winkelfunktionen (Sinus und Kosinus) enthalten, die für einen bestimmten Winkel einen konkreten Wert ergeben, der dann an der entsprechenden Stelle in der Matrix steht. Da die Winkelfunktionen auf Computern in der Regel die Winkelangabe im Bogenmaß verlangen, muss man bei einer konkreten Implementierung aufpassen, wie man den Winkel tatsächlich angibt.

Bei Rotationen ist wichtig, dass man die Reihenfolge der Transformationen beachtet. In Abbildung 1.9 wird der Vektor zuerst verschoben und anschließend um die y-Achse rotiert. In Abbildung 1.10 wird er dagegen zuerst rotiert und danach verschoben.

Abbildung 1.9: Erst eine Translation, dann eine Rotation

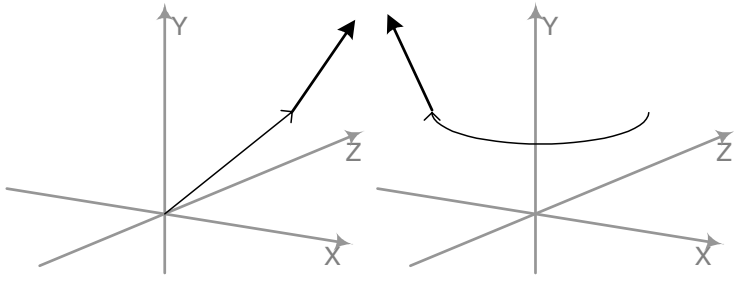
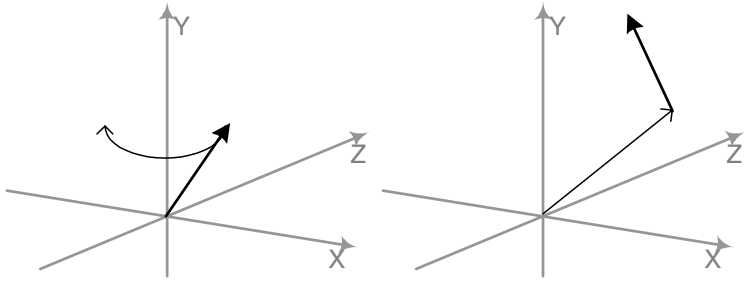


Abbildung 1.10: Erst eine Rotation, dann eine Translation



Rotation um die x-Achse:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(wx) & -\sin(wx) & 0 \\ 0 & \sin(wx) & \cos(wx) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um die y-Achse:

$$M = \begin{pmatrix} \cos(wy) & 0 & \sin(wy) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(wy) & 0 & \cos(wy) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um die z-Achse:

$$M = \begin{pmatrix} \cos(wz) & -\sin(wz) & 0 & 0 \\ \sin(wz) & \cos(wz) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Projektion (von 3D zu 2D)

Die Projektion ist die letzte Transformation, die man für jeden Punkt durchführen muss, damit man weiß, an welcher Stelle auf dem Bildschirm der Punkt erscheinen soll. Die Punkte, mit denen wir arbeiten, befinden sich alle an einer beliebigen Position im 3D-Raum, der theoretisch unendlich groß ist (je nachdem welches Datenformat man für die Koordinatenspeicherung wählt). Irgendwie müssen diese Koordinaten aber letztendlich auf unserem Bildschirm landen, der zum Beispiel nur 800 Pixel in der Breite und 600 Pixel in der Höhe anzeigen kann. Genau für diesen Zweck führt man eine Projektion durch.

Für die Projektion stellen wir uns eine virtuelle Kamera vor, die sich ebenfalls im 3D-Raum befindet und in eine bestimmte Richtung blickt. Man kann sich den Bildschirm jetzt wie eine Fensterfläche vorstellen. Durch dieses Fenster blickt die Kamera.

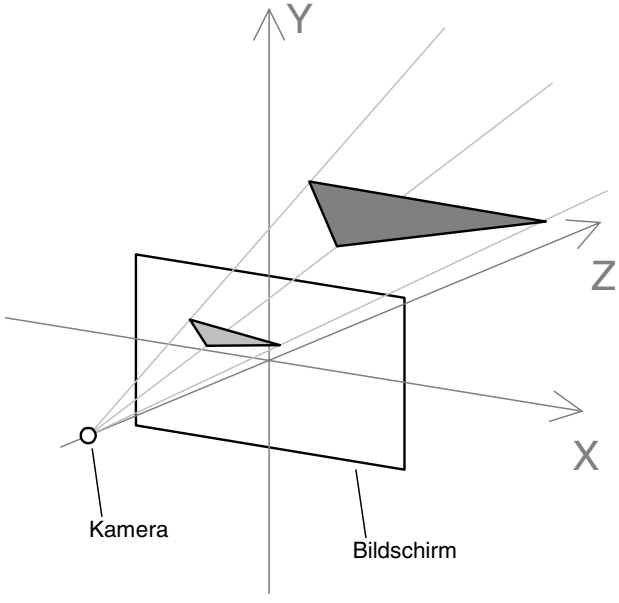


Abbildung 1.11: Die Projektion eines Dreiecks in die Bildschirmebene

Je nachdem, wie die Kamera im Raum platziert ist und in welche Richtung sie blickt, muss man für diese Projektion noch einigen Rotationen und Translationen durchführen. In vielen Fällen reicht es jedoch aus, wenn man für die Kameraposition vereinfachende Annahmen trifft. Dann werden die entsprechenden Berechnungen schon viel übersichtlicher. Wir wollen also für so eine vereinfachte Projektion annehmen, dass sich die Kamera an einer bestimmten Position im Raum befindet und direkt in Richtung der Projektionsebene (des Fensters) blickt. Für einen beliebigen Punkt im 3D-Raum `(x,y,z)` berechnen sich dann die entsprechenden Bildschirmkoordinaten wie folgt:

```
bildschirm_x = (z * kamera_x - x * kamera_z) /
               (z - kamera_z) + bildschirm_offset_x;

bildschirm_y = (z * kamera_y - y * kamera_z) /
               (z - kamera_z) + bildschirm_offset_y;
```

In den Berechnungen werden außer den Koordinaten des zu projizierenden Punktes (x, y, z) noch die Koordinaten der virtuellen Kamera (kamera_x, kamera_y, kamera_z) und ein Versatz (bildschirm_offset_x, bildschirm_offset_y) benutzt. Der Versatz ist deshalb wichtig, weil der Koordinatenursprung des 3D-Raumes in der Mitte liegen soll, der Ursprung der Bildschirmkoordinaten aber in der linken oberen Ecke liegt. Deshalb werden bei einer Bildschirmauflösung von beispielsweise 800x600 Pixeln zu der x-Koordinate 400 und zur y-Koordinate 300 Pixel hinzu addiert. Die Berechnung würde sich noch weiter vereinfachen, indem man den Betrachter direkt auf die z-Achse setzen würde.

Ein Begriff, der in diesem Zusammenhang noch wichtig ist, ist das Clipping. Damit bezeichnet man das Abschneiden von Elementen, die man nicht sehen möchte oder auch gar nicht sehen kann, da sie außerhalb der Bildschirmfläche liegen. Durch dieses Abschneiden kann man sich unter Umständen viel Rechenzeit sparen für Elemente, die der Rechner dann nicht zeichnen muss. Außerdem entstehen dadurch teilweise auch interessante Effekte, wenn man alles auf der einen Seite einer Fläche abschneidet und so in ein Objekt hineinsehen kann.

1.3 Realismus

Im Abschnitt Mathematik haben wir uns damit beschäftigt, wie man Punkte im 3D-Raum dreht, verschiebt und dergleichen und schließlich auf dem Bildschirm darstellt. Eine Punktewolke auf unserem Bildschirm wirkt aber noch nicht allzu ansprechend. Lassen Sie uns also schrittweise mehr Realismus einführen:

Drahtgittermodell (Wireframe Model)

Indem man bestimmte Punkte miteinander durch Linien verbindet, kann man die Form eines Objektes schon recht gut erkennen. In der Praxis hat man für ein Objekt eine Liste mit den enthaltenen Eckpunkten. Für ein Drahtgittermodell erstellt man eine weitere Liste, die die Indizes der Eckpunkte enthält, die jeweils miteinander verbunden werden sollen.

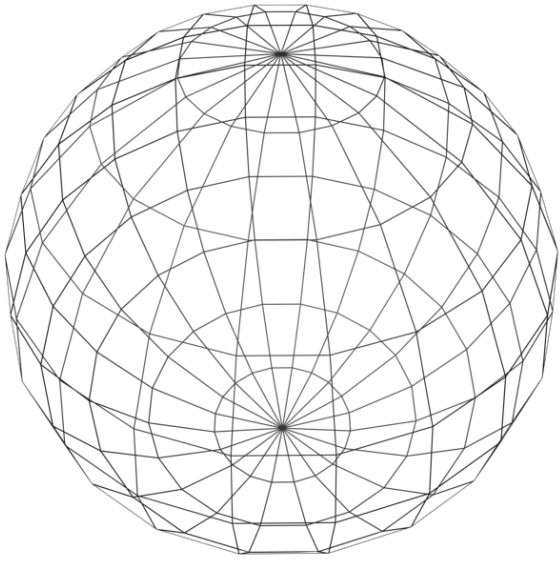


Abbildung 1.12: Drahtgittermodell

Für ein Drahtgittermodell müssen nur Linien gezeichnet werden. Das geht auf den meisten Rechnern sehr schnell.

Verdeckte Linien

Ein Drahtgittermodell kann schon recht reizvoll sein. In dem Beispiel mit der Kugel wollen wir diese aber nun mit einer Haut überziehen. Das bedeutet, dass man die Linien auf der Rückseite der Kugel nicht mehr sehen sollte. Um zu ermitteln, welche Linien verdeckt sind, brauchen wir Informationen, wo sich welche Flächen befinden. Zu diesem Zweck legt man eine weitere Liste mit Flächenangaben an, wobei für jede Fläche gespeichert wird, durch welche Eckpunkte sie definiert wird. In Abbildung 1.13 sind Flächen mit drei und mit vier Ecken zu sehen. Für die meisten in Echtzeit berechneten Flächen werden allerdings nur drei Eckpunkte benutzt, da diese leichter zu verarbeiten und schneller darzustellen sind.

Eine Fläche mit vier Eckpunkten würde also einfach in zwei Dreiecke zerlegt werden.

Für die Darstellung mit verdeckten Linien sind schon erheblich mehr Rechenschritte notwendig, da nun geprüft werden muss, welche Flächen näher beim Betrachter liegen und somit über den weiter weg liegenden gezeichnet werden müssen.

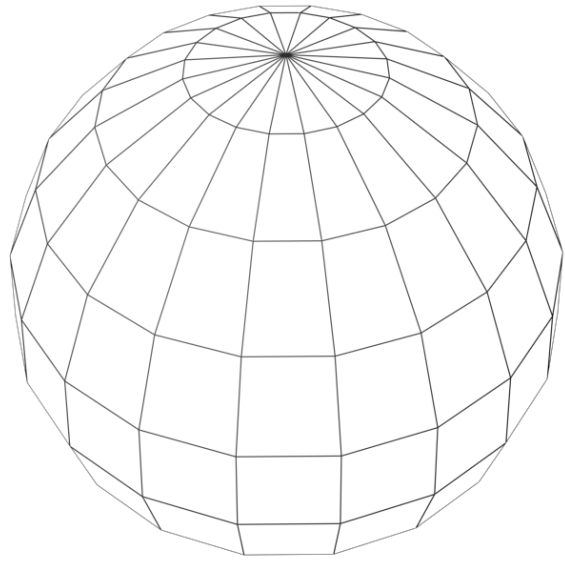


Abbildung 1.13: Drahtgittermodell mit verdeckten Linien

Es gibt drei Methoden, mit denen man diese Darstellung erreichen kann:

- Rückflächenunterdrückung (back face culling): Eine Fläche hat immer eine Vorder- und eine Rückseite. Man kann diese unterscheiden durch die Anordnung der Eckpunkte. Die Eckpunkte werden in einer Liste gespeichert und haben dadurch eine bestimmte Reihenfolge. Wenn man nun auf die Fläche schaut, und die Eckpunkte sind entsprechend der Liste im Uhrzeigersinn sortiert, dann hat man die Vorderseite vor sich. Wenn sie gegen den Uhrzeigersinn sortiert sind, hat man die Rückseite vor sich. Diese Definition ist allerdings nicht festgelegt, sondern hängt von der jeweiligen Programmierung ab. Normalerweise kann man angeben, ob bei den Modellangaben die Eckpunkte für die Vorderseite im Uhrzeigersinn (CW: Clockwise) oder gegen den Uhrzeigersinn (CCW: Counter Clockwise) sortiert sind. Bevor man nun eine Fläche, bzw. deren Kanten auf dem Bildschirm zeichnet, kann man überprüfen, ob nach der Projektion die Vorder- oder die Rückseite zu sehen wäre und entsprechend das Zeichnen unterlassen. Diese Methode funktioniert problemlos bei vollständig konvexen Körpern, wie zum Beispiel einer Kugel. Bei teilweise konkaven Körpern, wie zum Beispiel einem Hufei-

sen, funktioniert das Verfahren nicht mehr. Man kann damit aber auf jeden Fall Rechenzeit sparen, wenn man die Rückflächenunterdrückung auch bei den folgenden Methoden zuerst durchführt.

- Flächensortierung: Anstatt nur die Kanten einer Fläche zu zeichnen, kann man die Fläche auch ausfüllen und damit bereits gezeichnete Teile übermalen. Damit das korrekt funktioniert, muss man zunächst die Flächen der Tiefe nach sortieren und die hintersten zuerst zeichnen. Dieses Verfahren funktioniert auch für konkave Körper. Es funktioniert allerdings nicht mehr, wenn sich zwei Flächen A und B durchdringen. Dann liegt nämlich ein Teil von A vor B und ein Teil dahinter. Dafür benötigt man die nächste Methode.
- Tiefenpuffer: Bei dieser Methode wird für jeden Bildschirmpunkt noch ein weiterer Wert gespeichert, nämlich die Tiefe des darauf gezeichneten Punktes. Wenn man also eine Fläche auf dem Bildschirm zeichnet, setzt man nicht nur Punkte auf dem Bildschirm, sondern schreibt auch die Tiefe der einzelnen Punkte in den Tiefenpuffer. Genauer gesagt prüft man vor dem Setzen des Punktes erst mal, ob die Tiefe des zu zeichnenden Punktes näher am Betrachter liegt als die des bereits vorhandenen Punktes. Bei geringerer Tiefe überschreibt man den Punkt auf dem Bildschirm und die Angabe im Tiefenpuffer.

Für einfache, eigene 3D-Darstellungen kann man am besten die erste Methode verwenden. Die meisten professionellen Renderer verwenden allerdings einen Tiefenpuffer, da dieser sehr effizient und schnell arbeitet.

Schattierte Flächen

Diese Darstellungsform beruht auf der vorhergehenden. Nun müssen die Flächen (Dreiecke oder Vierecke), die gezeichnet werden, noch mit einer Farbe gefüllt werden. Damit in Abbildung 1.14 nicht nur eine flache Scheibe zu sehen ist, muss durch eine Variation der Modellfarbe für jede einzelne Seitenfläche der räumliche Eindruck erzeugt werden. Die Variation der Modellfarbe wird dadurch bestimmt, wie viel Licht auf die jeweilige Fläche triff. In dem Beispiel befindet sich eine imaginäre Lichtquelle im 3D-Raum links oben neben dem Betrachter. Eine Seitenfläche wird nun umso heller gezeichnet, je mehr das Licht senkrecht auf die Fläche fällt. Wenn der Einfallswinkel des Lichtes größer als 90° ist, fällt gar kein Licht mehr auf die Fläche und sie kann dunkel gezeichnet werden.

Abbildung 1.14: Gefärbte Seitenflächen

In Abbildung 1.15 sehen Sie, wie der Lichteinfallswinkel für eine Fläche bestimmt wird. Wie weiter oben schon erwähnt, ist der Normalenvektor so definiert, dass er eine Länge von 1 hat und auf der Vorderseite einer Fläche steht.

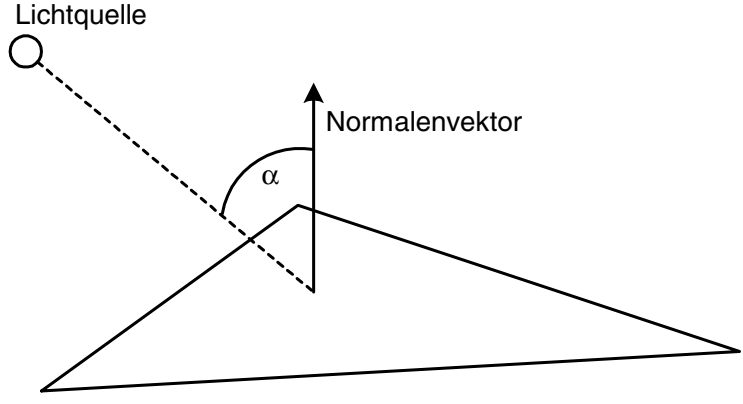


Abbildung 1.15: Lichteinfallswinkel

Spezielle Schattierungen

Um Rundungen schöner darzustellen und die Schattierung realistischer wirken zu lassen (siehe Abbildung 1.18) gibt es spezielle Methoden zur Schattierung der Seitenflächen. Die einfachsten davon sind bekannt unter dem Namen Gouraud- und Phong-Shading und benutzen die Tatsache, dass für jeden Eckpunkt ein eigener Normalenvektor definiert werden kann. Der Normalenvektor definiert die Ausrichtung der Fläche in dem jeweiligen Eckpunkt. Wenn man damit nun die Farbschattierung für jeden der drei Eckpunkte bestimmt hat, werden beim Gouraud-Shading alle übrigen Punktfarben des Dreiecks aus diesen drei Farben interpoliert (siehe Abbildung 1.16). Beim Phong-Shading werden nicht die Farben in den Eckpunkten interpoliert, sondern die Normalenvektoren in den Eckpunkten werden für jeden Dreieckspunkt interpoliert und die Farbschattierung damit bestimmt (siehe Abbildung 1.17). Mit dem Phong-Shading werden Lichtreflexe besser dargestellt.

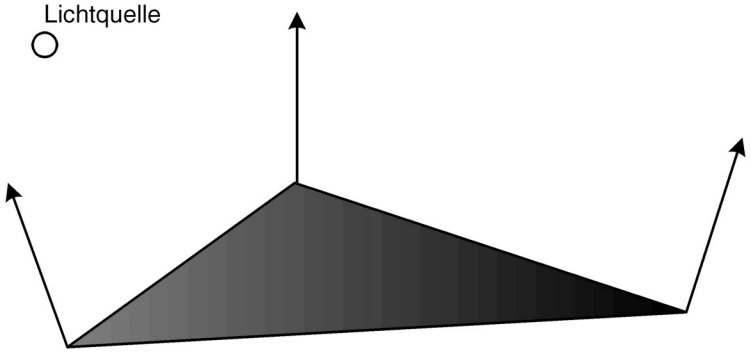


Abbildung 1.16: Gouraud-Shading

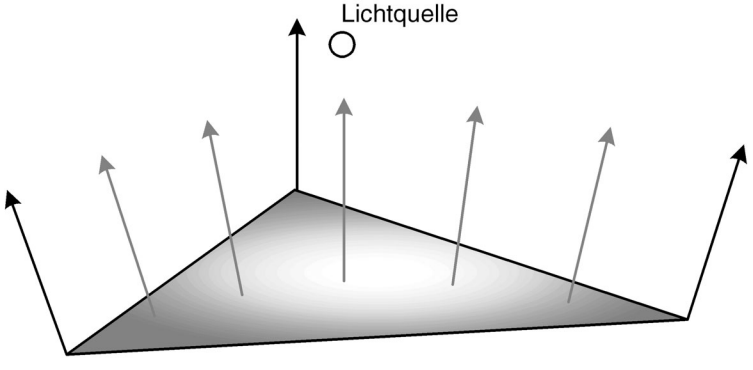


Abbildung 1.17: Phong-Shading

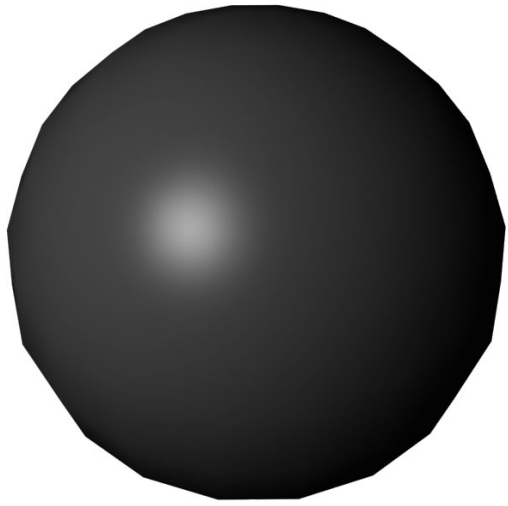


Abbildung 1.18: Schattierte Seitenflächen

Texturierte Flächen

Anstatt einer Fläche eine einheitliche Farbe zu geben, kann man ihr auch ein Bild zuweisen. Das Rechenintensive daran ist, die Bilder entsprechend perspektivisch zu verzerren, so dass sie anschließend noch gut aussehen. Dafür werden in der Regel Filtermethoden verwendet, die mehr oder weniger aufwändig sind.

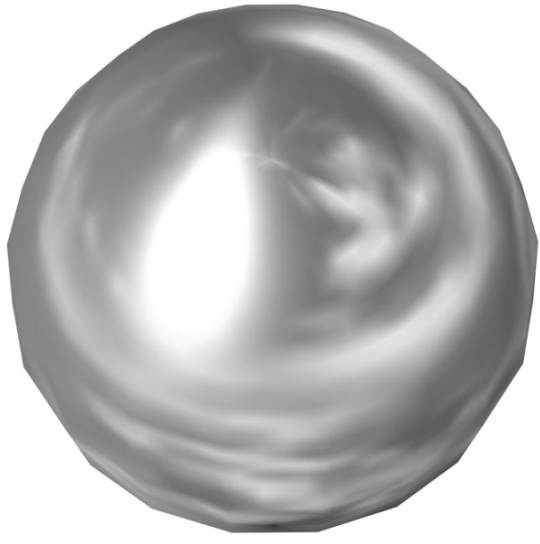


Abbildung 1.19: Texturierte Seitenflächen

Schattenwurf, Raytracing, Radiosity und dergleichen

Zu den Techniken, um noch mehr Realismus bei berechneten 3D-Szenen zu erlangen, gehören Schattenwurf, Raytracing und Radiosity. Diese Techniken sind allerdings so rechenaufwändig, dass sie zurzeit nicht in Echtzeit berechnet und dargestellt werden können. Diese Techniken können allenfalls für vorberechnete Bilder oder Filmsequenzen verwendet werden. Sie sollen hier aber nicht näher erläutert werden.

Spezielle Darstellungstechniken

Neben den zuvor beschriebenen prinzipiellen Darstellungsmöglichkeiten möchte ich noch ein paar besondere Techniken erläutern, die für die Darstellung von 3D-Grafik wichtig sind. An dieser Stelle soll es nicht darum gehen, wie diese Techniken genau funktionieren, aber wenn Sie sich mit 3D-Grafik beschäftigen, werden Ihnen diese Begriffe häufiger begegnen. Sie sollten also wissen, worum es dabei geht, damit Sie besser abschätzen können, ob dieses oder jenes Feature für Sie wichtig ist.

- **Antialiasing**

Jeder Computerbildschirm besteht aus einem Punkteraster, welches für die Darstellung von Grafiken benutzt wird. Die meisten Bildschirme sind dabei auf ein Raster von 1024 mal 768 Punkten eingestellt (laut frei zugänglicher Webstatistiken ca. 50% der Webbenutzer). Das bedeutet also, dass sich zum Beispiel bei einem Standard 17-Zoll-Monitor auf einer Breite von etwa 32,5 cm 1024 Pixel verteilen. Das sind pro Millimeter etwa 3 Pixel. Auch ohne mit der Nase ganz an den Bildschirm heranzugehen, kann man deshalb bei schrägen Linien Treppenstufen erkennen, wenn der Kontrast hoch genug ist, zum Beispiel bei einer schwarzen Linie auf weißem Grund (siehe Abbildung 1.20 links)

Abbildung 1.20: Eine Linie ohne (links) und mit Antialiasing (rechts)

Um diesem Problem entgegenzuwirken und die Darstellung besser aussehen zu lassen, benutzt man Antialiasing. Hierbei werden die Treppenstufen dadurch abgemildert, dass ein leichter Farbverlauf eingefügt wird. Diese Technik erzeugt erheblich besser aussehende Grafiken, ist aber sehr aufwändig zu berechnen. Deshalb verwenden viele Plugins kein Antialiasing oder schalten es nur bei still stehender Grafik ein und bei bewegter Grafik aus.

- **Multitexturing**

In der Regel haben 3D-Objekte nur eine Texturgrafik, mit der sie überzogen werden. Um das Aussehen aber realistischer zu gestalten kann man dem Objekt mehrere Texturebenen zuweisen, zum Beispiel eine halbdurchsichtige Grafik, die sich bei einer Rotation des Objektes nicht mitdreht. Dadurch wird eine spiegelnde Oberfläche simuliert.

- **Bumpmapping**

Wenn wir als Beispiel einen Würfel nehmen, der durch sechs Seitenflächen beschrieben ist, so sind diese Seitenflächen erst mal ganz glatt. Nun könnte man auf diese Seitenflächen Texturen setzen, die die einzelnen Punkte eines Spielwürfels darstellen. Mit Hilfe des Bumpmappings könnte man den einzelnen Punkten auch eine Vertiefung zuweisen, ohne dass man dafür die Geometrie- bzw. Modelldaten des Würfels verändern müsste.

- **Reflektionen**

Um Reflektionen sehr wirklichkeitsgetreu darzustellen, müsste man das Raytracing-Verfahren verwenden, bei welchem tatsächlich einzelne Lichtstrahlen berechnet werden. Dieses Verfahren dauert aber für einzelne Bilder viel zu lange, als dass man es für Animationen einsetzen könnte. Deshalb benutzt man Tricks, die Spiegelungen vortäuschen, damit aber sehr eindrucksvolle Effekte erzeugen. Eine Möglichkeit ist dabei, eine halbdurchsichtige Textur zu verwenden, die sich bei einer Drehung des Objektes nicht mitdreht. Dadurch wird eine Spiegelung der Umgebung auf dem Objekt simuliert.

- **Schatten**

Auch Schatten sollten, um wirklichkeitsgetreu zu sein, mit dem Raytracing-Verfahren berechnet werden. Man kann aber auch Schatten simulieren, indem man zum Beispiel entsprechend berechnete Texturen oder flache Objekte, d.h. Objekte ohne Tiefe, verwendet.

1.4 Grafik-Hardware

Während früher die Grafikkarten noch damit ausgelastet waren, die Grundfunktionen der täglich benötigten Arbeit wie Textausgaben, Linienzeichnen oder Inhalteverschieben zu verrichten, sind seit einigen Jahren immer mehr Funktionen in die Hardware gewandert, die man für 3D-Grafik benötigt. Seit etwa drei Jahren werden nur noch Grafikkarten produziert, die zumindest grundlegende Funktionen für 3D-Grafik mitbringen. Das bedeutet, man kann sich in dieser Hinsicht auf gewisse Standards verlassen.

Welche Aufgaben sind dies nun, die von der Hardware übernommen werden? Hier sind die Hardware-Hersteller von hinten her an das Problem herangegangen. Erst nachdem alle Punkte transformiert und projiziert worden sind, unterstützt die Hardware dabei, die Oberflächen auf den Bildschirm zu zeichnen. In der Regel sind das in Dreiecke unterteilte Oberflächen. Hierbei gibt es dann aber auch die Möglichkeit, Farbverläufe und vor allem Texturen für die Dreiecke anzugeben. Gerade bei den Texturen hilft die Hardware, indem diese korrekt perspektivisch verzerrt dargestellt werden und dabei durch Filterung immer noch gut aussehen. Für die Texturen ist es auch wichtig, wie viel RAM eine Grafikkarte besitzt, denn selbst wenn die Bilder auf der Festplatte wenig Speicherplatz benötigen, da sie per JPEG komprimiert sind, kann die Grafikkarte sie in der Regel nur unkomprimiert verwenden und speichern.

Erst seit etwa einem Jahr gibt es immer mehr Grafikkarten, die auch T&L (Transformation and Lighting) unterstützen, d.h. die Transformation von Vektoren und die Beleuchtungsberechnungen für die Oberflächendreiecke. Dadurch wird dem Hauptprozessor sehr viel Arbeit abgenommen, die dann spezialisierte Grafikprozessoren übernehmen. Insgesamt wird es dadurch möglich, komplexere Szene zu erstellen, d.h. Objekte mit sehr vielen Dreiecken und viele Lichtquellen für realistischeres Aussehen.

Auch wenn die Hardware schon sehr viel Unterstützung bietet, muss diese Hilfe natürlich noch genutzt werden durch die Software. Das kann auf den meisten Betriebssystemen über die Programmierschnittstellen OpenGL oder DirectX erfolgen. In der Übersicht im nächsten Kapitel ist angegeben, welche Techniken jeweils auf vorhandene Hardwareunterstützung zurückgreifen.

1.5 Einbindung von Plugins in HTML

In späteren Kapiteln werden wir uns intensiv mit einzelnen Plugins beschäftigen. Das sind Erweiterungen für den Webbrowser, die sich entweder auf Nachfrage beim Benutzer selbst installieren (wenn der Browser eine HTML-Seite lädt, auf der das Plugin verwendet wird) oder man installiert es per Hand. Nach der Installation können die entsprechenden Funktionen dann verwendet werden. Ein Beispiel für solch ein Plugin ist Flash von Macromedia oder Java.

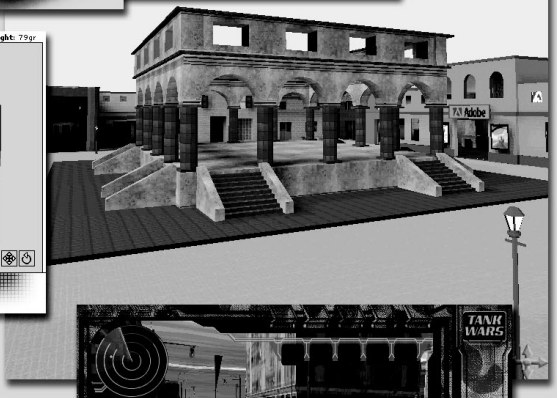
Dieser Abschnitt zeigt, wie man den entsprechenden Code für ein Plugin in HTML einbettet. Als Beispiel soll hier ein Flashfilm dienen.

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8
        -444553540000" codebase="http://download.
        macromedia.com/pub/shockwave/
        cabs/flash/swflash.cab#version=5,0,0,0"
        WIDTH=600 HEIGHT=400>
<PARAM NAME=movie VALUE="movie.swf">
<PARAM NAME=quality VALUE=high>
<PARAM NAME=bgcolor VALUE=#000000>
<EMBED src="movie.swf" quality=high bgcolor=#000000
        WIDTH=600 HEIGHT=400
        TYPE="application/x-shockwave-flash"
        PLUGINSPAGE="http://www.
        macromedia.com/shockwave/download/
        index.cgi?P1_Prod_Version=ShockwaveFlash">
</EMBED>
</OBJECT>
```

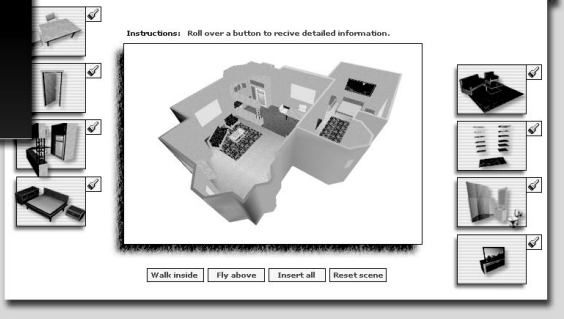
Sie sehen hier ein `Object-Tag`, welches auf einer HTML-Seite eingebaut werden könnte. Das Ganze würde bewirken, dass ein Flashfilm mit dem Namen MOVIE.SWF in einem 600x400 Pixel großen Bereich auf der Seite angezeigt werden würde. Zur Spezifikation des benötigten Plugins werden noch die `classid` und `codebase` mit angegeben, damit der Browser genau weiß, welches Plugin er benutzen bzw. nachinstallieren muss. Die zusätzlich angegebenen Parameter wie `quality` und `bgcolor` werden direkt an das Plugin übergeben und von diesem verarbeitet. Innerhalb des `Object-Tags` ist noch ein `Embed-Tag` angegeben, welches im Wesentlichen die gleichen Angaben noch mal enthält. Dies ist einfach notwendig, um sicherzugehen, dass der Internet Explorer und Netscape die Angaben verstehen, da diese beiden Browser Plugins leicht unterschiedlich behandeln.


style...red
style_rede


Dimensions: 101.5 x 44.5 x 17.4mm, 66cc Weight: 79gr
NOKIA
Self rotate


Adobe


TANK WARS
ENEMY REMAINING
1
SCORE
100
HELP
QUIT


Instructions: Roll over a button to recive detailed information.
Walk inside
Fly above
Insert all
Reset scene

2 Übersicht 3D-Technologien

In diesem Kapitel bekommen Sie einen Überblick, welche Technologien es gibt, 3D-Grafiken und -Animationen auf Webseiten zu bringen. Ich werde darstellen, welche Stärken und Besonderheiten es dabei jeweils zu beachten gilt.

Im Wesentlichen ist für 3D-Grafik auf Webseiten immer ein Plugin notwendig. Ich spreche hier allerdings bei der Übersicht nicht von „Plugins", sondern von „Technologien" – aus zwei Gründen. Zum einen gibt es die Möglichkeit, 3D-Grafik in beschränktem Maße nur mit DHTML darzustellen, was für manche Anwendungen ausreichen kann. Zum anderen gibt es eine Reihe von Technologien, die alle auf einem Java-Applet beruhen. Das heißt, sie nutzen alle das gleiche Plugin, nämlich Java, aber in unterschiedlicher Art und Weise.

2.1 Anwendungsgebiete

Dieser Überblick dient dazu, für ein Projekt die jeweils richtige Technologie auswählen zu können. Wichtig für diese Auswahl ist natürlich die Anwendung, für die man die 3D-Darstellung benötigt.

Die Anwendungsgebiete für 3D-Grafik und -Animation lassen sich in fünf Kategorien einteilen:

- Objektvisualisierung
- Raumvisualisierung/Architektur
- Menüsteuerung/Navigation
- Spiele
- Simulation

Mit Objektvisualisierung ist die Darstellung von einzelnen Objekten gemeint. Sie wird eingesetzt, um dem Besucher einer Site ein Objekt, dass er zum Beispiel kaufen möchte, besser zeigen zu können. Dies kann etwa ein Walkman, ein Auto oder ein Kleidungsstück sein. Eine weitere Möglichkeit ist die Erklärung der Funktionsweise eines Gerätes. Durch die 3D-Darstellung kann der Besucher das Objekt von allen Seiten betrachten und damit interagieren. Die Objektvisualisierung wird schon auf vielen eCommerce-Sites benutzt.

Die Raumvisualisierung wird dann benutzt, wenn man zum Beispiel die Innenarchitektur eines Gebäudes zeigen möchte. Der Besucher kann sich frei in dem Gebäude bewegen und gewinnt erst dadurch einen konkreten Eindruck von Dimensionen und Wirkung. Eine weitere Möglichkeit ist die Darstellung von Landschaften oder einer virtuellen Stadt.

3D-Menüs und Navigationen bieten zum einen dem Benutzer mehr Freiheitsgrade, wodurch unter Umständen eine natürlichere Navigation ermöglicht wird. Zum anderen kann die 3D-Darstellung aber auch einfach nur ein „Eye-Catcher" sein, da man hiermit mit relativ kleinen Dateigrößen attraktive Effekte erzielen kann.

Spiele auf Webseiten werden häufig für Marketingaktionen genutzt. Im kommerziellen Computerspielbereich sind 3D-Egoshooter seit langem sehr erfolgreich. Auch wenn derartige spezialisierte Spiele auf hohe Performance programmiert sind, kann man vergleichbare Spiele auch mit bereits verfügbaren Plugins realisieren.

Simulationen basieren teilweise auch auf Objektdarstellungen oder Raumvisualisierungen. Der Unterschied ist jedoch, dass es hierbei nicht so sehr um den Gegenstand oder den Raum geht, sondern darum, dass Veränderungen über einen bestimmten Zeitraum hinweg visualisiert werden. Das bedeutet, dass Simulationen erheblich mehr Flexibilität von einem Plugin erwarten.

All diese Anwendungsgebiete haben besondere Anforderungen, die es bei der Auswahl einer Technologie zu beachten gilt:

- Objektvisualisierung: Hochwertige, detailreiche Darstellung
- Raumvisualisierung/Architektur: Realistische Darstellung
- Menüsteuerung/Navigation: Geringe Ladezeiten
- Spiele: Komplexe Interaktion, schnelle Darstellung
- Simulation: Komplexe Animation

2.2 Vergleichskriterien

Neben der Beschreibung der wichtigsten Fakten wurden bei den einzelnen Technologien die folgenden Aspekte betrachtet:

Hersteller
In der sich schnell ändernden Internetwelt ist der Name und die Stabilität eines Herstellers unter Umständen ein Garant dafür, dass das Plugin auch in einigen Monaten noch weiterentwickelt wird, weil der Hersteller noch nicht bankrott ist.

Darstellungsqualität
Die Darstellungsqualität hängt von den verfügbaren Funktionen wie Multitexturing, Schattierung und vor allem Antialiasing ab.

Darstellungsgeschwindigkeit
Je komplexer eine Szene wird, desto langsamer wird die Darstellungsgeschwindigkeit bei Animationen. Irgendwann geht hierbei jede Technik in die Knie, aber es gibt durchaus Unterschiede, wie schnell. Die Darstellungsgeschwindigkeit hängt vor allem auch davon ab, ob 3D-Hardware vorhanden ist, auf die das Plugin zugreifen kann.

Authoring-Werkzeuge
3D-Welten sind oft sehr aufwändig zu erstellen. Deshalb ist es wünschenswert, dass man dafür professionelle und komfortable 3D-Editoren zur Verfügung hat.

Dynamische Veränderung der Inhalte
Webseiten dynamisch zu generieren mit ASP, JSP, PHP oder anderen Techniken ist bei vielen Sites heute Standard. Deshalb kann es auch wünschenswert sein, die Inhalte einer 3D-Szene gemäß aktueller Bedürfnisse (z.B. für Personalisierung) zu verändern.

Funktionalitäten
Zum Teil bieten die Plugins Standardfunktionalitäten für die Darstellung von 3D-Grafiken und -Animationen. Teilweise jedoch verfügen sie auch über ganz spezielle Funktionen für bestimmte Einsatzgebiete.

Downloadgröße der Installationsdateien
Da kein Browser zurzeit die native Unterstützung von 3D-Grafik bietet, ist immer ein Download eines Plugins oder einer entsprechenden Funktionsbibliothek notwendig. Je nachdem wie groß dieser Download ist, kann das einen Besucher der Site abschrecken.

Downloadgröße der Inhalte
Für die Verwendung auf Webseiten ist es meistens entscheidend, wie gut die Beschreibungsdaten einer Szene komprimiert werden, um die Downloadgrößen zu minimieren. Manche Plugins bieten auch Streaming an. Das bedeutet, dass schon während des Herunterladens der Daten das angezeigt wird, was bereits da ist.

Verfügbarkeit beim Nutzer
Verschiedene Technologien besitzen schon eine große Verbreitung unter den Internetnutzern, andere erfordern fast immer das Herunterladen eines Plugins.

Kosten
Welche Lizenzkosten fallen für die Nutzung des Plugins oder der Authoringsoftware an? Bei manchen Produkten fallen laufende Kosten an, manche haben nur einmalige Anschaffungskosten und andere kosten gar nichts.

Um einen konkreten Eindruck von der Technologie zu bekommen, enthält jede Beschreibung einen Screenshot von einem Beispiel.

Die einzelnen Angaben wurden jeweils den Produktbeschreibungen von den Webseiten des entsprechenden Unternehmens entnommen oder beruhen auf eigenen Erfahrungen. Aus diesem Grund konnte auch nicht zu jedem Aspekt eine Aussage getroffen werden.

2.3 Untersuchte Technologien

Untersucht wurden die folgenden Technologien:

VR Viewer	
ZAP	Seite 66
iPix	Seite 67
QuicktimeVR	Seite 68

Appletbasierte Techniken	
Anfy3D	Seite 69
3Danywhere	Seite 70
Critical Reach	Seite 71
Shout3D	Seite 72

Sie sehen in der obigen Liste eine Einteilung der Technologien in die folgenden Rubriken:

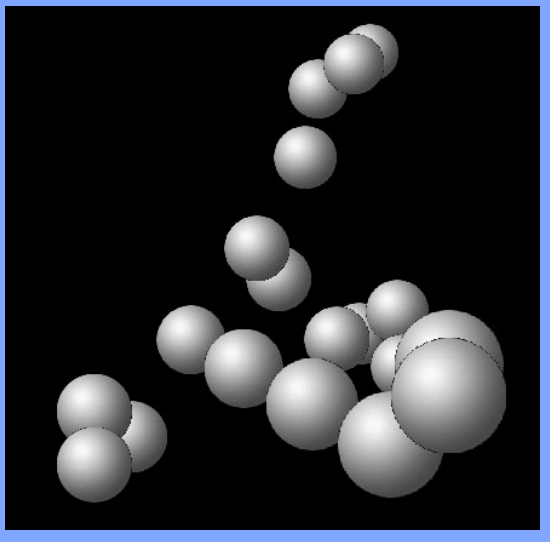
- Ohne Plugin: Diese Sonderstellung hat nur DHTML als eingebaute Technologie in den Standardbrowsern.
- Allgemeine Plugins: In dieser Rubrik sind alle Plugins zusammengefasst, die für 3D-Darstellungen geeignet oder darauf spezialisiert sind.
- VRML Viewer: VRML Viewer sind auch Plugins, die 3D-Grafiken anzeigen. Allerdings beruhen diese auf dem VRML-Standard und haben kein eigenes, proprietäres Format.
- VR Viewer: VR (Virtual Reality) Viewer habe ich diejenigen Plugins genannt, die keine echte 3D-Darstellung bieten, sondern Fotos von realen Dingen so abbilden, als könnte man die Dinge drehen und wenden bzw. als würde man sich selbst in einer Umgebung drehen und bewegen können.
- Appletbasierte Viewer: Diese Viewer haben in der Regel ein eigenes Format, in welchem sie 3D-Grafiken speichern und auf einer Webseite anzeigen. Allerdings liegen die Viewer als Java-Applets vor, so dass die Grafiken mit jedem Browser und auf jeder Plattform angezeigt werden können, die eine Java 1.1-konforme Virtual Machine unterstützen.

2.4 Übersicht der Technologien

DHTML/JavaScript

Fakten

Hersteller	Microsoft, Netscape und andere
URL	*www.w3c.org*
Plattformen	Alle, für die es Browser mit HTML4-Support gibt
Installations-dateigröße	keine
Lizenzkosten	keine



Stärken

- Größte Verfügbarkeit
- Keine Kosten

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität ist recht beschränkt, die Linien bei der Drahtgitterdarstellung sind unsauber und die Grafiken für Sprites haben kein Antialiasing und lassen sich nur unsauber skalieren.

Darstellungsgeschwindigkeit: Da alle Berechnungen mit JavaScript durchgeführt werden, ist die Geschwindigkeit nicht sehr hoch. Es lassen sich nur sehr wenig Objekte darstellen.

Funktionalitäten: Die möglichen Darstellungsfunktionalitäten werden im Kapitel über DHTML näher erläutert. Real berechnete 3D-Grafik ist aber damit kaum möglich.

Datenformat: Die Modelldaten müssen in JavaScript programmiert werden.

Dynamik: Durch die direkte Umsetzung in DHTML ist die dynamische Änderung der Inhalte über die üblichen Mechanismen (ASP, JSP, PHP etc.) problemlos möglich.

Authoring: Für das Authoring bieten sich nur normale Grafikprogramme zur Erzeugung von statischen Bildern an. 3D-Geometriedaten lassen sich zwar theoretisch auch mit Spezialsoftware erstellen, allerdings muss trotzdem immer noch von Hand nachbearbeitet werden.

Kosten: Bei den Kosten ist DHTML mit Sicherheit die günstigste aller Varianten.

Plattformen: DHTML ist zwar mit Netscape auf den meisten Plattformen verfügbar. Allerdings gestaltet sich die Programmierung durch unterschiedliche Implementierungen der Browser zum Teil als sehr schwierig.

Verbreitung: Die Verbreitung ist natürlich unschlagbar, da DHTML in jeden Browser „eingebaut“ ist. Allerdings gibt es auch Browserinstallationen, bei denen JavaScript deaktiviert ist.

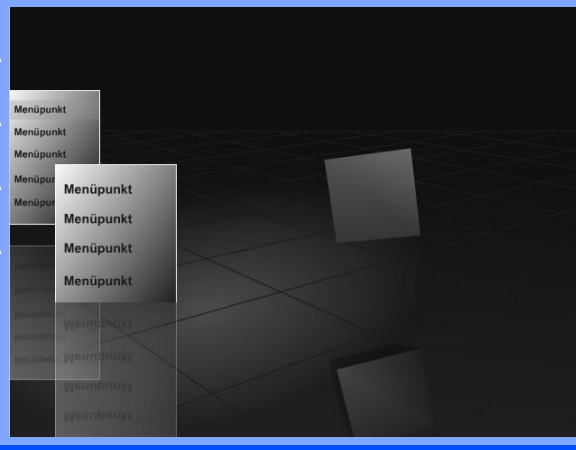
Zusammenfassung

Von der Verfügbarkeit beim Nutzer kann es keine andere Technologie mit DHTML aufnehmen. Auch wenn die Möglichkeiten sehr eingeschränkt sind, hat diese Technologie durchaus ihre Anwendungsgebiete. Wenn die potenziellen Nutzer der Site innerhalb einer Firma sitzen, in der keine Plugins erlaubt sind, bleibt DHTML als einzige Alternative.

Flash

Fakten

Hersteller	Macromedia, Sitz in den USA
URL	*www.macromedia.com/software/flash*
Plattformen	Windows, Mac, Solaris, Linux
Installations-dateigröße	ca. 0,3 MB
Lizenzkosten	Plugin/Publishing: keine Authoring Software: ca. 600 EUR



Stärken

- Sehr große Verfügbarkeit
- Geringe Kosten

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität von Flash ist sehr hoch, da alle Elemente mit Antialiasing gezeichnet werden können. Allerdings sind die Möglichkeiten für echte 3D-Darstellungen eingeschränkt. Zum Beispiel können keine Texturen verwendet werden.

Darstellungsgeschwindigkeit: Bei real berechneter 3D-Grafik sind nur sehr einfache Modelle möglich, da alle Berechnungen mit ActionScript durchgeführt werden müssen. Bei vorberechneten Grafiken ist die Geschwindigkeit besser, aber auch hier können nicht zu viele Elemente auf einmal gezeichnet werden.

Funktionalitäten: Die möglichen Darstellungsfunktionalitäten werden im Kapitel über Flash näher erläutert. Real berechnete 3D-Grafik ist möglich, aber nur eingeschränkt.

Datenformat: Die Modelldaten müssen in ActionScript programmiert werden.

Dynamik: Durch die Möglichkeit des Nachladens von Daten vom Server oder durch Interaktion mit der Webseite kann Dynamik realisiert werden.

Authoring: Die eigentlichen Authoring-Werkzeuge für Flash-Filme stellen keine Hilfsmittel für 3D-Grafik zur Verfügung. Es gibt allerdings viele Zusatzprogramme, um entsprechende Dateien zu erzeugen. Für real berechnete Objekte ist allerdings Handarbeit notwendig.

Kosten: Bei den Kosten ist Flash recht günstig, da nur die Authoringsoftware gekauft werden muss.

Verbreitung: Flash ist mit Sicherheit das Plugin, das am weitesten verbreitet ist, da es standardmäßig mit den meisten Browsern mitinstalliert wird.

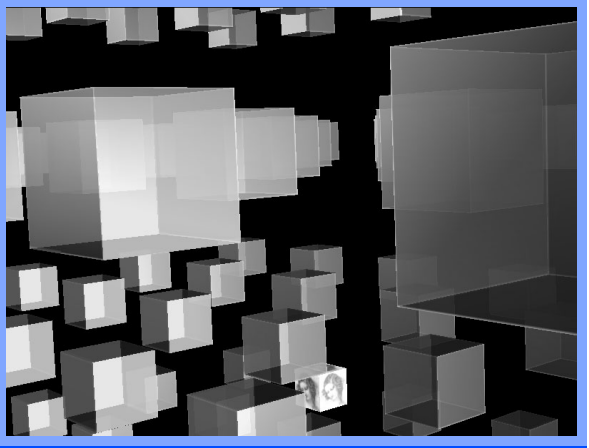
Zusammenfassung

Bei real berechneter 3D-Grafik gibt es viele Einschränkungen, da Flash einfach keine richtige 3D-Engine enthält. Das äußert sich z.B. in der geringen Darstellungsgeschwindigkeit und den fehlenden Texturen. Man kann aber Drahtgittermodelle, schattierte Modelle und vorberechnete Bilder im 2D-Vektorformat sehr gut benutzen. Gerade mit der neuen Version MX gestaltet sich die Programmierung real berechneter Grafik inzwischen einfacher.

Director Shockwave 8.5

Fakten

Hersteller	Macromedia, Sitz in den USA
URL	*www.macromedia.com/software/director*
Plattformen	Windows, Mac
Installationsdateigröße	ca. 3,5 MB
Lizenzkosten	Plugin/Publishing: Keine Authoring Software: ca. 1500 EUR



Stärken

- Sehr viele Möglichkeiten, komplett programmierbar in Lingo
- Gute Hardwareunterstützung

Anmerkungen

Darstellungsqualität: Durch die sehr vielfältigen Möglichkeiten von Director lassen sich sehr realistische Bilder erstellen. Allerdings unterstützt Director bisher kein Antialiasing im 3D-Modus.

Darstellungsgeschwindigkeit: Director kann entweder den eingebauten Softwarerenderer benutzen oder auf vorhandene Hardwareschnittstellen wie OpenGL oder DirectX zugreifen. Dadurch hängt die Geschwindigkeit im Wesentlichen vom Rechner ab. Man kann innerhalb eines Films die vorhandene Hardware auch überprüfen und dann entsprechend darauf reagieren.

Datenformat: Director verwendet ein eigenes Datenformat w3d, welches allerdings nur für den Datenaustausch mit der Authoringsoftware verwendet wird. Ansonsten wird für die Publikation eines Films eine dcr-Datei erzeugt, die die Modelldaten und alles andere enthält.

Dynamik: Durch das Nachladen von Daten und die Interaktion mit der Webseite kann Dynamik realisiert werden.

Authoring: Director selbst enthält keinen 3D-Grafikeditor. Es gibt allerdings für die meisten professionellen Werkzeuge Exportmodule, die das entsprechende Dateiformat erzeugen.

Kosten: Es entstehen nur die Kosten für die Authoringsoftware, die allerdings schon recht hoch sind.

Verbreitung: Nach Flash besitzt das Director Shockwave-Plugin die größte Verbreitung. Da es allerdings nicht mit dem Browser mitgeliefert wird, muss der Anwender es selbst herunterladen und installieren.

Zusammenfassung

Mit Director 8.5 ist es möglich, fast alles, was man möchte, darzustellen. Die Einschränkungen bestehen hauptsächlich durch die beim Betrachter verfügbare Hardware. Es gibt Mehrfachtexturen, komplexe Modelle, Keyframe- und Bones-Animationen, Partikelsysteme u.v.m.

Director könnte sich zurzeit als Standard-Plugin für 3D-Grafik im Web durchsetzen aufgrund der Marktposition von Macromedia und der guten Umsetzung des Produktes.

Cult3D

Fakten

Hersteller	Cycore, gegründet 1996, Sitz in Schweden
URL	*www.cult3d.com*
Plattformen	MS Windows, MacOS, Linux, Solaris, HP-UX, AIX
Installations-dateigröße	ca. 1,2 MB
Lizenzkosten	Kommerziell: 500 USD pro Jahr + eine monatliche Gebühr abhängig von der Anzahl der Besucher der Site Nicht-kommerziell: frei, abhängig jedoch von einer individuellen Entscheidung von Cycore

Stärken

- Gute Referenzen
- Keine Hardwareabhängigkeit
- Kompaktes Datenformat

Anmerkungen

Darstellungsqualität: Der Viewer unterstützt Antialiasing, welches zur Geschwindigkeitssteigerung aber abgeschaltet werden kann.

Darstellungsgeschwindigkeit: Die Grafik-Engine ist in Software implementiert und benutzt keine Hardwarebeschleunigung.

Datenformat: Der Viewer unterstützt Kompression und Streaming. Für eine verbesserte Bildkompression wird sogar eine Wavelet-Komprimierung angeboten. Die Daten werden in einer .co-Datei übertragen. Die Daten in solch einer Datei lassen sich nicht mehr auslesen, sind also gegen Kopieren geschützt.

Dynamik: Durch Kommunikation mit der Webseite können Ereignisse ausgelöst werden.

Authoring: Als Authoring-Werkzeuge können 3D Studio Max und Maya verwendet werden. Anschließend muss man mit dem mitgelieferten Cult3D-Designer die interaktiven Funktionalitäten hinzufügen.

Kosten: Die Kosten sind abhängig von der Anzahl der Besucher einer Site und somit nicht so gut zu kontrollieren wie bei Produkten, die nur einmal gekauft werden müssen.

Plattformen: Für ein reines 3D-Plugin unterstützt der Viewer sehr viele Plattformen und eignet sich daher gut für kommerzielle Sites.

Verbreitung: Nach eigenen Angaben gibt es 10 Millionen installierte Cult3D-Viewer. Der Viewer wird auch schon auf vielen kommerziellen Sites verwendet.

Zusammenfassung

Cult3D ist ein vielseitiges Plugin, welches vor allem für viele Plattformen verfügbar ist. Es unterstützt die wichtigsten Funktionalitäten wie Transparenzen, Schattenwurf und Antialiasing.

Cult3D ist ein professionelles Produkt mit vielen Referenzen, welche vor allem zur Produktpräsentation benutzt werden.

Viewpoint Media Player

Fakten

Hersteller	Viewpoint, gegründet 2000 aus MetaCreations/ Metastream, Sitz in den USA
URL	*www.viewpoint.com*
Plattformen	MS Windows, MacOS
Installations-dateigröße	ca. 1,3 MB
Lizenzkosten	Kommerziell: komplexes Lizenzmodell, z.B. 10000 USD für eine kleine Site Nicht-kommerziell: frei, muss allerdings nach 6 Monaten erneuert werden.

Stärken

- XML-Datenformat
- Bietet neben 3D auch Unterstützung für andere Medien
- Gute Referenzen

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität ist sehr gut und es lassen sich unterschiedliche Elemente zusammenmischen, um besondere Effekte zu erreichen.

Darstellungsgeschwindigkeit: Das Plugin nutzt eine eigene 3D-Engine und keine Hardwarebeschleunigung.

Datenformat/Dynamik: Die Dateien basieren auf XML, welches zur Laufzeit dynamisch generiert/geändert werden könnte.

Authoring: Es gibt Export-Plugins für 3D Studio Max, Maya und Lightwave. Für das endgültige Zusammenfügen der Dateien gibt es eine mitgelieferte Authoring-Software.

Kosten: Das Lizenzmodell ist recht komplex. Für detaillierte Informationen sollte man sich direkt mit Viewpoint in Verbindung setzen.

Verbreitung: Nach eigenen Angaben gab es 100 Millionen Installationen des Viewers Ende 2001.

Zusammenfassung

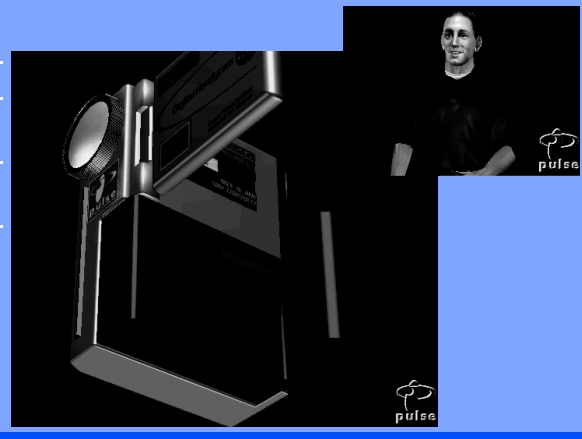
Viewpoint platziert sich mit der Viewpoint Experience Technology (VET) als Rich Media Plattform. Der Viewpoint Media Player (VMP) ist die Basis davon und kann viele verschiedene Medien wiedergeben, darunter 3D-Welten, Flash, QTVR und andere. Der Player ist dabei komponentenbasiert und lädt selbstständig fehlende Komponenten nach.

Viewpoint ist ein professionelles Produkt und macht einen guten Eindruck. Es hat deshalb aber auch einen hohen Preis.

Pulse3D

Fakten

Hersteller	Pulse, gegründet 1994, Sitz in den USA
URL	*www.pulse3d.com*
Plattformen	MS Windows, MacOS, PocketPC
Installations-dateigröße	Ca. 0,4 MB
Lizenzkosten	30-Tage-Demo frei, danach 3000 USD für das Pulse Animation Studio



Stärken

- Character-Animation
- Gute Darstellungsqualität
- Inhalte können gegen Kopieren geschützt werden
- Streaming

Anmerkungen

Darstellungsqualität: Das Plugin unterstützt Antialiasing und bietet eine gute Darstellungsqualität.

Darstellungsgeschwindigkeit: Pulse benutzt zur Wiedergabe entweder einen Softwarerenderer oder eine vorhandene Hardwareschnittstelle wie DirectX oder OpenGL (Letzteres auf dem Mac).

Datenformat: Die Daten werden gut komprimiert. Das Plugin unterstützt sogar Wavelet-Komprimierung für Bilder und Unterstützung für Streaming.

Dynamik: Dynamische Änderungen zur Laufzeit können in PulseScript programmiert oder über ein Servermodul erzeugt werden.

Authoring: Es sind Importfunktionen für die wichtigsten Formate enthalten. Außerdem ist ein 3D-Editor enthalten.

Kosten: Die Kosten für die Software sind zwar fix, bewegen sich aber im oberen Bereich.

Zusammenfassung

Pulse3D kann eng integriert werden mit dem Realplayer oder Apples Quicktime. Er bietet gute 3D-Unterstützung und viele Extrafunktionen im Bereich Characteranimation und Sprachsynchronisation.

Rover

Fakten	
Hersteller	Flatland, gegründet 1998, Sitz in den USA
URL	*www.flatland.com*

Plattformen	MS Windows
Installationsdateigröße	ca. 1,2 MB
Lizenzkosten	Shareware

Stärken

- Günstiger Preis
- Einfaches Datenformat

Anmerkungen

Darstellungsqualität: Die Qualität der Darstellung ist nicht so gut wie bei anderen Produkten.

Darstellungsgeschwindigkeit: Vorhandene Hardware wird mittels DirectX unterstützt.

Datenformat: Das Plugin benutzt ein eigenes Datenformat namens 3DML, welches ähnlich XML ist.

Authoring: Es gibt einen Editor namens Sputnik für das 3DML-Format. Ansonsten muss man die Daten von Hand editieren.

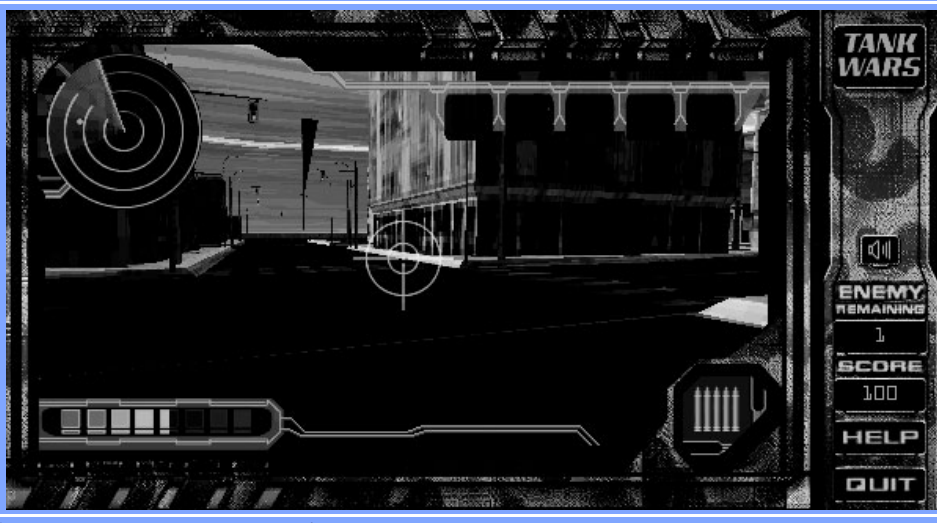
Verbreitung: Über die Verbreitung gibt es keine Angaben, aber in der Regel muss man bei diesem Plugin davon ausgehen, dass es nicht bereits installiert ist.

Zusammenfassung

Die letzten Pressenachrichten auf der Website von Flatland sind aus dem Jahr 1999, auch wenn die Entwicklung des Plugins scheinbar noch weitergeht. Insgesamt sind die Informationen darüber jedoch nur sehr spärlich.

3D Groove

Fakten	
Hersteller	The Groove Alliance, gegründet 1999, Sitz in den USA
URL	*www.3dgroove.com*



Plattformen	MS Windows, MacOS
Installationsdateigröße	Version 1.8 ca. 390 KB, basiert auf Macromedia Shockwave. Version 2.0 ist ein eigenes Plugin
Lizenzkosten	Keine Information. Es fallen Lizenzkosten für die Groove-Plattform an.

Stärken

- Spezialisierung auf Spiele
- Gute Referenzen
- Relativ gute Verfügbarkeit, da es auf Shockwave basiert

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität des Plugins ist gut. Es unterstützt Antialiasing und gute Texturfunktionen (Bump-Mapping, Reflectionmaps etc.)

Darstellungsgeschwindigkeit: Es wird keine 3D-Hardware unterstützt. Die Darstellungsgeschwindigkeit ist aber sehr hoch.

Funktionalitäten: Da das Plugin auf Spiele ausgerichtet ist, werden vor allem hierfür benötigte Funktionalitäten geboten, z.B. Kollisionserkennung.

Datenformat: Bei dem Plugin handelt es sich vor allem um eine Programmierschnittstelle.

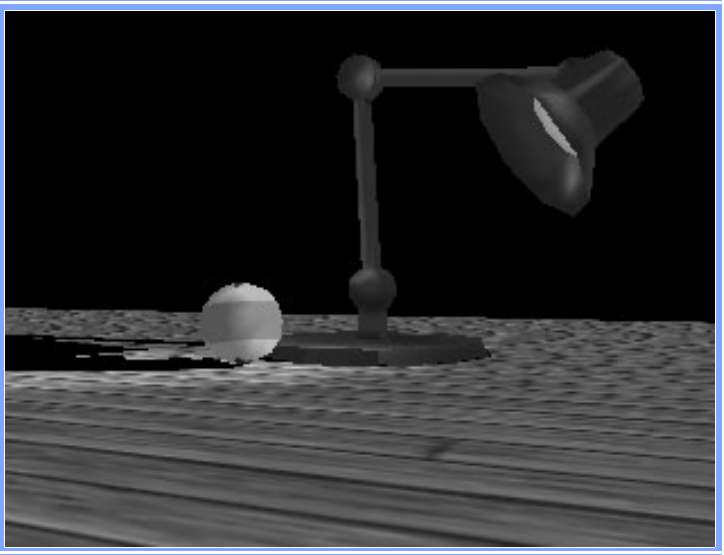
Authoring: Mit dem mitgelieferten Authoring-Werkzeug können die Spiele erstellt werden.

Verbreitung: Nach eigenen Angaben gibt es 30 Millionen Installationen des Plugins.

Zusammenfassung

Bis Version 1.8 basiert die Engine auf Shockwave, erst ab Version 2.0 ist es auch als ein eigenständiges Plugin verfügbar. Das Plugin ist spezialisiert auf Spiele für das Web und hat in diesem Bereich auch viele Referenzen.

Alice

Fakten	
Hersteller	Carnegie Mellon Universität
URL	*www.alice.org*
	
Plattformen	MS Windows
Installationsdateigröße	ca. 0,9 MB
Lizenzkosten	keine

Stärken

- Freies Produkt
- Gut für Testzwecke

Anmerkungen

Darstellungsgeschwindigkeit: Das Plugin greift über DirectX auf vorhandene Hardware zu.

Funktionalitäten: Über die Programmierung mit Python-Modulen lassen sich sehr viele Effekte realisieren, ansonsten scheinen die Funktionalitäten eher eingeschränkt zu sein. Interessant ist aber die Videorekorder-Funktion. Damit können Animationen aufgezeichnet und später wieder abgespielt werden.

Dynamik: Das Plugin kann mit Python programmiert werden.

Authoring: Es können Daten aus 3D Studio Max importiert werden. Hauptsächlich jedoch werden die Inhalte programmiert.

Kosten: Das Plugin und die zugehörige Software sind frei verfügbar.

Plattformen: Zurzeit gibt es nur eine Implementierung für Windows.

Zusammenfassung

Alice ist eine Eigenentwicklung der Carnegie Mellon Universität. Das Plugin hat daher auch weniger einen kommerziellen Anspruch. Es ist vor allem dafür entwickelt worden, um schnell Konzepte in 3D ausprobieren zu können.

B3D

Fakten	
Hersteller	Brilliant Digital, gegründet 1995, Sitz in den USA
URL	*www.brilliantdigital.com*
	
Plattformen	MS Windows
Installationsdateigröße	ca. 1,1 MB
Lizenzkosten	Plugin/Publishing: Keine Authoring Software: 995 USD

Stärken

- Lippen-synchrone Wiedergabe von Animationen
- Funktionalitäten für Videos und Bannerwerbung

Anmerkungen

Darstellungsqualität: Das Plugin unterstützt Antialiasing.

Darstellungsgeschwindigkeit: Vorhandene 3D-Hardware wird unterstützt.

Funktionalitäten: Bei Webvideos kann der Nutzer zwischen Handlungssträngen wählen. Ansonsten werden für 3D-Grafiken die grundlegenden Funktionen zur Verfügung gestellt.

Dynamik: Es ist möglich, Skripte mit einzubinden und über JavaScript mit der Webseite zu kommunizieren.

Authoring: Es sind Importfunktionen für die wichtigsten Formate enthalten. Außerdem ist ein 3D-Editor enthalten.

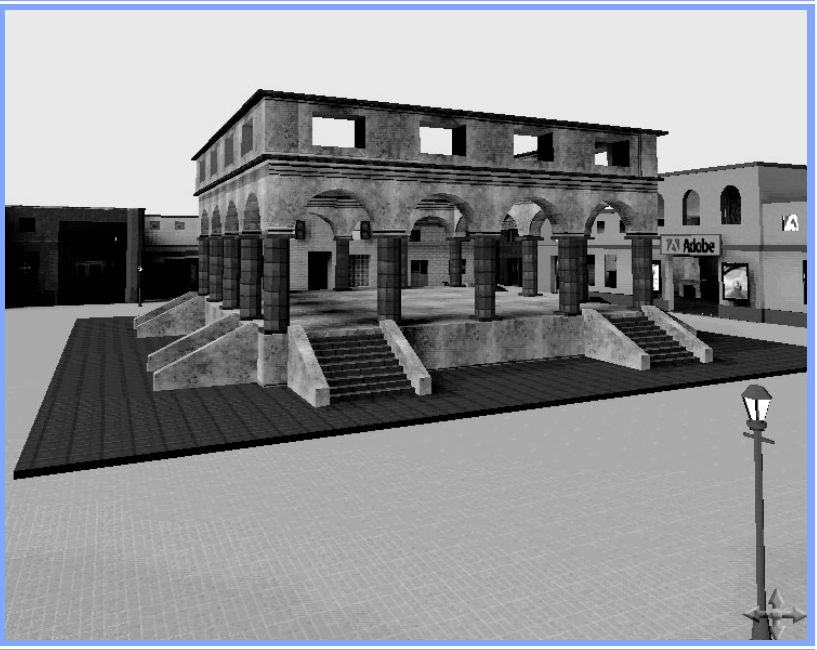
Verbreitung: Nach eigenen Angaben ist b3d „der am häufigsten heruntergeladene 3D-Viewer mit über 2,8 Millionen Downloads pro Woche“.

Zusammenfassung

B3D ist ein Plugin mit vielen Funktionalitäten, welches vor allem auf die Erstellung von Web-Videos und Avataren spezialisiert ist. Hierfür werden Video-spezifische Funktionalitäten (Lippensynchronisation, Handlungsstränge, Fullscreen-Wiedergabe etc.) geboten.

Atmosphere

Fakten	
Hersteller	Adobe, Sitz in den USA
URL	*www.adobe.com/products/atmosphere*



Plattformen	MS Windows
Installationsdateigröße	ca. 4 MB
Lizenzkosten	Keine Information (noch kein offizielles Release)

Stärken

- Adobe hat eine starke Stellung im Markt
- Einfacher eigener 3D-Editor
- Gute Darstellung

Anmerkungen

Darstellungsqualität: Das Plugin unterstützt Antialiasing.

Darstellungsgeschwindigkeit: Das Plugin benutzt einen Softwarerenderer.

Datenformat: Atmosphere benutzt das Viewpoint(früher Metastream)-Datenformat.

Authoring: Es ist ein eigener Editor enthalten, mit dem die Welten zusammengebaut werden können. Für die Erstellung der Modelle können aber auch die meisten professionellen 3D-Editoren verwendet werden.

Plattformen: Es ist zu erwarten, dass zumindest Windows und Mac unterstützt werden.

Zusammenfassung

Atmosphere macht einen guten ersten Eindruck. Allerdings kam die Software schon Anfang 2001 als Betaversion heraus und ist bis jetzt über dieses Stadium nicht hinausgegangen. Evtl. wurde die Entwicklung auch komplett gestoppt.

EON

Fakten	
Hersteller	EON Reality, gegründet 1999, Sitz in den USA
URL	*www.eonreality.com*
	
Plattformen	MS Windows
Installationsdateigröße	ca. 1,1 MB, basiert auf Director Shockwave (ohne 3D)
Lizenzkosten	Keine Informationen

Stärken

- Relativ gute Verfügbarkeit, da es auf Shockwave basiert
- Spezialisierung auf Architektur

Anmerkungen

Darstellungsqualität: Das Plugin unterstützt Antialiasing und macht einen sehr guten Eindruck bei der Darstellungsqualität.

Darstellungsgeschwindigkeit: Es wird keine Hardware unterstützt. EON wirbt damit, dass selbst auf älteren Rechnern eine schnelle Darstellung gewährleistet ist.

Funktionalitäten: Es werden nur einfache Animationen und Interaktion unterstützt.

Datenformat: Das Plugin nutzt ein eigenes Datenformat, welches die 3D-Daten besonders gut komprimieren soll.

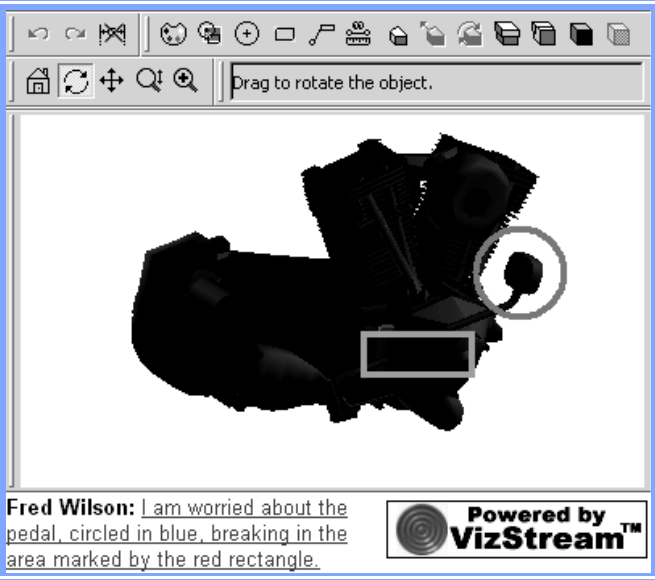
Authoring: Es ist möglich, existierende Modelle zu importieren.

Verbreitung: Das eigentliche Plugin ist eine Ergänzung zu Director Shockwave. Es ist also darauf angewiesen, dass Shockwave bereits installiert ist.

Zusammenfassung

EON ist fokussiert auf Entwicklungswerkzeuge zur Erstellung dreidimensionaler Objekte oder Umgebungen, vor allem im Bereich der Architektur.

VizStream

Fakten	
Hersteller	RealityWave, gegründet 1996, Sitz in den USA
URL	*www.realitywave.com*
	
Plattformen	MS Windows
Installationsdateigröße	Diverse unterschiedliche Konfigurationen
Lizenzkosten	Komplexes Lizenzmodell je nach Anwendungsgebiet und Nutzeranzahl

Stärken

- Funktionen, die die Zusammenarbeit mehrerer Personen an einem Modell unterstützen

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität ist nicht besonders hoch. Dies liegt aber auch an der Ausrichtung auf Funktionalitäten zur Kollaboration, statt auf grafische Effekte.

Funktionalitäten: Es werden viele Funktionen zu Kollaboration unterstützt, wie zum Beispiel Teile eines Modells zu markieren und eine Anmerkung dazu zu schreiben.

Datenformat: Das Plugin nutzt XGL als Datenformat und unterstützt durch Streaming auch das Anzeigen von sehr großen Modellen.

Authoring: Die Erstellung der Modelle kann mit Editoren vorgenommen werden, die XGL exportieren können.

Kosten: Für die Kosten für ein konkretes Projekt muss man sich direkt an den Hersteller wenden. Für kleinere Anwendungen bietet die Firma das Ganze auch als Service an, ohne dass man eine Serverlizenz erwerben muss.

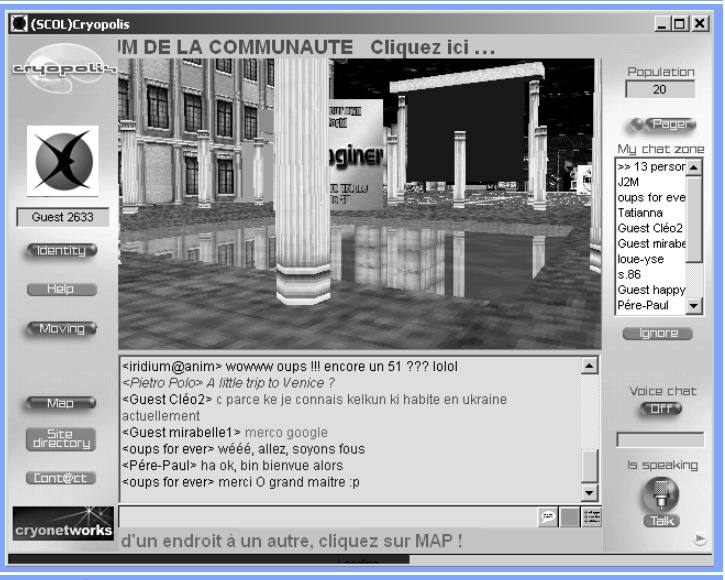
Zusammenfassung

VizStream ist spezialisiert auf Kollaboration, d.h. das Zusammenarbeiten mehrerer Personen (an verschiedenen Standorten) an einem (3D-)Modell.

Das Produkt nimmt damit eine Sonderposition ein. Es ist weniger geeignet für Produktpräsentationen.

SCOL

Fakten	
Hersteller	cryonetworks, gegründet 1999, Sitz in Kanada
URL	*www.cryonetworks.com*



Plattformen	MS Windows, MacOS
Installationsdateigröße	ca. 1,3 MB
Lizenzkosten	Keine Informationen

Stärken

- Gestaltung von 3D-Community-Welten

Anmerkungen

Funktionalitäten: SCOL bietet die Möglichkeiten, 3D-Welten zu erstellen, durch die man sich mit einem Avatar bewegen kann. In der Welt kann man dann mit anderen Personen Kontakt aufnehmen und mit diesen sprechen.

Datenformat: Zur Beschreibung der Welten benutzt das Produkt eine Art Programmiersprache.

Authoring: Die Firma bietet Softwarepakete an, um 3D-Welten wie Communities oder Einkaufserlebnisse ohne Programmierkenntnisse zu erstellen.

Zusammenfassung

SCOL ist vom Grundsatz her ganz anders aufgebaut als die meisten anderen Plugins. Zunächst ist SCOL eine Programmiersprache, die die Erstellung von interaktiven Welten und auch Online-Spielen erlaubt. Die 3D-Welt wird außerdem nicht in der Webseite dargestellt, sondern in einem eigenen Fenster.

SCOL unterstützt mit dem Site-Construction-Set das Erstellen von Welten ohne Programmierkenntnisse.

Virtue3D

Fakten	
Hersteller	Virtue3D, gegründet 1997, Sitz in den USA
URL	*www.virtue3d.com*
	
Plattformen	MS Windows
Installationsdateigröße	ca. 0,4 MB
Lizenzkosten	Keine Informationen

Stärken

- Sehr kompaktes Datenformat
- Kleine Installationsdatei

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität des Plugins ist sehr gut und es unterstützt Antialiasing. Allerdings sind die verfügbaren Darstellungstechniken nicht so weit reichend wie beispielsweise bei Director.

Funktionalitäten: Alle interaktiven Funktionalitäten müssen in der Webseite programmiert werden. Hier kann man mittels einer definierten Schnittstelle auf verschiedene Funktionen des Plugins über JavaScript zugreifen.

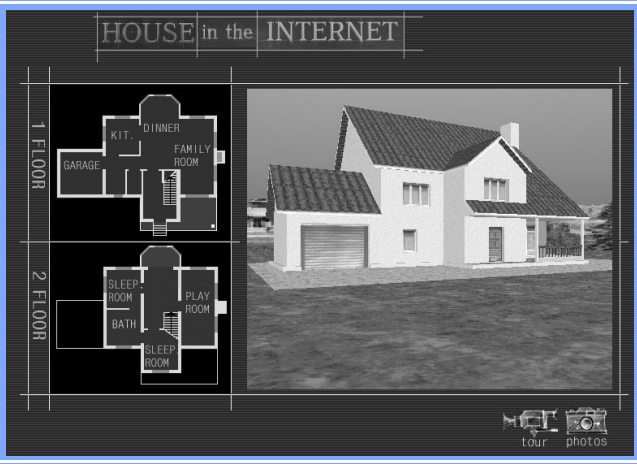
Datenformat: Das Plugin nutzt das eigene VTU-Format, welches besonders kompakte Dateien ermöglicht.

Authoring: Für das Erstellen der Objekte ist ein extra 3D-Editor notwendig. Die Virtue3D-Software importiert Modelle in verschiedenen Formaten (z.B. dxf, 3ds, obj, vrml 2.0) und konvertiert diese ins VTU-Format.

Zusammenfassung

Die Stärke vonVirtue3D ist vor allem, 3D-Objekte in möglichst kleine Dateien umzuwandeln, um sie so schneller über das Internet verbreiten zu können. Das Plugin eigent sich vor allem für Produktpräsentationen.

Cortona

Fakten	
Hersteller	Parallelgraphics, gegründet 1998, Sitz in Irland
URL	*www.parallelgraphics.com*
	
Plattformen	MS Windows, MacOS, Pocket PC, Java
Installationsdateigröße	ca. 0,96 MB
Lizenzkosten	Keine

Stärken

- Verfügbarkeit für viele Plattformen
- Volle Abdeckung von VRML97 mit zusätzlichen Erweiterungen

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität ist sehr gut durch Antialiasing und viele erweiterte Darstellungsmöglichkeiten.

Darstellungsgeschwindigkeit: In der neuesten Version wird neben dem Software-Renderer auch DirectX und OpenGL unterstützt.

Funktionalitäten: Der VRML-Viewer deckt den komplette VRML97-Standard ab und erweitert diesen um einige Funktionalitäten.

Datenformat: Der Viewer benutzt den VRML97-Standard.

Dynamik: Über Java oder JavaScript können dynamische Funktionen programmiert werden.

Authoring: Für das Erstellen der Modelle kann jeder Editor benutzt werden, der VRML97 exportieren kann. Die Firma bietet aber auch eigene Authoringsoftware für verschiedene Spezialgebiete an.

Kosten: Der VRML-Viewer selbst ist kostenlos und durch das Standardformat VRML97 muss auch nicht zwangsläufig weitere Software gekauft werden.

Zusammenfassung

Bei diesem Plugin handelt es sich um einen VRML-Viewer. Das Plugin wird in Zukunft mit Sicherheit auch X3D unterstützen.

Man benötigt für die Erstellung der Modelle keinen extra Editor, da die meisten Werkzeuge VRML erzeugen können.

Cybercore Entrance

Fakten	
Hersteller	CyCo Systems, Sitz in Deutschland
URL	*www.cycosys.com*

Plattformen	MS Windows
Installationsdateigröße	ca. 0,4 MB
Lizenzkosten	Plugin: Keine Editor: 10000 EUR

Stärken

- Kleine Installationsdatei
- Spezialisierung auf Online-Spiele

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität ist durch Antialiasing und viele Effekte wie Schattenwurf und prozedurale Texturen sehr gut.

Darstellungsgeschwindigkeit: Neben dem Software-Renderer wird auch DirectX und OpenGL unterstützt.

Funktionalitäten: Das Plugin enthält spzielle Funktionalitäten für Online-Spiele wie zum Beispiel Kollisionserkennung und Chatmöglichkeiten.

Authoring: Für die Erstellung der Welten ist der Editor der Firma notwendig.

Zusammenfassung

Das Plugin macht einen sehr guten Eindruck durch die vielfältigen Darstellungsmöglichkeiten, die enthalten sind. Die Beispiele auf der Website zielen eindeutig auf den Bereich Online-Spiele und hierbei vor allem auf Egoshooter ab. Insgesamt sind die Informationen aber sehr spärlich und es sind nicht viele Referenzen zu finden.

RichFX

Fakten	
Hersteller	RichFX, gegründet 1997, Sitz in Israel
URL	*www.richfx.com*
	
Plattformen	MS Windows
Installationsdateigröße	ca. 0,7 MB
Lizenzkosten	Keine für nicht-kommerziellen Gebrauch

Stärken

- Sehr gute Darstellungsqualität
- Keine Kosten für nicht-kommerziellen Einsatz

Anmerkungen

Darstellungsqualität: Das Plugin hat eine sehr gute Darstellungsqualität durch Antialiasing und Radiosity (laut Produktbeschreibung; tatsächlich kann Radiosity nur simuliert sein, was aber recht gut funktioniert).

Funktionalitäten: Über eine Programmierschnittstelle können mit JavaScript interaktive Funktionalitäten realisiert werden.

Datenformat: Das Plugin nutzt ein eigenes Datenformat, welches Streaming unterstützt.

Dynamik: Dynamik ist nur über die Programmierung mit JavaScript möglich.

Authoring: Für das Erstellen der Modelle muss 3D Studio Max verwendet werden. Dafür stellt RichFX ein Plugin zur Verfügung, um das benötigte Datenformat exportieren zu können.

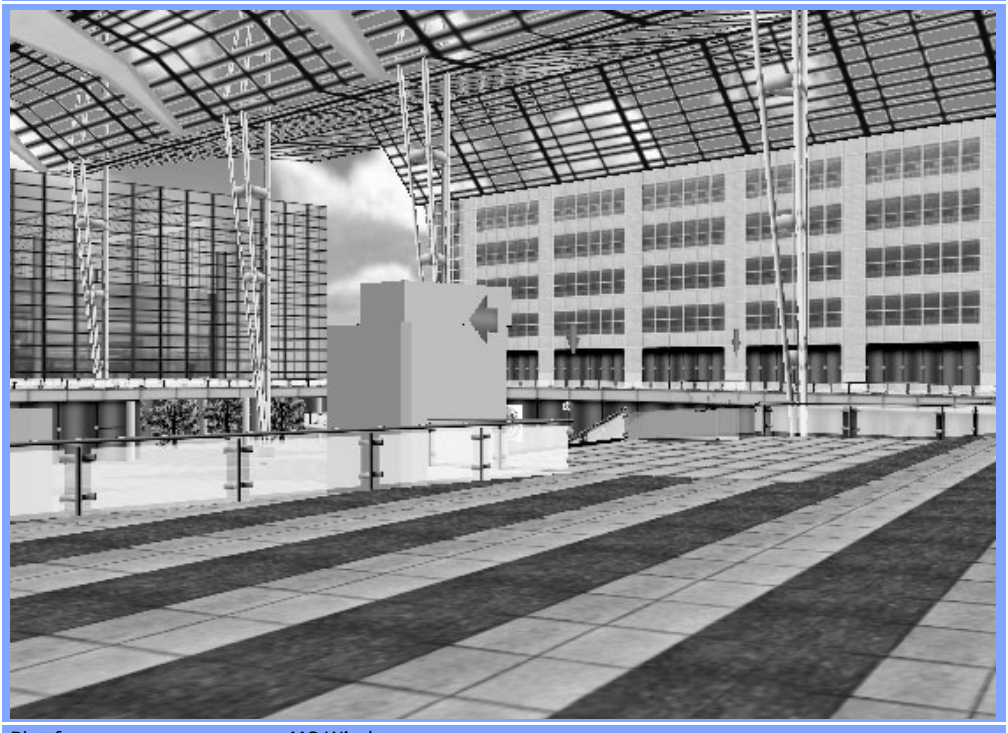
Verbreitung: Laut eigenen Angaben wird der Player standardmäßig mit dem RealPlayer installiert, ist also auf einer großen Anzahl von Systemen schon vorinstalliert.

Zusammenfassung

RichFX ermöglicht ein schnelleres Laden von 3D-Welten und -Objekten durch ein effizientes Kompressions- und Streamingverfahren. Die Darstellungsqualität ist durch das simulierte Radiosity sehr realistisch.

Blaxxun Contact

Fakten	
Hersteller	Blaxxun, gegründet 1995, Sitz in Deutschland
URL	*www.blaxxun.de*



Plattformen	MS Windows
Installationsdateigröße	ca. 1,5 MB
Lizenzkosten	Keine

Stärken

- Mitwirkung bei der Standardisierung von X3D
- Sourcen verfügbar

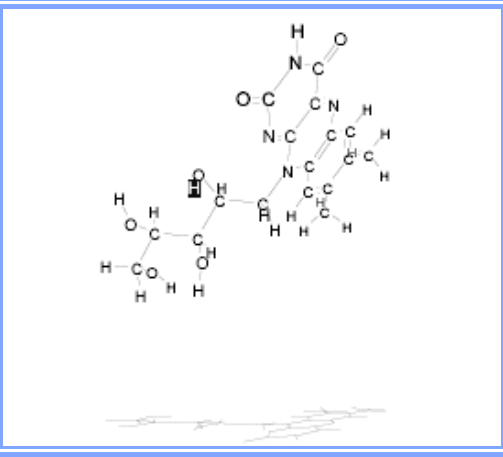
Zusammenfassung

Blaxxun ist mit dem VRML-Viewer Contact schon recht lange am Markt und ist auch aktiv tätig beim Vorantreiben der Standardisierung von X3D. Das Plugin bietet unter anderem die Möglichkeit zum Chatten über den Community Server von Blaxxun.

Zurzeit hat Blaxxun allerdings ein Insolvenzverfahren laufen und es ist nicht sicher, wie und ob die Entwicklung der Software weitergehen wird.

SVG

Fakten

Hersteller	Offizieller Standard, Implementierung z.B. von Adobe
URL	*www.w3c.org/graphics/svg/* *www.adobe.com/svg/*
	
Plattformen	MS Windows, MacOS, Linux
Installationsdateigröße	ca. 2,2 MB
Lizenzkosten	Keine

Stärken

- In Zukunft möglicher Standardbestandteil der Browser

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität ist für die vorhandenen Möglichkeiten sehr gut.

Darstellungsgeschwindigkeit: Alle 3D-Berechnungen müssen in JavaScript programmiert werden. Dadurch ist nur eine geringe Geschwindigkeit bzw. Komplexität möglich.

Funktionalitäten: Die Funktionalitäten sind auf sehr einfache 3D-Darstellungen beschränkt.

Datenformat: Alle Modelldaten müssen in JavaScript programmiert werden.

Dynamik: Durch die direkte Einbindung in die Webseite stehen alle herkömmlichen Mechanismen für dynamische Seiten zur Verfügung.

Authoring: Zurzeit gibt es nur im 2D-Bereich Werkzeuge zur Unterstützung.

Kosten: Das Plugin ist derzeit kostenlos verfügbar und wird es in Zukunft wohl auch bleiben.

Plattformen: Es ist zu hoffen, dass durch die Standardisierung die Technologie für viele Plattformen verfügbar sein wird.

Verbreitung: Zurzeit ist SVG noch nicht sehr verbreitet.

Zusammenfassung

SVG bietet im Wesentlichen die gleichen Möglichkeiten wie Flash, d.h. keine eingebauten 3D-Funktionalitäten, aber die Möglichkeit, sich eine 3D-Engine zu programmieren. Da SVG inzwischen ein verabschiedeter Standard ist, wird das Plugin hoffentlich bald zum Standardumfang jedes Browsers gehören.

ZAP

Fakten	
Hersteller	Point Cloud, Sitz in den USA
URL	*www.pointcloud.com*
	
Plattformen	MS Windows, MacOS
Installationsdateigröße	Java-Applet
Lizenzkosten	Keine Informationen

Stärken

- Keine Installationsdatei (Java-Applet)
- Spezialisierung auf Produktpräsentation mit Realbildern

Anmerkungen

Darstellungsqualität: Die Darstellungsqualität ist direkt abhängig von der Qualität der verwendeten Fotos.

Plattformen: Als Java-Applet ist das Plugin prinzipiell plattformunabhängig.

Verbreitung: Als Java-Applet wird das notwendige Plugin jedes Mal mit heruntergeladen.

Zusammenfassung

Point Cloud stellt im Wesentlichen Techniken zur Verfügung, um 3D-Produktansichten zu erzeugen. Dies funktioniert, indem das Produkt aus möglichst vielen Perspektiven fotografiert wird, und das Applet ermöglicht es dann, sich virtuell um das Objekt herumzubewegen. Es handelt sich dabei aber um keine echte 3D-Darstellung.

iPix

Fakten

Hersteller	Internet Pictures Corporation, Sitz in den USA
URL	*www.ipix.com*
	
Plattformen	MS Windows
Installationsdateigröße	ca. 0,2 MB
Lizenzkosten	Keine Informationen

Stärken

- Kleine Installationsdatei
- Spezialisierung auf Raum-/Umgebungsdarstellungen mit Realbildern

Anmerkungen

Darstellungsqualität: Die Qualität der Darstellung hängt direkt von der Qualität der verwendeten Bilder ab.

Funktionalitäten: iPix bietet im Wesentlichen die Möglichkeit, sich in dem virtuellen 3D-Panorama frei zu drehen.

Datenformat: Die Bilder liegen in einem eigenen Datenformat vor, das nur von der iPix-Software gelesen werden kann.

Authoring: Für die Erstellung der Bilder bietet iPix einen Service oder Software an, zum Beispiel aus zwei mit einer Fischauge-Linse aufgenommenen Bildern das virtuelle 3D-Panorama zu erzeugen.

Verbreitung: iPix ist schon recht lange am Markt und hat durch einige Referenzen auch schon eine gewisse Verbreitung erlangt.

Zusammenfassung

iPix arbeitet ähnlich wie QuicktimeVR und bietet 3D-Panoramabilder, Hotspots und virtuelle 3D-Objekte, aber keine echte 3D-Darstellung.

QuicktimeVR

Fakten	
Hersteller	Apple, Sitz in den USA
URL	*www.quicktime.com*
	
Plattformen	MS Windows, MacOS
Installationsdateigröße	Enthalten in Quicktime; Quicktime komplett ist ca. 9 MB groß
Lizenzkosten	Plugin: Keine Authoring: Keine Angaben

Stärken

- Gute Verfügbarkeit
- Schon lange am Markt etabliert

Anmerkungen

Darstellungsqualität: Die Qualität der perspektivisch dargestellten Bilder ist sehr gut. Allerdings hängt die Qualität natürlich auch direkt von der Qualität der verwendeten Bilder ab.

Funktionalitäten: QuicktimeVR bietet nicht nur die üblichen Panoramabilder, sondern auch die Möglichkeit, Hotspots in den Bildern zu platzieren, über die man sich weiterbewegen kann. Außerdem kann man sich auch um ein Objekt herumbewegen, wenn Bilder aus mehreren Perspektiven vorliegen.

Datenformat: Das Plugin nutzt das eigene weit verbreitete Quicktime-Format.

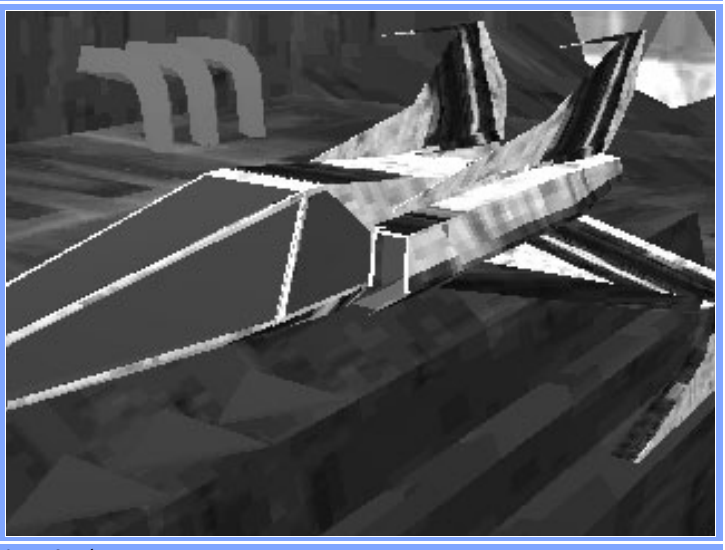
Authoring: Für das Erstellen der Filme muss eine spezielle Software verwendet werden.

Verbreitung: Quicktime ist schon seit sehr vielen Jahren am Markt und weit verbreitet.

Zusammenfassung

Quicktime ist schon sehr lange am Markt und war der Vorreiter für 3D-Panoramabilder. QuicktimeVR bietet weiterhin die Möglichkeit, sich über Hotspots weiterzubewegen und sich virtuell um Objekte herumzubewegen.

Anfy3D

Fakten	
Hersteller	Anfyteam
URL	*anfyteam.com/panfy3d.html*
	
Plattformen	Java-Applet
Installationsdateigröße	Java-Applet
Lizenzkosten	Shareware

Stärken

- Keine Installationsdatei (Java-Applet)
- Geringe Kosten

Anmerkungen

Darstellungsqualität: Das Plugin unterstützt zwar Antialiasing, die Darstellungsqualität kann aber nicht mit anderen Plugins mithalten.

Darstellungsgeschwindigkeit: Anfy3D ist komplett in Java implementiert und hat einen eigenen Renderer, der standardmäßig keine 3D-Hardware unterstützt. Die Darstellungsgeschwindigkeit ist dadurch nicht besonders hoch, in der Regel aber ausreichend. Es gibt allerdings auch die Möglichkeit, Hardwareunterstützung nachzuinstallieren.

Funktionalitäten: Interaktive Funktionalitäten müssen in Java oder JavaScript programmiert werden.

Dynamik: Dynamische Änderungen müssen in Java oder JavaScript programmiert werden.

Authoring: Für die Erstellung der Modelle und Animationen kann man jeden Editor benutzen, der VRML exportieren kann.

Plattformen: Als Java-Applet ist das Plugin prinzipiell plattformunabhängig.

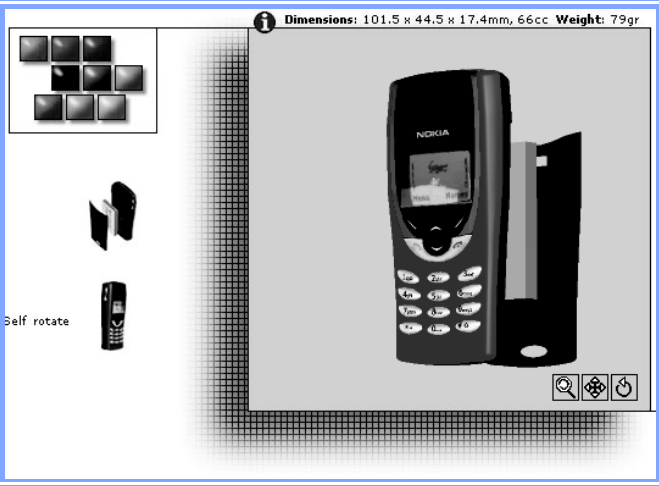
Verbreitung: Als Java-Applet wird das notwendige Plugin jedes Mal mit heruntergeladen.

Zusammenfassung

Anfy3D ist für den Preis als Shareware sehr leistungsfähig.

3Danywhere

Fakten	
Hersteller	3Di
URL	*www.3danywhere.com*



Plattformen	Java-Applet
Installationsdateigröße	Java-Applet
Lizenzkosten	Keine Informationen

Stärken

- Keine Installationsdatei (Java-Applet)
- Guter Editor
- Gute Darstellungsqualität

Anmerkungen

Darstellungsqualität: 3Danywhere bietet Antialiasing und gute Darstellungsfunktionalitäten.

Darstellungsgeschwindigkeit: 3Danywhere ist komplett in Java implementiert und hat einen eigenen Renderer, der standardmäßig keine 3D-Hardware unterstützt. Die Darstellungsgeschwindigkeit ist dadurch nicht besonders hoch, in der Regel aber ausreichend.

Funktionalitäten: Interaktive Funktionalitäten müssen in Java oder JavaScript programmiert werden. Der mitgelieferte Editor stellt allerdings auch die Erstellung einiger interaktiver Funktionalitäten zur Verfügung.

Dynamik: Dynamische Änderungen müssen in Java oder JavaScript programmiert werden.

Authoring: Der mitgelieferte Editor dient vor allem dem Zusammenstellen des Films inklusive interaktiver Funktionen. Für die Erstellung der Modelle und Animationen kann man jeden Editor benutzen, der VRML exportieren kann.

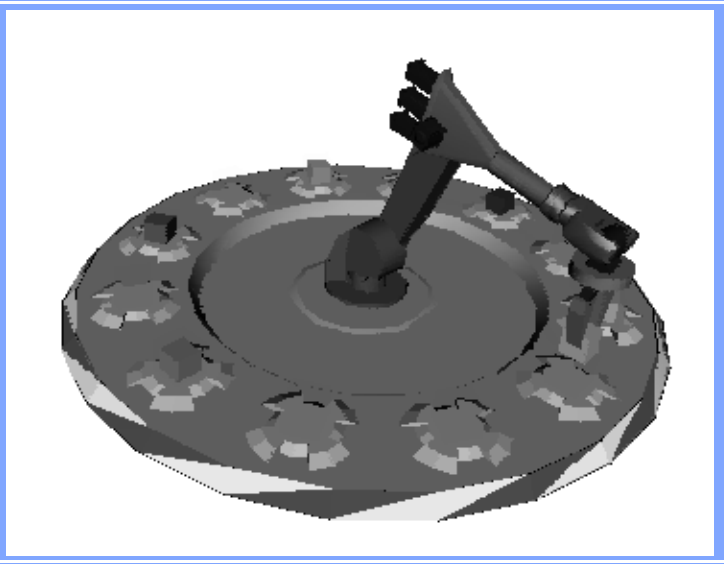
Plattformen: Als Java-Applet ist das Plugin prinzipiell plattformunabhängig.

Verbreitung: Als Java-Applet wird das notwendige Plugin jedes Mal mit heruntergeladen.

Zusammenfassung

3Danywhere ist vor allem durch die Möglichkeiten des mitgelieferten Editors interessant, der die Erstellung interaktiver Funktionalitäten erlaubt, ohne dafür programmieren zu müssen.

Critical Reach

Fakten	
Hersteller	Critical Reach, gegründet 1996 (Janet), Sitz in Deutschland
URL	*www.criticalreach.com*
	
Plattformen	Java-Applet
Installationsdateigröße	Java-Applet
Lizenzkosten	Keine Informationen

Stärken

- Keine Installationsdatei (Java-Applet)
- Spezialisierung auf Produktkataloge

Anmerkungen

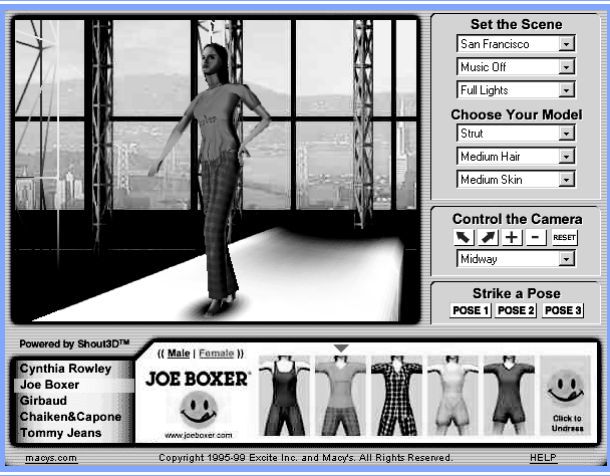
Plattformen: Als Java-Applet ist das Plugin prinzipiell plattformunabhängig.

Verbreitung: Als Java-Applet wird das notwendige Plugin jedes Mal mit heruntergeladen.

Zusammenfassung

Die Firma Critical Reach setzt die Fähigkeiten des Plugins vor allem für ihr Produkt Critical Reach Catalog ein, welches für Produkt- und Ersatzteilbestellungen das Zerlegen eines Produktes in Einzelteile und damit eine genauere Spezifikation des gewünschten Teiles erlaubt. Die 3D-Möglichkeiten des Produktes sind zwar nicht besser als vergleichbare Technologien, aber die Kombination mit dem Produktkatalog macht es für manche Einsatzgebiete interessant.

Shout3D

Fakten	
Hersteller	Shout Interactive
URL	*www.shout3d.com*
	
Plattformen	Java-Applet
Installationsdateigröße	Java-Applet
Lizenzkosten	195 USD pro Domäne und Jahr

Stärken

- Keine Installationsdatei (Java-Applet)
- Gute Referenzen

Anmerkungen

Darstellungsqualität: Shout3D bietet Antialiasing und gute Darstellungsfunktionalitäten.

Darstellungsgeschwindigkeit: Shout3D ist komplett in Java implementiert und hat einen eigenen Renderer, der standardmäßig keine 3D-Hardware unterstützt. Die Darstellungsgeschwindigkeit ist dadurch nicht besonders hoch, aber in der Regel ausreichend.

Funktionalitäten: Interaktive Funktionalitäten müssen in Java oder JavaScript programmiert werden. Es werden allerdings einige Basis-Applets mitgeliefert, die benötigte Funktionalitäten abdecken oder als Grundlage dienen können.

Datenformat: Shout3D benutzt ein eigenes Datenformat s3d oder das VRML-Format.

Dynamik: Dynamische Änderungen müssen in Java oder JavaScript programmiert werden.

Authoring: Am besten geeignet ist 3D Studio Max. Man kann aber jedes Programm benutzen, welches VRML exportieren kann.

Plattformen: Als Java-Applet ist das Plugin prinzipiell plattformunabhängig.

Verbreitung: Als Java-Applet wird das notwendige Plugin jedes Mal mit heruntergeladen.

Zusammenfassung

Shout3D ist schon sehr lange am Markt und auch sehr bekannt. Mehr darüber erfahren Sie im Java-Kapitel dieses Buches.

2.5 3D-Editoren

In diesem Abschnitt möchte ich noch ein paar Tools vorstellen, die bei der Erstellung von 3D-Grafiken und Animationen nützlich bzw. zum Teil unerlässlich sind. Dabei geht es vor allem um 3D-Editoren und Konvertierungsprogramme.

3D Studio Max und Plasma

Ich persönlich arbeite gerne mit 3D Studio Max (3DS Max) der Firma Discreet. Das Programm soll hier aber nur stellvertretend erwähnt werden für eine Reihe von professionellen 3D-Editoren, wie zum Beispiel Maya, Cinema4D und andere.

Besonders hervorzuheben ist, das Discreet im Juni 2002 eine neue Software names „Plasma“ vorgestellt hat. Plasma kostet nur rund ein Fünftel von dem, was 3D Studio Max kostet, es enthält aber alle für den Webbereich wichtigen Funktionalitäten. Vor allem kann es direkt Daten für Flash und Director erzeugen. Da die Final Version des Programms bei Drucklegung des Buches noch nicht vorlag, finden Sie weitere Informationen dazu auf der Website zum Buch *3d.franklamprecht.de*.

Im Allgemeinen enthalten 3D-Softwarepakete wie 3D Studio Max in der Regel drei verschiedene Komponenten:

- den eigentlichen Editor zum Erstellen von dreidimensionalen Objekten
- einen Animationsteil, mit dem komplexe Bewegungen oder Filme erstellt werden können
- den Renderer, der für eine realitätsnahe Darstellung einer Szene verantwortlich ist

Der 3D-Editor ist der Teil, der uns hier am meisten interessiert. Denn auch wenn die verschiedenen Technologien recht gut die 3D-Szenen darstellen können, so ist deren Erstellung ein komplizierter Akt. Ein guter Editor bietet deshalb zum Beispiel die folgenden Funktionalitäten:

- Import von vielen verschiedenen 3D-Dateiformaten, um Modelle, die man zum Beispiel im Internet gefunden hat, wiederverwenden zu können
- Extrusion von zweidimensionalen Formen (Buchstaben zum Beispiel)
- Boolesche Operationen auf 3D-Modellen (Vereinigung, Schnittmenge, Differenz)
- Komfortable Möglichkeiten zur Verformung der Modelle

Aufgrund des guten Preis-/Leistungsverhältnisses hat sich 3D Studio Max eine gute Position im Markt erarbeitet und inzwischen bieten die meisten Plugins eine mehr oder weniger enge Integration mit dem Editor an. Das kann bedeuten, dass das Dateiformat des Editors gelesen werden kann oder sogar, dass es ein Plugin für den Editor gibt, der direkt das benötigte Format exportieren kann.

Einen Screenshot von 3D Studio Max zeigt Abbildung 2.1.

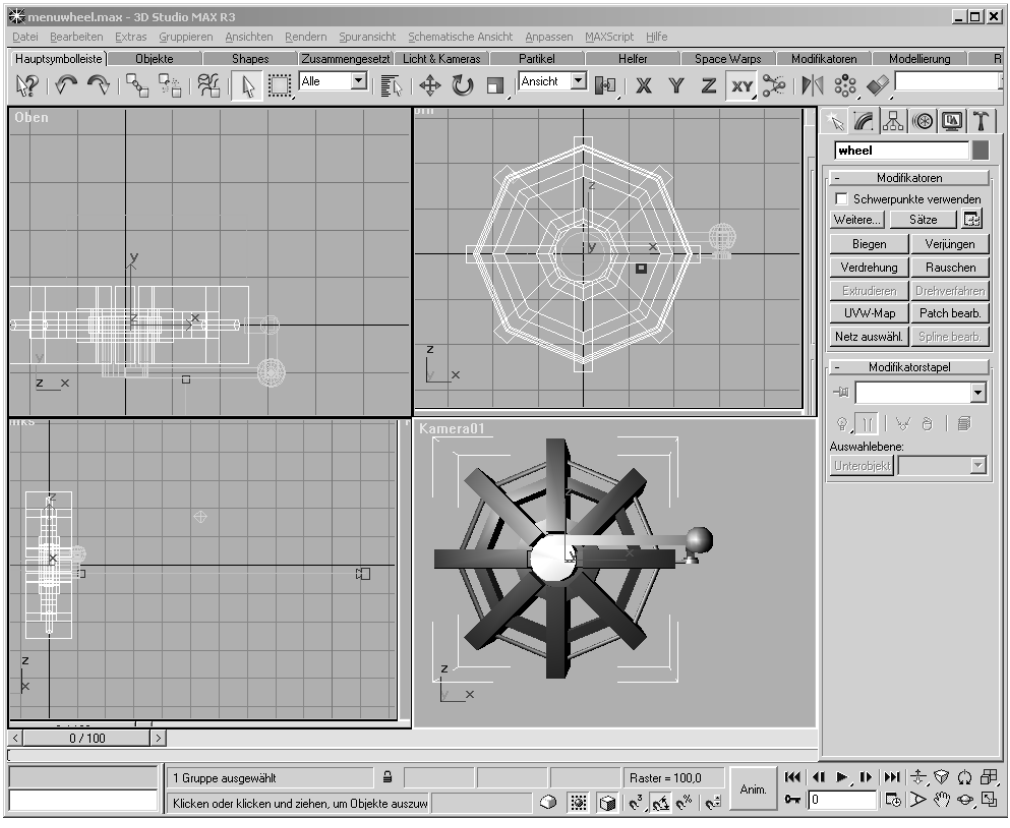


Abbildung 2.1: Der Editor in 3D Studio Max

Swift3D

Swift3D von electric rain ist eine Kombination von 3D-Editor und Konvertierungsprogramm. Der 3D-Editor ist nicht wirklich gut oder erwähnenswert, aber der Konvertierungsteil ist sehr interessant.

Swift ist in der Lage, im 3D-Editor erstellte oder aus 3D Studio Max importierte Modelle so zu konvertieren, dass sie im zweidimensionalen Flash-Vektorformat vorliegen. Hier bietet Swift vor allem auch die Möglichkeit, Verläufe innerhalb von Flächen zu erzeugen und damit den Realismus zu steigern.

Ein Beispiel mit Swift zeigt das Kapitel über Flash. In Abbildung 2.2 sehen Sie einen Screenshot von Swift3D.

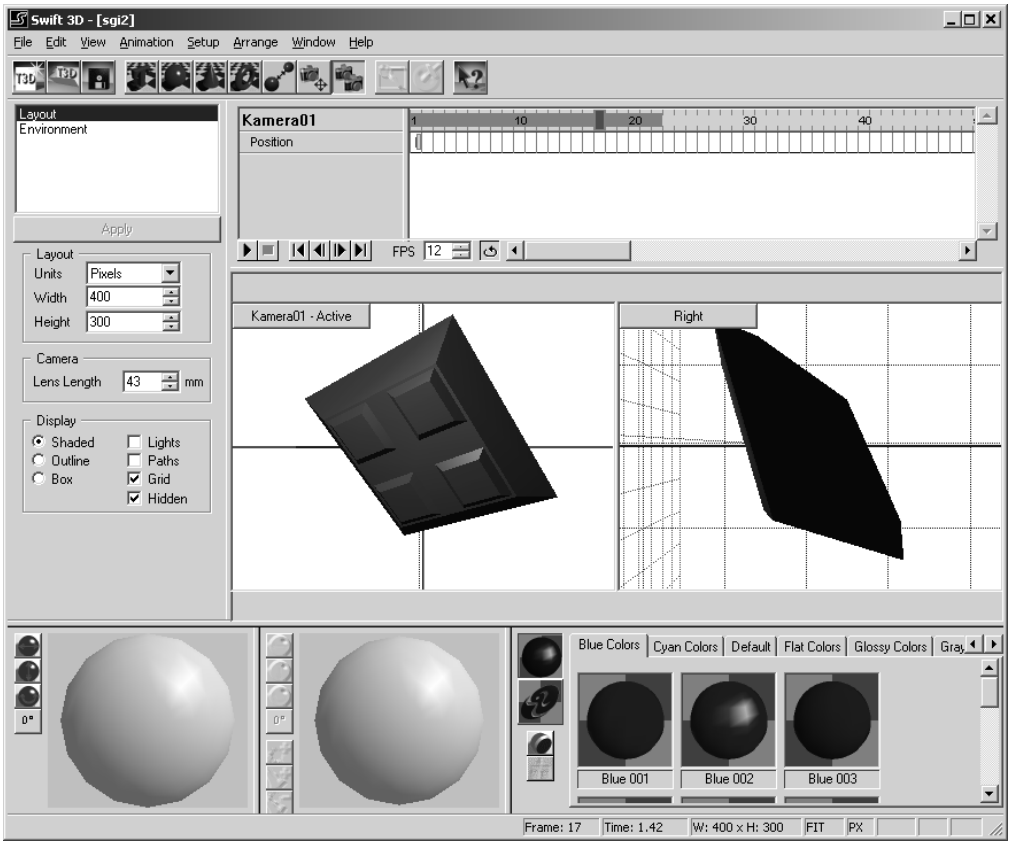


Abbildung 2.2: Swift3D

Blender

Blender ist ein völlig kostenloser 3D-Editor. Er ist mit Sicherheit nicht so komfortabel und umfangreich wie die professionellen Softwarepakete, aber vom Preis-/Leistungsverhältnis ist er natürlich unschlagbar. Sie können damit unter anderem VRML-Dateien exportieren, die von vielen Plugins weiterverarbeitet werden können.

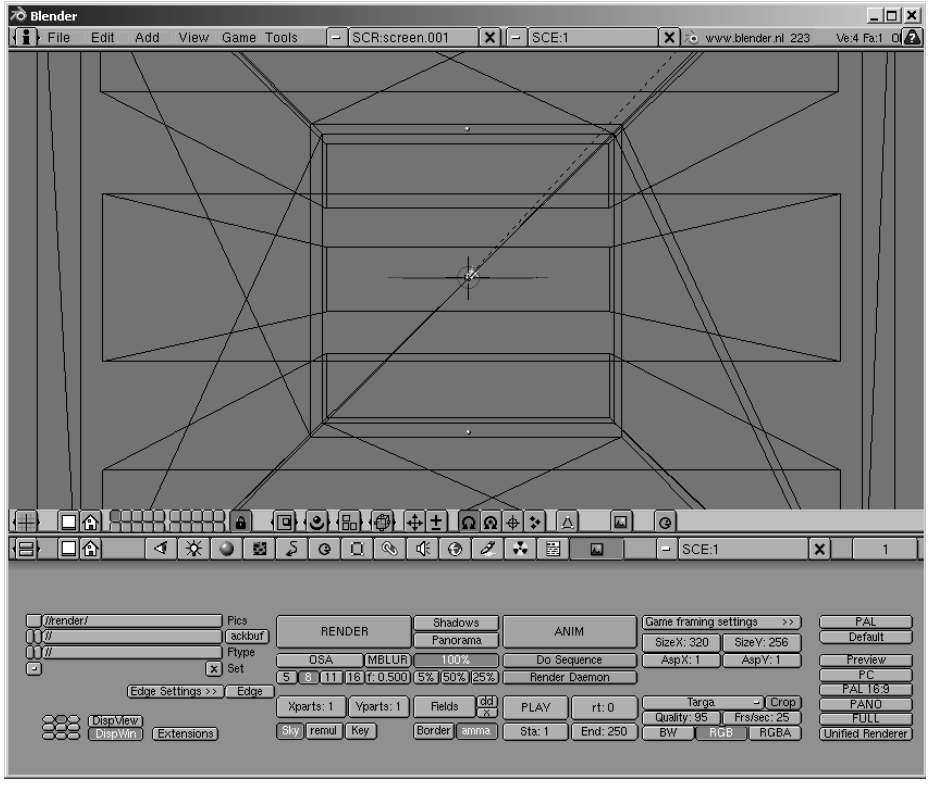


Abbildung 2.3: Das Blender-Interface

Leider musste der Hersteller des Editors vor kurzem Konkurs anmelden. Es bleibt zu hoffen, dass sich ein Geldgeber findet, mit dessen Hilfe die Entwicklung weitergeht.

URLs

www.discreet.com/products/3dsmax	3D Studio Max Website
www.discreet.com/products/plasma	Plasma Website
www.swift3d.com	Swift3D Website
www.blender3d.com	Blender Website

DHTML

Es gibt verschiedene Definitionen, was DHTML (Dynamic HTML) genau bedeutet. In diesem Kapitel ist damit das Zusammenspiel zwischen JavaScript und dem DOM (Document Object Model) einer HTML-Seite gemeint (siehe nächster Abschnitt). Die Möglichkeiten, die man damit an die Hand bekommt, um eindrucksvolle Interaktionen mit einer HTML-Seite zu ermöglichen, haben ständig zugenommen. Aber erst seit den Browserversionen ab 4.x (Netscape und MS Internet Explorer) sind sie in einem Stadium, dass man damit verlässlich arbeiten kann.

Die grundlegenden Werkzeuge zur Darstellung dreidimensionaler Grafiken mit DHTML sind der Austausch von Bildern zur Simulation von Bewegung und die Nutzung von Ebenen, um Verdeckungen und Tiefe darzustellen. Es ist nicht möglich, Grafikelemente wie Linien oder Flächen direkt mit JavaScript zu erzeugen. (Man kann allerdings mit ein paar Tricks doch den Eindruck dynamisch generierter Linien erzeugen. Aber dazu später mehr.)

Für die Visualisierung von 3D-Grafik ist DHTML zwar nicht wirklich die beste Technologie. Doch DHTML ist die Technologie, die am Markt die größte Verbreitung besitzt. Außerdem gibt es in manchen größeren Firmen Richtlinien, die die Installation von Plugins auf Arbeitsplatzrechnern verbieten. Wenn die Zielgruppe einer Website nun innerhalb einer solchen Firma zu finden ist, können diese Personen Webseiten, die ein Plugin verwenden, nicht benutzen. Deshalb lohnt sich eine Beschäftigung mit den „Bordmitteln“ der Browser auf jeden Fall.

Die Beispiele sind ohne weitere Bearbeitung nur für den Internet Explorer geeignet.

Die Beispiele in diesem Kapitel sind programmiert für den MS Internet Explorer ab Version 5, da dieser am Markt die größte Verbreitung hat. Die Beispiele lassen sich aber auch so anpassen, dass sie mit Netscape oder Internet Explorer ab Version 4 zusammenarbeiten. Allerdings erfordert dies die Behandlung von so vielen Sonderfällen, dass es vom eigentlichen Kern der Beispiele zu sehr ablenken würde. Am Ende dieses Kapitels werde ich kurz auf die Besonderheiten eingehen, die zu beachten sind, wenn man die Beispiele Cross-Browser-fähig machen möchte. Unter der Linksammlung finden Sie außerdem Quellen, wie man Cross-Browser-Funktionalität recht einfach kapseln kann.

3.1 Grundlagen

In diesem Abschnitt möchte ich kurz die DHTML-Techniken darlegen, die die folgenden Beispiele verwenden werden. Für eine detaillierte Einführung muss ich Sie aber auf entsprechende Spezialbücher verweisen.

HTML ist eine struktierte Beschreibungssprache zur Darstellung von statischen Inhalten. Ähnlich wie ein XML-Dokument ist HTML eine hierarchische Beschreibung mit Hilfe so genannter „Tags“, die jeweils eine spezielle Bedeutung haben und ein Inhaltsteil umschließen. Eine Überlagerung mit Elementen auf verschiedenen Ebenen ist mit reinem HTML nicht möglich.

Cascading Style Sheets (CSS) sind eine Erweiterung für HTML, die unter anderem die Möglichkeit mitbringen, Ebenen zu definieren, mit deren Hilfe sich HTML-Bestandteile übereinander lagern und pixelgenau platzieren lassen. Die gängige Art und Weise, eine Ebene zu definieren, erfolgt mit dem DIV-Tag, welches zunächst nur die Aufgabe hat, mehrere HTML-Elemente zusammenzufassen. Diesem DIV-Tag kann man nun aber noch weitere Eigenschaften zuweisen, zum Beispiel eine absolute Position auf der Seite, eine Stapeltiefe und eine eigene Hintergrundfarbe.

JavaScript ist eine Programmiersprache, die in eine HTML-Seite eingebettet werden kann. Sie wird während des Aufbaus der Seite ausgeführt oder zu bestimmten Ereignissen, etwa wenn der Mauszeiger einen bestimmten Bereich überfährt (Rollover-Ereignis). JavaScript ist von der Syntax her Java sehr ähnlich, daher auch der Name. Mit JavaScript ist auch objektorientierte Programmierung möglich. Dies erfolgt allerdings auf komplett andere Art und Weise als bei Java.

Das **Document Object Model** (DOM) ist eine Art Programmierschnittstelle, um auf die HTML-Objekte einer Seite zugreifen zu können. Damit können einzelne Eigenschaften der Objekte auch nach dem Aufbau der Seite noch geändert werden, zum Beispiel die Position einer Ebene oder der Inhalt eines Bildes. Die Unterstützung für die Zugriffsmöglichkeiten gehen dabei mit den neueren Browserversionen immer weiter.

Die Techniken, die nun im Folgenden vor allem verwendet werden, sind die Benutzung von Ebenen und das Austauschen von Bildern, wie es schon im nächsten Beispiel zu sehen ist.

3.2 Beispiel: Aufklappmenü

In diesem Abschnitt möchte ich ein Beispiel vorstellen, welches das Prinzip der vorberechneten Einzelbilder in Zusammenhang mit einem Animated-GIF benutzt.

Ziel des Beispiels, welches wir hier in DHTML programmieren wollen, soll ein Ausklappmenü sein, wie es die Bildfolge in Abbildung 3.1 zeigt. Sie finden das Beispiel auf der CD im Verzeichnis *dhtml\beispiel_anigif*. Die einzelnen Kästchen im Bild ganz links können drei Auswahlmöglichkeiten eines Menüs darstellen. Beim Klick auf ein Kästchen öffnet sich das entsprechende Untermenü mit weiteren Auswahlmöglichkeiten. Ein Klick außerhalb des geöffneten Untermenüs schließt dieses wieder.

Abbildung 3.1: Bildfolge eines Ausklappmenüs

Vorbereitung der Bildsequenz

Animated-GIFs sind eine Abfolge von einzelnen Bildern, die mit einer vordefinierten Geschwindigkeit abgespielt wird. Man hat dabei die Möglichkeit, für jedes einzelne Bild die Zeit anzugeben, die verstreicht, bis das nächste Bild gezeigt wird. Weiterhin kann festgelegt werden, ob die Bildfolge nur einmal gezeigt wird oder ob die Animation ständig wiederholt wird. Abbildung 3.2 zeigt die Definition eines Animated-GIF in Adobe ImageReady. ImageReady ist ein Bestandteil von Photoshop. Zur Erstellung von Animated-GIFs gibt es aber eine ganze Reihe von Programmen. Die einzelnen Bilder wurden mit einem 3D-Animationsprogramm vorberechnet und dann importiert. Wichtig ist bei dem GIF, dass ein transparenter Hintergrund gewählt wurde.

Platzverbrauch optimieren

Ein Vorteil bei der Speicherung der Bildfolge im GIF-Format ist der Platzverbrauch. Man könnte die Bilder auch einzeln speichern und mit JavaScript abspielen. Allerdings erlaubt es die Speicherung als GIF, dass gleiche Bildteile in der gesamten Folge nur einmal gespeichert werden müssen. Die entsprechende Komprimierung übernimmt das Programm zur Erstellung des Animated-GIF.

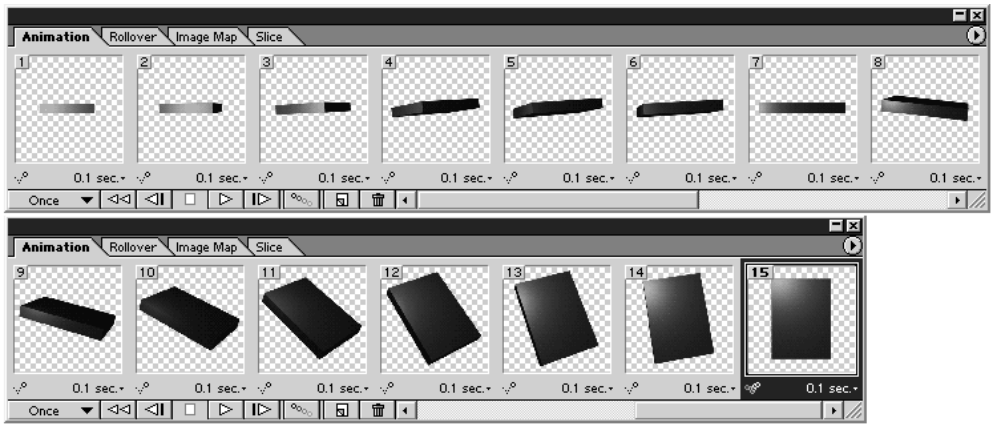


Abbildung 3.2:
Ein Animated-GIF in ImageReady

Um die Ladezeiten gering zu halten, wird für den Animationsteil der einzelnen Untermenüs immer dasselbe Animated-GIF benutzt. Die tatsächlichen Menüpunkttexte des Untermenüs können dann auf einer extra Ebene darüber gelegt werden.

Der Quellcode

Beginn des Listings zu beispiel_anigif/index.htm

Der Quellcode zu dem Beispiel sieht nun folgendermaßen aus:

```
<html>
<head>
    <title>Menue</title>
</head>

<script>
<!--
var currentMenue = 0;
```

In der Variable `currentMenue` wird die Nummer des aktuell geöffneten Menüs gespeichert, damit darauf beim Schließen des Untermenüs zugegriffen werden kann.

```
function swapImage (layername, picname, filename) {
    eval("document." + picname + ".src = filename;");
}
```

Diese Funktion tauscht in der angegebenen Ebene das Bild mit dem angegebenen Namen gegen das Bild mit dem angegebenen Dateinamen aus.

```
function openMenue(nr) {
    swapImage("menue" + nr, "bild" + nr,"fill.gif");
    document.all.menueSub.style.top = 50*nr + 13;
    swapImage("menueSub", "bildSub", "menue.gif");
    currentMenue = nr;
}
```

Diese Funktion wird aufgerufen, wenn ein Untermenü geöffnet werden soll. Dazu wird nur die Nummer des entsprechenden Menüpunktes (siehe unten) übergeben. Als Erstes wird dann das Bild des Kästchens durch ein unsichtbares Füllbild (*fill.gif*) ersetzt. Als Nächstes wird die Ebene mit dem generischen Untermenü an die richtige Position verschoben und das enthaltene Bild durch das Animated-GIF ersetzt. Dadurch wird dann auch das Abspielen der Bildfolge ausgelöst.

```
function closeMenue() {
    swapImage("menue" + currentMenue, "bild" +
                currentMenue,"menue_a.gif");
    document.all.menueSub.style.top = -300;
    swapImage("menueSub", "bildSub", "fill.gif");
}
```

Die Funktion ist das Pendant zu der öffnenden Funktion.

```
-->
</script>

<body bgcolor="#ffffff">
```

Der eigentliche HTML-Layout-Teil folgt hier.

```
<div id="menue0" style="position:absolute; left:70;
                        top:120;z-Index:1">
  <a href="#" onMouseUp="openMenue(0)">
    <img name="bild0" src="menue_a.gif" width=131
         height=25 border=0>
  </a>
</div>
```

Die Ebenen für die Anzeige der Menüpunkte sind für alle drei Menüpunkte gleich, bis auf die jeweiligen Bezeichner. Die Beschriftungen werden am einfachsten mit einer Unterebene darüber gelegt. Auf Mausklick wird dann die Funktion zum Öffnen des Untermenüs aufgerufen.

```
<div id="menue1" style="position:absolute; left:70;
                        top:170;z-Index:2">
  <a href="#" onMouseUp="openMenue(1)">
    <img name="bild1" src="menue_a.gif" width=131
         height=25 border=0>
  </a>
</div>

<div id="menue2" style="position:absolute; left:70;
                        top:220;z-Index:3">
  <a href="#" onMouseUp="openMenue(2)">
    <img name="bild2" src="menue_a.gif" width=131
         height=25 border=0>
  </a>
</div>
```

Die folgende Ebene wird generisch für alle Untermenüs verwendet, indem sie bei Bedarf an die richtige Stelle verschoben wird. Ansonsten wird sie außerhalb des sichtbaren Bereichs platziert.

Ende des Listings zu beispiel_anigif/index.htm

```
<div id="menueSub" style="position:absolute; left:16;
                          top:-300;z-Index:10">
  <a href="#" onMouseUp="closeMenue()">
    <img name="bildSub" src="fill.gif" width=250
         height=244 border=0>
  </a>
</div>

</body>
</html>
```

Sie können jetzt eines der Kästchen anklicken, worauf sich ein Untermenü öffnet.

Nachdem wir nun also die Grundlagen für das 3D-Menü geschaffen haben, werden wir als Nächstes beispielhaft das Ganze um die fehlenden Elemente erweitern, damit man auch wirklich ein beschriftetes Menü hat.

Die Beschriftungen

Als Erstes wird dazu über jedes der Kästchen mittels einer eigenen Ebene ein Text gelegt. Die entsprechende Zeile HTML sieht so aus:

```
<div id="menue0a" style="position:absolute; left:10;
                         top:5;">
<a href="#" onMouseUp="openMenue(0)">Informationen</a>
</div>
```

Diese Ebene wird innerhalb der Kästchen-Ebene platziert, die Positionsangaben sind also relativ zur linken oberen Ecke der Kästchen-Ebene. Die `href`-Angabe enthält nur das Gatterzeichen, öffnet also keine neue Seite, sondern nur das Untermenü. Der entsprechende Befehl folgt in der `MouseUp`-Anweisung.

Die drei Kästchen-Ebenen stellen sich nun wie folgt dar:

```
<div id="menue0" style="position:absolute; left:70;
                        top:120;z-Index:1">
  <a href="#" onMouseUp="openMenue(0)">
    <img name="bild0" src="menue_a.gif" width=131
         height=25 border=0>
  </a>
<div id="menue0a" style="position:absolute; left:10;
                         top:5;">
<a href="#" onMouseUp="openMenue(0)">Informationen</a>
</div>
</div>
```

```
<div id="menue1" style="position:absolute; left:70;
                        top:170;z-Index:2">
  <a href="#" onMouseUp="openMenue(1)">
    <img name="bild1" src="menue_a.gif" width=131
         height=25 border=0>
  </a>
<div id="menue1a" style="position:absolute; left:10;
                         top:5;">
<a href="#" onMouseUp="openMenue(1)">Produkte</a>
</div>
</div>

<div id="menue2" style="position:absolute; left:70;
                        top:220;z-Index:3">
  <a href="#" onMouseUp="openMenue(2)">
    <img name="bild2" src="menue_a.gif" width=131
         height=25 border=0>
  </a>
<div id="menue2a" style="position:absolute; left:10;
                         top:5;">
<a href="#" onMouseUp="openMenue(2)">Services</a>
</div>
</div>
```

Damit haben wir die drei blauen Kästchen beschriftet mit den Menüpunkten „Informationen“, „Produkte“ und „Services“. Um die Beschriftung etwas ansprechender zu gestalten fügen wir in den Kopfbereich der Seite noch folgende Stilangaben ein:

```
<style type="text/css">
<!--
a {font-family: arial, helvetica, sans-serif;
   font-size: 12px; color:#ffffff; font-weight:bold;
   text-decoration: none;}
a:hover {font-family: arial, helvetica, sans-serif;
         font-size: 12px; color:#ffff00;
         font-weight:bold; text-decoration: none;}
-->
</style>
```

Dadurch stellen sich die Menüpunkte als weißer Text ohne Unterstrich dar und verändern ihre Farbe bei einem Rollover zu Gelb.

Der HTML-Code für das Untermenü

Kommen wir zum Untermenü. Wir werden hier nur ein Untermenü für alle drei Menüpunkte verwenden. Dieses Untermenü verschieben wir genauso wie die Filmsequenz jeweils an die richtige Position und schalten die Sichtbarkeit jeweils an und aus. Über die Sichtbarkeit könnte man dann auch verschiedene Untermenüs realisieren.

Das HTML für das Untermenü wird ebenfalls als eine zusätzliche Ebene über die Filmsequenz gelegt:

```
<div id="menueSub" style="position:absolute; left:16;
                          top:-300;z-Index:10">
  <a href="#" onMouseUp="closeMenue()">
    <img name="bildSub" src="fill.gif" width=250
         height=244 border=0>
  </a>
<div id="menue0b" style="position:absolute; left:70;
                         top:40;visibility:hidden">
   <a href="#">Untermen&uuml;1</a><br><br>
   <a href="#">Untermen&uuml;2</a><br><br>
   <a href="#">Noch ein Punkt</a><br><br>
   <a href="#">Was anderes</a><br><br>
   <a href="#">Das Letzte</a><br><br>
</div>
</div>
```

Sie sehen bei der Stilangabe der Ebene das Attribut `visibility:hidden`, wodurch die Sichtbarkeit zunächst ausgeschaltet wird. Die href-Angaben der einzelnen Unterpunkte enthalten nur ein Gatter. Hier müssten nun die tatsächlichen Verweise für die anderen Seiten angegeben werden.

Die Sichtbarkeit des Untermenüs wird eingefügt.

Um die Sichtbarkeit der Untermenüpunkte zu steuern fehlen noch die entsprechenden Angaben in den jeweiligen Funktionen. Diese sehen nach der Erweiterung so aus:

```
function openMenue(nr) {
    swapImage("menue" + nr, "bild" + nr,"fill.gif");
    document.all.menueSub.style.top = 50*nr + 13;
    swapImage("menueSub", "bildSub", "menue.gif");
    currentMenue = nr;
    window.setTimeout(
    "document.all.menue0b.style.visibility ='visible';"
                      ,1500);
}

function closeMenue() {
    document.all.menue0b.style.visibility = 'hidden';
    swapImage("menue" + currentMenue,
              "bild" + currentMenue,"menue_a.gif");
    document.all.menueSub.style.top = -300;
    swapImage("menueSub", "bildSub", "fill.gif");

}
```

Bei der öffnenden Funktion versteckt sich die Sichtbarkeits-Angabe in der `setTimeout`-Anweisung. Dies bewirkt, dass die entsprechende Ebene erst nach 1,5 Sekunden angezeigt wird. So lange braucht die Filmsequenz, bis das Untermenü tatsächlich geöffnet ist. Bei der schließenden Funktion kann das Untermenü sofort verschwinden.

Mit diesem Beispiel verfügen Sie nun über alles, was Sie benötigen, um ein komplettes Navigationskonzept zu programmieren.

3.3 Beispiel: Objektpräsentation

Das Beispiel, welches in diesem Abschnitt als Erläuterungsgrundlage dient, ist manchem vielleicht schon als „QuicktimeVR“ der Firma Apple bekannt. Hierbei soll mit Hilfe einer Reihe von Fotos der Eindruck erzeugt werden, dass man sich einen Gegenstand von allen Seiten betrachten kann. Apples QuicktimeVR beinhaltet allerdings noch eine andere Technik, welche ein Panoramabild darstellt, innerhalb dessen sich der Nutzer drehen kann. Diese letztgenannte Technik wird hier nicht betrachtet.

Informationen zu QuicktimeVR sind im Kapitel Übersicht zu finden.

Die hier vorgestellte Technik findet vor allem in zwei Bereichen Verwendung: zur Präsentation eines Gegenstandes und zur Visualisierung eines Raumes bzw. einer Landschaft. Das Beispiel, welches ich hier entwickeln möchte, bezieht sich auf den ersten Fall. Der „Gegenstand“ für dieses Beispiel ist eine Person auf einem Drehstuhl, um die man sich drehen kann. Realisiert wird dies, indem man in das Bild klickt und bei gedrückter Maustaste die Maus nach links oder rechts bewegt.

Die Vorbereitung der Bilder

Die Grundlage für das Beispiel bildet die Bildfolge in Abbildung 3.3.

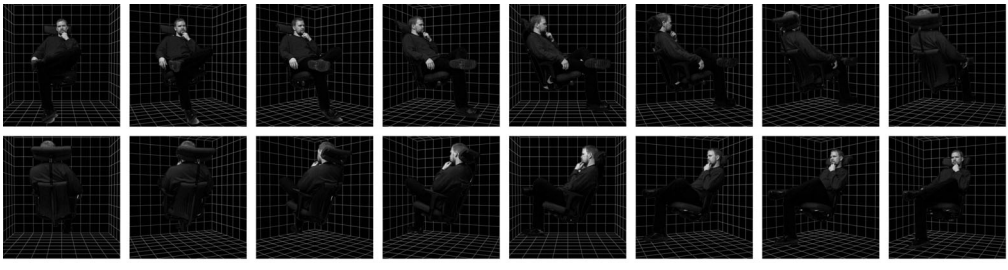


Abbildung 3.3: Bildsequenz

Das Praktische an dieser Bildfolge ist, dass man sie recht einfach in guter Qualität erstellen kann. Die Kamera wird auf einem Stativ oder einer festen Grundlage fixiert. Der Hintergrund hinter der Person sollte eine einheitliche Farbe haben, damit man das Objekt später leicht mit einem Grafikprogramm freistellen (den Hintergrund löschen) kann. Anschließend muss für die einzelnen Bilder nur noch der Stuhl um ein paar Grad gedreht und ein einzelnes Bild aufgenommen werden. Ich habe hier 16 einzelne Positionen festgehalten, d.h. die Person musste für jedes Bild um 360/16= 22,4 Grad weitergedreht werden. 16 Bilder sind schon eine recht feine Abstufung, bei der die Drehung ziemlich flüssig wahrgenommen wird.

Beachten Sie die Gesamtgröße aller zu ladenden Bilder.

Für die Anzahl der Bilder ist natürlich die akzeptable Menge an herunterzuladenden Daten zu berücksichtigen. Für dieses Beispiel haben die Bilder eine Größe von etwa 400x400 Pixeln und jedes Bild eine Dateigröße von etwa 40 KB. Das bedeutet, dass etwa 640 KB übertragen werden müssen. Um auf solch eine Datenmenge zu warten, muss der Nutzer schon einen guten Grund haben (wobei solch eine ansprechende Produktpräsentation natürlich ein guter Grund sein kann). Wenn es bei der Darstellung weniger um weiche Übergänge und „optischen Anspruch" geht, sondern vielmehr um harte Fakten („So sieht das Auto von links vorne aus und so von hinten"), sollten weniger Bilder verwendet werden. Bei dem Beispiel können Sie an entsprechender Stelle (siehe unten) mal ausprobieren, wie die Darstellung mit weniger Bildern aussieht.

Um die Bilder optisch noch etwas interessanter zu gestalten, wurde nach dem Freistellen der Person ein gerasterter Hintergrund eingefügt. Dieser wurde mit einem 3D-Programm vorberechnet. Für den Hintergrund brauchten allerdings nicht 16 Bilder berechnet zu werden, sonder nur 16/4=4 Bilder, da die vier Seiten dieses virtuellen Raumes alle gleich aussehen sollten. Die Bilder und die Beispielseite finden Sie auf der CD im Verzeichnis *dhtml\beispiel_vr*.

Der Quellcode

Das Beispiel ist zunächst sehr einfach gehalten, um die grundlegenden Funktionen zu verdeutlichen. Anschließend gehe ich dann noch auf die Erweiterungsmöglichkeiten ein.

Die Bilder, die mit einer Digitalkamera aufgenommen und mit Photoshop nachbearbeitet wurden, liegen alle in einem Unterverzeichnis namens *images/* und sind benannt mit *fro.jpg – fr15.jpg* .

Beginn des Listings zu beispiel_vr/index.htm

```
<html>
<head>
     <title>turn</title>
</head>
```

Bis jetzt noch nichts Spannendes. Die HTML-Seite trägt den Titel „turn".

```
<script>
var img = new Array(20);
var ino = 15;

// preload the images
for (i = 0;i < ino; i++) {
    img[i] = new Image();
    img[i].src = "images/fr" + i + ".JPG";
}
```

Um sicherzustellen, dass die Darstellung verzögerungsfrei vonstatten geht, werden alle Bilder vorgeladen.

```
var dragging = false;
var akt = 0;
var oldX = 0;
document.onmousemove = mouseMove;
document.onmousedown = mouseDown;
document.onmouseup = mouseUp;
window.onmousemove = mouseMove;
window.onmousedown = mouseDown;
window.onmouseup = mouseUp;

function mouseDown (e) {
     var x = event.offsetX;
     var y = event.offsetY;
    if ((x > 100) && (x < 500) && ( y > 100) &&
        (y < 500))
        dragging = true;
}

function mouseUp(e) {
    dragging = false;
}

function mouseMove(e)
{
     var x = event.offsetX;
     var y = event.offsetY;

    if (oldX == -9999) {
        oldX = x;
        return;
    }
    if (dragging) {
        if ((oldX > x) && (oldX - x > 10)) {
            oldX = x;
            updateImg(-1);
        }
        if ((oldX < x) && (x - oldX > 10)) {
            oldX = x;
            updateImg(1);
        }
    }
     return;
}
```

Der Code in diesem Block sorgt dafür, dass die Mausbewegungen erfasst und entsprechend verarbeitet werden.

`MouseDown` setzt ein Flag namens `dragging` auf wahr, wenn innerhalb des Bildbereiches die linke Maustaste gedrückt wurde.

`MouseUp` setzt das Flag entsprechend wieder auf falsch, wenn die Maustaste losgelassen wurde.

`MouseMove` wird aufgerufen, sobald sich die Maus bewegt. Wenn dabei noch das Flag auf wahr steht, sollte das Objekt gedreht werden. Das geschieht allerdings nur, wenn die Maus sich um mehr als 10 Pixel bewegt hat. Mit diesem Wert legt man fest, wie empfindlich das Objekt mit einer Drehung auf Mausbewegungen reagiert.

```
function updateImg(dir) {
    akt = akt + dir;
    if (akt < 0) akt = ino - 1;

    if (akt >= ino) akt = 0;

    document.mainImg.src = img[akt].src;

}
```

Diese Funktion bewegt die Filmsequenz in die eine oder andere Richtung (abhängig vom Übergabeparameter). Wenn ein Ende der Sequenz erreicht ist, springt die Funktion an das andere Ende. Das darzustellende Bild wird dann aus dem Zwischenspeicher geladen.

```
</script>
<style type="text/css">
<!--
#turnDIV {position:absolute; left:100; top:100;
          width:400; height:440;z-Index:0}
#turnOverlayDIV {position:absolute; left:100; top:100;
                 width:400; height:420;  background-
                 image:url("images/frame.gif"); z-Index:1}
-->
</style>
```

Das Bild wird in einer extra Ebene dargestellt. Dies wäre nicht unbedingt notwendig, erlaubt dafür aber die genaue Platzierung.

Ende des Listings zu beispiel_vr/index.htm

```
<body bgcolor="#000000" text="#6666ff">

<div id="turnDIV">
<img src="images/fr0.JPG"  border=0 name="mainImg">
<br>click and drag in the image.
</div>
<div id="turnOverlayDiv" onMouseOut="dragging = false;"
                         onMouseDown="dragging = true;"
                         onMouseUp="dragging = false;" >
 </div>

</body>
</html>
```

Erweiterung

Zusätzlich zu einer einzigen Runde um den darzustellenden Gegenstand herum kann man noch weitere Winkel fotografieren (siehe Abbildung 3.4).

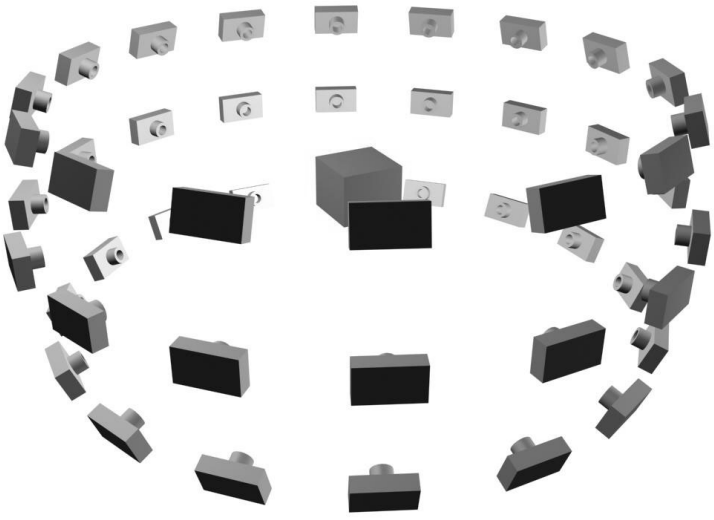


Abbildung 3.4: Ein Objekt kann von mehreren vertikalen Winkeln aus fotografiert werden.

Diese zusätzlichen Freiheitsgrade können mehr Überblick über den dargestellten Gegenstand bieten oder einfach auch sehr wirkungsvoll sein.

Im Gegensatz zu der einfachen Drehung, die die oben beschriebene HTML-Seite ausführt, ist für die zusätzlichen vertikalen Winkel weitere Programmlogik notwendig. Die vertikale Position muss vorgemerkt werden und die Mausbewegungen in der vertikalen Richtung müssen berücksichtigt werden. Im Folgenden sind nur die Programmteile aufgezeigt, die sich von der letzten Version unterscheiden.

```
<script>
var img = new Array(20);
var anzahlX = 8;
var anzahlY = 3;
var dragging = false;
var aktX = 4;
var aktY = 1;
var oldX = 0;
var oldY = 0;
```

Die vertikale Position wird in einer neuen Variable festgehalten und genauso werden auch die möglichen vertikalen Freiheitsgrade gespeichert (`anzahlY`).

```
for (j = 0;j < anzahlY; j++) {
   for (i = 0;i < anzahlX; i++) {
       img[i + j * anzahlX] = new Image();
       img[i + j * anzahlX].src = "images2/img" + i +
                                  "_" + j + ".JPG";
   }
}
```

Das Vorladen der Bilder muss jetzt die zusätzlichen Bilder berücksichtigen. Für dieses Beispiel werden 24 Bilder benutzt, 8 Bilder pro Umrundung und 3 unterschiedliche Winkel in der Vertikalen. Bei der Erstellung der Bilder erhielten diese einen Zusatz im Dateinamen „_0“, „_1“ oder „_2“, die die unterschiedlichen Bildfolgen für die unterschiedlichen vertikalen Winkel markieren.

Das Abfragen der Mausbewegungen

```
function mouseDown () {
     var x = event.offsetX;
     var y = event.offsetY;
   dragging = true;
   oldY = y;
   oldX = x;
}

function mouseMove()
{
     var x = event.offsetX;
     var y = event.offsetY;
   var dirX = 0;
   var dirY = 0;

    if (dragging) {
        if ((oldX > x) && (oldX - x > 40)) {
            oldX = x;
            dirX = 1;
        }
        if ((oldX < x) && (x - oldX > 40)) {
            oldX = x;
            dirX = -1;
        }
        if ((oldY > y) && (oldY - y > 80)) {
            oldY = y;
            dirY = 1;
        }
        if ((oldY < y) && (y - oldY > 80)) {
            oldY = y;
            dirY = -1;
        }
        updateImg(dirX,dirY);
    }
     return;
}
```

Bei den Mausbewegungen werden jetzt auch die vertikalen Bewegungen abgefangen. Da es jedoch nur drei verschiedene Positionen in der Vertikalen gibt, wird die Empfindlichkeit für die Erfassung der Mauspositionen geringer eingestellt, d.h. es wird nur bei einer Änderung um mehr als 80 Pixel die Position verändert.

```
function updateImg(dirX,dirY) {
    aktX = aktX + dirX;
    if ((aktY > 0) && (dirY < 0)) aktY = aktY + dirY;
    if ((aktY < anzahlY - 1) && (dirY > 0))
                                    aktY = aktY + dirY;
    if (aktX < 0) aktX = anzahlX - 1;
    if (aktX >= anzahlX) aktX = 0;
    document.mainImg.src =
                        img[aktX + aktY * anzahlX].src;
}
</script>
```

Die Funktion zur Änderung des Bildes wurde so angepasst, dass nun beide Rotationsachsen berücksichtigt werden.

Abbildung 3.5: Die einzelnen Bilder des zweiten Beispiels

Die erweiterte Darstellung mit mehr Freiheitsgraden bietet sich vor allem dann an, wenn man z.B. ein Produkt darstellen möchte. Dadurch, dass der Nutzer sich das Produkt von allen Seiten ansehen kann, gewinnt er mehr Vertrauen, dass es auch genau das ist, was er haben möchte. Je nach Art des Produktes ist der potenzielle Käufer dann auch bereit, längere Wartezeiten für den Download in Kauf zu nehmen.

3.4 Beispiel: Molekül

Im vorangegangenen Beispiel haben wir nur ein stillstehendes Bild ausgetauscht, um den 3D-Eindruck zu erzeugen. Für das nächste Beispiel soll das Austauschen eines Bildes höchstens noch notwendig sein, um eine Ansicht zu korrigieren. Ansonsten wollen wir den 3D-Eindruck durch Bewegung erzeugen.

Dazu benutzen wir einzelne Objekte (im Folgenden Sprites genannt). Diese Sprites platzieren wir jeweils auf einer eigenen Ebene und diese Ebenen bewegen wir dann.

Für dieses Beispiel wird es notwendig sein, dass Sie mit den Grundlagen aus Kapitel 1 vertraut sind. Wenn Sie dies also noch nicht gelesen haben, wäre es jetzt ein guter Zeitpunkt, das nachzuholen (es sei denn, Sie kennen sich mit Vektoren, 3D-Drehungen und Projektion schon aus).

Das Grund-Sprite

Als Vorlage für dieses Beispiel möchte ich eine Art Molekül auf der Webseite darstellen und drehen. Die einzelnen Atome des Moleküls werden dabei vereinfacht als Kugeln dargestellt.

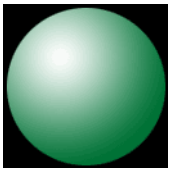


Abbildung 3.6: Das Sprite des Beispiels

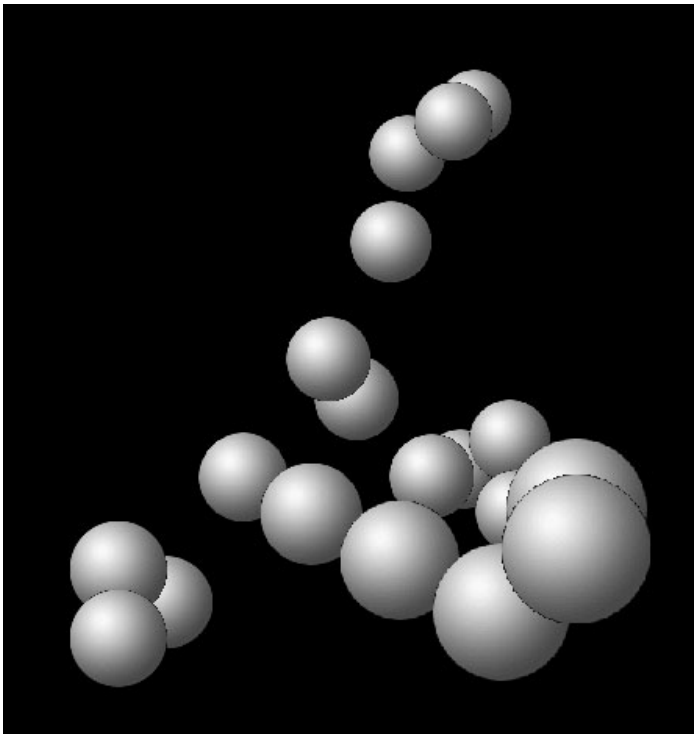


Abbildung 3.7: Eine Art „Molekül“ aufgebaut aus einzelnen Kugeln

Die Kugeln sind hier das Einzige, was vorberechnet wird. Dazu kann man z.B. in Photoshop einen runden Bereich markieren und diesen mit einem radialen Verlauf von hell nach dunkel füllen. Als Format kann man nur GIF (oder PNG) benutzen, da nur hiermit ein transparenter Hintergrund möglich ist. D.h. also der Bereich des Bildes, der nicht von der Kugel bedeckt wird, soll transparent sein. In Photoshop kann man zudem für das Antialiasing beim Übergang zum transparenten Bereich die Farbe angeben, zu der übergeblendet werden soll. Da wir für die Seite als Hintergrund Schwarz benutzen wollen, sollte auch hier Schwarz angegeben werden.

Der Quellcode

Wenn nun das Kugelbild vorliegt und die mathematischen Grundlagen geklärt sind, können wir uns dem Quellcode zuwenden. Sie finden das entsprechende Beispiel auf der CD im Verzeichnis *dhtml\beispiel_sprite*.

Beginn des Listings zu beispiel_sprite/index.htm

```
<html>
<head>
     <title>Sprite</title>
</head>

<script>
var objects = new Array(50);
var objectsTrans = new Array(50);
var objectsCount = 0;
var divs = "";
var winkelY = 0;
var winkelX = 0;
var dragging = false;
var akt = 0;
var oldX = 0;
var oldY = 0;
```

Es werden zwei Feldvariablen initialisiert, die die Koordinaten der Kugeln beinhalten. Das Feld `objects` hält jeweils immer die ursprünglichen Koordinaten, während `objectsTrans` die transformierten Koordinaten enthält. Dies ist notwendig, da bei den Transformationen notwendigerweise immer Rundungsfehler entstehen, die sich unangenehm aufsummieren würden mit der Zeit, wenn man die Transformationen nicht immer auf die „sauberen“ Ursprungskoordinaten anwenden würde. Zur Verwendung der anderen Variablen werde ich später zurückkommen.

Die Definition eines Vektor-Objektes in JavaScript

```
function Vector(x,y,z) {
   this.x = x ;
   this.y = y ;
   this.z = z ;
   this.s = 1.0;
   return this;
}
```

Diese Funktion definiert ein Vektor-Object in JavaScript-Schreibweise. Die `this.xyz`-Zuweisung beschreibt dabei die Properties des Objektes, auf die dann später in OO-Manier zugegriffen werden kann.

```
function addObject(x,y,z) {
   objects[objectsCount] = new Vector(x,y,z);
   objectsTrans[objectsCount] = new Vector(x,y,z);
   divs = divs + "<div id ='div" + objectsCount +
            "' style='position:absolute;'> </div>\n"
   objectsCount++;
}
```

Diese Funktion fügt der Szene eine weitere Kugel hinzu. Dazu wird den beiden Koordinatenfeldern ein neues Objekt vom Typ Vector hinzugefügt und der Objektzähler um eins erhöht. Der String `divs`, der hier um eine Ebenendefinition erweitert wird, stellt ein Stück HTML-Code dar. Dieser String wird später im Body-Tag per `document.write` ausgegeben. Im Wesentlichen wird dadurch genau die Anzahl benötigter Ebenen auf der HTML-Seite erzeugt. Diese Ebenen haben bei der Initialisierung nur eine ID, aber noch keinen Inhalt, da dieser sowieso immer wieder neu erzeugt wird.

```
function displayObject(nr) {

   //projektion des 3D-Punktes
   px = (objectsTrans[nr].z * camera.x -
         objectsTrans[nr].x * camera.z) /
        (objectsTrans[nr].z - camera.z) + 250;
   py = (objectsTrans[nr].z * camera.y -
         objectsTrans[nr].y * camera.z) /
        (objectsTrans[nr].z - camera.z) + 250;
   pb = Math.abs((128 * camera.z) /
        (objectsTrans[nr].z*1.2 - camera.z) / 2);

   div = eval("document.all.div" + nr);
   div.innerHTML =
          "<img src='ball.gif' border=0 width=" +
          pb + " height=" + pb + "> " ;
   div.style.top = py;
   div.style.left = px;
   div.style.zIndex =
        - parseInt(objectsTrans[nr].z - camera.z);
}
```

Diese Funktion stellt das Objekt mit der angegebenen Nummer auf der Seite dar. In den Variablen `px` und `py` wird dafür die Position auf dem Bildschirm berechnet. Die Projektion wird dabei bezüglich einer virtuellen Kameraposition durchgeführt, die sich an der Position camera befindet und Richtung Nullpunkt blickt. Der Offset +250 bezieht sich jeweils auf das Anzeigefenster mit einer Größe von 500x500 Pixeln, wie es später im HTML-Code definiert ist, und besagt, dass der Nullpunkt des Koordinatensystems in der Mitte des Anzeigefensters liegen soll.

Die Größe der Kugeln muss korrekt dargestellt werden.

Die Variable `pb` erhält die Größe der Kugeln. Dies ist ein sehr wichtiger Aspekt, um die Darstellung realistischer zu gestalten. Dadurch, dass weiter entfernte Kugeln kleiner sind als näher liegende, wird der 3D-Effekt verstärkt. Diese Größe wird berechnet, indem die Referenzgröße von 128 Pixeln genauso wie die Position projiziert wird. Dafür werden quasi zwei Punkte, die 128 Pixel auseinander liegen, separat projiziert und dann der resultierende Abstand genommen. Man erhält die Zeile also, indem man einen Punkt p1 (beliebige x-Koordinate, y-Koordinate = 0 und z-Koordinate entsprechend der des Objektpunktes) und p2 (genauso wie p1, nur die x-Koordinate ist gleich p1.x + 128) voneinander subtrahiert. Es sind allerdings noch zwei zusätzliche Modifizierer eingefügt. Die Multiplikation *1.2 bewirkt, dass der Effekt etwas verstärkt wird, näher liegende Objekte also noch etwas größer und weiter weg liegende noch etwas kleiner wirken. Die Division / 2 bewirkt, dass alle Kugeln insgesamt nur halb so groß dargestellt werden.

Nach diesen Berechnungen kann das HTML für das Bild geschrieben werden mit den entsprechenden Größenangaben. Dies geschieht durch die Zuweisung `div.innerHTML = ...`. Weiterhin wird die Position der Ebene gesetzt. Die letzte Zeile wirkt zwar einfach, stellt jedoch einen bedeutenden Trick dar. Um den 3D-Effekt zu erhalten, ist es unumgänglich, dass weiter entfernt liegende Objekte hinter den weiter vorne liegenden Objekten sind. Wer vor wem liegt, kann sich jedoch durch eine einfache Rotation der Kugel grundsätzlich ändern. Deshalb muss die Bestimmung, wer hinter wem liegt, nach jedem Transformationsschritt neu durchgeführt werden.

Die Kugeln werden tiefensortiert.

Normalerweise wird die Verdeckung so gemacht, dass für alle Objekte die Entfernung zur Kamera berechnet wird und anschließend die Objekte nach dieser Entfernung sortiert werden. Mithilfe dieser Sortierung kann dann das am weitesten entfernte Objekt zuerst gezeichnet werden und dann der Reihe nach die näher liegenden. Solch eine Sortierung ist allerdings recht zeitaufwändig, da eine Sortierung immer im Mittel eine Laufzeit von O(n) = n*log(n) hat. Hier wird allerdings eine simple Annahme getroffen, und zwar, dass nicht mehr als 1000 Tiefenunterschiede betrachtet werden müssen. Das heißt, die z-Koordinate jedes Objektes wird direkt in den z-Index der Ebene umgesetzt und damit die Verdeckungsberechnung dem HTML-Renderer des Browsers überlassen. (Dieser Sortieralgorithmus ist auch bekannt unter dem Namen „Hinkelstein-Sort“. Man verdeutliche sich das an dem Beispiel, dass Obelix einen Haufen von Hinkelsteinen dem Gewicht nach sortieren möchte. Dazu nimmt er jeden Stein und wirft ihn mit immer gleicher Kraft in die immer gleiche Richtung. Anschließend liegen alle Hinkelsteine schön sortiert vor ihm, der schwerste ganz nah und der leichteste weit entfernt. Dass dieser Algorithmus gewisse Einschränkungen hat, wird sofort klar, aber für manche Anwendungsfälle ist er durchaus geeignet, wie wir sehen :-)

```
function init() {
   camera = new Vector(0.0,0.0,-500.0);

   addObject(0,200,0);
   addObject(0,140,0);
   addObject(0,80,0);
   addObject(0,20,0);
   addObject(0,-40,0);
   addObject(0,-100,0);
   addObject(60,-100,0);
   addObject(120,-100,0);
}
```

Hier wird zuerst die virtuelle Kameraposition gesetzt und anschließend werden die Objekte initialisiert. Bei der Initialisierung wird jeweils die Position mit übergeben.

```
function mouseDown(){
    dragging = true;
}

function mouseUp(){
    dragging = false;
}

function mouseMove(){
     var x = event.offsetX;
     var y = event.offsetY;

    if (dragging) {
        if ((oldX > x) && (oldX - x > 5)) {
            oldX = x;
            winkelY += 10;
        }
        if ((oldX < x) && (x - oldX > 5)) {
            oldX = x;
            winkelY -= 10;
        }
        if ((oldY > y) && (oldY - y > 5)) {
            oldY = y;
            winkelX -= 10;
        }
        if ((oldY < y) && (y - oldY > 5)) {
            oldY = y;
            winkelX += 10;
        }
        rotiere(winkelX, winkelY);
       updateScene();
    }
     return;
}
```

Die Mausfunktionen dürften aus den vorangegangenen Beispielen schon bekannt sein. Bei einer Änderung wird zunächst die Rotation berechnet und anschließend die Szene neu gezeichnet.

```
function rotiere(winkelX, winkelY) {
   var i = 0;
   winkelX = Math.PI/180.0 * winkelX;
   winkelY = Math.PI/180.0 * winkelY;
   status = winkelY;
   for (i = 0;i < objectsCount; i++) {
      x = objects[i].x;
      y = objects[i].y;
      z = objects[i].z;
      // rotation um die x-Achse
      objectsTrans[i].y = y * Math.cos(winkelX) -
                          z * Math.sin(winkelX);
      objectsTrans[i].z = y * Math.sin(winkelX) +
                          z * Math.cos(winkelX);
      // rotation um die y-Achse
      y = objectsTrans[i].y;
      z = objectsTrans[i].z;
      objectsTrans[i].x = x * Math.cos(winkelY) +
                          z * Math.sin(winkelY);
      objectsTrans[i].z = (-1) * x * Math.sin(winkelY) +
                          z * Math.cos(winkelY);
   }
}
```

Diese Funktion rotiert alle Objekte um die x- oder y-Achse. Die Winkel werden an die Funktion in Gradangaben übergeben und als Erstes ins Bogenmaß umgerechnet. Beim Bogenmaß werden die Winkel zwischen 0° und 360° auf den Bereich zwischen 0 und 2 abgebildet.

Die Rotation um die x- und die um die y-Achse werden nacheinander durchgeführt. Vor der Berechnung einer Rotation werden die aktuellen Koordinaten in den Variablen `x`, `y` und `z` zwischengespeichert, um sie in den Berechnungen verwenden zu können. Das Ergebnis landet dann in dem Feld `objectsTrans`. Die Details zu den Drehberechnungen werden im Grundlagenkapitel erklärt.

```
function updateScene() {
   var i = 0;
   for (i = 0;i < objectsCount; i++)
      displayObject(i);
}
```

Hier werden einfach alle Objekte mit der Funktion `displayObject` neu gezeichnet.

```
</script>
<style type="text/css">
<!--
#DIV1 {position:absolute; left:100; top:100; width:600;
       height:600;z-Index:0}
#DIV2 {position:absolute; left:0; top:0; width:800;
       height:800; background-image:url("fill.gif");
       z-Index:1}
-->
</style>

<body bgcolor="#000000" text="#6666ff">

<div id="DIV2" onMouseUp="mouseUp()" onMouseDown="mouseDown()"
onMouseMove="mouseMove()">
 
</div>
```

Für die Darstellung wird eine extra Ebene benutzt, die dann auch alle Mausevents abfängt.

```
<div id="DIV1">
<script>
init();
document.write(divs);
updateScene();
</script>
</div>
```

Im Body-Tag wird die Initialisierung aufgerufen und danach der durch die Initialisierung entstandene String `divs` ausgegeben.

Ende des Listings zu beispiel_sprite/index.htm

```
</body>
</html>
```

Mit Hilfe der Sprite-Methode können sehr viele unterschiedliche Effekte erreicht werden. Selbst mit diesem einfachen Beispiel kann man noch vieles ausprobieren, zum Beispiel verschiedene Gebilde mit den Kugeln formieren, den Kugeln unterschiedliche Farben geben, die Drehung automatisch durchführen lassen, weitere Transformationen ermöglichen und vieles mehr.

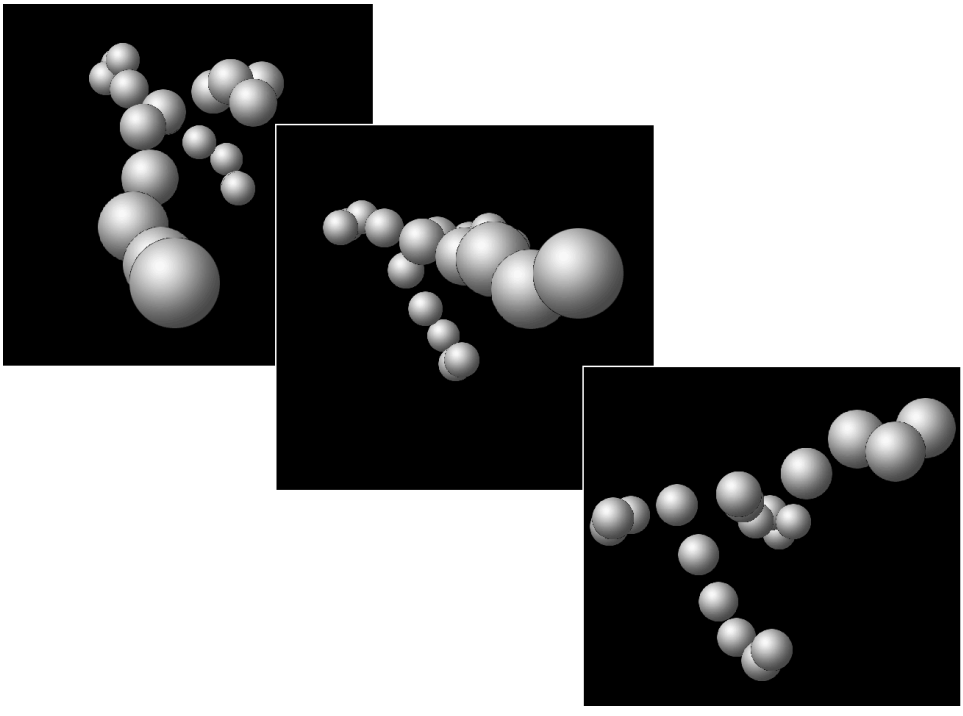


Abbildung 3.8: Das „Molekül“ kann mit der Maus gedreht werden

Es lassen sich aber auch die bisher in diesem Kapitel beschriebenen Verfahren kombinieren. Zum Beispiel könnte der Hintergrund sich beim Drehen der Kugeln mitdrehen.

Auch für neue Navigationsmöglichkeiten ist diese Methode nutzbar. Es wäre zum Beispiel denkbar, dass die einzelnen Sprites Menüeinträge in einer Informationsstruktur darstellen. Der Nutzer könnte sich dann durch dieses 3D-Menü hindurchbewegen.

Der Phantasie sind also keine Grenzen gesetzt. Na ja, fast keine, immerhin reden wir hier immer noch von HTML. Aber wie man die Grenzen noch etwas weiter ausdehnen kann, soll das nächste Beispiel zeigen.

3.5 Real berechnete Wireframe-Modelle

Für das nächste Beispiel soll nichts mehr vorberechnet, sondern alles dynamisch generiert werden. Na ja, ein einziges Element müssen wir vorher erzeugen, und zwar eine Linie. Da man mit HTML keine Linien auf dem Bildschirm zeichnen kann, müssen wir hier auf einen Trick zurückgreifen.

Doch zunächst klären wir, was eigentlich gezeigt werden soll. Die einfachste und schnellste Darstellungsform von 3D-Grafiken ist ein Drahtgittermodell. Dies wird so genannt, weil die Darstellung nur aus Li-

nien besteht und alle Kanten bzw. Flächenbegrenzungen des 3D-Objektes enthält. Ursprünglich wurde diese Darstellungsform nur wegen der höheren Geschwindigkeit benutzt. Mit den heutigen Grafikfähigkeiten eines Standard-PC ist es jedoch nicht mehr notwendig, sich damit zu begnügen. Heute benutzt man diese Darstellungsform nur noch, wenn es um wirklich extrem große Objekte (mit zigtausend Flächen) geht oder wenn man in der Konstruktionsphase sich dadurch einen besseren Überblick verschaffen möchte.

Aber abgesehen von diesen Überlegungen hat die Drahtgitterdarstellung natürlich auch noch ästhetische Aspekte. Und genau darum wollen wir sie hier benutzen (unabhängig von der Tatsache, dass man mit DHTML gar nicht mehr machen kann im Bereich real berechneter 3D-Grafik).

Wie schon erwähnt können wir mit HTML keine Linien zeichnen. Deshalb müssen wir zunächst die Grafik für die Linie erzeugen. Dies ist wie im vorherigen Beispiel wieder ein GIF-Bild mit transparentem Hintergrund, welches genau eine Linie von der linken oberen Ecke bis zur rechten unteren Ecke enthält. Im Gegensatz zum letzten Beispiel jedoch soll hier kein Antialiasing benutzt werden, da diese Grafik extrem hin- und herskaliert wird, so dass sie eh nicht mehr allzu ästhetisch wirkt. Um genau zu sein benötigen wir noch zwei weitere Grafiken, und zwar eine Linie von links unten nach rechts oben und einen einzelnen Punkt. Mit diesen zwei Linien (siehe Abbildung 3.9) und dem Punkt sollte es möglich sein, beliebige Linien darzustellen.

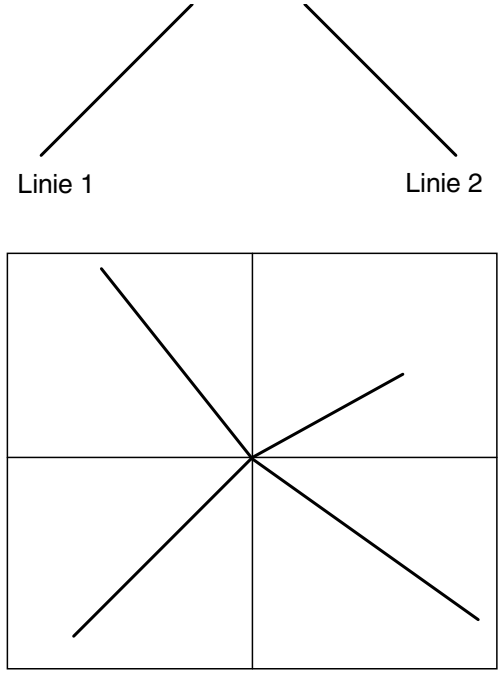


Abbildung 3.9: Beispiele für mögliche Linien

Abbildung 3.9 unten zeigt, welche Arten von Linien es geben kann. Es wird deutlich, dass die Linien aus dem Quadranten links oben und rechts unten mit der Grafik Linie 2 dargestellt werden können, indem man die Grafik entsprechend ungleichmäßig skaliert. Anders ausgedrückt können Sie durch vertikales und horizontales Stauchen und Strecken der Linie 2 ein Rotieren der Linie simulieren. Ebenso können die Linien aus den anderen Quadranten mit der Grafik Linie 1 dargestellt werden. Die exakt senkrechten und waagerechten Linien können mit dem einzelnen Punkt dargestellt werden, indem dieser entsprechend skaliert wird.

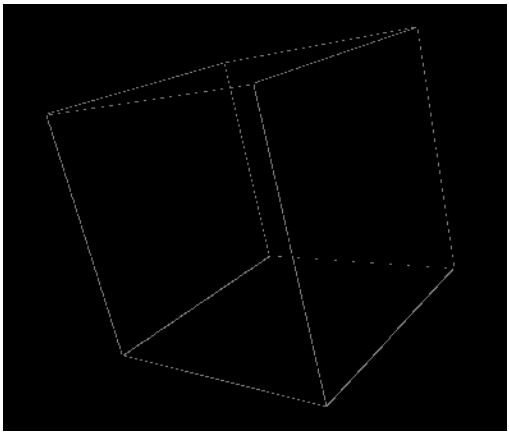


Abbildung 3.10: Darstellung und Bewegung eines real berechneten Würfels

Wie Sie in dem fertigen Beispiel sehen werden, ist die Skalierung des Browsers nicht besonders gut, da er ab einer gewissen Stauchung unter Umständen viele Punkte einfach weglässt (siehe Abbildung 3.10). Dies könnte man umgehen, indem man noch mehr Liniengrafiken zur Verfügung stellt und dann jeweils die passendste auswählt. Der ambitionierte Leser kann das gerne ausprobieren.

Kommen wir nun zum eigentlichen Programm. Einiges daraus werden Sie aus den vorangegangenen Beispielen schon kennen.

Beginn des Listings zu beispiel_real/index.htm

```
<html>
<head>
     <title>Real</title>
</head>

<script>
var objects = new Array(50);
var objectsTrans = new Array(50);
var objectsCount = 0;
var divs = "";
var winkelY = 0;
var winkelX = 0;
var dragging = false;
```

```
var akt = 0;
var oldX = 0;
var oldY = 0;
var lines = new Array(12);
var linesCount = 0;
```

Neben den bekannten Feldern `objects` und `objectsTrans`, die wieder die ursprünglichen Koordinaten bzw. die transformierten Koordinaten halten, wird hier noch ein Feld `lines` definiert, welches die zu zeichnenden Linien hält. Ein Eintrag für eine Linie besteht hierbei aus zwei Indizes. Diese Indizes verweisen auf die entsprechende Koordinate im Feld `objects`.

```
function Vector(x,y,z) {
   this.x = x ;
   this.y = y ;
   this.z = z ;
   this.s = 1.0;
   return this;
}
```

Hier wird wie üblich in OO-Manier ein Vektor-Objekt definiert.

```
function addObject(x,y,z) {
   objects[objectsCount] = new Vector(x,y,z);
   objectsTrans[objectsCount] = new Vector(x,y,z);
   objectsCount++;
}
```

Diese Funktion füllt die Koordinaten, die zum Aufbau der Szene benötigt werden, in die entsprechenden Felder.

```
function addLine(a,b) {
   lines[linesCount] = new Array(2);
   lines[linesCount][0] = a;
   lines[linesCount][1] = b;
   divs = divs + "<div id ='div" + linesCount +
          "' style='position:absolute;'></div>\n"
   linesCount++;
}
```

Wie oben schon erwähnt, werden hier die Linien definiert. Die Parameter `a` und `b` bezeichnen dabei die Positionen der Koordinaten im Feld `objects`, die die Endpunkte der Linie bilden sollen.

Da wir in diesem Beispiel keine Sprites an den Koordinaten, sondern Linien, die die Koordinaten verbinden, darstellen wollen, wird in dieser Funktion der String mit der HTML-Definition zum Aufbau der entsprechenden Ebene erzeugt.

```
function projiziere(v1) {
   p = new Vector(0,0,0);
    //projektion des 3D-Punktes
   p.x = (v1.z * camera.x - v1.x * camera.z) /
         (v1.z - camera.z) + 250;
```

```
    p.y = (v1.z * camera.y - v1.y * camera.z) /
          (v1.z - camera.z) + 250;
  return p;
}
```

Diese Funktion projiziert einen übergebenen 3D-Punkt in die Bildschirmebene. Das Ergebnis wird der Einfachheit halber wieder in einem 3D-Vektor zurückgegeben, es sind jedoch nur die x- und y-Koordinate relevant.

Die nun folgende Funktion ist der interessanteste Teil. Deshalb werde ich die einzelnen Teile direkt beschreiben.

```
function displayLine(nr) {
   p1 = projiziere(objectsTrans[lines[nr][0]]);
   p2 = projiziere(objectsTrans[lines[nr][1]]);
```

Zunächst werden die Endpunkte der Linie in Bildschirmkoordinaten überführt und in `p1` und `p2` gespeichert. Als Nächstes muss ermittelt werden, welches der drei möglichen Liniengrafikbilder zur Darstellung benutzt werden soll.

```
   img = "linie1.gif";
   if (p2.y < p1.y) {p3 = p1; p1 = p2; p2 = p3;}
```

Wenn `p2` tiefer liegt als `p1`, werden die beiden Punkte vertauscht. Das ändert nichts am späteren Aussehen, erleichtert aber die weitere Verarbeitung. Jetzt kann die Linie von `p1` nach `p2` nur noch von links oben nach rechts unten gehen oder von rechts oben nach links unten.

```
   if (p2.x < p1.x) {img = "linie2.gif";}
```

Falls also `p2.x` kleiner als `p1.x` ist, geht die Linie von rechts oben nach links unten und wir müssen das zweite Bild nehmen.

```
   h = Math.abs(p2.y - p1.y);
   w = Math.abs(p2.x - p1.x);
```

In h und w wird nun die Höhe und Breite des darzustellenden Bildes gespeichert.

```
   if (h < 3) {h = 1; img = "punkt.gif";}
   if (w < 3) {w = 1; img = "punkt.gif";}
```

Falls die Linie fast senkrecht oder waagerecht ist, kann der Punkt benutzt werden.

```
   div = eval("document.all.div" + nr);
   div.innerHTML = "<img src='" + img +
                   "' border=0 width=" + w +
                   " height=" + h + "><br>  " ;
   div.style.top = p1.y;
   if (p2.x < p1.x) {p1.x = p2.x;}
   div.style.left = p1.x;
}
```

Hier wird nun das Bild geschrieben und die Ebene entsprechend positioniert. Eine Tiefensortierung wie bei den Sprites muss hier nicht vorgenommen werden, da man bei dünnen Linien nicht unterscheiden kann, welche vor oder hinter einer anderen liegt.

```
function init() {
   camera = new Vector(0.0,0.0,-500.0);

   addObject(-100,-100,-100);
   addObject(100,-100,-100);
   addObject(100,-100,100);
   addObject(-100,-100,100);
   addObject(-100,100,-100);
   addObject(100,100,-100);
   addObject(100,100,100);
   addObject(-100,100,100);

   addLine(0,1);
   addLine(1,2);
   addLine(2,3);
   addLine(3,0);
   addLine(4,5);
   addLine(5,6);
   addLine(6,7);
   addLine(7,4);
   addLine(0,4);
   addLine(1,5);
   addLine(2,6);
   addLine(3,7);
}
```

Als darzustellendes Objekt bauen wir hier einen einfachen Würfel auf. Als Erstes werden mittels `addObject` die acht Eckpunkte definiert. Danach werden mit `addLine` die 12 Verbindungslinien erzeugt.

```
function mouseDown(){
    dragging = true;
}

function mouseUp(){
    dragging = false;
}

function mouseMove(){
     var x = event.offsetX;
     var y = event.offsetY;

    if (dragging) {
        if ((oldX > x) && (oldX - x > 5)) {
            oldX = x;
            winkelY += 10;
```

```
        }
        if ((oldX < x) && (x - oldX > 5)) {
            oldX = x;
            winkelY -= 10;
        }
        if ((oldY > y) && (oldY - y > 5)) {
            oldY = y;
            winkelX -= 10;
        }
        if ((oldY < y) && (y - oldY > 5)) {
            oldY = y;
            winkelX += 10;
        }
        rotiere(winkelX, winkelY);
       updateScene();

     }
      return;
}
```

Die Mausfunktionen sind wieder die gleichen wie in den vorherigen Beispielen, damit man den Würfel beliebig drehen kann.

```
function rotiere(winkelX, winkelY) {
   var i = 0;
   winkelX = Math.PI/180.0 * winkelX;
   winkelY = Math.PI/180.0 * winkelY;
   status = winkelY;
   for (i = 0;i < objectsCount; i++) {
      x = objects[i].x;
      y = objects[i].y;
      z = objects[i].z;
      // rotation um die x-Achse
      objectsTrans[i].y = y * Math.cos(winkelX) -
                          z * Math.sin(winkelX);
      objectsTrans[i].z = y * Math.sin(winkelX) +
                          z * Math.cos(winkelX);
      // rotation um die y-Achse
      y = objectsTrans[i].y;
      z = objectsTrans[i].z;
      objectsTrans[i].x = x * Math.cos(winkelY) +
                          z * Math.sin(winkelY);
      objectsTrans[i].z = (-1) * x * Math.sin(winkelY) +
                          z * Math.cos(winkelY);
   }
}
```

Die Rotation der einzelnen Punkte im Raum entspricht wieder der Rotation im Beispiel mit den Sprites.

```
function updateScene() {
   var i = 0;
   for (i = 0;i < linesCount; i++)
      displayLine(i);
}

</script>
<style type="text/css">
<!--
#DIV1 {position:absolute; left:100; top:100; width:600;
       height:600;z-Index:0}
#DIV2 {position:absolute; left:0; top:0; width:800;
       height:800; background-image:url("fill.gif");
       z-Index:1}
-->
</style>

<body bgcolor="#000000" text="#6666ff">

<div id="DIV2" onMouseUp="mouseUp()"
               onMouseDown="mouseDown()"
               onMouseMove="mouseMove()">
 
</div>

<div id="DIV1">
<script>
init();

document.write(divs);
rotiere(0,0);
updateScene();
</script>
</div>

</body>
</html>
```

Ende des Listings zu beispiel_real/index.htm

Der Rest der Seite entspricht dem Sprite-Beispiel mit dem Molekül.

3.6 Cross-Browser-Besonderheiten

Die in diesem Kapitel gezeigten Beispiele verdeutlichen, wie viel tatsächlich mit den einfachen Mitteln von DHTML möglich ist. Natürlich gibt es dabei immer noch die Cross-Browser-Probleme zu beachten. Zwar wird von der Mehrzahl der Internetnutzer der Internet Explorer benutzt, es gibt aber auch einen großen Anteil an Nutzern, die den Netscape verwenden. Zum Beispiel ist in manchen großen Firmen der Netscape als Standardbrowser eingerichtet und vorgeschrieben. Wenn sich die Zielgruppe einer Site nun aus Geschäftskunden zusammensetzt, die in solch einer Firma sitzen könnten, sollte man diese Zielgruppe nicht durch Verwendung von proprietären Funktionen ausschließen.

Die Beispiele, die ich in diesem Kapitel gezeigt habe, sind alle auch mit Netscape ab Version 4 umsetzbar. Was bei Netscape zu beachten ist, sind vor allem die folgenden Punkte:

- Das Ansprechen der Ebenen (z.B. zum Verändern der Position) erfolgt in Netscape anders
- Ebenen werden ein wenig anders positioniert in Netscape
- Das Beschreiben der Ebenen (d.h. das Füllen mit HTML-Code) erfordert ein anderes Vorgehen mit Netscape
- Das Abfangen der Mausevents erfordert eine andere Behandlung

Wie man all diese Punkte am besten angeht, haben schon sehr viele Leute beschrieben. Am einfachsten ist es, wenn man eine JavaScript-Bibliothek benutzt, die all diese Probleme kapselt und eine einheitliche Programmierschnittstelle zur Verfügung stellt.

Cross-Browser-Links	
cross-browser.com	Freie Cross-Browser-Bibliothek
www.dansteinman.com/dynduo/	Cross- Browser-Tutorial
www.w3schools.com/dhtml/	DHTML Tutorial

3.7 Zusammenfassung

In diesem Kapitel haben Sie die vielfältigen Möglichkeiten kennen gelernt, die allein in der Nutzung von DHTML liegen. Vor allem die Möglichkeiten, die sich mit Sprites ergeben, haben sehr viel Potenzial. Hier lohnt es sich, noch andere Sachen auszuprobieren.

Aber auch wenn all die Beispiele in diesem Kapitel zeigen, dass mit DHTML sehr viel möglich ist, kann dies nur als rudimentärer Anfang gewertet werden. Um das Ganze noch ansprechender und flexibler zu gestalten, sind zurzeit leider Plugins in irgendeiner Form notwendig. Diese werden in den weiteren Kapiteln dieses Buches behandelt.

Flash

Während es beim vorhergehenden Kapitel über DHTML eigentlich vor allem darum ging, zu zeigen, wie viel man schon mit DHTML alleine erreichen kann, fängt es mit Flash erst an, richtig Spaß zu machen.

Flash kombiniert durch die vektororientierte Darstellung und das Antialiasing kompakte Dateien und gute Darstellungsqualität mit einer sehr weiten Verbreitung. Es lassen sich damit alle Techniken zur 3D-Darstellung umsetzen, seien es Sprites, vorberechnete Filmsequenzen oder real berechnete Darstellungen.

Die folgenden Beispiele zeigen jeweils die Verwendung dieser Techniken.

4.1 Grundlagen

Bei Flash handelt es sich um ein Browser-Plugin der Firma Macromedia, welches zurzeit in der Version MX (Nachfolger der Version 5) existiert. Flash nennt sich aber auch der Editor der Firma Macromedia, mit dem sich die Filme erstellen lassen, die das Flash-Plugin abspielt. Es gibt auch Editoren von anderen Herstellern. Diese unterstützen aber in der Regel nicht die aktuellste Version und nicht alle Funktionen des Plugins. Für die Beispiele in diesem Kapitel wird Flash als Editor benutzt.

Der Flash-Editor erzeugt so genannte „Filme“ im .swf-Format. Diese „Filme“ sind aber nicht einfach nur eine Sequenz von Bildern, sondern es gibt vielfältige Manipulations- und Interaktionsmöglichkeiten mit den Objekten innerhalb des Films.

Von den grundlegenden Möglichkeiten liegt Flash nahe bei DHTML, allerdings gehen die Funktionen etwas weiter und vor allem die Darstellungsqualität ist erheblich besser. Die Möglichkeiten von Flash umfassen die Darstellung von übereinander liegenden Ebenen und das freie Positionieren, Skalieren und Rotieren von Objekten auf den Ebenen. Wie oben bereits erwähnt, ist die Darstellungsqualität von Flash erheblich besser als mit DHTML. Das liegt zum einen daran, dass die Grafikelemente in Flash vektorbasiert sind. Dadurch wird vor allem das Skalieren und Rotieren mit gleich bleibender Qualität möglich. Ein weiterer Vorteil ist, dass Flash alle Elemente mit Antialiasing darstellen kann. Das be-

deutet, dass die Kanten von Grafikobjekten geglättet dargestellt werden.

Programmiert wird Flash mit ActionScript, was seit der Version 5 an JavaScript angepasst wurde. Das bedeutet, dass viele Code-Teile aus den DHTML-Beispielen hier mehr oder weniger direkt übernommen werden können.

Zur Erstellung von Flash-Filmen gibt es sehr viele Bücher, die bei einem Einstieg in Flash hilfreich sein können. Dieses Kapitel setzt ein gewisses Grundwissen im Umgang mit Flash voraus. Zum Mitprogrammieren und Abändern der Beispiele können Sie auch die auf der CD enthaltene voll funktionsfähige 30-Tage-Testversion von Flash MX installieren.

4.2 Das Hilfsprogramm Swift3D

In den Beispielen werden teilweise vorberechnete Bildsequenzen verwendet. Durch das vektorbasierte Grafikformat bietet es sich hierbei an, als Grundlage die Ausgabe von 3D-Grafikprogrammen wie zum Beispiel 3D Studio Max zu verwenden. In derartigen Programmen liegen die Daten ebenfalls vektororientiert vor, allerdings im Gegensatz zu Flash dreidimensional. Es existieren aber ein Vielzahl von Programmen, die diese dreidimensionalen Daten verwenden können, um daraus zweidimensionale Vektorgrafiken zu erzeugen, die Flash dann verarbeiten kann. Die Schwierigkeit dabei ist, dass verdeckte Kanten dann nicht mehr zu sehen sein sollen, und nach Möglichkeit sollen auch realistische Farbverläufe innerhalb der Flächen dargestellt werden. Ein Programm, welches all das recht gut beherrscht, ist Swift3D von electric rain.

Die Verwendung von Swift3D möchte ich hier einmal kurz darstellen. Zunächst erzeuge ich in 3D Studio einen einfachen Würfel und lasse ihn in einer kurzen Animation um 90° um die z- und y-Achse drehen.

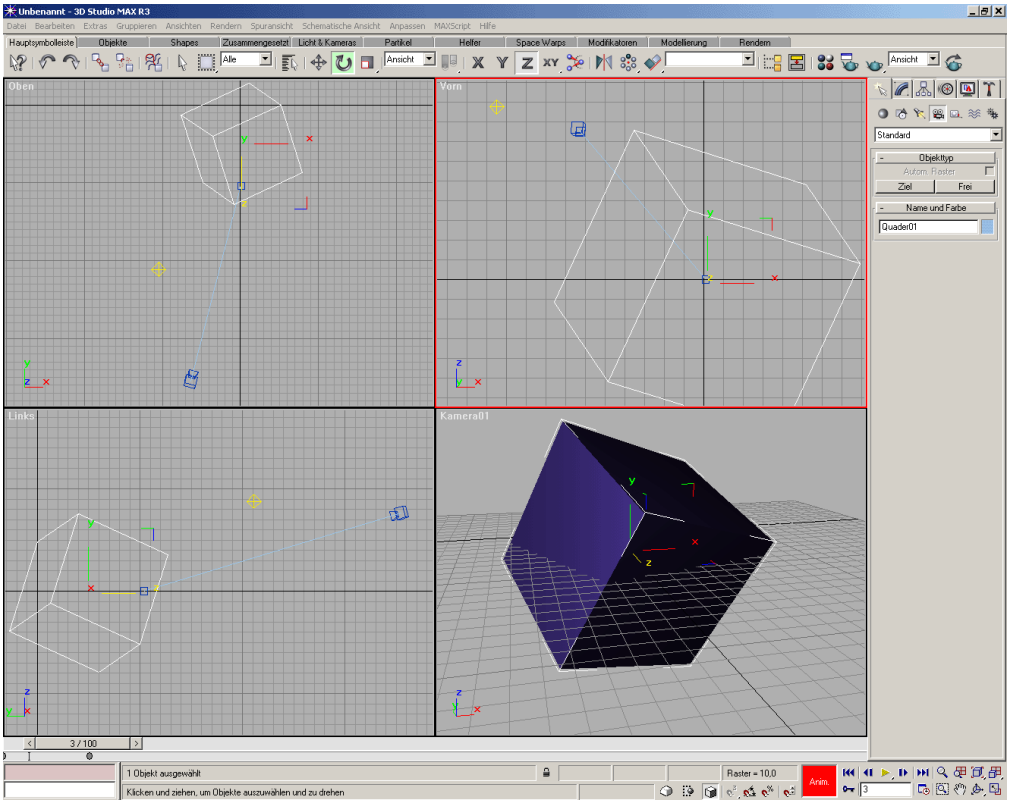


Abbildung 4.1: In 3D Studio werden das Ausgangsmodell und die Animation erzeugt.

Das Ganze wird dann im 3DS-Format (3D Studio) exportiert und kann anschließend mit Swift3D wieder eingelesen werden. Innerhalb von Swift braucht man im einfachsten Fall nichts weiter zu machen und kann gleich auf Export gehen.

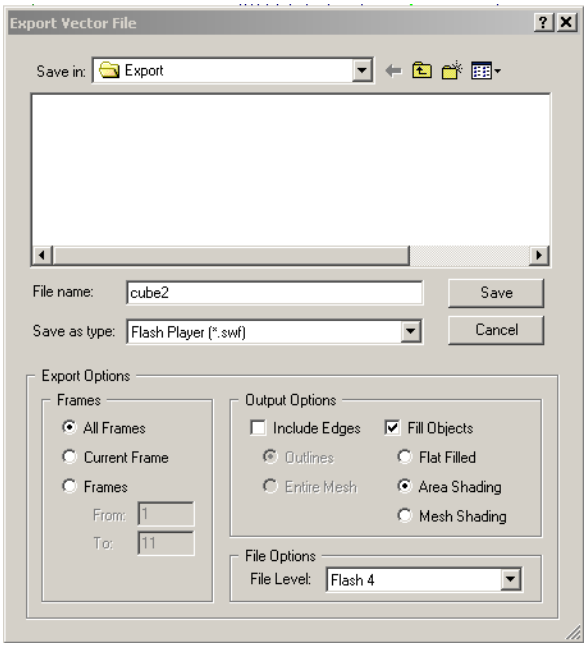


Abbildung 4.2: Die Export-Optionen in Swift3D

Hierbei hat man dann verschiedene Möglichkeiten für die Art der exportierten Bilder:

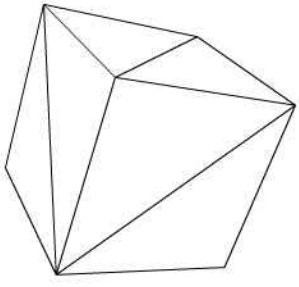


Abbildung 4.3: Entire Mesh

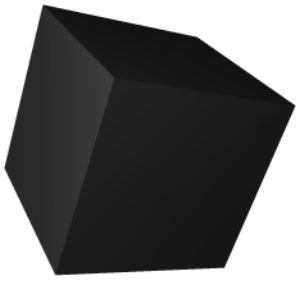


Abbildung 4.4: Area Shading

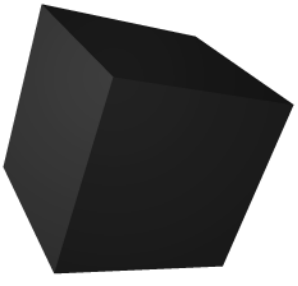


Abbildung 4.5: Mesh Shading

Die verschiedenen Shading-Möglichkeiten von Swift3D

Das beste Ergebnis liefert hierbei *Mesh Shading*, da dabei für jedes Polygon des Objektes ein eigener Farbverlauf generiert wird, während bei *Area Shading* für zusammenhängende Flächen nur ein Farbverlauf generiert wird. Bei dem Beispielwürfel hängen jeweils zwei Dreiecke zusammen, für die dann jeweils nur ein gemeinsamer Farbverlauf oder zwei eigene und damit genauere Farbverläufe gebildet werden.

Die von Swift exportierte SWF-Datei kann bereits für sich alleine genutzt werden und stellt einen rotierenden Würfel dar. Diese SWF-Datei

kann nun in Flash in einen anderen Film importiert und beliebig verwendet werden. Hierbei wird noch einmal der Vorteil des Vektorformats deutlich. Anstatt für jedes der 10 Bilder ein JPEG-Bild zu speichern, werden nur die Eckpunkte der Flächen und die Angaben zu den Farbverläufen gespeichert. Somit ist die resultierende Datei erheblich kleiner und kann auch beliebig skaliert werden.

Wer nicht über die eben beschriebenen Softwareprodukte verfügt, ist aber nicht verloren. Den Part von 3D Studio können viele günstigere Programme ebenso übernehmen und auch zu Swift gibt es Alternativen. Ein Mittel ist zum Beispiel, tatsächlich die Einzelbilder im JPEG-Format in Flash zu importieren und dann in Flash direkt „tracen" zu lassen oder sie von Hand zu „tracen", also zu vektorisieren.

4.3 Beispiel: Objektpräsentation

Mit 3D-Programmen erstellte Bildsequenzen lassen sich u.U. sehr Platz sparend speichern, wenn sie vektorisiert werden.

Wir wollen als erstes Beispiel das schon im vorangegangenen Kapitel DHTML vorgestellte Beispiel zur Objektpräsentation in Flash entwickeln. Zum einen soll daran gezeigt werden, wie man die Nutzerinteraktion in Flash nachprogrammiert, da dies ein häufiger Anwendungsfall ist. Zum anderen hat das Prinzip Bildsequenz bei Flash einen entscheidenden Vorteil: Der oben erwähnte Punkt, dass Flash Grafikobjekte vektorbasiert speichert, hat auch die Auswirkung, dass für viele Bilder weniger Speicherplatz verbraucht wird. Daraus ergibt sich, dass die Daten schneller über das Netz geladen sind. Für die Bildsequenzen, die wir in den Beispielen des letzten Kapitels verwendet haben, ist dies zwar nicht relevant, da sie aufgrund ihrer Art besser als Pixelgrafiken gespeichert werden, wir werden aber in den weiteren Beispielen in diesem Kapitel sehen, an welchen Stellen die Vektorgrafiken von Vorteil sind.

Die fertige Beispieldatei finden Sie unter dem Namen *vr.fla* im Verzeichnis Flash auf der CD.

Die Vorbereitung der Bildsequenz

Als Bildsequenz soll für dieses Beispiel die gleiche Sequenz mit der Person auf dem Drehstuhl dienen, d.h. es existiert nur ein Freiheitsgrad, in welchem gedreht werden kann.

Die Bildsequenz mit den 16 Einzelbilder wird importiert und in einem eigenen MovieClip abgelegt. Beim Import bietet Flash übrigens von sich aus an, wenn man das erste Bild importiert, die nachfolgend nummerierten Bilder gleich mit zu importieren, und legt diese dann in aufeinander folgenden Frames ab (siehe Abbildung 4.6).
Der MovieClip erhält den Namen `sequenz` und wird in Szene 1 platziert.

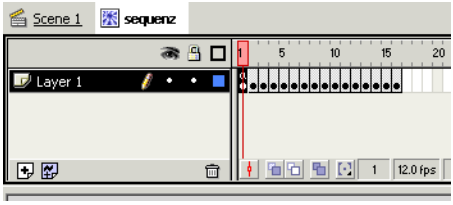


Abbildung 4.6: Bildsequenz

Der Quellcode

In Frame 1 in Szene 1 wird der Code platziert, um notwendige Initialisierungen vorzunehmen und allgemeine Funktionen zu definieren.

Die Initialisierungen

```
var dragging = false;
var oldX = 0;
var akt = 1;
var anzahlX = 16;

function updateImg(dir) {
    _root.akt = _root.akt + dir;
    if (_root.akt < 1) _root.akt = _root.anzahlX;
    if (_root.akt > _root.anzahlX) _root.akt = 1;

    _root.sequenz.gotoAndStop(_root.akt);

}
```

Die boolesche Variable `dragging` enthält den Wert, ob gegenwärtig mit der Maus das Bild „gezogen“ wird. `OldX` ist der x-Wert der letzten Abfrage (um zu überprüfen, wie groß die Änderung war). `akt` bezeichnet das aktuelle Bild der Sequenz und anzahlX die Anzahl der Bilder in x-Richtung. Die Funktion `updateImg` schließlich ändert das dargestellte Bild, je nach übergebener Richtung `dir`.

In Frame 3 steht nur noch der Code, um zurück zu Frame 2 zu springen, d.h. also eine Endlosschleife zu bilden.

Der Code, um die Nutzerinteraktionen abzufangen, steht als Objekt-Action bei dem MovieClip, welcher die Bildsequenz enthält.

```
onClipEvent(mouseDown) {
     _root.dragging = true;
}

onClipEvent(mouseUp) {
     _root.dragging = false;
}

onClipEvent (mouseMove) {
     var x;
     x = _xmouse;

     if (_root.dragging == true) {
        if ((_root.oldX > x) && (_root.oldX - x > 10)) {
            _root.oldX = x;
            _root.updateImg(-1);
        }
        if ((_root.oldX < x) && (x - _root.oldX > 10)) {
            _root.oldX = x;
            _root.updateImg(1);
        }
    }
}
```

Hier werden die entsprechenden Events abgefangen. Wenn der Nutzer die Maustaste drückt, wird das Flag `dragging` auf Wahr gesetzt und somit angezeigt, dass ab jetzt die Mausbewegungen das Objekt drehen sollen. Wird die Maustaste wieder losgelassen, endet die Bewegung.

Beim Bearbeiten der Bewegungen selbst führen nur Mausbewegungen größer als 10 Pixel zu einem Weiterschalten der Bildsequenz in die entsprechende Richtung.

Auch hier kann das Programm so erweitert werden, dass mehrere Freiheitsgrade möglich sind, um die das Objekt gedreht werden kann, falls die entsprechenden Bildsequenzen vorliegen. Der Code ist entsprechend dem im DHTML-Beispiel.

4.4 Beispiel: Hierarchisches Menü

Das folgende Beispiel stellt eine andere Ausprägung des Bildsequenz-Prinzips dar. Die Filmsequenz, die wir hier benutzen, wird während des Abspielens skaliert und verschoben. Außerdem wird auf einen Mausklick hin die gesamte Sequenz durchgespielt.

Das Beispiel kommt vielleicht einigen Leuten bekannt vor, die früher schon mal mit SGI-Rechnern gearbeitet haben. Realisiert wird hier ein Menüsystem, welches eine hierarchische Abbildung ermöglicht.

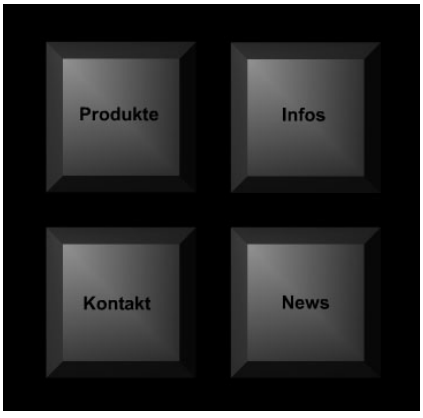


Abbildung 4.7: Hauptmenü

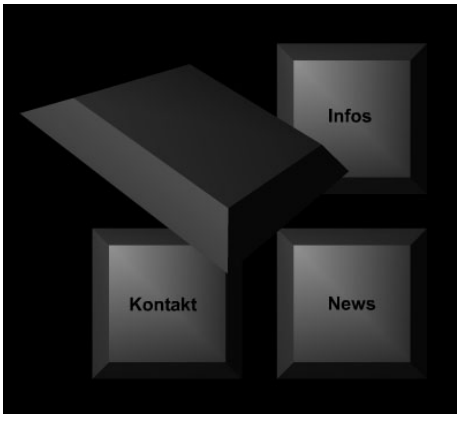


Abbildung 4.8: „Produkte“ wurde angeklickt

Abbildung 4.9: Das Untermenü von „Produkte“ wird geöffnet

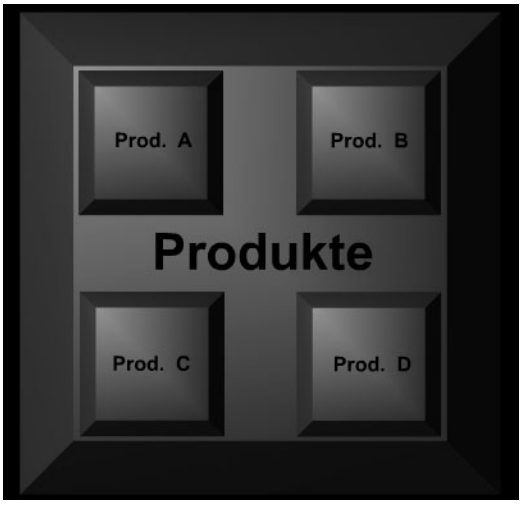


Abbildung 4.10: Eine Auswahl im Untermenü öffnet auf gleiche Art und Weise ein Unteruntermenü.

Die Bildsequenz

Die Filmsequenz, die wir hierfür benutzen, sieht folgendermaßen aus:

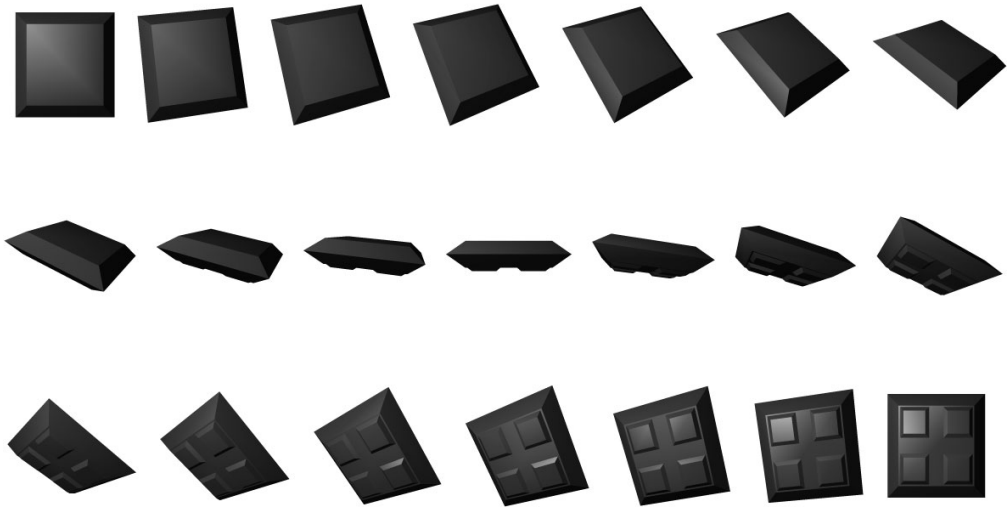


Abbildung 4.11: Bildsequenz für das 3D-Menü

Diese gesamte Bildsequenz verbraucht nur gut 20 KB Speicherplatz, kann aber wie schon erwähnt beliebig groß skaliert werden. Den kompletten Beispielfilm finden Sie auf der CD unter dem Name *squares.fla*.

Sie können entweder die HTML-Datei öffnen, um das Flash Plugin zum Betrachten zu benutzen, oder Sie öffnen direkt die SWF-Datei, wenn Sie den Flash-Player installiert haben.

In dem Beispiel ist nur jeweils der Knopf links oben aktiviert und auch nur zwei Ebenen tief. Zurück kommen Sie, indem Sie auf eine Fläche außerhalb des Knopfes klicken.

Für dieses Beispiel wird so gut wie keine Programmierung benötigt. Lediglich für die Mausklicks und die Sprünge sind einfache Befehle notwendig.

Der Flash-Film

Gehen wir nun Schritt für Schritt durch die Erstellung dieses Beispiels (das fertige Beispiel finden Sie auch auf der CD unter dem Namen *squares_beispiel.fla*):

Öffnen Sie in Flash einen neuen Film und geben Sie ihm die Hintergrundfarbe Schwarz (es ist auch jede andere Farbe möglich).

Erzeugen Sie anschließend ein neues Symbol mit Strg+F8. Geben Sie dem Symbol den Namen `Sequenz` und den Typ `Graphic`. Importieren Sie in den neuen MovieClip nun die Bildsequenz, pro Frame ein Bild. Die Bildsequenz hat auf der CD den Namen *square_sequenz.swf*. Löschen Sie anschließend den ersten Frame, da hierin nur eine Meldung von Swift steht. Zum Löschen eines Frames verwenden Sie die Funktion *Remove Frame*.

Wir werden vier Layer verwenden, die Sie jetzt anlegen sollten: einen für die Knöpfe, einen für die Animation, einen mit den aktiven Flächen für die Mausklicks und einen für die Framelabels.

Ziehen Sie das Symbol `sequenz` aus der Bibliothek auf die unterste Ebene in Frame1. Verschieben Sie den Centerpunkt des Symbols in dessen Mittelpunkt und skalieren Sie den Knopf nun so groß, dass vier davon auf die verfügbare Fläche passen. Geben Sie außerdem auf dem Reiter Instanz an, dass das Symbol nur einen einzigen Frame anzeigt und nicht die gesamte Bildfolge wiederholt.

Nun können Sie drei zusätzliche Kopien des Symbols erzeugen und im Frame platzieren. Zum Positionieren der Symbole verwenden Sie am besten die direkte Eingabe der Koordinaten, damit die Symbole alle den gleichen horizontalen und vertikalen Abstand haben. Erzeugen Sie weiter eine fünfte Kopie, die Sie hinter die anderen Symbole legen. Alle fünf Buttons sollten nun auf der untersten Ebene wie in Abbildung 4.12 zu sehen positioniert sein.

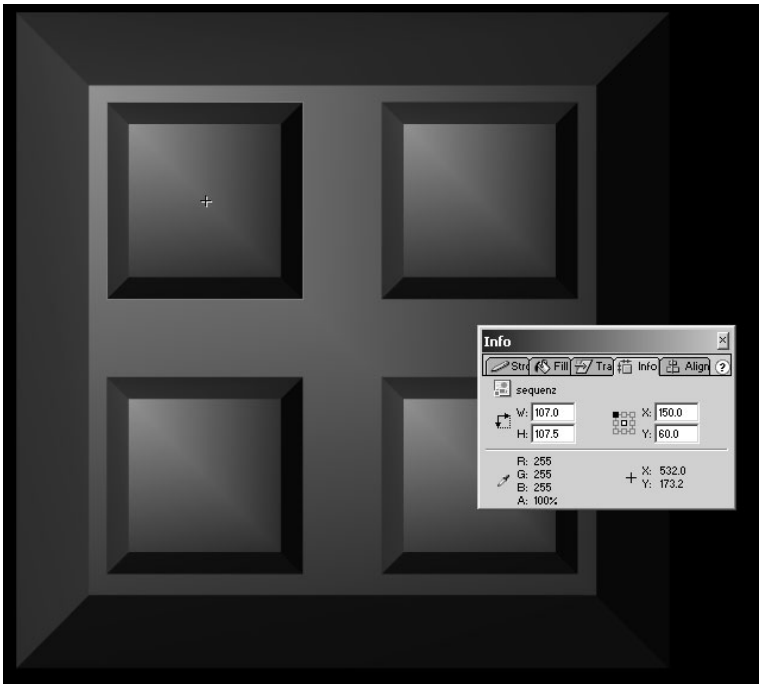


Abbildung 4.12: Die Grundposition der Buttons

Da die Bildfolge insgesamt 21 Bilder hat, legen wir nun unsere Keyframes auf der untersten Ebene *buttons* im Abstand von 20 Frames an. Gehen Sie dazu auf Frame 20, 40, 60 und 80 und drücken Sie jeweils F6. Wiederholen Sie das Einfügen der Keyframes auch für die Ebenen *animations* und *labels* (auch wenn diese noch leer sind).

Gehen Sie anschließend auf Frame 20 und selektieren Sie den Button links oben. Schneiden Sie ihn mit Strg+X aus und fügen Sie ihn mit Strg+⇧+V auf der darüber liegenden Ebene an der gleichen Stelle wieder ein. Stellen Sie für die eingefügte Instanz ein, dass sie jetzt nicht nur den ersten Frame zeigt, sondern einmal ganz durchspielt.

Wählen Sie nun die eingefügte Bildsequenz aus und konvertieren Sie sie in ein Motion Tween. Wählen Sie Frame 39 aus und fügen Sie mit F6 einen Keyframe ein. Skalieren Sie die Instanz in Frame 39 so, dass sie fast den gesamten Bereich abdeckt (siehe Abbildung 4.13).

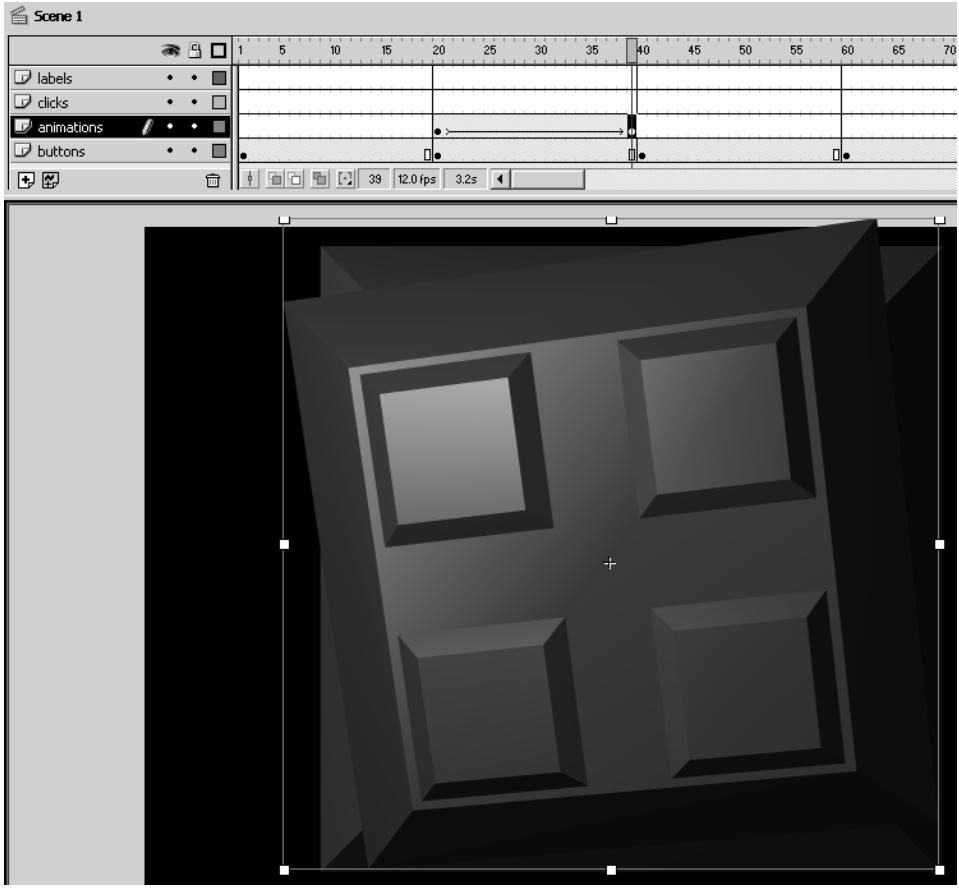


Abbildung 4.13: Skalierung des letzten Bildes der Sequenz

Wenn Sie den Film jetzt testen, werden Sie sehen, dass das Ganze schon eine flüssige Bewegung bildet. Jetzt brauchen wir noch die Rückrichtung. Öffnen Sie dazu die Bibliothek, duplizieren Sie das Symbol `sequenz` und geben Sie dem Duplikat den Namen `sequenzrevers`. Öffnen Sie jetzt dieses Symbol, wählen Sie alle Frames darin aus und kehren Sie deren Reihenfolge um (den entsprechenden Menüpunkt finden Sie im Kontextmenü über die rechte Maustaste).

Gehen Sie nun auf Frame 60, schneiden Sie wieder den linken oberen Button aus und fügen ihn an der gleichen Stelle in der darüber liegenden Ebene ein. Wählen Sie die eingefügte Instanz aus und tauschen Sie das Symbol gegen `sequenzrevers` aus (über den Button auf dem Reiter Instanz). Erzeugen Sie aus der Bildfolge wieder ein Motion Tween und fügen Sie in Frame 79 einen Keyframe ein. Skalieren Sie die Instanz in Frame 60 nun so, dass sie den gesamten Bereich abdeckt.

Fügen Sie jetzt auf der obersten Ebene *labels* die Labels und Befehle ein: Frame 1 erhält das Label `start` und den Befehl `stop()`, Frame 20 erhält das Label `open`, Frame 40 erhält den Befehl `stop()`, Frame 60 das Label

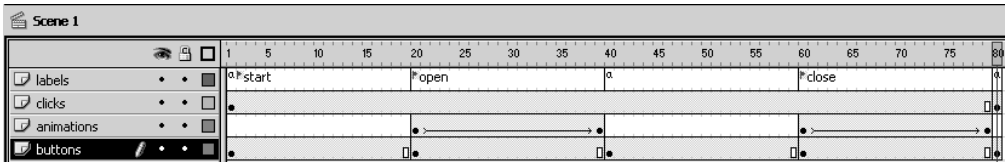


Abbildung 4.14: Die fertige Filmleiste

`close` und Frame 80 den Befehl `gotoAndStop(start)`. Letztendlich sieht die Filmleiste aus wie in Abbildung 4.14.
Es fehlen nur noch die Buttons, um alles zu steuern. Erzeugen Sie dazu auf Ebene 2 im ersten Frame ein Rechteck, das alle Elemente überdeckt. Klicken Sie dann auf diesen ersten Frame, um alles zu selektieren. Drücken Sie F8, um die Selektion in ein Symbol zu verwandeln. Geben Sie dem Symbol den Namen `button` und den Typ Button. Doppelklicken Sie auf den Button und verschieben Sie den Inhalt des ersten Frames des Buttons vom Up-Status zum Hit-Status. Dadurch ist der Button nicht mehr zu sehen, hat aber trotzdem eine aktive Fläche. Wechseln Sie zurück in den Film und geben Sie der Instanz des Buttons folgenden Code:

```
on (release) {
    gotoAndPlay("close");
}
```

Kopieren Sie nun die Instanz des Buttons und skalieren Sie die Kopie so, dass sie nur noch den linken oberen Button überdeckt. Ändern Sie den Code der Kopie wie folgt ab:

```
on (release) {
    gotoAndPlay("open");
}
```

Wenn Sie das Ganze nun speichern und publizieren, können Sie das Menü schon verwenden.

4.5 Beispiel: Spiegelfläche

Der Effekt des folgenden Beispiels ist sehr verblüffend und dabei aber recht einfach umzusetzen. Er lässt sich sogar ohne jegliche der bisher beschriebenen 3D-Hilfsmittel erreichen, sondern einfach nur durch den optischen Eindruck, dass sich ein Objekt in der „hochglanzpolierten" Bodenfläche spiegelt. Um den Effekt noch zu verstärken, habe ich in dem Beispiel auf der CD (*flash\mirror.swf*) allerdings noch den Würfel aus dem ersten Flash-Beispiel mit Swift3D eingesetzt.

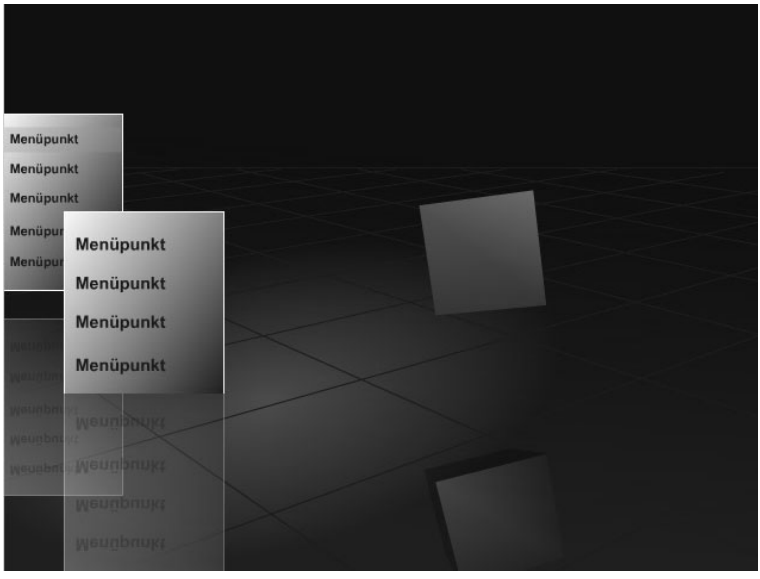


Abbildung 4.15: Beispiel einer Spiegelfläche

Die Erstellung dieses Effektes ist relativ einfach. Den Hintergrund können Sie beliebig gestalten. In dem Beispiel habe ich die Linien eingefügt, um den perspektivischen Eindruck zu verstärken. Außerdem habe ich eine Ellipse eingefügt, die einen radialen Verlauf von Weiß zu Durchsichtig hat, um den Hochglanzeffekt zu erzeugen (siehe Abbildung 4.16).

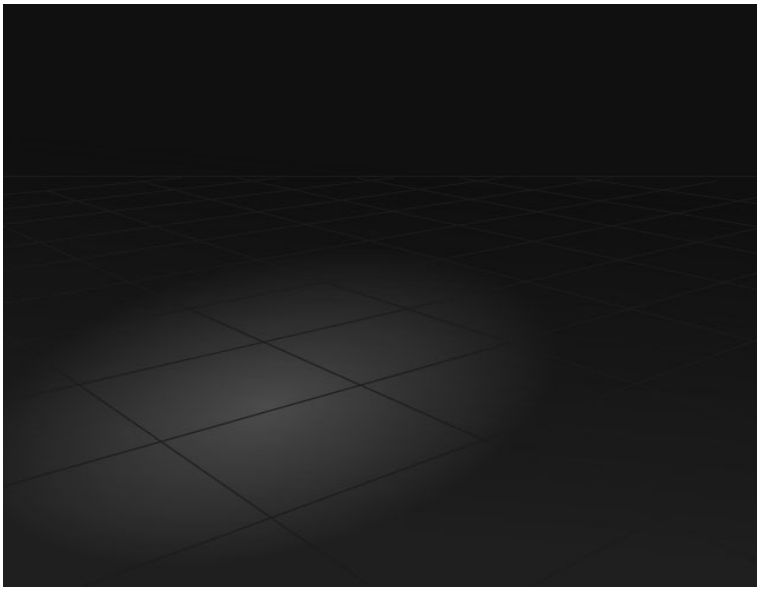


Abbildung 4.16: Der Hintergrund

Nun können Sie beliebige Objekte erzeugen, wie zum Beispiel eine Menütafel wie in der Abbildung. Solch ein Objekt kann aus mehreren anderen Objekten zusammengesetzt sein, sollte aber letztendlich zu einem Symbol zusammengefasst werden. Dieses Symbol wird dann kopiert, vertikal gespiegelt und mit einem Alpha-Effekt (z.B. 30%) versehen (siehe Abbildung 4.17). Bei dem rotierenden Würfel wird genauso verfahren. Wenn man genau hinsieht, kann man allerdings feststellen, dass das Spiegelbild nicht exakt einem wirklichen Spiegelbild des Würfels entspricht. In der Praxis fällt dies aber nicht auf.

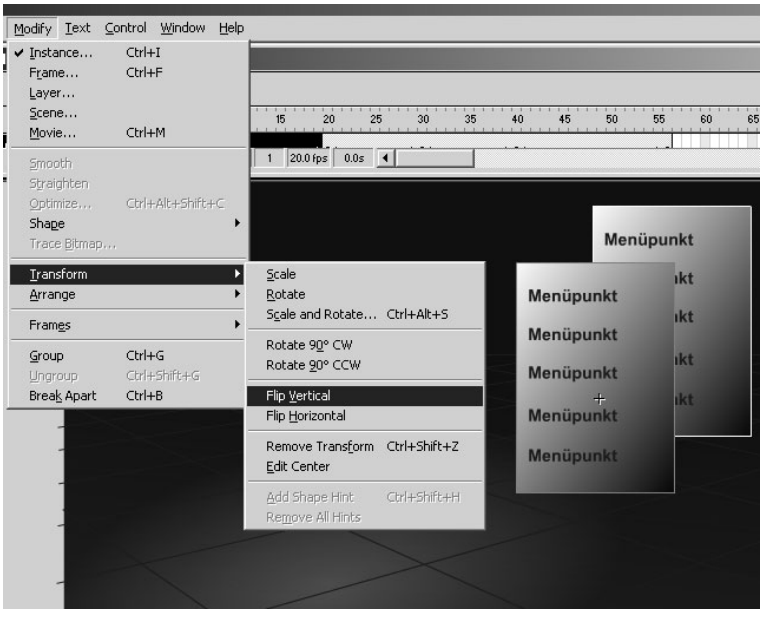


Abbildung 4.17: Das Spiegelbild wird erzeugt

Für den Abstand des Spiegelbildes vom Objekt müssen Sie beachten, dass der Abstand vom Spiegelbild abnimmt, wenn sich das Objekt weiter vom Betrachter wegbewegt (siehe Abbildung 4.18). Das Symbol kann sich natürlich frei bewegen. Sie müssen nur sicherstellen, dass das Spiegelbild immer allen Bewegungen des gespiegelten Symbols folgt.

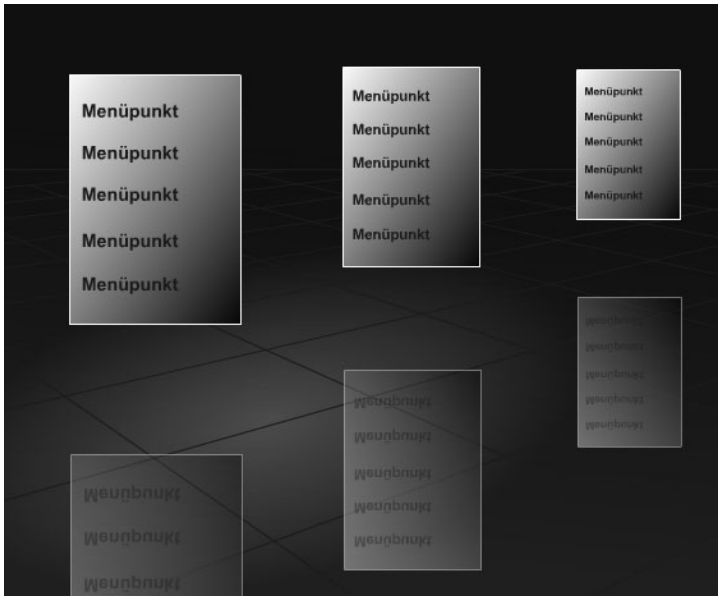


Abbildung 4.18: Verschiedene Positionen mit Spiegelbild

Ein sehr interessanter Effekt ist, wenn das Objekt im Boden verschwindet oder daraus auftaucht. Alles was Sie dafür machen müssen ist, das Symbol und sein Spiegelbild auf verschiedenen Ebenen zu platzieren und mit Masken zu versehen. Am besten ist das zu verstehen, wenn Sie sich dazu das Beispiel auf der CD ansehen.

Die Ebene `menue` enthält das eigentliche Menü, die Ebenen mit der Erweiterung `_m` enthalten jeweils das Spiegelbild zu einer anderen Ebene und die Ebenen mit der Erweiterung `_mask` die Maskierung zu der entsprechenden Ebene (siehe Abbildung 4.19). Wenn Sie die Sichtbarkeit der Masken umschalten, werden Sie sehen, wie diese jeweils platziert sind.

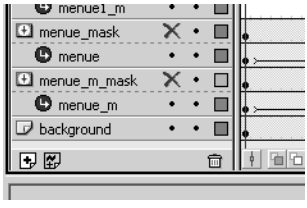


Abbildung 4.19: Maskierung der Ebenen

Der Ablauf des Films selber kann beliebig gestaltet werden. Im Beispiel auf der CD bewegt sich das Menü bei einem Klick auf den ersten Eintrag hin einfach in den Hintergrund und aus dem Boden taucht ein neues Menü auf.

Sie sehen an diesem Beispiel, wie man ohne komplizierte Mathematik einen guten 3D-Eindruck bewirken kann. Komfortabler und exakter wird

das Ganze allerdings, wenn man die Symbol-Größen und -Positionen per Skript berechnet. Die Berechnungen dazu entsprechen denen, die das Molekül-Beispiel im vorangegangenen DHTML-Kapitel benutzt hat. Auch im folgenden Beispiel benutzen wir Sprites und berechnen deren Positionen mit ActionScript.

4.6 Beispiel: Partikelsystem

Partikelsysteme haben einen festen Platz in der 3D-Computergrafik und lassen sich für viele verschiedene visuelle Effekte verwenden. Diese Effekte sind in der Regel die Nachbildung von natürlichen, chaotischen Effekten wie Wasser, Feuer, Rauch oder auch die Bewegung von Tieren.

Im Wesentlichen ist ein Partikelsystem ein Ansammlung von mehreren Partikeln, die einer Reihe bestimmter Regeln gehorchen. Diese Regeln bestimmen, wie ein Partikel entsteht, wie es sich bewegt, wie es sein Aussehen im Laufe der Zeit verändert und wie lange es „lebt". Die Anzahl der Partikel kann im Bereich des digitalen Films von tausenden bis millionen für einen Effekt betragen, für unsere Zwecke müssen wir uns aber mit etwas weniger begnügen, so etwa 30–50. Dies liegt einfach an der Verarbeitungsgeschwindigkeit des Rechners. Aber keine Angst, das ist auch schon recht ansehnlich.

Das folgende Beispiel beschreibt, wie man ein Partikelsystem in Flash programmiert. Die entsprechende Technik orientiert sich hierbei am Sprite-Prinzip. Man kann ein Partikelsystem auch im zweidimensionalen benutzen, aber es passt auch sehr gut in das 3D-Thema, da es sich mit den anderen Techniken kombinieren lässt und sich dadurch sehr attraktive Effekte erzielen lassen.

Als Erstes wollen wir ein kleines Feuerwerk erstellen. Dabei lassen wir aber die Raketen weg und zeigen nur die Explosionen am Himmel. Tatsächlich lässt sich so ein Partikelsystem schon recht einfach erstellen.

Ein einzelner Partikel

Öffnen Sie in Flash einen neuen Film und geben Sie ihm einen schwarzen Hintergrund für den Nachthimmel.

Als Erstes brauchen wir das äußere Erscheinungsbild eines einzelnen Partikels. Dazu erzeugen Sie in Flash zunächst eine runde Kreisfläche ohne Rand. Wählen Sie anschließend die eben erzeugte Kreisfläche aus und öffnen Sie den Fill-Dialog (siehe Abbildung 4.20). Geben Sie der Fläche einen radialen Verlauf von Weiß (innen) über Blau nach durchsichtigem Schwarz. Das Ganze sieht jetzt aus, als wäre es ein glühender Funken. Öffnen Sie anschließend den Info-Dialog und geben Sie der Flä-

che eine Höhe und Breite von 15 Pixeln. Danach können Sie mit F8 die Fläche in ein Symbol umwandeln. Geben Sie dem Symbol den Namen `partikel` und den Typ MovieClip. Geben Sie zudem noch der Instanz des Symbols im ersten Frame den gleichen Namen `partikel`.

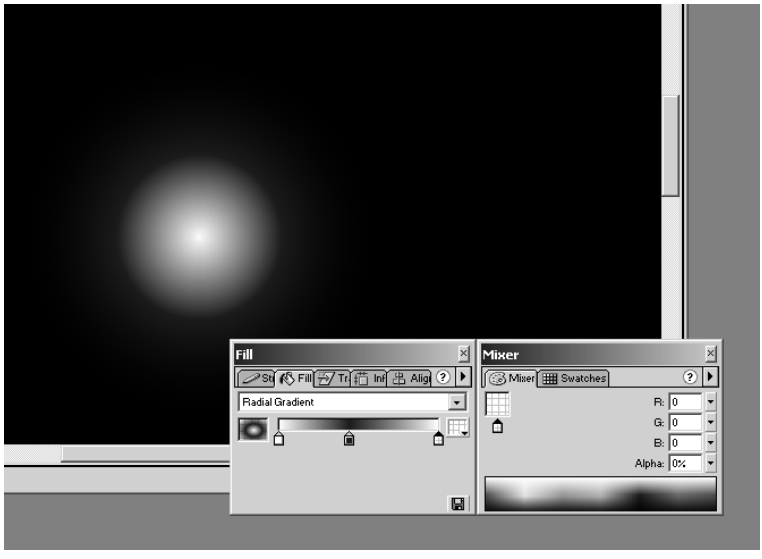


Abbildung 4.20: Ein einzelner Partikel

Gehen Sie nun zu Frame 30 und fügen Sie einen Keyframe mit F6 ein. Dadurch erhält der Film eine Länge von 30 Frames und fängt dann wieder von vorne an.

Der Quellcode

Ein einzelner Partikel ist allerdings tatsächlich etwas wenig. Fügen Sie also in Frame 1 folgendes Skript ein:

```
for (n=0; n < 20; n++) {
    duplicateMovieClip ("partikel", "partikel" + n, n);
}
```

Dieses Skript erzeugt 20 Partikel. Jetzt müssen wir den Partikeln nur noch Leben einhauchen. Gehen Sie dazu durch einen Doppelklick auf den Partikel in den zugehörigen MovieClip. Verlängern Sie hier die Dauer des ersten Frame auf insgesamt drei Frames und fügen Sie einen weiteren Layer für die jeweiligen Frame-Actions ein (siehe Abbildung 4.21).

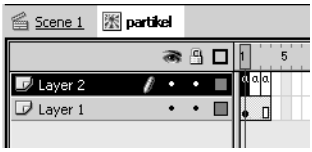


Abbildung 4.21:
Der MovieClip „partikel“

Für den ersten Frame geben Sie folgendes ActionScript ein:

```
xMax = 550;
yMax = 400;
xOrg = xMax / 2;
yOrg = yMax / 2;
step = 0;

xDir = random(30) - 15;
yDir = random(30) - 15;
xPos = 0;
yPos = 0;
life = 100;
alphaDir = random(10) + 2;
```

Hier werden die grundlegenden Initialisierungen für das einzelne Partikel vorgenommen.

- `xMax` und `yMax` enthalten die Breite und Höhe des Flash-Films.
- In `xOrg` und `yOrg` wird dann der Ursprung (der Mittelpunkt des Films) gespeichert.
- `Step` ist eine Laufvariable, die bei jedem Framedurchlauf um eins erhöht wird.
- `xDir` und `yDir` geben einen zufälligen Richtungsvektor an, in dessen Richtung sich das Partikel bewegen soll. Durch die zufällige Länge des Vektors wird damit auch gleichzeitig eine zufällige Geschwindigkeit erzeugt.
- `xPos` und `yPos` enthalten die gegenwärtige Position des Partikels. Da sie hier auf o gesetzt werden, erscheinen alle Partikel zunächst im Mittelpunkt.
- Die Variable `Life` wird mit 100 initialisiert und dann pro Schritt um den zufälligen Wert in `alphaDir` erniedrigt. Den Namen `alphaDir` habe ich gewählt, weil bei diesem Partikelsystem das Alter des Partikels durch seinen Alphawert visualisiert werden soll. D.h. zu Beginn ist das Partikel völlig undurchsichtig und mit fortschreitendem Alter wird es immer transparenter, was den Effekt hat, dass das Partikel quasi verglüht.

Im zweiten Frame werden die Anweisungen platziert, die die Veränderungen des Partikels über die Zeit vornehmen.

```
step ++;
this._alpha = life;

life = life - alphaDir;

if (life <= 0) {
     this.removeMovieClip();
}

if (life > 0) {

  xPos += xDir;
  yPos += yDir;

  yPos += step;
  this._x = xPos + xOrg;
  this._y = yPos + yOrg;

}
```

Im Wesentlichen werden hier die schon zuvor beschriebenen Änderungen vorgenommen, d.h. die Transparenz wird gesetzt und die Position verändert. Sobald der Lebenswert unter o sinkt, wird das Partikel gelöscht.

Bei der Veränderung der Position ist eine Sache noch zu beachten: Zur y-Position wird jeweils noch die aktuelle Schrittzahl hinzuaddiert. Dies bewirkt, dass das Partikel im Laufe der Zeit immer stärker nach unten gezogen wird. Dadurch wird die Schwerkraft simuliert. Über solche Modifikatoren können auch Effekte wie zum Beispiel Wind visualisiert werden.

Im dritten Frame schließlich steht nur noch der Befehl, zum zweiten Frame zurückzuspringen.

```
gotoAndPlay(2);
```

Wenn Sie das Beispiel ausprobieren, werden Sie sehen, dass das Beispiel zwar schon recht effektvoll ist, aber als echtes Feuerwerk noch etwas monoton wirkt. Sie können aber leicht selbst Erweiterungen einführen, indem Sie den Startpunkt zufällig auswählen und die Farbe des Partikels zufällig ändern.

Partikel im 3D-Raum

Eine weitere Erweiterung, die ich oben schon erwähnte, ist, die Partikel im dreidimensionalen Raum zu bewegen. Wir bleiben dafür bei einem ähnlichen Beispiel: Die Partikel erscheinen im Mittelpunkt, bewegen sich in zufälliger Richtung bis zu einer bestimmten konstanten Entfernung zum Mittelpunkt und „verglühen“ dann. Dabei wird die ganze Szene um den Mittelpunkt gedreht.

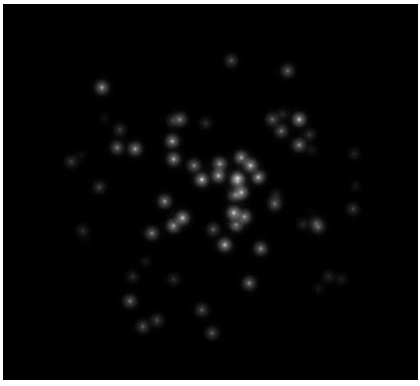


Abbildung 4.22: Partikel im Raum

Für dieses Beispiel öffnen Sie bitte die Datei *partikel3d.fla*.

```
step = 0;

xDir = random(30) - 15;
yDir = random(30) - 15;
zDir = random(30) - 15;
xPos = 0;
yPos = 0;
zPos = 0;

life = 100;
alphaDir = random(10) + 2;
```

Die Initialisierung bleibt im Wesentlichen gleich, nur dass wir jetzt noch jeweils ein Komponente für die z-Koordinate benötigen.

```
step ++;
this._alpha = life;

life = life - alphaDir;

l = Math.sqrt(xPos*xPos + yPos*yPos + zPos*zPos);
```

Hier wird die Entfernung zum Nullpunkt berechnet.

```
if (life <= 0) {
     gotoAndPlay(1);
}
```

Falls die Lebensspanne abgelaufen ist, wird das Partikel „neu gestartet“.

```
if (life > 0) {

  if (l <= 100) {
  xPos += xDir;
  yPos += yDir;
  zPos += zDir;
  }
```

Nur, wenn eine bestimmte Entfernung zum Mittelpunkt noch nicht erreicht wurde, wird das Partikel weiter bewegt.

```
  //Roation
  w = Math.PI/180 * _root.angle;
  x = xPos * Math.cos(w) + zPos * Math.sin(w);
  y = yPos;
  z = (-1) * xPos * Math.sin(w) + zPos * Math.cos(w);

  //Projektion
  this._x = ( - x * (-300)) / (z + 300) + 400;
  this._y = ( - y * (-300)) / (z + 300) + 300;
}
```

Die Rotation wird durch einen zentralen Winkel bestimmt.

Nachdem dieses Beispiel schon etwas mehr Mathematik verwendet hat, um die Position eines Partikels zu bestimmen, beruht das nächste Kapitel fast vollständig auf berechneter Grafik.

4.7 Real berechnete 3D-Grafik

Kommen wir nun zur „Königsdisziplin“, der real berechneten dreidimensionalen Darstellung.

Auch bei Flash (bis Version 5) gibt es hierbei das Problem, dass Linien oder Flächen nicht dynamisch erzeugt werden können, wie es für die 3D-Darstellung nötig wäre. Aber ebenso wie im vorhergehenden Kapitel zu DHTML ist es auch in Flash möglich, diese Elemente über Umwege zu erzeugen.

Drahtgittermodell

Fangen wir zunächst damit an, ein Drahtgittermodell zu erzeugen. Eine Linie zwischen zwei beliebigen Punkten in der Fläche zu zeichnen funktioniert relativ einfach. Wir definieren im Flash-Film zunächst einen MovieClip mit einer Linie im 45° Winkel (siehe Abbildung 4.23). Diese Linie zeichnen Sie am besten mit dem Linienwerkzeug bei gedrückter Shifttaste, damit der Winkel genau 45° beträgt. Wählen Sie anschließend die

Linie aus und erzeugen Sie daraus einen MovieClip und geben Sie der Instanz im Film den Namen `baseline`. Wechseln Sie durch einen Doppelklick auf die Linie in den MovieClip und bringen Sie das Info-Fenster in den Vordergrund (siehe Abbildung). Hier geben Sie für `x` und `y` jeweils 0 ein, damit die Linie genau im Ursprung beginnt.

Später kann dieser MovieClip nun entsprechend mit ActionScript in Breite und Höhe angepasst werden, um eine beliebige Linie zu erzeugen. Im folgenden Quellcode gehe ich darauf an der entsprechenden Stelle noch näher ein.

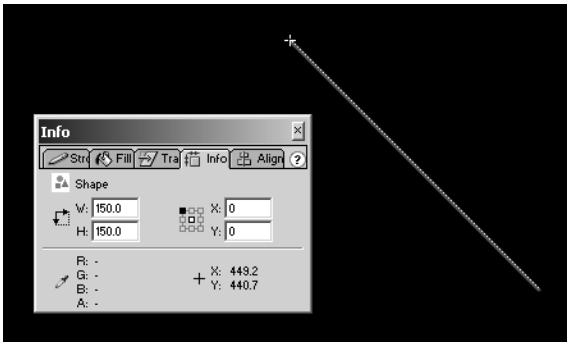


Abbildung 4.23: Die Basislinie

Das folgende Beispiel stellt einen Würfel dar, der sich im Raum dreht. Dieses Beispiel kann die Grundlage für beliebige andere Modelle und Animationen sein.

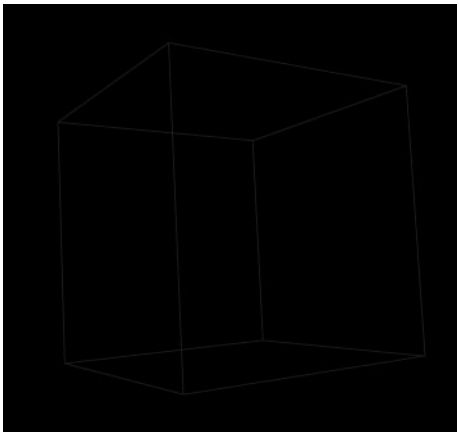


Abbildung 4.24: Der real berechnete Würfel in Drahtgitterdarstellung

```
var point = new Array(8);
var pointTrans = new Array(8);
var pointProj = new Array(8);
var pointCount = 0;
var line = new Array(12);
var lineCount = 0;
```

Hier werden drei Felder für die Eckpunkte definiert:

- Für die Ursprungspunkte
- Für die transformierten Punkte
- Für die projizierten Punkte

Außerdem wird ein Feld definiert, welches die Linien enthalten wird. Die Größen der Felder werden hier schon mit der exakten Anzahl vorbestimmt. Für ein anderes Objekt müssen die Größen natürlich angepasst werden.

```
var camera = new Vector(0.0,0.0,-500.0);
var rx=0,ry=0,rz=0;
```

Die Kameraposition wird festgelegt. Außerdem werden die Rotationen um die x-,y- und z-Achse mit o initialisiert.

```
addPoint(-100,100,-100);
addPoint(100,100,-100);
addPoint(100,100,100);
addPoint(-100,100,100);
addPoint(-100,-100,-100);
addPoint(100,-100,-100);
addPoint(100,-100,100);
addPoint(-100,-100,100);
```

Die Eckpunkte des Würfels werden definiert, und zwar relativ zum Koordinatenursprung. Der Koordinatenursprung ist damit auch der Punkt, um den die Eckpunkte später rotiert werden.

```
addLine(0,1);
addLine(1,2);
addLine(2,3);
addLine(3,0);
addLine(4,5);
addLine(5,6);
addLine(6,7);
addLine(7,4);
addLine(0,4);
addLine(1,5);
addLine(2,6);
addLine(3,7);
```

Die Linien des Modells werden definiert, indem im entsprechenden Feld die Indizes der beiden Eckpunkte gespeichert und jeweils miteinander zu einer Linie verbunden werden.

```
function Vector(x,y,z) {this.x = x; this.y = y; this.z = z;
return this;}
```

Diese Funktion definiert ein Objekt mit dem Namen Vector. Dieses Objekt ist für den weiteren Umgang mit Eckpunkten ganz praktisch. Man könnte sich hier auch gleich eine komplette Bibliothek mit Vektor- und Matrixfunktionen aufbauen, was aber für dieses einfach Beispiel zu weit führt.

```
function addPoint(x,y,z) {
    _root.point[_root.pointCount] = new Vector(x,y,z);
    _root.pointTrans[_root.pointCount] =
                                    new Vector(x,y,z);
    _root.pointProj[_root.pointCount] =
                                    new Vector(x,y,z);
    _root.pointCount++;
}
```

Diese Funktion fügt einen Punkt zu dem Objekt hinzu. Es werden damit auch gleich die anderen Felder für die transformierten und projizierten Punkte vorbelegt.

```
function addLine(p1,p2) {
    _root.line[_root.lineCount] = new Array(2);
    _root.line[_root.lineCount][0] = p1;
    _root.line[_root.lineCount][1] = p2;
    duplicateMovieClip(_root.baseline,
          "baseline"+_root.lineCount,_root.lineCount+50);
    _root.lineCount++;
}
```

Die Funktion `addLine` fügt eine Linie hinzu. Die Übergabeparameter bestimmen dabei die Indizes der Eckpunkte im Punktefeld. Gleichzeitig wird auch das Liniensymbol dupliziert, mit dem dann später die entsprechende Linie gezeichnet wird. Als Tiefeninformation (der letzte Parameter in `duplicateMovieClip`) wird die Liniennummer plus 50 angegeben. Das wird deshalb gemacht, damit die Linien über den Flächen zu sehen sein werden, wenn wir in einer Erweiterung des Beispiels noch Flächen hinzufügen.

```
function transformiere(rx,ry,rz,tx,ty,tz) {
      var i,x,y,z;
      rx = Math.PI/180.0 * rx;
      ry = Math.PI/180.0 * ry;
      rz = Math.PI/180.0 * rz;
      for (i = 0;i < _root.pointCount; i++) {
          x = _root.point[i].x;
          y = _root.point[i].y;
          z = _root.point[i].z;
          // rotation um die x-Achse
          _root.pointTrans[i].y = y * Math.cos(rx)
                                    - z * Math.sin(rx);
```

```
            _root.pointTrans[i].z = y * Math.sin(rx)
                                    + z * Math.cos(rx);
            // rotation um die y-Achse
            y = _root.pointTrans[i].y;
            z = _root.pointTrans[i].z;
            _root.pointTrans[i].x = x * Math.cos(ry)
                                    + z * Math.sin(ry);
            _root.pointTrans[i].z = (-1) * x * Math.sin(ry)
                                    + z * Math.cos(ry);
          _root.pointTrans[i].x += tx;
          _root.pointTrans[i].y += ty;
          _root.pointTrans[i].z += tz;
        }
    }
```

Hier werden die Eckpunkte transformiert. Die Parameter geben jeweils die Rotation und Translation (Verschiebung) um die x-,y- und z-Achse an. Dabei ist zu beachten, dass zuerst immer rotiert und anschließend erst verschoben wird. Das heißt, dass die Punkte immer um den Ursprung rotiert werden. Möchte man die Rotation um einen anderen Punkt vornehmen, müsste man die Punkte erst entsprechend verschieben.

```
function projeziere() {
    var i;

    for (i=0; i < _root.pointCount; i++) {
           _root.pointProj[i].x =
                (- _root.pointTrans[i].x * camera.z) /
                (_root.pointTrans[i].z - camera.z) + 300;
           _root.pointProj[i].y =
                (- _root.pointTrans[i].y * camera.z) /
                (_root.pointTrans[i].z - camera.z) + 200;
           _root.pointProj[i].z = 0;
    }
}
```

In dieser Funktion werden die Punkte in die Bildschirmebene projiziert. Dabei wird davon ausgegangen, dass der Film 600 Pixel breit und 40 Pixel hoch ist, das heißt der Ursprung wird um 300 Pixel nach rechts und 200 Pixel nach unten verschoben. Für die Kamera wird, um das Ganze einfach zu halten, nur die z-Koordinate berücksichtigt. Wie man komplexere Kamerapositionen berechnet, ist im Grundlagenkapitel beschrieben.

```
function drawline(nr) {
    var w,h,x1,y1,x2,y2;
    var div;
    var x1 = _root.pointProj[_root.line[nr][0]].x;
    var y1 = _root.pointProj[_root.line[nr][0]].y;
    var x2 = _root.pointProj[_root.line[nr][1]].x;
    var y2 = _root.pointProj[_root.line[nr][1]].y;

    h = (y2 - y1);
    w = (x2 - x1);
    div = eval("baseline"+nr);
    div._x = x1;
    div._y = y1;

    div._width = Math.abs(w);
    div._height = Math.abs(h);
    if (w < 0) div._xscale = -div._xscale;
    if (h < 0) div._yscale = -div._yscale;

}
```

Hier wird schließlich eine Linie gezeichnet, wie weiter oben beschrieben.

Die restlichen Skripte sind recht einfach. In Frame 2 wird folgendes Skript platziert:

```
var t;

rx+= 5;ry += 3;rz += 7;

_root.transformiere(rx,ry,rz,0,0,0);
_root.projeziere();

for (t=0; t<_root.lineCount; t++) {
    _root.drawLine(t);

}
```

Die Rotationswinkel werden verändert, die Transformation wird durchgeführt und die Punkte projiziert. Anschließend werden die Linien gezeichnet.

In Frame 3 schließlich wird zurück zu Frame 2 gesprungen, womit wir eine Endlosschleife haben.

```
gotoAndPlay(_currentFrame - 1);
```

Um dieses Beispiel zu erweitern, kann man nun beliebig Punkte und Verbindungslinien einfügen. Dabei ist nur zu beachten, dass die Feldgrößen entsprechend angepasst werden.

Schattierte Seitenflächen

Eine wirklich reizvolle Erweiterung wäre es, die Seitenflächen des Würfels auch gefüllt und schattiert darstellen zu können. Zunächst scheint es nicht möglich zu sein, beliebige Dreiecke mit Flash (bis Version 5) dynamisch zu zeichnen. Tatsächlich findet man aber ab und zu im Web Flash-Filme, die genau das tun. Dieser Abschnitt wird Ihnen zeigen, wie es geht.

In 3D-Programmen werden in der Regel keine beliebigen Polygone gefüllt, sondern immer nur Dreiecke, weil das erheblich performanter durchzuführen ist. Dazu werden ggf. vorhandene Polygone vorher trianguliert. Bei unserem Würfelbeispiel ist das sehr einfach. Jede Seitenfläche besteht aus zwei rechtwinkligen Dreiecken. Das Problem der Flächenfüllung lässt sich also reduzieren auf: Wie kann man mit Flash ein beliebig geformtes Dreieck dynamisch generieren mit den zur Verfügung stehenden Transformationsmöglichkeiten (Skalieren in Höhe und Breite und Rotieren eines vordefinierten Objektes)?

Die Lösung ist, dass man das Dreieck weiter unterteilt in zwei Dreiecke (siehe Abbildung 4.25). Wenn man also ein Dreieck hat, definiert durch die Eckpunkte ABC, zerlegt man dieses in zwei Dreiecke ACD und BCD. Diese beiden Dreiecke sind auf jeden Fall rechtwinklig und können damit durch ein einziges vordefiniertes Symbol dargestellt werden, indem man das vordefinierte Dreieck entsprechend skaliert und rotiert.

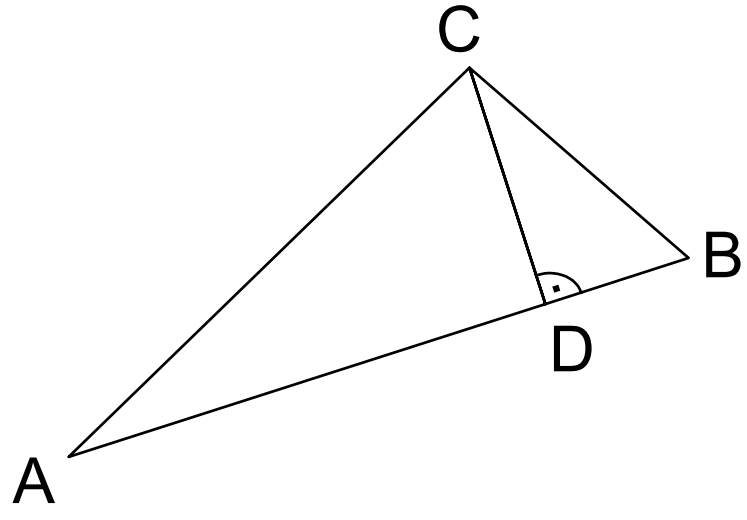


Abbildung 4.25: Ein beliebiges Dreieck wird in zwei rechtwinklige Dreiecke zerlegt.

Das Grundprinzip ist damit erklärt. Etwas komplizierter ist die Umsetzung davon. Wie findet man z.B. den Punkt D? Der Punkt D ist die Projektion des Punkts C auf die Linie AB. Dadurch, dass wir ein beliebiges Dreieck zeichnen können müssen, ist allerdings nicht immer gewährleistet, dass der Punkt D dann zwischen A und B liegt. Deshalb müssen die Punkte ggf. durchgetauscht werden.

Wenn das vordefinierte Dreieck seinen Ursprung in der Ecke mit dem rechten Winkel hat, verschiebt man dieses nun in den Punkt D, rotiert es um den entsprechenden Winkel und skaliert die Höhe auf die Länge von DC und die Breite auf die Länge von AD. Ggf. muss das Dreieck noch gespiegelt werden.

Wie diese Berechnungen durchgeführt werden, erkläre ich am besten direkt am Sourcecode.

Das vorhergehende Beispiel werden wir jetzt also so erweitern, dass die Seitenflächen des Würfels gefüllt sind, dass nur die sichtbaren Seitenflächen gezeichnet werden und dass die Helligkeit der Seitenflächenfarbe proportional zum Winkel des einfallenden Lichtes ist. Das Ganze sieht dann so aus, wie in Abbildung 4.26 zu sehen ist.

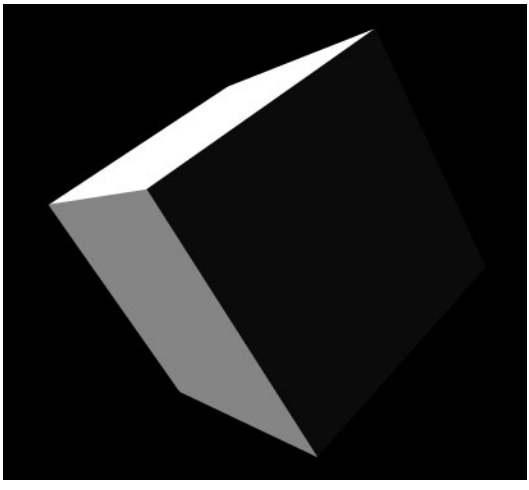


Abbildung 4.26: Der Würfel mit gefüllten und schattierten Flächen

Bevor wir mit dem Quellcode beginnen, sollten wir in Flash noch die Basisdreiecke erzeugen. Gehen Sie dazu wie folgt vor:

1. Erzeugen Sie ein weißes Quadrat.
2. Ziehen Sie eine andersfarbige Linie mit gedrückter Shifttaste durch die linke untere Ecke des Quadrates. Dadurch können Sie anschließend den rechten oberen Teil und danach die Linie selbst löschen und erhalten so ein rechtwinkliges Dreieck (siehe Abbildung 4.27).
3. Wählen Sie das Dreieck aus und konvertieren Sie es in einen MovieClip. Geben Sie der Instanz im Film den Namen `tri0`.

4. Wechseln Sie durch Doppelklick auf das Dreieck in den MovieClip. Verschieben Sie das Dreieck dort mit der linken unteren Ecke in den Ursprung (siehe Abbildung 4.28).
5. Wechseln Sie wieder in den Hauptfilm und kopieren Sie die Instanz des Dreiecks. Geben Sie der Zweiten Instanz den Namen `tri1` und spiegeln Sie das zweite Dreieck horizontal (siehe Abbildung 4.29).
6. Verschieben Sie die Instanzen außerhalb des sichtbaren Bereichs, damit sie nicht stören.

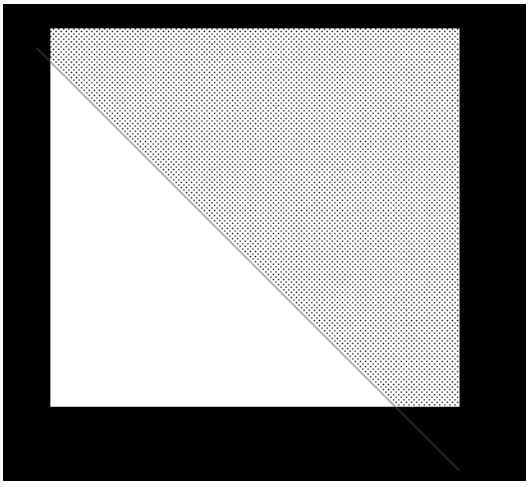


Abbildung 4.27: Ein rechtwinkliges Dreieck wird erzeugt.

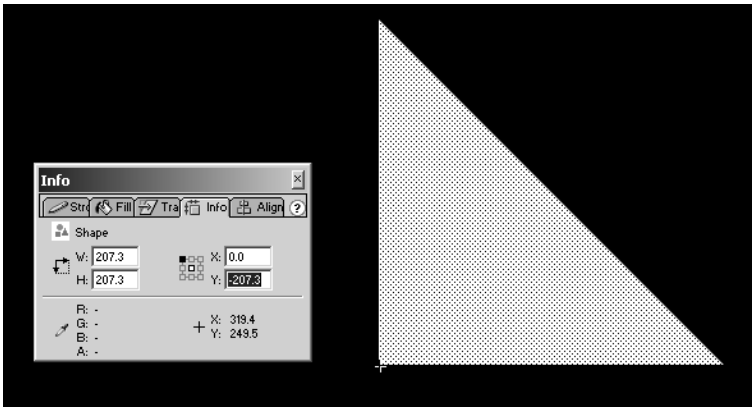


Abbildung 4.28: Das Dreieck muss mit der linken unteren Ecke genau im Ursprung liegen.

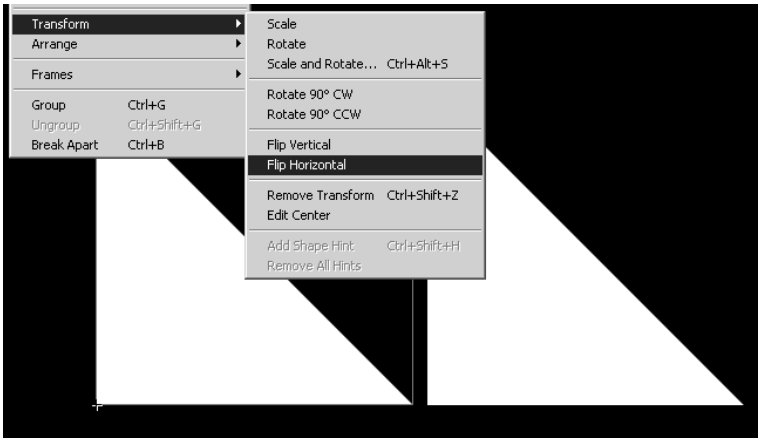


Abbildung 4.29: Per Kopieren und Spiegeln wird das zweite Basisdreieck erzeugt.

Die folgenden Codestücke stellen nur Funktionen dar, die zusätzlich zu den schon bestehenden eingefügt werden müssen.

```
var tri = new Array(12);
var triCount = 0;
```

Zusätzlich zu den Punkten und Linien benötigen wir nun noch ein Feld für die Dreiecksdaten.

```
addTri(0,3,1);
addTri(1,3,2);
addTri(0,1,4);
addTri(1,5,4);
addTri(5,1,2);
addTri(2,6,5);
addTri(2,3,7);
addTri(2,7,6);
addTri(7,3,0);
addTri(7,0,4);
addTri(7,4,5);
addTri(7,5,6);

function addTri(p1,p2,p3) {
    _root.tri[_root.triCount] = new Array(3);
    _root.tri[_root.triCount][0] = p1;
    _root.tri[_root.triCount][1] = p2;
    _root.tri[_root.triCount][2] = p3;
    duplicateMovieClip(_root.tri0,"tri0"+_root.triCount,
                                    _root.triCount*2);
    duplicateMovieClip(_root.tri1,"tri1"+_root.triCount,
                                   _root.triCount*2+1);
    _root.triCount++;
}
```

Die Funktion `addTri` definiert ein Dreieck, wobei die übergebenen Nummern die Indizes der Eckpunkte im Punktfeld sind. Bei der Reihenfolge der Nummern ist zu beachten, dass die Nummern im Uhrzeigersinn angegeben werden. Dies ist notwendig, um später bestimmen zu können, ob dem Betrachter die Rückseite oder die Vorderseite des Dreiecks zugewandt ist. Diese Information wird benutzt, um die Sichtbarkeit des Dreiecks zu bestimmen.

In der Funktion selbst werden die Basisdreiecke dupliziert, für jedes Dreieck zwei Basisdreiecke, wie oben beschrieben.

Jetzt wird es spannend, denn jetzt kommt die Funktion, die ein Dreieck tatsächlich zeichnet. In dieser Funktion sind aber noch zwei zusätzliche Funktionalitäten mit abgedeckt, die sich um Sichtbarkeit und die Einfärbung kümmern.

```
function drawTri(nr) {
    var t,xt,yt,n,xd,yd,a;
    var    trib = eval("_root.tri1"+nr);
    var    tria = eval("_root.tri0"+nr);
```

An die Funktion wird nur die Nummer des zu zeichnenden Dreiecks übergeben.

In `tria` und `trib` werden die Referenzen auf die schon bestehenden duplizierten Basisdreiecke gespeichert, um später einfacher darauf zugreifen zu können.

```
    var x1 = _root.pointProj[_root.tri[nr][0]].x;
    var y1 = _root.pointProj[_root.tri[nr][0]].y;
    var x2 = _root.pointProj[_root.tri[nr][1]].x;
    var y2 = _root.pointProj[_root.tri[nr][1]].y;
    var x3 = _root.pointProj[_root.tri[nr][2]].x;
    var y3 = _root.pointProj[_root.tri[nr][2]].y;
```

Hier werden in lokalen Variablen die Werte der projizierten Eckpunkte, d.h. der Bildschirmkoordinaten, des Dreiecks gespeichert.

```
    var xa = _root.pointTrans[_root.tri[nr][2]].x
                - _root.pointTrans[_root.tri[nr][0]].x;
    var ya = _root.pointTrans[_root.tri[nr][2]].y
                - _root.pointTrans[_root.tri[nr][0]].y;
    var za = _root.pointTrans[_root.tri[nr][2]].z
                - _root.pointTrans[_root.tri[nr][0]].z;
    var xb = _root.pointTrans[_root.tri[nr][1]].x
                - _root.pointTrans[_root.tri[nr][0]].x;
    var yb = _root.pointTrans[_root.tri[nr][1]].y
                - _root.pointTrans[_root.tri[nr][0]].y;
    var zb = _root.pointTrans[_root.tri[nr][1]].z
                - _root.pointTrans[_root.tri[nr][0]].z;
    var xc = ya*zb - za*yb;
    var yc = za*xb - xa*zb;
    var zc = xa*yb - ya*xb;
```

Die vorhergehenden Zeilen sind ausschließlich notwendig für die Berechnung der Flächenfarbe. Dazu wird aus den transformierten 3D-Koordinaten des Dreiecks der Normalenvektor `(xc,yc,zc)` der Dreiecksfläche mittels des Vektorkreuzproduktes berechnet. Die eigentliche Farbberechnung findet weiter unten statt.

```
vis = (y2-y1)*x3 + (x1-x2)*y3 - (y2-y1)*x1
                                        - (x1-x2)*y1;
if (vis < 0) {
    tria._visible = false;
    trib._visible = false;
    return;
} else {
    tria._visible = true;
    trib._visible = true;
}
```

In der Variable `vis` wird berechnet, ob die projizierten Bildschirmkoordinaten des Dreiecks im Uhrzeigersinn vorliegen (`vis` größer 0) oder nicht (`vis` kleiner 0). Falls nicht, dann sähe der Betrachter die Rückseite des Dreiecks, was bedeutet, dass das Dreieck nicht gezeichnet werden muss. Die entsprechenden Basisdreiecke werden also ausgeblendet und die Funktion verlassen.

```
c = ((xc+yc-zc)/(Math.sqrt(xc*xc+yc*yc+zc*zc)));
```

In `c` wird eine Vektormultiplikation von einem imaginären Lichtquellenvektor (1,1,-1) und der Dreiecksnormale berechnet und durch die Länge der beiden Vektoren geteilt. Dies ergibt den Cosinus des Winkels zwischen den beiden Vektoren, d.h. den Wert 1, wenn das Licht direkt auf die Fläche scheint, und einen kleiner werdenden Wert, je größer der Winkel zwischen den beiden Vektoren wird (bis zu –1).

```
xa=x1;ya=y1;xb=x2;yb=y2;xc=x3;yc=y3;
t = xc*xc + yc*yc - xb*xc - yb*yc - xa*xc - ya*yc
                                    + xa*xb + ya*yb;
if (t > 0) {
    xa=x1;ya=y1;xb=x3;yb=y3;xc=x2;yc=y2;
    t = xc*xc + yc*yc - xb*xc - yb*yc - xa*xc
                              - ya*yc + xa*xb + ya*yb;
    if (t > 0) {
        xa=x3;ya=y3;xb=x2;yb=y2;xc=x1;yc=y1;
        t = xc*xc + yc*yc - xb*xc - yb*yc - xa*xc
                              - ya*yc + xa*xb + ya*yb;
    }
}
```

Diese Zeilen bewirken, dass anschließend in `(xc,yc)` der Punkt gespeichert ist, der zwischen den Punkten `(xa,ya)` und `(xb,yb)` liegt. Die Formel ergibt sich aus folgenden Überlegungen: Der Vektor von A nach B wird als Normalenvektor für zwei Flächen durch die Punkte A und B ge-

nommen. In der Hesse-Form dieser Flächen wird der Punkt C eingesetzt. Wenn man in die Hesse-Form einen Punkt einsetzt, ergibt das Ergebnis einen positiven oder negativen Wert, je nachdem auf welcher Seite der Fläche der Punkt liegt. Das heißt, wenn der Punkt zwischen den beiden Flächen liegt, muss einmal ein positiver und einmal ein negativer Wert herauskommen. Multipliziert man diese beiden Werte, ergibt sich also nur ein negativer Wert. Genau das wird hier überprüft. Wenn der Wert nicht negativ ist, Punkt C also nicht zwischen A und B liegt, werden die Punkte durchgetauscht und erneut geprüft.

```
xt = yb - ya;
yt = xa - xb;
n = 1.0 / Math.sqrt(xt*xt + yt*yt);
xt = n*xt;
yt = n*yt;
t = xt*xc + yt*yc - xt*xa - yt*ya;
xd = xc - t*xt;
yd = yc - t*yt;
```

Hier werden die Koordinaten des Punkt D `(xd,yd)` berechnet.

```
a = Math.atan((ya-yd)/(xa-xd))*180/Math.PI;
if (xa - xd < 0) a = a - 180;
```

Der Winkel, um den das Basisdreieck gedreht werden muss, wird berechnet.

```
if (t > 0) {
    tria = eval("_root.tri1"+nr);
    trib = eval("_root.tri0"+nr);
}
```

Ggf. müssen die beiden Basisdreiecke noch vertauscht werden, je nach Lage des Dreiecks.

```
cola = new Color(tria);
colb = new Color(trib);
colTrans = new Object();
colTrans.ra = colTrans.ga = colTrans.ba = c*100 + 80;
colTrans.rb = colTrans.gb = 0;
colTrans.bb = 255;
colTrans.bb = 255;
cola.setTransform(colTrans);
colb.setTransform(colTrans);
```

Hier findet nun die Farbgebung statt. Dazu werden zunächst Farbobjekte für die beiden Basisdreiecke erzeugt. In einem Farbtransformationsobjekt wird ein Blauton angelegt, der proportional zum weiter oben berechneten Wert c ist.

```
    tria._rotation = 0;
    tria._height = Math.sqrt((xd-xc)*(xd-xc)+(yd-yc)
                                                *(yd-yc));
    tria._width = Math.sqrt((xd-xa)*(xd-xa)+(yd-ya)
                                                *(yd-ya));
    tria._rotation = a;
    tria._x = xd;
    tria._y = yd;
```

Bevor alle errechneten Transformationen auf ein Basisdreieck angewendet werden können, wird die Rotation zunächst auf 0 zurückgesetzt. Danach wird die Höhe und Breite auf die Länge des jeweiligen Vektors gesetzt, die Rotation durchgeführt und das Dreieck an die entsprechende Position verschoben.

```
    a = Math.atan((yb-yd)/(xb-xd))*180/Math.PI;
    if (xb - xd < 0) a = a - 180;
    trib._rotation = 0;
    trib._x = xd;
    trib._y = yd;
    trib._height = Math.sqrt((xd-xc)*(xd-xc)+(yd-yc)
                                                *(yd-yc));
    trib._width = Math.sqrt((xd-xb)*(xd-xb)+(yd-yb)
                                                *(yd-yb));
    trib._rotation = a;
}
```

Die gleichen Berechnungen finden auch für das zweite Basisdreieck statt.

Was jetzt noch fehlt, ist, dass wir in der Schleife in Frame 2 nicht die Linien, sondern die Dreiecke zeichnen. Das Skript in Frame 2 sieht also folgendermaßen aus:

```
var t;

rx+= 5;ry += 3;rz += 7;

_root.transformiere(rx,ry,rz,0,0,0);
_root.projeziere();

for (t=0; t<_root.triCount; t++) {
    _root.drawTri(t);

}
```

Damit haben wir es geschafft. Der Würfel dreht sich in seiner ganzen Pracht. Sie können nun nach Belieben neue Formen und Animationen ausprobieren.

Ein Beispiel mit Tiefensortierung zeigt der folgende Abschnitt.

Es gibt allerdings ein paar Punkte, die zu beachten sind:

Da die Sichtbarkeit der Dreiecke hier nur über die Orientierung gelöst wird, funktioniert das Ganze nur korrekt für konvexe Objekte. Wenn man die Darstellung für allgemeine Objekte korrekt haben möchte, muss man noch eine Tiefensortierung der Dreiecke durchführen, damit sich weiter weg liegende Flächen nicht vor weiter vorne liegenden Flächen befinden. Eine Möglichkeit, dieses Problem zu umgehen, ist, den Flächen eine Transparenz von 50% zu geben. Das ist ein netter Effekt und man spart sich die Sortierung, da die Reihenfolge nicht mehr auffällt.

Da die Berechnungen alle in ActionScript durchgeführt werden, ist die Leistungsfähigkeit nicht besonders hoch. Wenn man also sehr komplexe Objekte darstellen möchte, sollte man eine Geschwindigkeitsoptimierung durchführen. Z.B. ist es möglich, die Sinus- und Cosinus-Werte vorzuberechnen und in einem Feld zu speichern.

Texturen für die Flächen sind eine Sache, die nun definitiv nicht geht, auch wenn wir schon recht weit gekommen sind mit den Möglichkeiten von Flash. Für Texturen müssen Sie bis zum nächsten Kapitel über Director Shockwave warten.

4.8 Flash MX

Im März 2002 kam die neueste Version von Flash heraus. Diese Version entspräche eigentlich der Version 6, wurde aber von Macromedia Flash MX getauft. Diese neue Version hat viele interessante Neuerungen, vor allem eine, die für unsere Bemühungen interessant ist: die neue Zeichenfunktion.

Mit den neuen Zeichenfunktionen ist es möglich, in einem MovieClip beliebige Linien und Flächen zu zeichnen. Es gibt zwar außerdem noch die Möglichkeit, Symbol-Graphics perspektivisch zu verzerren, da sich das aber nicht auf importierte Bilder anwenden lässt, ist diese Funktion weniger interessant für uns.

Was genau leisten die neuen Funktionen? Die folgende Liste beschreibt die neuen in ActionScript verfügbaren Funktionen:

- Anlegen eines leeren MovieClips: Es ist möglich, einen leeren MovieClip mit der Funktion `createEmptyMovieClip` anzulegen, in den dann gezeichnet werden kann.
- Zeichnen von Linien: Ein MovieClip besitzt nun einen Grafikcursor. Dies ist die virtuelle Position eines Zeichenstiftes. Man kann nun die `lineTo`-Funktion benutzen, um von der aktuellen Position des Grafikcursors eine Linie zu einem beliebigen Punkt zu zeichnen. Zuvor kann man noch bestimmen, welche Stärke, Farbe und welchen Alpha-Wert die Linie haben soll.

- Zeichnen von Füllungen: Wenn man mit der oben beschriebenen Linienfunktion ein geschlossenes Polygon zeichnet, kann man zuvor mit `beginFill` angeben, dass das Polygon gefüllt gezeichnet werden soll. Nach dem Zeichnen der Linien muss allerdings mit `endFill` noch angegeben werden, wann das Polygon fertig gestellt ist. Auch hier können vor dem Zeichnen der Füllung die Farbe und der Alpha-Wert angegeben werden.
- Löschen der gezeichneten Inhalten: Mit der Funktion `clear` können die mit den Zeichenfunktionen gezeichneten Inhalte wieder gelöscht werden, zum Beispiel wenn ein Objekt transformiert wurde und neu gezeichnet werden muss.

Für ein Beispiel mit Flash MX nehmen wir als Grundlage den Film aus dem letzten Beispiel. In diesem sahen die Funktionen zum Zeichnen der Linien und Dreiecke folgendermaßen aus:

```
function drawline(nr) {
    var w,h,x1,y1,x2,y2;
    var div;
    var x1 = _root.pointProj[_root.line[nr][0]].x;
    var y1 = _root.pointProj[_root.line[nr][0]].y;
    var x2 = _root.pointProj[_root.line[nr][1]].x;
    var y2 = _root.pointProj[_root.line[nr][1]].y;

    h = (y2 - y1);
    w = (x2 - x1);
    div = eval("baseline"+nr);
    div._x = x1;
    div._y = y1;

    div._width = Math.abs(w);
    div._height = Math.abs(h);
    if (w < 0) div._xscale = -div._xscale;
    if (h < 0) div._yscale = -div._yscale;

}


function drawTri(nr) {
    var t,xt,yt,n,xd,yd,a;
    var    trib = eval("_root.tri1"+nr);
    var    tria = eval("_root.tri0"+nr);
    var x1 = _root.pointProj[_root.tri[nr][0]].x;
    var y1 = _root.pointProj[_root.tri[nr][0]].y;
    var x2 = _root.pointProj[_root.tri[nr][1]].x;
    var y2 = _root.pointProj[_root.tri[nr][1]].y;
    var x3 = _root.pointProj[_root.tri[nr][2]].x;
    var y3 = _root.pointProj[_root.tri[nr][2]].y;
    var xa = _root.pointTrans[_root.tri[nr][2]].x
                - _root.pointTrans[_root.tri[nr][0]].x;
```

```
var ya = _root.pointTrans[_root.tri[nr][2]].y
             - _root.pointTrans[_root.tri[nr][0]].y;
var za = _root.pointTrans[_root.tri[nr][2]].z
             - _root.pointTrans[_root.tri[nr][0]].z;
var xb = _root.pointTrans[_root.tri[nr][1]].x
             - _root.pointTrans[_root.tri[nr][0]].x;
var yb = _root.pointTrans[_root.tri[nr][1]].y
             - _root.pointTrans[_root.tri[nr][0]].y;
var zb = _root.pointTrans[_root.tri[nr][1]].z
             - _root.pointTrans[_root.tri[nr][0]].z;
var xc = ya*zb - za*yb;
var yc = za*xb - xa*zb;
var zc = xa*yb - ya*xb;

vis = (y2-y1)*x3 + (x1-x2)*y3 - (y2-y1)*x1
                                        - (x1-x2)*y1;
if (vis < 0) {
    tria._visible = false;
    trib._visible = false;
    return;
} else {
    tria._visible = true;
    trib._visible = true;
}

c = ((xc+yc-zc)/(Math.sqrt(xc*xc+yc*yc+zc*zc)));

xa=x1;ya=y1;xb=x2;yb=y2;xc=x3;yc=y3;
t = xc*xc + yc*yc - xb*xc - yb*yc - xa*xc - ya*yc
                                    + xa*xb + ya*yb;
if (t > 0) {
    xa=x1;ya=y1;xb=x3;yb=y3;xc=x2;yc=y2;
    t = xc*xc + yc*yc - xb*xc - yb*yc - xa*xc - ya*yc
                                    + xa*xb + ya*yb;
    if (t > 0) {
        xa=x3;ya=y3;xb=x2;yb=y2;xc=x1;yc=y1;
        t = xc*xc + yc*yc - xb*xc - yb*yc - xa*xc
                            - ya*yc + xa*xb + ya*yb;
    }
}
xt = yb - ya;
yt = xa - xb;
n = 1.0 / Math.sqrt(xt*xt + yt*yt);
xt = n*xt;
yt = n*yt;
t = xt*xc + yt*yc - xt*xa - yt*ya;
xd = xc - t*xt;
yd = yc - t*yt;
```

```
    a = Math.atan((ya-yd)/(xa-xd))*180/Math.PI;
    if (xa - xd < 0) a = a - 180;
    if (t > 0) {
        tria = eval("_root.tri1"+nr);
        trib = eval("_root.tri0"+nr);
    }
    cola = new Color(tria);
    colb = new Color(trib);
    colTrans = new Object();
    colTrans.ra = colTrans.ga = colTrans.ba = c*100 + 80;
    colTrans.rb = colTrans.gb = 0;
    colTrans.bb = 255;
    colTrans.bb = 255;
    cola.setTransform(colTrans);
    colb.setTransform(colTrans);

    tria._rotation = 0;
    tria._height = Math.sqrt((xd-xc)*(xd-xc)+(yd-yc)
                                            *(yd-yc));
    tria._width = Math.sqrt((xd-xa)*(xd-xa)+(yd-ya)
                                            *(yd-ya));
    tria._rotation = a;
    tria._x = xd;
    tria._y = yd;

    a = Math.atan((yb-yd)/(xb-xd))*180/Math.PI;
    if (xb - xd < 0) a = a - 180;
    trib._rotation = 0;
    trib._x = xd;
    trib._y = yd;
    trib._height = Math.sqrt((xd-xc)*(xd-xc)+(yd-yc)
                                            *(yd-yc));
    trib._width = Math.sqrt((xd-xb)*(xd-xb)+(yd-yb)
                                            *(yd-yb));
    trib._rotation = a;
}
```

Die Verwendung der neuen Zeichenfunktionen

Genau erklärt wurden diese Funktionen im vorhergehenden Beispiel. Mit Flash MX sehen diese beiden Funktionen (allerdings ohne korrekte Schattierung der Seitenflächen) folgendermaßen aus:

```
function drawline(nr) {
    _root.render.lineStyle (0, lineColor, lineAlpha);
    _root.render.moveTo (
                _root.pointProj[_root.line[nr][0]].x,
                _root.pointProj[_root.line[nr][0]].y);
    _root.render.lineTo (
                _root.pointProj[_root.line[nr][1]].x,
                _root.pointProj[_root.line[nr][1]].y);
}
```

```
function drawTri(nr) {
    _root.render.beginFill (faceColor, faceAlpha);
    _root.render.lineStyle (0, lineColor, lineAlpha);
    _root.render.moveTo (
                    _root.pointProj[_root.tri[nr][0]].x,
                    _root.pointProj[_root.tri[nr][0]].y);
    _root.render.lineTo (
                    _root.pointProj[_root.tri[nr][1]].x,
                    _root.pointProj[_root.tri[nr][1]].y);
    _root.render.lineTo (
                    _root.pointProj[_root.tri[nr][2]].x,
                    _root.pointProj[_root.tri[nr][2]].y);
    _root.render.lineTo (
                    _root.pointProj[_root.tri[nr][0]].x,
                    _root.pointProj[_root.tri[nr][0]].y);
    _root.render.endFill();

}
```

Das sieht schon etwas einfacher aus. Aber wir haben hier ja auch, wie schon erwähnt, die lichtabhängige Schattierung weggelassen. Das werden wir gleich nachholen. Die obigen Funktionen sind eigentlich selbsterklärend genug.

Die Funktion zum Zeichnen eines Dreiecks inklusive Schattierung sieht wie folgt aus:

```
function drawTri(nr) {
    var x1 = _root.pointProj[_root.tri[nr][0]].x;
    var y1 = _root.pointProj[_root.tri[nr][0]].y;
    var x2 = _root.pointProj[_root.tri[nr][1]].x;
    var y2 = _root.pointProj[_root.tri[nr][1]].y;
    var x3 = _root.pointProj[_root.tri[nr][2]].x;
    var y3 = _root.pointProj[_root.tri[nr][2]].y;

    vis = (y2-y1)*x3 + (x1-x2)*y3 - (y2-y1)*x1
                                        - (x1-x2)*y1;
    if (vis < 0) return;
```

Hier wurden zunächst wieder einfach die bereits projizierten Punkte in einfache Variablen gespeichert, um sie leichter verwenden zu können. Anschließend wird getestet, ob die auf den Bildschirm projizierten Eckpunkte eines Dreiecks im Uhrzeigersinn vorliegen oder nicht. Dementsprechend wird die Funktion verlassen, ohne das Dreieck zu zeichnen, falls dem Betrachter nur die Rückseite zugewandt ist.

```
    var xa = _root.pointTrans[_root.tri[nr][2]].x
                - _root.pointTrans[_root.tri[nr][0]].x;
    var ya = _root.pointTrans[_root.tri[nr][2]].y
                - _root.pointTrans[_root.tri[nr][0]].y;
    var za = _root.pointTrans[_root.tri[nr][2]].z
                - _root.pointTrans[_root.tri[nr][0]].z;
```

```
        var xb = _root.pointTrans[_root.tri[nr][1]].x
                    - _root.pointTrans[_root.tri[nr][0]].x;
        var yb = _root.pointTrans[_root.tri[nr][1]].y
                    - _root.pointTrans[_root.tri[nr][0]].y;
        var zb = _root.pointTrans[_root.tri[nr][1]].z
                    - _root.pointTrans[_root.tri[nr][0]].z;
        var xc = ya*zb - za*yb;
        var yc = za*xb - xa*zb;
        var zc = xa*yb - ya*xb;
```

Diese Zeilen haben den Normalenvektor der transformierten Dreiecksfläche berechnet und in `(xc,yc,zc)` gespeichert.

```
        c = ((xc+yc+zc)/(Math.sqrt(xc*xc+yc*yc+zc*zc)
                        * Math.sqrt(3)));
        if (c < 0) c = 0;
        _root.render.beginFill ((255*c) << 16
                                | (255*c) << 8 | (255), 100);
```

In der Variablen c wird der Kosinus des Winkels zwischen dem Normalenvektor und dem Vektor `(1,1,1)` gespeichert. Damit wird eine Lichtquelle, die sich links oben neben dem Betrachter befindet, simuliert. Mit dem Wert c wird dann die Helligkeit des Blautons der Seitenflächen bestimmt.

```
        _root.render.lineStyle (0, 0, 0);
        _root.render.moveTo (
                        _root.pointProj[_root.tri[nr][0]].x,
                        _root.pointProj[_root.tri[nr][0]].y);
        _root.render.lineTo (
                        _root.pointProj[_root.tri[nr][1]].x,
                        _root.pointProj[_root.tri[nr][1]].y);
        _root.render.lineTo (
                        _root.pointProj[_root.tri[nr][2]].x,
                        _root.pointProj[_root.tri[nr][2]].y);
        _root.render.lineTo (
                        _root.pointProj[_root.tri[nr][0]].x,
                        _root.pointProj[_root.tri[nr][0]].y);
        _root.render.endFill();

    }
```

Das eigentliche Zeichnen des Dreiecks ist dann nicht mehr kompliziert. Der Rest des notwendigen ActionScripts unterscheidet sich nur wenig vom letzten Beispiel.

```
    var point = new Array(8);
    var pointTrans = new Array(8);
    var pointProj = new Array(8);
    var pointCount = 0;
    var tri = new Array(12);
    var triCount = 0;
```

```
var camera = new Vector(0.0,0.0,-500.0);
var rx=0,ry=0,rz=0;

_root.createEmptyMovieClip("render",1);
_root.render._x = Stage.width/2;
_root.render._y = Stage.height/2;
```

Nach den Variablendeklarationen wird hier der MovieClip erzeugt, auf dem später gezeichnet wird. Außerdem wird der MovieClip in den Bühnenmittelpunkt verschoben.

```
addPoint(-100,100,-100);
addPoint(100,100,-100);
addPoint(100,100,100);
addPoint(-100,100,100);
addPoint(-100,-100,-100);
addPoint(100,-100,-100);
addPoint(100,-100,100);
addPoint(-100,-100,100);

addTri(0,3,1);
addTri(1,3,2);
addTri(0,1,4);
addTri(1,5,4);
addTri(5,1,2);
addTri(2,6,5);
addTri(2,3,7);
addTri(2,7,6);
addTri(7,3,0);
addTri(7,0,4);
addTri(7,4,5);
addTri(7,5,6);


function Vector(x,y,z) {this.x = x; this.y = y;
                        this.z = z; return this;}

function addPoint(x,y,z) {
    _root.point[_root.pointCount] = new Vector(x,y,z);
    _root.pointTrans[_root.pointCount] =
                                    new Vector(x,y,z);
    _root.pointProj[_root.pointCount] =
                                    new Vector(x,y,z);
    _root.pointCount++;
}
```

Diese Funktionen sind genau gleich geblieben. Lediglich die Funktionen zum Speichern der Linien wurden weggelassen.

```
function addLine(p1,p2) {
    _root.line[_root.lineCount] = new Array(2);
    _root.line[_root.lineCount][0] = p1;
    _root.line[_root.lineCount][1] = p2;
    _root.lineCount++;
}

function addTri(p1,p2,p3) {
    _root.tri[_root.triCount] = new Array(3);
    _root.tri[_root.triCount][0] = p1;
    _root.tri[_root.triCount][1] = p2;
    _root.tri[_root.triCount][2] = p3;
    _root.triCount++;
}
```

Bei diesen beiden Funktionen wurden die Befehle zum Einfügen der MovieClips entfernt. Die MovieClips waren notwendig, da keine Grafikelemente neu erzeugt werden konnten.

```
function transformiere(rx,ry,rz,tx,ty,tz) {
      var i,x,y,z;
      rx = Math.PI/180.0 * rx;
      ry = Math.PI/180.0 * ry;
      rz = Math.PI/180.0 * rz;
     for (i = 0;i < _root.pointCount; i++) {
            x = _root.point[i].x;
          y = _root.point[i].y;
          z = _root.point[i].z;
          // rotation um die x-Achse
          _root.pointTrans[i].y = y * Math.cos(rx)
                                    - z * Math.sin(rx);
          _root.pointTrans[i].z = y * Math.sin(rx)
                                    + z * Math.cos(rx);
          // rotation um die y-Achse
          y = _root.pointTrans[i].y;
              z = _root.pointTrans[i].z;
          _root.pointTrans[i].x = x * Math.cos(ry)
                                    + z * Math.sin(ry);
          _root.pointTrans[i].z = (-1) * x * Math.sin(ry)
                                    + z * Math.cos(ry);
        _root.pointTrans[i].x += tx;
        _root.pointTrans[i].y += ty;
        _root.pointTrans[i].z += tz;
       }
}

function projeziere() {
    var i;

    for (i=0; i < _root.pointCount; i++) {
           _root.pointProj[i].x =
```

```
                    (- _root.pointTrans[i].x * camera.z)
                  / (_root.pointTrans[i].z - camera.z);
            _root.pointProj[i].y =
                    (- _root.pointTrans[i].y * camera.z)
                  / (_root.pointTrans[i].z - camera.z);
            _root.pointProj[i].z = 0;
        }

    }
```

Die Funktionen zur Transformation und Projektion sind wieder gleich geblieben.

Die Bewegung des Objektes

In Frame 2 werden wieder die Befehle zum Drehen und Anzeigen platziert.

```
var t;

rx+= 5;ry += 3;rz += 7;

_root.transformiere(rx,ry,rz,0,0,0);
_root.projeziere();

_root.render.clear();

for (t=0; t<_root.triCount; t++) {
    _root.drawTri(t);
}
```

Was hier dazukommt, ist der Befehl zum Löschen der Grafikelemente, bevor das Modell neu gezeichnet wird.

In Frame 3 bleibt der Befehl zum Sprung zurück zu Frame 2.

```
gotoAndPlay(_currentFrame - 1);
```

Das Ergebnis des Beispiels ist in Abbildung 4.30 zu sehen.

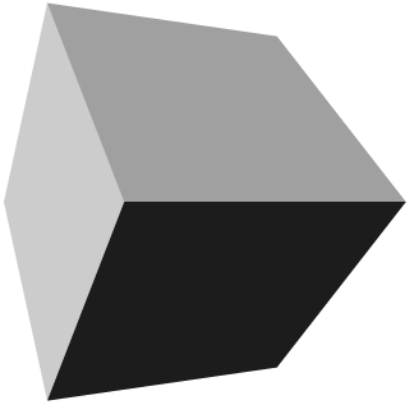


Abbildung 4.30: Der Würfel mit Flash MX dargestellt

Bisher wird bei diesem Beispiel nur ein einzelner konvexer Körper korrekt dargestellt. Um mehrere Körper oder auch konvexe Körper korrekt darzustellen, müssen die Dreiecke vor dem Zeichnen erst der Tiefe nach sortiert werden. Nur dadurch kann sichergestellt werden, dass weiter vorne liegende Dreiecke nicht durch weiter hinten liegende verdeckt werden.

Das Beispiel mit Tiefensortierung

Um dies zu erreichen, verpacken wir die Zeichenschleife aus Frame 2 in eine extra Funktion, die folgendermaßen aussieht:

```
function drawAll() {
    var t;
    var triSort = new Array(triCount);

    for (t=0; t<triCount; t++) triSort[t] = t;
    triSort.sort(sortOrder);
    _root.projeziere();
    _root.render.clear();
    for (t=0; t<_root.triCount; t++) {
        _root.drawTri(triSort[t]);
    }
}
```

Diese Funktion hat die Aufgabe, alle Dreiecke der Tiefe nach zu sortieren und diese dann zu zeichnen, beginnend mit dem am weitesten entfernten. Dazu wird zunächst ein Feld angelegt, welches vorbelegt wird mit den Nummern der Dreiecke. Dieses Feld wird dann sortiert, allerdings nicht mit einem normalen „größer“-Vergleich, sondern mit einer speziellen Vergleichsfunktion:

```
function sortOrder(a,b) {
    if (_root.pointTrans[_root.tri[a][0]].z
        < _root.pointTrans[_root.tri[b][0]].z) return 1;
    else return 0;

}
```

Diese Sortierfunktion nimmt die Indizes zweier Dreiecke und vergleicht jeweils den transformierten z-Wert des ersten Punktes. Das ist zwar nur eine grobe Annäherung, die aber für die meisten Zwecke genügt.

Nachdem nun die Dreiecke sortiert sind, werden die Punkte projiziert, die Zeichenfläche gelöscht und die Dreiecke gezeichnet.

Auf der CD finden Sie unter dem Namen *mx_cubes.fla* einen Film, der damit vier Würfel rotiert, wie in Abbildung 4.31 zu sehen ist.

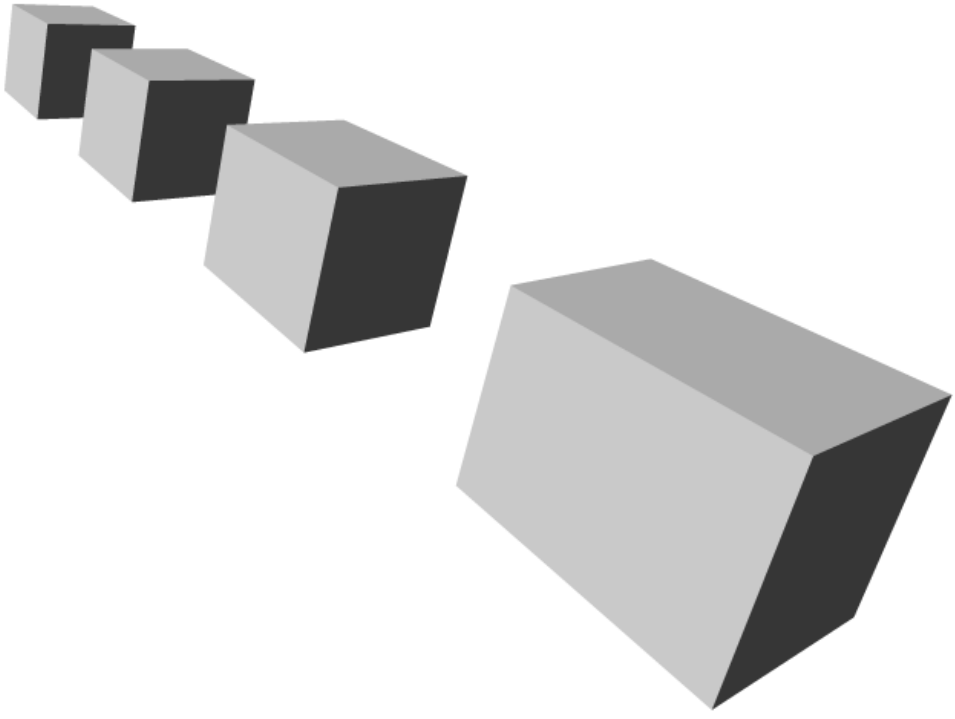


Abbildung 4.31: Vier Würfel mit Tiefensortierung korrekt dargestellt

Geschwindigkeitsoptimierungen

Bei vielen Dreiecken und zusätzlicher Sortierung gehen manche Rechner schon langsam in die Knie. Es lassen sich aber noch einige Geschwindigkeitsoptimierungen vornehmen. Diese umfassen unter anderem:

- Vorberechnung von Sinus- und Kosinus-Werten. Diese könnten dann in einem Feld gespeichert werden und müssten nur noch per Winkelindex nachgesehen werden.
- Zeichnen von Rechtecken anstatt Dreiecken: Wenn die Körper es erlauben, kann es sinnvoller sein, jeweils nur ein Rechteck anstatt zweier Dreiecke zu zeichnen. Dadurch kann die Anzahl einiger Berechnungen halbiert werden.
- Heruntersetzen der Abspielqualität: Beim Flashplayer kann die Wiedergabequalität auf Medium oder Low gesetzt werden, damit das Antialiasing abgeschaltet wird.

4.9 Zusammenfassung

Der besondere Reiz bei Flash liegt in der weiten Verbreitung dieses Plugins und in der sehr guten Darstellungsqualität. Mit Flash können auch sehr kleine Dateien erzeugt werden, die wenig Übertragungszeit benötigen und dennoch sehr attraktive Effekte erzeugen.

Doch trotz alldem ist Flash nicht wirklich das ideale Werkzeug zur Darstellung von 3D-Grafik und Animation. Dazu benötigt man zurzeit noch spezialisierte Plugins. Eine Auswahl dieser Plugins wird nun in den weiteren Kapiteln besprochen.

Director und Shockwave

Director ist ein Programm der Firma Macromedia. Mit Hilfe dieses Programms lassen sich Filme erstellen, die man entweder auf CD oder im Internet verbreiten kann. Für Webseiten gibt es ein Plugin, welches die Filme im Shockwave-Format abspielen kann. Dieses Plugin besitzt im Web eine große Verbreitung, weshalb wir uns dessen Möglichkeiten an dieser Stelle genauer ansehen möchten.

Das Kapitel beschreibt die Versionen 8 und 8.5

Zurzeit liegt Director mit dem zugehörigen Plugin in der Version 8.5 vor. Beim Sprung von der Version 8 zu 8.5 kam eine wesentliche Neuerung hinzu: Die direkte Unterstützung von 3D-Grafik. Allerdings besitzen die vorhergehenden Versionen immer noch eine große Verbreitung, weshalb ich dieses Kapitel unterteilt habe in einen Abschnitt, der sich mit den Möglichkeiten von Director bis einschließlich Version 8 beschäftigt, und dem (größeren) Rest des Kapitels, welcher sich mit den überragenden Möglichkeiten ab der Version 8.5 befasst.

5.1 Grundlagen

Der Editor, um Shockwave-Dateien zu erzeugen, heißt genau genommen „Director Shockwave Studio“. Die Dateien, die man damit erzeugen kann, sind im .dcr-Format und können vom Director Shockwave Plugin wiedergegeben werden. Es gibt keine anderen Programme, die das gleiche Format erzeugen können.

Director ist in weiten Teilen vergleichbar mit Flash. Das Endergebnis einer Produktion ist ein so genannter Film, der frame-basiert abgespielt wird, in dem Akteure vorkommen, die sich zeitgesteuert bewegen und verändern können und das Ganze kann mit einer Skriptsprache programmiert werden. Allerdings ist in einigen Punkten Director Flash doch überlegen. So kann man in Director z.B. beliebige Grafikobjekte direkt per Skript erzeugen und hat mehr Freiheiten, bestehende Grafikobjekte zu verändern.

Director wird häufig auch zur Produktion von CD-basierten Anwendungen oder Demonstrationen benutzt.

Zur Programmierung von Director stellt das Programm die Programmiersprache Lingo zur Verfügung. Diese Sprache war ursprünglich natürlich-sprachlich ausgerichtet und dadurch für „normale“ Programmierer

umständlich zu benutzen. Allerdings sollte dadurch Anfängern der Einstieg erleichtert werden. Es ist inzwischen auch möglich, in Lingo objektorientiert zu programmieren.

Auch bei diesem Kapitel sei wieder darauf hingewiesen, dass es nicht als Ersatz für einen Kurs über Director dienen kann oder soll. Um den Rahmen des Buches nicht zu sprengen, muss ich an dieser Stelle voraussetzen, dass Sie die grundlegenden Prinzipien von Director kennen und sich auch schon mit der Programmiersprache Lingo beschäftigt haben.

5.2 Director bis Version 8

Mit Director können beliebige Grafiken dynamisch erzeugt werden.

Im vorhergehenden Kapitel hatten wir z.B. das Problem, Linien und Dreiecke zu zeichnen. Das ist mit Director kein Problem und wir können hier sogar Texturen verwenden, die uns in Flash noch verwehrt waren. Mit einem Beispiel für Texturen wollen wir auch gleich einsteigen.

Quads

Quads sind eine Funktionalität von Director, die es ermöglicht, Sprites beliebig zu verzerren. Das heißt, man kann jeder der vier Ecken eines Sprites eine beliebige neue Position geben. Wir können diese Funktion also nutzen, um die perspektivisch verzerrte Seitenfläche eines Würfels darzustellen.

In dem nächsten Beispiel benutzen wir das Bild aus Abbildung 5.1, um damit die Darstellung in Abbildung 5.2 zu erzeugen. Das fertige Beispiel ist auf der CD im Director-Verzeichnis unter *Quad.dir* zu finden.

Abbildung 5.1: Das Bild für die Textur

Mit Quads können perspektivisch verzerrte Bilder erstellt werden.

Abbildung 5.2: Perspektivische Darstellung des Würfels

Für den Film benötigen wir nur diese Textur und zwei Skripte. Das Bild platzieren wir in den ersten sechs Sprites des Films (siehe auch Abbildung 5.3). Diese sechs Sprites werden dann die sechs Seiten des Würfels bilden. Sie können für die sechs Seiten natürlich auch jedes beliebige andere Bild benutzen.

Das erste Skript ist ein Filmskript, in dem wir die notwendigen Initialisierungen vornehmen und allgemeine Funktionen definieren.

```
global gCamera, gPoint, gPointTrans, gPointProj, gQuad
global rx,ry,rz
```

Die Variable `gCamera` wird die Position der virtuellen Kamera speichern. `gPoint`, `gPointTrans` und `gPointProj` speichern die Ausgangspunkte, die transformierten Punkte und die projizierten Punkte. In `rx`, `ry` und `rz` werden die Rotationswinkel für die drei Achsen gespeichert.

```
on startmovie
  gCamera = [0 ,0,-500]
  gPoint = [[ 100 ,  100 , 100],[-100 ,  100 , 100],
            [-100 , -100 , 100],[ 100 , -100 , 100],
            [ 100 ,  100 , -100],[-100 ,  100 , -100],
            [-100 , -100 , -100],[ 100 , -100 , -100]]
  gQuad = [[2,1,4,3],[1,5,8,4],[5,6,7,8],
           [6,2,3,7],[2,6,5,1],[3,4,8,7]]

  gPointTrans = gPoint.duplicate()
  gPointProj = gPoint.duplicate()

  rx = 0
  ry = 0
  rz = 0
end
```

Bei Start des Films wird die virtuelle Kameraposition gesetzt und der Würfel initialisiert. In `gPoint` werden die acht Eckpunkte relativ zum Koordinatenursprung gespeichert, wobei ein Punkt jeweils durch eine Liste aus drei Elemente (x-, y- und z-Koordinate) repräsentiert wird. In `gQuad` werden die Seitenflächen definiert, wobei eine Seite jeweils durch eine Liste dargestellt wird, die die Indizes der Eckpunkte enthält. Diese Indizes müssen jeweils im Uhrzeigersinn vorliegen, damit die Seitenflächen später auf Sichtbarkeit überprüft werden können.

```
on projiziere
  repeat with i = 1 to count (gPoint)
    x = gPointTrans[i][1]
    y = gPointTrans[i][2]
    z = gPointTrans[i][3]
    px = (gCamera[1] * z - (x) * gCamera[3])
           / (z - gCamera[3]) +300
    py = (gCamera[2] * z - (y) * gCamera[3])
           / (z - gCamera[3]) +200

    gPointProj[i] = [px,py]
  end repeat
end
```

Hier werden alle transformierten Punkte entsprechend der Kameraposition in die Bildebene projiziert. Dabei wird von einer Filmgröße von 600 mal 400 Punkten ausgegangen, wodurch sich ein Offset von 300 bzw. 200 Punkten ergibt.

```
on display
  repeat with i = 1 to 6
    x1 = gPointProj[gQuad[i][1]][1]
    y1 = gPointProj[gQuad[i][1]][2]
    x2 = gPointProj[gQuad[i][2]][1]
    y2 = gPointProj[gQuad[i][2]][2]
    x3 = gPointProj[gQuad[i][3]][1]
    y3 = gPointProj[gQuad[i][3]][2]
    x4 = gPointProj[gQuad[i][4]][1]
    y4 = gPointProj[gQuad[i][4]][2]
    vis = (y2-y1)*x3 + (x1-x2)*y3 - (y2-y1)*x1
          - (x1-x2)*y1
    if (vis < 0) then
      Sprite(i).visible = true
    else
      Sprite(i).visible = false
    end if
    Sprite(i).quad = [[x1,y1],[x2,y2],[x3,y3],[x4,y4]]
  end repeat
end
```

Die Rückflächenunterdrückung

Für die Anzeige des Würfels werden in einer Schleife alle Flächen durchlaufen und zunächst die Bildschirmkoordinaten der Eckpunkte in lokalen Variablen zwischengespeichert. In der Variable `vis` wird berechnet, ob die Eckpunkte (zumindest die ersten drei) im Uhrzeigersinn vorliegen. Wenn nicht, bedeutet das, dass nur die Rückseite der Fläche zu sehen ist, was wiederum bedeutet, dass die Fläche nicht gezeigt werden muss.

Schließlich werden die Ecken des Sprites an die projizierten Eckpunkte gesetzt.

```
on rotiere rx,ry,rz
  repeat with i = 1 to count (gPoint)
    xr = gPoint[i][1]
    yr = gPoint[i][2]
    zr = gPoint[i][3]

    -- Rotation um die z-Achse
    xt = xr * cos(rz) - yr * sin(rz)
    yt = xr * sin(rz) + yr * cos(rz)
    xr = xt
    yr = yt

    -- Rotation um die x-Achse
    yt = yr * cos(rx) - zr * sin(rx)
    zt = yr * sin(rx) + zr * cos(rx)
    yr = yt
    zr = zt

    -- Rotation um die y-Achse
    xt = xr * cos(ry) + zr * sin(ry)
    zt = -xr * sin(ry) + zr * cos(ry)
    xr = xt
    zr = zt

    gPointTrans[i] = [xr,yr,zr]
  end repeat
end
```

In dieser Funktion werden die Ursprungspunkte um die x-,y- und z-Achse gedreht. Das Ergebnis einer Teilberechnung wird jeweils in einer temporären Variable gespeichert, um mit den Originalen noch den Rest berechnen zu können.

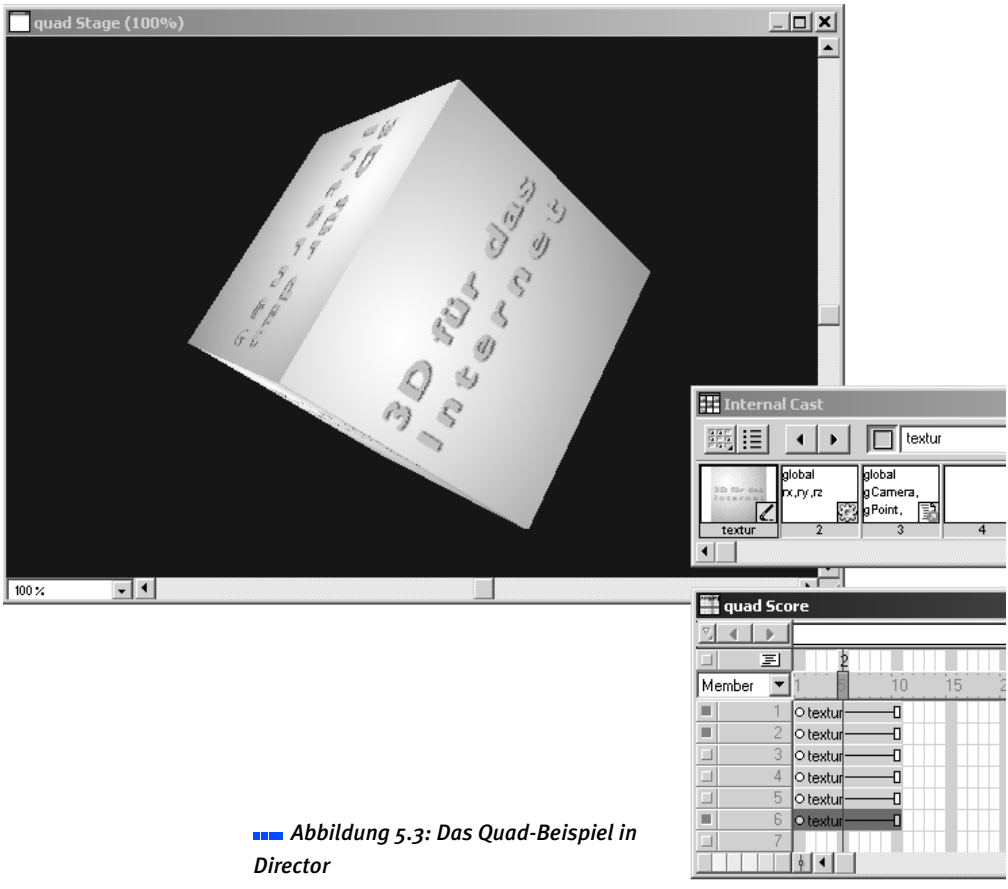


Abbildung 5.3: Das Quad-Beispiel in Director

In Frame 5 wird nun noch ein Skript platziert, welches die Rotationswerte hochzählt und wieder zum gleichen Frame springt, damit wir eine Schleife haben.

```
global rx,ry,rz

on exitFrame
  rx = rx + 0.03
  ry = ry + 0.06
  rz = rz + 0.09
  rotiere(rx,ry,rz)
  projiziere
  display
  go the frame
end
```

Sie sehen, dass alleine durch die Möglichkeit, Texturen darzustellen, die Qualität der 3D-Grafik erheblich zunimmt. Allerdings ist der Aufwand, mit der hier gezeigten Methode anspruchsvolle Animationen zu erzeu-

gen, recht hoch. Es fehlt auch noch eine Tiefensortierung der Flächen, um nicht-konvexe Körper korrekt darzustellen.

Um sich das alles etwas zu vereinfachen gibt es so genannte 3D-Engines.

3D-Engines

3D-Engines sind eine Sammlung von Funktionen und Hilfsroutinen, die einem die Arbeit mit der 3D-Grafik erleichtern oder neue Möglichkeiten eröffnen.

Die 3D-Engine ist nur bis Director Version 8 sinnvoll.

Was ich hier beispielhaft vorstellen möchte, ist eine 3D-Engine von Barry Swan *(http://www.theburrow.co.uk/t3dtesters)*. Barry Swan ist eine bekannte Größe in der Director-3D-Szene. Er stellt die 3D-Engine kostenlos zur Verfügung, solange man die Funktion `t3dDetails()` unverändert lässt.

Eine weitere 3D-Engine, die eine recht große Bekanntheit besitzt, ist die von Dave Cole *(http://www.dubbus.com/devnull/3D)*. Sie ist allerdings nicht ganz so performant und leistungsstark wie die von Barry Swan.

Die 3D-Engine von Barry Swan bietet sehr viele Optionen und nimmt einem sehr viel Arbeit ab. Allerdings erfordert es auch einiges an Einarbeitung, bis man damit gute Ergebnisse bekommt. Ich werde in diesem Abschnitt nicht alle Funktionen und Details erläutern, sondern nur einen groben Überblick liefern und einige Beispiele zeigen.

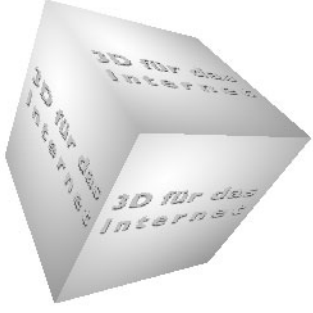


Abbildung 5.4: Ein Beispiel für T3D

Die Engine besteht aus einer Sammlung von Darstellern, die alle benötigten Skripte enthalten (siehe Abbildung 5.5). In der Abbildung sind nur die letzten vier Einträge selbst erstellt für das Beispiel in Abbildung 5.4. Wir werden zunächst das Gleiche entwikkeln wie auch schon im Abschnitt zuvor. Im Abschnitt über Quads haben wir allerdings alles selbst programmieren müssen und ich hatte erläutert, was sogar noch alles fehlt, um komplexere Modelle und Animationen zu erstellen.

Für die Erstellung des gleichen Beispiels mit Hilfe der Engine ist Folgendes notwendig:

1. Legen Sie einen neuen Film an.
2. Öffnen Sie die Cast-Library T3D.cst und kopieren Sie alle Member daraus in die interne Cast-Library des neuen Films.
3. Importieren Sie ein Bild als Textur, z.B. das Bild *textur.tif,* und geben Sie ihm den Namen `Textur`.
4. Legen Sie ein Movie-Skript an, welches die notwendigen Initialisierungen vornimmt (das Skript ist weiter unten erläutert).
5. Erzeugen Sie einen Dummy-Darsteller, indem Sie das Paint-Fenster öffnen, einen Namen eintragen (z.B. `Dummy`) und ↵ drücken.
6. Platzieren Sie sechs Kopien (für jede Würfelseite eine) von `Dummy` irgendwo auf der Bühne.
7. Erzeugen Sie in Frame 5 ein Skript, welches den Würfel rotiert und zum gleichen Frame zurückspringt, um eine Schleife zu bilden.

Die Darsteller der 3D-Engine. Nur die letzten vier Einträge sind selbst erstellt.

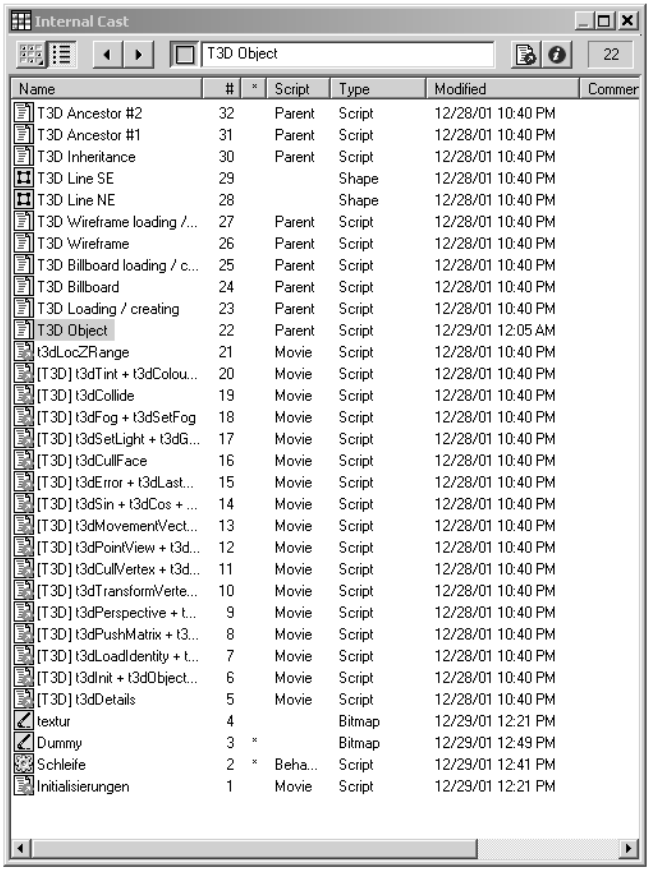


Abbildung 5.5: Die Darsteller der 3D-Engine

Das Movie-Skript für die Initialisierungen sieht folgendermaßen aus:

```
global gObject

on prepareMovie

  t3dInit( #T3D_DEGREES )
  t3dObjectInit( #T3D_CULL_FACE )

  gObject = script("T3D Object").new( #T3D_CUBE, 1,
                                      2.0, "Textur")

end
```

Die globale Variable `gObject` wird benötigt, um darin das Grafikobjekt zu speichern. Die Variable muss global sein, da auf das Objekt später in der Schleife noch zugegriffen werden muss.

Der Befehl `t3dInit()` nimmt notwendige Initialisierungen der Engine vor und bestimmt durch den übergebenen Parameter, dass alle Winkelangaben in Grad gemacht werden und nicht im Bogenmaß, was der Standard ist.

Der Befehl `t3dObjectInit( #T3D_CULL_FACE )` sorgt dafür, dass nur die Flächen gezeichnet werden, deren Vorderseite dem Betrachter zugewandt ist. Das Gleiche hatten wir auch in dem vorhergehenden Beispiel gemacht, allerdings mussten wir das da selbst programmieren.

Die letzte Zeile schließlich erzeugt das Objekt. Die Engine bietet die Möglichkeit, verschiedene vordefinierte Objekte zu erzeugen, eines davon ist ein Standardwürfel. Der Parameter 1 bewirkt, dass die Sprites, die für die Seitenflächen des Würfels benötigt werden (sechs Stück, für jede Seite eins), ab Spritekanal 1 platziert werden. Die 2.0 gibt die Größe des Würfels an und der String „Textur“ sagt der Engine, den Darsteller mit diesem Namen als Textur zu verwenden.

Das Skript für die Schleife sieht folgendermaßen aus:

```
global gObject

on exitFrame
  gObject.pYAngle = gObject.pYAngle + 3
  gObject.pXAngle = gObject.pXAngle + 5

  t3dLoadIdentity()
  t3dPointView( 0.0, 0.0, 4.0 )
  gObject.t3dDraw()

  updateStage

  go the frame

end
```

Die beiden ersten Zeilen der Funktion `exitFrame` verändern den Rotationswinkel des Würfels um 3 bzw. 5 Grad.

Der Befehl `t3dLoadIdentity()` setzt die aktive Transformationsmatrix zurück. Die 3D-Engine ist in ihrer Funktionsweise angelehnt an OpenGL. Das bedeutet, dass sie auch mit Transformationsmatrizen arbeitet (siehe auch Grundlagenkapitel). Durch das Laden der Einheitsmatrix werden also alle Transformationen gelöscht. Anschließend wird mit `t3dPointView()` die Kameraposition festgelegt und das Objekt gezeichnet. Beim Zeichnen des Objektes werden die zuvor gesetzten Winkel berücksichtigt.

Schließlich wird die Bühne noch aktualisiert, damit die Änderungen auch sichtbar werden, und zum gleichen Frame gesprungen.

Das waren auch schon alle Anweisungen, um das gewünschte Ergebnis zu erzeugen. Sie sehen also, dass die Arbeit sich erheblich vereinfacht hat. Bevor wir nun ein weiteres Beispiel besprechen, möchte ich kurz die wichtigsten Funktionen auflisten, die die 3D-Engine zu bieten hat:

Die wichtigsten Funktionen der T3D-Engine

- Einfache Kollisionserkennung (dabei wird allerdings nur eine Kugelhülle um das gesamte Objekt gelegt, sonst wäre die Prüfung zu rechenaufwändig)
- Mehrere Lichtquellen
- Standardtransformationen (Verschieben, Rotieren, Skalieren)
- Tiefennebel (weiter zurückliegende Objekte verschwinden in einer Art Nebel)
- Standardkörper Würfel, Kugel, Kegel, Zylinder, Polygon und Höhenfeld
- Einfache Animationen
- Mausinteraktion mit Modellen

Mit all diesen Funktionen können Sie beeindruckende Animationen erstellen. Viele Beispiele dazu finden Sie auch auf der Website von Barry Swan.

Als ein weiteres Beispiel für die Programmierung der Engine wollen wir nun ein kleines Menüsystem entwickeln, wie es in Abbildung 5.6 zu sehen ist.

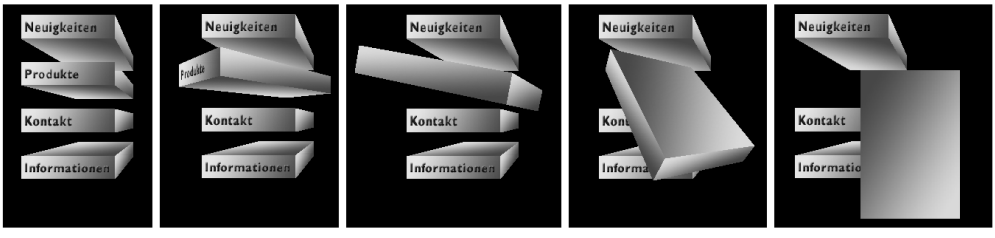


Abbildung 5.6: Ein Menüsystem

Sie können das Beispiel zunächst ausprobieren, indem Sie auf der CD im Verzeichnis *Director* die Datei *t3d_menue.htm* aufrufen. Sie können beliebig auf die Kästchen klicken, worauf diese sich öffnen und ein Untermenü zeigen (welches in dem Beispiel aber nicht realisiert ist).

Der Film ist ganz ähnlich aufgebaut wie der vorhergehende. Wir haben ein Initialisierungsskript, ein Skript für die Schleife, einen Dummy-Darsteller, eine Textur für die Seitenflächen und schließlich Texturen für die Beschriftungen (die Texturen sind in Abbildung 5.7 dargestellt).

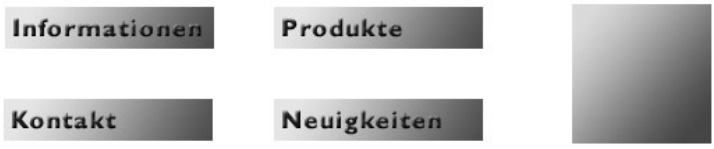


Abbildung 5.7: Die Texturen für das Menüsystem

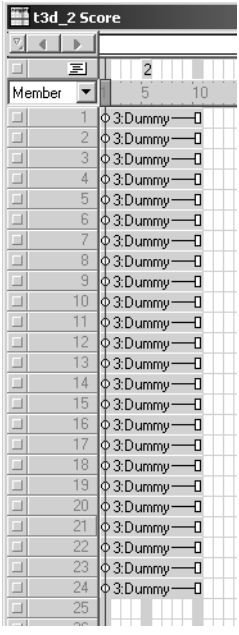


Abbildung 5.8: Sprites für die Seitenflächen

Da wir in diesem Beispiel vier Quader benutzen werden, müssen Sie entsprechend den `Dummy` in 24 Spritekanäle kopieren (siehe Abbildung 5.8).

Betrachten wir das Initialisierungsskript:

```
global gObjects, gCurrentObject, gSchritt, gImpuls
```

Da wir nun vier Objekte haben, werden wir diese in einer Liste namens `gObjects` speichern. `gCurrentObject` wird das Objekt enthalten, welches angeklickt wurde. Die beiden restlichen globalen Variablen werden bei der Schleife noch näher erläutert.

```
on prepareMovie

  t3dInit( #T3D_DEGREES )
  t3dObjectInit( #T3D_CULL_FACE )

  gObjects = []
  gObjects.add(script("T3D Object").new( #T3D_CUBE, 1,
                                        1.0, "textur2" ))
  gObjects.add(script("T3D Object").new( #T3D_CUBE, 7,
                                        1.0, "textur2" ))
  gObjects.add(script("T3D Object").new( #T3D_CUBE, 13,
                                        1.0, "textur2" ))
  gObjects.add(script("T3D Object").new( #T3D_CUBE, 19,
                                        1.0, "textur2" ))
  gObjects[1].plAllTextures[1][4] = "menue1"
  gObjects[2].plAllTextures[1][4] = "menue2"
  gObjects[3].plAllTextures[1][4] = "menue3"
  gObjects[4].plAllTextures[1][4] = "menue4"
```

Die vier Objekte werden erzeugt und erhalten zunächst die allgemeine Textur `textur2`. Anschließend wird jeweils der vierten Seitenfläche jedes Objektes die entsprechende Textur mit dem jeweiligen Menüpunkt zugewiesen.

```
  repeat with a = 1 to 4
    repeat with b = 1 to 6
      gObjects[a].t3dSetScript( b, #T3D_MOUSEUP, "mUp" )
    end repeat

    gObjects[a].t3dEnable( #T3D_SCRIPTS )
  end repeat
```

Diese Schleifen weisen den Seitenflächen jeweils ein Skript zu, welches aufgerufen wird, wenn die linke Maustaste auf der Seitenfläche (bzw. dem entsprechenden Sprite) losgelassen wird. Weiterhin wird für das jeweilige Objekt die Skript-Eigenschaft eingeschaltet.

```
end

on mUp tObject
  gCurrentObject = tObject
  gSchritt = 1
  gImpuls = 1
end
```

Die Funktion `mUp` wird aufgerufen, wenn ein Objekt angeklickt wurde. Hier wird dann in der globalen Variable `gCurrentObject` das angeklickte Objekt gespeichert. Außerdem werden die „Animationsvariablen“ initialisiert. Eine Animation ist unterteilt in mehrere Teilanimationen, die ich hier Impulse nennen werde. Eine Teilanimation ist zum Beispiel eine Rotation, die

nächste Teilanimation eine Verschiebung. Jede Teilanimation ist wieder unterteilt in eine Anzahl von Teilschritten. Im Folgenden ist jede Teilanimation unterteilt in 15 Schritte, danach folgt die nächste Teilanimation.

Das Skript für die Schleife in Frame 5 sieht folgendermaßen aus:

```
global gObjects, gCurrentObject, gSchritt, gImpuls

on exitFrame
  t3dLoadIdentity()
  t3dPointView( 0.0, 0.0, 6.0, 0.0, 0.0, 0.0 )
  t3dPushMatrix()
```

Diese Befehle sollten bekannt sein, bis auf `t3dPushMatrix()`. Dieser Befehl bewirkt, dass die aktuelle Transformationsmatrix zwischengespeichert wird.

```
  repeat with i=1 to 4
    t3dCopyMatrix()
```

Hier wird die oben zwischengespeicherte Transformationsmatrix wieder geladen.

```
    t3dTranslate(-2.0, i*1-2,0.0)
```

Alle Quader werden nach links verschoben und entsprechend ihrer Nummer in der Vertikalen verteilt.

```
    if (gCurrentObject = gObjects[i]) then
```

Hier wird überprüft, ob das gegenwärtig durchlaufene Objekt dasjenige ist, welches angeklickt wurde.

Anschließend wird geprüft, welche Teilanimation abgespielt werden soll.

```
      if (gImpuls = 1 ) then
        t3dTranslate(0.0, 0.0,0.1 * gSchritt)
        t3dRotateY(gSchritt*-6)
      end if
```

In der ersten Teilanimation wird der Quader um insgesamt 1,5 Einheiten nach rechts verschoben und um insgesamt 90° gedreht.

```
      if (gImpuls = 2 ) then
        t3dTranslate(0.0, 0.0,1.5)
        t3dTranslate(0.1 * gSchritt, -0.1 * gSchritt,0.0)
        t3dRotateY(-90)
        t3dRotateY(gSchritt*-6)
        t3dRotateX(gSchritt*-6)
      end if
```

In der zweiten Teilanimation werden zunächst die Endtransformationen der vorhergehenden Teilanimation durchgeführt und anschließend der Quader weiter verschoben und rotiert.

```
if (gImpuls > 2 ) then
  t3dTranslate(1.5, -1.5,1.5)
  t3dRotateY(-180)
  t3dRotateX(-90)
end if
```

Sobald die Animationen beendet sind, werden nur noch die Transformationen für die Endposition durchgeführt.

```
if (gSchritt = 15) then
  gSchritt = 1
  gImpuls = gImpuls + 1
end if
```

Wenn 15 Schritte durchlaufen wurden, wird der Zähler für die Teilanimationen hochgezählt und der Schrittzähler zurückgesetzt.

```
    end if

    t3dScale(2.0,0.5,3.0)
    gObjects[i].t3dDraw()
  end repeat

  gSchritt = gSchritt + 1

  updateStage

  go the frame

end
```

Zum Schluss der Schleife werden die Quader noch skaliert und schließlich gezeichnet.

Dieses Beispiel kann dazu dienen, ein eindrucksvolles Menüsystem zu entwickeln. Zusammen mit den in Director 8 eingeführten Möglichkeiten von Imaging Lingo können die Menüeinträge dann auch dynamisch erzeugt werden.

5.3 Director ab Version 8.5

Director 8.5 bietet zurzeit die umfassendsten 3D-Grafikmöglichkeiten aller am Markt befindlichen Plugins. Dies beinhaltet unter anderem

- Import von komplexen Modellen und Animationen aus kommerziellen 3D-Editoren
- Keyframe- und Bonesplayer-Animationen
- Partikelsysteme
- Komplexe Beleuchtungsmodelle
- Multitexturing (mehrere Texturebenen)
- Vielfältige Interaktionsmöglichkeiten
- Verschiedene nichtrealistische Renderer (z.B. Cartoon)
- Kameraeffekte wie Backdrops und Overlays
- Benutzung von OpenGL oder DirectX zur Grafikbeschleunigung

All diese Möglichkeiten stehen allerdings nicht über ein komfortables Benutzerinterface zur Verfügung, wie z.B. bei einem Programm wie 3D-Studio Max. Stattdessen muss fast alles mittels Lingo programmiert werden. Alle dafür notwendigen Funktionen hier zu erläutern würde bei weitem den Rahmen dieses Buches sprengen. Wir werden in diesem Kapitel zunächst die beiden Beispiele aus dem vorhergehenden Kapitel mit den Möglichkeiten von Director 8.5 umsetzen, um zu sehen, welche Unterschiede es in der Programmierung gibt. Anschießend werden wir noch weitere Möglichkeiten von Director 8.5 untersuchen. Für ein tieferes Studium sollten Sie allerdings ein Buch nur für dieses Thema durcharbeiten.

Beispiel: Texturierter Würfel

Bei diesem Beispiel benötigen wir vier verschiedene Darsteller (siehe Abbildung 5.9):

1. 3D-Sprite: Ein 3D-Sprite ist in Director 8.5 praktisch ein Fenster in eine 3D-Welt. Innerhalb dieser Welt kann es mehrere Modelle oder Akteure geben. Das Sprite selbst kann wie herkömmliche Sprites behandelt werden.
   Das Sprite in unserem Beispiel enthält nichts. Sie erzeugen es einfach, indem Sie das 3D-Fenster öffnen, einen Namen für das Sprite eingeben und ↵ drücken. Das Sprite muss keine Modelle enthalten, da wir diese später selbst erzeugen werden.
2. Initialisierungs-Skript: Dieses Skript erzeugt wie schon zuvor die notwendigen Einstellungen, generiert den Würfel und weist die Textur zu.
3. Schleifen-Skript: Die Schleife nimmt nur noch die gewünschten Änderungen an der 3D-Welt vor, in diesem Fall die Rotation des Würfels.
4. Texturbild: Dies ist das übliche Bild, eine beliebige Pixelgrafik. Aus Performancegründen gibt es für Texturen ein paar Regeln, die wir hier jedoch nicht weiter betrachten wollen. Es sei nur so viel erwähnt,

dass Höhe und Breite eine Zweierpotenz sein sollten und dass alle Texturgrafiken insgesamt nicht zu groß sein dürfen, da sie komplett in den Grafikspeicher der Grafikkarte geladen werden.

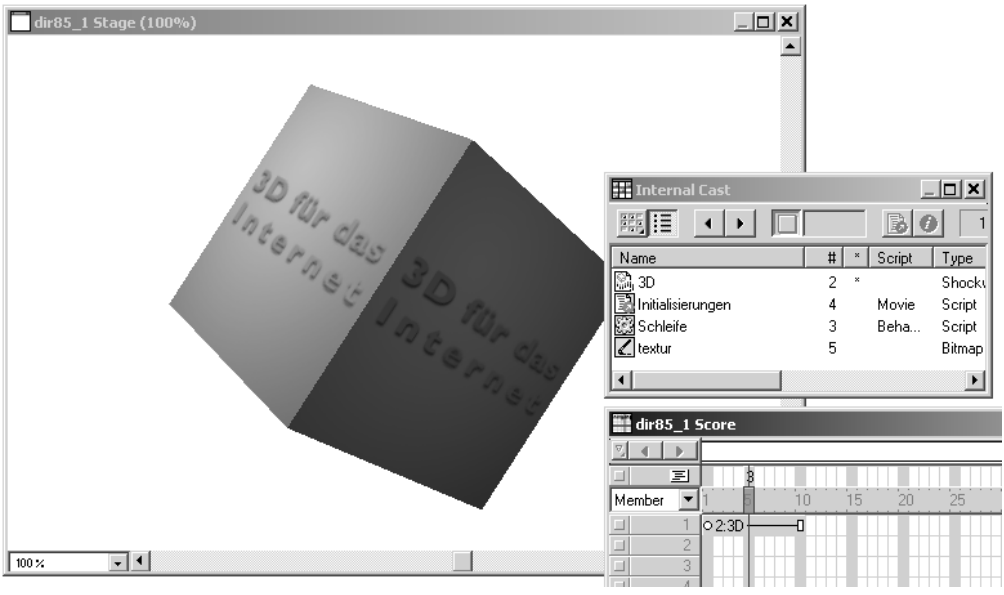


Abbildung 5.9: ***Das Würfelbeispiel in Director 8.5***

Das Initialisierungsskript sieht folgendermaßen aus:

```
global gCube

on startMovie
  pMember = sprite(1).member
  pMember.resetWorld()
```

Wir erzeugen uns zunächst eine Referenz auf das 3D-Sprite für den leichteren Umgang damit. Danach wird die 3D-Welt zurückgesetzt, da Änderungen, die bei einem vorhergehenden Abspielen gemacht wurden, erhalten bleiben.

Erzeugen des 3D-Objektes in Lingo

```
  tCubeRes = pMember.newModelResource("test_cube",#box)
  tCubeRes.length = 80.0
  tCubeRes.height = 80.0
  tCubeRes.width = 80.0
  gCube = pMember.newModel("test_cube",tCubeRes)
```

Hier wird eine Modellressource erzeugt, d.h. es werden die Eckpunkte und Flächeninformationen für einen Würfel mit der Seitenlänge 80 erzeugt. Diese Daten liegen zunächst nur als Grunddaten vor und erzeugen noch nichts Sichtbares in der Welt. Erst durch die letzte Zeile oben wird ein Modell in der Welt erzeugt, welches auf den Grunddaten beruht.

```
tShader = pMember.newShader("test_shader",#standard)
gCube.shaderList = tShader
tTexture = pMember.newTexture("test_texture",
                  #fromCastMember,member("textur"))
tTexture.renderformat = #rgba8880
gCube.shader.texture = tTexture
```

Hier wird als Nächstes ein Shader erzeugt. Dieser enthält alle Definitionen, wie eine Oberfläche aussehen soll. Wir benutzen erst mal nur die Standardeinstellungen und weisen dem Shader nur unsere Textur zu.

```
end startMovie
```

Das ist schon alles, was wir für die Initialisierung benötigen. Die Befehle erscheinen unter Umständen komplexer als in den vorhergehenden Beispielen. Dies liegt schlicht daran, dass wir erheblich mehr Möglichkeiten haben, das Aussehen zu beeinflussen.

Das Schleifenskript für das Beispiel ist sehr einfach:

Die Bewegung des Objektes

```
global gCube

on exitFrame

  gCube.rotate(2,6,4)

  go to the frame

end
```

Der Würfel wird um seinen Mittelpunkt rotiert.

Bei dem Beispiel haben wir uns zu einem großen Teil auf Standardeinstellungen verlassen. Wir haben uns zum Beispiel nicht um die Kameraposition oder Lichter gekümmert. Trotzdem benutzen wir eine Kamera und es ist auch eine Lichtquelle vorhanden, die unser Objekt beleuchtet.

Beispiel: 3D-Menü

Das 3D-Menü gestaltet sich etwas schwieriger in der Programmierung. Als Darsteller benötigen wir die gleichen wie im Beispiel zuvor und zusätzlich die Grafiken für die Menüpunkte (siehe Abbildung 5.10). Wir benötigen auch wieder nur ein 3D-Sprite, da wir uns die vier Quader wieder per Skript erzeugen und alle in derselben Welt existieren.

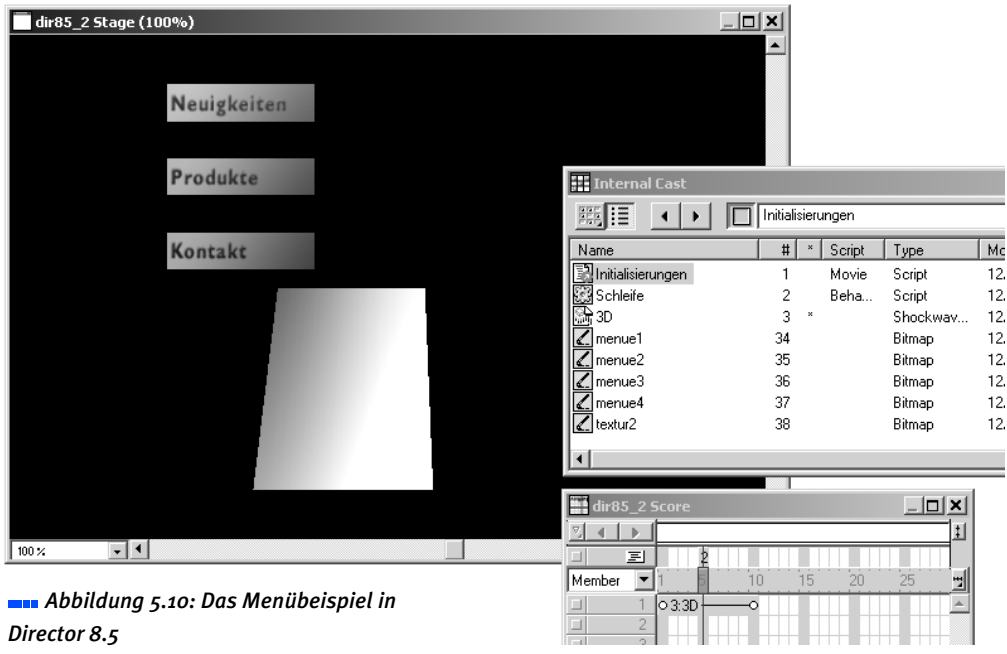


Abbildung 5.10: Das Menübeispiel in Director 8.5

Das Initialisierungsskript stellt sich also wie folgt dar:

```
global gObjects, gCurrentObject, gSchritt, gImpuls
```

Die Referenzen auf die vier Objekte speichern wir wieder in einer Liste `gObjects`. In `gCurrentObject` wird das angeklickte Objekt gespeichert. Die restlichen Variablen werden wie im vorhergehenden Abschnitt wieder für die Animation genutzt.

```
on startMovie
  pMember = sprite(1).member
  pMember.resetWorld()

  tCubeRes = pMember.newModelResource("test_cube",#box)
  tCubeRes.length = 60.0
  tCubeRes.height = 10.0
  tCubeRes.width = 40.0
```

Die Modellressource wird wie gehabt angelegt. Nur die Größenangaben sind an die gewünschte Form angepasst. Wir benötigen das Ganze auch nur einmal, da alle Quader gleich aussehen sollen und somit auch auf der gleichen Ressource beruhen können.

```
gObjects = []
repeat with i=1 to 4
    gObjects[i] = pMember.newModel("test_cube" & i,tCubeRes)
```

In der Schleife wird als Erstes ein Modell für den jeweiligen Quader erzeugt.

```
tShader = pMember.newShader("test_shader"
                                        & i,#standard)
tShader.Texturemode = #wrapplanar
tTexture = pMember.newTexture("test_texture"
           & i,#fromCastMember,member("menue" & i))
tTexture.renderformat = #rgba8880
gObjects[i].shaderList = tShader
gObjects[i].shader.texture = tTexture
```

Hier werden die Shader-Eigenschaften festgelegt, die für alle Quader gleich sind. Für die Textur wird ein kleiner Trick angewendet: Da für alle Seitenflächen eines Quaders das gleiche Bild angegeben wird, wird der „Umwicklungsmodus“ (TextureMode) so festgelegt, dass das Bild nur auf der Kopf- und Fußfläche zu sehen ist. Auf den restlichen Seitenflächen ist praktisch nur noch der Rand des Bildes, skaliert auf die gesamte Fläche, zu sehen.

```
gObjects[i].translate(-40,20*i-30,0)
```

Zum Schluss wird der Quader noch an seine Ausgangsposition verschoben.

```
  end repeat

end
```

Das Schleifenskript muss sich nun um die Animation der Quader bei einem Mausklick kümmern:

Bei einem Mausklick wird das Objekt gefunden, welches am nächsten beim Betrachter liegt.

```
global gObjects, gCurrentObject, gSchritt, gImpuls, gTransform,
gOldObject

on mouseUp me
  gCurrentObject = sprite(1).camera.modelunderloc(the
      clickloc - point(sprite(1).left,sprite(1).top))
```

Bei einem Mausklick irgendwo in dem Sprite wird das Modell ermittelt, welches sich unter der Mausposition befindet.

```
  if (gOldObject <> void) then
                        gOldObject.transform = gTransform
```

Falls gerade schon ein Menü ausgeklappt war, wird dies in seinen ursprünglichen Zustand zurückversetzt.

```
  gOldObject = gCurrentObject
  if (gCurrentObject <> void) then
          gTransform = gCurrentObject.getworldtransform()
```

Falls tatsächlich mit dem Klick ein Modell getroffen wurde, wird dessen gegenwärtige Transformation zwischengespeichert.

```
    gImpuls = 1
    gSchritt = 1
  end
```

Schließlich werden noch die Variablen für die Animation initialisiert.

Beim Verlassen des Frames werden dann die notwendigen Animationen durchgeführt.

```
on exitFrame

  if (gCurrentObject <> void) then
```

Falls ein Quader angeklickt wurde, werden die Animationsteilschritte, wie schon im letzten Abschnitt erklärt, durchlaufen.

```
    if (gImpuls = 1 ) then
      gCurrentObject.translate(2.0, 0.0,2,#world)
      gCurrentObject.rotate(0,-12,0,#self)
    end if
    if (gImpuls = 2 ) then
      gCurrentObject.translate(0, -2,2,#world)
      gCurrentObject.rotate(-6,0,0,#self)
    end if
```

Die Verschiebungen werden hier relativ zum Weltursprung durchgeführt (letzter Parameter #world), während die Rotationen relativ zum Modellursprung durchgeführt werden, damit sich die Quader um ihre eigene Achse drehen und nicht um den Koordinatenursprung.

```
    if (gImpuls > 2 ) then
      gImpulse =0
      gCurrentObject = void
    end if
```

Wenn die Animationen beendet sind, werden die Variablen zurückgesetzt.

```
    if (gSchritt = 15) then
      gSchritt = 1
      gImpuls = gImpuls + 1
    end if
  end if
  gSchritt = gSchritt + 1

  go the frame

end
```

Die restlichen Zeilen folgen dem gleichen Muster, wie es im letzten Abschnitt bei dem Menüsystem erklärt wurde.

Grundlagen 3D in Director 8.5

Sie haben an den vorangegangenen Beispielen gesehen, wie man 3D-Grafik in Director 8.5 erstellt. Wichtig ist dabei zu beachten, dass die schon vorher vorhandenen Funktionalitäten in keiner Weise verändert wurden, und auch die Bedienung des Programms ist in dieser neuen Version weitgehend gleich geblieben. Einzig durch die Einführung eines neuen Sprite-Typs wird die 3D-Grafik unterstützt. Das bedeutet allerdings auch, dass es keine nahtlose Integration von herkömmlichen 2D-Elementen mit 3D-Elementen gibt, sondern man hat immer nur ein rechtwinkliges Fenster in eine 3D-Welt. Es gibt zwar auch Möglichkeiten, z.B. den Hintergrund dieses Fensters durchsichtig zu machen und so Elemente zu überlagern, allerdings geht dies einher mit nicht unerheblichen Geschwindigkeitsverlusten.

Gehen wir noch mal auf einige Besonderheiten im Umgang mit der Software ein, um das Potenzial und die Beschränkungen besser verstehen zu können.

▸ Modellerzeugung

Es gibt grundsätzlich zwei verschiedene Wege, Modelle für die Verwendung mit Director zu erzeugen:

- Erstellung mit einem externen Programm, welches die Geometrie und Animationsdaten im Director-Format exportieren kann
- Programmierung der Modelle in Lingo und somit die Erzeugung zur Laufzeit

Für die erste Möglichkeit stehen Export-Plugins für die meisten großen Softwareprodukte zur Verfügung. 3D Studio Max war eines der ersten, welches auch die Möglichkeit bietet, komplexe Modelle, Keyframeanimationen, Kameras, Lichter und Oberflächen zu exportieren. In der Regel geht das aber nicht 1-zu-1. Da die Animationsprogramme darauf ausgerichtet sind, vorberechnete Filme zu erzeugen, können hier auch rechenintensive Effekte wie Volumenlichter, Radiosity und dergleichen angewendet werden. In Director allerdings wird alles in Echtzeit berechnet und dargestellt. Deshalb können allzu rechenintensive Effekte nicht benutzt und auch nicht exportiert werden.

Das Export-Fenster in 3D Studio Max sehen Sie in Abbildung 5.11. Hier kann ausgewählt werden, welche Teile exportiert werden sollen und wie beispielsweise Animationen behandelt werden.

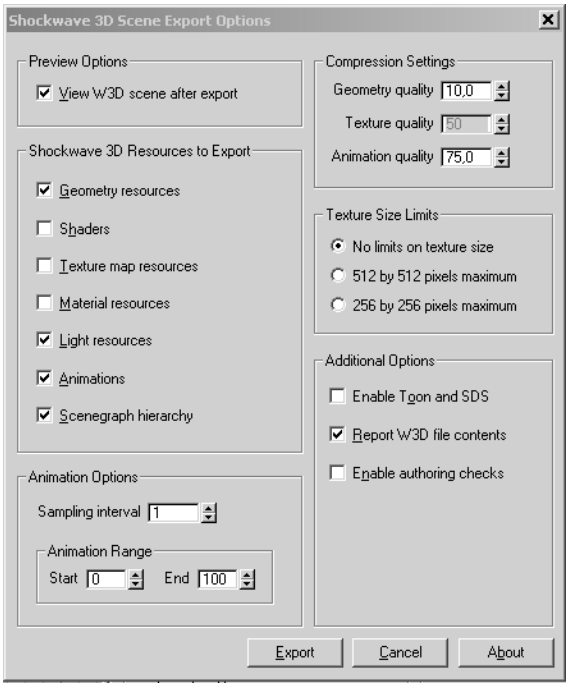


Abbildung 5.11: Shockwave-Export-Fenster in 3D Studio Max

Innerhalb von Director können Sie dann auf alle exportierten Elemente zugreifen, d.h. auf alle Modelle und Untergeometrien, auf Texturen und auf Animationen.

Mit Lingo erzeugte Szenen sind erheblich kompakter als mit einem Editor erstellte.

Die Erstellung aller Geometrien und Animationen in einer externen Software mag komfortabler sein als die Erzeugung per Lingo. Allerdings interessieren wir uns ja vor allem für Filme, die für das Internet erzeugt werden, d.h. die Größe des Films ist entscheidend für Ladezeiten. Mit Lingo erzeugte Daten sind in der Regel erheblich kleiner und somit schneller herunterzuladen. Ein Beispiel dafür sehen Sie weiter unten bei dem Beispiel zum Balkendiagramm. Es stehen die folgenden Grundkörper für eine Modellerzeugung mit Lingo zur Verfügung:

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

Weiterhin besteht noch die Möglichkeit ein Mesh anzulegen, also ein Objekt komplett durch die Angabe einzelner Eckpunkte und Flächendefinitionen zu generieren.

Natürlich ist es auch möglich, beide Methoden der Modellerzeugung zu kombinieren. Sie können die Modellgeometriedaten in einer externen Software erstellen und nur die Geometriedaten exportieren. In Director erzeugen Sie dann die Oberflächen und Animationen.

▸ Transformationen und Animation

Intern arbeitet Director mit Transformationsmatrizen (siehe Grundlagenkapitel), die man theoretisch auch direkt bearbeiten kann. Viel komfortabler geht es aber über die Funktionen zur Rotation, Translation und Skalierung. Diese bieten die Möglichkeit, Objekte beliebig im Raum zu bewegen, zu drehen (auch relativ zu beliebigen anderen Objekte) und zu skalieren. Man kann weiterhin die aktuelle Transformationsmatrix speichern und das Objekt zu einem späteren Zeitpunkt wieder in diesen Zustand versetzen.

Animieren kann man die Objekte einer 3D-Welt entweder „per Hand", indem man framebasiert die Transformationen benutzt und alle Bewegungen in Lingo programmiert. Einfacher ist es aber, wenn man den Keyframeplayer oder den Bonesplayer benutzt.

Ein Beispiel zur Keyframe-Animation finden Sie am Ende dieses Kapitels.

Der Keyframeplayer kann auf Animationen zugreifen, die mit einem 3D-Editor erstellt und als Shockwave-Datei exportiert wurden. Damit wird es dann möglich, einzelne Animationen genau zu gewünschten Zeitpunkten abzuspielen, die Geschwindigkeit zu steuern und zwischen Teilanimationen überzublenden.

Der Bonesplayer bezieht seinen Namen tatsächlich von virtuellen Knochen. In einem 3D-Editor kann zu einem Modell ein Skelett definiert werden. Die Modellbestandteile können dann mit diesem Skelett verknüpft werden und Bewegungen des Skelettes wirken sich auf die Modellbestandteile aus. Am einfachsten kann man sich hierfür einen Roboter vorstellen. Mit der Bones-Animation kann man für einen Schritt, den die Figur macht, einfach das Skelett entsprechend bewegen, ohne alle Punkte, die das Bein definieren, in die verschiedenen Positionen bringen zu müssen, die für einen Schritt notwendig sind.

▸ Oberflächen und Realismus

Alle Objekte in einer Director 3D-Welt haben ein vordefiniertes Standardaussehen mit einer Standardtextur (ein Karomuster). Um dieses Aussehen zu verändern, definiert man einen Shader, der die Oberflächenbeschaffenheit genau beschreibt. Der Shader besitzt Eigenschaften, die beschreiben, in welcher Farbe die Oberfläche Umgebungslicht (ambient), direkten Lichteinfall (diffuse) und Glanzpunkte (specular) reflektiert. Er definiert, wie stark die Oberfläche glänzt, ob sie selbst Licht ausstrahlt, wie durchsichtig sie ist oder ob von der Oberfläche nur Punkte oder Linien zu sehen sein sollen. In Abbildung 5.12 sehen Sie ein Objekt ohne Textur, bei der nur die Standardeigenschaften definiert sind.

Ein wichtiger Bestandteil des Shaders sind aber vor allem die Texturen. Erst durch eine gute Kombination von Texturen wirkt ein Objekt wirklich realistisch. In Abbildung 5.13 sehen Sie ein Objekt, dem ein Wolkenbild als Textur zugewiesen wurde. Ein Standard-Shader bietet acht Texturkanäle an. Das bedeutet, das man acht Texturen übereinander legen kann. Das macht für normale Texturen wie zum Beispiel ein Schachbrettmuster kei-

nen Sinn. Die Texturkanäle können aber auch für Spezialeffekte benutzt werden. So kann man beispielweise eine `reflectionmap` definieren. Diese bewirkt den Eindruck, als würde sich das Texturbild (sphärisch verzerrt) in der Oberfläche spiegeln (siehe Abbildung 5.14). Dieser Effekt wird beim nächsten Beispiel des Balkendiagramms benutzt. Ein anderer Effekt ist eine `opacitymap`. Damit kann man definieren, dass bestimmte Teile der Objektoberfläche durchsichtig sind.

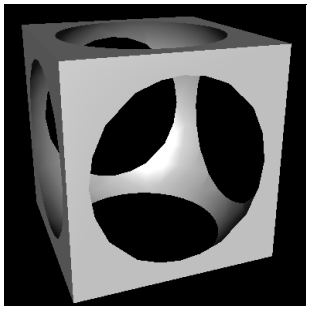


Abbildung 5.12: Ohne Textur

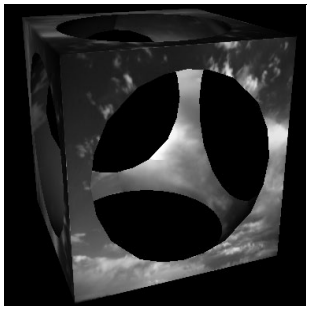


Abbildung 5.13: Mit Textur

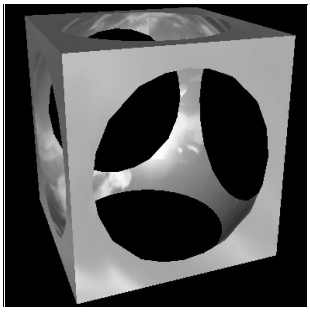


Abbildung 5.14: Mit Reflectionmap

Mit all diesen Möglichkeiten kann man sehr realistisch wirkende Oberflächen generieren. Genauso reizvoll kann es aber auch sein, unrealistische Oberflächen zu erzeugen. Für diesen Zweck gibt es die speziellen nichtfotorealistischen Shader oder Modifizierer. Die speziellen Shader sind:

- Painter: Dieser Shader zeichnet ein Objekt mit einer reduzierten Anzahl an Farben (siehe Abbildung 5.15).
- Newsprint: Dieser Shader simuliert das Aussehen eines Zeitungsdrucks (siehe Abbildung 5.16).
- Engraver: Dieser Shader soll das Aussehen von graviertem Metall simulieren (siehe Abbildung 5.17).

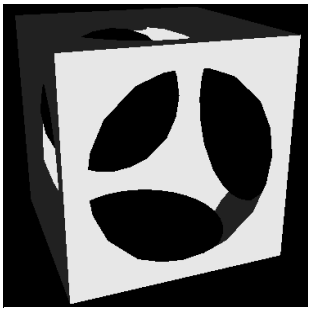


Abbildung 5.15: Painter

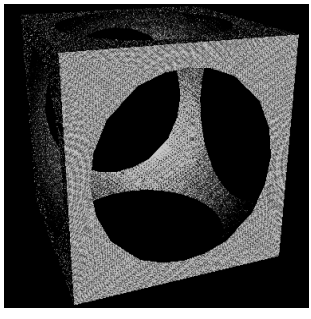


Abbildung 5.16: Newsprint

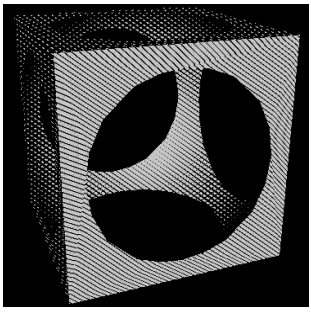


Abbildung 5.17: Engraver

Für die Nicht-Standard-Shader ist das spätere Aussehen immer stark von den jeweils eingestellten Parametern abhängig. Deshalb sollte man damit einfach viel ausprobieren, bis man den gewünschten Effekt erreicht. Es wird aber deutlich, dass man hiermit viele Möglichkeiten an die Hand bekommt, wirklich ungewöhnliche Grafiken und Animationen zu erzeugen.

Beispiel: Balkendiagramm

Ein weiteres Beispiel, welches wir hier mit den neuen Möglichkeiten von Director 8.5 entwickeln möchten, ist ein Balkendiagramm. Balkendiagramme findet man sehr häufig auf Webseiten, sei es zur Visualisierung von Kursverläufen, Statistiken oder anderen numerischen Daten.

Das Balkendiagramm, welches wir entwickeln wollen, soll dreidimensional sein, also nicht nur richtige Balken haben, sondern auch mehrere Reihen. Für das Beispiel werden wir zufällige Werte nehmen. Das spätere Ergebnis ist zu sehen in Abbildung 5.18.

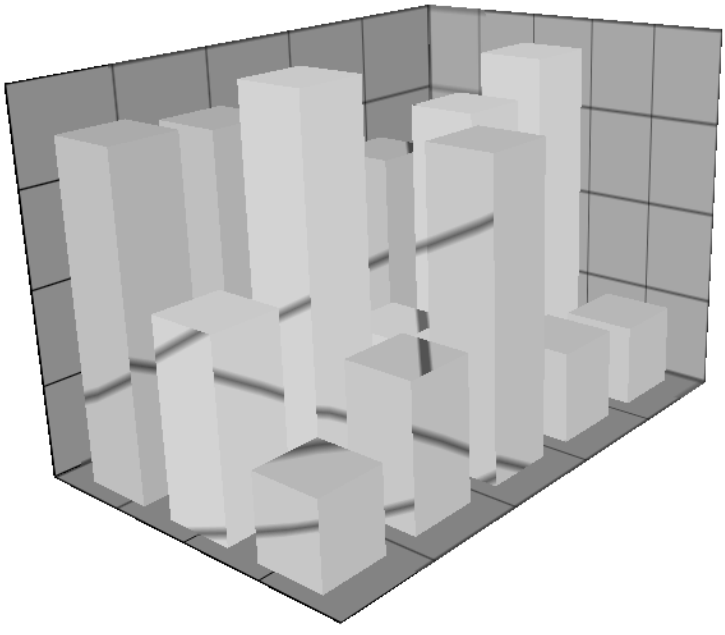


Abbildung 5.18: 3D-Balkendiagramm

Solch ein Balkendiagramm in 3D zu programmieren hat verschiedene Vorteile:

- Es ist übersichtlich, vor allem durch die Tatsache, dass man es drehen und von allen Seiten betrachten kann.
- Es ist attraktiv und somit hervorragend geeignet, einen Geschäftsbericht, der ansonsten nicht viel Positives zeigt, wenigstens schön darzustellen.
- Es ist sehr kompakt. Die Beispieldatei, die wir hier erzeugen, hat eine Größe von etwa 8 KB. Das entspricht normalerweise schon allein einer zweidimensionalen Grafik im GIF-Format, welche dann aber keine Interaktion bietet.

Den fertigen Film finden Sie auf der CD unter dem Namen *dir85_chart* im entsprechenden Verzeichnis.

Betrachten wir zunächst einmal die einzelnen Bestandteile des Films (siehe Abbildung 5.19):

- 3D-Darsteller `Chart`: Dieser 3D-Darsteller ist völlig leer und kann einfach über den entsprechenden Knopf in Director angelegt werden. Die einzige Änderung gegenüber den Standardeinstellungen ist, dass die Hintergrundfarbe auf Weiß gesetzt wurde.
- Initialisierungsskript `init`: Dieses Skript wird beim Start des Films aufgerufen und enthält alle notwendigen Initialisierungen sowie zwei Funktionen, die bei Bedarf aufgerufen werden können. Dazu kommen wir weiter unten noch.
- Vektorgrafik `button`: Dies ist ein einfaches Rechteck, welches im Film platziert ist, um ein wenig Interaktion zu ermöglichen. Bei einem Klick auf den Knopf werden neue zufällige Werte in dem Diagramm angezeigt.
- Verhaltensskript `klick`: Dieses Skript sorgt dafür, dass bei einem Klick auf den Knopf die entsprechende Aktion ausgeführt wird.
- Verhaltensskript `schleife`: Dieses Skript sorgt dafür, dass der Film immer den gleichen Frame wiederholt.

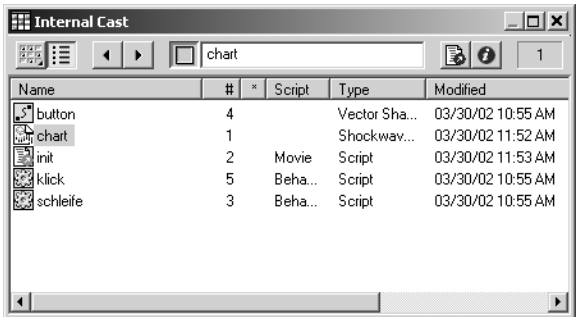


Abbildung 5.19: Die Bestandteile des Films

Beginnen wir zuerst damit, uns das Initialisierungsskript genauer anzusehen:

```
global scene, grid, bars

on startMovie
  -- Zeichne das Rasterbild
  gridImg = image(512,256,32,0)
  grayCol = rgb(200,200,200)
  blackCol = rgb(0,0,0)
  gridImg.fill(0,0,511,255,grayCol)
  repeat with i = 0 to 5
    repeat with m = 0 to 5
      gridImg.draw(m*102,0,m*102,255,blackCol)
      gridImg.draw(0,m*64,511,m*64,blackCol)
    end repeat
  end repeat
  gridImg.draw(511,0,511,255,blackCol)
  gridImg.draw(0,255,511,255,blackCol)
```

Mit Lingo erstellte Texturen sind erheblich kompakter als vordefinierte Bilder.

Um die Datei klein zu halten und die Möglichkeiten von Director auszuschöpfen, wird eine Textur mit Imaging Lingo erstellt. Für den Hintergrund der Balken benötigen wie ein Rasterbild, welches eine Größeneinteilung bietet. Die obigen Zeilen erzeugen ein leeres Bildobjekt, füllen dieses mit einem hellen Grauwert und zeichnen dann in regelmäßigen Abständen horizontale und vertikale Linien darauf. Das Ergebnis davon ist in Abbildung 5.20 zu sehen.

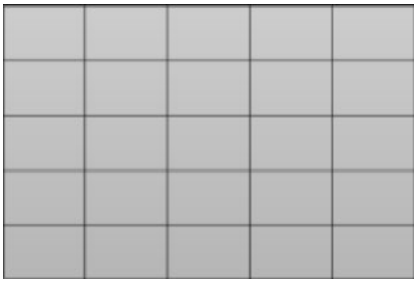


Abbildung 5.20: Das erzeugte Rasterbild

```
  -- Initialisiere die 3D-Welt
  scene = sprite(1).member
  scene.resetWorld()
  scene.light[1].color = rgb(200,200,200)
```

Die 3D-Welt ist zunächst völlig leer. Wir führen dennoch einen Reset durch, um Veränderungen des letzten Abspielens zurückzusetzen. Eine Referenz auf die Welt wird in der Variablen `scene` gehalten. Ein Standard-3D-Darsteller enthält immer eine Lichtquelle, die wir hier benutzen wollen.

```
-- Bilde den umgebenden Kasten
res = scene.newModelResource("res_grid",#box)
res.length = 100.0
res.height = 100.0
res.width = 150.0
grid = scene.newModel("grid",res)
```

Für das Raster um die Balken herum erzeugen wir zunächst eine rechteckige Box und fügen sie in die 3D-Welt unter dem Namen `grid` ein.

```
shd = scene.newShader("sha_grid",#standard)
shd.diffuse = rgb(200,200,0)
shd.ambient = rgb(50,50,50)
grid.shaderList = shd
tex = scene.newTexture("tex_grid",
                  #fromImageObject,gridImg)
tex.renderformat = #rgba8888
grid.shader.texture = tex
```

Damit diese umgebende Box das Raster korrekt anzeigt, müssen wir ihr einen Shader und als Textur das Bild des Rasters zuweisen.

```
grid.visibility = #back
```

Vielleicht haben Sie sich schon gewundert, wie man die Balken sehen soll, wenn diese sich in einer Box befinden. Die Zeile oben dient dazu, nur die Rückseiten der Box anzuzeigen. Bei der `visibility`-Eigenschaft kann man angeben, ob nur die Vorderseite eines Objektes, nur die Rückseite, beide Seiten oder keine zu sehen sein soll. Dadurch, dass nun also nur die Rückseiten, nicht aber die Vorderseite angezeigt werden, kann man in die Box hineinsehen, wie auch immer die gedreht ist. In Abbildung 5.21 sind beispielsweise die Rückseiten der beiden Seitenflächen dem Betrachter zugewandt.

```
-- Bilde die einzelnen Balken
repeat with i = 0 to 2
  shd = scene.newShader("sha_bar" & i,#standard)
  if (i = 0) then col=rgb(200,0,0)
  if (i = 1) then col=rgb(0,200,0)
  if (i = 2) then col=rgb(0,0,200)
  shd.diffuse = col
  shd.ambient = col
```

Hier wird für jede der drei Balkenreihen ein eigener Shader definiert, so dass die Reihen unterschiedliche Farben haben.

```
  shd.reflectionMap = tex
  shd.textureTransformList[3].scale(0.5,0.5,0.5)
```

Als besonderer Effekt wird dem Balken noch das Rasterbild als Reflectionmap zugewiesen. Dadurch sieht es so aus, als würde sich das Raster in den Balken spiegeln. Um das ansprechender aussehen zu lassen,

muss allerdings die Größe der Textur (die noch in der Variablen tex gespeichert ist) angepasst werden.

```
    repeat with m = 0 to 4
      res = scene.newModelResource("res_bar_" & i & "_"
                                   & m,#box)
      res.length = 20.0
      res.height = 10.0
      res.width = 20.0
      obj = scene.newModel("bar_" & i & "_" & m,res)
      obj.shaderList = shd
      obj.shader.texture = void
      grid.addChild(obj)
      obj.translate(m*30 - 60,-45,i*30 - 30)
```

Für jeden Balken wird eine eigene Modell-Ressource angelegt mit einer Standardhöhe von 10 Einheiten. Es wäre auch möglich, nur eine Modell-Ressource für alle Balken zusammen anzulegen und diese für jeden Balken zu skalieren. Um die Höhe aber leichter bestimmen zu können, benutzen wir eigene Modell-Ressourcen. Den Balken wird nun der Shader zugewiesen und das Modell des Balken selber wird als Kind der Box mit dem Rastermuster zugewiesen. Dadurch wirkt sich jede Drehung der Rasterbox auch auf die Balken aus.

```
    end repeat
  end repeat

  grid.rotate(30,0,0)
  grid.rotate(0,-30,0)
  fillChart
```

Nach dem Erzeugen der Balken wird die Rasterbox schließlich noch so gedreht, dass man eine gute Sicht auf alles hat. Die Funktion `fillChart` belegt die Balken mit zufälligen Höhen.

```
end

on fillChart
  set bars = []
  repeat with i = 0 to 2
    set dum = []
    repeat with m = 0 to 4
      dum.add (random(100))
    end repeat
    bars.add(dum)
  end repeat
end
```

Die Funktion `fillChart` legt eine zweidimensionale Liste für die Balkenhöhen an und füllt diese mit zufälligen Werten.

```
on setBar m,n,h
  ho = scene.modelresource("res_bar_" & m & "_"
                           & n).height
  scene.modelresource("res_bar_" & m & "_" & n).height =
                                                        h
  scene.model("bar_" & m & "_" & n).translate(0,
                                            (h - ho)/2,0)
end
```

Die Funktion `setBar` ändert für den Balken in Reihe m und Spalte n die Höhe auf den Wert `h`. Da sich die Höhe des Balken zu gleichen Teilen um den Mittelpunkt des Balkens verteilt, muss der Balken anschließend noch in der Höhe entsprechend verschoben werden.

Das Schleifenskript sieht folgendermaßen aus:

```
global scene, grid, bars

on exitFrame me
  grid.rotate(0,0.2,0)
```

Bei jedem Durchlauf wird das Diagramm ein wenig um die y-Achse gedreht. Bei einer erweiterten Version des Beispiels könnte man solch eine Drehung natürlich dem Benutzer überlassen.

Bei einem Klick auf den Knopf, der neben dem Diagramm platziert ist (außerhalb der 3D-Welt), wird die zweidimensionale Liste mit den Balkenwerten neu belegt. Allerdings werden die Höhen der Balken im Diagramm nicht sofort neu gesetzt, sondern „gleiten“ in die richtige Höhe. Das wird erreicht durch die folgende Schleife.

```
  repeat with i = 0 to 2
    repeat with m = 0 to 4
      h = scene.modelresource("res_bar_" & i & "_"
                              & m).height
      if (abs(h - bars[i+1][m+1])>1) then
        setBar(i,m,(h + bars[i+1][m+1]) / 2)
      end if
    end repeat
  end repeat
```

Diese Schleife prüft für jeden Balken, ob seine Höhe der entsprechenden Angabe in der Liste entspricht. Falls der Abstand größer als eine Einheit ist, wird die Höhe des Balkens um die Hälfte des Gesamtabstands angepasst. Dadurch springt der Balken nicht in die Position, sondern nähert sich langsam an.

```
  go to the frame
end
```

Schließlich wird noch zum gleichen Frame zurückgesprungen, womit die Schleife komplett wäre.

Das Aussehen des kompletten Beispiels ist in Abbildung 5.21 zu sehen. Kommen wir noch einmal auf die Größe des Beispiels zu sprechen. Die dcr-Datei, die Director beim Publizieren erstellt, ist ca. 8 KB groß. Director bietet damit beliebige Interaktionsmöglichkeiten, es könnten etwa beim Klick auf einen Balken Informationen über diesen Wert angezeigt werden. Weiterhin können neue Daten problemlos vom Server nachgeladen werden, ohne dass ein neues Bild generiert werden müsste.

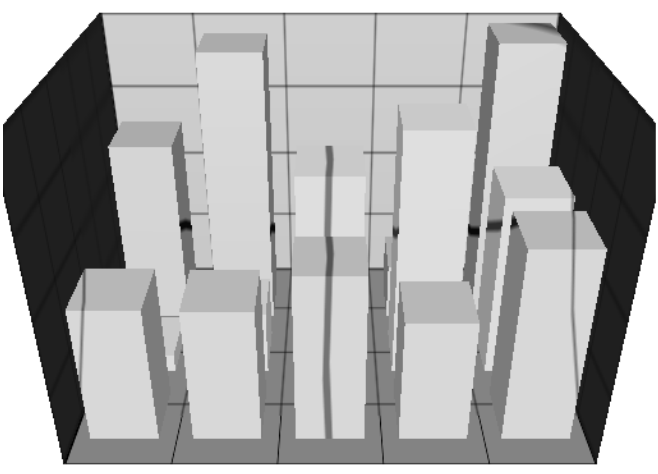


Abbildung 5.21: Das fertige Beispiel

Beispiel: Partikelsystem

Eine der sehr mächtigen neuen Funktionen Director 8.5 sind die Partikelsysteme. Damit ist es möglich, mit wenig Aufwand recht realistische Effekte zu erzeugen.

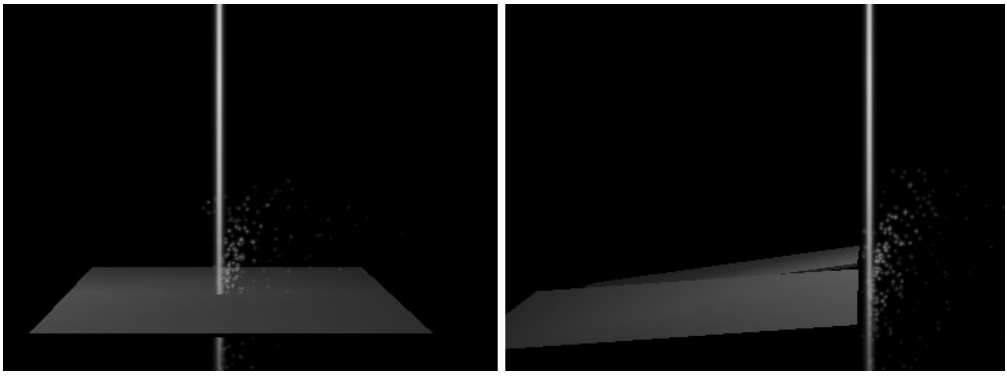
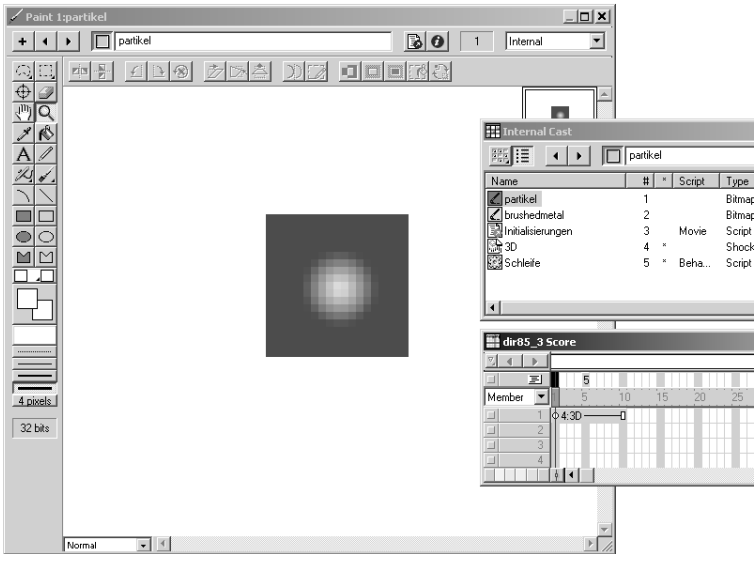


Abbildung 5.22: Die Simulation eines Laserschnittes

In Abbildung 5.22 sehen Sie zum Beispiel die Simulation eines Laserschnittes durch ein Blech. Die sprühenden Funken dabei sind mit einem Partikelsystem erzeugt worden.

Sehen wir uns an, wie das Beispiel aufgebaut ist.

Abbildung 5.23:
Das Beispiel in Director



Der Film enthält zwei Texturen, `brushedmetal` für das Blech und `partikel` für die Partikel und den Laserstrahl. In Abbildung 5.23 ist das Bild für die Partikel zu sehen, wobei das Bild auch einen Alphakanal hat, der die rote Fläche zum Rand hin ausblendet.

Weiterhin gibt es in dem Film noch den Dummy 3D-Darsteller und die beiden bekannten Skripte für die Initialisierungen und die Animationsschleife.

Das Initialisierungsskript wird im Folgenden beschrieben. Ich werde hier nicht alles im Einzelnen erklären, sondern nur auf Besonderheiten eingehen.

```
global gPlatte1, gPlatte2, gFunken, gStep, resFunke

on startMovie
  pMember = sprite(1).member
  pMember.resetWorld()
  pMember.deletelight(2)
  pMember.newlight("licht",#point)
  pMember.light[2].translate(-20,10,20)
  pMember.light[2].attenuation = vector(1.0, 0.01, 0.0)
```

Hier wird das zweite Standardlicht durch ein Punktlicht ersetzt, um etwas mehr „Atmosphäre“ zu erzeugen.
Im Folgenden werden zwei separate Platten erzeugt, die zusammen dann das Blech bilden.

```
tCubeRes1 = pMember.newModelResource("res_platte",#box)
tCubeRes1.length = 50.0
tCubeRes1.height = 1.0
tCubeRes1.width = 100.0
gPlatte1 = pMember.newModel("platte",tCubeRes1)
gPlatte2 = pMember.newModel("platte2",tCubeRes1)
tShader = pMember.newShader("sha_platte",#standard)
gPlatte1.shaderList = tShader
gPlatte2.shaderList = tShader
tTexture = pMember.newTexture("tex_platte",
           #fromCastMember, member("brushedmetal"))
tTexture.renderformat = #rgba8880
gPlatte1.shader.texture = tTexture
gPlatte2.shader.texture = tTexture
gPlatte1.translate(70,-50,25)
gPlatte2.translate(70,-50,-25)
```

Ab hier erfolgt die Definition des Lasers, der einfach eine selbst leuchtende Box ist, die das Partikel als Textur erhält. Dadurch, dass die Box so sehr in die Länge gezogen ist, erscheint das Ganze als ein Strahl.

```
tCubeRes2 = pMember.newModelResource("res_laser",#box)
tCubeRes2.length = 5.0
tCubeRes2.height = 500.0
tCubeRes2.width = 5.0
gLaser = pMember.newModel("laser",tCubeRes2)
tShader = pMember.newShader("sha_laser",#standard)
tShader.emissive = rgb(255,255,255)
gLaser.shaderList = tShader
tTexture = pMember.newTexture("tex_laser",
               #fromCastMember,member("partikel"))
tTexture.renderformat = #rgba8888
gLaser.shader.texture = tTexture
gLaser.translate(0,30,0)
```

Hier kommt nun die Definition des Partikelsystems, wobei die Lebensspanne auf eine Millisekunde gesetzt wird. Sie wird auf einen „sichtbaren“ Wert gesetzt, wenn die Platte den Strahl erreicht.

```
resFunke = pMember.newmodelresource("funken",
                                    #particle)
resFunke.emitter.maxspeed = 100
resFunke.emitter.numparticles = 500
resFunke.emitter.direction = vector(1,4,0)
resFunke.emitter.angle = 20
resFunke.sizerange.start = 3
resFunke.sizerange.end = .1
resFunke.blendrange.start = 60
resFunke.blendrange.end = 0
resFunke.lifetime = 1
resFunke.texture = tTexture
```

```
  resFunke.Gravity = vector(0,-4,0)
  gFunken = pMember.newmodel("funken", resFunke)
  gFunken.translate(0,-50,0)
  gStep = 70
end startMovie
```

Die Schleife in Frame 5 übernimmt nur noch die Aufgabe, die beiden Plattenteile zu bewegen und die Funken im richtigen Moment einzuschalten.

```
global gPlatte1, gPlatte2, gFunken, gStep, resFunke

on exitFrame me
  gPlatte1.translate(-1,0,0)
  gPlatte2.translate(-1,0,0)

  if (gStep = 50) then resFunke.lifetime = 1800
```

Hier wird die Lebensspanne eines Funken auf 1,8 Sekunden gesetzt, er hat also genügend Zeit, zu verglühen.

```
  if (gStep = -50) then resFunke.emitter.loop = false
```

Anstatt zum Ausschalten des Funkenflugs die Lebensspanne wieder auf den Minimalwert zu setzen, wird hier das weitere Ausstoßen von Funken ausgeschaltet. Dadurch verschwinden nicht gleich alle Funken, sondern die, die noch da sind, haben noch Zeit zu verglühen, wodurch die Szene realistischer wird.

```
  if (gStep < -50) then
    gPlatte1.rotate(3,0,2)
    gPlatte2.rotate(-1,0,4)
  end if

  gStep = gStep - 1
  go to the frame
end
```

Beispiel: Jukebox

Das Beispiel „Jukebox“ zeigt eine Anwendung von einer in 3D Studio Max vordefinierten Szene und deren Animation in Director. Das Beispiel finden Sie auf der CD unter dem Namen *dir85_menueRad.dir*. Es simuliert ein Menüsystem in der Art einer Jukebox. Um die Programmierung der Interaktion einfach zu halten kann man irgendwo auf das Sprite klicken, um das Öffnen eines Untermenüs zu bewirken und es ebenso wieder zu schließen (siehe Abbildung 5.24).

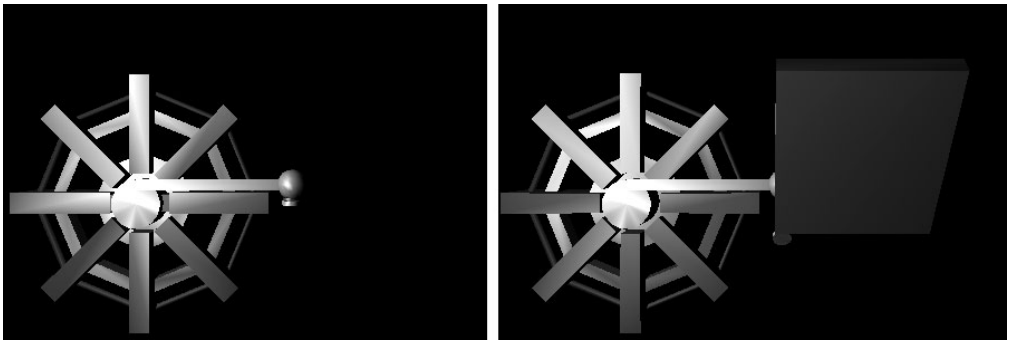


Abbildung 5.24: Menüsystem

Dieses Beispiel soll zeigen, wie einfach man auf importierte Modelle zugreifen kann und wie man einfach Transformationen durchführt.

Das Modell selbst wurde in 3D Studio Max erstellt. Es besteht aus dem Rad, drei Gelenkarmen und einem Quader. Schon innerhalb von 3D Studio Max wurden die Mittelpunkte der einzelnen Objekte in die jeweiligen Gelenkachsen gelegt. Der Film enthält außer dem 3D-Objekt noch vier Skripte, die im Folgenden näher erläutert werden sollen.

```
global scene
global arm1, arm2, arm3, wheel, box, speed

on startMovie

  scene = member("menu")
  scene.resetWorld()

  wheel = scene.Model[1]
  arm1 = scene.Model[3]
  arm2 = scene.Model[4]
  arm3 = scene.Model[2]
  box = scene.Model[5]

  arm3.rotate(0,0,-90)

  speed = 5
end
```

Das Initialisierungsskript speichert Referenzen auf die einzelnen Objekte in der 3D-Welt in globale Variablen. Außerdem wird das Objekt `arm3` noch in die richtige Ausgangslage gebracht. Die Variable `speed` gibt die Winkeländerung während eines Durchlaufs, also die Geschwindigkeit an, mit der die Drehungen durchgeführt werden. Sie könnte auch auf 15 gesetzt werden.

Das folgende Skript enthält den Code für Frame 1 und wartet im Wesentlichen nur auf einen Mausklick auf das Sprite:

```
global scene
global step

on exitFrame me
  go the frame
end

on mouseUp me
  step = 0
  go next
end
```

Sobald mit der Maus auf das Sprite geklickt wurde, springt der Film zum Öffnen-Skript. Dieses führt das Öffnen in 4 Schritten durch:

```
global scene
global arm1, arm2, arm3, wheel, box
global step, speed

on exitFrame me

  case step of
    0 :
      box.visibility = #none
      wheel.rotate(0,speed,0)
      if (wheel.transform.rotation.y mod 45 = 0) then
        step = 1
        box.visibility = #front
      end if
    1 :
      arm3.rotate(0,0,speed)
      if (arm3.transform.rotation.z >= 0) then step = 2
    2 :
      arm3.rotate(0,0,-speed)
      box.rotate(0,0,-speed,arm3)
      if (box.transform.rotation.z <= 95 and
          box.transform.rotation.z > 80) then
        step = 3
      end if
    3 :
      arm2.rotate(-speed,0,0)
      arm3.rotate(-speed,0,0,arm2)
      box.rotate(-speed,0,0,arm2)
      if (arm2.transform.rotation.x <= -90) then step = 4
  end case

  go the frame
end
```

Die einzelnen Schritte sind deutlich zu erkennen. Sobald ein Schritt vollendet ist, wird die Variable `step` um 1 erhöht und somit der nächste Schritt durchgeführt. Für einen Schritt wird jeweils geprüft, ob der entsprechende Winkel bereits erreicht wurde, damit zum nächsten Schritt gegangen werden kann.

Bei manchen Rotationen sehen Sie noch einen vierten Parameter angegeben. Dies bewirkt, dass die Drehung nicht um den Ursprung des Objektes durchgeführt wird, sondern um den des als Parameter übergebenen Objektes.

```
on mouseUp me
  if (step = 4) then
    step = 3
    go next
  end if
end
```

Nachdem alle Schritte durchgeführt wurden, wird wieder auf einen Mausklick gewartet und dann das Schließen durchgeführt.

Das Schließen-Skript arbeitet die einzelnen Schritte in umgekehrter Reihenfolge ab, ist ansonsten aber genauso aufgebaut wie das Öffnen-Skript:

```
global scene
global arm1, arm2, arm3, wheel, box
global step, speed

on exitFrame me

  case step of
    0 :
      go 1
    1 :
      arm3.rotate(0,0,-speed)
      if (arm3.transform.rotation.z <= -90) then step = 0
    2 :
      arm3.rotate(0,0,speed)
      box.rotate(0,0,speed,arm3)
      if (box.transform.rotation.z >= 0 and
      box.transform.rotation.z < 10) then
        step = 1
      end if
    3 :
      arm2.rotate(speed,0,0)
      arm3.rotate(speed,0,0,arm2)
      box.rotate(speed,0,0,arm2)
      if (arm2.transform.rotation.x >= 0) then step = 2
  end case

  go the frame
end
```

Beispiel: Film

Der große Vorteil der Programmierung mit Lingo ist, dass die resultierenden Filme sehr kompakt und damit gut für die Benutzung auf einer Website geeignet sind. Die Problematik daran ist allerdings die Konstruktion der Modelle und Animationen mit Lingo. Im letzten Beispiel hatten wir ein mehr oder weniger kompliziertes Modell mit einem 3D-Editor erstellt und die Animationen dann in Lingo programmiert. Wenn man allerdings komplizierte Animationen darstellen möchte, wird die Programmierung mit Lingo sehr schwierig. Deshalb werden wir in diesem Beispiel die Animation in 3D Studio Max erstellen und die Modelle in Lingo.

Das Beispiel erzeugt einen kleinen Film, der etwa als Werbefilm oder als Intro benutzt werden könnte (siehe Abbildung 5.25). Die zugehörigen Dateien finden Sie im Director-Verzeichnis unter dem Namen *dir85_flight*.

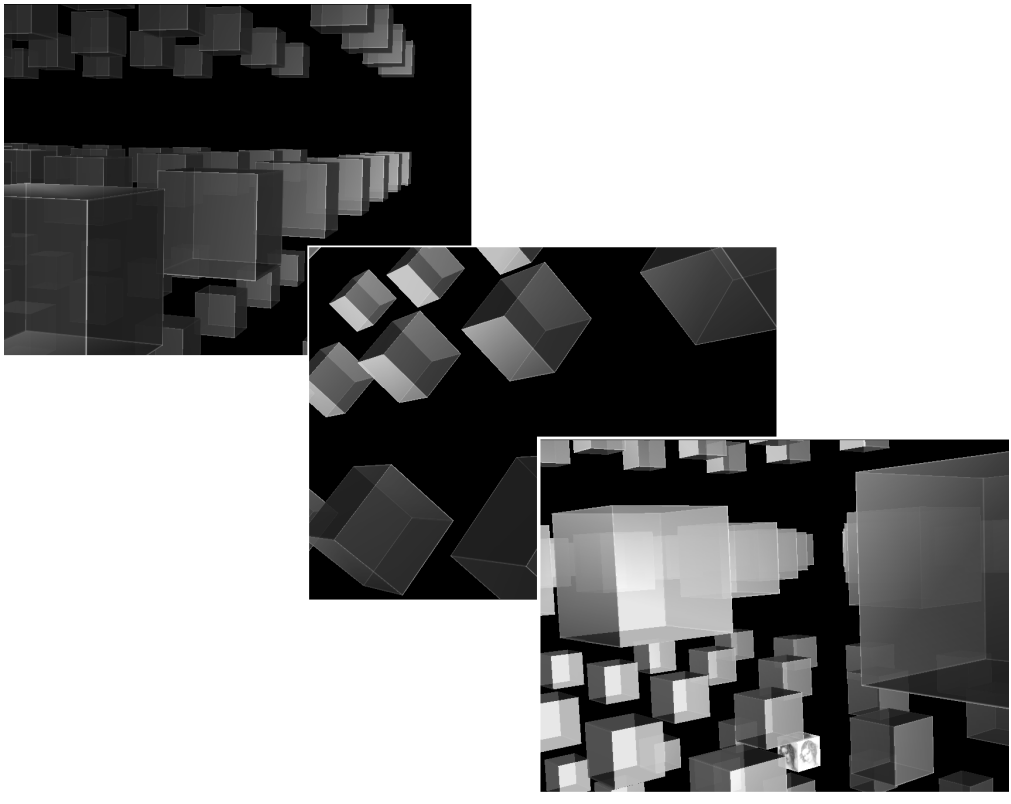


Abbildung 5.25: Filmausschnitte

▶ Erstellung der Animation

Der Film enthält 4 mal 8 mal 8 Würfel, die gleichmäßig um den Koordinatenursprung angeordnet sind. Die Kamera fliegt durch diese Würfel hindurch und bleibt bei einzelnen Würfeln länger stehen. Als Einleitung des Fluges verfolgt die Kamera zunächst drei einzelne Würfel, die sich erst noch in das Feld einordnen.

Die Kamerabewegung wird über eine Keyframe-Animation erzeugt. Sie finden die entsprechende Datei für 3D Studio Max im Director-Verzeichnis. Vor dem Export der Daten werden die Würfel allerdings gelöscht. Sie dienen lediglich der Erstellung der korrekten Animation, werden später aber mit Lingo direkt erzeugt.

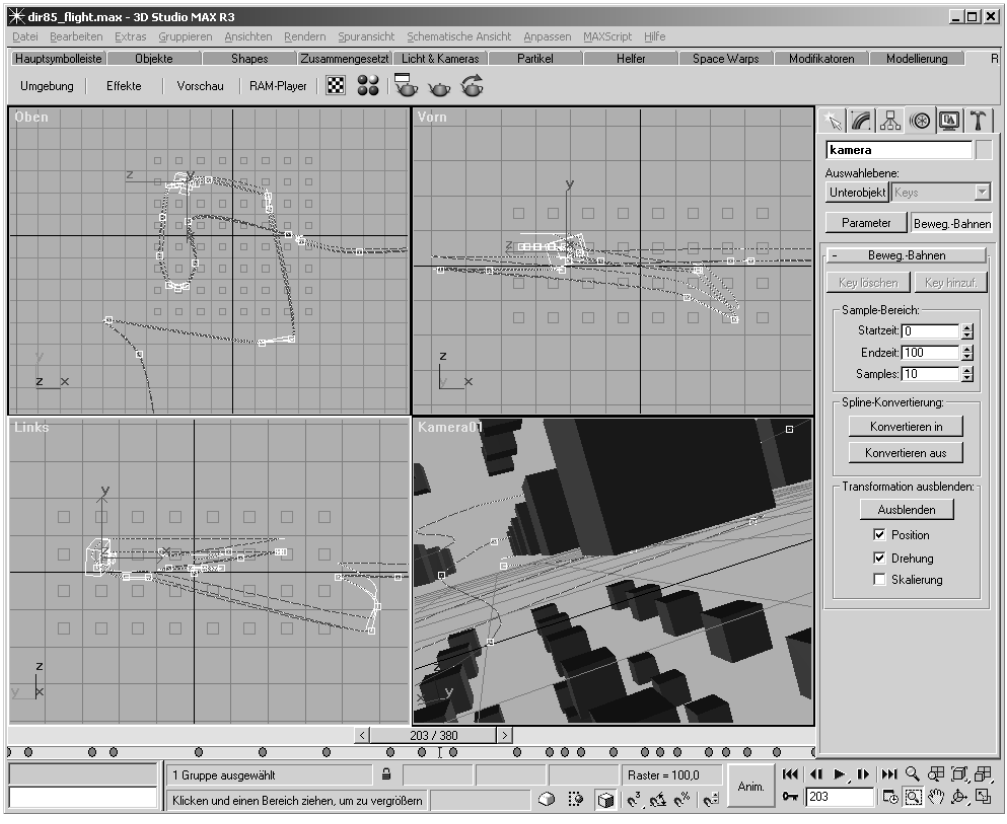


Abbildung 5.26: Die Animation in 3D Studio Max

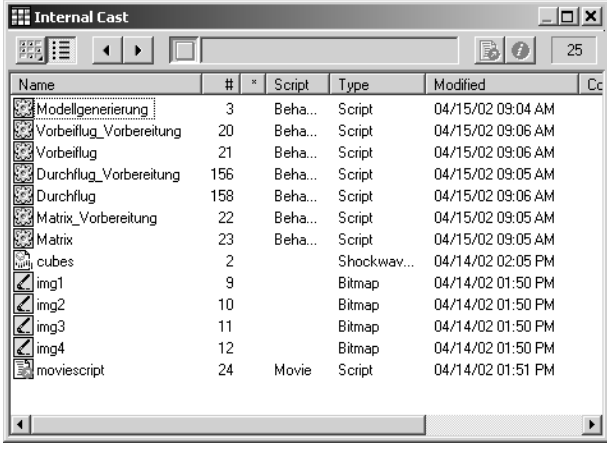
Die exportierte Datei enthält nun nur noch die Daten zu den Lichtquellen und Kamerabewegungen. Sie hat damit noch eine Größe von etwa 5 KB.

▶ Die Erzeugung der Modelldaten

Die Shockwave-Filmdatei *dir85_flight.dir* enthält folgende Darsteller:

- `cubes`: Dies ist die importierte Datei aus 3D Studio Max.
- `moviescript`: Dieses Skript enthält die Funktionen, um die drei Würfel am Anfang flüssig zu bewegen.
- `img1` – `img4`: Dies sind vier beliebige Bilder, die als Texturen für vier spezielle Würfel benutzt werden.
- `Modellgenerierung`: In diesem Skript werden die Würfel und die zugehörigen Texturen erzeugt.
- `Vorbeiflug_Vorbereitung` und `Vorbeiflug`: Hier werden notwendige Initialisierungen für den Vorbeiflug der drei Würfel vorgenommen und durchgeführt.
- `Durchflug_Vorbereitung` und `Durchflug`: Hier wird die Animation für den Durchflug vorbereitet und durchgeführt.

Abbildung 5.27: Die Darsteller des Films



Den Quellcode zur Modellgenerierung erläutert der folgende Abschnitt.

```
global scene, cube

on exitFrame (me)

  sprite(1).visible = false

  set cube = []
  set shaders = []

  scene = member("cubes")
  scene.resetWorld()
```

Zunächst werden die notwendigen Initialisierungen vorgenommen und die Sichtbarkeit des 3D-Sprite ausgeschaltet, damit die Vorbereitungen

nicht zu sehen sind. In den Listen `cube` und `shaders` werden die Würfel und zugehörigen Shader gespeichert.

```
c = scene.camera(2)
sprite(1).addcamera(c)
sprite(1).camera = c
sprite(1).camera.fieldOfView = 26

scene.model("kamera").keyframeplayer.playnext()
```

Da das 3D-Sprite standardmäßig eine andere Kamera anzeigt, muss für die Anzeige der Animation auf die korrekte Kamera umgeschaltet werden.

```
nmr = scene.newModelResource("blockres", #box, #front)
nmr.height = 30
nmr.width = 30
nmr.length = 30
repeat with z=0 to 3
  repeat with y=0 to 7
    repeat with x=0 to 7
      aa = scene.newModel("Quader" & (x+8*y+64*z), nmr)
      aa.transform.preRotate(90,0,0)
      aa.transform.position =
          vector(x*100 - 350, y*100 - 350, z*100 - 150)
    end repeat
  end repeat
end repeat
```

Diese Schleifen erzeugen die Modelle für die Würfel. Die Würfel werden in ihrer Position noch um 90 Grad gedreht, damit die Texturen in der richtigen Position zu sehen sind für diese Animation.

```
gridImg = image(256,256,32,0)
gridImg.fill(0,0,255,255, rgb(0,250,0))
gridImg.fill(1,1,254,254, rgb(50,50,50))
```

Diese Zeilen erzeugen das Bild eines grauen Rechtecks mit einem grünen Rand.

```
shaBox = scene.newShader("boxshader",#standard)
tex =
 scene.newTexture("boxtex",#fromImageObject,gridImg)
shaBox.texture = tex
shaBox.blend = 70
```

Das zuvor erzeugte Bild wird benutzt, um damit einen Shader zu definieren, der den Würfeln zugewiesen wird. Der Shader wird auf „leicht durchsichtig“ gesetzt.

```
  repeat with i = 1 to 4
    sha = scene.newShader("real_shader" & i,#standard)
    tex = scene.newTexture("real_texture" & i,
                  #fromCastMember,member("img" & i))
    sha.texture = tex
    sha.ambient = rgb(188,188,188)
    shaders.append(sha)
  end repeat

  repeat with i = 1 to scene.model.count
    if (chars(scene.model[i].name,1,6) = "Quader") then
      cube.append(scene.model[i])
    end if
  end repeat

  repeat with x = 1 to 256
    cube[x].shaderList = shaBox
    cube[x].visibility = #both
  end repeat

  cube[7].shaderList = shaders[1]
  cube[109].shaderList = shaders[2]
  cube[170].shaderList = shaders[3]
  cube[146].shaderList = shaders[4]

  sprite(1).visible = true

end exitFrame
```

Die restlichen Zeilen der Initialisierung weisen die jeweiligen Shader den Modellen zu und schalten dann die Sichtbarkeit des Sprites wieder ein.

▶ Der Vorbeiflug

Als Nächstes folgt der Vorbeiflug der drei Würfel. Als Vorbereitung dafür werden drei Würfel ausgewählt und weit weg platziert. Das entsprechende Vorbereitungsskript sieht folgendermaßen aus:

```
global scene
global step, c1, c2, c3, c, z1, z2, z3
global motionended
global cube

on exitFrame (me)
  step = 0

  c3 = scene.model("Quader" & (2+0*8+0*64))
  c2 = scene.model("Quader" & (2+0*8+1*64))
  c1 = scene.model("Quader" & (3+0*8+1*64))
  z1 = c1.getworldTransform().position
```

```
    z2 = c2.getworldTransform().position
    z3 = c3.getworldTransform().position

    c1.translate (0,0,10000)
    c2.translate (0,0,10000)
    c3.translate (0,0,10000)
  end exitFrame
```

In den Variablen `c1`, `c2` und `c3` werden Referenzen auf die drei Würfel gespeichert. Bevor die Würfel dann um 10.000 Einheiten auf der z-Achse verschoben werden, werden die aktuellen Positionen in den Variablen `z1`, `z2` und `z3` gesichert.

Die Animation der Würfel selbst wurde nicht vordefiniert, sondern muss programmiert werden.

```
global scene
global step, c1, c2, c3, z1, z2, z3

on exitFrame
  slideto(c3,z3)
  if (step>30) then slideto(c2,z2)
  if (step>60) then slideto(c1,z1)
```

Bei jedem Bilddurchlauf wird die Variable `step` um eins erhöht. Der dritte Würfel fliegt sofort los, die anderen zwei jeweils erst bei Schritt 30 bzw. 60. Die Funktion `slideTo` wird in einem globalen Filmskript definiert, welches weiter unten beschrieben wird.

```
    if (step= 60) then  scene.model("kamera").keyframe-
    player.play("kamera-Key", 0, 5, 1640, 0.45)
```

Bei Schritt 60 wird der erste Teil der Animation gestartet, aber nicht mit der vollen Geschwindigkeit.

```
    step = step + 1
    if (step>=250) then go next

    go to the frame
  end exitFrame
```

Sobald Schritt 250 erreicht ist, springt der Film zur nächsten Marke. Die oben schon erwähnte Funktion `slideTo` ist folgendermaßen aufgebaut:

```
on slideto obj, target
  distanz = target - obj.getworldTransform().position
  if (distanz.length > 80) then
    distanz = distanz.getNormalized() * 80
    obj.translate(distanz, #world)
  else  if (distanz.length > 1) then
    distanz = distanz * 0.2
```

```
    obj.translate(distanz, #world)
  end if
end
```

Übergeben wird an diese Funktion eine Referenz auf ein Modell und eine Zielposition als Vektor. Durch die Subtraktion der aktuellen Position des Modells von der Zielposition erhält man im Vektor `distanz` die Richtung, in die das Modell verschoben werden muss, und den Abstand des Modells vom Zielpunkt. Sollte der Abstand größer als 80 Einheiten sein, wird das Modell um 80 Einheiten verschoben, ansonsten um ein Fünftel der Distanz zum Zielpunkt. Dadurch „gleitet" das Modell in Position.

▶ Der Durchflug

Nachdem die drei Würfel in ihrer Position angekommen sind, verharrt die Kamera noch einen Moment in Position, um die Spannung zu steigern. Danach wird die Animation mit folgendem Skript fortgesetzt:

```
global scene
global motionended

on exitFrame (me)
  motionended = false
  scene.registerForEvent(#animationEnded, #motionend, me)
  scene.model("kamera").keyframeplayer.play("kamera-Key",
                                    0, 1640, 12666, 0.4)
end exitFrame
```

Damit wir mitbekommen, wann die Animation beendet ist, müssen wir allerdings noch eine Funktion definieren, die dann aufgerufen wird. Die Funktion heißt `motionend` und ist definiert im globalen `moviescript`:

```
on motionend
  motionended = true
end
```

Sie macht nichts weiter, als die globale Variable `motionended` auf Wahr zu setzen. In einer Bildschleife muss nun nur noch der Wert dieser Variable abgeprüft werden:

```
global motionended

on exitFrame (me)
  if (not motionended) then go to the frame
end exitFrame
```

Nachdem die finale Position der Kamera erreicht ist, löst sich das Würfelfeld auf, indem die Würfel in zufälliger Reihenfolge in Richtung Kamera fliegen. Als Vorbereitung dafür wird eine Liste mit den Würfelnummern in zufälliger Reihenfolge generiert:

```
global step
global rndlist

on exitFrame me
  step = 0

  repeat with nr=1 to 256
    rndlist[nr] = nr
  end repeat
  repeat with nr=1 to 256
    m = random(256)
    n = random(256)
    dum = rndlist[m]
    rndlist[m] = rndlist[n]
    rndlist[n] = dum
  end repeat
end
```

In der folgenden Bildschleife werden die Würfel dann in dieser Reihenfolge bewegt:

```
global step
global rndlist
global cube

on exitFrame me
  repeat with nr = 1 to 256
    if (step*5 > nr) then
      cube[rndlist[nr]].translate(200,0,0)
    end if
  end repeat

  step = step + 1

  go to the frame
end
```

Durch die Abfrage `if (step*5 > nr)` werden nicht alle Würfel auf einmal bewegt, sondern erst nach und nach alle.

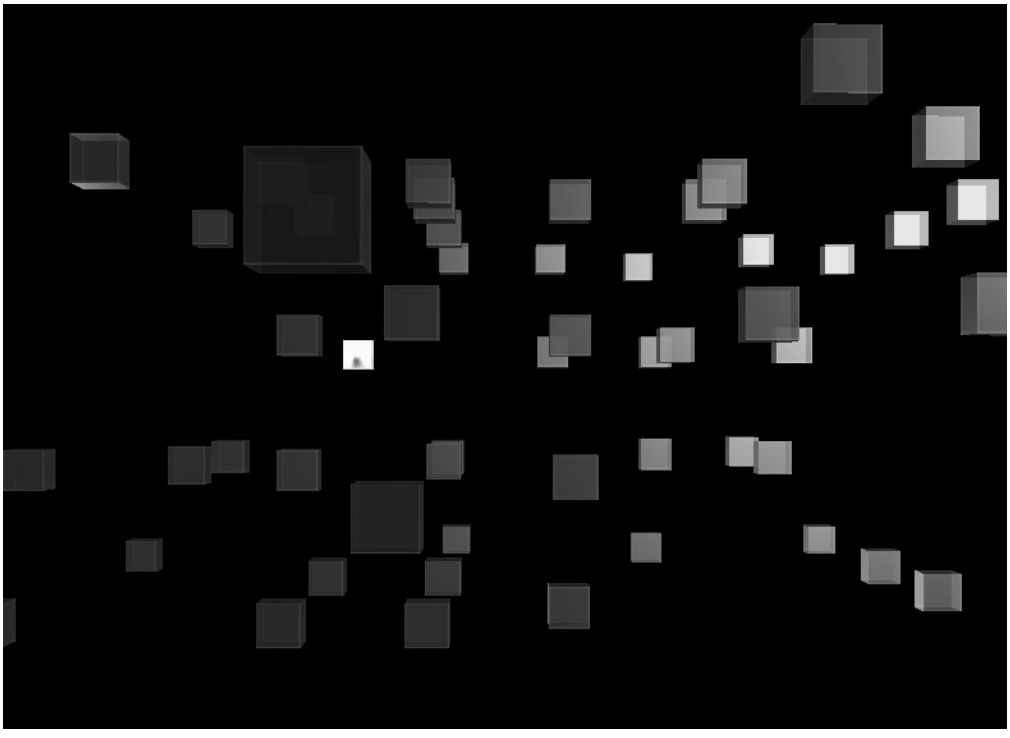
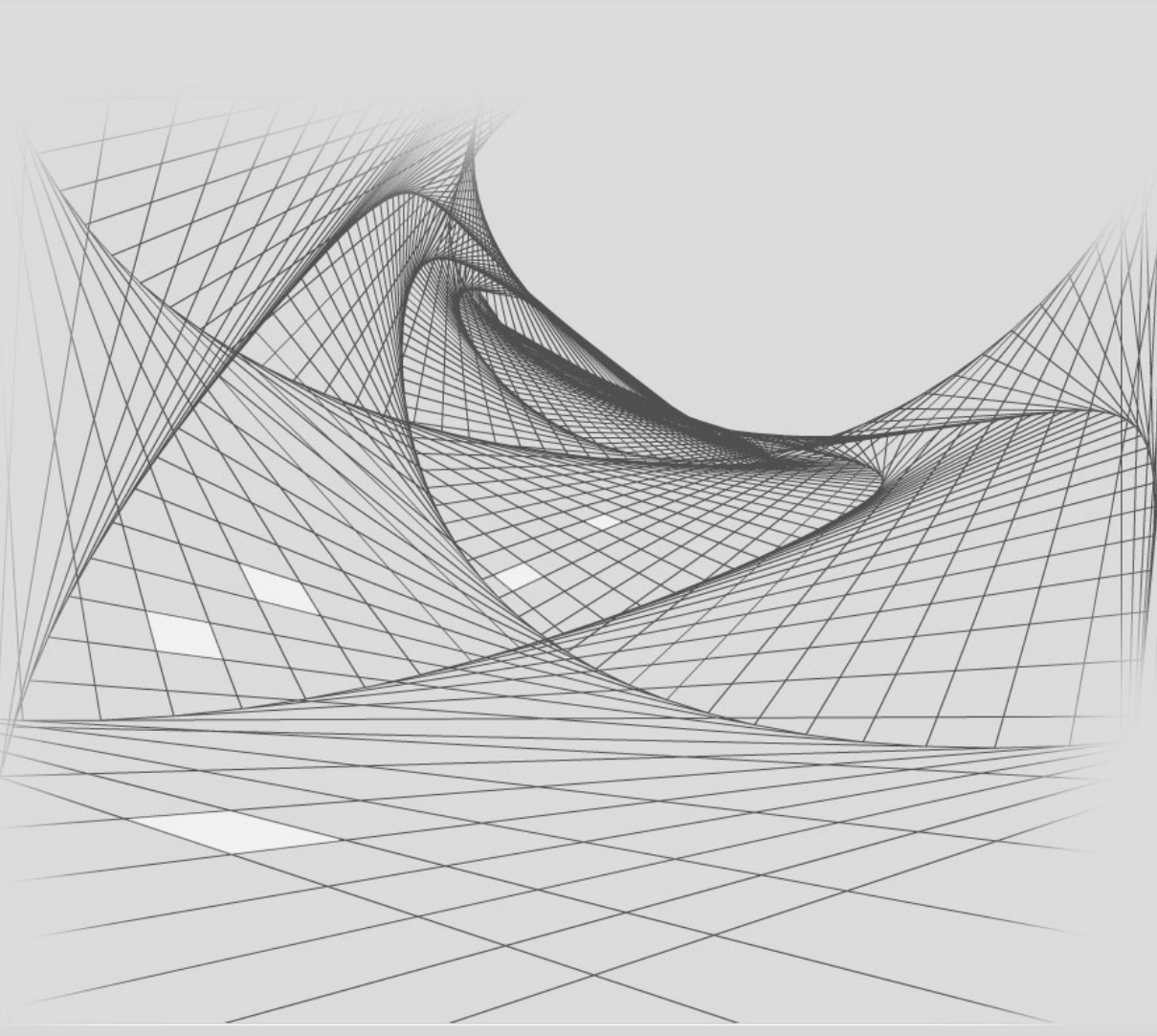


Abbildung 5.28: Finale des Films

Wenn Sie den fertigen Film exportieren, werden Sie sehen, dass die Größe der entsprechenden .dcr-Datei nur etwa 42 KB beträgt. Das ist für die gezeigten Effekte sehr kompakt. Um den Film noch attraktiver zu gestalten, kann man noch mehr Texturen und auch Musik hinzufügen. Diese Bestandteile blähen die Datei natürlich auf. Deshalb ist es hierfür notwendig, abzuwägen welche Bestandteile wirklich notwendig sind. Manche Texturen kann man auch innerhalb des Films mit Lingo erzeugen, so wie das die Beispiele gezeigt haben.

5.4 Zusammenfassung

Die Beispiele in diesem Kapitel zeigen, welch vielfältigen Möglichkeiten Director Shockwave zu bieten hat. Macromedia hat bei diesem Produkt aber zunächst weniger Wert auf eine einfache Bedienung gelegt, sondern auf eine komplette Programmierschnittstelle. Die Erstellung der 3D-Welten wird Programmen überlassen, die darauf spezialisiert sind. Diese Vorgehensweise macht viel Sinn, erfordert für gute Ergebnisse aber sehr viel Einarbeitung.



Java

Neben Flash ist Java das einzige Plugin, welches standardmäßig mit jedem Browser mitgeliefert wird bzw. wurde. Microsoft hat sich entschieden mit der vorinstallierten Version 6 des Internet Explorers unter WindowsXP Java standardmäßig nicht mehr mitzuinstallieren. Das entsprechende Plugin kann jedoch problemlos nachinstalliert werden.

Es bleibt zu hoffen, dass Microsoft diese Entscheidung wieder zurücknimmt, da mit Java eine Plattformunabhängigkeit geboten wird, die kein anderes Plugin vorweisen kann.

In Zukunft könnten Java-Applets von der Verfügbarkeit schlechter dastehen als z.B. Flash.

Was genau leistet Java nun für die 3D-Darstellung von Animationen? Java ist zunächst nur eine mächtige Programmiersprache, die auch einfache Grafikfunktionen bietet. Mit diesen einfachen Grafikfunktionen ist es möglich, ein Applet zu programmieren, welches im Browserfenster dann beliebige Darstellungen zeichnen kann. Die Funktionen zur 3D-Darstellung müssen allerdings im Applet erst noch implementiert werden, um damit 3D-Grafiken zu erzeugen, da die vorhandenen Grundfunktionen nur 2D-Grafiken unterstützen. Es existieren also nur Funktionen zum Zeichnen von Linien, Polygonen etc., aber nicht zum Zeichnen und perspektivischen Darstellen von Würfeln oder anderen 3D-Objekten.

Es gibt allerdings vorgefertigte Bibliotheken, die eine Schnittstelle zur Darstellung von 3D-Grafik anbieten. Eine davon wurde als eine Standarderweiterung zu Java entwickelt. Der Name dieser Erweiterung ist Java3D. Es gibt aber auch andere Anbieter, die eigene Erweiterungen anbieten. Zwei davon sind Anfy3D und 3Danywhere.

6.1 Java3D

Java3D ist eine Programmierschnittstelle, die von Sun als Standarderweiterung zu Java 2 definiert wurde. Von Sun selber gibt es Implementierungen dieses Standards für Windows und Solaris. Es existieren aber auch Implementierungen für die meisten anderen Unix-Betriebssysteme.

Die Implementierungen setzten dabei in der Regel auf eine vorhandene Grafikschnittstelle wie OpenGL oder DirectX auf. Die Darstellungsgeschwindigkeit hängt dann davon ab, ob diese Schnittstellen auf vor-

handene Hardware zugreifen können oder nicht. Damit ist zwar in der Regel eine hohe Darstellungsgeschwindigkeit gewährleistet, allerdings geht die Plattformunabhängigkeit von Java verloren.

Die Installation von Java3D ist u.U. recht kompliziert.

Um Java3D im Browser zu verwenden ist es notwendig, dass der Nutzer das Java 2 Plugin installiert hat. Außerdem muss er die Java3D-Erweiterungen heruntergeladen und installiert haben (ca. 2,6 MB). Diese Punkte erschweren eine problemlose Verwendung von Java3D für Webseiten, die für eine große Anzahl von Nutzern gedacht sind. Für Spezialanwendungen kann sich der Einsatz aber durchaus lohnen.

Java3D-Möglichkeiten

Im Folgenden möchte ich kurz auf die Funktionen und die prinzipielle Verwendung von Java3D eingehen. Eine ausführliche Einleitung würde allerdings den Rahmen dieses Kapitels sprengen. Dafür gibt es eigene Literatur, die sich diesem Thema widmet.

Java3D stellt eine extrem flexible und mächtige Programmierschnittstelle zur Verfügung, um 3D-Grafiken zu erzeugen. Die Funktionen umfassen unter anderem:

- Die gebräuchlichen Renderingmethoden wie Multitexturing, Beleuchtung usw.
- Java-basierte Ereignisse
- Kollisionserkennung
- 3D-Sound
- Vielfältige Eingabegeräteunterstützung
- Komplexe „Views“, zum Beispiel für Virtual-Reality-Anwendungen mit Rundumsicht und dergleichen (diese Funktionen sind für das Web aber eher uninteressant)

Java3D verwendet wie viele andere 3D-Bibliotheken einen Szenengraph.

Die Beschreibung der 3D-Objekte und -Welten erfolgt über so genannte Szenengraphen. Ein Szenengraph ist eine hierarchische Beschreibung einer Szene, welche sowohl die Geometriebeschreibung als auch Transformationen enthalten kann. Man kann sich solch einen Szenengraph als einen Baum vorstellen. Wenn man nun zum Beispiel eine menschliche Figur damit beschreiben möchte, so steht an der Wurzel die komplette Person und Unterknoten davon sind Linker Arm, Rechter Arm, Kopf, Torso, Linkes Bein und Rechtes Bein. Unterhalb von Rechter Arm gibt es dann die Knoten Oberarm, Unterarm und Hand. Unterhalb des Knoten Hand gibt es schließlich die fünf Finger. Wenn man nun den Unterast ab dem Knoten Rechter Arm dreht, drehen sich alle Unterknoten mit. Das Gleiche gilt auch für die Hand oder für die ganze Person, wenn diese vielleicht selbst nur ein Unterast in einem Baum Personengruppe ist.

Solch eine Beschreibung eines zusammengesetzten 3D-Objektes oder einer ganzen Welt kann man direkt in Java programmieren oder man benutzt einen der so genannten Loader, die es für verschiedene Formate gibt.

Die einzelnen Knoten können aber statt Geometriedaten oder Transformationen auch Beschreibungen für das Aussehen oder für Animationen enthalten.

Java3D-Beispiel

Wie sieht nun ein einfaches Beispiel mit Java3D aus? Dazu wollen wir zunächst einfach einen Würfel darstellen. Der Sourcecode dazu sieht wie folgt aus:

Das Listing zu Cube.java

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.ColorCube;
import javax.media.j3d.*;
import javax.vecmath.*;
```

Zunächst werden alle benötigten Bibliotheken importiert. Das Beispiel ist so ausgelegt, dass es als Applet oder als Applikation benutzt werden kann.

```
public class Cube extends Applet {
   public Cube() {
      setLayout(new BorderLayout());
      Canvas3D canvas3D = new Canvas3D(
             SimpleUniverse.getPreferredConfiguration());
      add("Center", canvas3D);
```

Im Konstruktor der Applikation wird zunächst ein spezieller Canvas3D angelegt, auf dem unsere Szene gezeichnet werden kann.

```
      BranchGroup scene = createSceneGraph();

      scene.compile();
      SimpleUniverse universe =
                           new SimpleUniverse(canvas3D);
      universe.getViewingPlatform().
                           setNominalViewingTransform();
      universe.addBranchGraph(scene);
```

Danach werden ein Szenengraph und ein einfaches Universum erzeugt. Der Szenengraph ist hierbei die Datenstruktur, die unsere Szene beschreiben wird. Das Universum stellt ein Fenster oder eine Sicht auf unsere Szene dar.

```
    }

    public BranchGroup createSceneGraph() {
       BranchGroup objRoot = new BranchGroup();
       Transform3D rotate = new Transform3D();
       Transform3D rotate2 = new Transform3D();
       rotate.rotX(Math.PI/4.0d);
       rotate2.rotY(Math.PI/4.0d);
       rotate.mul(rotate2);
       TransformGroup objRotate = new
                                    TransformGroup(rotate);
```

Hier wird nun unser Szenengraph aufgebaut. Dazu erzeugen wir zunächst einen Wurzelknoten `objRoot` als `BranchGroup`. Außerdem erzeugen wir gleich eine Transformation, die als Erstes unter die Wurzel gehängt wird. Dieser Transformationsknoten sorgt dafür, dass der gesamte Ast darunter entsprechend der Angaben rotiert wird. Wie Sie sehen, werden gleich zwei Rotationen erzeugt, eine um die x- und eine um die y-Achse. Man könnte diese zwei Rotationen als separate Knoten untereinander hängen. Es ist aber eleganter, sie zusammenzufassen. Das geschieht durch die Multiplikation `rotate.mul()` (Wie Sie sich vielleicht aus dem Grundlagenkapitel erinnern, werden Transformationen in der Regel als Matrizen dargestellt, die miteinander multipliziert werden können).

```
       objRotate.addChild(new ColorCube(0.3));
       objRoot.addChild(objRotate);
       return objRoot;
    }
```

Hier wird unterhalb des Transformationsknoten noch ein Farbwürfel angehängt, damit wir auch etwas zu sehen bekommen. Abschließend wird dann noch der Ast, der ab dem Transformationsknoten existiert, an die Wurzel des Szenengraphen angehängt, und die Wurzel wird zurückgegeben.

```
    public static void main(String[] args) {
       Frame frame = new MainFrame(new Cube(), 400, 400);
    }

}
```

Ganz zum Schluss wird nun der Hauptrahmen für die Applikation definiert mit einer Größe von 400 mal 400 Pixel.

Das Konzept des Szenengraphen mag auf den ersten Blick etwas verwirrend erscheinen. Es ist aber sehr konsequent und logisch konzipiert, und wenn man es erst richtig verstanden hat, kann man damit die komplexesten Szenen aufbauen. Im nächsten Kapitel zu X3D und VRML finden Sie noch weitere Informationen zum Szenengraph.

Wenn wir nun noch Animation mit in die Szene bringen wollen, müssen wir in den Szenengraph noch einen Behavior-Knoten einfügen. Es gibt verschiedene vordefinierte Behaviors, man kann aber auch beliebige eigene programmieren. Für unser Beispiel werden wir einfach den Würfel um die y-Achse rotieren lassen. Dazu muss die Methode zur Generierung des Szenengraphen wie folgt aussehen:

```
public BranchGroup createSceneGraph() {
   BranchGroup objRoot = new BranchGroup();
   TransformGroup objSpin = new TransformGroup();
   objSpin.setCapability(
               TransformGroup.ALLOW_TRANSFORM_WRITE);
   objRoot.addChild(objSpin);

   objSpin.addChild(new ColorCube(0.3));

   Alpha rotationAlpha = new Alpha(-1, 2000);
   RotationInterpolator rotator =
   new RotationInterpolator(rotationAlpha, objSpin);
   BoundingSphere bounds = new BoundingSphere();
   rotator.setSchedulingBounds(bounds);

   objSpin.addChild(rotator);

   return objRoot;
}
```

Die Erklärung dieses Animationsteils möchte ich an dieser Stelle nicht weiter vertiefen. Dazu sei auf weiterführende Literatur zu diesem Thema verwiesen.

Was an dieser Stelle erreicht werden sollte, war, dass Sie ein Gefühl dafür bekommen, was notwendig ist, um mit Java3D zu programmieren. Wie Sie sehen, sind die Möglichkeiten sehr umfassend und flexibel. Es erfordert aber einiges an Einarbeitung, die durch das umzusetzende Projekt gerechtfertigt sein muss. Auch die tatsächliche Verwendung in einer Webumgebung kann problematisch sein, wenn die Java-Runtime-Umgebung und die Java3D-Erweiterung nicht korrekt aufgesetzt sind.

Für kleinere 3D-Anwendungen sind in der Regel andere Techniken sinnvoller. Zwei davon sollen nun im Anschluss erläutert werden.

6.2 Shout3D

Shout3D ist ein Player für 3D-Inhalte auf der Basis eines Java-Applets. Sie können damit beliebige 3D-Szenen in einem 3D-Editor wie zum Beispiel 3D Studio Max erstellen, als VRML 97 exportieren und mit dem Shout3D Applet auf einer Webseite anzeigen. Da das Applet auf Java 1.1 beruht, kann es mit Browsern ab der Version 4 angezeigt werden.

Es werden einige Basisfunktionen zur Verfügung gestellt, um ohne Programmieraufwand Szenen mit oder ohne Animation anzuzeigen. Es ist aber auch möglich, die vorhandenen Klassen zu erweitern, um beliebige Interaktivität zu gestalten. Diese Erweiterungen können sogar mit JavaScript in der HTML-Seite vorgenommen werden.

Shout3D ist für Testzwecke frei verfügbar unter *www.shout3d.com*, für eine kommerzielle Verwendung fallen jedoch Lizenzgebühren an. In der Testversion wird allerdings immer ein Schriftzug mit dem Namen von Shout3d angezeigt.

Ein einfaches Beispiel

Wie sieht nun die Arbeit mit Shout3D aus? Dazu wollen wir zunächst wieder mal unser Würfelbeispiel heranziehen. Zum Nachvollziehen dieses Beispiels sollten Sie sich die neueste Version von Shout3D von deren Website geholt und installiert haben. Die in diesem Beispiel benutzten Dateien finden Sie wie üblich auf der CD im entsprechenden Verzeichnis.

Erzeugen Sie als Erstes mit einem 3D-Editor einen einfachen Würfel, weisen Sie diesem eine Textur zu und speichern Sie das Ganze im VRML 97-Format als *cube.wrl* ab. Wenn Sie für die Erstellung 3D Studio Max verwenden, sieht das Exportfenster wie in Abbildung 6.1 aus. Sie können in der Regel die vorgewählten Einstellungen einfach so belassen. Die exportierte Datei können Sie anschließend mit einem einfachen Texteditor wie zum Beispiel Notepad öffnen und bearbeiten. Die Datei sollte in etwa wie folgt aussehen:

Das Listing zu models/cube.wrl

```
#VRML V2.0 utf8

DEF cube Transform {
  translation 0.0 0.0 -250
  rotation -1 0 0 -1.571
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0.2667 0.1922 0.5451
          ambientIntensity 0.1115
          specularColor 0.045 0.045 0.045
          shininess 0.2875
          transparency 0
        }
        texture ImageTexture {
          url "textur.gif"
        }
      }
      geometry DEF cube-FACES IndexedFaceSet {
        ccw TRUE
        solid TRUE
        coord DEF cube-COORD Coordinate {
               point [-100 0 100, 100 0 100, -100 0 -100,
                       100 0 -100, -100 200 100,
                       100 200 100, -100 200 -100,
                       100 200 -100]
        }
        texCoord DEF cube-TEXCOORD TextureCoordinate {
               point [0 0, 1 0, 0 1, 1 1, 0 0, 1 0, 0 1,
                      1 1, 0 0, 1 0, 0 1, 1 1]
        }
        coordIndex [0, 2, 3, -1, 3, 1, 0, -1, 4, 5, 7,
                    -1, 7, 6, 4, -1, 0, 1, 5, -1, 5, 4,
                    0, -1, 1, 3, 7, -1, 7, 5, 1, -1, 3,
                       2, 6, -1, 6, 7, 3, -1, 2, 0, 4, -1,
                       4, 6, 2, -1]
        texCoordIndex [9, 11, 10, -1, 10, 8, 9, -1, 8, 9,
                       11, -1, 11, 10, 8, -1, 4, 5, 7,
                       -1, 7, 6, 4, -1, 0, 1, 3, -1, 3,
                       2, 0, -1, 4, 5, 7, -1, 7, 6, 4,
                       -1, 0, 1, 3, -1, 3, 2, 0, -1]
        }
    }
  ]
}
```

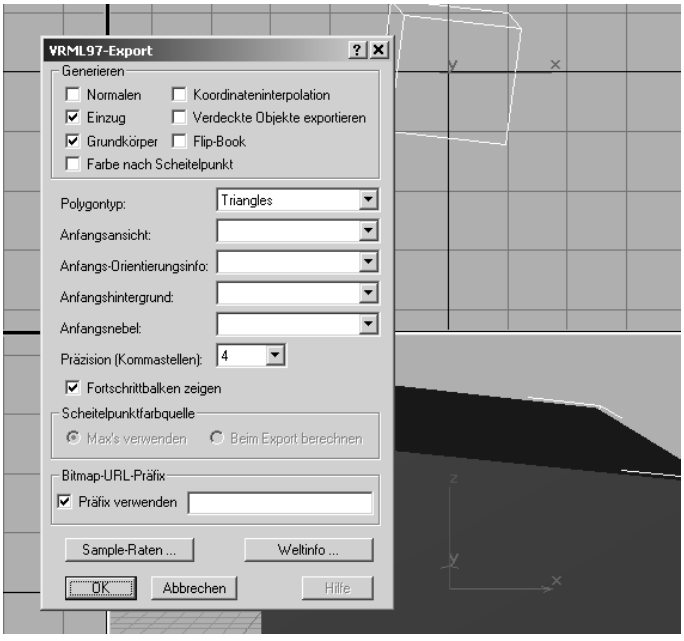


Abbildung 6.1: VRML-Exportfenster in 3D Studio Max

Auf den Aufbau derartiger Dateien wird im Kapitel VRML/X3D noch näher eingegangen, deshalb möchte ich das an dieser Stelle nicht weiter vertiefen. Stellen Sie nur bitte sicher, dass die URL für die Textur in der Datei keine Pfadangaben enthält. Sie können ggf. die Datei problemlos mit dem Texteditor verändern.

Kopieren Sie nun die Datei *cube.wrl* in das Unterverzeichnis *Shout-3d_runtime\codebase\models* Ihrer Shout3D-Installation. Kopieren Sie auch die Texturdatei *textur.gif* in das gleiche Verzeichnis. Die Textur kann dabei ein beliebiges GIF-Bild sein.

Nun benötigen wir nur noch eine HTML-Seite, die das Ganze anzeigt. Diese kann wie folgt aussehen:

```
<HTML>
<HEAD>
<TITLE>Shout3D Demo</TITLE>
</HEAD>
<BODY>
   <APPLET CODEBASE="..\codebase"
           CODE="applets/ExamineApplet.class"
           ARCHIVE="shout3dClasses.zip" WIDTH=600
           HEIGHT=400>
      <param name="src" value="models/cube.wrl">
      <param name="headlightOn" value="true">
   </APPLET>
</BODY>
</HTML>
```

Die Seite enthält im Wesentlichen nur den Code für das Applet. Der Parameter `CODEBASE` gibt hierbei an, relativ zu welchem Pfad nach den Dateien für die Klassendefinitionen und Modelldaten gesucht werden soll. Im Parameter `CODE` wird angegeben, dass hier das `ExamineApplet` benutzt werden soll. Dieses Applet erlaubt die Darstellung beliebiger Szenen und das Drehen der Szene mit der Maus. Der Parameter `ARCHIVE` gibt das Archiv mit den Shout3D-Klassen an. Der Parameter `SRC` gibt den Pfad zu der VRML-Datei an (auch relativ zur Codebase). Da in der VRML-Datei keine Beleuchtung definiert ist, wird in einem weiteren Parameter noch eine einfache Beleuchtung eingeschaltet. Kopieren Sie diese HTML-Seite am besten in das Unterverzeichnis *Shout3d_runtime\demos* Ihrer Shout3D-Installation.

Wenn alles korrekt ist, sollte das Beispiel wie in Abbildung 6.2 aussehen.

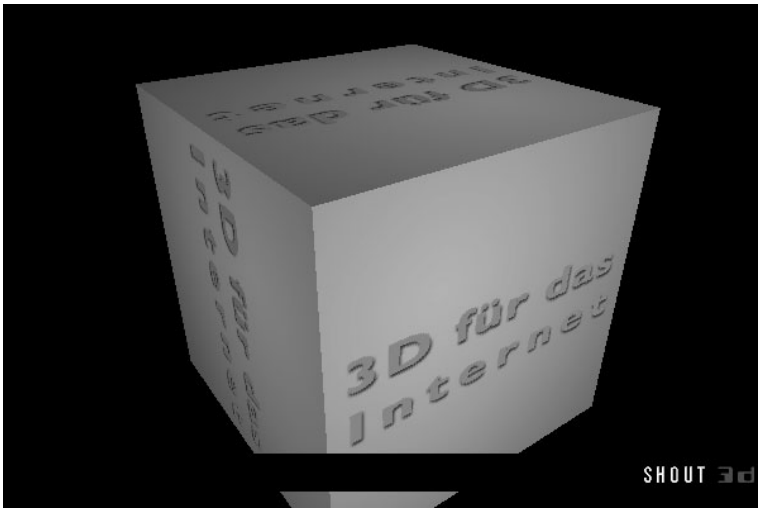


Abbildung 6.2: Ein einfaches Beispiel mit Shout3D

Der Shout3D-Wizard

Für eine einfache Anwendung von Shout3D gibt es auch einen Wizard, der einem ein paar Teile der Aufgaben abnimmt. Die üblichen Schritte bei der Erstellung einer Seite mit Shout3D sind:

- Erstellung einer 3D-Szene

Die Szene können Sie mit jedem beliebigen Editor erstellen, der VRML 97 (manchmal auch VRML 2.0 genannt) exportieren kann. Bei der Erstellung der Szene können auch gleich die entsprechenden Animationen, Kameraansichten und Lichtquellen erstellt werden. Bei den Animationen können Sie auch mehrere Teilanimationen erstellen, auf die später per Programmierung zugegriffen werden kann.

- Umwandlung/Export der Szene in das passende Format

Für 3D Studio Max gibt es neben der Möglichkeit des Exports ins VRML-Format auch die Möglichkeit, direkt das von Shout erstellte Export-Plugin zu benutzen. Wie Sie dies installieren, entnehmen Sie bitte der Shout-Dokumentation. Dieses Plugin kann dann direkt Dateien im Shout3D-Format s3d erstellen. Dieses entspricht in wesentlichen Punkten dem VRML-Format, enthält aber einige eigene Erweiterungen. Außerdem gibt es dieses Format auch noch in komprimierter Form s3z. Da das eigentliche Format immer textdateibasiert ist, kann durch eine Komprimierung viel Platz gespart werden. Die Umwandlung von VRML in s3d oder s3z kann aber auch immer mit dem Wizard von Shout3D erfolgen, der im Folgenden besprochen werden soll.

- Anpassungen vornehmen, wie zum Beispiel interaktive Funktionalitäten programmieren

Ein Vorteil des VRML- oder s3d-Formats ist, dass das Ganze in einer normalen Textdatei vorliegt und man von Hand viel anpassen und optimieren kann. Die interaktiven Funktionalitäten kann man durch eine Erweiterung des Java-Applets vornehmen oder mit JavaScript in der HTML-Seite programmieren.

- Publizieren auf dem Server

Bei diesem Schritt kann wieder der Wizard von Shout3D hilfreich sein. Wichtig ist, alle notwendigen Dateien, wie Szenendateien, Texturen und die Klassendateien, auf den Server zu kopieren. Die Klassendateien können dabei auch zusammengefasst und komprimiert werden. Außerdem bietet es sich evtl. an, nicht benötigte Klassendateien zu löschen, um so die Ladezeiten zu verringern.

Sehen wir uns als Nächstes einmal den Shout3D Wizard an. Sie finden ihn im Unterverzeichnis shout3d_wizard Ihrer Shout3D-Installation. Nach dem Start der Anwendung wird das Fenster in Abbildung 6.3 angezeigt.

Abbildung 6.3: Der Shout3D-Wizard

Hier öffnen Sie zunächst über das Menü File die VRML-Datei, die Sie mit Ihrem 3D-Editor erstellt haben. Wenn Sie nun Änderungen an den Publish-Optionen vornehmen, können Sie mittels Preview immer die gegenwärtigen Einstellungen überprüfen. Die Publish-Optionen sehen Sie in Abbildung 6.4:

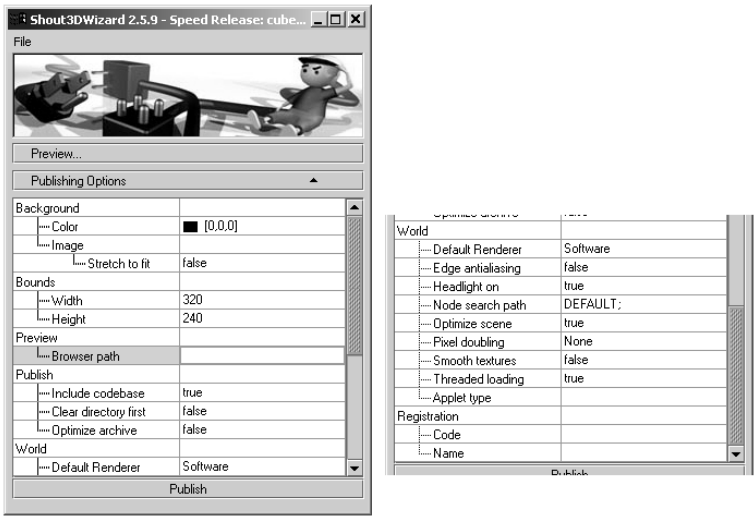


Abbildung 6.4: Die Publish-Optionen des Wizards

Wenn Sie auf den Button Publish klicken, werden alle benötigten Dateien in das Unterverzeichnis published kopiert. Im Einzelnen können Sie folgende Publish-Einstellungen vornehmen:

- Background

Color: Hier können Sie über Schieberegler die Hintergrundfarbe des Applets einstellen.

Image: Statt einer Farbfläche können Sie auch ein Bild in den Hintergrund legen und dieses automatisch auf die richtige Größe skalieren lassen (Stretch to fit).

- Bounds

Hier kann die Größe des Applets auf der HTML-Seite festgelegt werden. Je größer das Applet ist, desto langsamer wird natürlich die Darstellung.

- Preview

Hier können Sie ggf. einen alternativen Browser für das Preview angeben.

- Publish

Include codebase: Setzen Sie diesen Wert auf false, wenn Sie nur die entsprechende HTML-Datei erzeugen möchten, ohne die benötigten Javaklassen zu kopieren.

Clear directory first: Wenn dieser Wert aus true steht, wird vor dem Publizieren immer erst das publish-Verzeichnis geleert.

Optimize archive: Wenn dieser Wert auf true steht, wird speziell für dieses Applet ein Archiv erzeugt, welches nur die Klassen enthält, welche auch tatsächlich benötigt werden.

- World

Default Renderer: Der Standardrenderer ist normalerweise der Softwarerenderer. Es besteht aber die Möglichkeit für den Browser ein Plugin zu installieren, welches es Shout3D ermöglicht, auf eine vorhandene OpenGL-Schnittstelle zuzugreifen, die ihrerseits vorhandene Hardwarebeschleunigung benutzen kann.

Edge Antialiasing: Durch diese Option können die Kanten der Modelle, die über dem Hintergrund liegen (nicht aber wenn sie über anderen Modellteilen liegen) mit Antialiasing gerendert werden. Dadurch wird die ganze Darstellung aber langsamer. Es besteht außerdem die Möglichkeit, in der Szenendatei progressives Antialiasing einzuschalten, welches immer dann, wenn keine Bewegung in der Szene stattfindet, die gesamte Szene mit Antialiasing rendert, wodurch die Darstellungsqualität erheblich verbessert wird.

Headlight on: Hiermit wird ein Lichtquelle beim Betrachter eingeschaltet. Dadurch können Modelle getestet werden, ohne extra eine Lichtquelle definieren zu müssen.

Node search path: Hier können zusätzliche Verzeichnisse für Knoten-Klassen angegeben werden. Dieser Punkt ist interessant, wenn Sie eigene Knoten-Klassen für die Verwendung in den Szenedateien verwenden möchten.

Pixel doubling: Hiermit ist es möglich, das Bild mit nur einem Viertel der eingestellten Auflösung zu berechnen und die einzelnen Pixel dann in doppelter Größe darstellen zu lassen. Der einzige Sinn dieser Einstellung ist, die Berechnungsgeschwindigkeit zu steigern.

Smooth textures: Diese Option schaltet bilineare Filterung für Texturen ein, wodurch die Geschwindigkeit sinkt und die Darstellungsqualität gesteigert wird.

Threaded loading: Durch Einschalten dieser Option werden alle benötigten Ressourcen gleichzeitig vom Server geladen. Das Laden geht dadurch nicht schneller, aber die Szene wird ggf. schon angezeigt, auch wenn noch nicht alle Texturen geladen sind.

Applet type: Hier können Sie das zu verwendende Applet auswählen. In dem vorhergehenden Beispiel haben wir ExamineApplet.class benutzt, welches ein einfaches Drehen der angezeigten Szene ermöglicht.

- Registration

Hier können Sie die Registrierungsdaten ändern, wenn Sie Shout3D registriert haben.

Ein einfaches Auswahlsystem mit Shout3D

Als Nächstes wollen wir ein etwas komplexeres Beispiel entwickeln. Das Beispiel stellt ein kleines Auswahlsystem dar, bei dem mit 3D Studio Max die Modelle erstellt werden, Animationspfade definiert werden und mit JavaScript dann die Teilanimationen gesteuert werden. Das Ganze sieht dann so aus wie in Abbildung 6.5.

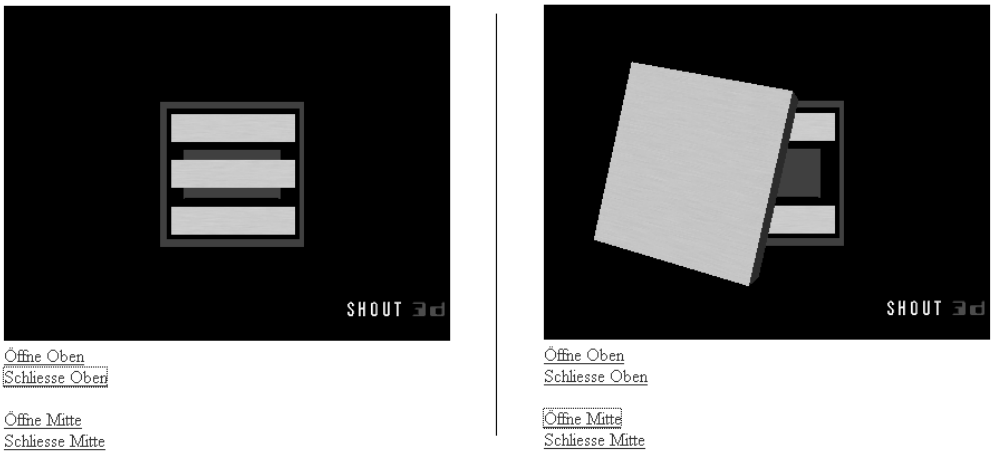


Abbildung 6.5: Auswahlsystem mit JavaScript-Interaktion

Fangen wir zunächst mit der Erstellung der Szene und der Animationen in 3D Studio Max an. Hier wurden drei Quader erzeugt und `BoxOben`, `BoxMitte` und `BoxUnten` benannt. Für die umgebende Hülle wurde mittels einer booleschen Operation aus einem Quader ein zweiter kleinerer ausgeschnitten. Außerdem wurden eine einfache Punktlichtquelle und Kamera eingesetzt.

Die Erstellung der Animationen

Die Animationslänge habe ich auf 30 Frames und die Geschwindigkeit auf 30 Frames pro Sekunde gesetzt. Danach habe ich eine Animationssequenz für die obere Box mit vier Keyframes erzeugt:

- In Frame 0 ist die Box in der Ausgangslage
- In Frame 9 ist die Box in Richtung des Betrachters verschoben und leicht nach vorne gekippt
- In Frame 19 ist die Box dem Betrachter komplett zugewandt
- In Frame 29 ist die Box wieder in der Ausgangslage

Für die mittlere Box gilt die gleiche Abfolge nur mit leicht anderen Positionen. Die untere Box wird in diesem Beispiel nicht benutzt.

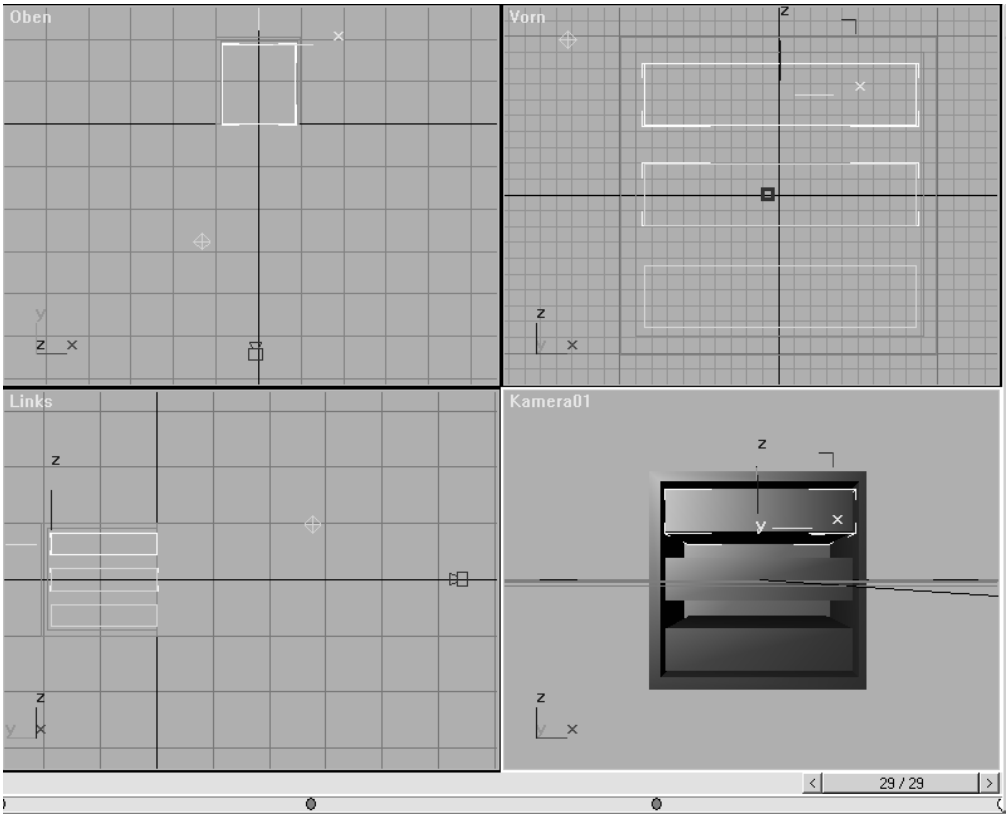


Abbildung 6.6: ***Erstellung der Szene in 3D Studio Max***

Die Szene wird nun als VRML 97 exportiert. Bei den Optionen für den Export muss die Sample-Rate noch jeweils auf 3 FPS (Frames pro Sekunde) eingestellt sein, damit nicht zu viele Zwischenschritte erzeugt werden, die die weitere Bearbeitung unübersichtlich machen. Wenn Sie sich das Beispiel ansehen, werden Sie allerdings merken, dass die Boxen sich teilweise durch Wände bewegen. Aber das nehmen wir an dieser Stelle im Sinne der Übersichtlichkeit in Kauf.

Manuelle Nacharbeiten an der exportierten Datei

Wenn die Datei nun exportiert ist, sind noch ein paar kleine Änderungen daran notwendig. Die fertige Datei finden Sie auf der CD im Verzeichnis Java\Shout3D\models unter dem Namen cube3.wrl. In der Datei wurde wie bei den anderen Beispielen der Name der Texturen ersetzt. Außerdem wurde der String `loop TRUE` durch `loop FALSE startTime -1` ersetzt. Dadurch werden die Animationen nicht immer wiederholt, sondern laufen nur einmal durch. Zum anderen starten die Animationen zunächst gar nicht, da die Startzeit auf –1 gesetzt ist.

Nun können wir den Wizard verwenden, um uns eine HTML-Seite zu erzeugen. Starten Sie den Wizard und öffnen Sie damit die Datei *cube3.wrl*. Bei den publish-Optionen ist wichtig, dass Sie unter World „Optimize scene“ ausschalten und als „Applet type“ Shout3Dapplet an-

geben (dieses befindet sich nicht im Applet-Unterverzeichnis, sondern im Shout3D-Unterverzeichnis). Klicken Sie abschließend auf Publish.

Wenden wir uns der HTML-Datei zu. Öffnen Sie die generierte HTML-Datei in einem Editor. Öffnen Sie gleichzeitig auch die Datei *cube3.wrl*, da wir daraus noch ein paar Werte übernehmen müssen.

Die HTML-Datei sollte folgendermaßen aussehen bis jetzt:

```
<HTML>
<HEAD>
<TITLE>Autogenerated HTML : cube3</TITLE>
</HEAD>
<BODY>

<APPLET CODEBASE="." CODE="shout3d/Shout3DApplet.class"
        ARCHIVE="shout3dClasses.zip" WIDTH=400
        HEIGHT=300>
<param name="src" value="cube3/cube3.WRL">
<param name="headlightOn" value="true">
<param name="regcode" value="">
<param name="regname" value="">
<param name="antiAliasingEnabled" value="false">
<param name="bilinearFiltering" value="false">
<param name="loadResourcesInSeparateThread" value="true">
</APPLET>

</BODY>
</HTML>
```

Erzeugen wir in dieser Datei zunächst die Buttons, die die Animationen steuern sollen. Dazu fügen wir nach dem Applet-Tag den folgenden Code ein:

```
<br>
<a href="javascript: oeffneOben()">
&Ouml;ffne Oben</a><br>
<a href="javascript: schliesseOben()">
Schliesse Oben</a><br>
<br>
<a href="javascript: oeffneMitte()">
&Ouml;ffne Mitte</a><br>
<a href="javascript: schliesseMitte()">
Schliesse Mitte</a><br>
```

Dies sind einfache Links, die keine andere Seite öffnen, sondern jeweils eine JavaScript-Funktion ausführen.

Wenn Sie sich mit dem Wizard die Szene im Preview angesehen haben, werden Sie bemerkt haben, dass in der Ausgangsszene die Boxen ungesteuert durcheinander fliegen. Das heißt, dass wir die vordefinierten Animationen ändern müssen. Das haben wir erst mal dadurch schon gemacht, dass wir ein Abspielen der Animationen zunächst ganz unter-

bunden haben (durch das Setzen der Startzeit auf –1). Mit Shout3D ist es möglich, auf fast alle Elemente der Szene zuzugreifen und sie zu ändern. Die Animationen in der VRML-Datei sehen folgendermaßen aus:

```
...
  DEF BoxOben-TIMER TimeSensor { loop FALSE startTime -1
                                 cycleInterval 0.9667 },
  DEF BoxOben-POS-INTERP PositionInterpolator {
    key [0, 0.3448, 0.6897, 1, ]
    keyValue [0.3884 62.68 0, 4.41 53 211.1,
              40.88 -81.75 127.4,
              0.8648 63.27 -188.5, ] },
  DEF BoxOben-ROT-INTERP OrientationInterpolator {
    key [0, 0.3448, 0.6897, 1, ]
    keyValue [-1 0 0 -1.571, -1 0 0 -2.334,
              -1 0 0 -3.259, 1 0 0 -1.571, ] },
...
```

Diese Zeilen beschreiben die Animation für die obere Box. Die erste Definition (`DEF BoxOben-TIMER`) beschreibt den Timer der Animation. Die zweite Definition (`DEF BoxOben-POS-INTERP`) beschreibt die Änderung der Position. Die Liste `key` gibt dabei die Zeitpositionen in Sekunden an. Die Liste `keyValue` gibt die Position in x-,y- und z-Koordinaten zu dem jeweiligen Zeitpunkt aus der Liste key an. Das Gleiche gilt für die nächste Definition (`DEF BoxOben-ROT-INTERP`) der Rotationsangaben, wobei die Rotationen jeweils durch vier Werte angegeben werden: Die ersten drei Werte bestimmen die Drehachse und der vierte Wert den Drehwinkel im Bogenmaß.

Übernahme der Animationsdaten

Diese drei Definitionen müssen wir nun mit JavaScript anpassen, damit für das Öffnen einer Box nur die ersten drei Keyframes abgespielt werden und für das Schließen dann nur die letzten beiden. Der Code für das Öffnen sieht also folgendermaßen aus:

```
function oeffneOben() {
   var q1pos = document.Shout3D.getNodeByName(
                                  "BoxOben-POS-INTERP");
   q1pos.key.setValueByString("0 0.3448 0.6897");
   q1pos.keyValue.setValueByString(
     "0.3884 62.68 0 4.41 53 211.1 40.88 -81.75 127.4");
    var q1rot = document.Shout3D.getNodeByName(
                                  "BoxOben-ROT-INTERP");
   q1rot.key.setValueByString("0 0.3448 0.6897");
   q1rot.keyValue.setValueByString(
           "-1 0 0 -1.571 -1 0 0 -2.334 -1 0 0 -3.259");
    var stimer = document.Shout3D.getNodeByName(
                                       "BoxOben-TIMER");
   s_timer.restart();

}
```

In der Variablen `q1pos` wird hier eine Referenz auf den Knoten mit der Definition für die Animation der Position gespeichert. Anschließend können die Werte für die Keyframe-Zeitpunkte und -Positionen neu gesetzt werden. Das Gleiche wird für die Rotationen gemacht. Abschließend wird dann der Timer der Animation neu gestartet, damit die Animation auch abgespielt wird.

Der Code für das Schließen sieht ganz entsprechend aus:

```
function schliesseOben() {
   var q1pos = document.Shout3D.getNodeByName(
                                    "BoxOben-POS-INTERP");
   q1pos.key.setValueByString("0 0.3448");
   q1pos.keyValue.setValueByString(
               "40.88 -81.75 127.4 0.8648 63.27 -188.5");
    var q1rot = document.Shout3D.getNodeByName(
                                    "BoxOben-ROT-INTERP");
   q1rot.key.setValueByString("0 0.3448");
   q1rot.keyValue.setValueByString(
                          " -1 0 0 -3.259 1 0 0 -1.571");
    var s_timer = document.Shout3D.getNodeByName(
                                         "BoxOben-TIMER");
   s_timer.restart();
}
```

Hier werden eben dann nur die Werte für die letzten beiden Keyframes gesetzt.

Der entsprechende Code wird nun auch für die mittlere Box umgesetzt und schon funktioniert das Ganze, wie im Beispiel zu sehen ist. Man kann das Beispiel natürlich noch so erweitern, dass man direkt auf die Boxen selbst klickt, um die Animationen zu starten.

Wie Sie gesehen haben, bietet Shout3D sehr viele komplexe Möglichkeiten zur Darstellung von 3D-Grafiken. Oftmals ist aber auch komplexe Programmierung notwendig, um das gewünschte Ziel zu erreichen. Es ist aber schon verblüffend, wie viel Funktionalität die Entwickler in die recht kompakten Javaklassen gepackt haben. Trotzdem ist es für den Besucher einer Site notwendig, immer den gesamten Code zusammen mit der eigentlichen Szene herunterzuladen.

6.3 Weitere Javatechniken

Anfy3D

Anfy ist eine Sammlung von Java-Applets, die im Wesentlichen von dem Italiener Fabio Cuicci entwickelt wurden. Eines davon ist Anfy3D, welches eine Reihe von 3D-Funktionen zur Verfügung stellt, um damit 3D-Grafiken und -Animationen auf Webseiten zu bringen.

Anfy3D ist Shout3D sehr ähnlich, allerdings ist es etwas weniger kommerziell aufgemacht. Das bedeutet zum einen, dass es nicht so komfortabel und vollständig ist (zum Beispiel bezüglich Dokumentation). Zum anderen ist es aber auch günstiger.

In Abbildung 6.7 und Abbildung 6.8 sehen Sie Beispiele, die vom Anfy-Team selbst erstellt wurden, um die Möglichkeiten zu zeigen.

Abbildung 6.7: Das Anfy3D-Logo

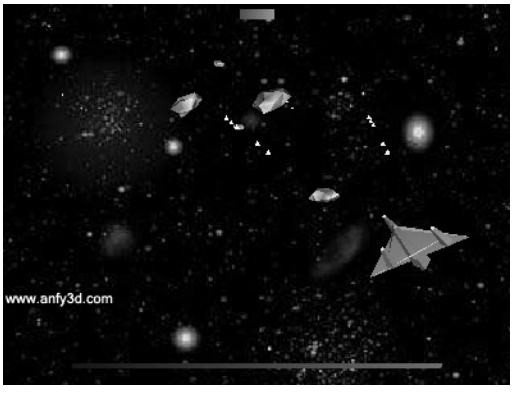


Abbildung 6.8: Ein Spiel, umgesetzt mit Anfy3D

Für Anfy3D gibt es auch wie für Shout3D einen Wizard, der den Import von Szenen und die Erstellung der passenden HTML-Datei erleichtert (siehe Abbildung 6.9).

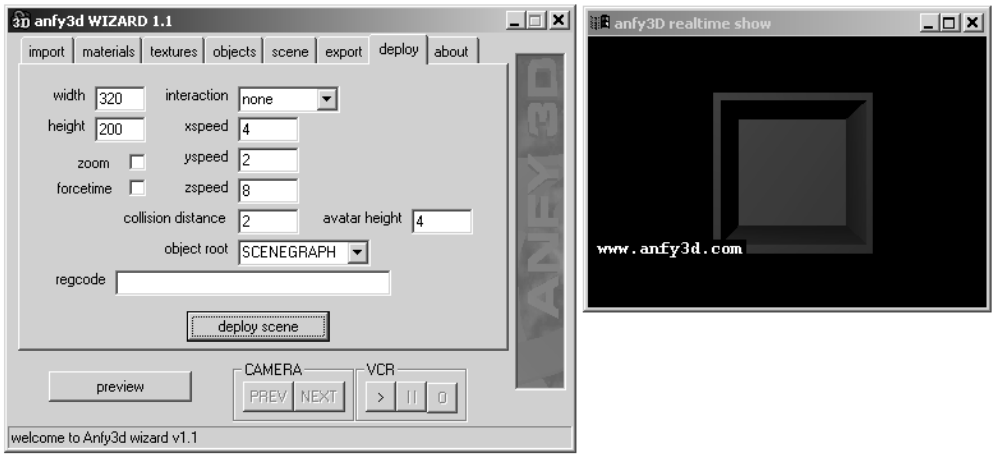


Abbildung 6.9: Der Anfy3D-Wizard

Dadurch, dass Anfy3D nicht so viele Funktionen bietet, sind die benötigten Javaklassen, die immer mit heruntergeladen werden müssen, für die Anzeige kleiner als bei Shout3D.

3DAnywhere

3Danywhere fällt in die gleiche Kategorie wie die zuvor beschriebenen Produkte. Es handelt sich auch hierbei um eine Sammlung von Java-Klassen, die es ermöglichen, mit einem Applet 3D-Grafiken auf eine Webseite zu bringen. Vom Funktionsumfang ist es den anderen Produkten auch sehr ähnlich. In Abbildung 6.10 und Abbildung 6.11 sehen Sie zwei Beispiele, die von der Firma 3Di selbst erstellt wurden.

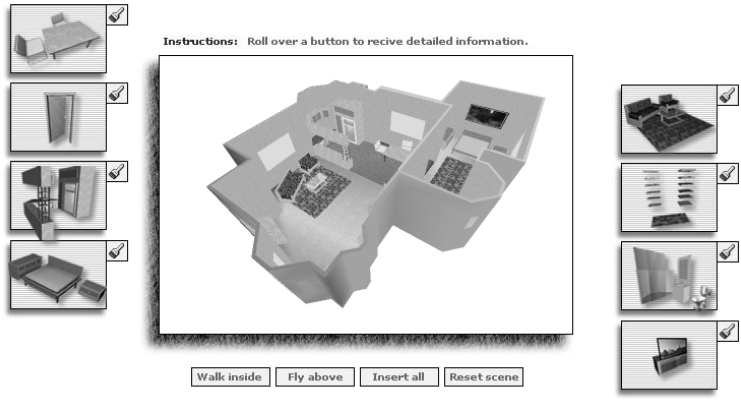


Abbildung 6.10: Beispiel einer Architektursimulation

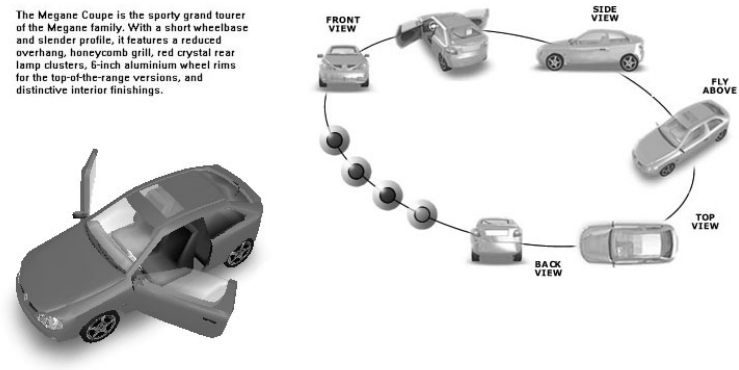


Abbildung 6.11: Produktpräsentation eines Autos

Ein wesentlicher Unterschied zu den beiden vorher beschriebenen Technologien ist der mitgelieferte Designer bei 3DAnywhere. Im Gegensatz zu den Wizards von Shout3D und Anfy3D bietet der Designer sehr viel weitergehende Möglichkeiten. Er ersetzt keinen 3D-Editor. Man muss also auch hier VRML-97-Dateien importieren. Anschließend kann man allerdings sehr viele Einstellungen daran vornehmen und vor allem auch Animationen definieren, ohne eine Zeile programmieren zu müssen (siehe Abbildung 6.12). Somit wäre es zum Beispiel möglich gewesen, für das Beispiel im Shout3D-Kapitel, bei dem wir die Animationen selbst auseinander nehmen mussten, diese in 3D-Studio einfach hintereinander zu definieren und dann mit dem Designer entsprechend zuzuweisen.

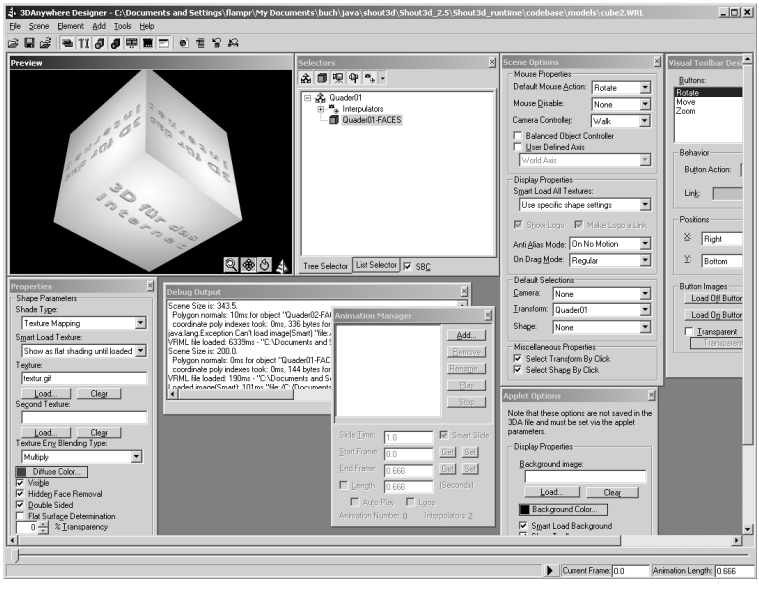


Abbildung 6.12: Der 3Danywhere-Designer

Trotz des Funktionsumfangs ist 3DAnywhere nicht sehr teuer. Für nichtkommerziellen Gebrauch fallen keine Gebühren an und für den kommerziellen Gebrauch nur ca. 50 USD pro Lizenz. Die genauen Konditionen sollten Sie aber der entsprechenden Webseite entnehmen.

6.4 Literaturhinweise und URLs

Java3D	*java.sun.com/products/java-media/3D/*	Website
	www.j3d.org	Community-Site
Shout3D	*www.shout3d.com*	Website
	Interactive Web Graphics with Shout3D ISBN 0782128602	Offizielles Buch zu Shout3D
	www.shout3d.com/wwwboard/index.html	Messageboard
Anfy3D	*anfyteam.com/panfy3d.html*	Website
	groups.yahoo.com/group/anfy3d-api/	Messageboard
3Danywhere	*www.3danywhere.com*	Website

6.5 Zusammenfassung

Java ist vor allem dann eine gute Wahl, wenn es auf Plattformunabhängigkeit ankommt. Dafür gibt es kaum etwas Besseres. Problematisch kann hier nur werden, wenn Microsoft weiter dagegen vorgeht und man in Zukunft nicht mehr davon ausgehen kann, dass auf Windows-Rechnern das Java-Plugin vorhanden ist. Wenn es erst heruntergeladen und installiert werden muss, ist man unter Umständen mit einem speziellen 3D-Plugin besser bedient.

Was vor allem nicht vergessen werden darf: Java-Applets auf Webseiten werden genauso behandelt wie andere Plugins bzgl. Sicherheit. Wenn innerhalb eines Firmennetzwerkes also keine Plugins erlaubt sind, funktionieren auch Java-Applets nicht.

X3D und VRML

Nachdem wir bisher einige proprietäre Technologien kennen gelernt haben, wird es Zeit, dass wir über Standards sprechen. Vielleicht haben Sie sich auch schon gewundert, wann die Sprache endlich auf VRML kommt. Schließlich sind bei den meisten Personen, die sich schon länger mit dem Web beschäftigen, die Begriffe Web 3D und VRML eng miteinander verbunden. Allerdings werden auch viele sagen, wenn sie den Begriff VRML hören, „Ach, das war doch dieser am meisten überbewertete Standard vor ein paar Jahren, der nie richtig gelebt hat, aber auch nie richtig tot zu kriegen war...“. VRML (ausgesprochen wird es „vermal“ in englischer Aussprache) ist die Abkürzung von Virtual Reality Markup Language und war der erste Versuch einer Standardisierung von 3D-Formaten für das Web.

VRML lebt weiter als „X3D“.

Nun, ich möchte hier nicht darüber diskutieren, was die Probleme von VRML waren. Was hier besprochen werden soll, ist die aktuelle Standardisierung von 3D für das Web, genannt X3D, und wie das mit VRML zusammenhängt. Um das besser zu verstehen, betrachten wir zunächst einen kleinen geschichtlichen Überblick:

- Die Version 1.0 von VRML wurde 1995 spezifiziert. Sie basierte zu großen Teilen auf dem in OpenInventor benutzten Format von SiliconGraphics.
- 1996 wird das VRML-Konsortium gegründet und reicht VRML 2.0 bei der ISO zur Standardisierung ein.
- 1997 wurde von der ISO VRML 97 als offizieller Standard verabschiedet. Dieser Standard ist seitdem stabil und wird von einem Großteil der professionellen 3D-Grafiksoftware unterstützt, z.B. bieten fast alle 3D-Grafikeditoren den Import/Export im VRML 97-Format an.
- 1998 allerdings verlaufen einige Initiativen zur Verbreitung von 3D im Web im Sande, SGI gibt ihre Cosmo-Implementierung auf und auch Microsoft beendet eigene Pläne in Richtung 3D.
- 1999 regruppiert sich das VRML-Konsortium als die Web3D-Gruppe.
- 2000 formt sich die X3D-Arbeitsgruppe.
- Im März 2002 wird der erste Entwurf von X3D freigegeben, welcher im August standardisiert werden soll.

Während all dieser Aktionen haben sich die proprietären Technologien weiterentwickelt, die wir zuvor alle besprochen haben. Diese haben teilweise zwar VRML als Grundlage benutzt, aber richtig durchgesetzt hat sich VRML im Web nicht, vor allem auch, weil der sehr komplexe Standard nie in die Grundausstattung der Standardbrowser aufgenommen wurde.

X3D wird hoffentlich eine bessere Zukunft haben, wofür aber auch viele Punkte sprechen:

- Die Zeit ist „reifer“. Inzwischen ist die Hardware der meisten PCs fähig, 3D-Grafik in ordentlicher Geschwindigkeit darzustellen.
- X3D ist voll abwärtskompatibel zu VRML 97. Das bedeutet zum einen, dass Dateien leicht von einem Format in das andere überführt werden können, und zum anderen kann jeder X3D-kompatible Browser direkt VRML-97 lesen.
- X3D gibt es in mehreren Ausprägungen, wodurch es einfacher wird, sich an den Teil des Standards zu halten, der den eigenen Bedürfnissen am nächsten kommt.
- X3D basiert auf XML und ist somit leichter lesbar für eine große Anzahl von Personen und Werkzeugen.

Auf den ersten Punkt brauche ich nicht näher einzugehen, sonst würden Sie dieses Buch nicht in den Händen halten. Die anderen Punkte möchte ich im Folgenden ausführlicher beleuchten.

7.1 Wie sieht X3D aus?

In den vorhergehenden Kapiteln haben wir teilweise schon VRML-Dateien benutzt und auch direkt editiert. An dieser Stelle möchte ich noch mal zum direkten Vergleich ein einfaches HelloWorld-Beispiel zeigen, einmal in VRML 97-Format und einmal im X3D-Format.

Die Beispiele finden Sie auf der CD im Verzeichnis X3D.

Das Beispiel stellt einen Würfel dar und davor den Schriftzug „Hello World!“ (siehe Abbildung 7.1).

Abbildung 7.1: Ein einfaches Beispiel in VRML und X3D

VRML-Dateien haben immer die Endung .wrl wie „World“ oder .wrz, wenn die Datei komprimiert ist. In komprimierter Fassung kann man die Datei nicht mehr mit einem beliebigen Texteditor ansehen und bearbeiten, ansonsten schon.

In VRML 97 sieht der Quellcode zu obigem Beispiel folgendermaßen aus:

Das Listing zu HelloWorld.wrl

```
#VRML V2.0 utf8

Group {
  children [
    Viewpoint {
      description "hello world!"
      orientation 0 0 1 0
      position 0 0 10
    }
    NavigationInfo {
      type [ "EXAMINE" "ANY" ]
    }
    Shape {
      geometry DEF B Box {
      }
      appearance Appearance {
          texture ImageTexture {
             url [ "textur.gif" ]
          }
      }
    }
    Transform {
      translation -2.4 -0.3 2
      children [
          Shape {
            geometry Text {
              string [ "Hello world!" ]
            }
            appearance Appearance {
              material Material {
                diffuseColor 0.1 0.5 1
              }
            }
          }
      ]
    }
  ]
}
```

Es werden hier nur zwei Grundkörper definiert, eine Box und ein Schriftzug, und dem Ganzen wird ein Aussehen gegeben.

Im X3D-Format sieht der Code entsprechend aus:

Das Listing zu HelloWorld.x3d

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "http://www.web3D.org/TaskGroups/x3d
/translation/x3d-compact.dtd" "/www.web3d.org/TaskGroups
/x3d/translation/x3d-compact.dtd">
<X3D>
  <head>
    <component name="helloComponent"/>
    <meta content="HelloWorld.x3d" name="filename"/>
    <meta content="Simple X3D example"
          name="description"/>
  </head>
  <Scene>
    <Group>
      <Viewpoint description="hello world!"
                 orientation="0 0 1 0"
                 position="0 0 10"/>
      <NavigationInfo type="EXAMINE ANY"/>
      <Shape>
        <Box DEF="B"/>
        <Appearance>
          <ImageTexture url="&quot;textur.gif&quot;"/>
        </Appearance>
      </Shape>
      <Transform translation="-2.4 -0.3 2">
        <Shape>
          <Text string="&quot;Hello world!&quot;"/>
          <Appearance>
            <Material diffuseColor="0.1 0.5 1"/>
          </Appearance>
        </Shape>
      </Transform>
    </Group>
  </Scene>
</X3D>
```

Der wesentliche Unterschied, wie Sie sehen, ist tatsächlich nur, dass die X3D-Datei im XML-Format vorliegt und teilweise andere Namen benutzt. Schon aus diesem Grunde wird es gleich nach Verabschiedung des Standards eine große Anzahl von Szenen, Modellen und Werkzeugen geben, die damit arbeiten können, da sich die Daten leicht konvertieren lassen.

Die X3D-Anweisungen im Einzelnen

Gehen wir etwas genauer auf den Aufbau der Datei ein. Ganz oben sehen Sie einige dokumentenspezifische Angaben, die den Dokumententyp und eine DTD (Document Type Definition, siehe unten bei XML) angeben.

Danach fängt die eigentliche Datei mit dem <X3D>-Tag an:

Im <HEAD>-Tag stehen einige Metainformationen über Name, Inhalt und dergleichen.

Die Datei kann mehrere Szenen enthalten, welche jeweils in einem ‹SCENE›-Tag stehen.

X3D beruht wie VRML auch auf einem Szenengraph. Das heißt, alle Szeneninformationen wie Geometriedaten, Aussehen, Animationen und Interaktionen sind in einer hierarchischen Baumstruktur angeordnet. Innerhalb dieser Struktur gibt es verschiedene Knoten, die sich jeweils auf die unter ihnen liegenden Äste beziehen. Es gibt unter anderem folgende Knoten:

- Group: Ein Group-Knoten gruppiert im Wesentlichen nur die darunter liegenden Äste zu einer Gruppe zusammen.
- Shape: Ein Shape-Knoten beschreibt ein Objekt und hat immer zwei Felder, einen Appearance-Knoten und einen Geometry-Knoten. Der Appearance-Knoten beschreibt das Aussehen oder Verhalten der Oberfläche des Objektes, zum Beispiel, wie stark es glänzt und welche Textur es besitzt. Der Geometry-Knoten beschreibt die Geometriedaten des Objektes, also aus welchen Eckpunkten und Flächen sich das Objekt zusammensetzt.
- Transform: Der Transform-Knoten beschreibt eine Transformation aller untergeordneten Äste relativ zu darüber liegenden Transformationen. Das heißt, wenn ein Transformations-Knoten eine Verschiebung durchführt, so werden nachfolgende Transformationen immer relativ zu dem verschobenen Punkt durchgeführt.

Dies ist nur eine kleine Auswahl der verfügbaren Knoten. Es gibt noch sehr viele andere und neue können beliebig hinzugefügt werden.

7.2 X3D-Grundlagen

Zunächst einmal steht X3D für „Extensible 3D“, also erweiterbares 3D. Dieses Merkmal ist auch die wesentliche Neuerung gegenüber VRML. Wie wir ja schon gesehen haben, ist das Format ansonsten hauptsächlich eine andere Notationsform von VRML.

Zusammensetzung/Erweiterbarkeit

Das X in X3D

Was macht X3D nun so erweiterbar? Die wichtigen Begriffe, die in diesem Zusammenhang immer fallen werden, sind Profile und Komponenten. Es war erkannt worden, dass nicht alle Firmen immer alle Merkmale eines Standards umsetzen wollten, wenn es ihnen nur auf einen kleinen Teil der Funktionalitäten ankam. Beispielsweise möchte eine Firma einen Browser für einfache Animationen anbieten, aber nicht Funktionen umsetzen, die etwa für die Anzeige geographischer Daten notwendig sind. Deshalb wurden die Bestandteile der Spezifikationen in mehrere Komponenten zerlegt.

Komponenten

Die Grundkomponenten sind:

- Core: Definiert die grundlegenden Funktionalitäten, die alle Profile benötigen
- Time: Definiert die Knoten, welche zeitbasierte Funktionalitäten zur Verfügung stellen
- Aggregation and Transformation: Definiert die Organisations- und Aggregations-Knoten zur Unterstützung der Hierarchie des Szenengraphen
- Geometric Properties: Definiert die notwendigen Knoten zur Spezifikation der Basiseigenschaften der Geometrieknoten
- Geometry: Definiert die verschiedenen Formen der sichtbaren Geometrie-Knoten
- Appearance: Definiert die verschiedenen Knoten, die das Aussehen der Geometrie und Umgebung bestimmen
- Lighting: Definiert die Knoten für die Beleuchtung
- Navigation: Definiert die Knoten, die die Bewegung eines Avatars durch die X3D-Welt unterstützen
- Interpolation: Definiert die Knoten zur Unterstützung der Änderung verschiedener Parameter durch Interpolation
- Text: Bietet Unterstützung zur Darstellung von Text in einer X3D-Welt
- Sound: Bietet Unterstützung für das Abspielen von Geräuschen in 3D-Szenen
- Pointing Device: Definiert die Knoten zur Unterstützung der Auswahl von Objekten durch eine Maus oder andere Eingabegeräte
- Environmental Sensor: Definiert die Knoten zur Unterstützung der Erkennung von Beziehungen innerhalb der Szene
- Texturing: Definiert die Knoten zur Anwendung von Texturen
- Prototyping: Definiert die Funktionalitäten, die zur Erweiterung der Standardfunktionalitäten notwendig sind
- Scripting: Definiert die für Scripting benötigte Funktionalität

Weitere vorgeschlagene Komponenten sind „Key device sensor“, „Geospatial“, „H-Anim“, „DIS“, „Nurbs“, „Multitexture“ und andere. Für jede Komponenten gibt es auch noch so etwas wie Leistungsstufen (Level). Verschiedene Komponenten werden nun in Profilen zusammengefasst.

Die Grundprofile sind zunächst:

Profile

- Interchange: Dieses Profil definiert das minimale Subset von X3D, welches zum Austausch von Geometrie- und Animationsdaten benötigt wird. Wenn also ein 3D-Editor X3D lesen können soll, müssen sich die Entwickler nur an die Funktionalitäten dieses Profils halten. Dieses Profil enthält die Komponenten
  - Core
  - Time
  - Aggregation and transformation
  - Geometric properties
  - Geometry
  - Appearance
  - Lighting
  - Navigation
  - Interpolation
  - Texturing
- Interactive: Dieses Profil definiert das minimale Subset von X3D, welches für interaktive Inhalte benötigt wird. Es enthält zusätzlich zum Profil Interchange die Komponenten
  - Pointing device sensor
  - Environmental sensor
- Extensibility: Dieses Profil definiert das minimale Subset von X3D, welches für erweiterte Komponenten und integrierte Applikationen benötigt wird. Zusätzlich zum Profil Interactive enthält dieses Profil die Komponenten
  - Sound
  - Prototyping
  - Scripting
  - Key device sensor
- VRML97 BASE: Dieses Profil bildet den gesamten Umfang von VRML97 ab.
- Full: Dieses Profil bildet den gesamten Umfang der X3D-Spezifikation ab.

Allerdings bilden die Basisprofile nicht jeweils den höchsten Level einer Komponente ab. Das bedeutet zum Beispiel, dass in Level 1 der Komponente Geometry kein ElevationGrid enthalten ist.

Durch die Zerlegung in Komponenten und Profile bleibt der Standard offen und dynamisch und läuft nicht so schnell Gefahr, in einer Sackgasse zu enden, wenn die technologische Entwicklung voranschreitet. Wenn Firmen neue Komponenten aufgenommen haben möchten, können diese beim X3D-Board eingereicht werden. Das Gleiche gilt auch für die Definition von neuen Profilen.

Knoten

Im vorhergehenden Abschnitt war häufig die Rede davon, dass eine Komponente irgendwelche Knoten beschreibt, und auch in dem X3D-Beispiel weiter oben wurden Knoten benutzt.

Alles besteht aus Knoten.

Die Knoten (eng.: Nodes) sind die grundlegenden Datenelemente, die alle Details der 3D-Welt beschreiben. Die Knoten selbst sind in einem Szenengraph aufgebaut, der weiter unten beschrieben wird. Die Komponente Geometry enthält zum Beispiel die folgenden Knoten:

- Box: Ein einfacher Quader
- Cone: Ein Kegel
- Cylinder: Ein Rohr
- ElevationGrid: Ein Feld von Höhenangaben, z.B. zur Erzeugung von Landkarten
- Extrusion: Ein Extrusion einer zweidimensionalen Form, z.B. um 3D-Text zu erzeugen
- IndexedFaceSet: Eine zusammenhängende Menge von Polygonen
- IndexedLineSet: Eine zusammenhängende Menge von Linien
- PointSet: Eine zusammenhängende Menge von Punkten
- Shape: Eine Zusammenfassung eines Geometry-Knotens mit einem Appearance-Knoten
- Sphere: Eine Kugel

Andere Knoten beschreiben Material, Lichtquellen, Transformationen oder Ereignisse.

Szenengraph

Die Datenstruktur, mit welcher in X3D eine 3D-Welt beschrieben wird, ist der Szenengraph. Dieser Szenengraph ist eine hierarchische Anordnung der oben beschriebenen Knoten.

Auf der CD ist neben der Beispiel X3D-Datei dieselbe Datei noch mal mit der Endung XML gespeichert. Wenn Sie diese in einem Internet Explorer ab Version 5 öffnen, zeigt er Ihnen die Datei in einer Baumstruktur an (siehe Abbildung 7.2).

```
<?xml version="1.0" encoding="UTF-8" ?>
- <X3D>
  + <head>
  - <Scene>
    - <Group>
        <Viewpoint description="hello world!" orientation="0 0 1 0" position="0 0 10" />
        <NavigationInfo type="EXAMINE ANY" />
      - <Shape>
          <Box DEF="B" />
        + <Appearance>
        </Shape>
      - <Transform translation="-2.4 -0.3 2">
        - <Shape>
            <Text string="'"Hello world!"'" />
          + <Appearance>
          </Shape>
        </Transform>
      </Group>
    </Scene>
  </X3D>
```

Die Datei HelloWorld.xml im Internet Explorer

Abbildung 7.2: Die XML-Datei im Internet Explorer

Daran können Sie schon recht gut den enthaltenen Szenengraph erkennen. Als Graph gezeichnet sähe das Ganze so aus, wie in Abbildung 7.3 zu sehen ist.

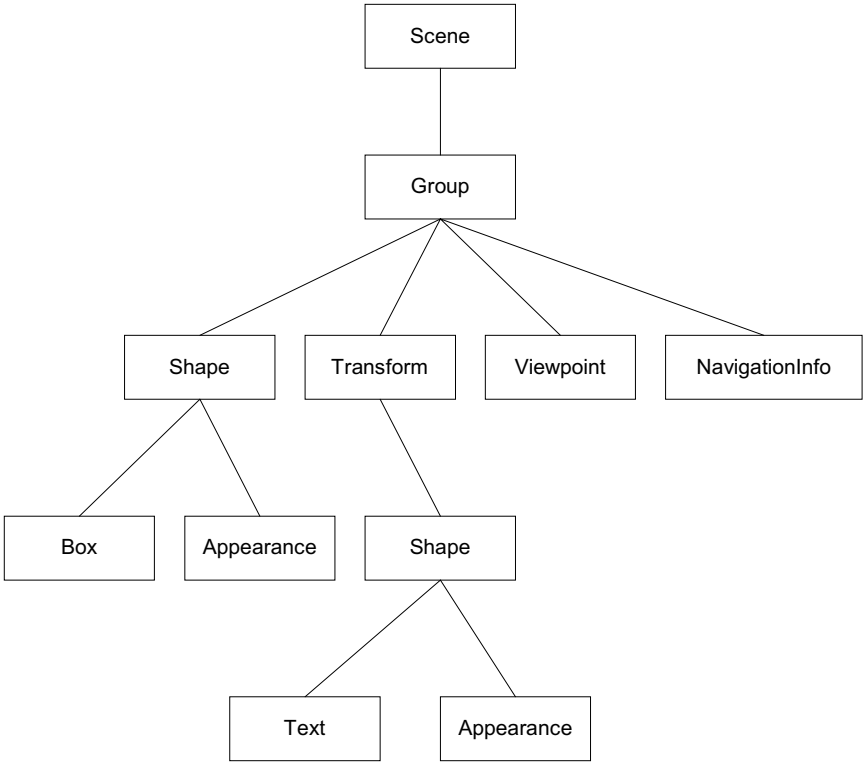


Abbildung 7.3: Beispiel Szenengraph

Wenn nun ein Element des Szenengraph gezeichnet wird, muss zum Beispiel der Renderer zunächst den Graph von dem Knoten bis zur Wurzel laufen und alle Transformationen auf dem Pfad auf das Objekt anwenden. Dies bewirkt, dass man zum Beispiel die Kamera versetzen oder die ganze Szene transformieren kann und sich diese Transformation auf alle darunter liegenden Äste auswirkt.

7.3 X3D und XML

X3D ist zwar eng verwandt mit VRML97 und ein X3D-Browser muss auch in der Lage sein, eine VRML97-Datei direkt zu verarbeiten, das Standardformat ist jedoch XML. Das heißt, dass X3D-Dateien immer im XML-Format definiert sind.

Es gibt auch eine eigene DTD (Document Type Definition), die für X3D-Dateien benutzt wird. Diese ist zu finden unter *http://www.web3D.org/TaskGroups/x3d/translation/x3d-compact.dtd*.

Die Verwendung von XML bietet diverse Vorteile:

- XML ist ein weithin anerkannter und etablierter Standard.
- Durch die große Verbreitung von XML gibt es sehr viele Werkzeuge, die bei der Erstellung, Validierung und Verarbeitung von XML-Dateien hilfreich sind.
- Während VRML97 nur für Insider wirklich „lesbar“ war, ist ein XML-Format zumindest syntaktisch leicht verständlich.
- X3D reiht sich damit ein in eine Reihe mit anderen Standards wie XHTML, SIML und SVG.

Für X3D gibt es auch einen Editor, der bei der Eingabe/Erstellung von Dateien unterstützt.

7.4 Zusammenfassung

X3D ist ein noch nicht abgeschlossener, aber schon viel versprechender Standard. Tatsächlich lebt er in Grundzügen als VRML ja schon seit vielen Jahren und hat sich dadurch bewährt. Vor allem im Bereich des Datenaustauschs ist zu erwarten, dass X3D sich durchsetzen wird. Aber natürlich hoffen wir darauf, dass bald jeder Browser, ohne ein Plugin herunterladen zu müssen, zumindest das Interactive-Profil unterstützt. Spätestens dann werden real berechnete 3D-Grafiken und -Animationen zum Alltag im Web gehören.

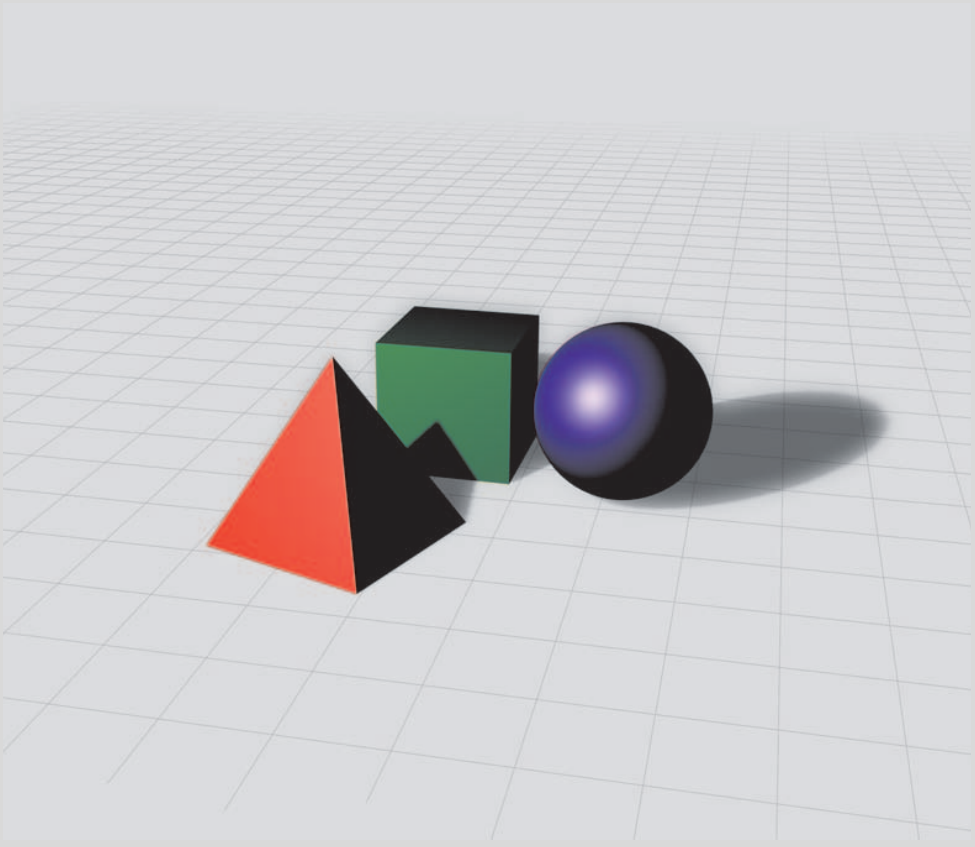
Index

X

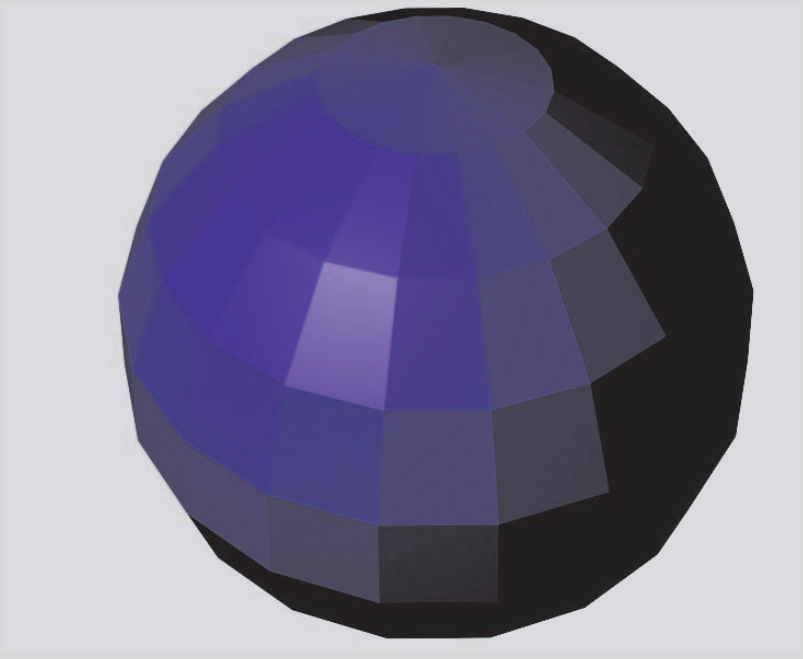
Z

Farbteil

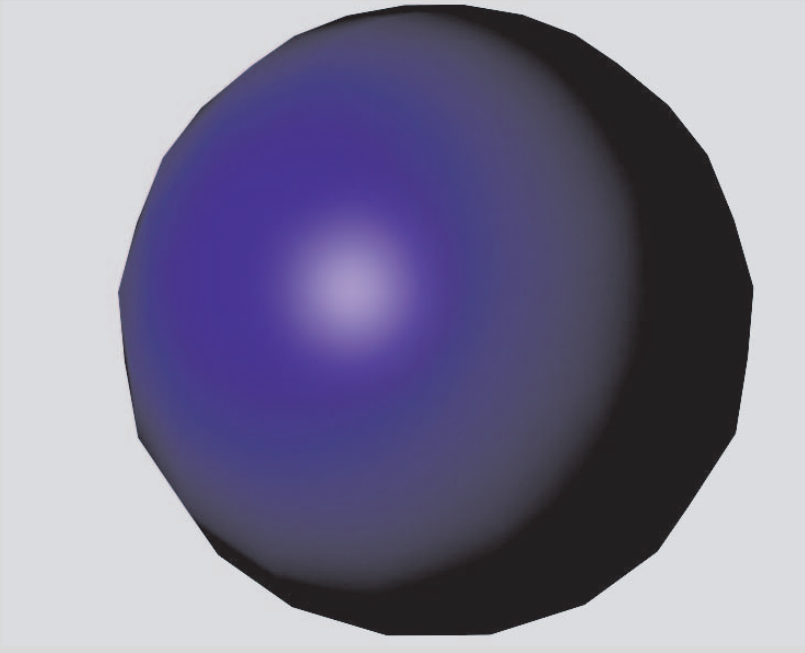
Grundlagen



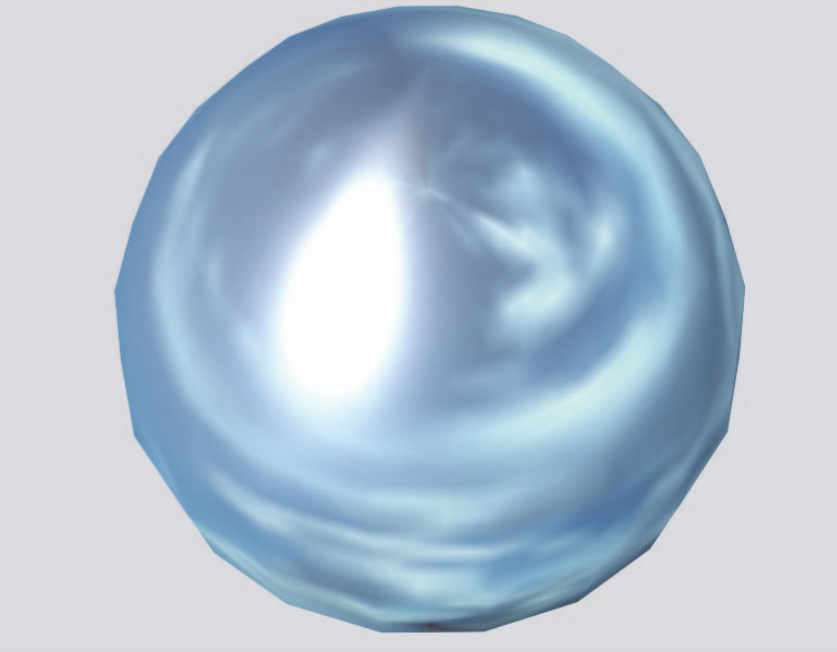
Einfache 3D-Darstellung



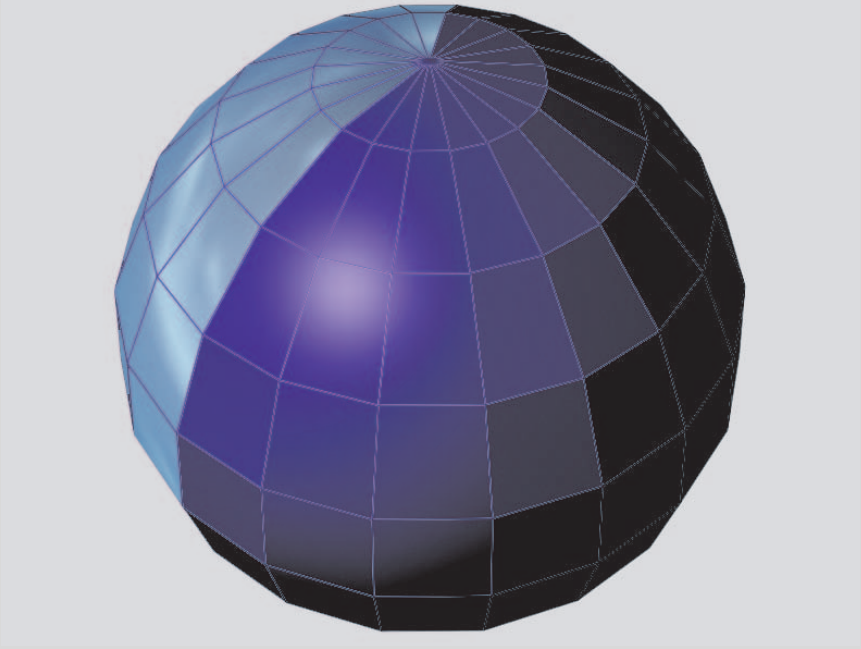
Flach schattierte Seitenflächen



Phong-schattierte Seitenflächen



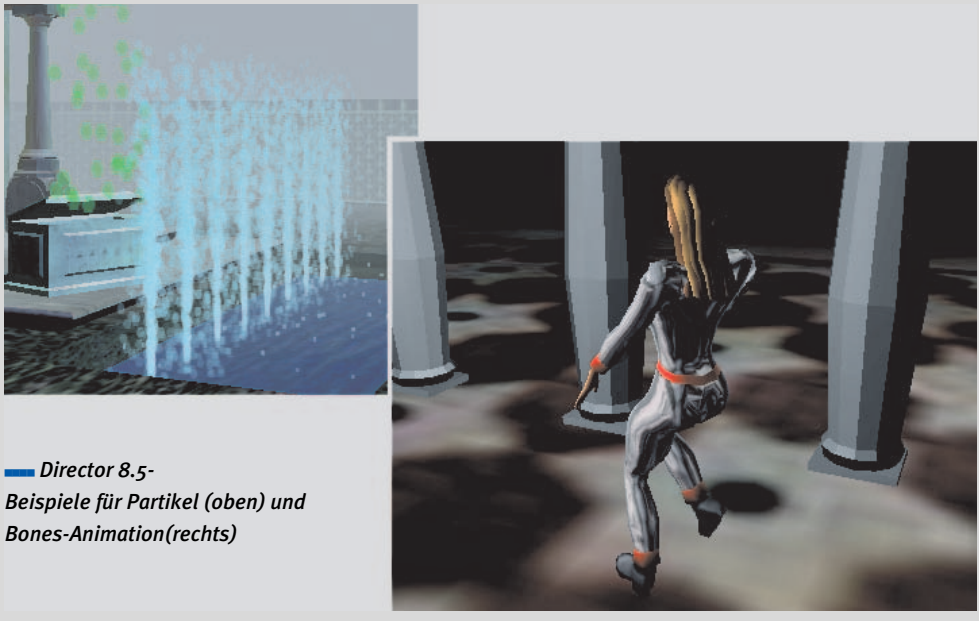
Texturierte Seitenflächen



Eine Mischung von Schattierungen

Übersicht 3D-Technologien

eCommerce-Beispiele mit Director 8.5



Director 8.5-
Beispiele für Partikel (oben) und
Bones-Animation(rechts)

Director 8.5 enthält eine sehr gute Simulation von physikalischen Eigenschaften. Die Würfel in dem Beispiel fallen gemäß korrekter Physik, nachdem das kleine Auto dagegen gefahren ist.

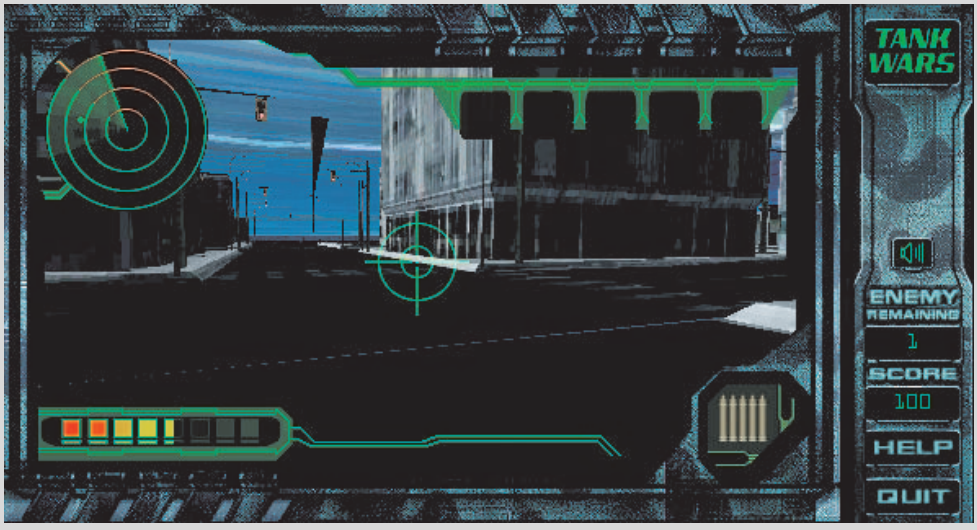
Beispielbilder Cult3D

Beispielbilder Viewpoint

Beispielbild EON

Beispielbild 3Danywhere

Beispielbild Blaxxun



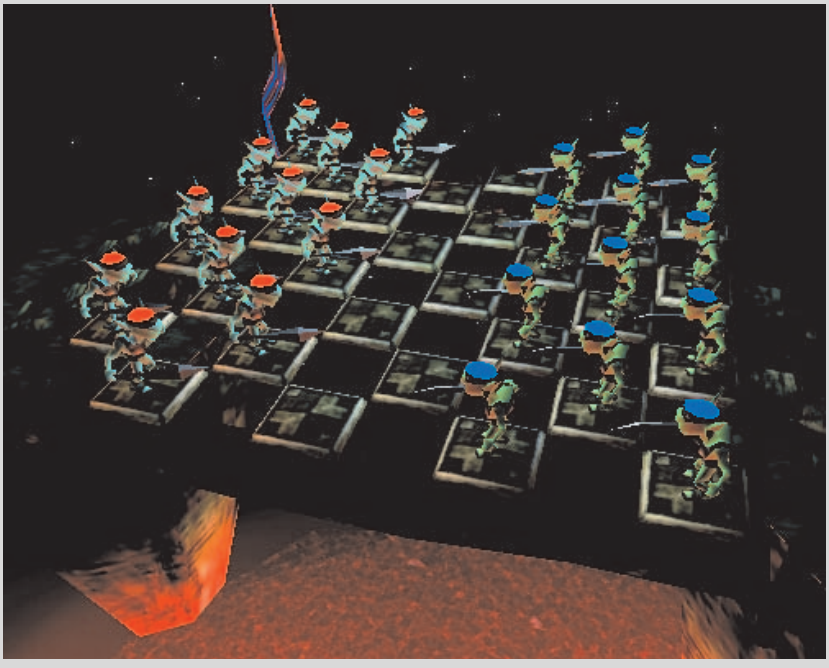
Beispielbild 3DGroove

Beispielbild Atmosphere

Beispielbild Pulse3D

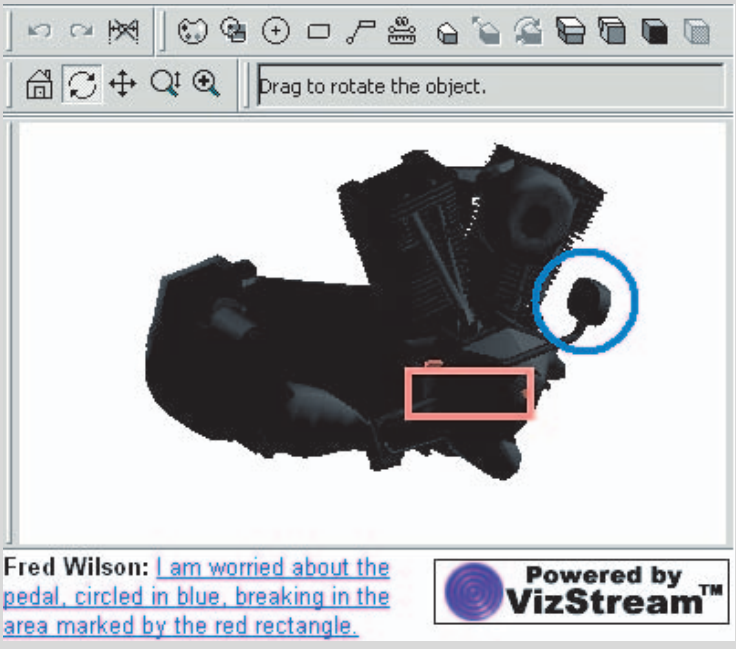
Beispielbild RichFX

Beispielbild Virtue3D



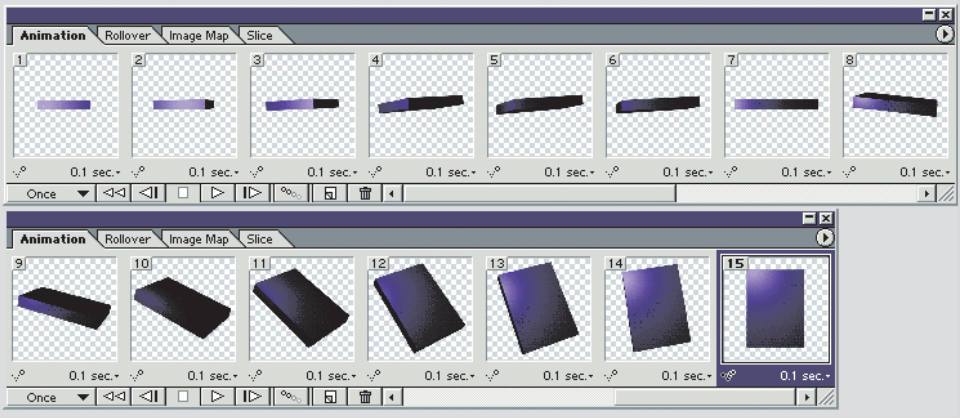
Beispielbild B3D

Beispielbild CyCo-Systems



Beispielbild VizStream

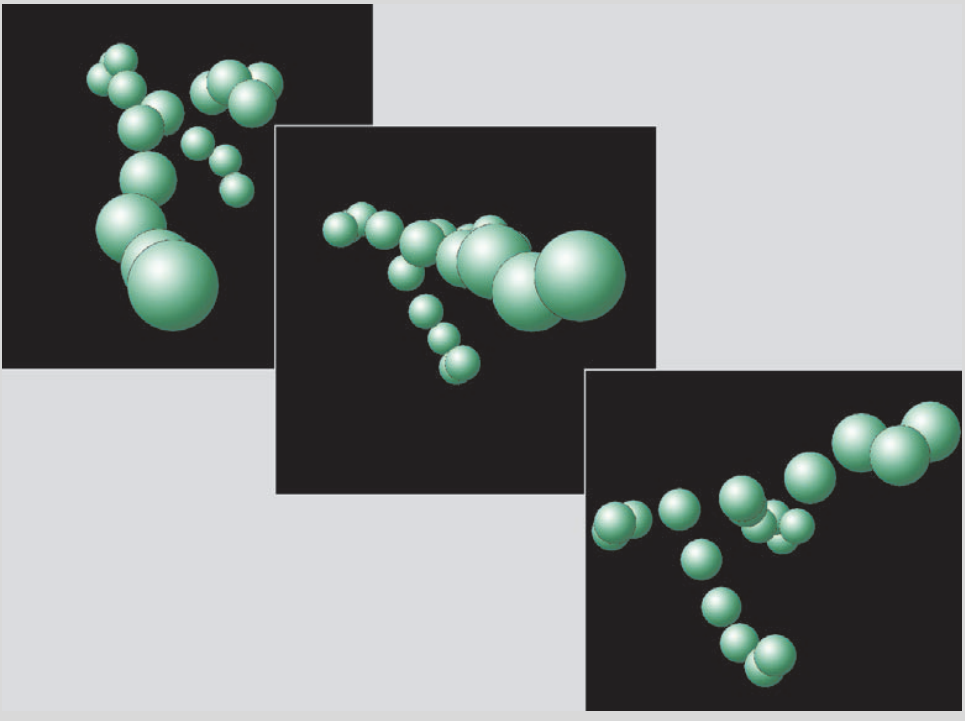
DHTML



Bildsequenz für Animated-GIF

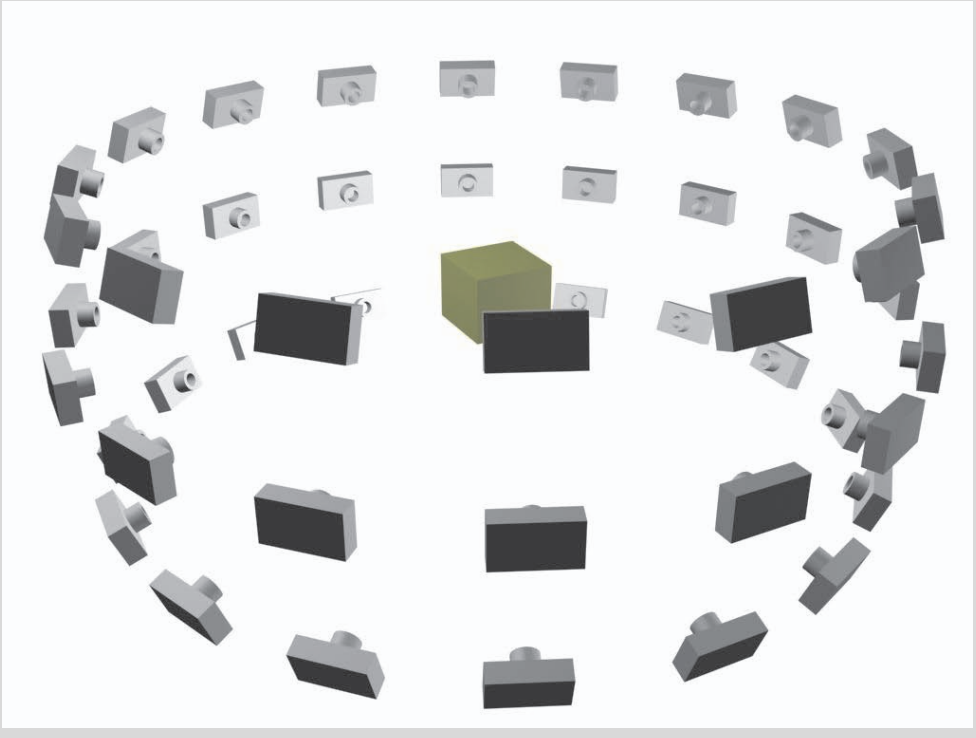
Fertiges Beispiel mit Animated-GIF

Ausgangs-Sprite für das Molekül-Beispiel

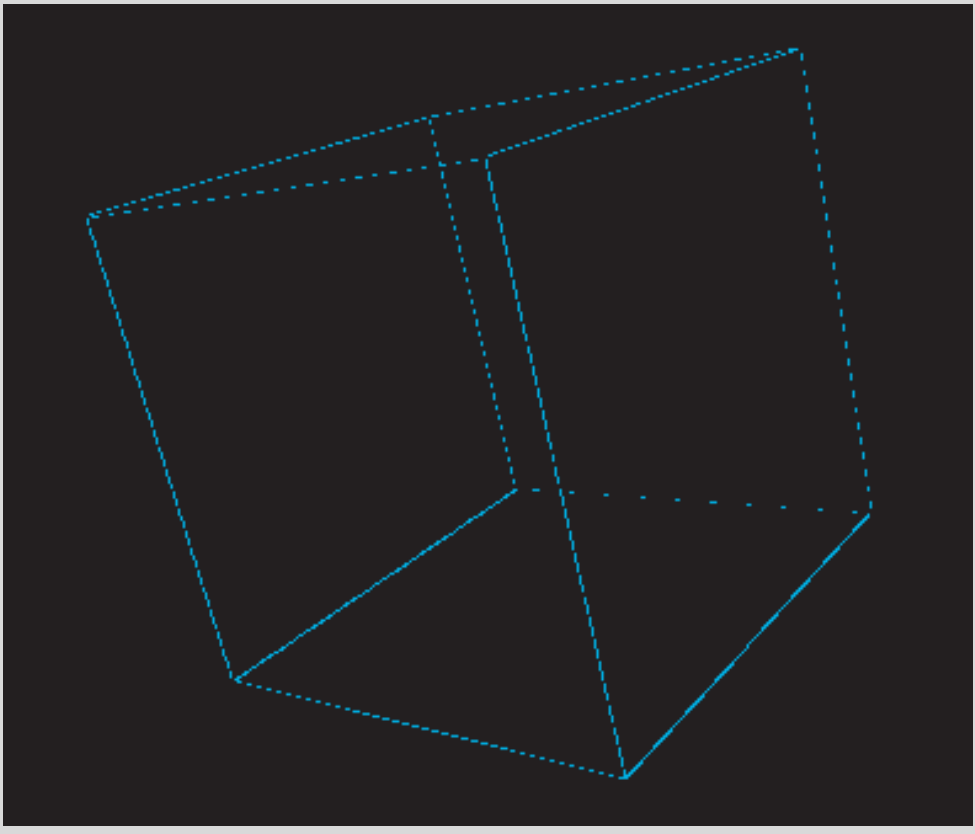


Sprite-Beispiel-Bilder

Bildsequenzen für Objektpräsentation

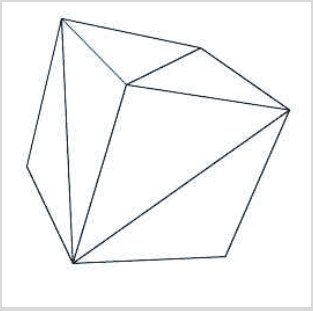


Die Positionen, aus denen das Objekt aufgenommen wurde

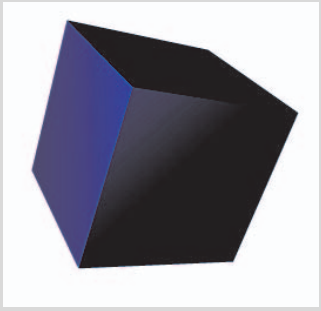


Real berechnete 3D-Grafik in DHTML. Durch die grobe Skalierung der Grafiken, werden die Linien unterbrochen gezeichnet.

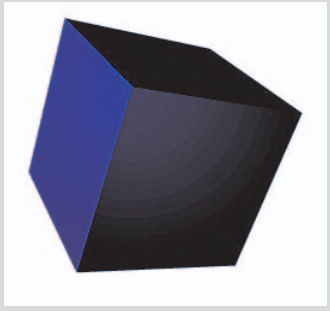
Flash



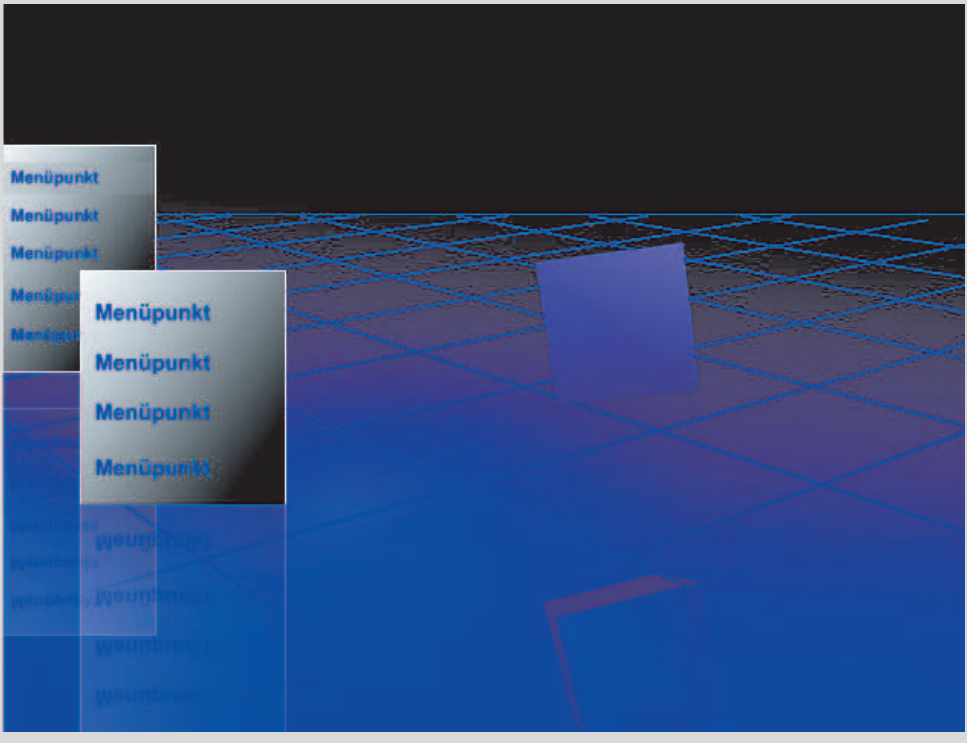
Swift3D Outline-Modus



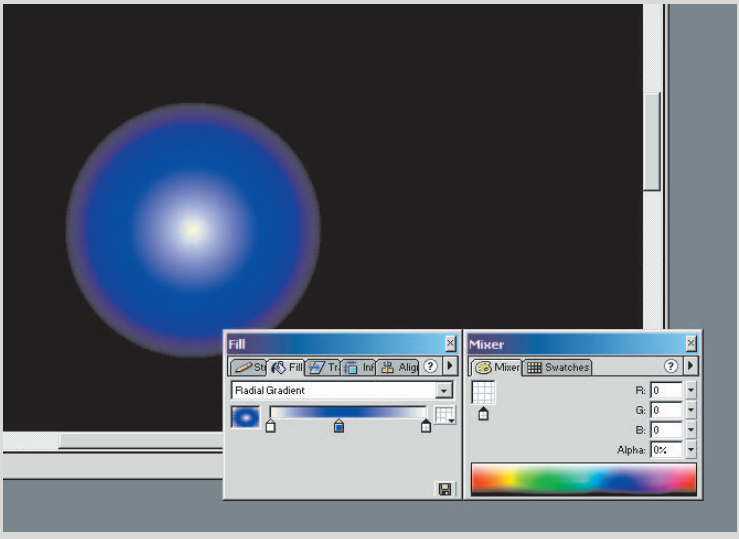
Swift3D Flat-Modus



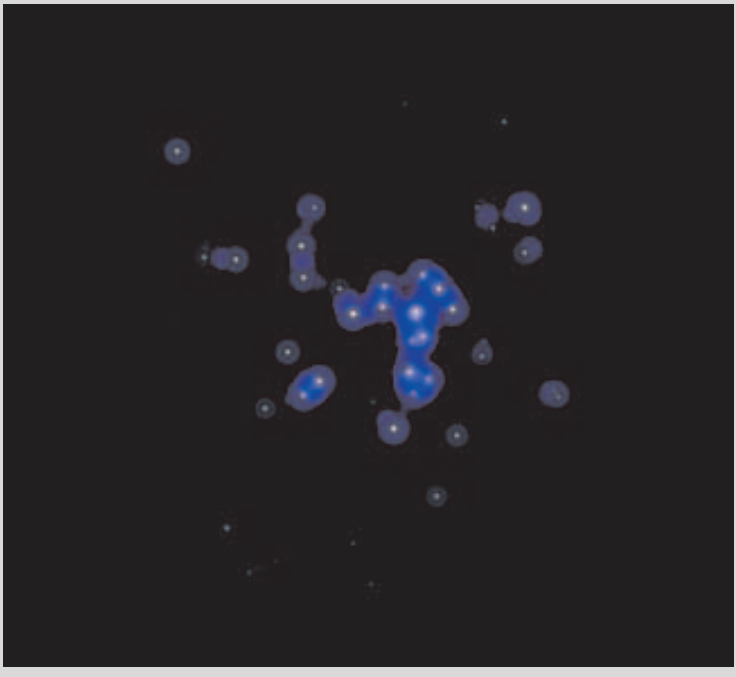
Swift3D Area-Modus



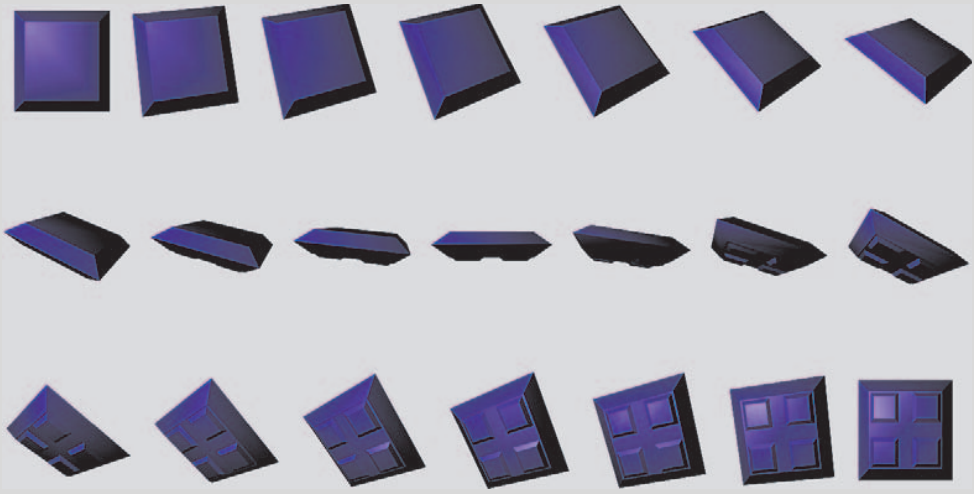
Beispielbild für Spiegelfläche



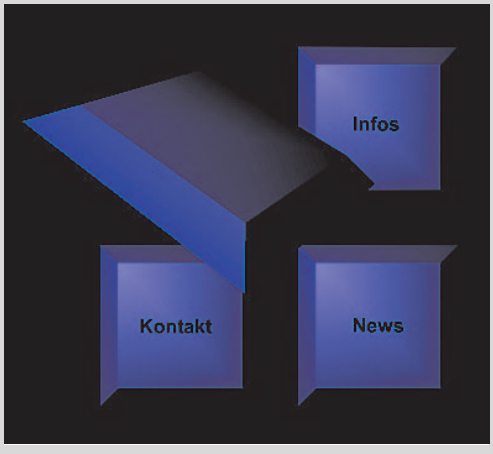
Die Konstruktion des Partikels in Flash



Partikel in Flash



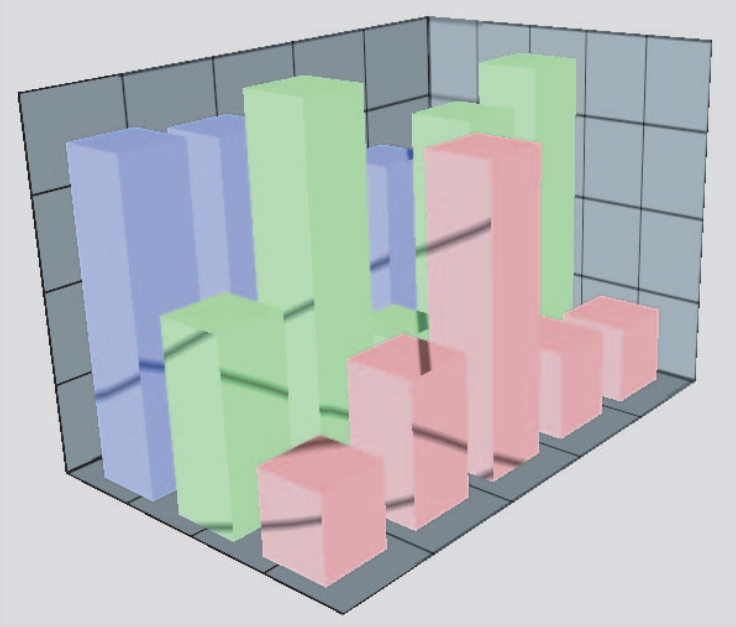
Bildsequenz für Menüsystem



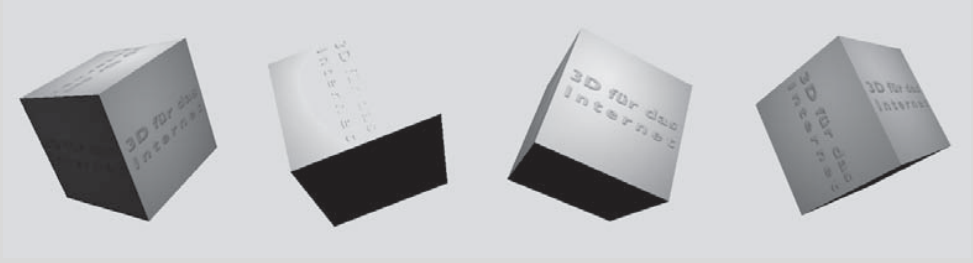
Beispiel mit obiger Bildsequenz

Director und Shockwave

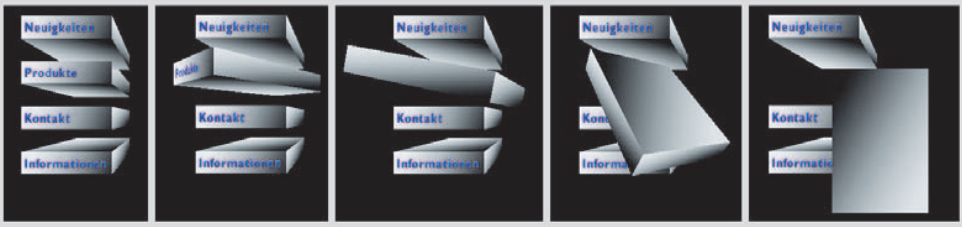
Eine mit „Imaging Lingo“ generierte Textur



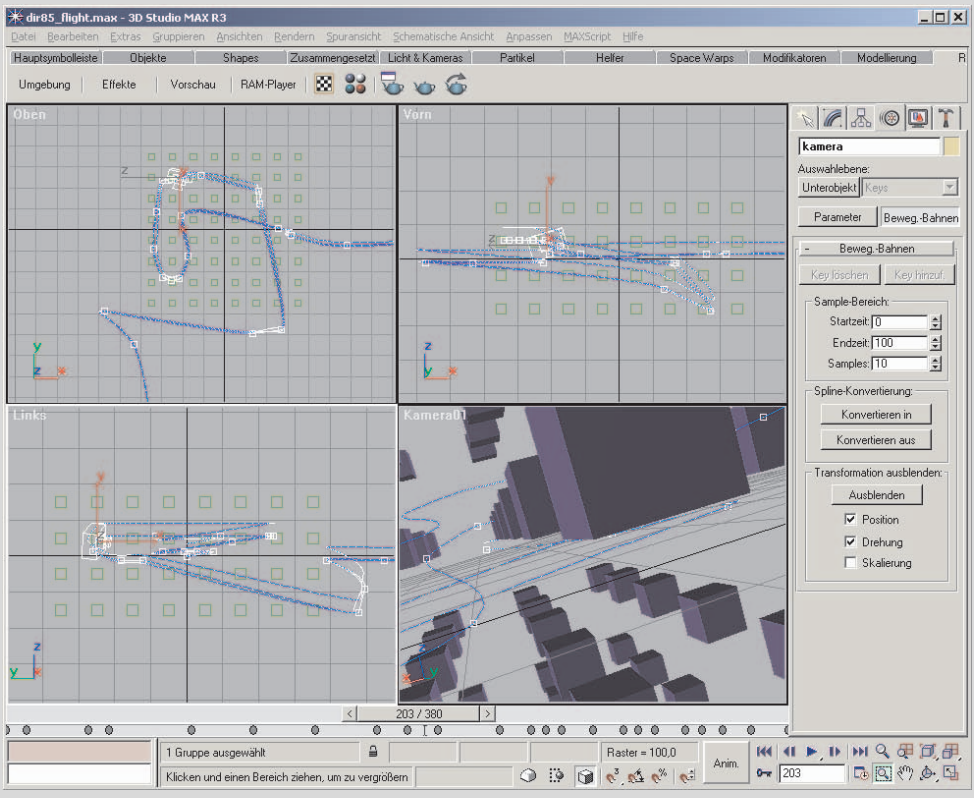
Beispiel Balkendiagramm



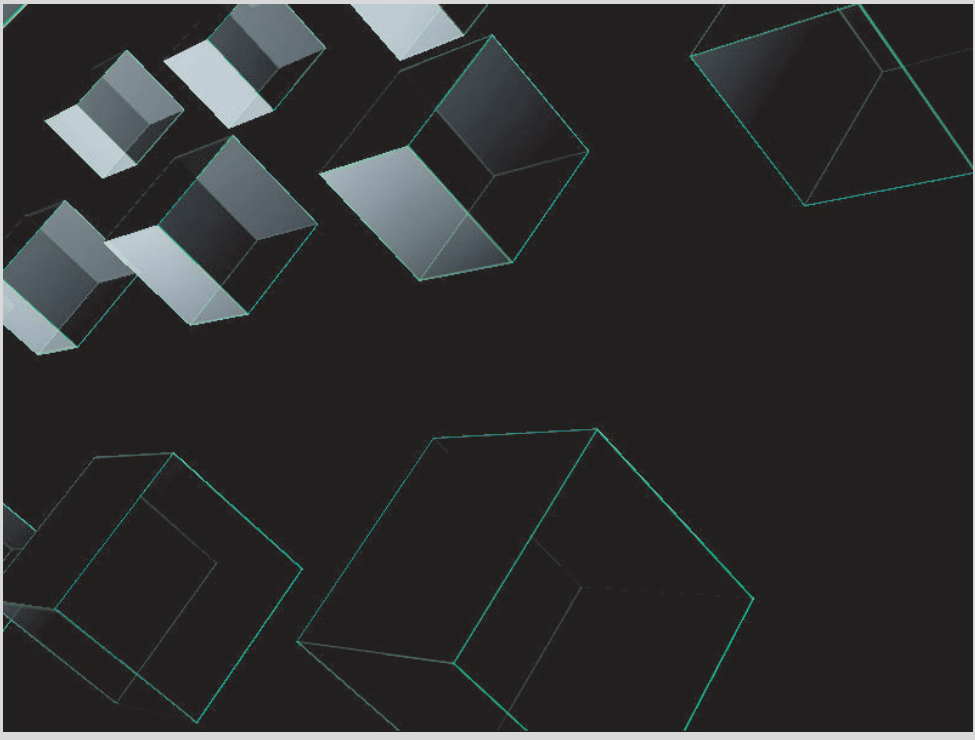
Rotation eines Würfels



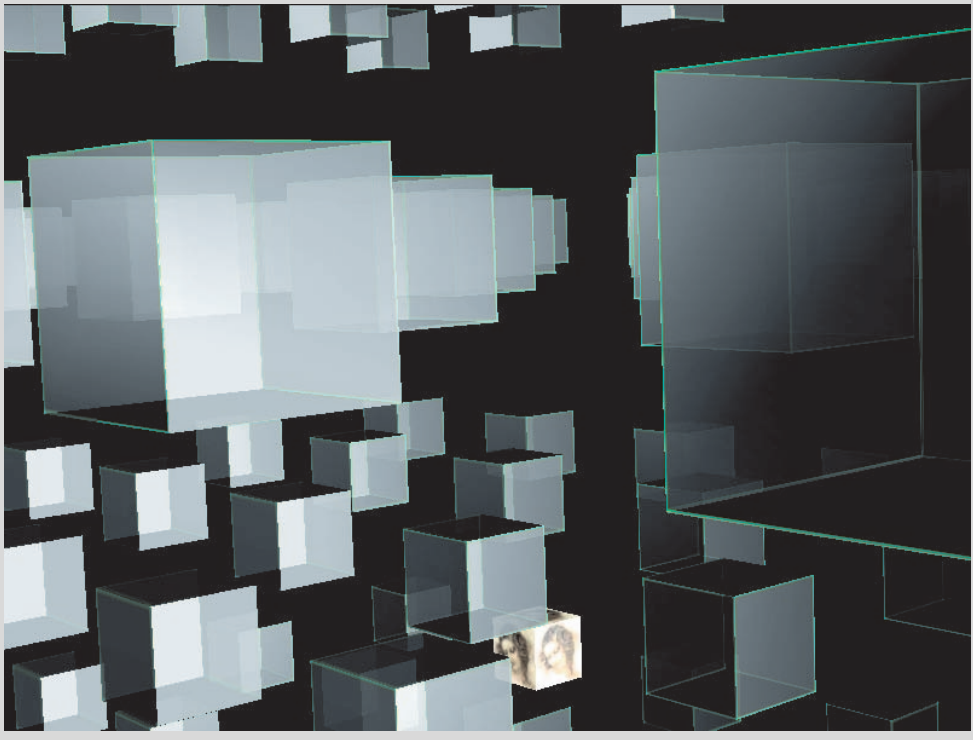
Beispiel Menüsteuerung



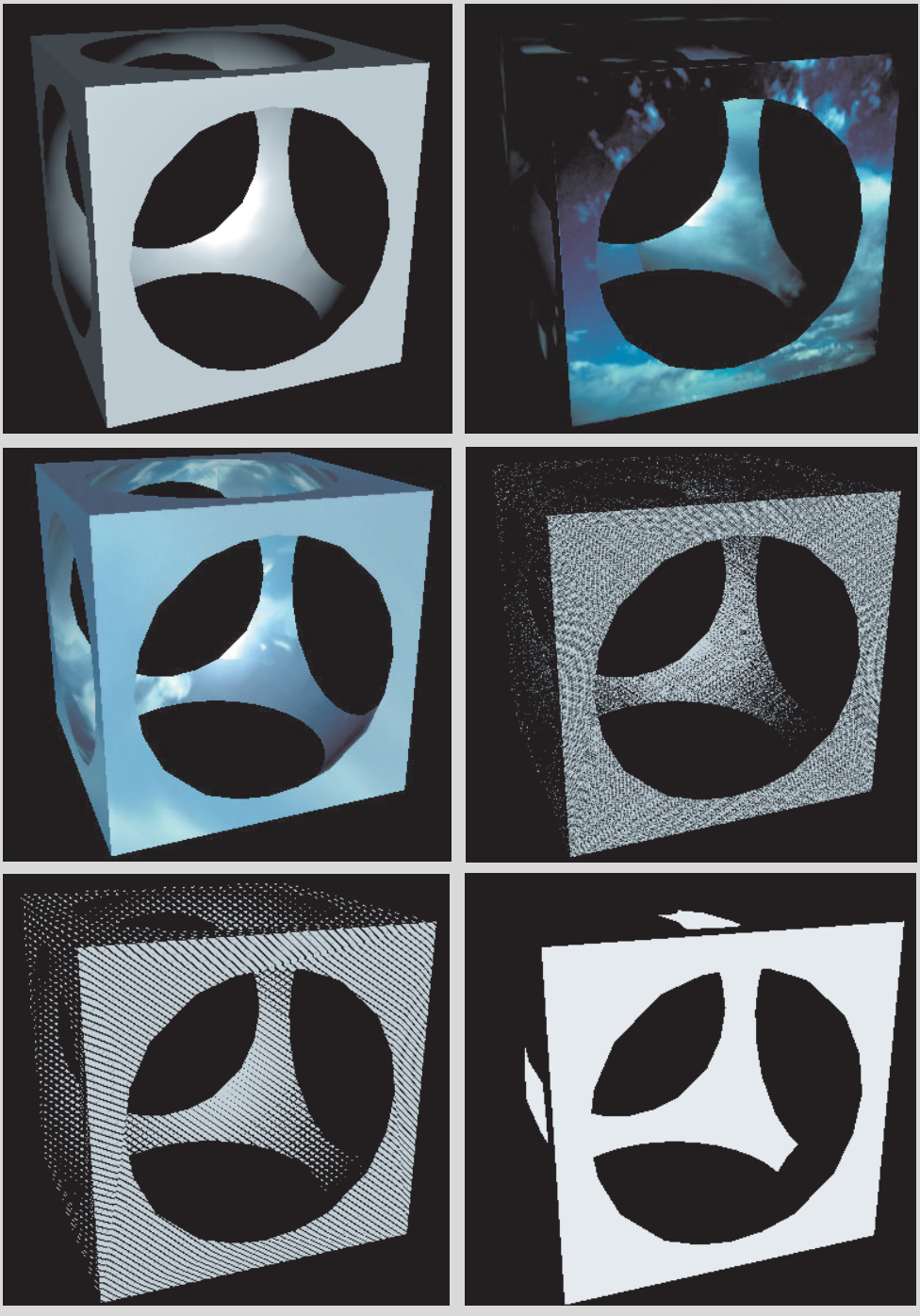
Film Animation in 3D Studio Max



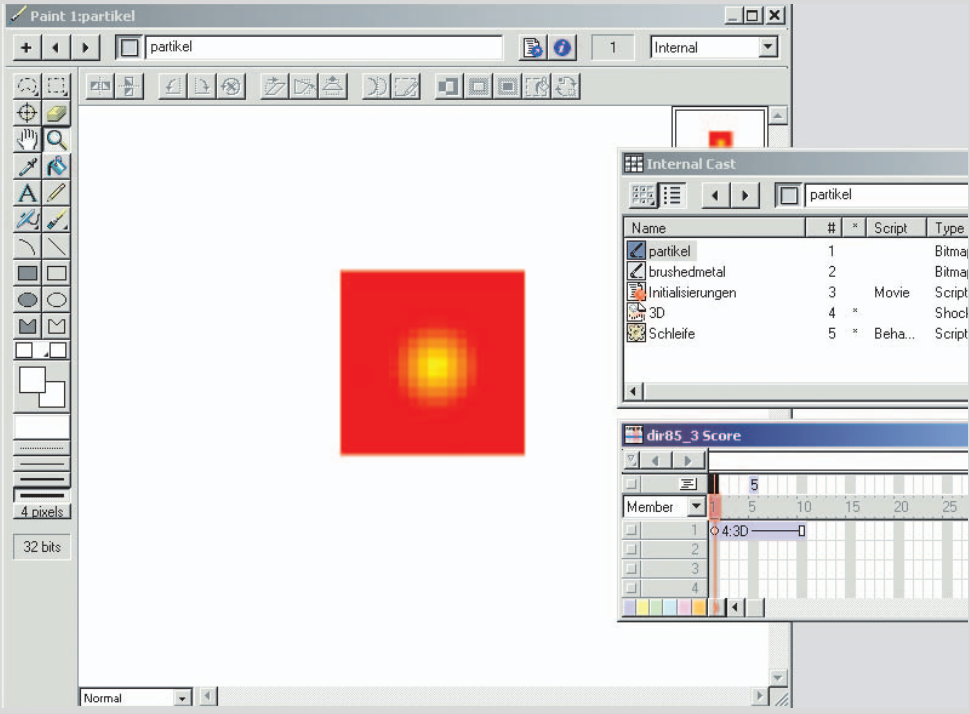
Beispiel Film



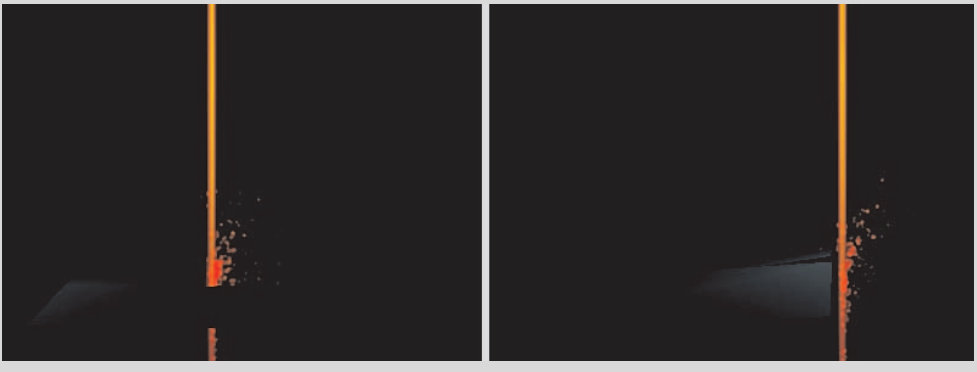
Beispiel Film



Verschiedene Schattierungsmodi



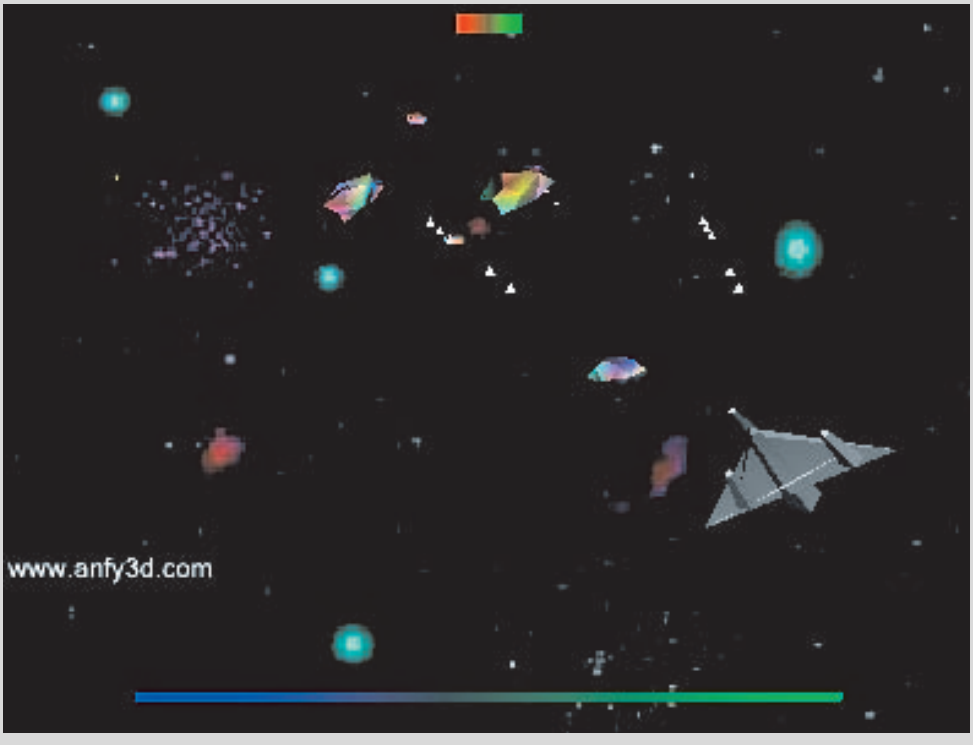
Die Erzeugung des Partikels in Director



Beispiel Partikel

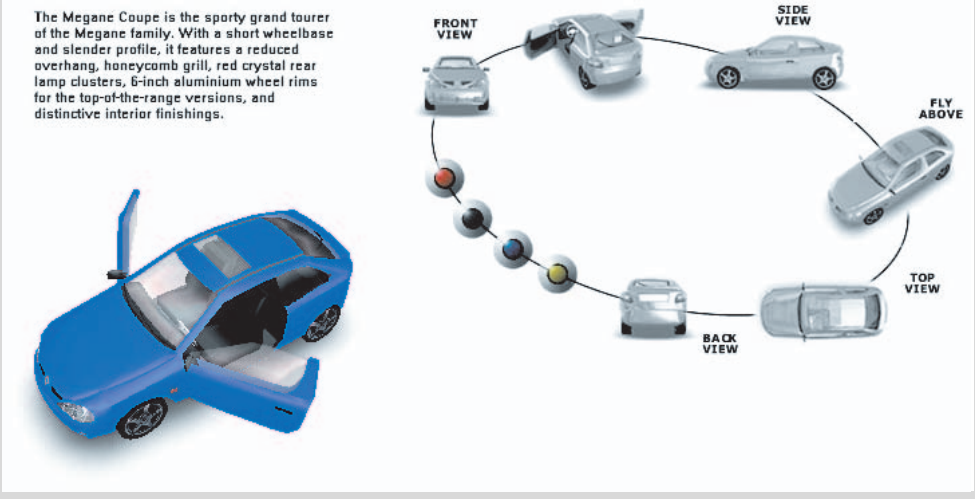
Java

Das Anfy3D-Logo



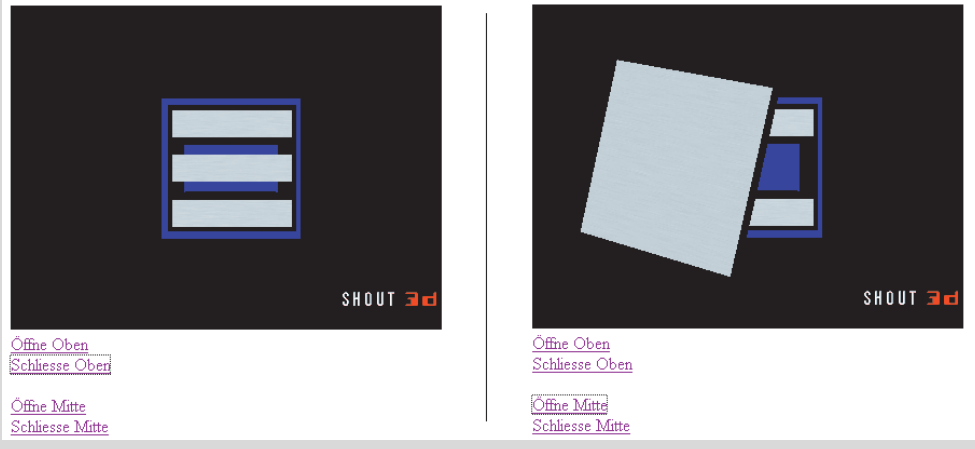
Beispiel für Anfy3D

Beispiel für 3Danywhere



Beispiel für 3Danywhere

Beispiel für Shout3D



Beispiel mit Shout3D

X3D und VRML

```
<?xml version="1.0" encoding="UTF-8" ?>
- <X3D>
+ <head>
- <Scene>
  - <Group>
      <Viewpoint description="hello world!" orientation="0 0 1 0" position="0 0 10" />
      <NavigationInfo type="EXAMINE ANY" />
    - <Shape>
        <Box DEF="B" />
      + <Appearance>
      </Shape>
    - <Transform translation="-2.4 -0.3 2">
      - <Shape>
          <Text string=""Hello world!"" />
        + <Appearance>
        </Shape>
      </Transform>
    </Group>
  </Scene>
</X3D>
```

Der Szenengraph als XML-Datei im Internet Explorer

X3D-Beispiel