VRML & X3D

Seminar:
Computer-Animation und Visualisierung
Prof. Dr. Bender

WS 2005/2006

24.11.2005

Christian Fritsche,
Florian Moritz,
Christoph Gerstle

FH Kaiserslautern, Zweibrücken
Studiengang: Digitale Medien

Inhaltsverzeichnis VRML & X3D

1 Vorwort....4

2 Geschichte....4

3 Stand der Dinge....5

4 Viewer....6

4.1 BS Contact VRML/X3D....6
4.2 Flux....6
4.3 Xj3D....6
4.4 Cortona....7

5 Autorenwerkzeuge....8

5.1 Blender....8
5.2 VRMLPad....9
5.3 X3D-Edit....10

6 Spezifikation VRML....11

6.1 Allgemeines zu VRML....11
6.2 Gruppenknoten....12
6.2.1 Group....12
6.2.2 Transform....12
6.2.3 Exkurs: Datentypen....13
6.2.4 Exkurs: Felder und Ereignisse....13
6.2.5 Inline....14
6.2.6 weitere Gruppenknoten:....14
6.3 Blattknoten (Kindknoten)....15
6.3.1 Shape....15
6.3.2 DirectionalLight....15
6.3.3 PointLight....15
6.3.4 SpotLight....15
6.3.5 Background....16
6.3.6 Viewpoint....17
6.3.7 Übersicht aller Kindknoten....18
6.4 geometrische Knoten (Objektknoten I)....18
6.4.1 Box....18
6.4.2 Cone....18
6.4.3 Cylinder....18
6.4.4 Sphere....18
6.4.5 Text....19
6.4.6 ElevationGrid....20
6.4.7 PointSet....20
6.4.8 IndexedLineSet....21
6.4.9 IndexedFaceSet....22
6.4.10 Extrusion....23
6.5 Hilfsknoten (Objektknoten II)....24
6.5.1 Color....24
6.5.2 Coordinate....24
6.5.3 Normal....24
6.5.4 TextureCoordinate....24
6.6 Knoten zum Aussehen von geometrischen Objekten (Objektknoten III)....25
6.6.1 Appearance....25

6.6.2 Material....25
6.6.3 ImageTexture....25
6.6.4 MovieTexture....25
6.6.5 PixelTexture....25
6.6.6 TextureTransformation....25
6.7 Instantiierung mit DEF und USE....26
6.8 Prototypen....27
6.9 Externe Prototypen....29
6.10 Animationen....31
6.10.1 Erläuterung....32
6.10.2 Routen....32
6.10.3 Interpolatoren....33
6.10.4 Sensoren:....33
6.11 JavaScript und Java in VRML....34
6.11.1 JavaScript in VRML....34
6.11.2 Java in VRML....36
7 X3D....37
7.1 Aufbau und Vergleich zu VRML....37
7.2 X3D Spezifikation....40
7.2.1 Eine Spezifikation im Detail:....41
7.3 Neue Möglichkeiten mit X3D....42
7.3.1 Metadata nodes....43
7.3.2 Keysensor....43
7.3.3 StringSensor....45
8 Beispielanwendung – Berlin-Szenerie....48
8.1 Siegessäule....48
8.2 Reichstag....50
8.3 Brandenburger Tor....56
8.4 Peripherie....59
9 Fazit....64
10 Quellennachweise....65

1 Vorwort

Diese Seminararbeit zu Computergrafik und Visualisierung gibt einen Überblick über VRML (Virtual Reality Modelling Language) und dessen Entstehung und Entwicklung. Außerdem gehen wir kurz auf das Zusammenspiel von VRML mit anderen Programmiersprachen ein.
Des Weiteren zeigen wir die Weiterentwicklung X3D (Extensible 3D). Hier geben wir eine Übersicht über den aktuellen Entwicklungsstand und gehen auf Unterschiede zu dem Vorgänger VRML ein. Auch mögliche zukünftige Entwicklungen sollen dem Leser präsentiert werden.
Auch stellen wir eine Auswahl an VRML/X3D Betrachtern vor, die es dem Anwender ermöglichen VRML Dateien darzustellen. Wir werden Autorenwerkzeuge aufzeigen, die von einem einfachen Texteditor bis zu komplexen 3D Programm reichen.
Als Beispiel stellen wir anschließend anhand eines größeren Projektes den Einsatz von VRML in der Praxis dar. Wir erläutern ausgewählte Programmsegmente, die Einblicke in die Möglichkeiten von VRML, wie beispielsweise Animation und Interaktion, geben. Anhand von zahlreichen Screenshots erhält der Leser somit einen guten Überblick über die Struktur komplexer Projekte.
Abschließend soll ein Fazit darlegen, wie wir persönlich den Umgang von VRML/X3D empfunden haben und wie wir die zukünftige Entwicklung abschätzen.

2 Geschichte

Mark Pesce [Geschichte1] und Tony Parisi [Geschichte2] entwickelten 1993 eine dreidimensionale Schnittstelle zum World Wide Web, die sie im Frühjahr 1994 auf der ersten WWW-Konferenz in Genf vorstellten. Die Version 1.0 von VRML („Wörml") wurde dann im November 1994 spezifiziert [Geschichte3]. Es war mehr oder weniger ein "Abfallprodukt" des API (Application Programming Interface) *Open Inventor* von Silicon Graphics Incorporated (SGI). Die virtuellen Welten, die man mit VRML 1.0 spezifizieren konnte, waren anfangs jedoch zu statisch. Interaktion war noch nicht möglich. Deshalb wurde dann 1997 VRML 2.0 spezifiziert und nach kleineren Änderungen als ISO Standard festgeschrieben (ISO/IEC 14772-1:1997 und ISO/IEC 14772-2:2002) [Geschichte4]. VRML 2.0 ist auch bekannt als VRML97. Erweiterungen waren gegenüber VRML 1.0 die größere Interaktivität und bewegliche Objekte (*Moving Worlds*). Offizieller Nachfolger und ebenfalls ISO Standard wurde im Dezember 2004 dann X3D. Ein Unterschied zwischen VRML und X3D ist die veränderte Syntax, die durch die jeweilige Spezifikation genau definiert wird. X3D ist im Gegensatz zu VRML modular aufgebaut. Die ganze X3D Spezifikation ist dabei in so genannte Profile unterteilt, die sich in Komplexität und Funktionalität unterscheiden. Mehr dazu im Kapitel Spezifikation. Die Spezifikation von X3D ist dabei noch komplexer wie die von VRML.

Die offiziellen Logos der beiden Sprachen:

VRML

X3D

3 Stand der Dinge

VRML wurde von X3D abgelöst, womit klar wird, dass es Neuerungen nur noch im X3D-Bereich gibt.
X3D hat den breiten Markt noch nicht erreicht. Seit August 2005 ist die neue Webseite des Web3D Konsortiums [StandDerDinge1] online. Hier laufen viele Fäden zusammen. Die Seite versucht auch einen Überblick über aktuelle Entwicklungen zu geben. Es gibt Projekte der Web3D Konsortium Gruppen im Bereich CAD und Medizin. Und natürlich auch "Siggraph-Projekte" (Tagungsreihe der Special Interest Group on Graphics and Interactive Techniques), wie z.B. das Projekt "Sazka Arena" der Firma Bitmanagement [StandDerDinge2] (Die Sazka Arena ist eine Multifunktionshalle in Prag, die Platz für bis zu 18 Tausend Besucher bietet). Und Wettbewerbe wie der "Diamond Award" [StandDerDinge3] der Firma SpaceTime 3D. Die X3D-Community ist auf jeden Fall überschaubar. Im öffentlichen Mailverteiler trifft man häufig auf die gleichen Namen.
Viele Teile der Spezifikation sind bis heute noch nicht realisiert und oft fehlen Codebeispiele für bereits vorhandene Features oder fertige Projekte. Außerdem gibt es momentan nur ein Buch welches "X3D" im Titel trägt! Auch gibt es nach wie vor Probleme bei den Viewern, die alle kleinere oder mittelgroße Schwächen haben. Fakt ist allerdings, es gibt einen offenen Standard auf den man aufbauen kann. Mit der XML-Fähigkeit ist dieser auch zukunftssicher. Man spürt auf jeden Fall den Open Source Gedanken im X3D Umfeld, auch die Entwickler sind eher intrinsisch motiviert.

Eine Studie über die Auswertung von Googlequeries verdeutlicht: beim Thema 3D Formate sind VRML und X3D spitze. Im Vergleich zu 2004 haben sich die monatlichen Anfragen im September 2005 nach VRML auf ca. 4 Millionen vervierfacht. Bei X3D haben sie sich gar um den Faktor 5,5 erhöht, allerdings auf "nur" 158 Tausend. Shockwave 3D und Adobe Atmosphere liegen mit ca. 60 bzw. 20 Tausend Anfragen weit abgeschlagen. [StandDerDinge4]

4 Viewer

4.1 BS Contact VRML/X3D

Hersteller: www.bitmanagement.de

Den ersten Viewer den wir vorstellen ist Contact von der deutschen Firma Bitmanagement mit Sitz in Berg (Bayern). Die Bitmanagement Software GmbH wurde durch das ehemalige Client-Team der blaxxun interaktive AG gegründet. Gründer der Bitmanagement Software GmbH ist Holger Grahn sein preisgekrönter VRML Viewer GLView wurde 1997 von blaxxun als der führende VRML Viewer eingestuft und erworben. [Viewer1]
BS Contact liegt aktuell in der Version 6.2 vor. Er ist quasi Standard. „Contact hat die meisten Features und die beste Performance...", schreibt Eckhard Jaeger Art Director bei area42, Berlin in einem X3D- Forum. [Viewer2]
Darüber hinaus unterstützt er viele Erweiterungen (BS Extensions) die fortgeschrittene Techniken, wie Bumpmapping, BSP-Trees oder Multitexturing ermöglichen.

BS Contact VRML/X3D Features-Übersicht:

- Collision Detection
- Level Of Detail
- Extended Nurbs (tessellation) und Splines
- Extended Textures bis zu Multitexturing und DirectX9 Shader
- Video Funktion der 3D Kamera
- Speicherung von hochauflösenden Bildern möglich

4.2 Flux

Hersteller: www.mediamachines.com

Die aktuelle Version ist 1.2.1 Final, Build 543, seit 1.2 besteht der X3D Support für den Viewer.
Flux hat bekannte Schwächen, wie zum Beispiel bei Kollision und der Clipping Plane Umsetzung und was viel ungünstiger ist, dass ActiveX Control ist nicht signiert und lässt sich so ohne weiteres unter WindowsXP SP2 nicht installieren. Maßgeblich an der Entwicklung von Flux war Tony Parisi beteiligt.

Die Features des oben genannten Releases sind u.a.:

- Humanoid Animation Component (H-Anim)
- Eingeschränkte NURBS Unterstüzung (NurbsPatchSurface, NurbsSurfaceInterpolator, NurbsTextureCoordinate nodes)
- Immersive Profile
- LoadSensor node
- MultiTexture
- External Scene Access Interface mit ECMAScript und COM bindings
- Internal Scene Access Interface mit ECMAScript

4.3 Xj3D

Hersteller: www.xj3d.org

Xj3D ist ein Projekt des Web3D Konsortiums. Es hat sich zum Ziel gesetzt ein VRML97 und X3D Toolkit zu schaffen, welches komplett in Java geschrieben ist. Auch ein Viewer ist in dem Toolkit enthalten. Die Entwicklung wird unter anderem von Sun Microsystems finanziell gefördert. Das Toolkit kann dazu genutzt werden VRML Inhalte in eigene Anwendungen einzubinden. Xj3D wird dazu genutzt die neuen X3D Spezifikationen zu verifizieren. Momentan aktuell ist Milestone 10. Der finale Status (1.0) soll mit dem nächsten Release erfolgen.

Nachtrag: Seit 20. Oktober gibt es den Release Candidate 1 von Xj3D!

Folgende X3D Komponenten wurden bisher implementiert:
- CADGeometry
- DIS
- GeoSpatial
- H-Anim

Neue proprietäre Erweiterungen für Xj3D sind:
- Rigid Body Physics
- Particle Systems
- Clipping planes
- Picking Utilities
- Abstract Device IO

4.4 Cortona

Hersteller: www.parallelgraphics.com

Cortona ist wohl einer der meist verbreitesten VRML Player. Das erste Final Release erschien bereits im Februar 1999. Der Player hat sehr gute Frameraten. Die Installation ist sehr einfach. Es gibt Plugins für den Internet Explorer, aber auch für Opera und Firefox genügen zwei Klicks für die Installation. Momentan aktuell ist die Version 4.2.

Die Features des o.g. Releases sind u.a.:
- komplette VRML97 Unterstützung
- Fortgeschrittenes Rendering: mipmapping, phong lighting, reflection mapping und enhanced anti-aliasing
- Unterstützung für Macromedia Flash
- Zusätzliche Knoten und Fähigkeiten die die VRML Spezifikation erweitern

Momentan wird gerade an der Version 5.0 gearbeitet (Beta Status). Diese Version wird dann auch X3D unterstützen.

Features des 5.0 Beta-Releases sind u.a.:
- EventUtilities Knoten (*BooleanFilter, BooleanToggle, BooleanTrigger, IntegerSequencer, IntegerTrigger, TimeTrigger*)
- 2D Geometry Knoten (*Circle, Rectangle, Coordinate2D, PointSet2D, IndexedLineSet2D, IndexedFaceSet2D*)
- 2D Knoten (*Transform2D, CoordinateInterpolator2D, Background2D*)

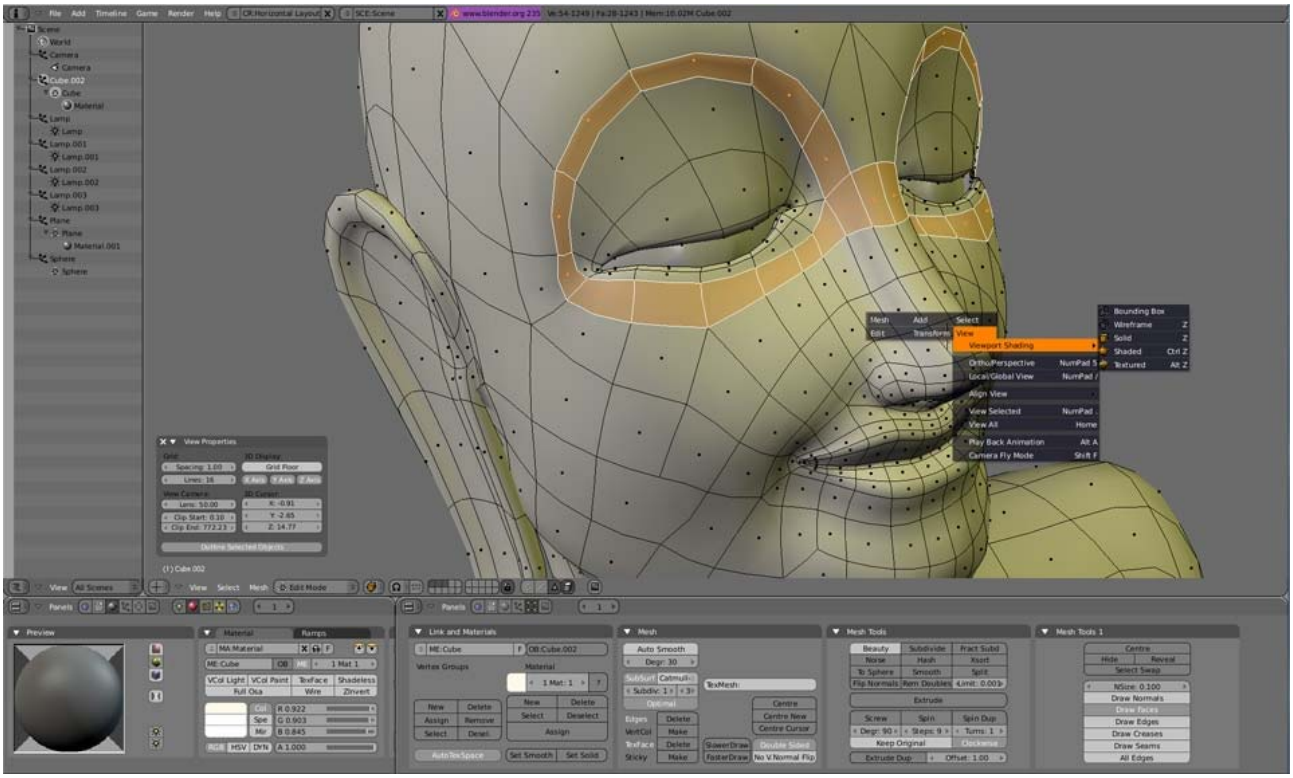
5 Autorenwerkzeuge

Nahezu alle aktuellen Modellierungswerkzeuge unterstützen VRML bzw. X3D Export.
In 3d Studio Max kann man seit Release 3 (2001) in VRML exportieren. Für Maya existieren auch Open Source Projekte die einen X3D Export ermöglichen [Autorenwerkzeuge1].
Wir werden uns nun drei Werkzeuge genauer anschauen die VRML/X3D-Autoren unterstützen können.

5.1 Blender

Homepage: http://www.blender.org

Blender ist ein freies 3D-Modellierungs- und Animationsprogramm. Die Einarbeitung in Blender ist als schwierig anzusehen, die Bedienung ist stark gewöhnungsbedürftig. Allerdings gibt es eine große Community und viele Ressourcen rund um Blender wie zum Beispiel Wiki Books [Autorenwerkzeuge2] und sogar kostenlose Tutorial-Videos auf der Homepage. Blender besitzt die Möglichkeit in VRML und X3D zu exportieren.



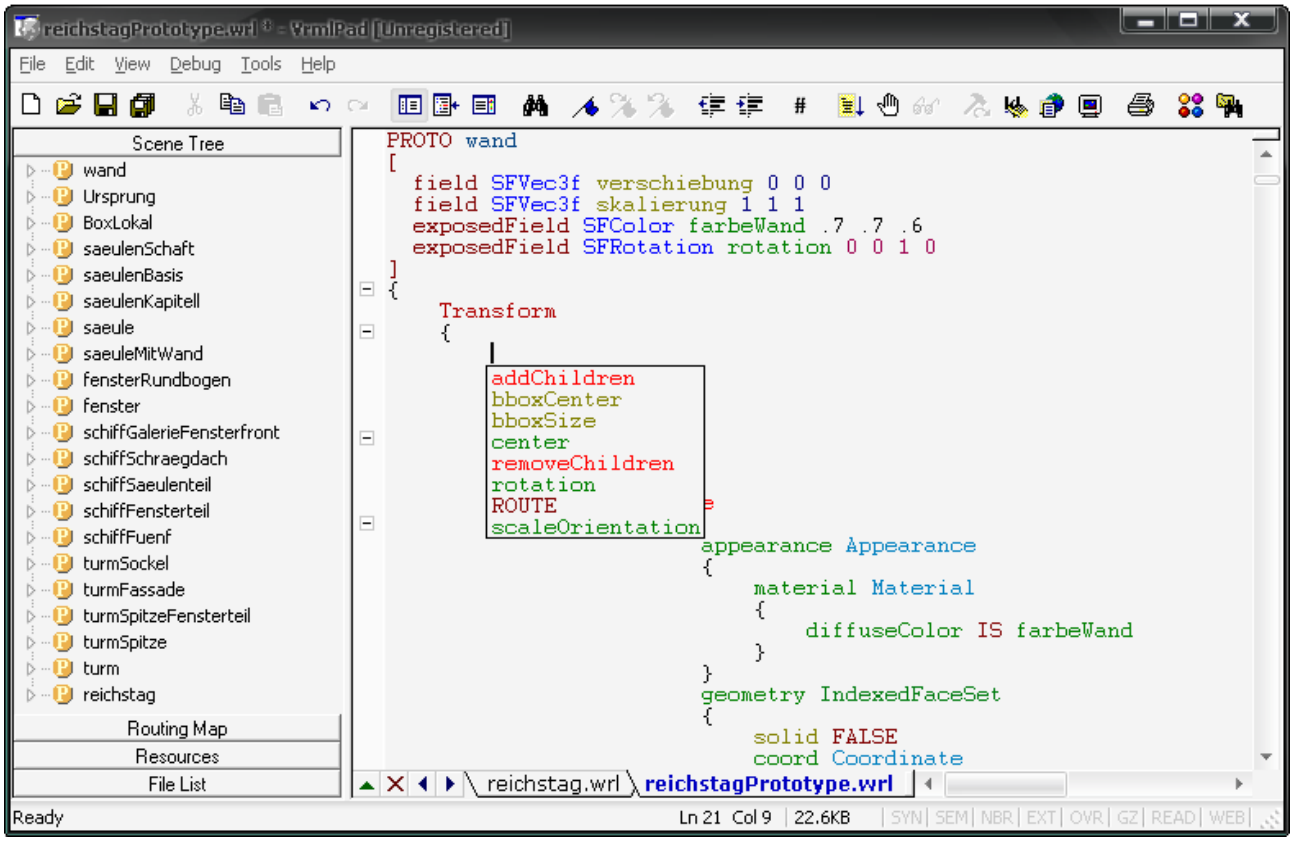
[Abbildung Autorenwerkzeuge1: Blender 2.36 Quelle: http://commons.wikimedia.org/wiki/Image:Blender_2.36_Screenshot.jpg]

Momentan aktuell ist Version 2.37a. ”Unter den neuen Funktionen befinden sich unter anderem Soft Bodys, die für pudding-ähnliche Animationen oder Kleidungsstücke verwendet werden können, echtes orthographisches Rendern, Unterstützung für mehrere CPUs und neue Transformationswerkzeuge. Außerdem verfügt die neue Game-Engine nun auch über eine analoge 4-Achsen-Joystick Schnittstelle, welche per Python-Scripting eingebunden werden.”
[Autorenwerkzeuge3]

5.2 VRMLPad

Homepage: http://www.parallelgraphics.com/products/vrmlpad

Das VRMLPad wurde ausschließlich bei unseren Modellerstellungen genutzt. Es ist ein kleiner (1 MB) und kompakter VRML Editor. Die Vollversion kostet 149 US Dollar. Die herunterladbare Evaluierungsversion 2.1 hat leider unter anderem die Einschränkung, dass man keine Inhalte in die Zwischenablage speichern bzw. heraus einfügen kann, die größer als 64k sind. Deshalb wurde von uns die ältere freie Version (1.3) verwendet.



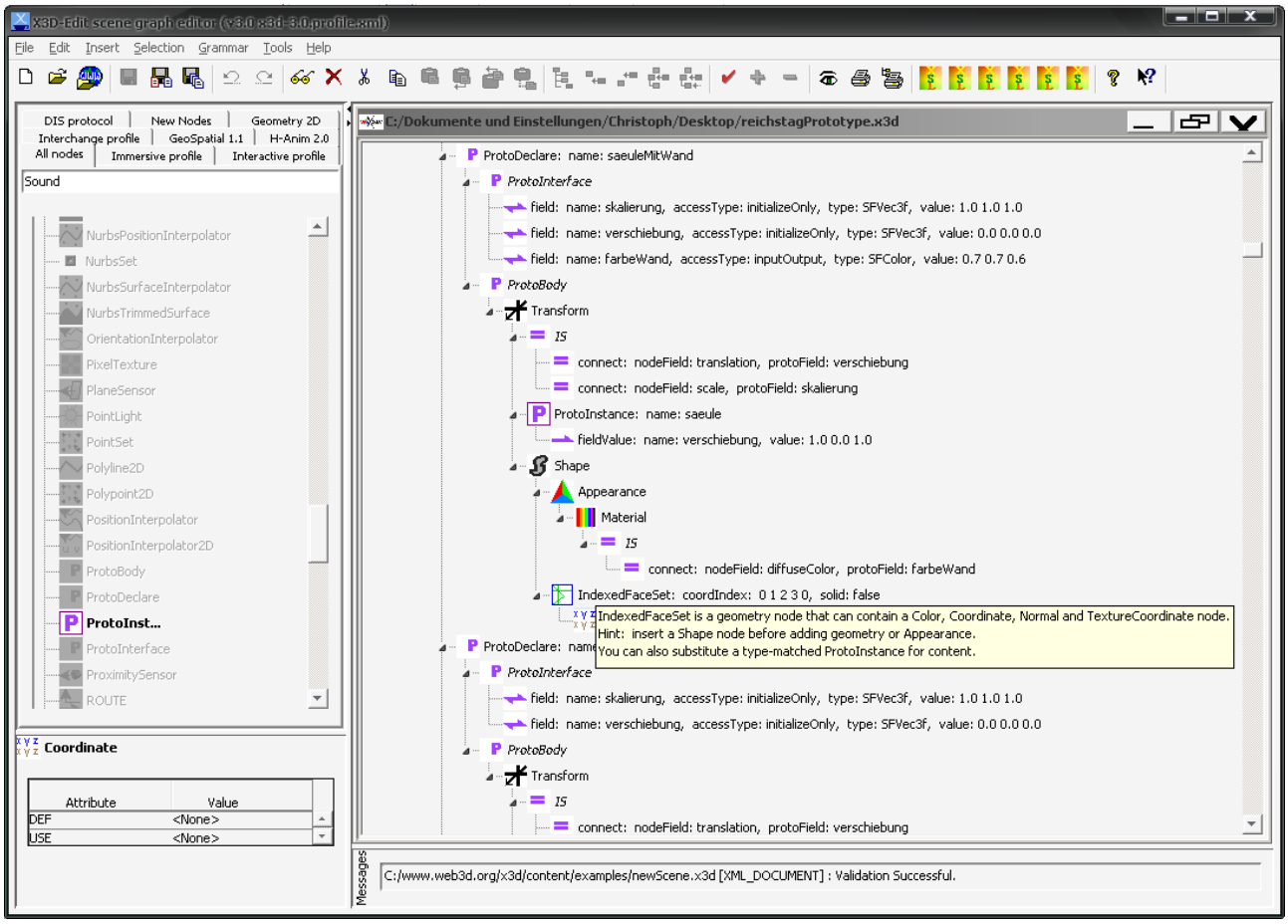
[Abbildung Autorenwerkzeuge2: VRMLPad Autovervollständigung]

Features vom VRML Pad sind neben dem Syntax Highlighting unter anderem die Autovervollständigung. So werden dem Autor mögliche Codeteile angezeigt, die ergänzt werden können. Außerdem ist eine dynamische Fehlerkontrolle vorhanden. Diese zeigt zum Beispiel an, dass eine Fließkommazahl erwartet wird und warnt vor unbelegten Feldern. Ein neues Feature der Version 2.1 ermöglicht das arbeiten mit Workspaces.

5.3 X3D-Edit

Homepage: http://www.web3d.org/x3d/content/README.X3D-Edit.html

X3D-Edit ist ein freier Grafikeditor für Extensible 3D (X3D). Er ermöglicht "fehlerfreies" Editing und Validierung von X3D oder VRML Szenengraphen. Kontextsensitive Tooltips bieten kurze Erläuterungen zu jedem VRML- Knoten und Attribut. Diese Tooltips sollen das Entwickeln wesentlich vereinfachen und das Verständnis für Neulinge sowie Experten gleichermaßen verbessern. X3D-Edit verwendet Java sowie den Xeena XML Editor von IBM. Maßgeblich an der Entwicklung beteiligt war Don Brutzman. Seit kurzem gibt es auch einen Auto-Installer der eine einfache Installation ermöglicht.



[Abbildung Autorenwerkzeuge3: X3D-Edit]

Features von X3D-Edit sind unter anderem das intuitive und mächtige User Interface, die Plattformunabhängigkeit (Java), die automatische Übersetzung von VRML in X3D Szenen sowie die Validierungsfunktion. Die automatische Übersetzung geschieht mittels des Programms *Vrml97ToX3dNist* [Autorenwerkzeuge4]. Der Übersetzer beruht auf dem XML Parser Xerces und XSLT Xlan. Die Übersetzung wird normalerweise per Hand durch eine Batchdatei bzw. eine C-Shell angestoßen. Xerces und Xlan sind in das jar-Packet eingebunden und müssen somit nicht extra heruntergeladen werden. Maßgeblich an der Entwicklung beteiligt war ebenfalls Brutzman.

6 Spezifikation VRML

Quellen:
[Spezifikation1]
[Spezifikation2]
[Spezifikation3]

6.1 Allgemeines zu VRML

- VRML-Dateien können mit jedem Texteditor bearbeitet werden.
  Näheres hierzu im Kapitel Autorenwerkzeuge.
- VMRL-Dateien haben die Endung *.wrl* für „world".
- Groß- und Kleinschreibung wird unterschieden!
- Folgende Zeichen dürfen nicht in Namen verwendet werden: { } [ ] „ ' # + . - ,
- Koordinaten werden in der Reihenfolgen X, Y, Z angegeben
- VRML verwendet ein rechtshändiges Koordinatensystem
- ein geometrischer Körper hat seinen Mittelpunkt bzw. Zentrum im Koordinatenursprung

- **Maßeinheiten**
  - lineare Distanzen werden in Metern angegeben
  - Zeitangaben in Sekunden
  - RGB-Werte sind Werte zwischen 0.0 und 1.0. z.B. 1.0 0.0 0.0 für Rot.
  - Winkel werden im Bogenmaß angegeben
  - Drehungen erfolgen im mathematisch positiven Sinn; d.h. gegen den Uhrzeigersinn

Gradmaß	0	45	90	135	180	225	270	315	360
Bogenmaß	0	0.786	1.571	2.356	3.142 = Pi	3.927	4.712	5.498	6.283 = 2*Pi

Umrechnungstabelle

- Jedes VRML-Dokument beginnt, wie auch bei anderen Skriptsprachen bekannt, mit einer Kommentarzeile der Form:

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

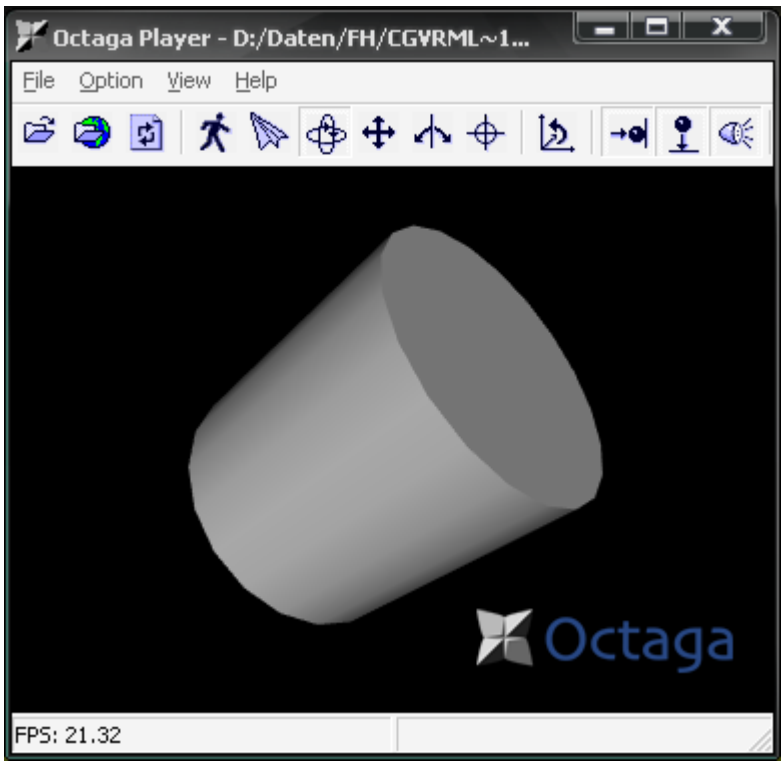
- Die Raute # steht für einen Zeilenkommentar. VRML natürlich für die Sprache. V2.0 für die Version und uft8 für die Codierung des Zeichensatzes. VRML wird also, wie hier gesehen, im UTF8 bzw. ASCII-Zeichensatz codiert.

Jetzt folgt der eigentliche Inhalt, die sog. Knoten.

```
Shape
{
   appearance Appearance
   {
      material Material {}
   }
   geometry Cylinder {}
}
```

Shape bildet den Hauptknoten, er ist für die Gestaltung von sichtbaren Objekten verantwortlich.
Es folgen zwei Unterknoten *appearance* und *geomety*. Als eigentliche Form wird hier ein Zylinder verwendet. Die Größe des Zylinders richtet sich nach den Default-Einstellungen. Für das Erscheinungsbild wird hier die vordefinierte Einstellung *Appearance* benutzt.

Nachfolgend wird diese Datei im Octaga-Player dargestellt.



VRML ist eine objektorientierte Sprache, jedoch spricht man eher von Knoten, als von Objekten. Es gibt 3 verschiedene Typen von Knoten: Gruppenknoten, Blattknoten und untergeordnete Knoten.
Knoten (*nodes*) haben wiederum Felder (*fields*).
Nun besteht eine VRML-Datei aus sehr vielen Knoten, die über eine Baumstruktur, den sog. Szenengraphen, in Beziehung zueinander stehen.

6.2 Gruppenknoten

Ein Gruppenknoten kann wiederum weitere Knoten enthalten, zum einen Gruppenknoten oder auch Blattknoten.
Folgende Gruppenknoten sind zu unterscheiden:

6.2.1 Group

Wie der Name schon verrät, gruppiert der Gruppenknoten mehrere Kindknoten zu einem Objekt.
Jedoch möchte man Objekte verschieben, rotieren und skalieren – somit findet der erweiterte und nun folgende Gruppenknoten *Transform* häufigere Verwendung.

6.2.2 Transform

Zugehörige Felder:
- children []
- translation 0.0 0.0 0.0 (X, Y, Z: Verschiebung)
- rotation 0.0 0.0 0.0 0.0 (X, Y, Z, Winkel der Rotation)
- scale 0.0 0.0 0.0 (X, Y, Z: Skalierung)

Diese Schreibweise, der Auflistung der zugehörigen Felder eines Knotens wird im folgenden auch so weitergeführt. Jedoch wird in der offiziellen Spezifikation eine ausführlichere Schreibweise verwendet, die hier etwas erläutert werden soll. Dieses Verständnis ist spätestens beim Abschnitt Prototypen von Bedeutung.
Die offizielle Spezifikation ist hier verfügbar:
http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-IS-VRML97WithAmendment1/

Die Erläuterung des Transform-Knotens ist hier abrufbar:
http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-IS-VRML97WithAmendment1/part1/nodesRef.html#Transform

Feldtyp	Datentyp	Feldbezeichner	Defaultwert	Wertebereich
Transform				
{				
eventIn	MFNode	**addChildren**		
eventIn	MFNode	**removeChildren**		
exposedField	SFVec3f	**center**	**0 0 0**	# (-∞,∞)
exposedField	MFNode	**children**	**[]**	
exposedField	SFRotation	**rotation**	**0 0 1 0**	# [-1,1], (-∞,∞)
exposedField	SFVec3f	**scale**	**1 1 1**	# (0,∞)
exposedField	SFRotation	**scaleOrientation**	**0 0 1 0**	# [-1,1], (-∞,∞)
exposedField	SFVec3f	**translation**	**0 0 0**	# (-∞,∞)
field	SFVec3f	**bboxCenter**	**0 0 0**	# (-∞,∞)
field	SFVec3f	**bboxSize**	**-1 -1 -1**	# (0,∞) or -1,-1,-1
}				

In unserer folgenden Dokumentation werden Feldtyp, Datentyp und der Wertebereich meist nicht genannt.

6.2.3 Exkurs: Datentypen

An dieser Stelle sollen die Datentypen ein wenig erläutert werden:

SF	SingleFieldValue, Felder bzw. Ereignisse mit einzelnen Werten	
MF	MultipleFieldValue, Werte in eckigen Klammern	
SFBool	Boolsche Variable: TRUE oder FALSE	
SFColor	3 Fließkommawerte: 0.0 bis 1.0	(oderSMColor)
SFFloat	Fließkommazahl einfacher Genauigkeit	(oder SMFloat)
SFImage	Pixel-Beschreibung eines Bildes	
SFInt32	32-Bit-Festkommazahl	(oder MFInt32)
SFRotation	3 Zahlen für die Rotationsachse + Rotationswinkel	(oder SMRotation)
SFString	Zeichenkette aus dem utf8-Zeichensatz	(oder MFString)
SFTime	Anzahl Sekunden seit dem 1.1.1970 als Fließkommazahl	(oder MFTime)
SFVec2f	Vektor: 2 Fließkommazahlen	(oder MFVec2f)
SFVec3f	Vektor: 3 Fließkommazahlen	(oder MFVec3f)

6.2.4 Exkurs: Felder und Ereignisse

Daten bzw. Werte werden in Felder gespeichert. Man unterscheidet 4 Feldtypen:
- field
- exposedField
- eventIn
- eventOut

field und *exposedField* definieren die **Ausgangswerte für die Knoten**.
Der *field*-Wert kann zur Laufzeit nicht geändert werden, das *exposedField* jedoch schon. Das *exposedField* wird somit nur bei dynamischen Szenen eingesetzt. Ist eine Szene statisch sind beide gleichwertig.

eventIn und *eventOut*, dienen der **Dynamisierung einer Szene**.
Bei dynamischen Szenen kann ein Knoten eingehende Ereignisse empfangen, welche einen *eventIn* darstellen. Im gegenteiligen Fall kann ein Knoten aber auch Signale aussenden welche als *eventOut* bezeichnet werden. Hierzu mehr im Abschnitt Animationen auf Seite 32.

Im Gegensatz zu *Group* bietet die Gruppierung mit *Transform*, noch die Möglichkeit der Translation, Rotation und Skalierung.

```
#VRML V2.0 utf8

Background
{
    skyColor 1.0 1.0 1.0   #white
}

Transform
{
  children
  [
    Shape
    {
      appearance Appearance
      {
        material Material {}
      }
      geometry Cylinder {}
    }
  ]
  translation 0 2.0 0
}
```

Hier wird der Gruppenknoten *Transform* verwendet. Diesem untergeordnet der Knoten *children*, welcher eine Liste von Kindknoten darstellt. In diesem Beispiel ist nur ein Kindknoten enthalten.
Außerdem wurde die Hintergrundfarbe auf den RGB-Wert von weiß gesetzt.
Des Weiteren wird unser Zylinder noch um den Wert 2.0 in Y-Richtung verschoben.

6.2.5 Inline

Der Inline-Knoten, stellt eine, wie auch aus anderen Skriptsprachen bekannte Anweisung dar, um eine komplette VRML-World zu inkludieren. Es bietet sich natürlich auch an wiederverwendete Elemente einzubinden.

```
Inline
{
  url "include.wrl"
}
```

6.2.6 weitere Gruppenknoten:

- Anchor
- Billboard
- Collision
- LOD
- Switch

6.3 Blattknoten (Kindknoten)

Blattknoten sind entweder eigenständige Knoten oder Kinderknoten von Gruppenknoten.
Einige wichtige Blattknoten werden unterschieden: *Shape*, *DirectionalLight*, *PointLight*, *SpotLight* und *Background*.

6.3.1 Shape

Der Knoten *Shape* ist für die Gestaltung zuständig und ist in zwei weitere Kindknoten aufgeteilt:
- appearance NULL (beschreibt das Aussehen des Objektes)
- geometry NULL (beschreibt die geometrische Form)

6.3.2 DirectionalLight

Allgemeiner Hinweis zu Lichtquellen: Lichtquellen bilden selbstständige Blattknoten, unabhängig von geometrischen Objekten.

Beim *DirectionalLight* handelt es sich um eine Lichtquelle, die parallel gerichtetes Licht aussendet, somit soll Sonnenlicht simuliert werden.
Untergeordnete Knoten sind:
- ambientIntensity 0.0 (0.0 bis 1.0, Licht von anderen Quellen)
- color 1.0 1.0 1.0 (Farbe des Lichtes mit den drei Werten für R, G und B)
- direction 0.0 0.0 -1.0 (Richtungsvektor des Lichtstrahles X, Y, Z)
- intensity 1.0 (0.0 bis 1.0, Intensität)
- on TRUE (TRUE oder FALSE, Zustand des Lichtschalters)

6.3.3 PointLight

Der Knoten PointLight beschreibt eine Lichtquelle, die sich punktförmig ausbreitet.
- ambientIntensity 0.0 (0.0 bis 1.0, Licht von anderen Quellen)
- attenuation 1.0 0.0 0.0 (Abschwächung des Lichtes)
- color 1.0 1.0 1.0 (Farbe des Lichtes mit den drei Werten für R, G und B)
- intensity 1.0 (0.0 bis 1.0, Intensität)
- location 0.0 0.0 0.0 (Position der Lichtquelle durch X, Y und Z-Werte)
- radius 100 (Radius, in dem andere Knoten beleuchtet werden)
- on TRUE (TRUE oder FALSE, Zustand des Lichtschalters)

6.3.4 SpotLight

Diese Lichtquelle breitet sich Kegelförmig aus.
Folgende Eigenschaften sind einstellbar:
- ambientIntensity 0.0 (0.0 bis 1.0, Licht von anderen Quellen)
- attenuation 1.0 0.0 0.0 (Abschwächung des Lichtes)
- beamWidth 1.570796 (Öffnungswinkel des Lichtkegels mit konstanter Intensität 0.0 bis 1.570796=90°)
- color 1.0 1.0 1.0 (Farbe des Lichtes mit den drei Werten für R, G und B)
- cutOffAngle 0.785398 (äußerer Mantel des Lichtkegels 0.0 bis Pi/2)
- direction 0.0 0.0 -1.0 (Richtungsvektor des Lichtstrahles X, Y, Z)
- intensity 1.0 (0.0 bis 1.0, Intensität)
- location 0.0 0.0 0.0 (Position der Lichtquelle X, Y, Z)
- radius 100 (Radius, in dem andere Knoten beleuchtet werden)
- on TRUE (TRUE oder FALSE, Zustand des Lichtschalters)

6.3.5 Background

Dieser Knoten stellt standardmäßig den schwarzen Hintergrund dar. Möchte man eine andere Farbe als Hintergrund kann man z.B. für einen gelben Hintergrund folgendes schreiben:

```
Background {skyColor 1.0 1.0 0.0}
```

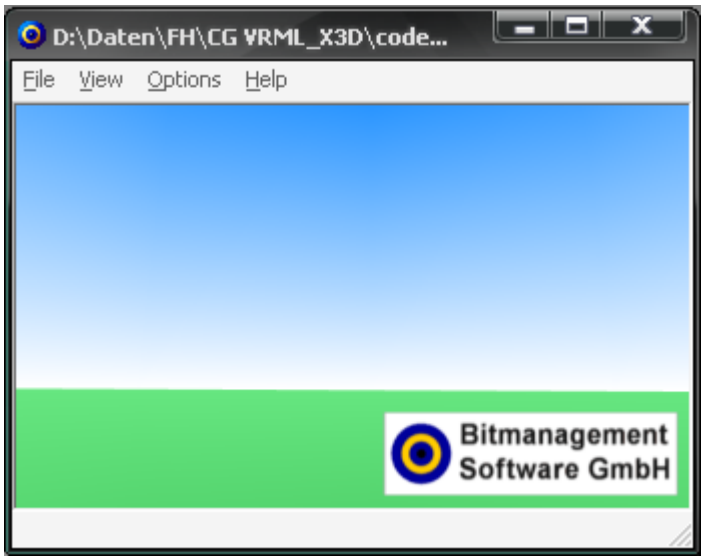
Es kann allerdings zwischen 2 Hintergründen unterschieden werden, dem Boden und dem Himmel.

- groundAngle [] (Liste von Winkelangaben)
- groundColor [] (Liste von Farbwerten für den Boden)
- skyAngle [] (Liste von Winkelangaben)
- skyColor [] (Liste von Farbwerten für den Himmel)

Im Beispiel wird der Himmel im skyColor-Knoten von oben angefangen, bei 0° dunkelblau (RGB: 0.0 0.1 0.8) über hellblau (RGB: 0.0 0.5 1.0), bei einem Winkel von 0.785, was 45° entspricht, nach weiß (RGB: 1.0 1.0 1.0) dargestellt. Die Gradangaben im Bogenmaß werden im skyAngle-Knoten definiert.
Für groundColor und groundAngle gilt entsprechendes, jedoch liegt hier ein Farbverlauf von hellgrün am Horizont nach dunkelgrün vor.

```
#VRML V2.0 utf8

Background
{
  skyColor
  [
       0.0 0.1 0.8,
       0.0 0.5 1.0,
       1.0 1.0 1.0
  ]
  skyAngle [0.785, 1.571]

  groundColor
  [
       0.1 0.4 0.2,
       0.2 0.7 0.3,
       0.4 0.9 0.5
  ]
  groundAngle [0.785, 1.571]
}
```

Hier das Beispiel aus dem BS Contact VRML/X3D Viewer.



Außerdem gibt es noch weitere Knotenattribute:
- backUrl []
- bottomUrl []
- frontUrl []
- leftUrl []
- rightUrl []
- topUrl []

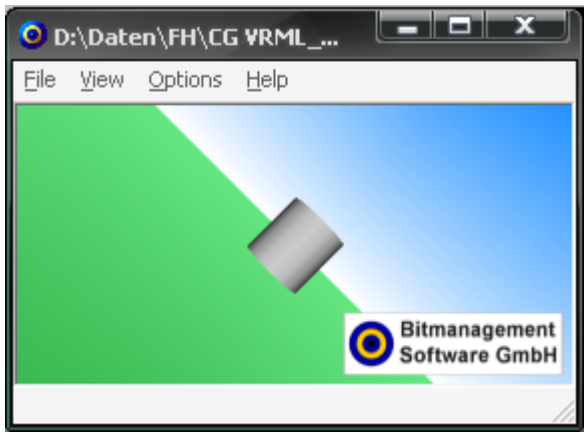
Hier können Adressen zu Grafiken angegeben werden.

6.3.6 Viewpoint

Ein Viewpoint ist ein Beobachtungspunkt bzw. eine Kameraposition innerhalb der VRML-Welt, von der aus die Welt betrachtet werden kann.
Folgende Knotenattribute können gesetzt werden:
- description " " (Zeichenkette, die den Viewpoint beschreibt)
- jump TRUE (als aktuelle Benutzersicht einstellen)
- orientation 0.0 0.0 1.0 0.0 (Drehachse 0 0 1 (X,Y und Z) und Drehwinkel um die Achse)
- position 0.0 0.0 10.0 (Betrachterposition innerhalb der Welt)

Das vorherige Beispiel wurde um folgenden Sourcecode erweitert:

```
Shape
{
   geometry Cylinder {}
   appearance Appearance
   {
      material Material {}
   }
}

Viewpoint
{
   description "Gekippte Sichtweise"
   orientation 0 0 1 0.785
}
```



Die Sichtweise bekommt ihre eigene Bezeichnung und die Drehung um die Z-Achse beträgt 45°.

6.3.7 Übersicht aller Kindknoten

- Anchor
- Billboard
- LOD
- PROTO'd children
- nodes
- ScriptShape
- Switch
- Sound
- Transform
- WorldInfo

Sensoren:

- CollisionCylinderSensor
- PlaneSensor
- ProximitySensor
- SphereSensor
- TimeSensor
- TouchSensor
- VisibilitySensor

bindbare Knoten:

- Background
- FogNavigationInfo
- Viewpoint

Interpolatoren:

- ColorInterpolator
- CoordinateInterpolator
- NormalInterpolator
- OrientationInterpolator
- PositionInterpolator
- ScalarInterpolator

Lichtquellen:

- DirectionalLight
- PointLight
- SpotLight
- Fog
- Group
- Inline

6.4 geometrische Knoten (Objektknoten I)

Hier folgen viele Standard-Formen, ein Hauptbestandteil von VRML.

6.4.1 Box

Diese Form stellt ein Quader dar, dessen Zentrum sich im Koordinaten-Ursprung befindet.
Es ist ein Knotenattribut verfügbar:

- size 2.0 2.0 2.0 (Die Abmessungen in X, Y und Z-Richtung)

6.4.2 Cone

Hier handelt es sich um einen Kegel, dessen Zentrum ebenfalls im Koordinaten-Ursprung liegt.

- bottomRadius 1.0 (Radius des Grundkreises)
- height 2.0 (Höhe des Kegels)
- side TRUE (Mantel darstellen)
- bottom TRUE (Grundkreis darstellen)

6.4.3 Cylinder

Das Zentrum liegt im Ursprung des Koordinatensystems, die Mittelachse ist standardmäßig die y-Achse.

- radius 1.0 (Radius des Cylinders)
- height 2.0 (Höhe des Cylinders)
- bottom TRUE (Grundfläche darstellen)
- top TRUE (Deckfläche darstellen)
- side TRUE (Mantel darstellen)

6.4.4 Sphere

Bei der Kugel liegt das Zentrum im Ursprung.
Sie hat lediglich das Knotenattribut:

- radius 1.0 (Der Radius)

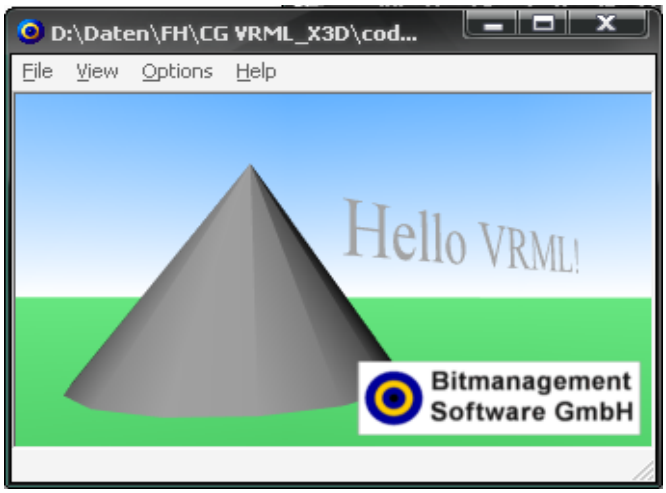
6.4.5 Text

Mit diesem Knoten kann man einfachen Text darstellen.

- string [] (Eine oder mehrere Zeichenketten in doppelten Anführungszeichen)
- fontStyle NULL (Hier kann ein FontStyle-Knoten angegeben werden, der Schriftart, -größe, ... festlegt)
- length 0.0 (Länge der Zeichenkette. Bei Werten verschieden von 0.0 führt dies zu einer Skalierung)
- maxExtent 0.0 (Obergrenze für die Ausdehnung der Zeichenketten des Knotens. Ein von 0.0 verschiedener Wert führt also gegebenfalls zu einer Skalierung.)

Ein Beispiel mit Kegel, Text und Transform-Knoten:

```
Transform
{
   children
   [
      Shape
      {
         appearance Appearance
         {
            material Material {}
         }
         geometry Text
         {
            string [ "Hello VRML!" ]
         }
      }
   ]
   translation 0 0 2.0
}
Shape
{
   appearance Appearance
   {
        material Material {}
   }
   geometry Cone
   {
        bottomRadius  2.0
        height 3.0
   }
}
```



6.4.6 ElevationGrid

Zur Darstellung von Geländeformationen.

- color NULL (Colorknoten mit Farbwerten pro Eckpunkt oder Flächenelement)
- height [] (Zweidimensionale Liste mit den einzelnen Höhenwerten)
- xDimension 0 (Anzahl der Felder in x-Richtung)
- yDimension 0 (Anzahl der Felder in y-Richtung)
- xSpacing 0.0 (Ausdehnung eines Feldes in x-Richtung)
- ySpacing 0.0 (Ausdehnung eines Feldes in y-Richtung)
- zSpacing 0.0 (Ausdehnung eines Feldes in z-Richtung)

Dem nächsten Beispiel liegt eine 5 auf 5 Einheiten große Grundfläche zu Grunde. Die Ausdehnung beträgt 1.0. Zur Orientierung: Im *height*-Knoten ist der Wert 4.0 der höchste Punkt.

```
Shape
{
   appearance Appearance
   {
        material Material {}
   }
   geometry ElevationGrid
   {
        xDimension 5
        zDimension 5
        xSpacing 1.0
        zSpacing 1.0
        height
        [
          0.0 0.0 0.0 3.0 4.0,
          0.0 1.0 1.5 3.0 3.0,
          0.0 1.5 2.5 1.5 0.0,
          0.0 1.0 1.5 1.0 0.0,
          0.0 0.0 0.0 0.0 0.0
        ]
   }
}
```

6.4.7 PointSet

Dieser Knoten definiert eine Liste von Punkten, welche zur Erzeugung von Flächen oder Linien verwendet wird.

- color NULL (optionaler Color-Knoten, mit einer Liste von Farbwerten für jeden Punkt)
- coord NULL (optionaler Coordinate-Knoten, mit einer Liste von Punkten)

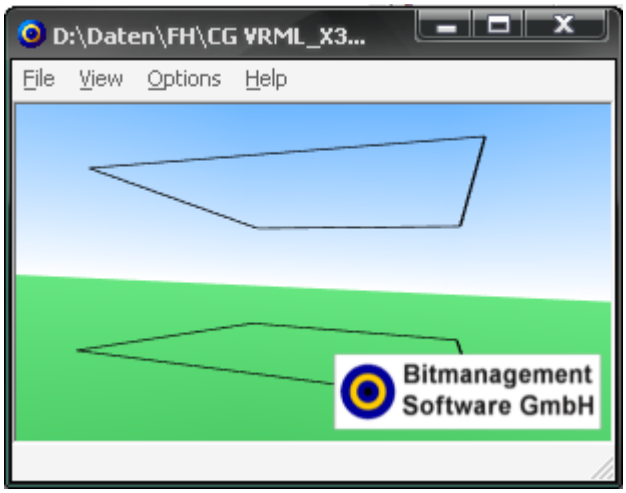
6.4.8 IndexedLineSet

Der Knoten *IndexedLineSet* legt eine Liste von Linien fest, wobei Punkte verbunden werden.

- color NULL (optionaler Color-Knoten, mit einer Liste von Farbwerten für jede Linie)
- colorIndex [] (Will man nicht für jeden Punkt eine andere Farbe benutzen, so kann man im Color-Knoten die Farben definieren und hier nur die Farbnummer den Linien zuordnen)
- coord NULL (optionaler Coordinate-Knoten, mit einer Liste von Punkten.)
- coordIndex [] (Liste von Linienzügen, die über die Nummern der Punkte beschrieben werden. Ein Linienzug endet mit -1)

Im folgenden Beispiel werden zuerst 8 Punkte definiert, die in 2 horizontalen Ebenen liegen aus denen dann anschließende 2 Rechtecke gezeichnet werden.

```
Shape
{
   appearance Appearance
   {
        material Material {}
   }
   geometry IndexedLineSet
   {
      coord Coordinate
      {
         point
         [
            #unten
            -1  0 -1, #0 hinten links
             1  0 -1, #1 hinten rechts
             1  0  1, #2 vorne links
            -1  0  1, #3 vorne rechts
            #oben
            -1  1 -1, #4 hinten links
             1  1 -1, #5 hinten rechts
             1  1  1, #6 vorne links
            -1  1  1, #7 vorne rechts
         ]
      }
      coordIndex
      [
         0, 1, 2, 3, 0, -1, # unteres Quadrat
         4, 5, 6, 7, 4, -1  # oberes  Quadrat
      ]
   }
}
```



6.4.9 IndexedFaceSet

Dieser Knoten definiert eine Liste von Flächenstücken, womit somit beliebige geometrische Objekte erzeugt werden können.

- gleiche Attribute wie *IndexedLineSet*.

zusätzlich:

- solid TRUE (Die Polygonstruktur wird bei TRUE als Festkörper behandelt, verdeckte Flächen werden nicht dargestellt. Um Objekte vernünftig drehen können, so muss der Wert auf FALSE gesetzt werden)
- convex TRUE (muss bei konkaven Objekten auf FALSE gestellt werden, sonst ist ein „hindurchschauen" nicht möglich)

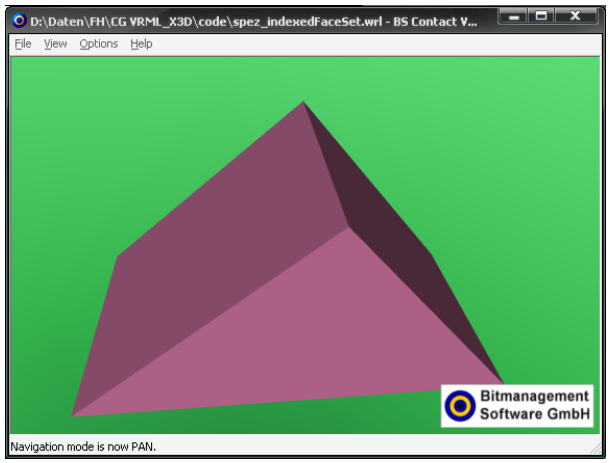
Bei der Benutzung von *IndexdFaceSet* ist folgendes zu beachten:

- Die Punkte zur Erstellung von Flächen bei *IndexdFaceSet* müssen immer, von außen gesehen, gegen den Uhrzeigersinn angeordnet werden!
- Die Vertices von einer Fläche müssen komplanar sein, d.h. auf einer Ebene liegen. Sind sie anders angeordnet kann der VRML-Player diese nicht korrekt interpretieren. Daher ist es sinnvoll, bzw. weniger fehleranfällig, nur Flächen zu erstellen, die aus 3 Vertices bestehen. Diese sind immer komplanar und man kann jeden Körpern aus Dreiecken realisieren. Siehe auch im Kapitel Beispielanwendung, „Brandenburger Tor" auf Seite 56.

Als Beispiel soll hier ein Dreiecksdach modelliert werden. Es enthält hier allerdings auch Flächen, die aus 4 Vertices bestehen.

```
#Dreieckdach
Shape
{
  appearance Appearance
  {
    material Material
    {
      diffuseColor .9 .5 .7
    }
  }
  geometry IndexedFaceSet
  {
    solid FALSE
    convex FALSE
    coord Coordinate
    {
      point
      [
        #Ebene hinten
        0 0 0, #0
        1 1 0, #1
        2 0 0, #2

        #Ebene vorne
        0 0 1, #3
        1 1 1, #4
        2 0 1, #5
      ]
    }
    coordIndex
    [
      #Verbinden der Punkte zu Flächen
      #Ende des Polygonzugs mit -1

      #linkes Dach
      0, 1, 4, 3, 0, -1,
      #rechtes Dach
      1, 2, 5, 4, 1, -1,
```

```
      #Boden
      0, 2, 5, 3, 0, -1,
      #hinten
      0, 1, 2, 0, -1,
      #vorne
      3, 4, 5, 2, -1,
    ]
  }
}
```



6.4.10 Extrusion

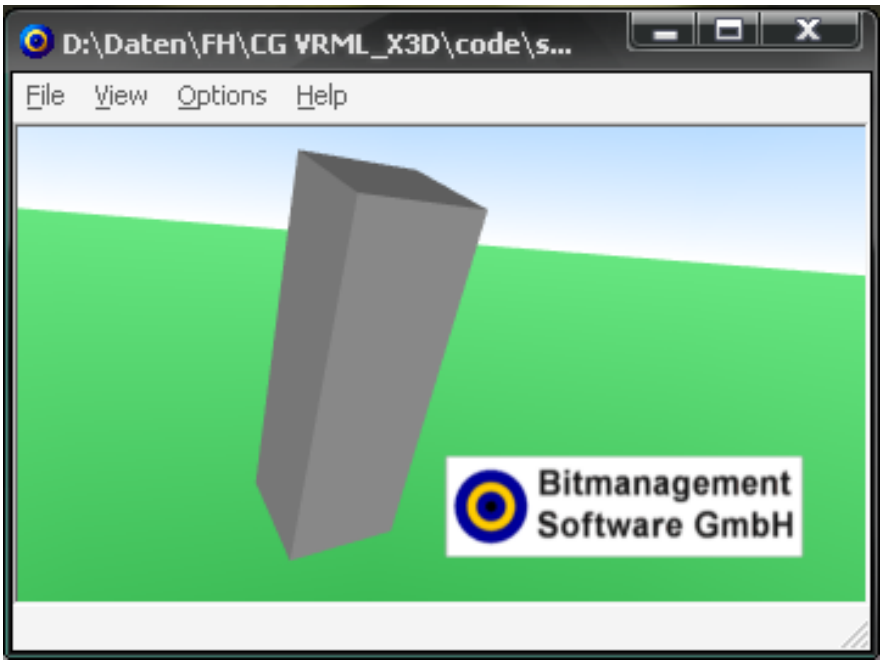
Grundlage bildet eine Fläche, die durch einen Linienzug erzeugt wird. Diese Fläche wird durch den Raum verschoben und es resultiert ein dreidimensionaler Körper.

- crossSection [1 1, 1 -1, -1 -1, -1 1, 1 1] (Der Linienzug, hier ein Quadrat, der als Querschnitt für das Objekt dient)
- spine [0 0 0, 0 1 0] (Linienzug, der die Verschiebung beschreibt. Beliebig viele Punkte sind möglich)
- beginCap TRUE (TRUE: Anfangsdeckel darstellen; FALSE: nicht darstellen)
- endCap TRUE (TRUE: Enddeckel darstellen; FALSE: nicht darstellen)
- scale [1 1] (Legt für jeden Verschiebungsschritt die Skalierungsfaktoren fest)
- convex TRUE (Bei TRUE ist der Körper nach außen gewölbt)
- creaseAngle 0.5 (Ist der Winkel zwischen zwei Flächenstücken kleiner als dieser Wert, so wird weich schattiert, was den Knick glättet. Ansonsten wird hart schattiert)
- solid TRUE (Bei TRUE gilt der Körper als massiv)
- orientation [0 0 1 0] (Beschreibt eine Drehung für jeden der Verschiebungsschritte. Wird nur ein Wert angegeben, soll gilt er für alle Schritte)

Im nachfolgenden Beispiel wurde aus einem Quadrat ein Quader erstellt.

```
Shape
{
  appearance Appearance
  {
       material Material {}
  }
  geometry Extrusion
  {
    solid FALSE
      crossSection
      [
        1 1,
        2 1,
        2 2,
        1 2,
```

```
            1 1
         ]
         spine
         [
            0 0 0,
            0 3 1
         ]
   }
}
```



6.5 Hilfsknoten (Objektknoten II)

Durch Hilfsknoten werden die Eigenschaften von Objekten beschrieben.

6.5.1 Color

Definition einer Farbe im RGB-Farbmodell

- color[] (3 Werte zwischen 0.0 und 1.0)

6.5.2 Coordinate

Festlegung von Punktkoordinaten z.B. für die Knoten *PointSet*, *IndexedLineSet* und *IndexedFaceSet*.

- point[] (Liste von Punkten, jeweils 3 Gleitkommazahlen)

6.5.3 Normal

Hiermit kann ein Normalenvektor gesetzt werden, welcher z.B. für die Schattierungsberechnung verwendet werden kann.

- vector[] (Der Normalenvektor sollte ein Einheitsvektor sein)

6.5.4 TextureCoordinate

Festlegung des Mappings von Texturen auf Oberflächen von Objekten.

- point[] (Texturkoordinaten, bestehend aus 2 Gleitkommazahlen)

6.6 Knoten zum Aussehen von geometrischen Objekten (Objektknoten III)

6.6.1 Appearance

Dieser Knoten beschreibt das Aussehen von geometrischen Objekten.

- material NULL (Möglichkeit der Einbindung eines Materialknotens, siehe nächster Punkt)
- texture NULL (Einbinden einer Textur; z.B. *ImageTexture*, *MovieTexture*, *PixelTexture*)
- textureTransform NULL (Operationen mit der Textur, bevor Sie gemappt wird)

6.6.2 Material

Hiermit können die Eigenschaften Leuchtkraft, Transparenz oder Reflexion festgelegt werden.

- ambientIntensity 0.2 (diffuse Lichtreflexion / Umgebungslicht)
- diffuseColor .8 .8 .8 (Farbe des vom Objekt reflektierten Lichts)
- emissiveColor 0 0 0 (Farbe des vom Objekt abgestrahlten Lichts)
- shininess 0.2 (Glanz)
- specularColor 0 0 0 (Reflektion in RGB-Farbwerten)
- transparency 0 (Grad der Transparenz)

6.6.3 ImageTexture

Es können jpg, png oder gif Dateien als Textur benutzt werden

- url[] (Liste von Bilder, das erste verfügbare Element wird genutzt)
- repeatS TRUE (Wiederholung der Textur in s-Richtung)
- repeatT TRUE (Wiederholung der Textur in t-Richtung)

6.6.4 MovieTexture

Dies entspricht der *ImageTexture* außer, dass hier ein Video im MPEG1-Format auf ein Objekt gelegt werden kann. Zusätzlich ist noch folgendes Attribut verfügbar:

- loop FALSE (Wiederholung des Videos bei TRUE)

6.6.5 PixelTexture

Hiermit können einfache Punktmuster erzeugt werden

- image 0 0 0 (Erste beiden Zahlen bestimmen Breite und Höhe des Bildes. Die dritte Zahl den Farbmodus)
- repeatS TRUE (Wiederholung der Textur in s-Richtung)
- repeatT TRUE (Wiederholung der Textur in t-Richtung)

6.6.6 TextureTransformation

Steuerung der Abbildung der Textur auf dem Objekt.

- center 0 0 (Festlegen des Ursprungs für die Skalierung und die Rotation)
- rotation 0 (Winkel der Rotation)
- scale 1 1 (Skalierungsfaktoren in s- bzw. t-Richtung, z.B. halbiert hier ein Wert von 2 die Größe der Textur)
- translation 0 0 (Verschiebung)

Hier ein Beispiel zur Verwendung einer Textur:

```
Shape
{
  appearance Appearance
  {
    material Material
    {
        diffuseColor 0.8 0.8 0.8
    }
    texture ImageTexture
```

```
    {
        url "http://www.google.de/intl/de_de/images/logo.gif"
    }
    textureTransform TextureTransform
    {
      scale 4 2   #Abbildung 4 Mal in s-Richutng (horizontal)
                  #und 2 Mal in t-Richtung (senkrecht)
    }
  }
  geometry Cylinder
  {
    radius 3
    height 4
  }
}
```

Ein Problem bei Texturen ist, dass man nicht auf jeder Fläche des Objektes die Textur unterschiedlich skalieren kann. So wäre es hier wünschenswert dem Mantel und dem Deckel bzw. Boden unterschiedliche Skalierungswerte zuzuweisen, da so eine Fläche, hier der Deckel, recht verzerrt wirkt. Abhilfe können hier nur die aufwendigere Verwendung von *IndexedFaceSets* liefern.

6.7 Instantiierung mit DEF und USE

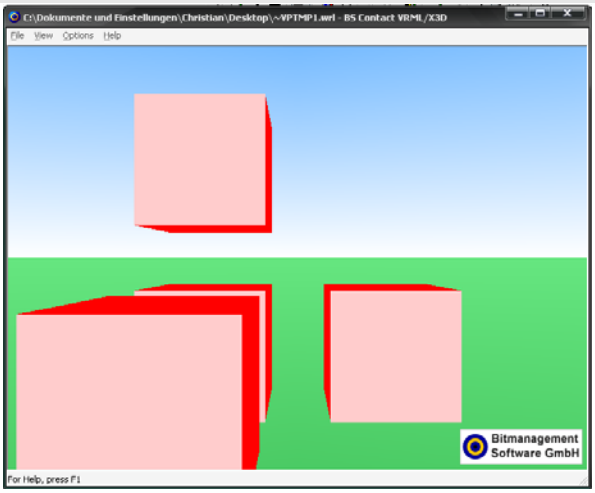
In Szenen werden Objekte sehr schnell mehrmals verwendet. Diese Objekte brauchen nicht mehrmals verwendet werden, sondern können einfach referenziert werden. Es werden also Instanzen dieser Objekte gebildet.

Es muss zuerst ein Objekt mittels DEF bezeichnet werden.
Über USE wird dieses Objekt dann referenziert.

Im folgenden Beispiel wird ein erstelltes Material mit der Bezeichnung Rot referenziert:

```
DEF Rot Appearance
{
  material Material
  {
    emissiveColor 1 0 0
  }
}

DEF Kasten Shape
{
  appearance USE Rot geometry Box
```

```
  {
     size 2 2 2
  }
}

Transform
{
  translation 3 0 0
  children USE Kasten
}

Transform
{
  translation 0 3 0
  children USE Kasten
}

Transform
{
  translation 0 0 3
  children USE Kasten
}
```



6.8 Prototypen

Effektiver jedoch als einfache Definitionen ist die Verwendung von Prototypen. Bei diesen können sich die einzelnen Instanzen sogar in Werten unterscheiden, die als Parameter übergeben werden.

Ein Prototyp definiert sich so:

```
PROTO <prototypname>
[
  <Schnittstelle>  # mögliche Datentypen s. Seite 13
]
{
  <Implementierung>
}
```

Einführendes Beispiel:

Die Datentypen, wie *SFVec3f* werden auf Seite 13 erläutert.

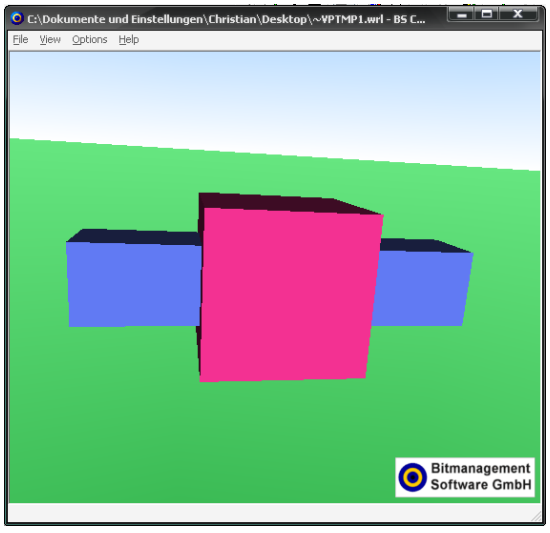
```
PROTO Kasten
[
  # Defaultwerte:
  field SFVec3f groesse 2 2 2
  field SFColor farbe   1 .2 .6
]

{
```

```
  Shape
  {
    appearance Appearance
    {
      material Material
      {
        diffuseColor IS farbe
      }
    }

    geometry Box
    {
      size IS groesse
    }

  }
}

# Instantiierung mit Defaultwerten (violetter Quader):
Kasten{}

# Instantiierung mit Übergabeparameter (blauer Quader):
Kasten
{
  groesse 1 1 5
  farbe .4 .5 1
}
```

Die Defaultwerte werden angewandt, wenn bei der Instanziierung keine Parameter übergeben werden.

Im Implementierungsteil werden die Bezeichner aus dem Interfaceteil wie folgt aufgerufen:

```
<feld> IS <parameter>
```



blauer Quader (parametrisiert), violetter Quader (default)

Alle auf Seite 18 beschriebenen Primitive haben ihren Schwerpunkt, d.h. ihr lokales Koordinatensystem im Zentrum. Nun ist dies aber nicht immer sehr praktisch, wenn man z.B. Objekte oder ganze Knoten zueinander positionieren möchte wäre es oft komfortabler das lokale Koordinatensystem z.B. in der unteren Ecke eines Objektes zu haben.

Im folgenden Beispiel wird das **lokale Koordinatensystem** in die **untere hintere linke Ecke** des Quaders verschoben.

```
PROTO BoxLokal
[
  field   SFVec3f     verschiebung 0 0 0
  field   SFVec3f     skalierung 1 1 1
  exposedField SFRotation   rotation 0 0 1 0
]
{
   Transform
   {
      children
      [
         Transform
         {
            children
            [
               Shape
               {
                  appearance Appearance
                  {
                     material Material { }
                  }
                  geometry Box
                  {
                     size 1 1 1
                  }
               }
            ]
            translation 0.5 0.5 0.5
         }
      ]
      scale IS skalierung
      translation IS verschiebung
   }
}
```

Um dies zu realisieren werden 2 Transform-Knoten ineinander geschachtelt. Der Quader ist hier ein Würfel und hat die Seitenlänge 1 in die X, Y und Z-Richtung. Der Quader wird somit um 0.5 in jede Richtung über das translation-Attribut verschoben. Nun brauchen wir aber zusätzlich noch einen zweiten Transform-Knoten um den Prototypen, der hier als „BoxLokal" bezeichnet wird, von außen auch wieder beliebig verschieben zu können. Nun jedoch mit dem lokalen Koordinatensystem in der unteren hinteren linken Ecke.

6.9 Externe Prototypen

Bei komplexen Projekten ist es sinnvoll die Prototypen in einer Datei zu sammeln. Möchte man nun einen Prototyp in einer anderen Datei verwenden, so wird die EXTERNPROTO-Zeile eingesetzt bevor der Prototyp wie bekannt benutzt werden kann.

Datei des Prototyps: extern.wrl

```
#VRML V2.0 utf8

PROTO zylinder
[
  field SFFloat Hoehe  4
  field SFFloat Radius 2
  field SFColor Farbe  0 .4 .6
]
{
  Shape
  {
     appearance Appearance
     {
        material Material
        {
           diffuseColor IS Farbe
```

```
      }
    }

    geometry Cylinder
    {
      height IS Hoehe
      radius IS Radius
    }
  }
}
```

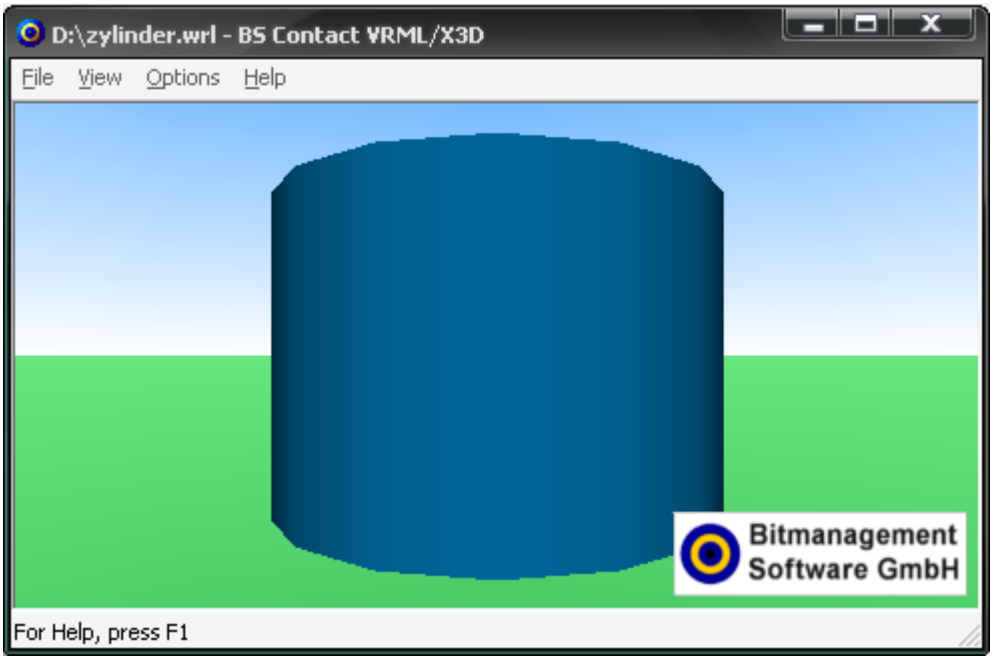
Aufzurufende Datei: zylinder.wrl

```
EXTERNPROTO zylinder
[
  field SFFloat Hoehe
  field SFFloat Radius
  field SFColor Farbe
] "extern.wrl#zylinder"

zylinder
{
  dRadius 3
}
```

Anmerkungen:

- Verwendet man externe Prototypen, so muss das entsprechende Interface in die jeweilige Datei mit aufgenommen werden. Die Defaultwerte dürfen aber nicht mit angegeben werden.
- Es brauchen nur die Felder angegeben werden, die von der Instanz als Parameter mit übergeben werden. So hätten im Beispiel auf die Angabe von *dHoehe* und *dFarbe* verzichtet werden können.
- Die Prototypdatei muss nicht auf dem gleichen Rechner liegen. So kann der Pfad zur Datei auch eine beliebige URL sein.
- Der Bezeichner nach EXTERNPROTO ist frei wählbar und muss nicht zwingend dem aus der Prototypdatei entsprechen.

Der Zylinder wird durch den Prototyp aus extern.wrl definiert
und in zylinder.wrl verwendet



6.10 Animationen

VRML ermöglicht es erstellte Objekte vielfältig zu animieren.

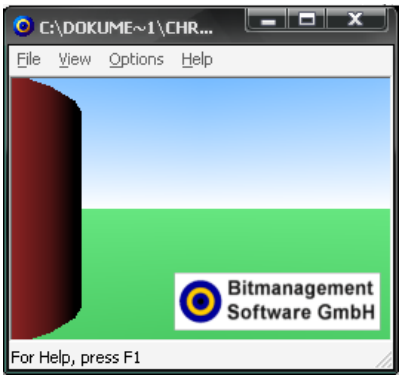
Die Bestandteile einer Animation sind:
- Objekt inkl. Transform-Knoten
- Sensor, der mit Ereignissen agiert
- Interpolator, der Zwischenschritte berechnet
- Routen zwischen Objekten

Bei geometrischen Objekten werden in der Regel folgende Eigenschaften animiert:
- translation
- rotation
- scale

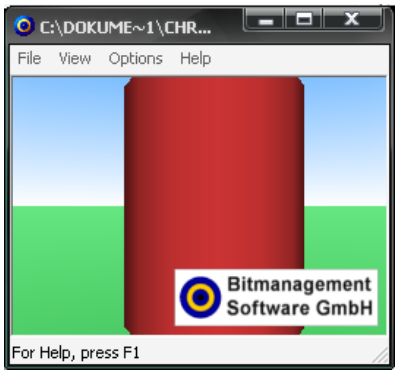
Beispiel (wiederholende horizontale Bewegung eines Zylinders)

```
DEF Saeule Transform
{
  children
  [
    Shape
    {
       appearance Appearance
       {
         material Material
         {
           diffuseColor .8 .2 .2
         }
       }

       geometry Cylinder
       {
         height 5
         radius 2
       }
    }

  ]
  translation 5 0 0
}

DEF Zeit TimeSensor
{
  cycleInterval 10.0
  loop TRUE
}


DEF Interpolator PositionInterpolator
{
  key [ 0.0 0.5 1.0 ]
  keyValue
  [
    0  0 0,
    10 0 0,
    0  0 0
  ]
}

ROUTE Zeit.fraction_changed TO Interpolator.set_fraction
ROUTE Interpolator.value_changed TO Saeule.set_translation
```

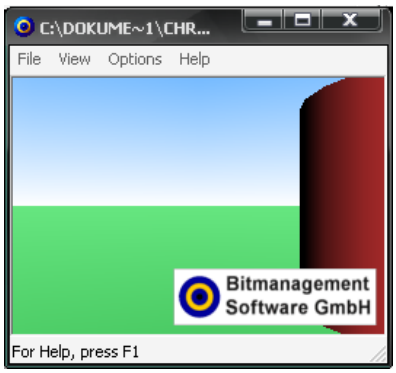
Animationstatus:



0% / 100%



25% / 75%



50%

6.10.1 Erläuterung

Der Zeittakt wird durch *cycleInterval* festgelegt. In dem Beispiel dauert ein Durchlauf 10 Sekunden und wiederholt sich permanent.

Der Interpolator berechnet die Zwischenwerte für die Animation:
Bei 0 wird 0 0 0,
bei 0.5 wird 1.0 0 0 ,
bei 1 wird 0 0 0 zurückgeliefert.

Wird beispielsweise der mittlere Wert von 0.5 auf 0.8 erhöht, wird die Animation in der Bewegung von links nach rechts verlangsamt und von rechts nach links stark erhöht.

Gekoppelt werden die drei Objekte über die zwei Routen. Die Uhr liefert über das Feld *fraction_changed* einen Wert zwischen 0.0 und 1.0 zurück. Dieser Wert wird an das Feld *set_fraction* des Interpolators übergeben, der daraufhin einen neuen Ausgabewert errechnet und über sein Feld *value_changed* an das Feld *set_translation* des Objektes Zylinder übergibt.

6.10.2 Routen

Zu jedem Feld in VRML gehört eine Zugriffsart.

Felder können entweder als *field, exposedField, eventIn, eventOut* definiert werden.

Bisher haben wir lediglich *field* verwendet. Dieses hat keinerlei einwirkende Ereignisse. *ExposedField* kann Ereignisse empfangen als auch auslösen. Bei *eventIn* und *eventOut* ist jeweils eine Richtung möglich.

Wenn ein Feld Ereignisse empfängt, so kann der aktuelle Wert über *set_name* geändert werden.
Können Ereignisse auch ausgelöst werden, so erfolgt das über *name_changed*.

Eine Route verknüpft die Felder, die Ereignisse auslösen mit den Feldern, die diese empfangen können.

Ein Feld ist aber nicht nur auf eine Route beschränkt. So können mehrere Routen von einem Feld ausgehen als auch empfangen werden.

6.10.3 Interpolatoren

Interpolatoren sind alle gleich aufgebaut, sie unterscheiden sich lediglich in der Art der Werte in der Liste auf der Ausgabenseite.

key[]
Liste mit Werten (z.B. Zeitwerte) auf der Eingabeseite.

KeyValue[]
Liste mit Werten auf der Ausgabenseite. Die Art der Werte hängt von den verwendeten Interpolator ab (im Beispiel translation). Die Anzahl muss mit dem des *key*-Feldes übereinstimmen.

Folgende Interpolatoren stehen zur Verfügung:
- PositionInterpolator (Änderung der Position)
- OrientationInterpolator (Änderung der Rotation)
- ColorInterpolator (Änderung der Farbe)
- Scalarinterpolator (Änderung der Transparenz)

6.10.4 Sensoren:

Den Sensor *TimeSensor* haben wir schon im Beispiel verwendet. Dieser reagiert auf die Systemuhr. Darüber hinaus gibt es aber noch weitere Sensoren, die auf die unterschiedlichsten Ereignisse reagieren können.

TouchSensor:
Der *TouchSensor* reagiert auf Mausaktionen, wie z.B. Mausklick oder MausRollOver.

VisibilitySensor:
Der *VisibilitySensor* ermittelt, ob ein quadratischer Bereich noch vollständig oder teilweise sichtbar ist. Sollte er nicht mehr sichtbar sein, kann dieser automatisch abgeschaltet werden.

ProximitySensor:
Der *ProximitySensor* kann feststellen, ob der Betrachter in einen fest definierten Bereich eingetreten ist.

PlaneSensor:
Mittels dem *PlaneSensor* können Objekte mit der Maus in der XY-Ebene interaktiv verschoben werden. So lassen sich hiermit beispielsweise Türen realisieren, die geöffnet werden können.

CylinderSensor:
Der *CylinderSensor* reagiert, wie der PlaneSensor, auf Mausereignisse. Allerdings werden Objekte nicht auf der Ebene translatiert, sondern um die Y-Achse rotiert.

SphereSensor:
Der *SphereSensor* erlaubt freie Drehungen von Objekten um den Koordinatenursprung.

6.11 JavaScript und Java in VRML

Quellen:
[Spezifikation6]

Viele Dinge, die man mit VRML realisieren will, sind zu komplex, um sie mit Sensoren und Interpolatoren zu erstellen. Für diesen Fall kann man mit dem *Scriptnode* andere Programmiersprachen einfügen und nutzen. Die Programmiersprachen können den *eventIn*, den *eventOut* oder *exposedFields* von definierten Knoten beeinflussen.

Formaler Aufbau des Scriptnode:

```
Script
{
  exposedField MFString url           []
  field        SFBool   directOutput  FALSE
  field        SFBool   mustEvaluate  FALSE
  eventIn      eventTypeName eventName
  field        fieldTypeName fieldName initialValue
  eventOut     eventTypeName eventName
}
```

Das Feld *url* kann zu einem externen Scriptfile verweisen, es kann aber auch das Script direkt hier eingefügt werden.

Die Programmiersprachen, die verwendet werden können sind **Javascript** und **Java**.

6.11.1 JavaScript in VRML

Im folgenden werden wir näher auf Javascript eingehen.

Javascript wird folgendermaßen initialisiert:

```
url „javascript: ...
```

Bei älteren VRML-Playern (z.B. Cosmo 1.02) wird Javascript nicht erkannt. Für diese muss folgender Code verwendet werden:

```
url: „vrmlscript: ...
```

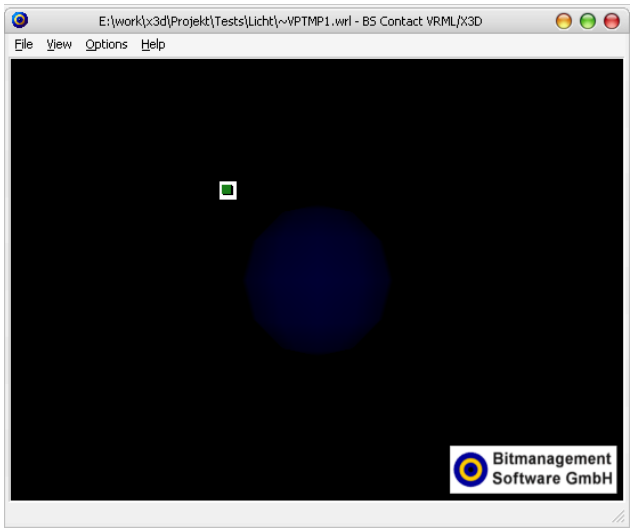
Um sicher zu gehen kann der Code dupliziert werden und mit beiden Wegen initialisiert werden. Der Browser wählt dann automatisch aus.

Als einführendes Beispiel zeigen wir exemplarisch, wie mit Javascript und dem *TouchSensor* ein Schalter realisiert wird. Es zeigt einen Schalter, der eine Lichtquelle ein schaltet (aus wäre analog zu realisieren).

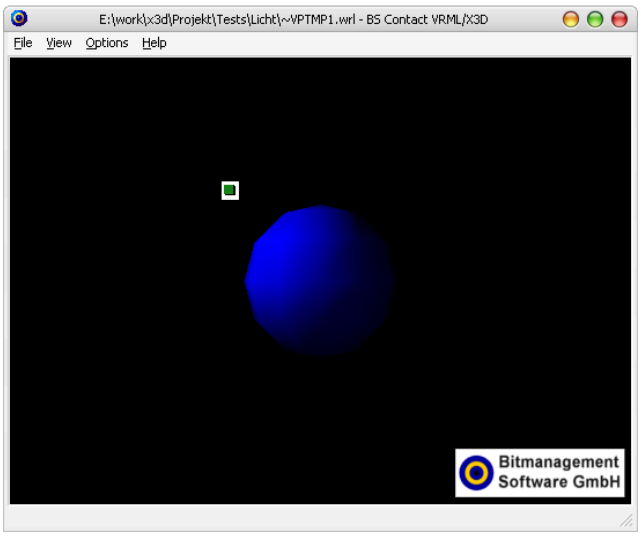
Wir werden im Beispiel nur auf die relevanten Bereiche eingehen:

```
# Der touchsensor, um das Licht an zu machen
Transform
{
  translation -0.5 0.5 4
  children
  [
     DEF LightONSwitch TouchSensor{}
     Inline { url ["lichtschalter.wrl"] }
  ]
}

# Zu Beginn ist die Lichtquelle aus
DEF LIGHT PointLight
{
   on FALSE
   location -3 2 2
}
# Ein Beispielobjekt, dass von der Lichtquelle beleuchtet werden soll
Shape
```

```
{
   appearance Appearance
   {
      material Material
      {
         diffuseColor 0.0 0.0 0.2
         shininess 1
      }
   }
   geometry Sphere
   { radius 1 }
}

# Lichtschalter wird ein geschaltet
DEF lightONScript Script
{
   eventIn SFBool   lightONIsActive
   eventOut SFBool  lightIsON

   url [
        "javascript: function lightONIsActive(active)
        {
           lightIsON = TRUE;
        }",

        # Für ältere VRMLPlayer
        "vrmlscript: function lightONIsActive(active)
        {
           lightIsON = TRUE;
        }"
     ]
}

# Aktiviert lightON des EventIn vom lightOnScript
ROUTE LightONSwitch.isActive TO lightONScript.lightONIsActive
ROUTE lightONScript.lightIsON TO LIGHT.set_on
```



Licht aus



Licht an

6.11.2 Java in VRML

Quelle:
[Spezifikation7]

Die Schnittstelle die VRML zu Java bietet ist ähnlich der zu Javascript. Es wird der gleiche Scriptnode verwendet, wie auf Seite 34 erläutert. Im *url* Feld des Scriptnodes wird bei Java auf den Bytecode einer Java-Class-Datei verwiesen.
Damit Java VRML verwenden kann benötigt man spezielle Packages, die es zu importieren gilt. Die Packages sind: *vrml.*; vrml.field.*; vrml.node.**

Diese Packages gehören nicht zum Standardumfang von Java, muss allerdings in javafähigen Browsern zum Sprachumfang gehören, sonst kann VRML mit Java nicht benutzt werden.

Ein Java-Programm hat nun folgenden Aufbau:

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class MyVRMLClass extends Script
{
   ...
}
```

wichtige Java-Klassen:
Diese Klassen sind von *java.lang* abgeleitet.

- vrml.Event (Methoden, die den Namen, den Zeitstempel oder den Wert eines Ereignisses liefern)
- vrml.Browser (Methoden zur Kommunikation mit dem Browser)
- vrml.Field (Methoden zur Konvertierung der Datenformate zwischen VRML und Java
  vrml.MField (SingleField & MultipleField).)
- vrml.ConstField (Lesen / Schreiben von Feldinhalten über getValue() / setValue())
  vrml.ConstMField
- vrml.BaseNode (direkter Zugriff auf eventIn / eventOut Felder (über die Angabe des Namens))
  - vrml.node.Node
  - vrml.node.Script

wichtige Java-Methoden:
Zum Programmstart wird einmalig die Methode *initialize()* aufgerufen.
Der Zugriff auf Felder und Events erfolgt über die Methoden *getField()*, *getEventOut()* und *getEventIn()*. Als Übergabeparameter wird der Name verwendet.
Um Ereignisse an einen Skriptknoten zu senden werden die Methoden *processEvent()* und *processEvents()* benutzt. Danach wird immer die Methode *eventsProcessed()* aufgerufen.

Bemerkung:
Die Vorgehensweise bei Java ähnelt die von Javascript. Jedoch bleibt das Problem, dass heutige Browser die VRML-Klassen trotz Java-Fähigkeit nicht installiert haben. Eine nachträgliche Installation ist uns bei unseren Tests leider nicht gelungen. So müsste das entsprechende Java Archiv (jar-Datei) nachinstalliert werden um z.B. das oben genannte *package vrml* benutzbar zu machen.
Da erstens dieses Java Archiv meist nachinstalliert werden muss und des weiteren Java-Applets aus der Mode gekommen sind, sollte man also eher damit rechnen, dass Benutzer Applikationen von VRML in Verbindung mit Java nicht betrachten können und daher von einer Verwendung absehen.

7 X3D

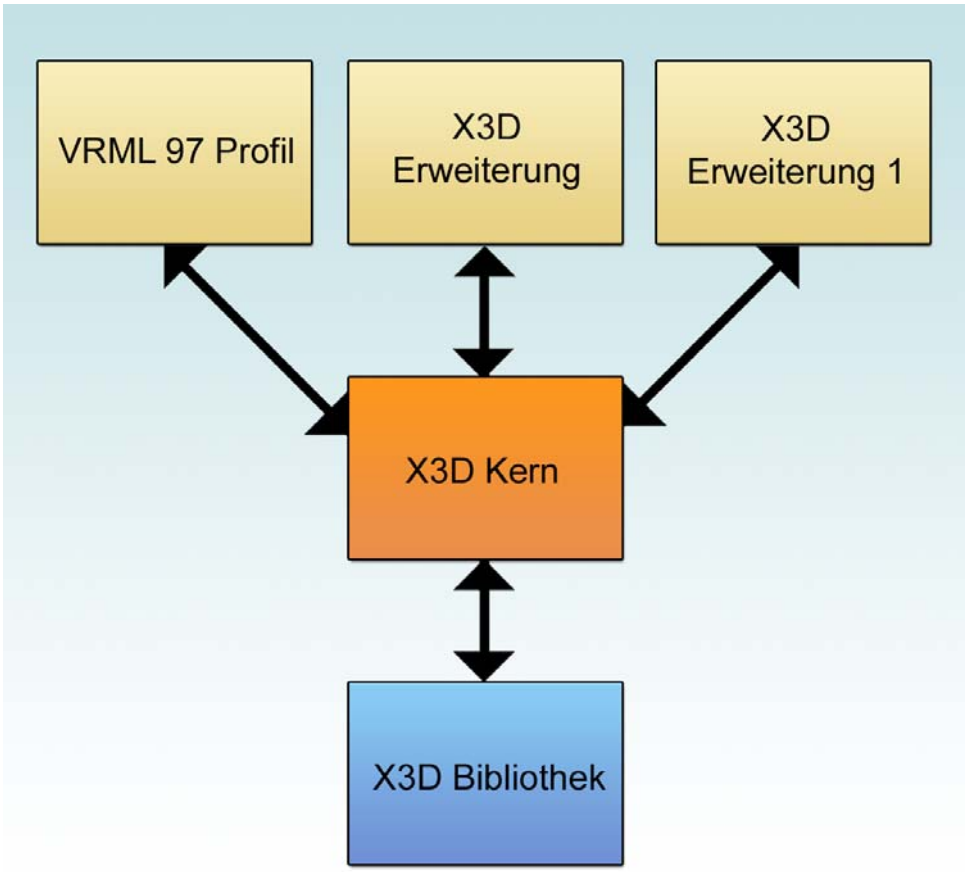
7.1 Aufbau und Vergleich zu VRML

Quelle: [Spezifikation4]

Im August 2001 hat das Web3D-Konsortium das auf der Beschreibungssprache XML basierende DTD eXtensible 3D (X3D) vorgestellt. Die Unterstützung von XML erlaubt es in 3D Szenen den Inhalt der Szene von der Darstellung getrennt zu betrachten. Somit wird mehr Flexibilität erreicht.
Ziel ist es mittels X3D alle 3D-Standards zu vereinen. Der Standard ist offen, plattformunabhängig und lizenzgebührenfrei.

Gründe für die Weiterentwicklung von VRML waren auch folgende Nachteile des alten Standards:
- VRML-Daten werden im ASCII Format gespeichert, wodurch zwar leicht veränderbar sind, jedoch auch viel Speicherplatz benötigen. Dateien wachsen mit ihrer Komplexität schnell an.
- VRML Inhalte sind erst nach dem vollständigen Download sichtbar.
- Verschiedene Browser-PlugIns stellen die gleiche VRML Datei unterschiedlich dar.

Die Beschreibung der Szene als hierarchischer Szenengraph, sowie die Möglichkeiten der Animation und Interaktion stehen, wie aus VRML bekannt, weiter zur Verfügung. Der Unterschied ist jedoch der modulare Aufbau von X3D.



Der **X3D Kern** enthält nur die nötigsten Spezifikationen. Er ähnelt zwar VRML, ist jedoch leistungsfähiger.

Das **VRML 97 Profil** enthält alle Eigenschaften von VRML. Somit ist X3D ohne Einschränkungen abwärts kompatibel. Konverter können somit X3D Dateien in VRML Dateien umwandeln und umgekehrt.

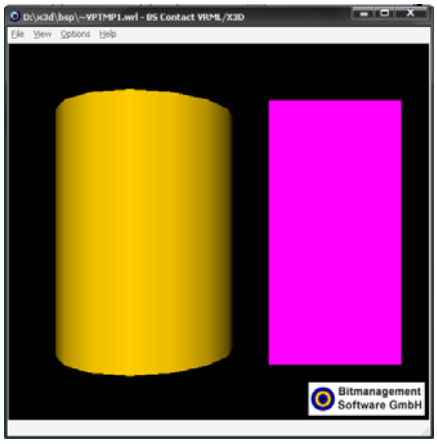
In der **X3D Erweiterung** sind Neuerungen enthalten, wie beispielsweise NURBS und Streaming.

Einer der größten Vorteile von X3D ist, dass es **unbegrenzt erweiterbar** ist. So können zu den Stammfunktionen Profile angelegt werden, die speziell auf die Anwender ausgerichtet sind. Beispiel für Erweiterungen sind z.B. H-Anim (humanoid animation standard) oder SAI (Scene Access Interface).

Zum Vergleich von VRML und X3D der funktionell identische Code in beiden Sprachen:

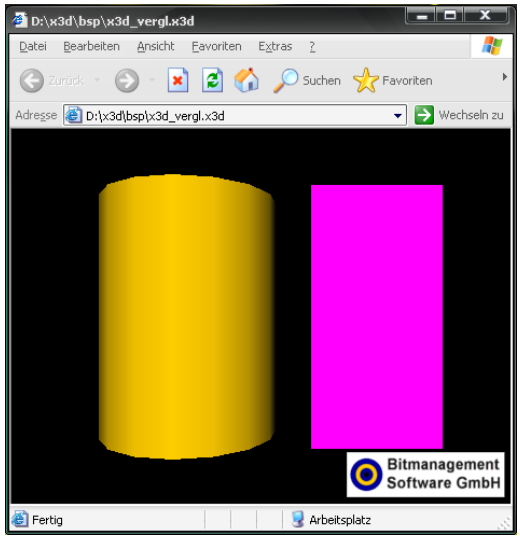
VRML:

```
#VRML V2.0 utf8

Transform
{
  children
  [
    Shape
    {
      appearance Appearance
      {
        material Material
        {
          diffuseColor 1 .8 0
        }
      }
      geometry Cylinder
      {
        radius 3
        height 8
      }
    }
  ]
}

Transform
{
  translation 6 0 0
  children
  [
    Shape
    {
      appearance Appearance
      {
        material Material
        {
          diffuseColor 1 0 1
        }
      }
      geometry Box
      {
        size 4 8 4
      }
    }
  ]
}
```



X3D:

```
<?xml version="1.0" encoding="UTF-8"?>

<X3D>
  <Scene>
    <Transform>
      <Shape>
        <Appearance>
          <Material
            diffuseColor="1 .8 0"
          />
        </Appearance>
        <Cylinder
          radius="3"
          height="8"
        />
      </Shape>
    </Transform>

    <Transform translation="6 0 0">
      <Shape>
        <Appearance>
          <Material
            diffuseColor="1 0 1"/>
        </Appearance>
        <Box
          size="4 8 4"
        />
      </Shape>
    </Transform>
  </Scene>
</X3D>
```



7.2 X3D Spezifikation

Quelle: [Spezifikation5]

Es gibt eine ganze Reihe von Spezifikationen rund um X3D. Hier werden sie nun kurz erläutert um später den Zusammenhang der Aussagen zuordnen zu können.

ISO/IEC 19775:2004 — *Extensible 3D (X3D)* 2004-12-01

- Definiert die abstrakte Funktionsspezifikation für das X3D-Framework und die Definitionen der standardisierten Bestandteile und der Profile.
- Definiert das Scene Access Interface (SAI) das benutzt werden kann, um auf X3D-Welten innerhalb von anderen Welten oder von den externen Programmen her einzuwirken.

ISO/IEC 19775-1:2004/FPDAM Amd 1 — *X3D Abstract Functionality — Amendment 1: Additional functionality* 2005-09-07

- Definiert einige Verbesserungen und Änderungen an der abstrakten Spezifikation X3D, ISO/IEC 19775 betrifft: CAD Interchange profile, Programmable shaders Komponente, 3D texturing Komponente und Cubic environment texturing Komponente.

ISO/IEC 19776:2005 — *X3D encodings (XML and Classic VRML)* 2005-07-25

- Definiert die Kodierung von XML (Extensible Markup Language) und klassischem VRML des X3D Frameworks.

ISO/IEC CD 19776-3 — *X3D encodings — Part 3: Binary encoding* 2004-12-01

- Definiert die Zuordnung zwischen abstrakten Objekten in X3D und einer spezifischen X3D-Kodierung in einer kompakten binären Form.

ISO/IEC FDIS 19777:2005 — *X3D language bindings (ECMAScript)* 2005-03-16

- Definiert die Anbindung zwischen Diensten in der X3D Architektur und der ECMAScript Programmiersprache, für die X3D interne Darstellung (Script nodes) und externen Programmzugriff.

ISO/IEC FDIS 19777:2005 — *X3D language bindings (Java)* 2005-03-16

- Definiert die Anbindung zwischen Diensten in der X3D Architektur und der Programmiersprache Java, für die X3D interne Darstellung (Script nodes) und externen Programmzugriff.

ISO/IEC FDIS 19777:2005 — *X3D language bindings (ECMAScript and Java)* 2005-03-16

- Definiert die Anbindung zwischen Diensten in der X3D Architektur und den Programmiersprache ECMAScript und Java, für die X3D interne Darstellung (Script nodes) und externen Programmzugriff.

7.2.1 Eine Spezifikation im Detail:

Die **ISO/IEC 19775:2004 — *Extensible 3D (X3D)*** Spezifikation vom Dezember 2004 besteht aus zwei Teilen:

Teil **eins "Architecture and base components"** beinhaltet die abstrakte funktionale Spezifikationen für das X3D Framework und Definitionen der standardisierten Komponenten (*components*) und Profile (*profiles*).

Zunächst werden Definitionen, Akronyme und Abkürzungen rund um X3D aufgeführt wie z.B. culling, field oder UTF-8. Danach wird auf die Konzepte von X3D eingegangen. Dies beinhaltet zum Beispiel Erläuterungen zum Erstellen und Abspielen von X3D-Szenen sowie der Modularisierung.
Begrifflichkeiten sind beispielsweise X3D Browser und Generatoren, der Szenengraph, die Laufzeitumgebung mit dem Objekt Modell (*Object model*), die DEF/USE Semantik (*DEF/USE semantics*), die internen sowie externen Prototyp-Semantik (*Prototype semantics*), die Import/Export Sematik, das Ereignis-Modell (*Event model*), sowie die Datenkodierung (Data encodings).

Nach dem Kapitel mit den Konzepten folgt die Feld-Typ-Referenz (*Field type reference*) und ein kurzes Kapitel über Anpassung (conformance) um Interoperabilität zu gewährleisten. Der erste Teil schließt mit ausführlichen Erklärungen zu den Komponenten *Core, Sound, Lighting, Texturing, Interpolation, Pointion device sensor, Key device, Environmental, Navigation, Environmental effects, Geospatial, Humanoid animation, NURBS, Distibuted interactive simulation, Scripting Event utilities* und dann im Anhang zu den vier Profilen *Core, Interchange, Interactive, MPEG-4, Immersive* und *Full*.

Der **zweite** Teil handelt vom **"Scene access interface (SAI)"**, welches dem Entwickler eine Vielzahl von Möglichkeiten gibt in die Szene einzugreifen. Das Interface kann dazu genutzt werden Knoten zu erstellen und zu löschen, *Events* zu den Knoten zu senden, Verbindungen (*routes*) zwischen den Knoten zu erstellen, Werte in den Knoten zu setzen oder zu lesen oder den Szenengraph zu traversieren.

Der Zugriff kann von innen (via *Script* Knoten) oder außen (z.B. via Web Browser).
Die Konzepte in diesem Teil beschreiben unter anderem Interface-Konstrukte (*Interface constructs*) wie *User Code, Session, Scene* und *Node*.

Ein weiteres Kapitel ist *Events.* Hier wird zwischen *Internal to browser* und *Browser to external application* unterschieden.

Das letzte Kapitel des zweiten Teiles ist umfangreicher:
Das Ausführungsmodell (*Execution model*), hier wird zunächst beschrieben wie man internen Interaktionscode evaluiert, danach werden verschiedene Punkte der internen Interaktion angesprochen. Angefangen von erlaubten Interaktionen über Browser-Interaktionen, Updating des Szenegraphs und dem "User Code lifecycle".

Dann wird zu den externen Interaktionen mit den gleichen Themenschwerpunkten übergegangen.
Zwei weitere wichtige Hauptkapitel sind die Datentyp Referenz (*Data type reference*) und die Dienste Referenz (*Services reference*). Bei der letzteren Referenz wird zwischen Diensten unterschieden die eine Verbindung aufbauen: Browser Dienste, Ausführungskontext-Dienste (*Execution context services*), Szenen-Dienste, Knoten-Dienste, Feld-Dienste, Route-Dienste, Prototyp-Dienste, Konfigurations-Dienste und Matrix Diensten unterschieden.

7.3 Neue Möglichkeiten mit X3D

Es ist sehr schwer den aktuellen Stand der neuen X3D Features (im Vergleich zu VRML 2.0) aufzudecken. Deshalb starteten wir eine Anfrage im öffentlichen Mail Verteiler x3d-public@web3d.org.

Alan Hudson teilte uns die Neuerungen mit [Neues1]:
"[...] X3D binary, h-anim, network sensor, DIS networking, event utilities, SAI, line properties, 4 component color, triangleset nodes, CAD profile, metadata nodes, keysensor, stringsensor, multitexture, TextureCoordinateGenerator.... lots of stuff that we haven't put simple examples up."

Bei der anschließenden Recherche konzentrierten wir uns auf folgende Punkte:
- Gab es das nicht auch schon bei VRML?
- Lässt es sich an einem einfachen Beispiel veranschaulichen?

Die Recherche ergab folgende Ergebnisse:

- *X3D binary* bedeutet, dass die Dateien auch binär und mit Kompression gespeichert werden können.
- *h-anim* steht für Humanoid Animation diese Arbeitsgruppe gab es auch schon bei VRML [Neues2], der Standard wird jedoch ständig weiterentwickelt.
- Die *network sensor* Knoten sollen Anpassungen an das Netzwerk wie zum Beispiel Bandbreitenoptimierung mit sich bringen. Es ist allerdings noch nicht in der Spezifikation aufzufinden, sondern nur ein Proposel [Neues3].
- *DIS networking*- DIS steht für *Distributed Interactive Simulation* [Neues4], Ziel der "DIS-Java-VRML Working Group" ist es mit Hilfe von Java und *DIS*, Zustände einer Szene (Welt) über Verteilte Systeme hinweg zu übermitteln. Das DIS Protokoll ist ein IEEE Standard. Das Projekt begann bereits 1997.
- *event utilities* bieten die Möglichkeit mit Hilfe von *ROUTEs* Szenarien ohne Scriptcode zu verwirklichen.
- *SAI* ist das Scene Access Interface, es ist dem von VRML sehr ähnlich [Neues5].
- *4 component color*- X3D besitzt die Felder *SFColorRGBA* und *MFColorRGBA* [Neues6], die zusätzlich zum Normalen RGB Farbmodell den Alpha-Kanal berücksichtigen können. Der Alphawert darf dabei zwischen 0.0 (vollständig transparent) und 1.0 (undurchsichtig) liegen.
- In X3D gibt es die neuen Knoten *IndexedTriangleFanSet, IndexedTriangleSet, IndexedTriangleStripSet, LineSet, TriangleFanSet, TriangleSet* und *TriangleStripSet* [Neues7]. Es scheint jedoch leider so als ob diese bei "einigen" Viewern noch nicht implementiert seien. Contact meldet z.B. "Unknown node class 'IndexedTriangleFanSet'" und auch der Xj3D Browser kann die Szene nicht darstellen. FreeWRL 1.13 soll "*triangleset nodes*" unterstützen, läuft aber leider nicht unter Windows [Neues8].
- Das *CADInterchange profile* beschreibt schlicht die Fähigkeiten von X3D im Hinblick auf Computer-Aided Design [Neues9]
- *keysensor, stringsensor* [neues10] generieren Events, falls bestimmte Tasten gedrückt werden, zu diesen Neuerungen folgen Beispiele.

Die Recherche war sehr zeitaufwendig. Wenn man Beispiele findet sind sie oft schön älter (2002/2003) und "abwärtskompatibel" zu VRML programmiert. Häufig bekommt man diese Beispiele nicht mehr auf aktuellen Viewern zum Laufen:

```
VRML syntax error: line   1 in
http://www.web3d.org/x3d/content/examples/X3dSpecification/AlarmClock.x3d
File does not have a valid header string
This can be caused by:
 - corrupt VRML/X3D file
 - invalid license
```

Außerdem ist der Code dadurch sehr unübersichtlich und eignet sich nicht zum Selbststudium. In der Spezifikation findet man öfters Bildbeispiele, allerdings keine Codebeispiele! Das macht den Einstieg natürlich sehr schwer. Viele Neuerungen sind schlicht nicht mit wenigen Codezeilen zu zeigen und eignen sich somit nicht hier dargestellt zu werden.

7.3.1 Metadata nodes

Metadaten Knoten sind ein neues Feature von X3D. Die Knoten werden nicht vom Viewer ausgelesen, sondern bieten einfach nur zusätzliche Informationen. Metadaten können einzelne Knoten beschreiben oder wie im unteren Beispiel die gesamte Welt.

```
<head>
    <meta name="filename" content="reichstag.x3d"/>
    <meta name="description" content="Der Berliner Reichstag"/>
    <meta name="author" content="Florian Moritz"/>
    <meta name="translator" content="Christoph Gerstle"/>
    <meta name="created" content="18 Oktober 2005*"/>
    <meta name="translated" content="19 Oktober 2005"/>
    <meta name="revised"    content="20 Oktober 2005"/>
    <meta name="version" content="1.0"/>
    <meta name="reference" content="http://www.flosweb.de/fh/cg"/>
    <meta name="reference" content="http://www.flosweb.de"/>
    <meta name="copyright" content="Copyright (c)Florian Moritz 2005"/>
    <meta name="drawing" content=""/>
    <meta name="image" content=""/>
    <meta name="movie" content=""/>
    <meta name="photo" content=""/>
    <meta name="keywords" content="VRML,Reichstag,X3D"/>
    <meta name="url" content="http://www.flosweb.de/fh/cg/reichstag.x3d"/>
    <meta name="generator" content="Vrml97ToX3dNist, http://ovrt.nist.gov/"/>
</head>
```

7.3.2 Keysensor

Referenz:
[Neues10]

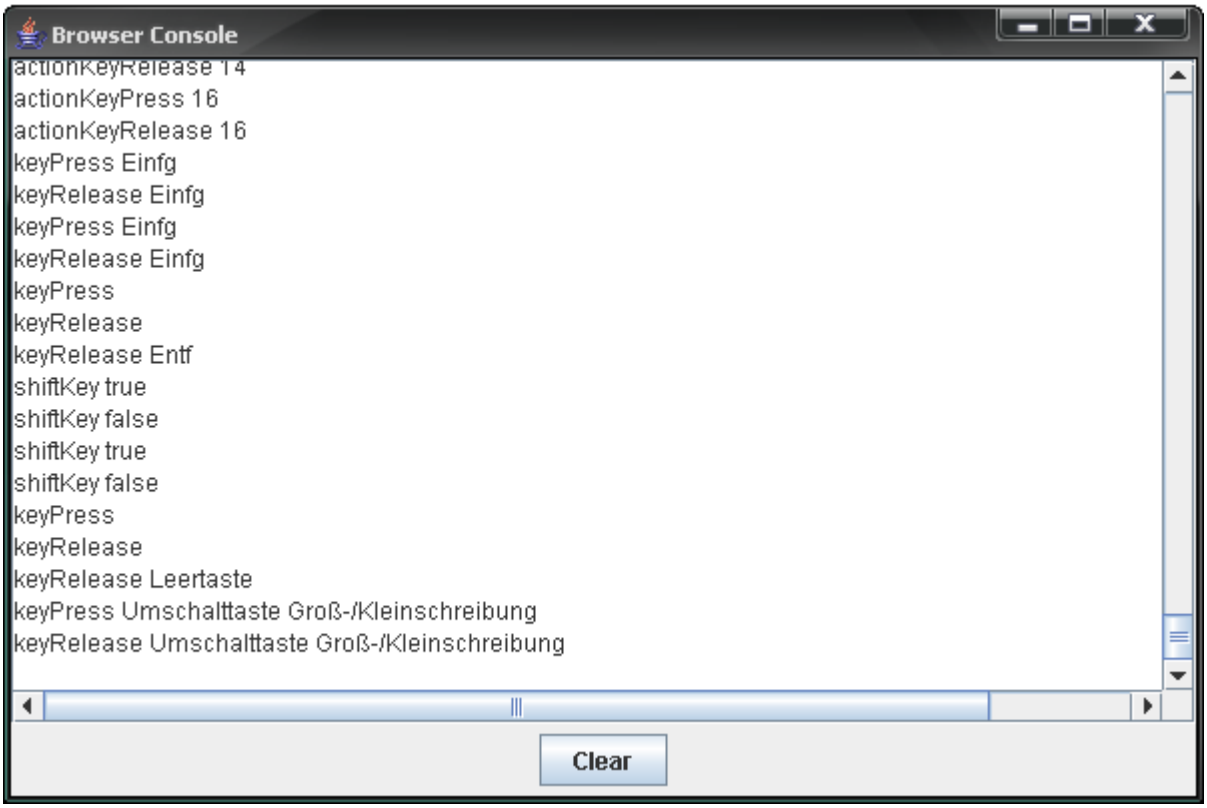
```
KeySensor : X3DKeyDeviceSensorNode {
  SFBool   [in,out] enabled          TRUE
  SFNode   [in,out] metadata         NULL [X3DMetadataObject]
  SFInt32  [out]    actionKeyPress
  SFInt32  [out]    actionKeyRelease
  SFBool   [out]    altKey
  SFBool   [out]    controlKey
  SFBool   [out]    isActive
  SFString [out]    keyPress
  SFString [out]    keyRelease
  SFBool   [out]    shiftKey
}
```

Hier ist nun ein einfaches Beispiel für die *KeyDeviceSensor* Kompomente. Der Code nutzt X3D, ist allerdings in klassischer VRML Syntax. Die Funktionalität ist die folgende:
Wenn einen Taste gedrückt wird, wird entsprechend ein String generiert und im Browserfenster ausgegeben.

```
#X3D V3.0 utf8

PROFILE Immersive
COMPONENT KeyDeviceSensor:1

NavigationInfo
{
   type ["NONE"]
}

DEF KEYS KeySensor
{
}
```

```
DEF SC Script
{
  inputOnly SFInt32 actionKeyPress
  inputOnly SFInt32 actionKeyRelease
  inputOnly SFBool  altKey
  inputOnly SFBool  controlKey
  inputOnly SFBool  isActive
  inputOnly SFString keyPress
  inputOnly SFString keyRelease
  inputOnly SFBool  shiftKey
  outputOnly SFBool bindVP1
  outputOnly SFBool bindVP2

#Script: Funktion je nach generiertem Event ausführen
  url ["ecmascript:
   function actionKeyPress(val)
   {
     Browser.println('actionKeyPress ' + val);
   }

   function actionKeyRelease(val)
   {
     Browser.println('actionKeyRelease ' + val);
   }

   function altKey(val)
   {
     Browser.println('altKey ' + val);
   }

   function controlKey(val)
   {
     Browser.println('controlKey ' + val);
   }

   function isActive(val)
   {
     Browser.println('isActive ' + val);
   }

   function keyPress(val)
   {
     Browser.println('keyPress ' + val);
   }

   function keyRelease(val)
   {
     Browser.println('keyRelease ' + val);
   }

   function shiftKey(val)
   {
     Browser.println('shiftKey ' + val);
    }
 "]
}

ROUTE KEYS.actionKeyPress TO SC.actionKeyPress
ROUTE KEYS.actionKeyRelease TO SC.actionKeyRelease
ROUTE KEYS.altKey TO SC.altKey
ROUTE KEYS.controlKey TO SC.controlKey
ROUTE KEYS.shiftKey TO SC.shiftKey
ROUTE KEYS.keyPress TO SC.keyPress
ROUTE KEYS.keyRelease TO SC.keyRelease
ROUTE KEYS.isActive TO SC.isActive
```

```
#einfache Box und 2 Viewpoints

Shape
{
    geometry Box {}
}

DEF VP1 Viewpoint {}
DEF VP2 Viewpoint
{
   position 1 1 10
}

ROUTE SC.bindVP1 TO VP1.set_bind
ROUTE SC.bindVP2 TO VP2.set_bind
```



7.3.3 StringSensor

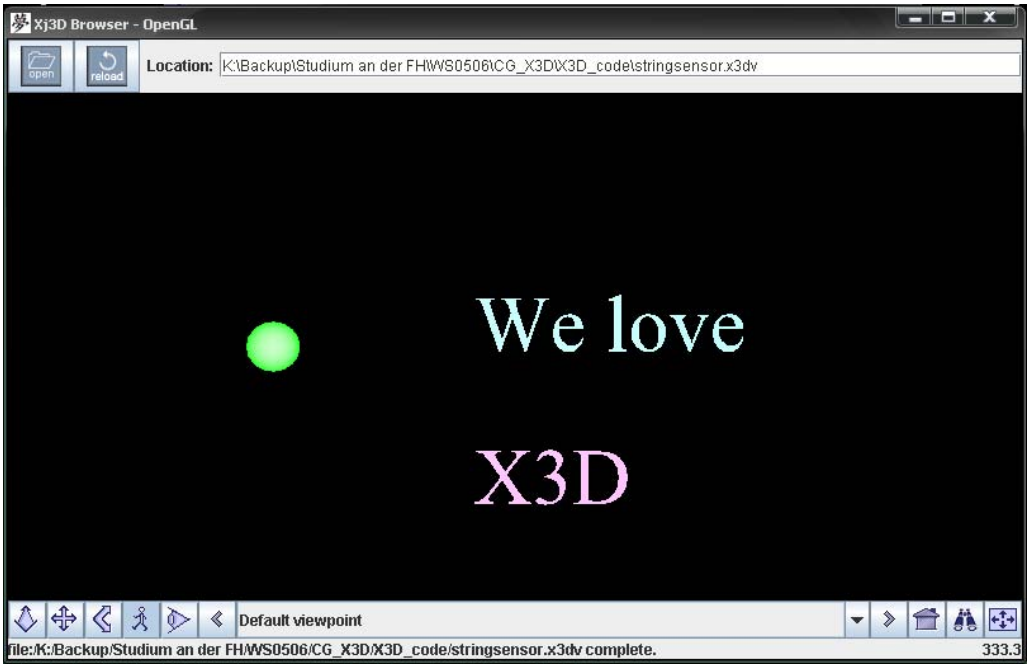
Referenz:
[Neues10]

```
StringSensor  : X3DKeyDeviceSensorNode
{
  SFBool   [in,out] deletionAllowed TRUE
  SFBool   [in,out] enabled         TRUE
  SFNode   [in,out] metadata        NULL [X3DMetadataObject]
  SFString [out]    enteredText
  SFString [out]    finalText
  SFBool   [out]    isActive
}
```

In dem Beispiel werden die aktuellen Eingaben im oberen Feld angezeigt. Die Eingabe wird durch die Return-Taste abgeschlossen. Der eingegebene String erscheint dann in der zweiten Zeile. In diesem Beispiel hier wurde zuerst "X3D" eingeben und anschließend "we love".

```
#X3D V3.0 utf8

PROFILE Immersive

DEF STRINGS StringSensor {}

# Shape signalisiert das Sensor aktiv ist
Transform
{
  translation -2 0 0
  children Shape
  {
    appearance Appearance
    {
      material DEF ACTIVE_MATERIAL Material
      {
        emissiveColor 0 0 1
      }
    }
    geometry Sphere { radius 0.25 }
  }
}

# Text Feld welches den gerade eingegebenen Text anzeigt
Transform
{
  translation 0 0.75 0
  children Shape
  {
    appearance Appearance
    {
      material Material
      {
        emissiveColor 0 1 1
      }
    }
    geometry DEF WORKING_TEXT Text
    {
      string "working text"
    }
  }
}

# Text Feld welches den letzten kompletten String anzeigt
Transform
{
  translation 0 -0.75 0
  children Shape
  {
    appearance Appearance
    {
      material Material
      {
        emissiveColor 1 0 1
      }
    }
    geometry DEF COMPLETE_TEXT Text
    {
      string "kompletter String "
    }
  }
}
```

```
DEF ACTIVE_SCRIPT Script
{
   inputOnly SFBool active
  outputOnly SFColor color
  url "ecmascript:
   function active(val)
   {
       if(val)
         color = new SFColor(0, 1, 0);
       else
         color = new SFColor(1, 0, 0);
   }
  "
}

ROUTE STRINGS.isActive TO ACTIVE_SCRIPT.active
ROUTE ACTIVE_SCRIPT.color TO ACTIVE_MATERIAL.emissiveColor

# Nutzung: Umwandeln des SFString des StringSensor Knoten in
# MFSting um es dem Sting Feld der Text Geometry anzuzeigen
DEF TEXT_SCRIPT Script
{
  inputOnly SFString enteredText
  inputOnly SFString finalText
  outputOnly MFString enteredStrings
  outputOnly MFString finalStrings
  url "ecmascript:
   function enteredText(val)
   {
     enteredStrings = new MFString(val);
   }
   function finalText(val)
   {
     finalStrings = new MFString(val);
   }
  "
}

ROUTE STRINGS.enteredText TO TEXT_SCRIPT.enteredText
ROUTE STRINGS.finalText TO TEXT_SCRIPT.finalText

ROUTE TEXT_SCRIPT.enteredStrings TO WORKING_TEXT.set_string
ROUTE TEXT_SCRIPT.finalStrings TO COMPLETE_TEXT.set_string
```



8 Beispielanwendung – Berlin-Szenerie

Als Beispielanwendung haben wir eine Szenerie aus Berlin gewählt. Es wurde das Brandenburger Tor, die Siegessäule, der Reichstag und die umgebene Landschaft mittels VRML modelliert. Außerdem wurden die drei Hauptobjekte maßstabsgetreu in der Peripherie platziert.

Wir möchten im Folgenden ein paar interessante Vorgehensweisen beim Modellieren der Objekte erläutern.

8.1 Siegessäule

Die Siegessäule ist insofern interessant, da es sich angeboten hatte diese fast ausschließlich aus primitiven Objekten zu modellieren, d.h. Quadern und Zylindern.

Es bot sich z.B. an das lokale Koordinatensystem der Siegessäule auf den Boden in Mitten der Säule zu legen. Die Primitive haben bekanntlich standardmäßig ihr lokales Koordinatensystem in ihrer Mitte im Innern, was die Translation für die meisten Elemente sehr einfach machte. Man musste lediglich die Höhen, also die Y-Koordinate verändern. Diesen Vorgehen bot sich z.B. für den Sockel, sowohl den runden, den eckigen und für den oberen schlanken Teil der Säule an. Der Säulengang mit den 16 kleinen Säulen in der Mitte hingegen war etwas komplexer.

Beim Modellieren in VRML bietet es sich immer an geeignete Prototypen zu definieren um möglichst viele Elemente wiederverwenden zu können. Dies soll an einem Beispiel verdeutlicht werden.

Hier wird ein Prototyp *Saeule* definiert, der z.B. für alle Elemente als Grundlage genommen werden kann, die eine zylindrische Form haben. Hierzu muss der Radius, die Höhe und die Farbe veränderbar sein. Verschieben sollte man den Zylinder natürlich auch können.

```
# Säulen Element
PROTO Saeule
[
  field SFFloat    radius 1
```

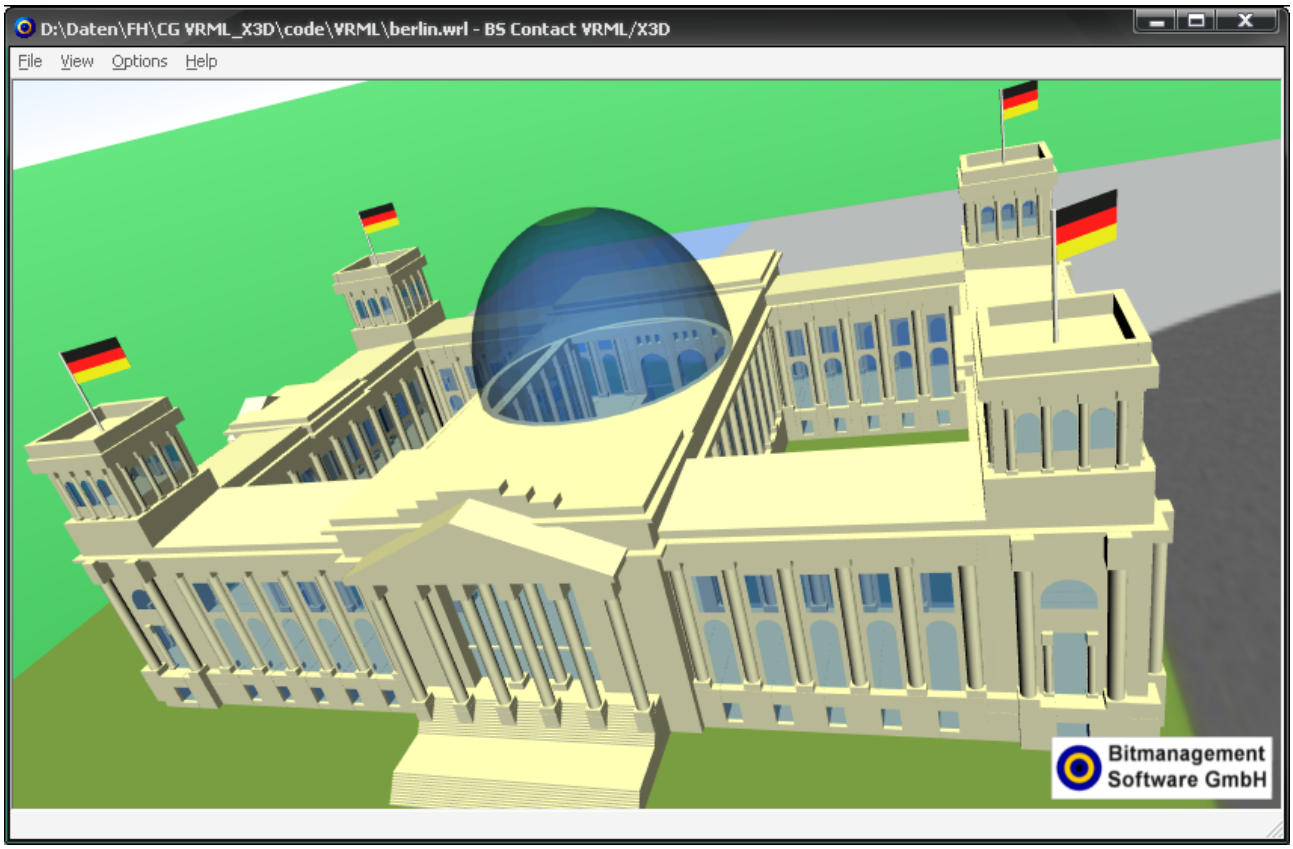
```
  field  SFFloat     hoehe  2
  field  SFVec3f     verschiebung 0 0 0
  exposedField SFColor    farbe .8 .8 .8
]
{
  Transform
  {
    children
    [
      Shape
      {
        appearance Appearance
        {
          material Material
          {
            diffuseColor IS farbe
          }

        }
        geometry Cylinder
        {
          radius IS radius
          height IS hoehe
        }
      }
    ]
    translation IS verschiebung
  }
}
```

Dieser Prototyp *Saeule* kann nun wiederverwendet werden.
Im nächsten Codeauszug wird dieser *Saeule*-Prototyp für den Teil über dem Säulenring verwendet, der aus 2 zylindrischen Prototypen besteht, also die beiden *Saeule* Knoten unterhalb des *children* Knoten. Zu dem wird aus gerade diesen beiden Säulen wieder ein neuer Prototyp *SockelScheibenOben* definiert, welchem wiederum bei einer späteren Verwendung eine neue Farbe zugewiesen werden kann bzw. welcher verschoben werden kann.

```
#Sockel über Säulenring
PROTO SockelScheibenOben
[
  field SFVec3f verschiebung 0 0 0
  exposedField SFColor farbe .6 .3 .2
]
{
  Transform
  {
    children
    [
      Saeule
      {
        verschiebung 0 1.25 0
        radius 6
        hoehe 0.5
        farbe IS farbe
      }
      Saeule
      {
        verschiebung 0 1 0
        radius 5.5
        hoehe 2
        farbe IS farbe
      }
    ]
    translation IS verschiebung
  }
}
```

8.2 Reichstag

Das Reichstagsgebäude stellt das größte und umfangreichste Objekt dar. Hier war es besonders wichtig geeignete Prototypen zu erstellen und die Wiederverwendung durchdacht zu nutzen.



Zudem war es häufig nötig auf andere Programmiersprachen zurückzugreifen, wie z.B. Java um beispielsweise Werte berechnen zu lassen, wie die Eckpunkte der Scheiben der Kuppel, die aus 364 Koordinaten besteht.
Ein etwas kleineres Beispiel war die Berechnung eines Fenster-Rundbogens.

Hierzu eine Java-Methode:
Bei dieser Methode *printCircle* gibt es als aufzurufende Parameter den Radius (radius), die Anzahl der Unterteilungen pro 360° (pieces) und die nicht genutzte Koordinatenachse (unusedDirection) X, Y oder Z.

```
/**
 * gibt die Eck-Werte eines Kreises aus
 * @param radius      Radius des Kreises
 * @param pieces      Anzahl der Unterteilungen
 * @param unusedDirection Nicht verwendete Koordinate x, y oder z
*/
public static void printCircle(double radius, int pieces, char unusedDirection)
{
  double interval = Math.PI*2/pieces;
  double currentAngle = 0.0;
  for(int i=0; i<pieces; i++)
  {
    //Berechnung
    double x2d = Math.cos(currentAngle) * radius;
    double y2d = Math.sin(currentAngle) * radius;

    //Nachkommastellen auf 3 begrenzen
    x2d = Math.rint(x2d*1000)/1000;
    y2d = Math.rint(y2d*1000)/1000;
```

```
    //Ausgabe
    if(unusedDirection == 'x') { System.out.print("0.0 "); }
    System.out.print(x2d + " ");
    if(unusedDirection == 'y') { System.out.print("0.0 "); }
    System.out.print(y2d + " ");
    if(unusedDirection == 'z') { System.out.print("0.0 "); }
    System.out.print(", #" + i + "\n");

    //Winkel erhöhen
    currentAngle += interval;
  }
}
```

Folgender Aufruf der Methode liefert die nachfolgenden Werte:

```
printCircle(1.0, 32, 'z');
```

```
1.0 0.0 0.0 , #0
0.981 0.195 0.0 , #1
0.924 0.383 0.0 , #2
0.831 0.556 0.0 , #3
0.707 0.707 0.0 , #4
0.556 0.831 0.0 , #5
0.383 0.924 0.0 , #6
0.195 0.981 0.0 , #7
0.0 1.0 0.0 , #8
... usw.
```

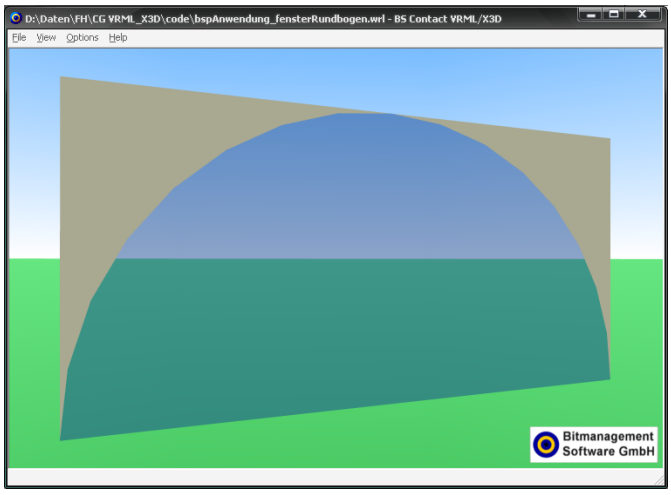
Aus dieser Konsolenausgabe des Java-Programms kann nur ein VRML-Prototyp *fensterRundbogen* konstruiert werden.

```
#Fenster mit Rundbogen und Wand
PROTO fensterRundbogen
[
  field   SFVec3f   verschiebung 0 0 0
  field   SFVec3f   skalierung 1 1 1
  field   SFColor   farbeWand .7 .7 .6
  field   SFColor   farbeFenster .1 .3 .6
  field   SFFloat   transparenz 0.5
]
{
  Transform
  {
    children
    [
      Transform
      {
        children
        [
          #Halbrundes Fenster
          Shape
          {
            appearance Appearance
            {
              material Material
              {
                diffuseColor IS farbeFenster
                transparency IS  transparenz
              }
            }
            geometry IndexedFaceSet
            {
              solid FALSE
              coord Coordinate
              {
                point
```

```
                [
                  #Java-Werte
                  #Halbkreis
                  1.0 0.0 0.0 , #0
                  0.981 0.195 0.0 , #1
                  0.924 0.383 0.0 , #2
                  0.831 0.556 0.0 , #3
                  0.707 0.707 0.0 , #4
                  0.556 0.831 0.0 , #5
                  0.383 0.924 0.0 , #6
                  0.195 0.981 0.0 , #7
                  0.0 1.0 0.0 , #8
                  -0.195 0.981 0.0 , #9
                  -0.383 0.924 0.0 , #10
                  -0.556 0.831 0.0 , #11
                  -0.707 0.707 0.0 , #12
                  -0.831 0.556 0.0 , #13
                  -0.924 0.383 0.0 , #14
                  -0.981 0.195 0.0 , #15
                  -1.0 -0.0 0.0 , #16
                  #unten links
                  -1.5 0.0 0.0 , #17
                  #unten rechts
                  1.5 0.0 0.0 #18
                ]
              }
              coordIndex
              [
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                10, 11, 12, 13, 14, 15, 16,
                17, 18, 0
              ]
            }
          }
          #Wand über Fenster
          Shape
          {
            appearance Appearance
            {
              material Material
              {
                diffuseColor IS farbeWand
              }
            }
            geometry IndexedFaceSet
            {
              solid FALSE
              convex FALSE
              coord Coordinate
              {
                point
                [
                  #Java-Werte
                  #Halbkreis
                  1.0 0.0 0.0 , #0
                  0.981 0.195 0.0 , #1
                  0.924 0.383 0.0 , #2
                  0.831 0.556 0.0 , #3
                  0.707 0.707 0.0 , #4
                  0.556 0.831 0.0 , #5
                  0.383 0.924 0.0 , #6
                  0.195 0.981 0.0 , #7
                  0.0 1.0 0.0 , #8
                  -0.195 0.981 0.0 , #9
                  -0.383 0.924 0.0 , #10
                  -0.556 0.831 0.0 , #11
                  -0.707 0.707 0.0 , #12
```

```
                    -0.831 0.556 0.0 , #13
                    -0.924 0.383 0.0 , #14
                    -0.981 0.195 0.0 , #15
                    -1.0 -0.0 0.0 , #16
                    #oben links
                    -1.0 1 0.0 , #17
                    #oben rechts
                    1.0 1 0.0, #18
                  ]
                }
                coordIndex
                [
                  0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                  10, 11, 12, 13, 14, 15, 16,
                  17, 18, 0
                ]
              }
            }
          ]
          translation 1 0 0
        }
      ]
      scale IS skalierung
      translation IS verschiebung
    }
}
```

Dieser Prototyp sieht, wenn man ihn als alleinstehendes Objekt mit *fensterRundbogen { }* aufruft, folgendermaßen aus:



Eine weiter Besonderheit beim Reichstag ist die Animation der Türen. Diese befindet sich an allen vier Eingängen. Durch Gedrückthalten der Maustaste werden die Türen geöffnet bzw. Wieder geschlossen.

Das lokale Koordinatensystem der Türen liegt in der linken unteren Ecke.
Der Quelltext der animierten Flügeltüren:

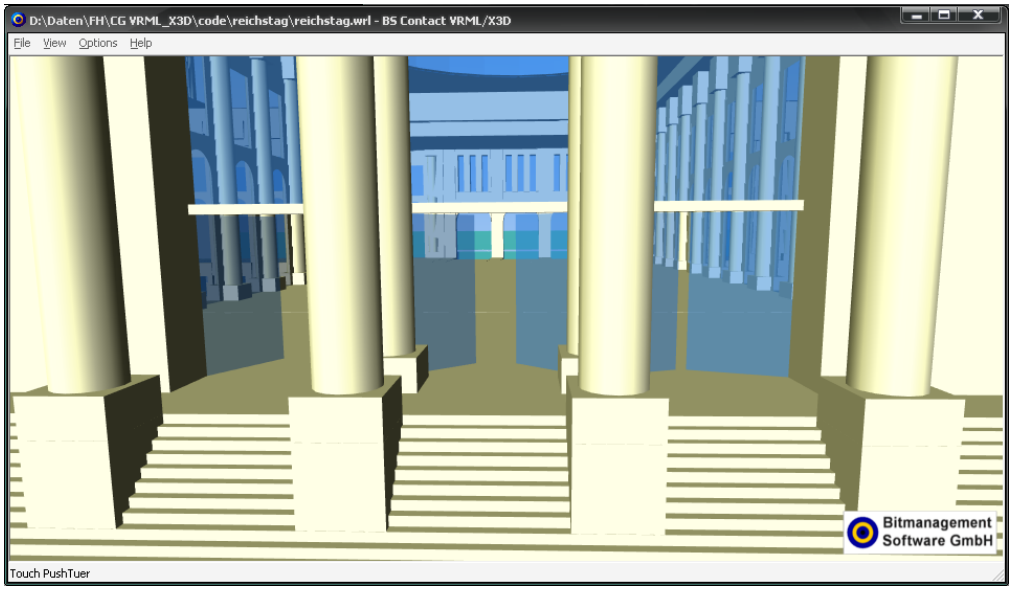
```
#Linke Tür
#lokales Koordinatensystem: unten links
DEF  FenterTuerLinks Transform
{
  children
  [
    Transform
    {
      children
      [
        Shape
        {
          appearance Appearance
```

```
            {
                material Material
              {
                diffuseColor .1 .3 .6
                transparency .5
              }
            }
            geometry Box
            {
              size 3 5 0
            }
          }
        ]
        translation 1.5 2.5 0
      }
    ]
}

#Rechte Tür
#lokales Koordinatensystem: unten rechts
DEF  FenterTuerRechts Transform
{
  children
  [
    Transform
    {
      children
      [
        Shape
        {
          appearance Appearance
          {
                material Material
            {
              diffuseColor .1 .3 .6
              transparency .5
            }
          }
          geometry Box
          {
            size 3 5 0
          }
        }
      ]
      translation -1.5 2.5 0
    }
  ]
  translation 6 0 0
}

#Timer zur Zeit der Öffnens und Schließens
DEF UhrTuer TimeSensor
{
  enabled FALSE
  cycleInterval 4.0
  loop TRUE
}

#initialisieren eines TouchSensors zum Öffnen und Schließen
DEF  PushTuer TouchSensor { }

#Definiert die Zeitintervalle und die Öffnungswinkel
#Tür öffnet sich schnell zuletzt langsam
#Tür schließt sich schnell und bleibt kurze Zeitspanne geschlossen
#links
DEF InterpolatorTuerOeffenLinks OrientationInterpolator
{
```

```
  key [ 0.0 0.3 0.4 0.5 0.8 1.0 ]
  keyValue
  [
  0 1 0 0, # 0° Grundposition = geschlossen
  0 1 0 1.4, # 80°
  0 1 0 1.55, # 88°
  0 1 0 1.4, # 80°
  0 1 0 0, # 0° Grundposition
  0 1 0 0 # 0° Grundposition
  ]
}

#rechts
DEF InterpolatorTuerOeffenRechts OrientationInterpolator
{
  key [ 0.0 0.3 0.4 0.5 0.8 1.0 ]
  keyValue
  [
  0 1 0 0, # 0° Grundposition
  0 1 0 -1.4, # -80°
  0 1 0 -1.55, # -88°
  0 1 0 -1.4, # -80°
  0 1 0 0, # 0° Grundposition
  0 1 0 0 # 0° Grundposition
  ]
}

#beim Klicken auf die Tür, wird der Timer gestartet
ROUTE PushTuer.isActive TO  UhrTuer.set_enabled

#ein ticken der Uhr, lässt die Tür rotieren - links
ROUTE UhrTuer.fraction_changed TO InterpolatorTuerOeffenLinks.set_fraction
ROUTE InterpolatorTuerOeffenLinks.value_changed TO FenterTuerLinks.set_rotation

#ein ticken der Uhr, lässt die Tür rotieren - rechts
ROUTE UhrTuer.fraction_changed TO InterpolatorTuerOeffenRechts.set_fraction
ROUTE InterpolatorTuerOeffenRechts.value_changed TO
FenterTuerRechts.set_rotation
```

Die Flügeltüren werden mit Hilfe eines *Inline*-Knotens in den Szenengraphen eingebunden und absolut im lokalen Koordinatensystem des Reichstags positioniert. Würde man die Flügeltüren im Seitenportal des Reichstags positionieren, hätte man das Problem, dass dieses Seitenportal im Norden und im Süden steht, also wiederverwendet wird. Ein Öffnen der Türen würde also beide Türen im Norden und im Süden öffnen. Deswegen müssen beide Türen unabhängig von den Seitenportalen absolut positioniert werden.

```
#Flügeltürenpaar Nord
Transform
{
  children
  [
    Inline
    {
      url  "reichstagFluegeltueren.wrl"
    }
  ]
  translation 48 2 2
  rotation 0 1 0 3.142
}
```



8.3 Brandenburger Tor

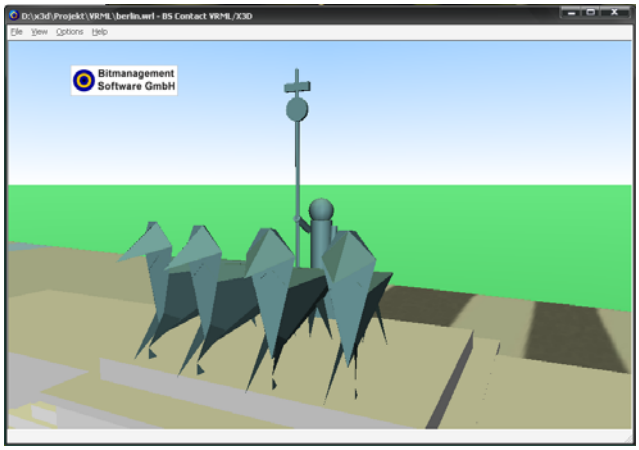
Das Brandenburger Tor besteht in seinen Grundzügen auch aus Primitiven. Erwähnenswert hierbei sind die Figuren auf dem Dach, die sogenannte Quadriga. Die Pferde und die Person sind zwar stark abstrahiert, jedoch interessant, da sie mit Hilfe von *IndexedFaceSet* modelliert wurden.

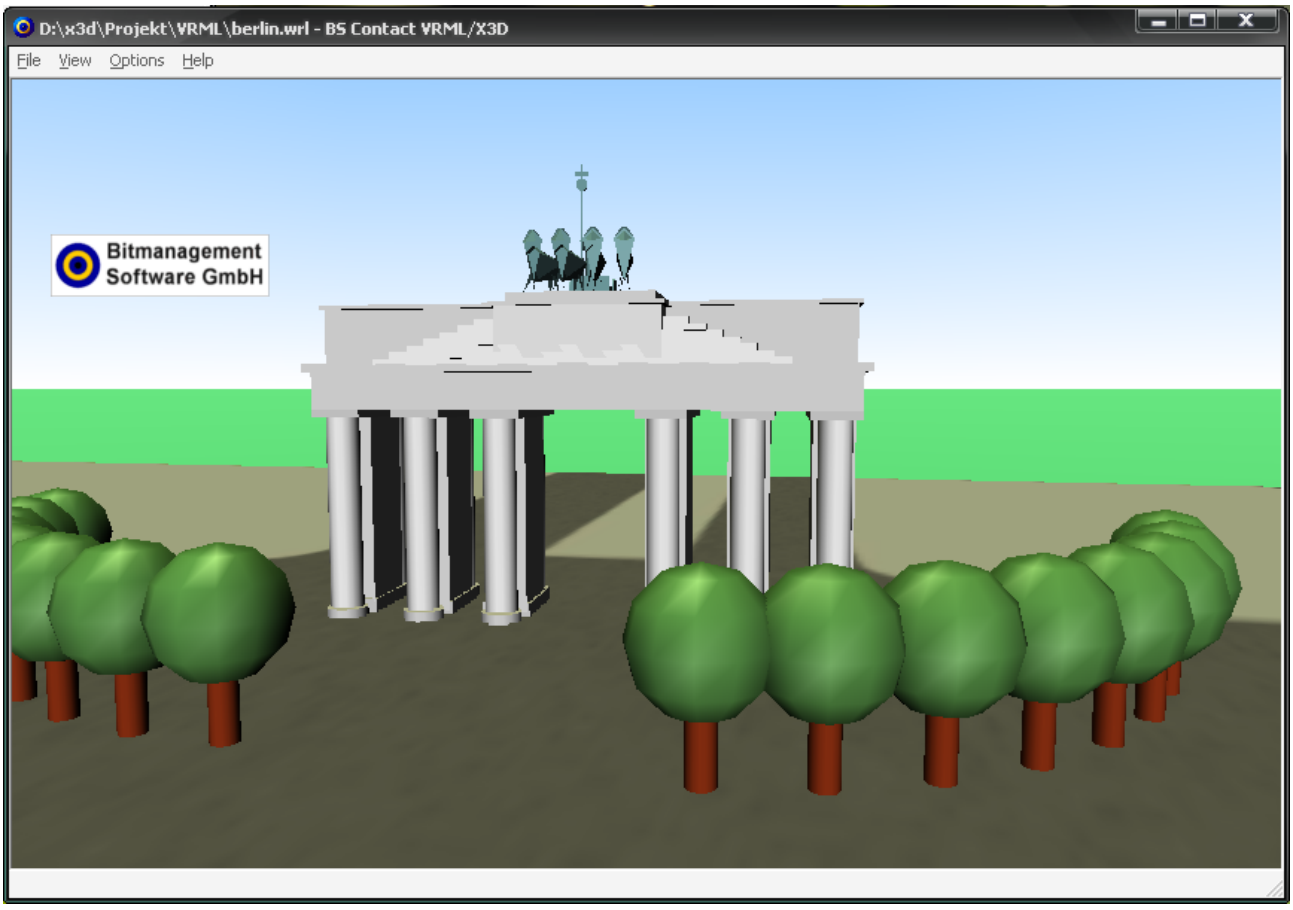
Als ausführliches Beispiel wird die Modellierung eines einzelnen Pferdes aufgezigt. Als Flächen dienen ausschließlich Dreiecke.

```
Shape
{
  appearance Appearance
  {
    material Material
    {
      ambientIntensity .137
      diffuseColor 1 .988 .953
      shininess 0
    }
  }

  geometry IndexedFaceSet
  {
    solid FALSE
    coord Coordinate
    {
      point
      [
        #Schnauze
         0 10.5 0 , #0 Schnauzenspitze
        -1 11  -2 , #1 links hinten
         1 11  -2 , #2 rechts hinten
         0 12  -2 , #3 mitte hinten

        #Ohren
        -0.5 12.5 -2 #4 Ohr links
         0.5 12.5 -2 #5 Ohr rechts

        #Hals
        0 7 -2, #6 unten

        #Hinterteil
         0   8   -8   #7 unten
```

```
        -0.8 9.5 -7.8 #8 links
         0.8 9.5 -7.8 #9 rechts

        #Koerper am Hals
        -0.7 9.8 -2 #10 Ansatz links
         0.7 9.8 -2 #11 Ansatz rechts

        #Ruecken
         0 10.5 -2 #12 Rueckenmitte

        #Schweif
         0 9.5 -7.8 #13 Mitte oben
         0 8.3 -9   #14 Schwanz hinten

        #Beine
        #hinten links
         -0.1 8.2 -7.6 #15 hinten oben
         -0.2 8.2 -6.2 #16 vorne oben
         -0.6 6.0 -7.6 #17 unten

        #hinten rechts
         0.1 8.2 -7.6 #18 hinten oben
         0.2 8.2 -6.2 #19 vorne oben
         0.6 6.0 -7.6 #20 unten

        #vorne rechts Oberschenkel
         0.15 7.5 -3.0 #21 hinten oben
         0.25 8.0 -2.0 #22 vorne oben
         0.5  6.8 -1.0  #23 unten

        #vorne rechts Unterschenkel
         0.5 6.3 -1.4 #24 unten hinten
         0.5 6.3 -1.0 #25 unten vorne

        #vorne links
         -0.15 8.0 -3.0 #26 oben hinten
         -0.25 8.0 -2.0 #27 oben vorne
         -0.60 6.0 -1.0 #29 unten
      ]
    }

    coordIndex
    [
      0, 3, 1, -1, # Schnauze links
      0, 2, 3, -1, # Schnauze rechts
      0, 2, 1, -1  # Schnauze unten

      1, 3, 4, -1  # Ohr links
      2, 5, 3, -1  # Ohr rechts

      6, 3, 1, -1  # Hals links
      6, 2, 3, -1  # Hals rechts

      7, 9, 8, -1  # Hinterteil

      6, 7, 8, -1  # linke Seite unten
      6, 9, 7, -1  # rechte Seite unten

      6, 10, 8, -1 # Halsansatz links
      6, 11, 9, -1 # Halsansatz rechts

      10, 12, 8, -1 # Ruecken links
      11, 9, 12, -1 # Ruecken rechts
      12, 9,  8, -1  # Ruecken mitte

      13, 14, 7, -1 # Schwanz
```

```
        15, 16, 17, -1 # Bein hinten links
        19, 18, 20, -1 # Bein hinten rechts

        22, 21, 23, -1 # Bein - Oberschenkel rechts

        23, 24, 25, -1 # Bein - Unterschenkel rechts

        26, 27, 28, -1 # Bein vorne links
      ]
    }
}
```





8.4 Peripherie

Die Gebäude Brandenburger Tor, Reichstag und die Siegessäule sind auf einem Grundkörper, einer Box, positioniert. Diese Box ist mit einer Textur versehen, die die markantesten Eigenschaften des Stadtteils aufweist, wie beispielsweise die Straße des 17. Junis, dem Tierpark und die Spree. Auf diese wird mittels Schriftzügen hingewiesen.
Am auffälligsten sind die Bäume entlang der Straße zwischen dem Brandenburger Tor und der Siegessäule. Dabei handelt es sich um 6 Baumreihen, jeweils 3 auf jeder Straßenseite, wobei die Reihen direkt an der Straße in regelmäßigen Abständen Straßenlaternen enthalten.
Da es sich um eine große Anzahl von Bäumen handelt, wurde der VRML-Code mit einem Javaprogramm erstellt:

```
public class Baumreihe
{
    public static void main(String[] args)
  {
    int grenze = 1290;
    int baumAbstand_ver = 10;
    int baumAbstand_hor = 10;
    int zentrumAbstand = 30;
    boolean laterne_flag = true;
    int laterne = 10;
    int start = 0;
    int countReihe = 3;


    for (int i=1; i<=countReihe; i++)
    {
      for (int j=start; j<grenze; j+=baumAbstand_ver)
      {
        if (j % 60 == 0 && laterne_flag == true && j>start)
        {
          int x = i*baumAbstand_ver + zentrumAbstand;
          System.out.println ("Laterne {verschiebung " + x + " 0 " + j + "}");
          System.out.println ("Laterne {verschiebung " + -1*x + " 0 " + j +
            "}");
        }
        else
        {
          int x = i*baumAbstand_ver +zentrumAbstand;
          System.out.println ("Baum {verschiebung " +
            x + " 0 " +j+ "}");
          System.out.println ("Baum {verschiebung " +
            -1*x + " 0 " +j+ "}");
        }
      }
      if (i==1) laterne_flag = false;
    }
  }
}
```

Die Variablen ermöglichen dabei eine sehr spezifische Angabe des auszugebenden Codes. So kann die Länge der Baumreihen, der Abstand der Bäume in vertikaler und horizontaler Richtung, den Abstand zum Szenenmittelpunkt, die Anzahl der Reihen, sowie angegeben werden, ob die Reihe Laternen enthalten soll oder nicht.

Hier ist ein Auszug aus der Ausgabe:

```
Baum {verschiebung 40 0 0}
Baum {verschiebung -40 0 0}
Baum {verschiebung 40 0 10}
Baum {verschiebung -40 0 10}
Baum {verschiebung 40 0 20}
Baum {verschiebung -40 0 20}
Baum {verschiebung 40 0 30}
```

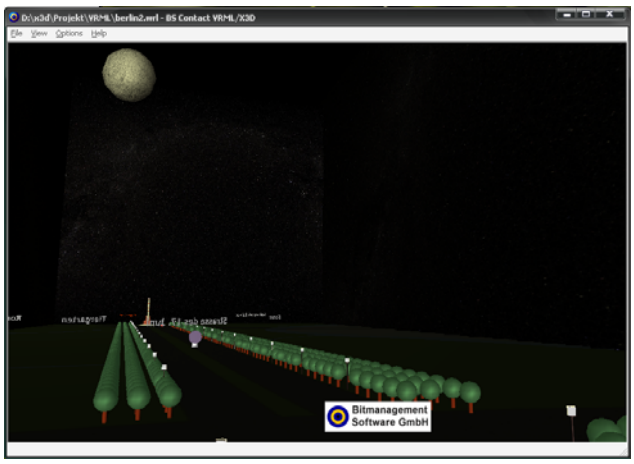
```
Baum {verschiebung -40 0 30}
Baum {verschiebung 40 0 40}
Baum {verschiebung -40 0 40}
Baum {verschiebung 40 0 50}
Baum {verschiebung -40 0 50}
Laterne {verschiebung 40 0 60}
Laterne {verschiebung -40 0 60}
Baum {verschiebung 40 0 70}
Baum {verschiebung -40 0 70}
...
Baum {verschiebung 60 0 1280}
Baum {verschiebung -60 0 1280}
```

Begrenzt wird der Boden durch 4 weitere Boxen, die Texturen aufweisen.

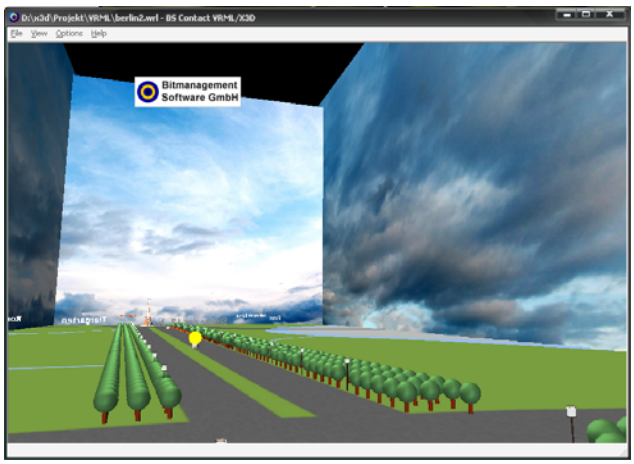
Anmerkungen:

- Eine Begrenzung durch eine *Sphere* war in diesem Fall nicht möglich, da in VRML ein Material, bzw. eine Textur nur sichtbar ist, wenn sich der Viewport außerhalb des Objektes befindet. Ist man allerdings innerhalb eines Objektes werden die Werte des *Appearance*-Knoten von VRML ignoriert.
- Eine weitere Möglichkeit stellt der Backgroundknoten dar. Dieser ermöglicht es für alle Seiten eine Grafik anzugeben (Seite 16).
  Platziert man diese nun innerhalb eines Switchknotens, werden sie von Contact nicht angezeigt. Bei anderen Playern, wie z.B. Cortona funktioniert dies ohne weiteres.
  Da aber der Player von Bitmanagement am fortschrittlichsten ist und der einzigste der unsere Szene flüssig darstellt, haben wir uns dafür entschieden die Texturen auf Boxen zu platzieren, da Primitive innerhalb des Switchknotens bei dem Player keine Probleme machten.

Durch eine Schaltfläche, die als Werbetafel modelliert wurde, kann der Benutzer zwischen Tag und Nacht in der Szene wechseln. Hierdurch ändert sich die Hauptbeleuchtung, sowie die Texturen der begrenzenden Boxen. Außerdem wird im Nachtzustand der Mond als *Spehre* dargestellt. Ein *PointLight*, dass die gleiche Position wie der Mond beleuchtet zudem die Nachtszene leicht.



Nachtzustand



Tagzustand

Die Tag/Nacht – Steuerung ist u.a. mittels Javascript und dem *SwitchNode* in VRML umgesetzt:

```
# touchsensors der zwischen Tag und Nacht wechselt
DEF LightSwitch Transform
{
   translation 58 0.5 4

   children
   [
     DEF Switcher Switch
     {
        whichChoice 0
        choice
        [
           # ES IST TAG
           Transform
           {
              children
              [
                 DEF LightSwitchONSensor TouchSensor
                 {
                 }
                 Inline { url ["switchDay.wrl"] }
              ]
              rotation 0 1 0 1.65
           }

           # ES IST NACHT
           Transform
           {
              children
              [
                 DEF LightSwitchOFFSensor TouchSensor
                 {
                    enabled FALSE
                 }
                 Inline { url ["switchNight.wrl"] }
              ]
              rotation 0 1 0 1.65
           }
        ]
     }
  ]
}

# JavaScript
# Licht AN
DEF lightONScript Script
{
```

```
   eventIn SFBool lightONIsActive
   field SFNode light USE LIGHT
   field SFNode lightswitch USE Switcher
   field SFNode dayswitch USE daySwitcher
   field SFNode offsensor USE LightSwitchOFFSensor
   directOutput TRUE

   url
   [
     "javascript:
     function lightONIsActive(active)
     {
        light.set_on = FALSE;
        lightswitch.set_whichChoice = 1;
        dayswitch.set_whichChoice = 1;
        offsensor.set_enabled = TRUE;
     }"
   ]
}
# Link activiation für lightON bei eventIn von lightOnScript
ROUTE LightSwitchONSensor.isActive TO lightONScript.lightONIsActive

# Licht AUS
DEF lightOFFScript Script
{
   eventIn SFBool lightOFFIsActive
   field SFNode light USE LIGHT
   field SFNode dayswitch USE daySwitcher
   field SFNode lightswitch USE Switcher
   directOutput TRUE

   url
   [
     "javascript:
     function lightOFFIsActive(active)
     {
        light.set_on = TRUE;
        lightswitch.set_whichChoice = 0;
        dayswitch.set_whichChoice = 0;
     }"
   ]
}

# Link activiation für lightOFF bei eventIn von lightOnScript
ROUTE LightSwitchOFFSensor.isActive TO lightOFFScript.lightOFFIsActive
```

Der Wechsel der Texturen, sowie die Einblendung des Mondes ist analog in einem weiteren SwitchNode realisiert.

Zur Umschaltung zwischen Tag und Nacht dient eine Werbetafel auf der Straße des 17. Junis. Diese ist komplett anklickbar. Auf der Werbetafel ist jeweils der nächste Zustand abgebildet.

Auf der „Straße des 17. Junis" fahren außerdem 2 Autos hin und her. Diese Animation verwendet den *PositionInterpolator* und den *TimeSensor*. Die Strecke der Autos führt vom Brandenburger Tor bis zur Siegessäule. Die Animation wiederholt sich. Ein Durchgang dauert 30 Sekunden.

```
DEF  Autofahrt1 Transform
{
   children
   [
      Auto
      {
         verschiebung 0 0 0
         drehung 0 1 0 1.571
         skalierung 2.5 2.5 2.5
      }
   ]
}
DEF ZeitAuto1 TimeSensor
{
  cycleInterval 30.0
  loop TRUE
}

DEF InterpolatorAuto1 PositionInterpolator
{
  key [ 0.0 0.5 1.0 ]
  keyValue
  [
    -30 1.4 -680,
     30 1.4    0,
     90 1.4  680
  ]
}

ROUTE ZeitAuto1.fraction_changed TO InterpolatorAuto1.set_fraction
ROUTE InterpolatorAuto1.value_changed TO Autofahrt1.set_translation
```

9 Fazit

Während der Entwicklung unseres Projektes wurde uns klar, dass VRML eher der Vergangenheit angehört. Es hat nur eine sehr geringe Verbreitung als Informationsdarstellung im Netz. Internetseiten die diese Technologie nutzen sind schon eher als Exoten anzusehen. Heutzutage wird VRML höchstens noch als Datenaustauschformat zwischen verschiedenen 3D Programmen genutzt, wobei es selbst dabei nur bedingt einsetzbar ist, da viele Features moderner 3D Programme nicht in VRML umgesetzt werden können.
Durch die immer weiter fortschreitende Entwicklung der Hardware, inbesondere von Grafikkarten, ist es heutzutage jedem möglich anspruchsvolle 3D Programme auf dem Heim-PC zu nutzen, die viel mehr Möglichkeiten bieten als VRML. Dabei kann das Argument des Geldes für die Software schon fast vernachlässigt werden, da sich mit Freeware-Programmen, wie beispielsweise Blender, sehr gute Ergebnisse erzielen lassen. Des Weitern haben Webtechnologien, wie beispielsweise Macromedia Flash zur Verdrängung von VRML beigetragen.
Der Nachfolger X3D, den es jetzt schon seit mehreren Jahren gibt, steckt immer noch in den Kinderschuhen. Viele Features wurden bis heute nicht umgesetzt und es kann auch nicht gesagt werden, ob dies überhaupt noch je geschehen wird. X3D ist kaum verbreitet, bzw. im breiten Umfeld bekannt. So ist eine kommerzielle Nutzung kaum denkbar. Das Potential ist zwar vorhanden, aber es fehlt an Weiterentwicklungen und Support für Einsteiger seitens der Entwickler.

10 Quellennachweise

[Geschichte1] http://www.playfulworld.com/, Zugriff: 10.10.2005

[Geschichte2] http://flux.typepad.com/the_flux_papers/, Zugriff: 10.10.2005

[Geschichte3] http://www.web3d.org/x3d/specifications/vrml/VRML1.0/index.html5, Zugriff: 10.10.2005

[Geschichte4] http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-IS-VRML97WithAmendment1, Zugriff: 10.10.2005

[StandDerDinge1] http://www.web3d.org/, Zugriff: 12.10.2005

[StandDerDinge2] http://www.sazkaarena.com/pg.php?id=M05S01A00&lang=2, Zugriff: 12.10.2005

[StandDerDinge3] http://www.spacetime3d.com/X3DiamondAwards/, Zugriff: 12.10.2005

[StandDerDinge4] http://www.karmanaut.com/virtuality/zeitgeist/, Zugriff: 12.10.2005

[Viewer1] http://www.bitmanagement.de/management/management.de.html, Zugriff: 07.10.2005

[Viewer2] http://www.neeneenee.de/vrml/forum/php/viewtopic.php?id=6&t_id=118l, Zugriff: 07.10.2005

[Autorenwerkzeuge1] http://rawkee.sourceforge.net/

[Autorenwerkzeuge2] http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro, Zugriff 09.10.2005

[Autorenwerkzeuge3] http://de.wikipedia.org/wiki/Blender_(Software), Zugriff 09.10.2005

[Autorenwerkzeuge4] http://ovrt.nist.gov/v2_x3d.html, Zugriff: 14.10.2005

[Spezifikation1] http://www.debacher.de/vrml/vrml.htm, Zugriff: 05.10.2005

[Spezifikation2] http://194.94.127.15/lehre/naturwissenschaft/infophysik/IP-WBT-Demo/content/01einfuehrung/0130vrml/013025koordinaten.html, Zugriff: 05.10.2005

[Spezifikation3] http://web3d.vapourtech.com/tutorials/vrml97/index.html, Zugriff: 05.10.2005

[Spezifikation4] http://cs.uni-muenster.de/u/lammers/EDU/ws03/Landminen/Abgaben/Gruppe7a/Thema07a-3DDatenformate-KoljaThierfelder.pdf, Zugriff: 12.10.2005

[Spezifikation5] http://www.web3d.org/x3d/specifications/

[Spezifikation6] http://tecfa.unige.ch/guides/vrml/vrmlman/vrmlman.html, Zugriff: 20.10.2005

[Spezifikation7] http://www.fh-wedel.de/~si/seminare/ws98/Ausarbeitung/5.Zint/java.htm, Zugriff: 20.10.2005

[Neues1] http://www.web3d.org/x3d/publiclists/x3dpublic_list_archives/0510/msg00045.html, Zugriff: 18.10.2005

[Neues2] http://www.ballreich.net/vrml/h-anim/h-anim-examples.html, Zugriff: 18.10.2005
http://www.web3d.org/x3d/specifications/ISO-IEC-19774-FCD-HumanoidAnimation/

[Neues3] http://planet-earth.org/x3d/networkSensor.html, Zugriff: 18.10.2005

[Neues4] http://web.nps.navy.mil/~brutzman/vrtp/dis-java-vrml/AnnotatedReferences.html, Zugriff: 18.10.2005

[Neues5] http://www.xj3d.org/tutorials/general_sai.html, Zugriff: 18.10.2005

[Neues6] http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/Part01/fieldsDef.html#SFColorRGBA, Zugriff: 18.10.2005

[Neues7] http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/Part01/components/rendering.html, Zugriff: 18.10.2005

[Neues8] http://www.web3d.org/x3d/publiclists/vrml_list_archives/0508/msg00000.html, Zugriff: 18.10.2005

[Neues9] http://www.web3d.org/x3d/specifications/ISO-IEC-19775-Amendment1-FPDAM-X3DAbstractSpecification/Part01/CADInterchange.html, Zugriff: 18.10.2005

[Neues10] http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/Part01/components/keyboard.html, Zugriff: 18.10.2005