Kapitel 20

VRML

VRML, sprich Wörmel, ist eine für das WWW entworfene Virtual Reality Modelling Language zur Beschreibung von 3-dimensionalen Szenen mit multimedialen Komponenten und Animation. Die gerenderte Projektion der Szene kann von jedem Web-Browser betrachtet werden, der über ein passendes Plugin verfügt.

20.1 Geschichte

Bei der Nutzung von Computer-Resourcen läßt sich ein weiterer Paradigmenwechsel beobachten: Während in der EDV-Gründerzeit speziell ausgebildete Techniker am Mainframe-Computer Befehle eintippten, danach einige Jahrzehnte später Kopfarbeiter per Drag & Drop Fensterelemente manipulierten, surft inzwischen jedermann und -frau in einer weltweit vernetzten multimedialen Landschaft. Der Kontext hat sich also von institutionell über persönlich zu sozial gewandelt.

Aus diesem Zeitgeist heraus diskutierten im April 1994 auf der 1st International WWW Conference in Genf Tim Berners-Lee, einer der Väter des WWW, mit den beiden Autoren des Systems Labyrinth, Mark Pesce und Tony Parisi. Es wurde eine Mailing List aufgesetzt, die schon nach wenigen Wochen mit mehr als 1000 Teilnehmern über Syntax für Strukturen, Verhalten und Kommunikation debattierte. Bereits im Oktober 1994 wurde auf der 2nd International WWW Conference in Chicago VRML 1.0 vorgestellt, ein Entwurf, der wesentlich vom Silicon Graphics System Open Inventor inspiriert war. VRML 1.0 konnte bereits geometrische Grundkörper und Polygone in einem Koordiantensystem plazieren und ihre Farbe und Materialeigenschaften spezifizieren. Auch ließen sich durch Hyperlinks Objekte beliebig im Web referieren. Abgesehen von dieser Möglichkeit der Interaktion handelte es sich allerdings um rein statische Szenen.

Diesem Manko sollte eine schnellstens eingerichtete VAG (VRML Architecture Group) abhelfen, welche Überlegungen zur Animation und zur Integration multimedialer Komponenten wie Sound und Video koordinierte. Zur 1st International VRML Conference im Dez. 1995 war es dann soweit: Als Sieger einer Ausschreibung für VRML 97 ging Moving Worlds von Silicon Graphics nach einer On-Line-Wahl. Überarbeitete Syntax sorgte für die Beschreibung statischer Szenen mit multimedialen Bestandteilen und ein neues Event-Handling-Konzept erlaubte Animation dieser Szenen sowie Interaktion mit dem Benutzer.

20.2 Einbettung

VRML-Szenen werden beschrieben in ASCII-Dateien mit der Endung *.wrl, welche innerhalb einer HTML-Seite mit dem EMBED-Kommando referiert werden, z.B.

```
<EMBED SRC=zimmer.wrl  WIDTH=600 HEIGHT=400>
```

Ein entsprechend konfigurierter Web-Server schickt dem anfordernden Clienten als Vorspann dieser Daten den Mime-Typ VRML, worauf das zur Betrachtung installierte Plugin, z.B. Cosmo Player 2.0 von Silicon Graphics, die eingehenden Daten in eine interne Datenstruktur einliest, von wo sie zur Berechnung einer fotorealistischen Projektion verwendet werden. In welcher Weise Blickwinkel und Orientierung in der Szene modifiziert werden können, bleibt dem Plugin überlassen: Mit Mauszeiger und Keyboard Shortcuts wandert der Benutzer durch eine virtuelle Welt, verkörpert im wahrsten Sinne des Wortes durch einen Avatar, seiner geometrischen Repräsentation, beschränkt in seiner Beweglichkeit durch physikalische Restriktionen und durch eine simulierte Schwerkraft.

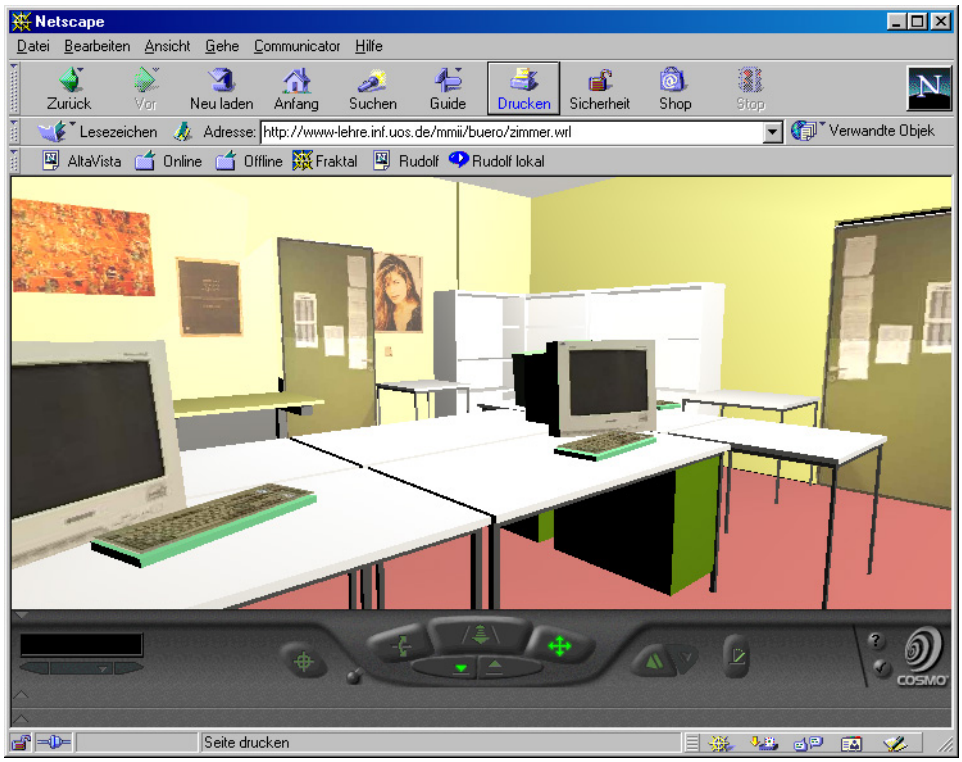


Abbildung 20.1: Screenshot vom Cosmo Player Plugin

20.3 Geometrie

Wichtigster Bestandteil von VRML-Szenen ist der sogenannte Knoten (meistens mit großem Anfangsbuchstaben geschrieben) der ähnlich eines Programmiersprachenrecords aus Feldern verschiedenen Typs besteht (meistens klein geschrieben). Diese Felder verweisen entweder auf nicht weiter strukturierte Objektknoten oder andere Gruppenknoten, die wiederum mittels ihrer Felder verzweigen können.

Beispiel 1 zeigt den Aufbau einer Szene, in der eine Kugel mit Radius 1.5 im Ursprung des Weltkoordinatensystems plaziert wird. Die x-Richtung entspricht der horizontalen Bewegung, y beschreibt die vertikale Richtung und z wächst auf den Betrachter zu. Der Sphere-Knoten hat dabei als einziges (optionales) Feld den Radius. Diese Kugel wird referiert über das geometry-Feld des Shape-Knotens, zuständig für die Gestaltung eines Objekts. Über das appearance-Feld wird die Materialbeschaffenheit in Form einer RGB-Farbe und eines Reflexions-Koeffizienten spezifiziert. Der Shape-Knoten wiederum ist als eins der Kinder im Transform-Knoten eingetragen, der über ein translation-Feld für die Verschiebung der Kugel sorgt.

```
# VRML V2.0 utf8
# kugel.wrl:
# gruene, stark reflektierende  Kugel mit Radius 1.5

Transform {                        # Plaziere
  translation 0 0 0                # im Ursprung
  children [
    Shape {                        # eine Gestalt
      geometry Sphere {            # von der Form einer Kugel
        radius 1.5                 # mit Durchmesser 1.5
      }
      appearance Appearance {      # in der Erscheinung
        material Material {        # mit Materialbeschaffenheit
          diffuseColor 0 1 0       # gruene Farbe
          shininess 0.9            # stark reflektierend
        }
      }
    }
  ]
}
```

kugel.wrl

20.4 Polygone

Neben den Grundbausteinen Sphere (Kugel), Box (Quader), Cone (Kegel) und Cylinder (Zylinder) lassen sich eigene geometrische Gebilde konstruieren. Ausgehend von einer Liste von 3-D-Punkten im Raum werden jeweils gegen den Uhrzeigersinn alle Punkte durchlaufen, die ein Face (durch Polygon approximierte Körperfläche) aufspannen. Beispiel 2 zeigt die Definition einer 5-farbigen Pyramide mit quadratischer Grundfläche.

```
# VRML V2.0 utf8
# pyramide.wrl: selbstdefinierte 5-seitige Pyramide

Shape {
  geometry IndexedFaceSet {

    coord Coordinate {
      point [            # beteiligte Punkte
         0  3  0         # 0. Pyramidenpunkt (Spitze)
         0  0 -2         # 1. Pyramidenpunkt (Norden)
        -2  0  0         # 2. Pyramidenpunkt (Westen)
         0  0  2         # 3. Pyramidenpunkt (Sueden)
         2  0  0         # 4. Pyramidenpunkt (Osten )
      ]
    }

    coordIndex [         # Polygone gegen Uhrzeiger, Ende: -1
         4  3  2  1 -1   # 0. Face: Punkte 4 3 2 1 (Grundflaeche)
         0  1  2    -1   # 1. Face: Punkte 0 1 2   (Nordwesten)
         0  2  3    -1   # 2. Face: Punkte 0 2 3   (Suedwesten)
         0  3  4    -1   # 3. Face: Punkte 0 3 4   (Suedosten)
         0  4  1         # 4. Face: Punkte 0 4 1   (Nordosten)
    ]

    colorPerVertex FALSE
    color Color {
        color [          # pro Face eine Farbe benennen
          0 1 1          # 0. Face: Cyan
          1 0 0          # 1. Face: Rot
          1 1 0          # 2. Face: Gelb
          0 1 0          # 3. Face: Gruen
          0 0 1          # 4. Face: Blau
        ]
    }
  }
}
```

pyramide.wrl

20.5 Wiederverwendung

Zur Reduktion der Dateigrößen und zur besseren Lesbarkeit lassen sich einmal spezifizierte Welten wiederverwenden. Beispiel 3 zeigt die Kombination der beiden graphischen Objekte Kugel und Pyramide, wobei die Pyramide leicht nach hinten gekippt oberhalb der Kugel positioniert wird. Ferner wurde ein Hyperlink eingerichtet, der zu einer weiteren VRML-Welt führt, sobald der Mauszeiger auf der Kugel gedrückt wird.

```
# VRML V2.0 utf8
# gruppe.wrl:
# Kugel mit Hyperlink unter gekippter Pyramide

Transform{                              # Plaziere
  children[                             # die folgenden Objekte:

    Anchor {                            # Hyperlink
      url "multimedia.wrl"              # zur VRML-Datei textur.wrl
      description "Next world"          # Anzeige im Statusfenster
      children[                         # zugeordnet ist die Welt
        Inline {url "kugel.wrl"}        # aus Datei kugel.wrl
      ]
    }

    Transform {
      translation  0 1   0              # hebe 1 Einheit nach oben
      scale        1 0.5 1              # skaliere bzgl. y auf 50 %
      rotation     1 0   0 -0.523333    # kippe um 30 Grad nach hinten
      children[                         # verwende Welt
        Inline {url "pyramide.wrl"}     # aus Datei pyramide.wrl
      ]
    }

  ]
}
```

gruppe.wrl

20.6 Multimedia

Neben geometrischen Strukturen können VRML-Szenen auch multimediale Bestandteile wie Bilder, Audio und Video enthalten.

Beispiel 4 zeigt einen Würfel, auf den ein jpg-Bild als Textur aufgebracht wurde. Einem Sound-Knoten ist per URL eine Wave-Datei mit Position und Schallrichtung zugeordnet.

Sobald der Betrachter bei Annäherung an den Würfel eine gewissen Grenzwert überschritten hat, beginnt der Sound-Knoten mit dem Abspielen der Musik.

```
# VRML V2.0 utf8
# multimedia.wrl:
# Quader mit Bild-Textur und Soundquelle

#VIEWPOINT{0 0 20}                 # 20 Einheiten vor dem Ursprung

Shape {                            # ein Gestaltknoten
  geometry Box {size 1 1 1}        # ein Quader der Kantenlaenge 1
  appearance Appearance {          # mit dem Aussehen
    texture ImageTexture {         # einer Bild-Textur
      url "posaune.jpg"            # aus der JPEG-Datei posaune.jpg
    }
  }
}

Sound {                            # ein Soundknoten
  source AudioClip {               # gespeist von einem Audio-Clip
    url  "party.wav"               # aus der Wave-Datei party.wav
    loop TRUE                      # in einer Endlosschleife
  }

  location  0 0 0                  # Schallquelle im Ursprung
  direction 0 0 1                  # dem Betrachter zugewandt
  minFront 1                       # Hoerbereichsanfang
  maxFront 8                       # Hoerbereichsende
}
```

multimedia.wrl

20.7 Interaktion

VRML97 bietet zahlreiche Möglichkeiten, mit denen einer Szene Dynamik und Interaktion verliehen werden kann. Die zentralen Bausteine für die hierzu erforderliche Ereignisbehandlung sind die EventIn- bzw. EventOut-Felder von Knoten, mit denen Meldungen empfangen und Zustandsänderungen weitergeschickt werden können. Es gibt Time,- Proximity-, Visibility-, Collision- und Touch-Sensoren, welche das Verstreichen einer Zeitspanne, das Annähern des Benutzers, die Sichtbarkeit von Objekten, das Zusammentreffen des Avatars mit einem Objekt und die Berührung mit dem Mauszeiger signalisieren. Verständlicherweise müssen Typ des verschickenden Ereignisfeldes und Typ des empfangenden Ereignisfeldes übereinstimmen.

Beispiel 5 zeigt die Kugel versehen mit einem Touch-Sensor, welcher bei Mausdruck eine Nachricht an den Soundknoten schickt, der auf diese Weise seinen Spielbeginnzeitpunkt erhält und die zugeordnete Wave-Datei startet.

```
# VRML V2.0 utf8
# interaktion.wrl:
# Kugel macht Geraeusch bei Beruehrung

Group {                              # plaziere Gruppenknoten
  children [                         # bestehend aus
    DEF Taste TouchSensor {}         # einem Touch-Sensor
    Inline { url "kugel.wrl" }       # und einer Kugel
  ]
}

Sound {                              # plaziere Soundknoten
  source DEF Tut AudioClip {         # gespeist von Audio-Clip
    url "tut.wav"                    # aus der Wave-Datei tut.wav
  }
  minFront  5                        # Anfang des Schallbereichs
  maxFront 50                        # Ende des Schallbereichs
}

ROUTE Taste.touchTime                # bei Beruehrung der Kugel
   TO Tut.set_startTime              # schicke Systemzeit an den Knoten Tut
```

interaktion.wrl

20.8 Animation

Die benutzergesteuerte oder automatische Bewegung von Objekten und Szenenteilen wird wiederum mit Hilfe der Ereignisbehandlung organisiert. Im Beispiel 6 wird die Ziehbewegung des gedrückten Mauszeigers zur Manipulation der lokalen Translation eines Objekts verwendet und das regelmäßige Verstreichen eines Zeitintervalls löst eine Nachricht an denselben geometrischen Knoten aus, der hierdurch seinen aktuellen Rotationsvektor erhält. Eine solche Konstruktion verlangt einen Orientation-Interpolator, welcher beliebige Bruchzahlen zwischen 0 und 1 auf die zugeordneten Werte seines Schlüsselintervalls abbildet, hier bestehend aus allen Drehwinkeln zwischen 0 und 3.14 (=180 Grad beschrieben in Bogenmaß), bezogen auf die y-Achse.

```
# VRML V2.0 utf8
# animation.wrl:
# selbstaendig sich drehende und interaktiv verschiebbare Pyramide

DEF Schieber PlaneSensor {}            # Sensor zum Melden einer Mausbewegung

DEF Timer TimeSensor {                 # Sensor zum Melden eines Zeitintervalls
  cycleInterval 5                      # Dauer 5 Sekunden
  loop TRUE                            # Endlosschleife
}

DEF Rotierer OrientationInterpolator{  # Interpolator fuer Rotation
  key      [0 , 1]                     # bilde Schluessel 0 und 1 ab auf
  keyValue [ 0  1  0  0                #   0 Grad Drehung bzgl. y
             0  1  0  3.14]            # 180 Grad Drehung bzgl. y
}

DEF Pyramide Transform {               # plaziere Objekt mit Namen Pyramide
      children [                       # bestehend aus
        Inline {url "pyramide.wrl"}    # VRML-Datei pyramide.wrl
      ]
}

ROUTE Timer.fraction_changed           # falls Zeitintervall sich aendert
   TO Rotierer.set_fraction            # schicke Bruchteil an Rotierer

ROUTE Rotierer.value_changed           # falls Drehung sich aendert
   TO Pyramide.set_rotation            # schicke Drehwert an Pyramide

ROUTE Schieber.translation_changed     # falls gedrueckter Mauszeiger bewegt wird
   TO Pyramide.set_translation         # schicke Translationswert an Pyramide
```

animation.wrl

20.9 Scripts

Manchmal reichen die in VRML angebotenen Funktionen wie Sensoren und Interpolatoren nicht aus, um ein spezielles situationsbedingtes Interaktionsverhalten zu erzeugen. Abhilfe schafft hier der sogenannte Script-Knoten, welcher Input empfangen, Daten verarbeiten und Output verschicken kann. Z.B. kann eine vom Touch-Sensor geschickte Nachricht eine Berechnung anstoßen, deren Ergebnis in Form einer Translations-Nachricht an ein bestimmtes Objekt geschickt und dort zur Neupositionierung genutzt wird.

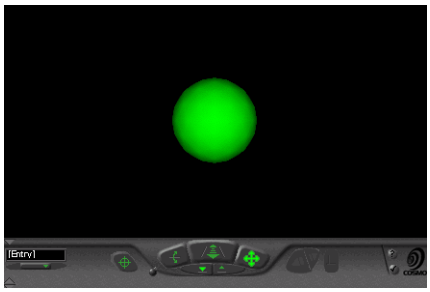
```
# VRML V2.0 utf8
# javascript.wrl: Rotation eines Objekts ueber Javascript

Viewpoint {position  0 2 8}             # Augenpunkt

Group {                                 # gruppiere
  children [                            # folgende Objekte
    DEF Taste TouchSensor{}             # Beruehrungssensor Taste
    DEF Pyramide Transform {            # Objekt Pyramide
      children[                         # bestehend aus
        Inline {url "pyramide.wrl"}     # VRML-Welt pyramide.wrl
      ]
    }
  ]
}

DEF Aktion Script {                     # Script mit Namen Aktion
  eventIn  SFBool     isActive          # Input-Parameter
  eventOut SFRotation drehung           # Output-Parameter
  url [                                 # gespeist von
     "javascript:                      // inline-Javascript
     function isActive(eventValue) {   // fuer eventIn zustaendig
       if (eventValue == true) {       // falls eventValue den Wert wahr hat
         drehung[0] = 0.0;             // drehe
         drehung[1] = 1.0;             // bzgl.
         drehung[2] = 0.0;             // der y-Achse
         drehung[3] += 0.174444;       // um weitere 10 Grad
       }
     }"
  ]
}

ROUTE Taste.isActive                    # bei Beruehren der Pyramide
   TO Aktion.isActive                   # sende Nachricht an das Script Aktion

ROUTE Aktion.drehung                    # vom Script Aktion erzeugter Dreh-Vektor
   TO Pyramide.set_rotation             # wird an die Pyramide geschickt
```

javascript.wrl

Die Formulierung des Berechnungsalgorithmus geschieht entweder durch ein Javascript-Programm, inline gelistet im Script-Knoten, oder durch eine assoziierte Java-Klasse, welche in übersetzter Form

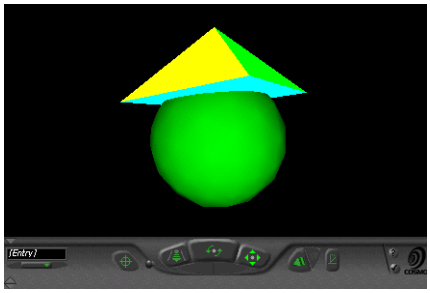
mit Dateiendung *.class lokal oder im Netz liegt. Zum Übersetzen der Java-Quelle ist das EAI (External Authoring Interface) erforderlich, welches in Form einiger Packages aus dem Verzeichnis importiert wird, in welches sie das VRML-Plugin bei der Installation deponiert hatte.

Beispiel 7 zeigt die Pyramide zusammen mit einem Java-Script, welches bei jedem Aufruf den Drehwinkel bzgl. der y-Achse um weitere 10 Grad erhöht.



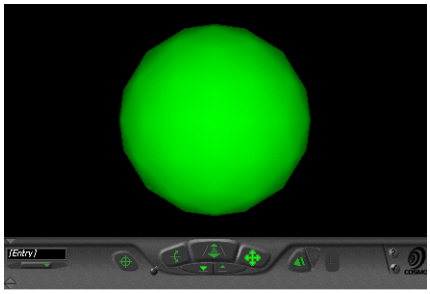
kugel.wrl

pyramide.wrl

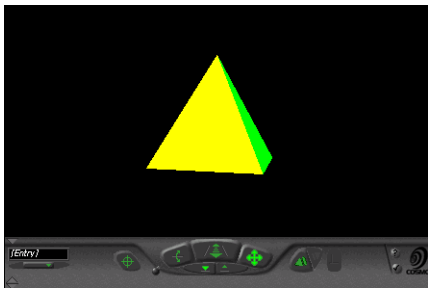


gruppe.wrl

multimedia.wrl



interaktion.wrl



animation.wrl

Abbildung 20.2: Screenshots der Beispiele 1 - 6

20.10 Multiuser

Eines der ursprünglichen Entwicklungsziele von VRML bleibt auch bei VRML 97 offen: es gibt noch keinen Standard für Multiuser-Welten. Hiermit sind Szenen gemeint, in denen mehrere Benutzer gleichzeitig herumwandern und interagieren können. Da jedes ausgelöste Ereignis von allen Beteiligten wahrgenommen werden soll, muß ein zentraler Server den jeweiligen Zustand der Welt verwalten und den anfragenden Klienten fortwährend Updates schicken. In diesem Zusammenhang erhält der Avatar eine aufgewertete Rolle: Zusätzlich zu seiner geometrischen Räumlichkeit, die schon zur Kollisionsdetektion in einer Single-User-Szenerie benötigt wurde, muß nun auch sein visuelles Äußeres spezifiziert werden, sicherlich ein weiterer wichtiger Schritt zur Verschmelzung eines real existierenden Benutzers mit der von ihm gespielten Rolle.

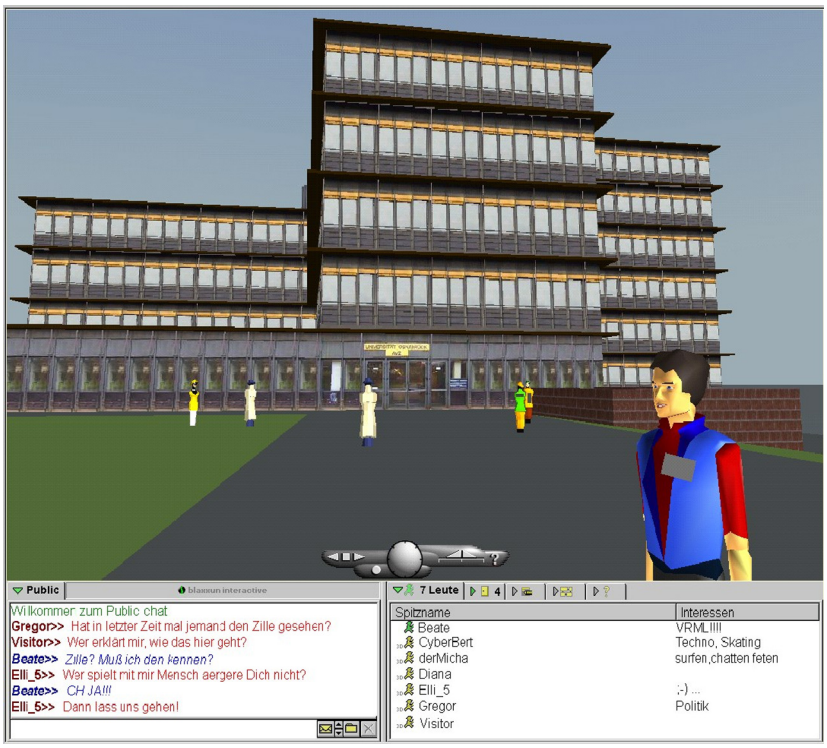


Abbildung 20.3: Screenshot vom Blaxxun Contact Plugin

Unter der Einstiegsseite `http://www-lehre.inf.uos.de/avz` kann das *Virtuelle AVZ* der Universität Osnabrück betreten werden, welches mit Hilfe eines Blaxxun-Community-Servers als Multiuser-Welt realisiert wurde.

Unter der Adresse `http://www-lehre.inf.uos.de/~cg/2006/VRML/uebersicht.html` sind die vorgenannten Dateien und weitere VRML-Beispiele abrufbar.

20.11 X3D

Als Nachfolger von VRML ist X3D vorgesehen, eine Beschreibungssprache für 3D-Welten auf Basis von XML. Ein Werkzeug zum Erstellen von X3D-Dateien bietet `http://www.vizx3d.com;` dort gibt es auch den Flux-Player zum Anzeigen von X3D-Welten im Web-Browser.

```
#VRML V2.0 utf8

Transform {
  translation  -0.03   0.00  -0.052
  rotation      0.82  -0.56  -0.039  2.10
  children [
    Shape {
      appearance Appearance {
        material Material {
          ambientIntensity 0.2
          shininess        0.2
          diffuseColor     1 0 0
        }
      }
      geometry Box {
        size 1 1 1
      }
    }
  ]
}
```

Beschreibung eines roten Würfel in VRML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
  "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D>
  <Scene>
    <Transform translation= "-0.03   0.00  -0.052"
               rotation=    " 0.82  -0.56  -0.039  2.10">
    <Shape>
      <Appearance>
        <Material ambientIntensity  ="0.2"
                  shininess         ="0.2"
                  diffuseColor      ="1 0 0" />
       </Appearance>
       <Box size="1 1 1"/>
    </Shape>
    </Transform>
  </Scene>
</X3D>
```

Beschreibung eines roten Würfel in X3D