

Kapitel 2

Koordinatensysteme und Transformationen mit VRML

2.1 Lernziele

In dieser Übung werden Sie das bisher vermittelte Wissen zu Koordinatensystemen und Transformationen aus der Vorlesung praktisch anwenden und vertiefen. Sie werden hierzu die Grundzüge der 3D-Beschreibungssprache *VRML* kennenlernen und danach selbst Quellcode schreiben.

2.2 Werkzeuge dieser Übung

ParallelGraphics Vrml-Pad - Zum Erstellen und Bearbeiten von *VRML*-Quellcode benutzen wir im Praktikum im Wintersemester 2003/2004 das *Vrml-Pad*, Version 1.3. Weitere Informationen und das *Vrml-Pad* kostenlos zum download finden Sie unter der URL des Herstellers:

<http://www.parallelgraphics.com/products/vrmlpad>

blaxxun CONTACT - Zum Anzeigen einer *VRML*-Datei benötigen Sie einen Browser und ein *VRML*-PlugIn. Im Praktikum im Wintersemester 2003/2004 benutzen wir hierzu den *Internet Explorer 6* und das PlugIn *blaxxun CONTACT* in der Version 5.1. Das Plug-In und weitere Informationen finden Sie unter der URL des Herstellers:

<http://www.blaxxun.com>

Dort können Sie das Plug-In kostenlos herunterladen.

2.3 Vorausgesetztes Wissen

Folgende Inhalte sollten Ihnen bekannt sein:

- Koordinatensysteme – welche es gibt, wozu sie dienen, Zusammenhänge mit Transformationen (auch die Inhalte der Vorlesung *Mathematik*).
- Transformationen – welche es gibt, wozu sie dienen, Zusammenhänge (auch Inhalte der Vorlesung *Mathematik*).
- Das *RGB-Farbmodell* (Inhalt der Vorlesung *Medientechnik*).

2.4 Weiterführende Literatur

Um mehr über *VRML* zu erfahren, seien Ihnen folgende Bücher empfohlen, die alle in der Bibliothek des Standorts Zweibrücken vorhanden sind:

- *The annotated VRML 2.0 reference manual* [BC97], das Standardwerk zu *VRML*.
- *Dynamische virtuelle Welten mit VRML 2.0* [Has97], ausnahmsweise ein deutsches Buch zu *VRML*.
- *VRML 2.0 sourcebook* [ANM97] *VRML*.

Im Internet finden Sie weitere Informationen unter folgenden URLs:

- Die offizielle *VRML97* Spezifikation (ISO/IEC 14772):
<http://tecfa.unige.ch/guides/vrml/vrml97/spec>
- *Floppy's VRML97 Tutorial*, ein gelungenes Einsteiger-Tutorial für *VRML* finden Sie unter:
<http://web3d.vapourtech.com/tutorials/vrml97>

2.5 Einführung in VRML

2.5.1 Über VRML

VRML steht für *Virtual Reality Modelling Language*. Sie ist eine Beschreibungssprache, die eine interaktive, dreidimensionale Darstellung von komplexen 3D-Welten bzw. 3D-Objekten ermöglicht. Objekte können statisch oder animiert sein. Man kann dabei auf vordefinierte Objekte zurückgreifen oder sich benutzerdefinierte Objekte selbst erstellen. Objekte werden durch ihre Eigenschaften beschrieben (Aussehen, Material, Geometrie). Die Einbindung von Sound- und Video-Dateien ist ebenso möglich wie die Einbindung in JavaScript oder Java. Die aktuelle Version 2.0 ist auch unter dem Namen *VRML97* bekannt und ist ISO-Norm.

2.5.2 Konventionen von VRML

Dateiformat

- *VRML*-Dateien haben die Dateierweiterung `.wrl` (*world*).
- *VRML*-Dateien sind immer reine ASCII-Text-Dateien.

Syntax

- *VRML* ist Case-Sensitive.
- Ein Kommentar beginnt mit einem `#` und endet am Zeilenende.
- Namen dürfen in *VRML* nicht mit einer Zahl beginnen oder folgende Zeichen enthalten:

`" ' # + . - , [] { }`

Maßeinheiten und Koordinaten

- Lineare Distanzen werden in Metern angegeben, Zeitangaben in Sekunden.
- Farbwerte werden durch das RGB-Modell dargestellt (Werte zwischen 0.0 und 1.0).
- Kommazahlen werden nicht mit einem Komma, sondern mit ein Punkt getrennt.
- *VRML* verwendet ein rechtshändiges Koordinatensystem.
- Winkel werden in Bogenmaß angegeben (siehe Tabelle 2.1).

Grad	Bogenmaß
0	0
45	0.78
90	1.57
135	2.36
180	3.14
225	3.93
270	4.71
315	5.5
360	6.28

Tabelle 2.1: Grad in Bogenmaß

2.5.3 Der Header einer VRML-Datei

Jede VRML-Datei muss mit folgender Zeile beginnen:

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

Dadurch erkennt das Browser-Plugin, dass es sich um VRML-Code der Version 2.0 im 8-bit UCS (Universal Character Set) Transformation Format-Zeichensatz enthält.

2.5.4 Der Szenengraph in VRML

Eine VRML-Datei besitzt eine Baumstruktur, den *Szenengraph*. Die wichtigsten Bestandteile des *Szenengraphen* sind:

Knoten

Ein Szenengraph besteht aus einem oder mehreren *Knoten* (*nodes*) und wird gefolgt von einem Paar geschweifter Klammern. Knoten werden im VRML-Code für gewöhnlich mit einem großen Anfangsbuchstaben geschrieben. Es gibt verschiedene Typen von Knoten (*Group*, *Shape*, *Transform*, ...), sie werden im VRML-Code für gewöhnlich mit einem großen Anfangsbuchstaben geschrieben. Eine Auflistung aller verfügbaren Knoten können Sie der offiziellen VRML97-Spezifikation (siehe Kapitel 2.4) entnehmen. Ein einfaches Beispiel für einen Knoten ist:

```
Group{
}
```

Innerhalb eines Knotens können weitere Unterknoten folgen, die sog. *children*. Diese werden nach dem Schlüsselwort *children*, in zwei eckigen Klammern deklariert.

Beispiel für einen Knoten vom Typ *Group* mit einem Unterknoten:

```
Group {
  children [
    Cylinder {
    }
  ]
}
```

Gruppenknoten

Die für uns sehr interessanten Knoten, sind die sog. *Gruppenknoten*. Mit ihnen lassen sich mehrere Kind-Knoten zusammenfassen. Dies macht es einfacher, komplexe Objekte zu beschreiben, die aus mehreren Objekten zusammengesetzt sind (wie z. B.: *Geometrie-Knoten* wie *Cylinder*, *Box* oder *Sphere*). Zudem wird es wesentlich leichter, Transformationen auf ein komplexes Objekt anzuwenden.

Felder

Jeder Knoten hat *Felder* (*fields*), welche die Knotenattribute beschreiben. Jedes Feld besitzt einen Feld-Namen, einen Datentyp und einen Anfangswert (*Default*). Werden bei der Knotendeklaration Felder nicht parametrisiert, werden automatisch die dafür in der Spezifikation festgelegten Default-Werte genommen. Felder werden im VRML-Code für gewöhnlich mit einem kleinen Anfangsbuchstaben geschrieben. Welche Felder für welchen Knoten standardmäßig zur Verfügung stehen und welche Default-Werte diese haben, entnehmen Sie der offiziellen VRML97-Spezifikation (siehe Kapitel 2.4). Hier ein Beispiel, wobei zwei Felder des *Geometrie-Knotens* jetzt parametrisiert sind:

```
Group {
  children [
    Cylinder {
      radius 0.2
      height 1
    }
  ]
}
```

2.5.5 Transformationen in VRML

Um Transformationen in VRML anwenden zu können, benötigen Sie den Knoten `Transform`. Dieser hat drei Felder, die drei Transformationen darstellen. Jedes dieser Felder hat folgende Parameter:

- `scale` – drei Parameter, einen pro Achse des Koordinatensystems, eine „1“ bedeutet nicht skaliert. Eine „0“ führt zu Anzeigefehlern.
- `rotation` – vier Parameter, die ersten drei legen die Rotationsachse fest, der vierte den Rotationswinkel in Bogenmaß.
- `translation` – drei Parameter, einen pro Achse des Koordinatensystems.

Die Transformation wird auf alle Knoten angewandt, die Unterknoten des `Transform`-Knotens sind und zwar in der oben genannten Reihenfolge.

Hier ein Beispiel für eine dreifache Transformation:

```
Transform {
  translation 1 1 1
  rotation 0 1 0 0.78
  scale 2 1 2

  children [
    Cylinder {
      radius 0.2
      height 1
    }
  ]
}
```

2.5.6 Der Knoten 'Shape' in VRML

Ein für uns ebenfalls wichtiger Knoten ist der Knoten `shape`. Er ist für die Darstellung von 3D-Objekten vorgesehen. Die meisten 3D- Objekte in einer VRML-Welt bestehen aus Shapes, die durch ihre Geometrie- Eigenschaft (`geometry`) und ihr Erscheinungsbild (`appearance`) beschrieben werden. So ist es möglich entweder einfache geometrische Objekte (z.B. `Cylinder`), komplexere Gebilde wie Extrusionen oder Verbundobjekte zu erzeugen und diesen dann ein bestimmtes Erscheinungsbild zu geben (Materialeigenschaften wie Farbe, Transparenz, Spiegelung oder Oberflächeneigenschaften wie Texturen).

```
Shape{
  geometry Cylinder {
    radius 0.2
    height 1
  }
}
```

```
}
}
```

Den Feldern (`geometry`) und (`appearance`) folgen in diesem Beispiel keine Parameter in Form von Werten, wie Sie das von anderen Feldern kennen, sondern ein *Geometrie-Knoten*. In diesem Beispiel wird unter dem Feld (`geometry`) ein *Geometrie-Knoten* vom Typ `Cylinder` definiert, um einen Zylinder zu erzeugen.

2.5.7 Materialien und Texturen in VRML

Um die Oberflächeneigenschaften eines Objekts zu beschreiben nutzen Sie das Feld (`appearance`), dass zum Knoten `shape` gehört. Dieses Feld hat als Parameter gewöhnlich den Knoten (`Appearance`). In diesem finden Sie Felder zum Erscheinungsbild des Objekts, wie das Feld `material`, mit dem Knoten `Material`.

```
Shape{
  appearance Appearance {
    material Material {
    }
  }
  geometry Cylinder {
    radius 0.2
    height 1
  }
}
```

Da hier der Knoten `Material` nicht näher parametrisiert wurde, verwendet *VRML* die Default-Werte.

Um dem Objekt eine bestimmte Farbe zu geben, wird dem Knoten `Material` folgende Parameter mitgegeben:

```
Shape{
  appearance Appearance {
    material Material {
      diffuseColor 0 0 1
    }
  }
  geometry Cylinder {
    radius 0.2
    height 1
  }
}
```

In diesem Fall wurde dem Feld `diffuseColor` (gibt die Farbe des Objekts im RGB-Farbmodell an) die Werte: 0% Rot, 0% Grün, 100% Blau gegeben. Näheres zum Knoten `Material` können Sie der offiziellen *VRML97*-Spezifikation entnehmen.

2.5.8 Benutzerdefinierte Knoten in VRML

Wenn in einer *VRML*-Szene mehrere gleiche Objekte auftreten, ist es sinnvoll, ein Objekt einmal zu definieren und dann immer wieder aufzurufen. Dazu bietet *VRML* die Funktion `DEF` an, die hier aber nicht näher behandelt werden soll. Interessanter ist die Weiterentwicklung dieser Funktion, die sog. *Prototypen*. Durch Prototypen haben Sie die Möglichkeit, ein Objekt einmal als Prototyp zu definieren, wobei die Parameter von beliebigen Attributen durch Variablen er-

setzt werden können. Die Parameter werden dann bei Aufruf des Prototyps übergeben. Dieses Konzept ähnelt dem von objektorientierten Programmiersprachen.

```
PROTO myCylinder[
  field SFVec3f myCylinderTranslation 0.0 0.0 0.0
] {

  Transform {
    translation IS myCylinderTranslation
    children [
      Shape{
        appearance Appearance {
          material Material {
            diffuseColor 0 0 1
          }
        }
        geometry Cylinder {
          radius 0.2
          height 1
        }
      }
    ]
  }
}

myCylinder{}
```

Nach dem Schlüsselwort **PROTO** wird hier ein Prototyp mit der Bezeichnung `myCylinder` deklariert. Der Prototyp hat ein variables Feld vom Typ `SFVec3f` (einen 3D-Vektor mit Float-Werten) und der Bezeichnung `myCylinderTranslation` und Default-Werten. Dann wird der Prototyp implementiert, wobei die Variable `myCylinderTranslation` einbezogen wird. Am Ende wird der Prototyp einmal (mit Default-Werten) aufgerufen.

2.7 Übungsaufgaben

2.7.1 Zur Vorbereitung vor dem Praktikum

1. Wir erwarten, dass Sie das Kapitel 2.5 vorher gelesen haben, damit Sie wissen was im Praktikum auf Sie zukommt und direkt mit dem praktischen Teil anfangen können.
2. Machen Sie sich mit der in Kapitel 2.5 angesprochenen Theorie vertraut.
3. Zum besseren Verständnis schauen Sie sich aus dem in Kapitel 2.4 empfohlenen Tutorial *Floppy's VRML97 Tutorial* den ersten Teil *Part 1: The Basics* an. Wichtig sind hier für diese Übung die Punkte 1.3 (Koordinatensysteme und Transformationen) und 1.7 (Prototypen).

2.7.2 Während des Praktikums

1. Öffnen Sie *Vrml-Pad*. Geben Sie die in Kapitel 2.5 beschriebenen Beispiele (außer des Beispiels in Kapitel 2.5.8) nacheinander per Hand ein und lassen Sie sich das Ergebnis im Browser anzeigen (F5-Taste). Wenn es sich anbietet, nehmen Sie sinnvolle Veränderungen vor (ändern Sie z.B. einen Parameter) und beobachten Sie die Veränderungen. Machen Sie sich so mit dem *Vrml-Pad* und mit der Bedienung des *blaxxun CONTACT* vertraut.
2. Unter

```
\\gauss\public\cav\vrml\Aufgabenstellung
```

finden Sie die Datei `lego.zip`. Kopieren Sie diese in Ihr Home-Verzeichnis auf `gauss`. Sie enthält die beiden Dateien `lego.wrl` und `legosteine.wrl`.
3. Öffnen Sie die Datei `lego.wrl` im Browser und im *Vrml-Pad*. In dieser Datei werden zwei externe Prototypen importiert, die einen großen und einen kleinen Lego-Stein darstellen. Beide werden jeweils einmal aufgerufen.
4. Bauen Sie ein Lego-Objekt, das aus mindestens 10 Lego-Steinen in verschiedenen Farben besteht. Fertigen Sie dazu zunächst eine Skizze auf kariertem Papier an.
5. Öffnen Sie die Datei `legosteine.wrl` im Browser und im *Vrml-Pad*. In dieser Datei finden Sie Implementierungen der zwei Prototypen für die Datei `lego.wrl`. Schauen Sie sich den Quellcode an und versuchen Sie ihn nachzuvollziehen.
6. Versuchen Sie die Datei `legosteine.wrl` um einen weiteren Prototypen `Legoschmal` zu erweitern (siehe Abbildung 2.2). Benutzen Sie hierzu einen der bestehenden Prototypen als Vorlage und bauen Sie den neuen Lego-Stein danach in ihr bestehendes Lego-Objekt ein.

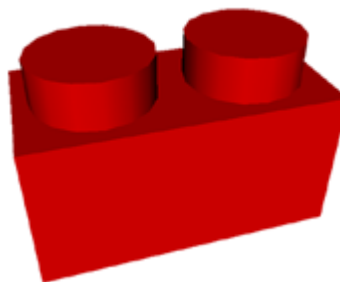


Abbildung 2.2: Ein VRML-Leogstein

2.7.3 Abgabe

Erstellen Sie unter

```
\\gauss\public\cav\vrml\Abgabe
```

einen Ordner (als Namen nehmen Sie die Nachnamen der Teammitglieder, getrennt durch ein "&„) und speichern Sie dort die von Ihnen erstellten *VRML*-Dateien (nur die Endversionen) ab.

2.7.4 Nachbereitung

Nach dem Praktikumstermin sollte Ihnen der prinzipielle Aufbau und Syntax eines *VRML*-Programms vertraut sein. Es ist zu empfehlen, mit dem *Vrml-Pad* und *CONTACT* zu Hause weiter zu experimentieren, um noch vertrauter mit *VRML* zu werden.

Zur Nachbereitung beantworten Sie bitte folgende Fragen und geben diese in schriftlicher Form auf einem DIN A4 Blatt mit ihrem Namen am nächsten Praktikumstermin im VisLab ab. Benutzen Sie hierzu die Vorlage, den Sie mit diesen Unterlagen erhalten haben.

1. Was ist Ihnen beim erstellen Ihres LEGO-Objekts aufgefallen? Welche Probleme sind aufgetreten?
2. Beschreiben Sie den Aufbau einer *VRML*-Datei.
3. Beschreiben Sie mit Hilfe der angegebenen Literatur die in *VRML* vorgesehenen Möglichkeiten für dynamische und interaktive Welten. Erstellen Sie zu Hause hierzu ein kleines Beispiel und kopieren Sie es im nächsten Praktikumstermin in das Abgabe-Verzeichnis zu ihrem Lego-Objekt.

Literaturverzeichnis

- [ANM97] AMES, ANDREA L., NADEAU, DAVID R. und MORELAND, JOHN L.: *VRML 2.0 sourcebook*. Wiley, 1997.
- [BB03] BENDER, MICHAEL und BRILL, MANFRED: *Computergrafik. Ein anwendungsorientiertes Lehrbuch*. Hanser, 2003.
- [BC97] BELL, GAVIN und CAREY, RIKK: *The annotated VRML 2.0 reference manual*. Addison-Wesley, 1997.
- [Bri01] BRILL, MANFRED: *Mathematik für Informatiker*. Hanser, 2001.
- [FDFH91] FOLEY, JIM, DAM, ANDRIES VAN, FEINER, STEVEN und HUGHES, JOHN: *Computer Graphics – Principles and Practice*. Addison-Wesley, 1991.
- [Has97] HASE, HANS-LOTHAR: *Dynamische virtuelle Welten mit VRML 2.0*. dpunkt, 1997.
- [Hil01] HILL, FRANCIS: *Computer Graphics using OpenGL*. Prentice Hall, 2001.
- [Wat99] WATT, ALAN: *3D Computer Graphics*. Addison-Wesley, 1999.
- [Wat01] WATT, ALAN: *3D Computergrafik*. Pearson Education, 2001.
- [Wol00] WOLFE, ROSALEE: *Computer Graphics – A Visual Approach*. Oxford University Press, 2000.