

Virtual Reality Modeling Language

VRML ist keine Programmiersprache, sondern eine Seitenbeschreibungssprache und zugleich auch eine Skriptsprache.

Die Vorteile von VRML sind:

- 3D „Sprache“ fürs Internet
- Einfach zu lernen
- Animationen und Interaktionen
- User kann sich frei im „Raum“ bewegen =>ideal für Firmen und die Wissenschaft
- Zum Programmieren braucht man keine speziellen Programme (nur Texteditor!)

Die Nachteile von VRML sind:

- Damals waren die Computer zu langsam und heute ist die Rendertechnik überholt
- Komplexere Objekte
- Ein Plug- In ist notwendig
- Kein einheitlicher Browser, der alles 100% richtig darstellt
- Navigation im „Raum“ ist recht schwer, ungenau und kompliziert
- Schlechte Zusammenarbeit mit anderen Programmiersprachen
- Datenbanken können nicht gut eingesetzt werden

In VRML wird (!) zwischen Groß und Kleinschreibung unterschieden. Die Syntax ist recht einfach und nach der „Baumstruktur“ aufgebaut. Ein großer Vorteil beim programmieren ist, dass es immer für alle Objekte Voreinstellungen gibt (Default), die man einfach „weglassen“ kann! Das Koordinaten System funktioniert nach dem rechtsdrehenden, kartesischen Koordinatensystem (das „normale“ System). VRML rechnet mit „VRML- Einheiten“, die man selbst definieren kann (Meter, Sekunden...)

Wie beim Haus die Ziegeln als Grundbausteine dienen, sind die Knoten in VRML die „Grundbausteine“. Sie beschreiben ein Objekt und definieren dessen Eigenschaften. Zuerst kommt immer die Typbeschreibung, gefolgt von { } und den in den Klammern befindlichen Attribute (die Reihenfolge der Attribute ist egal). Man kann die Namen der Knoten selbst festlegen:

DEF Knotenname (eigene Wahl) Geometrieknoten (festgelegt !) {Eigenschaften}

Um VRML zu programmieren braucht man nur einen Texteditor. Den Quelltext kann man selbst formatieren (genau wie in HTML),den dies hat keine Auswirkungen auf das Aussehen („Erscheinungsbild“ im Browser). Damit der Browser weiß, dass es sich um VRML- Code handelt und das Plug- In lädt, muss man in jeder VRML- Datei einen Header schreiben. Dieser Header liefert an den Browser Informationen über die Programmiersprache, die Version und die „Textart“ (ASCII, utf...). Der Header (1.Zeile) für VRML schaut so aus:

**#VRML V2.0 utf8 für VRML 2 und
#VRML V1.0 ascii für VRML 1**

Nun ein kleines Beispiel: Ich möchte eine einfache Kugel erstellen, um die man sich frei herum bewegen kann. Da das nur ein kleines Beispiel sein soll, laß ich speziellere Angaben weg.

#VRML V2.0 utf8

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

Wenn ich nun aber eine Kugel haben möchte, die einen anderen Radius hat, muss ich die Zeile „geometry Sphere {}“ ändern und die Eigenschaft „radius“ verwenden =>

```
  geometry Sphere {radius 4.0}
```

Nun hab ich eine 3D Kugel mit dem Radius 4. Das # „zeigt“, dass es ein Kommentar ist.

Damit das Alltagsleben eines „VRML- Programmierers“ leichter wird, wurden sogenannte „geometrische Primitive“ eingeführt. Dabei handelt es sich um einfache „Grundobjekte“, aus denen man viele andere Objekte zusammensetzen kann. Hier eine Liste mit allen Grundobjekten mit ihren Eigenschaften („Zahl“ bedeutet, dass man eine reelle, positive Zahlen eingeben muss, getrennt durch einen Beistrich!):

Box...ist ein Quader

size...Abmessung des Quaders

Cone...ist ein Kegel

bottomRadius Zahl.Zahl...Radius des Grundkreises

height Zahl.Zahl...Höhe des Kegels

side TRUE/FALSE...wird der Kegelmantel dargestellt?

bottom TRUE/FALSE...wird der Grundkreis dargestellt?

Cylinder...ist ein Zylinder

radius Zahl.Zahl...Radius des Zylinders

height Zahl.Zahl...Höhe des Zylinders

bottom TRUE/FALSE...Wird die Grundfläche dargestellt?

top TRUE/FALSE... Wird die Deckfläche dargestellt?

side TRUE/FALSE... Wird der Mantel dargestellt?

Sphere...ist eine Kugel

radius Zahl.Zahl...Radius der Kugel

Text...Text in Form von einem Objekt

string []...eine oder mehrere Zeichenketten

fontStyle...Hier könnte man nähere Angaben zu Schriftart, Größe...machen

length Zahl.Zahl...Länge der Zeichenkette (bei Werten $\neq 0$ kommt es zu einer Skalierung!)

maxExtent Zahl.Zahl...Obergrenze für die Ausdehnung der Zeichenkette (bei Werten $\neq 0$ => kommt es zu einer Skalierung)

ElevationGrid...“Einfache“ Darstellung von Geländeformationen

color „Farbwert“...Farbwerte pro Eckpunkt oder Flächenelement kann angegeben werden

height []...Zweidimensionale Liste mit den einzelnen Höhenwerten

„Koordinatenrichtung“Dimension Zahl...Anzahl der Felder in „Koordinaten“- Richtung

„Koordinatenrichtung“Spacing Zahl.. Ausdehnung eines Feldes in „Koordinatenrichtung“.

Hier nun ein kleines Beispiel für „ElevationGrid“:

```
Shape {
  appearance Appearance {
    material Material {}
  }
  geometry ElevationGrid {
    xDimension 5 #es gibt 5 Felder in der x Richtung
    zDimension 5 #es gibt 5 Felder in der y Richtung
    xSpacing 1.0 #ein Feld ist eine Einheit in x Richtung breit
    zSpacing 1.0 #ein Feld ist eine Einheit in z Richtung breit
    height [
      0.0 0.0 0.0 0.0 0.0, #hier steht die Liste mit den „Höhenpunkten“
      0.0 1.0 1.5 1.0 0.0,
      0.0 0.5 2.5 1.5 0.0,
      0.0 1.0 3.0 1.0 0.0,
      0.0 0.0 0.0 0.0 0.0
    ]
  }
}
```

Durch dieses Skript wird ein Art Berg dargestellt. Dies geschieht, indem das ElevationGrid wie ein Schachbrett verwendet wird. Auf die einzelnen Felder werden dann die Höhenkoordinaten eingetragen. Dadurch ergibt sich ein Höhenprofil, das noch zur Glättung interpoliert wird. Texturen bedecken dabei die gesamten Flächen.

Wenn man nun aber komplexere Objekte erstellen möchte, muss man sich mit Punktwolken auseinandersetzen. Dabei erstellt man einzelne Punkte im 3D Raum, die man dann beliebig weiterarbeiten kann. Dazu braucht man den Geometrieknoten PointSet
Zur Erklärung hab ich ein kleines Programm geschrieben, dass die 8 Eckpunkte eines Würfels definiert. Zur Erklärung hab ich es „kommentiert“!

```
#VRML V2.0 utf8
Background { skyColor 1.0 1.0 1.0 } # Der Knoten Background definiert Hintergrund
Shape {
  appearance Appearance {
    material Material { }
  }
  geometry PointSet { # Der Knoten PointSet wird „geöffnet“
    coord Coordinate {
      point [ #der Unterknoten point zum definieren der Punkte wird geöffnet
        -1.0 1.0 1.0, # Punkt links oben vorn wird definiert
        -1.0 -1.0 1.0, # Punkt links unten vorn wird definiert
        1.0 -1.0 1.0, # Punkt rechts unten vorn wird definiert
        1.0 1.0 1.0, # Punkt rechts oben vorn wird definiert
        -1.0 1.0 -1.0, # Punkt links oben hinten wird definiert
        -1.0 -1.0 -1.0, # Punkt links unten hinten wird definiert
        1.0 1.0 -1.0, # Punkt rechts oben hinten wird definiert
        1.0 -1.0 -1.0 # Punkt rechts unten hinten wird definiert
      ]
    }
  }
}
```

```

    }
  }
}

```

Da ich jetzt aber einen „ganzen“ Würfel haben möchte, also mit 12 Kanten, brauche ich Linie als Verbindung. Deshalb bediene ich mich dem Knoten **coordIndex** []

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

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

```
Shape {
```

```
  appearance Appearance {
```

```
    material Material { }
```

```
  }
```

```
  geometry IndexedLineSet {
```

```
    coord Coordinate {
```

```
      point [
```

```
        -1.0 1.0 1.0, # „Punkt“ 0 für unten (coordIndex)!!!
```

```
        -1.0 -1.0 1.0, # „Punkt“ 1 für unten (coordIndex)!!!
```

```
        1.0 -1.0 1.0, # „Punkt“ 2 für unten (coordIndex)!!!
```

```
        1.0 1.0 1.0, # „Punkt“ 3 für unten (coordIndex)!!!
```

```
        -1.0 1.0 -1.0, # „Punkt“ 4 für unten (coordIndex)!!!
```

```
        -1.0 -1.0 -1.0, # „Punkt“ 5 für unten (coordIndex)!!!
```

```
        1.0 -1.0 -1.0, # „Punkt“ 6 für unten (coordIndex)!!!
```

```
        1.0 1.0 -1.0 # „Punkt“ 7 für unten (coordIndex)!!!
```

```
      ]
```

```
    }
```

```
    coordIndex [ # „-1“ ist ein Trennwert/damit PC weiß: Liniendefinition abgeschlossen
```

```
      0, 1, 2, 3, 0, -1, # vorderes Quadrat („Punkte“ 0, 1, 2, 3, 0 verbinden)
```

```
      4, 5, 6, 7, 4, -1, # hinteres Quadrat („Punkte“ 4, 5, 6, 7, 4 verbinden)
```

```
      0, 4, -1, # Kante links oben („Punkte“ 0,4 verbinden)
```

```
      1, 5, -1, # Kante rechts unten („Punkte“ 1,5 verbinden)
```

```
      2, 6, -1, # Kante rechts unten („Punkte“ 2,6 verbinden)
```

```
      3, 7 # Kante rechts oben („Punkte“ 3,7 verbinden)
```

```
    ]
```

```
  }
```

```
}
```

Diese Beispiel „zeigt“, außer einem Würfel noch 4 Sachen:

1. Auch der Quelltext von „einfache“ Objekte kann schnell lang und kompliziert werden!
2. Die Grundobjekte sind praktisch! Ich hätte das selbe auch mit einem Grundobjekt machen könne und hätte mir die ganze Arbeit erspart!!!!
3. Eine gute Quelltextstrukturierung ist sehr wichtig! Ohne ihr würde man schnell die Übersicht verlieren und Klammern vergessen zu öffne bzw. zu schließen!
4. Da ich vielleicht das Programm später noch einmal brauchen könnte (da ich es in einem anderen einbauen möchte, erweitern oder verändern möchte), muss ich den Quelltext auch gut Kommentieren!!! Ansonsten hat man nach Jahren keine Ahnung mehr, was zum Beispiel in der Zeile **coordIndex[]** passiert. Es werden nämlich nicht die Koordinaten der Punkte verwendet, sondern die Punkte selbst!!!