Modellierung virtueller 3D-Welten für das Internet

Verfasser: Prof. Dr. D. Hannemann, Hermann-Löns-Str.6, 45849 Gelsenkirchen. eMail: 100302.1665@compuserve.com.
Internet-Homepage: http://www.DieterHannemann.de

Dieser Aufsatz will zeigen, wie mit den modernen Hilfsmitteln, die im Zusammenhang mit dem Internet entstehen, dreidimensionale virtuelle Welten geschaffen werden können, in denen physikalische Prozesse ablaufen. Neben einer allgemeinen, mehr philosophisch gehaltenen Einführung werden kurz die wichtigsten Elemente einer Sprache beschrieben, mit der man diese Welten modellieren kann. Die kurze Einführung in diese Sprache reicht aus, um an einem Beispiel den Fall eines Balls nachbilden zu können. Die Behandlung dieses Themas kann anregen, sich mit den Möglichkeiten dieser neuen Techniken zur Darstellung physikalischer Abläufe zu beschäftigen.

1 Einführung

Der Begriff virtuell begegnet uns z.B. in der Optik: Wenn man sich in einem Spiegel betrachtet und dort sein Bild sieht, so glaubt man, einer Person gegenüber zu stehen, die sich räumlich hinter dem Spiegel befindet. Wie man bereits als Kind gelernt hat, ist diese dort im Spiegel sichtbare Realität jedoch nicht wirklich hinter dem Spiegel vorhanden. Man spricht in diesem Fall von einem virtuellen Bild im Gegensatz zu einer Abbildung wie sie z.B. durch eine Linse erzeugt wird und die man auf einem Schirm abbilden bzw. auffangen kann und die an dieser Stelle als Bild im Raum wirklich entsteht. Virtuell bedeutet also „scheinbar, nicht wirklich, unseren Sinnen nur vorgetäuscht“. Der Begriff Realität dagegen beschreibt das, was wir als Menschen um uns herum wahrnehmen und als Wirklichkeit einordnen.

Virtuelle Realitäten begegnen uns somit im Alltag ununterbrochen, denn alles, was wir über unsere Sinne aufnehmen, sind nur Signale einer uns umgebenden Realität, zu der wir selbst natürlich auch gehören. Die Frage, ob die Vorstellungen, die sich aufgrund dieser Signale von außen in unserem Kopf formen, die Wirklichkeit exakt wiedergeben, muss verneint werden, denn es gibt viele zusätzliche Signale, die wir aufgrund der Beschaffenheit unserer Sensoren (Sinne) überhaupt nicht wahrnehmen können; insoweit sind wir nur in der Lage, eine eingeschränkte Wirklichkeitserfahrung zu machen.

Unsere moderne Technik, und hier insbesondere der Computer, ist heutzutage in der Lage, uns mit künstlichen Realitäten zu konfrontieren, die berechnet werden und über geeignete Hilfsmittel auf unsere Sinne gelangen. Schon relativ einfache Ausgabegeräte eines Computers, wie die Bildschirme, sind ja durchaus in der Lage, z.B. in Form von Spielen oder Simulationen, uns mit sehr komplexen virtuellen Realitäten zu konfrontieren. Die technischen Möglichkeiten in diesem Bereich entwickeln sich mit explosionsartiger Geschwindigkeit. Der Computer als Wirklichkeitssimulator wird immer perfekter. Es ist abzusehen, dass es immer besser gelingt, den Computer mit den menschlichen Sinnen zu verbinden: Zum Beispiel können Bildinhalte direkt auf die Retina (Augenhintergrund, Netzhaut) projiziert werden. Auch ist es bereits gelungen, Nerven auf Chips wachsen zu lassen, um somit eine Verbindung, und zwar eine unmittelbare Verbindung, zwischen den Computerbauelementen und dem Nervensystem herzustellen. Zu einer perfekten Wirklichkeitssimulation gehört natürlich auch die ***Interaktivität***, d.h. die Möglichkeit, auf das, was der Computer uns anbietet, zu reagieren. Die Technik hat auch hier bereits viele Möglichkeiten zur Interaktion geschaffen. Neben der Kopplung mit dem Computer über Bilder und Töne ist es heute bereits möglich, den Bereich der taktilen Reize in diese Interaktionen mit einzubeziehen (Datenhandschuh, Datenanzug).

Es ist unschwer zu erkennen, dass am Ende dieser Entwicklung eine mehr oder weniger vollkommene Verbindung zwischen einem Computer als fast perfektem Wirklichkeitssimulator und einem Menschen möglich sein wird.

Spätestens an dieser Stelle drängen sich auch philosophische Fragen in den Vordergrund. Z.B. ist das, was normalerweise in unserem Gehirn abläuft, d.h. unsere Interaktion mit unserer Umwelt über unsere Sinne, nichts anderes als eine Simulation der Wirklichkeit in unserem Gehirn. So gesehen entsteht auch in unserem Hirn nur ein virtuelles Bild der Realität. Viele Philosophen haben sich darüber bereits seit langem Gedanken gemacht, denn diese Fragen tangieren viele Grundfragen unserer menschlichen Existenz

Zwei Beispiele für heutige Wirklichkeitssimulatoren seien kurz skizziert:

- In ***Flugsimulatoren*** wird zum einen sowohl eine realitätsgetreue dreidimensionale bildhafte Umgebung erzeugt als auch die dazugehörigen Raumtöne. Zum anderen befindet sich der Pilot in einem nachgebauten Flugzeugcockpit, das über hydraulische Aktoren jede beliebige Lage im Raum einnehmen kann und auch Beschleunigungseffekte sind damit simulierbar.
- Ein Mensch in einem Datenanzug mit Datenhandschuhen und Datenhelm versehen ist innerhalb einer kardanischen Aufhängung frei im Raum beweglich. Alle Bewegungen des Menschen und seine Orientierung im Raum werden über Sensoren an den Computer gemeldet. Der Datenhelm vermittelte einen räumlichen Bildeindruck und Raumton. Auf

diese Weise kann ein Mensch sich frei in einer Computerwelt bewegen. Über den Datenanzug und die Datenhandschuhe sind auch taktile Reize mit dem Computer austauschbar, d.h. der Mensch spürt, wenn er innerhalb der virtuellen Wirklichkeit Dinge berührt – er spürt also den Gegendruck – und kann selbst, z.B. durch die Ausübung von Druck, Gegenstände bewegen und verändern und somit direkt in das Geschehen innerhalb der virtuellen Welt eingreifen.

Solche Wirklichkeitssimulatoren sind natürlich physikalische Objekte, die denselben physikalischen Gesetzen gehorchen wie alle anderen Körper. Aber der Simulator kann »tun als ob«, also etwas vortäuschen und vorgeben, ein ganz anderes Objekt zu sein, das falschen Gesetzen gehorcht.

Wenn ein Mensch in der virtuellen Welt neue Erfahrungen sammelt, so wird er mit bestimmten Gefühlen und anderen internen Erfahrungen auf diese externen Eindrücke reagieren. Die Frage, ob es je möglich sein wird, auch diese internen Erfahrungen eines Menschen zu simulieren, wollen wir an dieser Stelle ausklammern. Ein solcher „Persönlichkeitssimulator“ mag irgendwann möglich werden, ist jedoch heute (vielleicht Gott sei Dank) noch Utopie.

Eine weitere Eingrenzung ist dadurch gegeben, dass ein Wirklichkeitssimulator nie in der Lage sein kann, logisch unmögliche Erfahrungen zu vermitteln. Zwar kann z.B. ein Simulator durchaus die Erfahrung vermitteln, mit einem Flugzeug durch einen Berg hindurchzufliegen, was physikalisch unmöglich ist. Aber nichts kann einem z.B. die Erfahrung vermitteln, eine Primzahl in Primfaktoren zu zerlegen, denn dies ist logisch unmöglich [2].

Wo werden einmal die ***Grenzen*** liegen für die Erschaffung künstlicher Welten? Gemeint sind hier nicht technologische Grenzen, sondern grundsätzliche, logisch-physikalische Grenzen. Technologische Grenzen werden sich im Laufe der Entwicklung sicherlich überwinden lassen, wie das folgende Beispiel demonstriert: Wie steht es z.B. mit der Möglichkeit, in einem Flugsimulator die Erfahrung der Schwerelosigkeit hervorzurufen? Wenn ein Mensch sich in einem Flugsimulator auf der Erde befindet, wird er in jedem Fall die Schwerkraft spüren, nur mit besonderem technischen Aufwand, z.B. innerhalb eines Flugzeuges auf einer ballistischen Bahn, ist es möglich, für sehr kurze Zeit die Wirkung der Schwerkraft auszuschalten (bzw. zu kompensieren). Auf Dauer oder für längere Zeiten läßt sich in einem Flugsimulator auf der Erde dieses Phänomen also nicht realisieren. In ferner Zukunft wird man sich jedoch nicht damit begnügen, über äußere Einwirkungen auf z.B. die Augen, Ohren oder auf die Haut des Menschen Informationen an den Menschen weiterzugeben, sondern man wird in der Lage sein, direkt und unmittelbar auf die Nerven des Menschen einwirken zu können. In diesem Fall ist es unwichtig, wo der Mensch sich befindet, denn alle Eingaben an das Gehirn, die von den Nerven kommen, werden unmittelbar vom Computer simuliert und somit ist auch das Gefühl der Schwerelosigkeit direkt erzeugbar. Auch die vom Gehirn ausgesandten Impulse, seien sie nun elektrischer oder chemischer Natur, werden dann natürlich an den Computer weitergeleitet und können so in die Simulation mit einbezogen werden.

Die Genauigkeit einer virtuellen Realität kann man natürlich nur durch Vergleich mit der wirklichen Realität herausfinden. Bei der Betrachtung dieser Frage stoßen wir auf ein grundsätzliches Problem, das auch im Zusammenhang steht mit der Frage nach der Genauigkeit oder Wahrheit einer physikalischen Theorie über die Welt [2]. In beiden Fällen ist es möglich, festzustellen, wann z.B. eine Simulation ungenau ist oder eine Theorie nicht stimmt. Wenn man die Realität kennt und sie mit der Simulation vergleicht, wird man feststellen, wenn die Simulation an einer Stelle nicht genau ist und genauso wird man bei einer physikalischen Theorie, wenn sie bestimmte Phänomene auf einmal nicht richtig beschreibt, feststellen, dass sie falsch oder zumindest unzureichend ist. In beiden Fällen wird man aber nie sagen oder bescheinigen können, dass die simulierte Realität genau ist oder daß die Theorie richtig ist. Bei beiden Dinge ist ein Beweis für die absolute Genauigkeit oder die Richtigkeit nicht möglich, denn man kann nie wissen was die Zukunft noch an (vielleicht abweichenden) Erkenntnissen bringt.

Im folgenden Abschnitt wird anhand einer Beschreibungssprache für interaktive dreidimensionale virtuelle Welten im Internet gezeigt, wie man solche Welten „erschaffen“ kann und damit physikalische Vorgänge nachbildet.

2 VRML

2.1 Einführung

Zur Beschreibung virtueller Welten im Internet wurde 1995 die Szenen-Beschreibungssprache VRML (virtual reality modeling language, oft ausgesprochen wie Wörmel) von Mark Pesce [10] entwickelt. 1996 gab es bereits die Version 2 und Ende 1997 die Version VRML97, die auch ISO-Standard werden wird [9].

Mit Hilfe einfacher Anweisungen – die als Text in eine Datei geschrieben werden – lassen sich auf der Basis vordefinierter Körper etc. dreidimensionale Welten erschaffen. Die Dateien müssen die Erweiterung .wrl (world) haben und werden von einem Plug-In innerhalb eines Internetbrowsers ausgeführt. Ein solches Plug-In ist z.B. der Cosmoplayer, der von der amerikanischen Firma SGI (Silicon Grafics Incorporated: http://vrml.sgi.com) entwickelt wurde. Neuere Internetbrowser haben bereits einen VRML-Player integriert.

Das Standardisierungskommitee ist zu erreichen unter: http://www.web3d.org und eine Dokumentation findet man unter: http://www.vrml.org/Specification/VRML97.

Die Idee von VRML ist auch, Räume durch Hyperlinks im Internet untereinander zu verbinden, so daß mehrere Benutzer gleichzeitig in diesen Räumen agieren können. Auf diese Weise kann ein riesiger Raum entstehen (Cyberspace), bei dem man durch „Tunnel“ von einem Ort zu einem anderen springen kann. Weitere Möglichkeiten von VRML sind:

Einbetten anderer Medien, Architekturunabhängigkeit, Erweiterbarkeit, niedrige Bandbreiten (Modem).

Im Folgenden wird kurz beschrieben, wie man interaktive 3D-Welten erzeugen kann. Es wird gezeigt, wie sich physikalische Abläufe darstellen lassen und wie man interaktiv in die Szenen eingreifen kann.

Beispiel 1. Würfel

#VRML V2.0 utf8	obligate Startzeile einer VRML-Datei (utf8 = Zeichensatz)
Shape { appearance Appearance {	definiert das Aussehen eines geometrischen Körpers
material Material { diffuseColor 1 0 0 }	definiert die Materialbeschaffenheit und RGB-Farbe (rot)
}	Ende der Materialdefinition
geometry Box { size 2 2 2 }	Würfel mit der Kantenlänge 2m (x, y, z)
}	Ende der Geometriedefinition

Wenn man die vier Textzeilen aus Beispiel 1 in einer Datei des Namens Wuerfel.wrl speichert (z.B. mit Hilfe des Texteditors Notepad, der reinen ASCII-Text erzeugt) und auf diesem Dateinamen dann einen Doppelklick ausübt, wird der installierte Internetbrowser gestartet, das VRML-Plug-In geladen und die Datei ausgeführt. Als Ergebnis ist ein roter Würfel zu sehen, den man mit Hilfe der Steuerelemente des VRML-Players beliebig im Raum bewegen und verdrehen kann.

Beispiel 1 zeigt, wie mit VRML als ***Szenenbeschreibungssprache*** SDL (scene description language) auf sehr einfache Weise eine räumliche Szene kreiert werden kann. Es wird ein objektorientierter Ansatz benutzt, um die Konstellation von 3D-Objekten in einer Szene zu beschreiben.

2.2 Aufbau einer 3D-Szene

Ein VRML-Dokument besteht aus einer Liste von Objekten, die als Knoten (nodes) bezeichnet werden. Jeder Knoten hat einen Namen, gefolgt von weiteren Elementen die in geschweiften Klammern stehen ({ }, curly braces, siehe Beispiel 2).

- Gruppenknoten (grouping nodes)
- Kindknoten (children nodes)
- Objektknoten

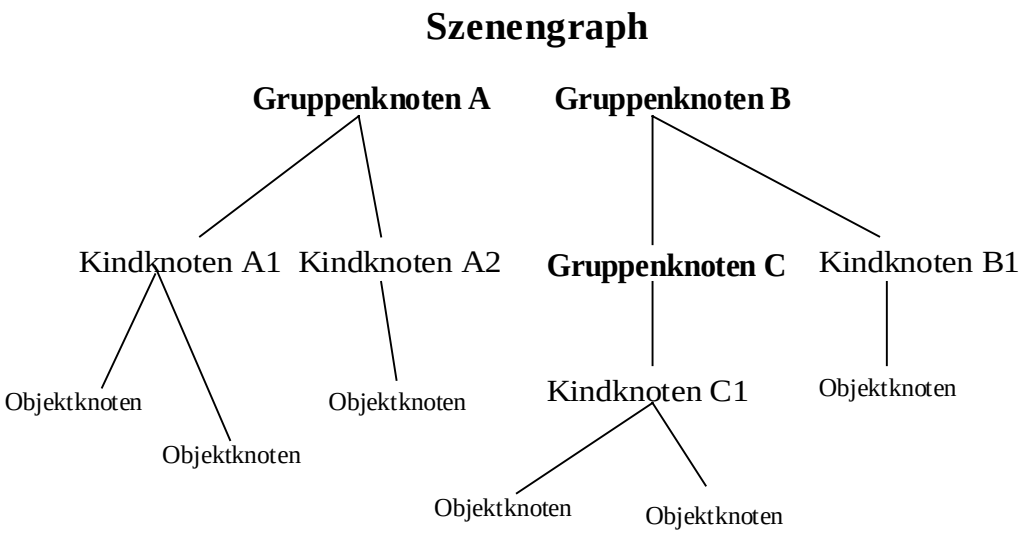

Abb.1. Szenengraph zur Beschreibung einer 3D-Welt, bestehend aus Knoten und Kanten (Feldern)

Kindknoten beginnen mit *children* und in eckigen Klammern eingeschlossen sind die weiteren Szenenelemente.

Durch die Kombination verschiedener Knoten in einer Datei entsteht eine hierarchische Struktur, die als sog. Szenengraph (scene graph) bezeichnet wird. Eine VRML-Datei enthält somit genau einen Szenengraphen.

In VRML gibt es die folgenden ***Gruppenknoten***:

- *Anchor:* Hyperlink-Knoten (Einbindung einer URL=uniform recource locator)
- *Billboard:* Permanente Ausrichtung der Objekte auf die Betrachterposition
- *Collision:* Berührungserkennung zur Modellierung realer physikalischer Effekte
- *Group:* Enthält Kindknoten die keiner Transformation unterliegen (s.a. *Transform*)
- *Inline:* spezifiziert Hyperlinks zu Kindknoten (Einbindung einer URL=uniform recource locator)
- *LOD:* (level of details) spezifiziert unterschiedliche Detailtreue bei einem Objekt
- *Swich:* Umschaltung auf die Abarbeitung eines anderen Knotens
- *Transform:* Festlegung eines Koordinatensystems für die „Kinder“ relativ zu den „Vorfahren“
- *PROTO:* Definition eigener Gruppenknoten

Diese stehen in der Hierarchie ganz oben, d.h. sie werden zuerst definiert. Darunter gibt es die ***Kindknoten***, z.B.:

- *Background:* Festlegung der Hintergrundeigenschaften, wie z.B. der Farbe
- *PositionInterpolator:* Kontinuierliche Bewegung eines Körpers zwischen zwei Raumkoordinaten (Bsp. 4)
- *Shape:* Erscheinungsbild geometrischer Körper (Box, Cone, Sphere, Color, Material, Texture, etc.)
- *TimeSensor:* Uhr in der virtuellen Welt die Zeitzeichen an andere Knoten senden kann (Bsp. 4)
- *TouchSensor:* Feststellung ob mit dem Cursor ein Objekt berührt wird
- *Transform:* Festlegung eines Koordinatensystems für die „Kinder“ relativ zum Gruppenknoten
- *Viewpoint:* Festlegung eines Start-Blickpunktes auf die Szene

Zusätzlich existieren weitere Knoten die ausschließlich der Erzeugung und Gestaltung geometrischer Objekte dienen.

Beispiele für ***Objektknoten***:

- Appearance
- Sphere { radius 0.5 }
- Cone { bottomRadius 3 height 5 }
- Box { size 0.5 2 3 }
- Text { string “Beispieltext“ fontstyle FontStyle { size 2 family “SANS“ style “BOLD“ } }
- IndexedLineSet
- IndexedFaceSet
- PointSet

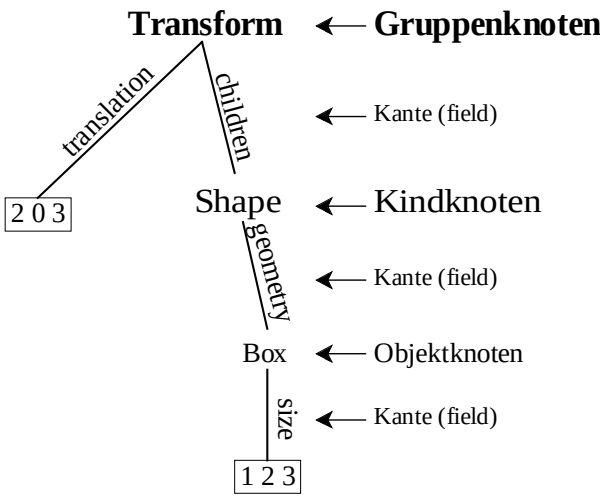

Abb.2. Szenengraph zum verschobenen Quader aus Beispiel 2

Beispiel 2. Verschobener Quader

#VRML V2.0 utf8	Hinter dem Zeichen # können Texte eingefügt werden
Transform{ # = Kommentar-Zeichen	Gruppenknoten zur Festlegung eines Koordinatensystems
translation 2 0 3	Verschiebung auf x = 2m, y = 0m, z = 3m
children [	Anfang des Kindknotens
Shape { geometry Box { size 1 2 3 } }	Definition eines Quarders mit x = 1m, y = 2m, z = 3m
]	Ende des Kindknotens
}	Ende des Gruppenknotens

Die Knoten werden über Kanten (bei VRML heißen diese Kanten „Felder“ (fields)) miteinander verbunden und so entsteht dann ein Szenengraph. Eine Veränderung im Gruppenknoten pflanzt sich über den Kindknoten bis zum Objektknoten fort, man nennt dies ***Vererbungsprinzip***.

Das Beispiel 2 zeigt dieses Prinzip und in der Abb.2 ist der zugehörige Szenengraph dargestellt. Wenn dem Quader aus Beispiel 2 noch eine Farbe zugewiesen werden soll, so zeigt Abb.3 wie sich das im Szenengraphen darstellen lässt.

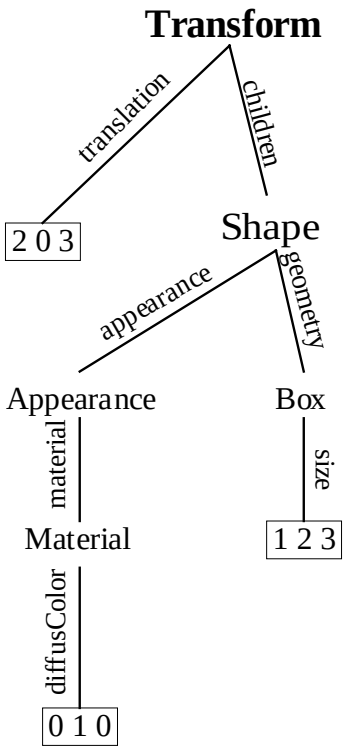

Abb.3. Szenengraph eines verschobenen grünen Quaders

2.3 Maßeinheiten und Koordinatensystem

Lineare Distanzen .... Meter (m)

Winkel ..... Bogenmaß (rad)

Zeit ..... Sekunden (s)

Farbraum ..... RGB([0.,1.], [0.,1.], [0. 1.])

Farbangaben erfolgen immer in der Reihenfolge rot, grün, blau (RGB)

Koordinatenangaben erfolgen immer in der Reihenfolge x, y, z. In der Abbildung 4 ist die räumliche Orientierung der drei Achsen dargestellt: es handelt sich um ein rechtshändiges Koordinatensystem, d.h. wenn man z.B. die x-Achsen zur y-Achse hin dreht, hat die z-Achse die Bewegungsrichtung einer Rechtsschraube. Dieses Prinzip gilt auch für jedes andere Achsenpaar unter Berücksichtigung der Reihenfolge im Alphabet.

Ein ***geometrische Körper*** hat seinen Mittelpunkt im Koordinatenursprung, d.h. z.B, dass ein Würfel von 2m Kantenlänge jeweils 1m in positiver Richtung und 1m in negativer Richtung der drei Achsen reicht (Abb.4).

Drehungen erfolgen im mathematisch positiven Sinn, d.h. entgegen der Uhrzeigerdrehung.

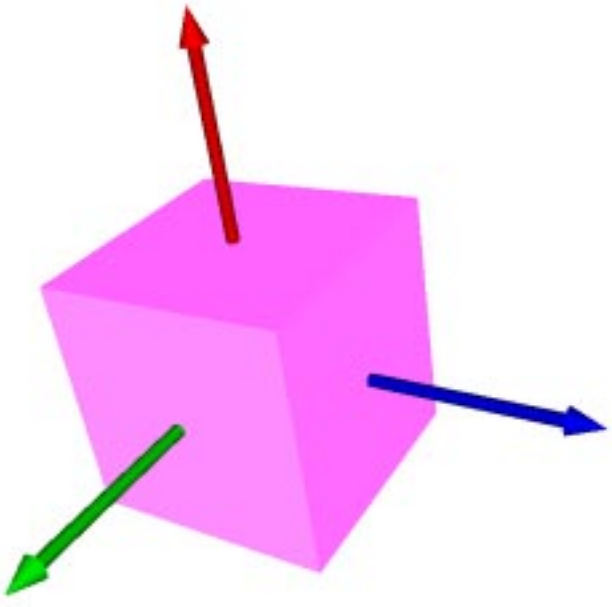

Abb.4. Würfel im Koordinatensystem

2.4 Instanziierung mit DEF und USE

Wenn innerhalb einer virtuellen Welt ein Objekt – z.B. ein Stuhl in einem Raum oder eine Materialeigenschaft – häufiger vorkommt, so kann man auf dieses Objekt beliebig oft ***referenzieren***, dies nennt man ***Instanziierung***.

Zu diesem Zweck muß man ein Objekt vorher über die Anweisung DEF bezeichnen und kann es dann über die Anweisung USE referenzieren.

Das Beispiel 3 zeigt, wie man die Eigenschaft »violett strahlend« { emissiveColor 0.8 0 0.8 }auf das Objekt Kugel anwenden kann und dann auf diese so instanziierte Kugel dreimal referenziert, um insgesamt vier violett strahlende Kugeln im Raum anzuordnen (Abb.5).

Beispiel 3. 4 Kugeln

```
#VRML V2.0 utf8

Background { skyColor 0.2 0.3 0.7 }

DEF VIO Appearance   {                    # Farbe definieren
          material Material { emissiveColor 0.8 0 0.8 }
                       }                   # Ende Farbdef.
DEF KUGEL Shape{   appearance USE VIO
                      geometry Sphere { radius 1 }
                   }          # Ende der Kugeldefinition
Transform{   translation 1.5 0 0          # Kugel anordnen
                children USE KUGEL   }

Transform{   translation 0 1.5 0          # Kugel anordnen
                children USE KUGEL   }

Transform{   translation 0 0 1.5          # Kugel anordnen
                children USE KUGEL   }
```

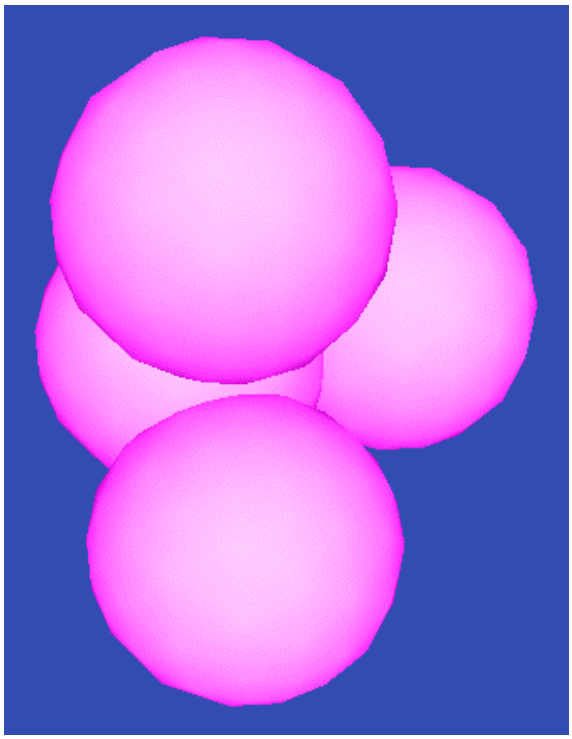

Abb.5. Vier Kugeln im Raum

3 VRML dynamisch

3.1 Interaktion und Animation

Neben den bisher beschriebenen statischen Welten kann man mittels VRML auch dynamische 3D-Welten erschaffen. Hierzu können den Objekten zahlreiche zusätzliche Funktionen und Mechanismen zugeschrieben werden, die es gestatten, dynamische Effekte, neben den bisher bekannten Eigenschaften wie z.B. Geometrie und Farbe, zuzuweisen. Um dies zu erreichen, müssen die Objekte bzw. ihre Knoten über entsprechende Ein-/Ausgabeschnittstellen verfügen.

Interaktive Animationen lassen sich in VRML steuern über: Events, Routes, Sensoren, Interpolatoren.

3.2 Ein-/Ausgabeschnittstellen für Ereignisse (events)

Ankommende Ereignisse sind Datenpakete oder Meldungen, die über ein *eventIn*-Feld (Kante) von einem Knoten empfangen werden und dort meistens Zustandsänderungen auslösen. Über die Kante *eventOut* kann der Knoten wiederum Meldungen an andere Knoten senden.

Die Verknüpfung zwischen einem Knoten, der eine Meldung sendet, mit dem, der diese Meldung empfängt, erfolgt über den ***ROUTE***-Befehl. Mit ROUTE sind keine Knoten gemeint, es ist lediglich ein Kontrollmechanismus zur Kommunikation von Meldungen oder Ereignissen zwischen Knoten über definierte Schnittstellen. Sie dienen somit der Steuerung des Datenflusses zwischen VRML-Knoten.

Im Folgenden werden kurz und beispielhaft die beiden Knoten *PositionInterpolator* und *TimeSensor* erklärt, da sie im Beispiel 4 Anwendung finden. Ausführliche Definitionen enthält das Internet unter http://www.vrml.org/Specification/VRML97 oder die HomePage des Autors http://www.DieterHannemann.de.

3.3 Sensoren

Es gibt eine ganze Reihe von Sensoren, über die Ereignisse generiert werden können, wie z.B. der TimeSensor oder der TouchSensor. Da wir im Beispiel 4 einen Zeitgeber brauchen, sollen hier kurz einige Merkmale des TimeSensor beschrieben werden.

Der ***TimeSensor*** erzeugt ein Ereignis aufgrund der ablaufenden Zeit. Der Startpunkt dieser virtuellen Uhr wurde festgelegt auf den 1. Januar 1970 0:0:0 Uhr GMT (Greenwich Mean Time). Im Beispiel 4 ist zu sehen, dass über *DEF CLOCK* ein Zeitgeber definiert wird, der fortlaufend Ereignisse generiert (*loop* TRUE) und zwar in einem Intervall (*cycleInterval*) von 2,4 Sekunden. Diesem Ereignis ist das *eventOut*-Feld *fraction_changed* zugeordnet.

Mit dem Befehl *ROUTE CLOCK.fraction_changed TO POSITION.set_fraction* werden diese Ereignisse dem *eventIn*-Feld des Positionsinterpolators zugeführt:

Ausschnitt aus Beispiel 4:

```
DEF CLOCK TimeSensor  { loop TRUE cycleInterval 2.4 }        # Definition der „Uhr“. Zykluszeit 2,4s
ROUTE CLOCK.fraction_changed  TO  POSITION.set_fraction      # Zeittakte an den Positionsinterpolator senden
```

3.4 Interpolatoren

Neben anderen Möglichkeiten zur Interpolation soll uns hier nur der ***PositionInterpolator*** interessieren. Im Beispiel 4 wird über *DEF POSITION* ein solcher Interpolator definiert. Er dient zur linearen Interpolation von Positionen, um z.B. ein Objekt entlang eines vorgegebenen Pfades gleiten zu lassen. Die entsprechenden Positionen werden über die Felder (Kanten) *key* und *keyValue* zugeordnet.

Ausschnitt aus Beispiel 4:

```
DEF POSITION PositionInterpolator {                          # Definition des Positionsinterpolators
    key         [   0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1 ]   # Zeitintervalle festlegen: 2,4s * key
    keyValue [   0 0 0,   0 -0.3 0,  0 -1.1 0,  0 -2.5 0,  0 -4.5 0,      # Position für die „key-Zeiten“ festlegen
                 0 -7 0,  0 -4.5 0,  0 -2.5 0,  0 -1.1 0,  0 -0.3 0,  0 0 0  ]   #   y = 0m bis -7m und wieder bis 0m
                                  }
ROUTE POSITION.value_changed  TO  BALL.set_translation       # Neue Positionen an Ballknoten senden
```

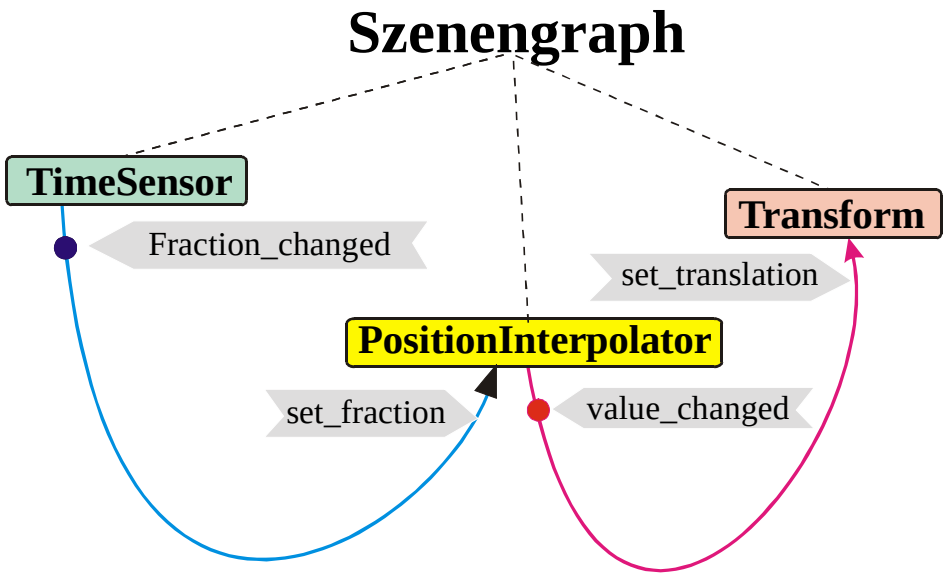

Abb.6. Zeitgesteuerte Position

3.5 Physikalisches Beispiel

Im Folgenden kommen wir nun zu einer dynamischen physikalischen Anwendung unter Ausnutzung der gerade gemachten Erklärungen. Das Beispiel 4 zeigt eine Animation, bei der ein Ball fällt, verlustfrei reflektiert wird und wieder fällt. Den Szenengraphen dazu enthält Abbildung 6.

Der Ball fällt aus 7m Höhe auf eine Fläche die darunter liegt. Da der Ball einen Durchmesser von 1m hat und sich die Höhenangabe auf den Ballmittelpunkt bezieht, muß die Fläche 7,5m unter dem Ball liegen. Die Fallzeit beträgt $t = (2s/g)^{1/2}$ und mit $s = 7m$ erhält man ca. $t = 1,2s$. Da der Ball wieder auf die selbe Höhe springen soll erhält man somit eine Zykluszeit von ca. 2,4s. Für die Zwischenwerte – bei gleichen Zeitintervallen von 0,24s – kann man dann die jeweiligen Positionen berechnen und in die Tabelle in Beispiel 4 eintragen (keyValue).

Beispiel 4. Hüpfball

```
#VRML V2.0 utf8
Background { skyColor 0.2 0.3 0.7 }
Transform{ translation 0 4 0                                    # Der Ball startet aus 4m Höhe
      children                                                  # 1. Kindknoten
      DEF BALL Transform   { children [                         # Definition des Balls
      Shape {    appearance Appearance { material Material { diffuseColor 1 0 0.2 } }
                 geometry Sphere { radius 1 }                   # Radius des Balls = 1m
            }                          ]                       # Ende des 2. Kindknotens
            }                  }                               # Ende des Gruppenknotens
Transform{    translation 0 -3.5 0          # Die Aufprallfläche liegt 7.5m unter dem Mittelpunkt des Balls
              rotation 1 0 0  0.4                 # Ankippen der Fläche nach vorne (0.4 rad um die x-Achse)
      children
      Shape {    appearance Appearance { material Material { diffuseColor 0.5 1 0.5 } }
                 geometry Box { size 10 0.001 10 }          # Die „Fläche“ besteht aus einer Box von 1mm Dicke
            }}
DEF CLOCK TimeSensor   { loop TRUE cycleInterval 2.4 }          # Definition der „Uhr“. Zykluszeit 2,4s
DEF POSITION PositionInterpolator   {                           # Definition des Positionsinterpolators
   key         [    0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1 ]   # Zeitintervalle festlegen: 2,4s * key
   keyValue [    0 0 0,   0 -0.3 0,  0 -1.1 0,  0 -2.5 0,  0 -4.5 0,     # Position für die „key-Zeiten“ festlegen
                 0 -7 0,  0 -4.5 0,  0 -2.5 0,  0 -1.1 0,  0 -0.3 0,  0 0 0  ]   #   y = 0m bis -7m und wieder bis 0m
                                    }
ROUTE CLOCK.fraction_changed  TO  POSITION.set_fraction      # Zeittakte an den Positionsinterpolator senden
ROUTE POSITION.value_changed  TO  BALL.set_translation     # Neue Positionen an Ballknoten senden
```

4 Ausblick

Die hier vorgestellten Techniken werden u.a. eingesetzt innerhalb des Bundesleitprojektes „Virtuelle Fachhochschule“ (gefördert durch das BMBF mit 43 MioDM: www.vfh.de). Innerhalb dieses Projektes, an dem sich 14 Hochschulen (Fachhochschulen und Universitäten) beteiligen, sollen zwei Studiengänge entwickelt werden die im wesentlichen über das Internet studierbar sind (Wirtschaftsingenieur und Medieninformatik mit den Abschlüssen Bachelor und Master). Der Autor ist Vizegesamtprojektleiter und Entwickler für das Fach Infophysik. Eine wesentliche Aufgabe besteht in der Entwicklung multimedialer Lehr- und Lernformen.

Ausführlichere Informationen und Ergänzungen zu diesem Artikel findet man im Internet unter www.DieterHannemann.de.

5 Literatur

[1] F..Dai: Lebendige virtuelle Welten. Berlin: Springer 1997.

[2] D. *Deutsch:* Die Physik der Welterkenntnis. Berlin: Birkhäuser 1996

[3] F. *Eckgold*, F., 1996: Virtual Reality. Braunschweig: Vieweg 1995

[4] R. *Däßler* – H. *Palm*: Virtuelle Informationsräume mit VRML. Heidelberg: dpunkt 1998

[5] D. *Hannemann*: Beispiele und Ergänzungen, siehe: Internet-Homepage: http://www.DieterHannemann.de

[6] D. *Hannemann*: Physik für Studierende der Technik und Informatik. Gelsenkirchen: Hannemann 1998

[7] D. *Hannemann*: Mikroinformatik. Bd. 1, 2. Auflage. Gelsenkirchen: Hannemann 1993. Bd. 2, 1. Auflage. Gelsenkirchen: Hannemann 1995

[8] H-L. *Hase*: Dynamische virtuelle Welten mit VRML 2.0. Heidelberg: dpunkt 1997

[9] J. *Kloss* – R. Rockwell – K. Szabó – M. Duchrow: VRML97. Bonn: Addison-Wesley 1998

[10] M. Pesce: VRML, Cyberspace-Welten erkunden und erschaffen. München: Carl Hanser 1997