VRML-Kurs, Teil 5

Kollisionserkennung

Generell sind alle Objekte außer IndexedLineSet-, PointSet- und Text-Knoten mit der Fähigkeit des Erkennens von *Kollisionen* mit dem virtuellen Betrachter versehen.
Beispiele: Simulation des Anstoßens an eine Wand, Berührung von Objekten, "realistisches" Begehen von Labyrinthen etc.

Beachte: "Kollision" bezieht sich hier immer auf die Interaktion des Nutzers (bzw. seiner virtuellen Repräsentation im Cyberspace) mit einer (Objekt-) Geometrie, nicht auf Objekt-Objekt-Kollisionen!

Die Kollisionserkennung ist als Default *ein*gestellt.
Das Verhalten bei einer erfolgten Kollision ist vom Browser abhängig.
Die Reichweite der Prüfung auf Kollisionen zwischen Nutzer und virtuellen Objekten wird in einem **`NavigationInfo`**-Knoten im ersten Wert des Feldes **`avatarSize`** festgelegt.

Objektspezifische Einstellungen des Kollisionsverhaltens erfolgen mittels des **`Collision`**-Knotens.

Aufgaben dieses Knotens:

- Ein- und Ausschalten der Kollisionserkennung für seine Kindknoten
- Spezifikation eines Ersatzobjekts (**`proxy`**) mit einfacherer Geometrie oder einer bounding box zur Beschleunigung der Kollisionserkennung
- Aussenden von Ereignissen an andere Knoten bei Kollision, um Effekte zu generieren
- Setzen "unsichtbarer Wände" (wenn keine **`children`**, sondern nur ein **`proxy`**-Knoten definiert ist)

Felder des **Collision**-Knotens:

children	**MFNode**	Kindknoten
collide	**SFBool**	Ein-Aus-Schalter
proxy	**SFNode**	Ersatzobjekt
bboxCenter	**SFVec3f**	) boundig box-Spezifikation
bboxSize	**SFVec3f**	)
addChildren	**MFNode**	**eventIn**
removeChildren	**MFNode**	**eventIn**
collideTime	**SFTime**	**eventOut** (Zeitmarke der Kollision)

Ist der **Collision**-Knoten der Wurzel-Knoten einer VRML-Szene und steht **collide** auf FALSE, so ist die Kollisions-erkennung für die gesamte Szene deaktiviert, unabhängig davon, ob darunterliegende Knoten **collide** auf TRUE gesetzt haben oder nicht.
Die bounding box wird deaktiviert durch den (Default-) Wert **–1 –1 –1** für **bboxSize**.
Ist **collide** auf TRUE gesetzt und **proxy** nicht definiert (= **NULL**), so werden die **children**-Knoten zur Kollisions-erkennung verwendet, sonst der **proxy**-Knoten, welcher aber nicht visuell dargestellt wird.

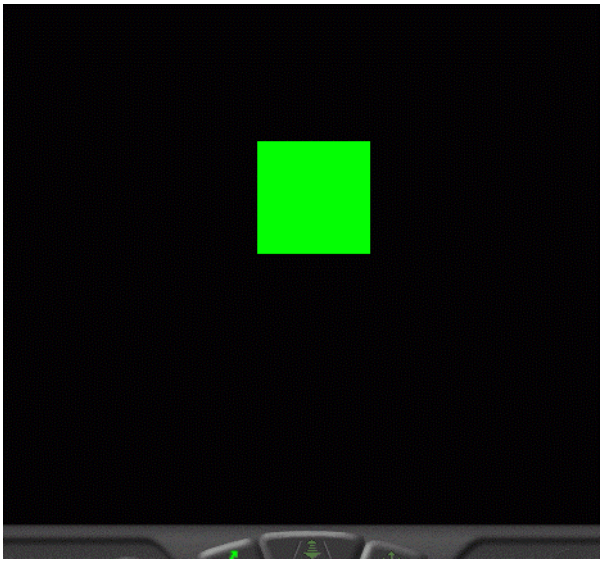
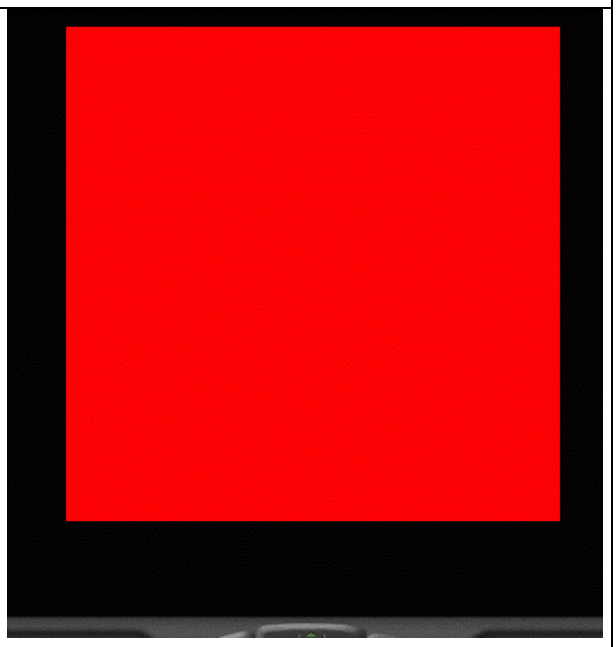
Beispiel:
Ein Würfel wird von einem größeren **proxy**-Würfel umgeben. Bei Kollision mit dem unsichtbaren, äußeren Würfel ändert der innere Würfel seine Farbe von Grün nach Rot:

```
#VRML V2.0 utf8

Transform
   {
   translation 0 0 -5
   children
      [
```

```
      DEF Tabu_Zone Collision
         {
         collide TRUE
         children
            [
            Shape
               {
               geometry Box { }
               appearance Appearance
                  {
                  material DEF BoxFarbe
                     Material
                     { diffuseColor 0 1 0 }
                  }
               }
            ]
         proxy Shape
            {
            geometry Box { size 8 8 8 }
            }
         }
      ]
   }
DEF Timer TimeSensor { }
DEF FarbCalc ColorInterpolator
   {
   key [ 0 1 ]
   keyValue [ 1 0 0, 1 0 0 ]
   }
ROUTE Tabu_Zone.collideTime_changed TO
   Timer.startTime
ROUTE Timer.fraction_changed TO
   FarbCalc.set_fraction
ROUTE FarbCalc.value_changed TO
   BoxFarbe.set_diffuseColor
```

Ergebnis (abhängig von Bewegungen des Benutzers):

Benutzer hält Abstand ein	*Benutzer ist mit dem unsichtbaren Außen-Würfel kollidiert*
	

Eine größere Annäherung als auf dem rechten Bild ist nicht mehr möglich.

Switch-Knoten

Funktion: Alternative Objekt-Auswahl je nach Zahlenwert eines Diskriminator-Feldes **`whichChoice`** (vgl. switch-Konstrukt in Java und C)

Felder des **`Switch`**-Knotens:

`choice` **`whichChoice`**	**`MFNode`** **`SFInt32`**	Liste der Alternativ-Knoten Diskriminator: Index des darzustellenden Kind-Knotens

Die implizite Indizierung der Kind-Knoten ist 0; 1; 2; ... in der Reihenfolge ihrer Auflistung. Ist der Wert von **`whichChoice`** kleiner als 0 oder größer als die Anzahl der vorhandenen Kindknoten – 1, so erfolgt gar keine Wiedergabe.
(Beispiel nach Einführung des **`Script`**-Knotens.)

Einbau von Scripten

Der **`Script`**-Knoten ermöglicht die Kommunikation mit prozeduralen und objektorientierten Programmen und damit die Verfügbarmachung einer intelligenteren Programmierlogik in VRML-Szenen, als es mit den Mitteln von VRML allein möglich wäre.

Anwendungen:
- mathematische Operationen (Arithmetik, vektorielle Berechnungen)
- Hilfsoperationen (z.B. Typkonvertierungen, Zerlegen zusammengesetzter Werte in ihre Komponenten u. umgekehrt)
- Zwischenspeichern von Werten, die bestimmte Zustände beschreiben
- Flexibilisierung der Ereignisverarbeitung
- Simulationen
- Multi-User-Anwendungen

unterstützte Programmiersprachen:
- Javascript (ECMAScript) und VRMLScript (inline oder in externen Dateien)
- Java (nur externe Bytecode-Dateien, Endung **`.class`**)

`Script`-Knoten werden ähnlich wie Prototypen definiert: In ihrer Schnittstellen-Beschreibung können beliebig viele Felder und Ereignisse definiert werden, wobei die Felder zur Speicherung von Werten und die Ereignisse der Kommunikation mit Knoten der Szene dienen.
Die Zugriffsart **`exposedField`** wird in **`Script`**-Knoten nicht unterstützt.
Zu und von den Ereignissen der **`Script`**-Knoten können die üblichen Routen vereinbart werden. Darüberhinaus verfügen **`Script`**-Knoten über eine direkte Interaktionsmöglichkeit mit anderen Knoten, sofern diese direkt in einem seiner Felder definiert sind (Feldtyp **`SFNode`** oder **`MFNode`**) oder wenn aus

einem solchen Feld eine Referenz mit **USE** auf einen benannten Knoten (anderswo def.) hergestellt wurde. Für die direkte Interaktion muss das Feld **directOutput** auf **TRUE** gesetzt sein.

Felder des **Script**-Knotens:

url	**MFString**	eigentliches Skript (inline oder als url)
directOutput	**SFBool**	ermöglicht direkte Interaktion mit Knoten
mustEvaluate	**SFBool**	steuert das Laufzeitverhalten
beliebige Anzahl von:		
Feldname	Feldtyp (beliebig)	+ Angabe von Defaultwert
Ereignisname	**eventIn**	
Ereignisname	**eventOut**	

Beispiele verschiedener Typen von Codereferenzen im **url**-Feld:

```
Script
{
url [
   "javascript: .... "  # JavaScript-Protokoll, inline
   "file://test.js"     # JavaScript-Protokoll aus Datei
   "file://test.class" ]  # Java Bytecode aus Datei
....
}
```

Das Scripting ermöglicht sehr vielfältige Interaktionsformen; hier nur ein sehr einfaches Beispiel:

Eine Kugel soll sich auf einer Ellipse bewegen. Im Script-Knoten werden die Koordinaten berechnet unter Rückgriff auf trigonometrische Funktionen in JavaScript:

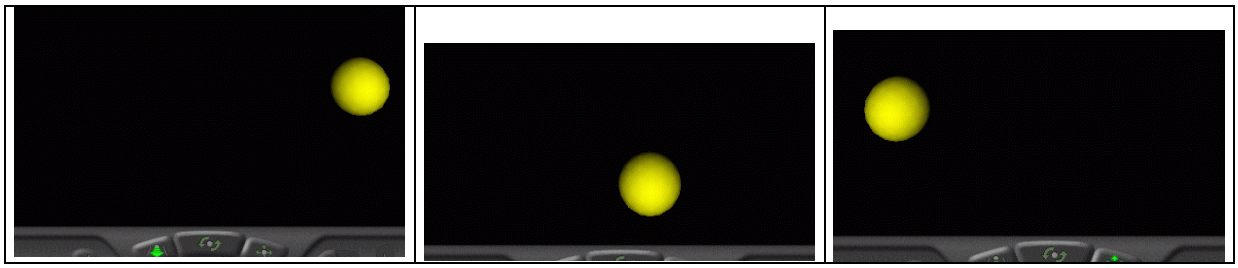
```
#VRML V2.0 utf8

DEF Timer TimeSensor
   {
   cycleInterval 5
   loop TRUE
   }
DEF Verschieb Transform
   {
   children
      [
      Shape
         {
         geometry Sphere { radius 0.2 }
         appearance Appearance
            {
            material Material
               { diffuseColor 1 1 0 }
            }
         }
      ]
   }
DEF Berechnung Script
   {
   eventIn SFFloat set_fraction
   eventOut SFVec3f value_changed
   url "javascript:
      // Funktionsberechnungen elliptische Bahn
      function set_fraction (wert, zeit)
         {
         value_changed[0] =
            Math.sin(wert * 6.283);
         value_changed[1] =
            0.5 * Math.cos(wert * 6.283);
         value_changed[2] = 5;
         }
      "
   }
```

```
ROUTE Timer.fraction_changed TO
   Berechnung.set_fraction
ROUTE Berechnung.value_changed
   TO Verschieb.set_translation
```

Der Name der JavaScript-Funktion entspricht demjenigen des zu verarbeitenden **`eventIn`**-Feldes (hier: **`set_fraction`**). An die Funktion wird immer ein Wert (der Wert des ankommenden Ereignisses) und eine Zeitmarke übergeben (die nicht notwendig verwendet werden müssen).

Ergebnis des Beispiels (drei snapshots):



Beispiel für die Anwendung des Script-Knotens zur Typumwandlung (hier: von SFFloat in SFInt32), zugleich Beispiel für einen Switch-Knoten:
Ein Objekt verwandelt sich fortlaufend von einem Würfel in eine Kugel und umgekehrt.

```
#VRML V2.0 utf8

DEF Auswahl Switch
   {
   choice
      [
      Shape
         { geometry Box { } }
      Shape
         { geometry Sphere { } }
      ]
   }
```

```
DEF Timer TimeSensor
   {
   cycleInterval 5 # Sekunden
   loop TRUE
   }
DEF Berechnung Script
   {
   eventIn SFFloat set_fraction
   eventOut SFInt32 value_changed
   url "javascript:
      // Typkonvertierung
      function set_fraction (wert, zeit)
         {
         value_changed = wert+0.5;
         }
      "
   }
ROUTE Timer.fraction_changed TO
   Berechnung.set_fraction
ROUTE Berechnung.value_changed TO
   Auswahl.set_whichChoice
```

Ergebnis (2 snapshots):