VRML-Kurs, Teil 4

Interaktion zwischen Knoten

Mehrere VRML-Knotentypen können Ereignisse (*events*) empfangen und/oder senden.
Ereignisse enthalten einen Wert und eine Zeitmarke (*timestamp*).

- Gesendete Ereignisse können die Änderung von Werten von Feldern mitteilen
- Empfangene Ereignisse können Werte von Feldern ändern

Ereignisse sind ihrerseits als Felder von Knoten definiert.

Ereignis, das die Änderung eines Feldes mitteilt:
Schlüsselwort **`eventOut`**
Ereignisname meist = Feldname und Suffix **"`_changed`"**
(Ausnahmen: Ereignisse mit den Typen **`SFBool`** und **`SFTime`**.)

Ereignis, das ein Feld ändern kann:
Schlüsselwort **`eventIn`**
Ereignisname meist = Feldname und Präfix **"`set_`"**
(Ausnahmen: **`addChildren`**, **`removeChildren`**, Ereignisse vom Typ **`SFTime`**).

Bei Feldern, zu denen beide Typen von Ereignissen gehören, kann statt der Definition des Feldes und beider Ereignisse eine Kurzform verwendet werden: Voranstellen des Schlüsselwortes **"`exposedField`"** vor den Feldnamen. Beispiel:
Bei einem Feld **"`position`"** innerhalb einer Knotendefinition kann statt der drei Angaben

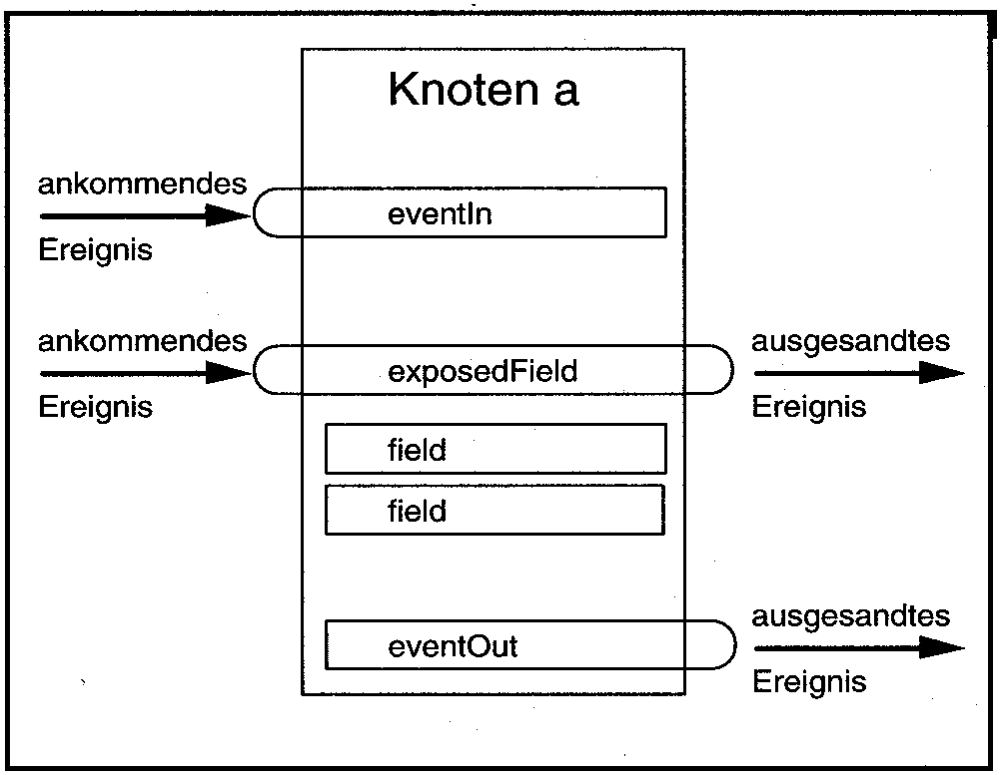
```
field position
eventIn set_position
eventOut position_changed
```

abgekürzt geschrieben werden:

```
exposedField position
```

Die Schlüsselwörter **`field`**, **`exposedField`**, **`eventIn`**, **`eventOut`** geben die *Zugriffsart* eines Feldes oder Ereignisses an.

Schnittstellen-Schema eines Knotens:



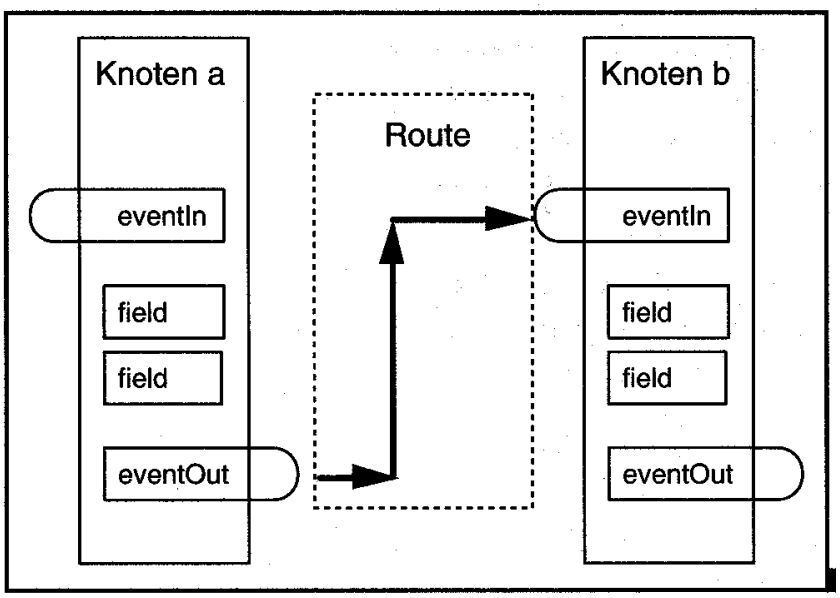
(Bei der Def. neuer Felder im Rahmen von Prototypen sind die Namenskonventionen für Präfix bzw. Suffix nicht zwingend vorgeschrieben, werden aber zur Verbesserung der Konsistenz und Lesbarkeit empfohlen.)

Ereignistyp: Typ des geänderten bzw. des zu ändernden Feldes.

Ereignisse werden durch *Routen* zwischen zwei Knoten übertragen.
Dabei können nur Knoten verwendet werden, die mit dem Schlüsselwort **DEF** einen Namen zugeordnet bekommen haben (siehe Kursteil 3a).
Voraussetzung: Die Ereignistypen des Output- und des Input-Ereignisfeldes müssen übereinstimmen.

Syntax der Routen-Vereinbarung:

ROUTE *NameKnotenA.feld1*_**changed TO** *NameKnotenB*.**set**_*feld2*



Ausnahmen von der Namenskonvention "**_changed**" für Output- und "**set_**" für Inputereignisse:

Ereignisse mit booleschen Werten: **is**... (z.B. **isActive**)
Ereignisse mit Zeitwerten: ...**Time** (z.B. **touchTime**)
Ereignisse zum Hinzufügen und Entfernen von Kindknoten:
add_children, **remove_children**

In der Routendefinition können "**set_**" und "**_changed**" bei Ereignissen von Feldern mit der Zugriffsart **exposedField** fortgelassen werden.
Beispiel:
bei

```
DEF Schalter TouchSensor { enabled TRUE }
DEF Licht DirectionalLight { on FALSE }
ROUTE Schalter.enabled_changed TO Licht.set_on
```

kann die letzte Zeile vereinfacht werden zu:

```
ROUTE Schalter.enabled TO Licht.on
```

Ein per Route weitergeleitetes Ereignis kann bei der Verarbeitung durch den empfangenden Knoten ein weiteres Ereignis auslösen, das wiederum weitergeleitet wird, usw.: *Ereignis-Kaskade*.

Alle Ereignisse einer Kaskade haben dieselbe Zeitmarke.
Bei der Konstruktion von Ereignis-Kaskaden dürfen keine Schleifen gebildet werden!

Zwei oder mehr Ereignisse gehen vom selben **`eventOut`**-Ereignis an verschiedene Knoten: "*fan-out*", erlaubt.

Zwei oder mehr Ereignisse mit derselben Zeitmarke werden an dasselbe **`eventIn`**-Ereignis geleitet: "*fan-in*", Ergebnis undefiniert, deshalb verboten.

Animation
(seit VRML 2.0)

Antrieb der Veränderung durch spezielle Knoten, z.B. **`TimeSensor`**, die Zeitwerte an Interpolator-Knoten senden. Diese erzeugen die gebrauchten (Zwischen-) Werte für Felder animierter Objekte.

`TimeSensor`	**`enabled`**	**`SFBool`**	Start/Stop
	`cycleInterval`	**`SFTime`**	Zykluszeit
	`loop`	**`SFBool`**	unendl. Wiederholung
	`startTime`	**`SFTime`**	Start des Timer-Ablaufs
	`stopTime`	**`SFTime`**	Lebensdauer des Timers
	`fraction_changed`	**`SFFloat`**	**`eventOut`**-Feld, Anteil des bereits verstrichenen Zeitintervalls (zw. 0 und 1)

	isActive **time** **cycleTime**	**SFBool** **SFTime** **SFTime**	**eventOut** **eventOut** **eventOut**, Start eines Zyklus

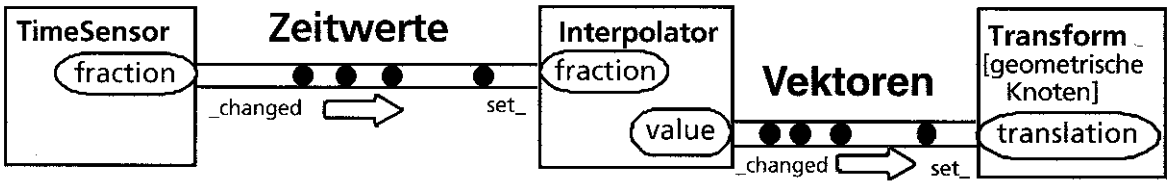
Die Interpolator-Knoten stellen anwendungsspezifische Interpolationswerte bereit:

Orien-tationInter-polator	**set_fraction** **key** **keyValue** **value_changed**	**SFFloat** **MFFloat** **MFRotation** **SFRotation**	**eventIn** Liste von Zwischen-werten (mindest. 0 und 1) Liste v. Zwischen-werten für die inter-polierte Größe **eventOut**
PositionInter-polator	**set_fraction** **key** **keyValue** **value_changed**	**SFFloat** **MFFloat** **MFVec3f** **SFVec3f**	
Coordi-nateInter-polator **NormalInter-polator**	**set_fraction** **key** **keyValue** **value_changed**	**SFFloat** **MFFloat** **MFVec3f** **MFVec3f**	
ScalarInter-polator	**set_fraction** **key** **keyValue** **value_changed**	**SFFloat** **MFFloat** **MFFloat** **SFFloat**	
ColorInter-polator	**set_fraction** **key** **keyValue** **value_changed**	**SFFloat** **MFFloat** **MFColor** **SFColor**	

Wirkung: Wert von **key** (zwischen 0 und 1) wird in Wert von **keyValue** "übersetzt" (durch lineare Interpolation zwischen den passenden Stützstellen). Die Länge der **keyValue**-Liste muss ein ganzzahliges Vielfaches der Länge der **key**-Liste sein (meist stimmen die Längen beider Listen überein). Die Kommunikation erfolgt über **set_fraction** (für den aktuellen Input) und **value_changed** (für den zugehörigen Output).

Beachte: Bei der Interpolation von Farben wird intern das HSV-Modell benutzt (obwohl die Farbwerte in **keyValue** und **value_changed** als RGB-Farbwerte spezifiziert werden).

Die Animation einer Szene mittels Interpolator-Knoten erfordert somit mindestens 2 ROUTE-Definitionen. Beispielsweise bei einer Positionsveränderung (Vektoren als Zwischenwerte):



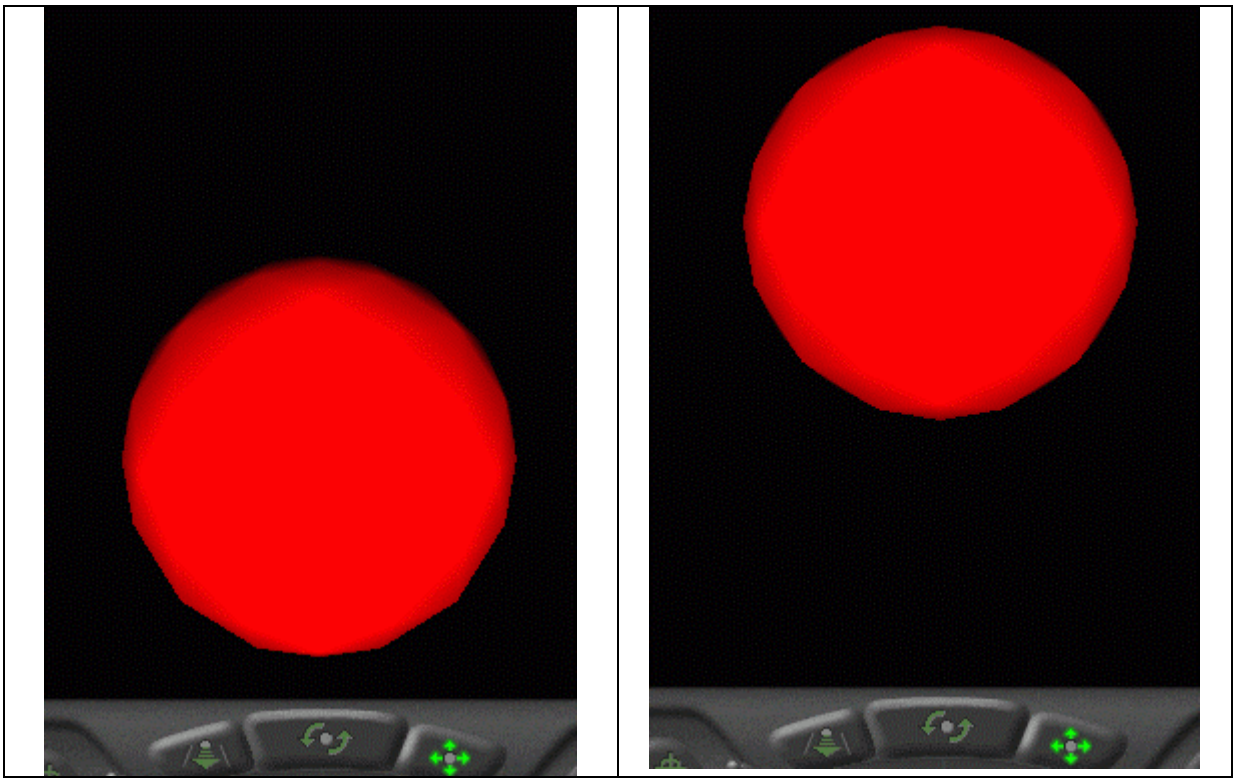
Beispiel:

Animation eines ständig hüpfenden Balls. Das Zeitintervall für den Vorgang (der unendlich wiederholt wird) wird dem **TimeSensor**-Knoten mit 2 Sek. vorgegeben. Der **PositionInterpolator**-Knoten enthält äquidistante Stützstellen im Feld **key** und zugehörige Punkte auf einer umgekehrten Parabel im Feld **keyValue**, um bei der Bewegung des Balles den Eindruck von Schwerkraft zu vermitteln.

```
#VRML V2.0 utf8
#Animation: Huepfender Ball

DirectionalLight
   { direction 0 0 -1 }
DEF Chronos TimeSensor
   {
   cycleInterval 2
   loop TRUE
   startTime 1
   }
DEF PosCalc PositionInterpolator
   {
   key [ 0 0.1 0.2 0.3 0.4 0.5
         0.6 0.7 0.8 0.9 1 ]
   keyValue [ 0 0 0, 0 0.09 0, 0 0.16 0,
              0 0.21 0, 0 0.24 0, 0 0.25 0,
              0 0.24 0, 0 0.21 0, 0 0.16 0,
              0 0.09 0, 0 0 0 ]
   }
DEF Verschieb Transform
   {
   children
      [
      Shape
         {
         geometry Sphere { radius 0.2 }
         appearance Appearance
            {
            material Material
               { diffuseColor 1 0 0 }
            }
         }
      ]
   }
ROUTE Chronos.fraction_changed TO
   PosCalc.set_fraction
ROUTE PosCalc.value_changed TO
   Verschieb.set_translation
```

Ergebnis (2 Snapshots aus der Animationssequenz):



Beispiel einer Farbinterpolation:
Farbwechsel einer Kugel

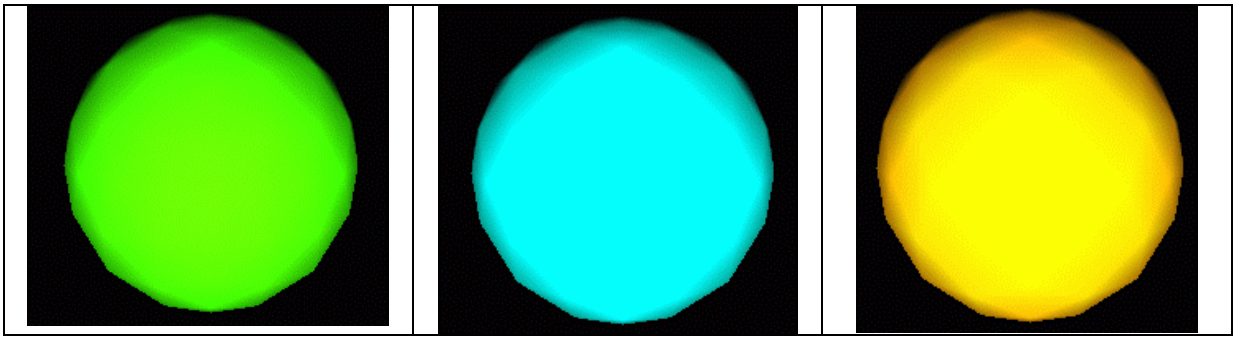
Die hierfür benötigten Ergebnisse des **`ColorInterpolator`**-Knotens werden an eines der Felder im **`Material`**-Knoten weitergeleitet, hier an **`diffuseColor`** (ein **`exposedField`**):

```
#VRML V2.0 utf8
#Animation: Farbwechsel einer Kugel

DEF Chronos TimeSensor
   {
   cycleInterval 10
   loop TRUE
   startTime 1
   }
```

```
DEF FarbCalc ColorInterpolator
   {
   key [ 0 0.333 0.667 1 ]
   keyValue [ 1 0 0, 0 1 0, 0 0 1, 1 0 0 ]
   }
Transform
   {
   children
      [
      DirectionalLight { }
      Shape
         {
         geometry Sphere { }
         appearance Appearance
           {
           material DEF KugelFarbe Material { }
           }
         }
      ]
   }
ROUTE Chronos.fraction_changed TO
   FarbCalc.set_fraction
ROUTE FarbCalc.value_changed TO
   KugelFarbe.set_diffuseColor
```

Ergebnis (3 Schnappschüsse):

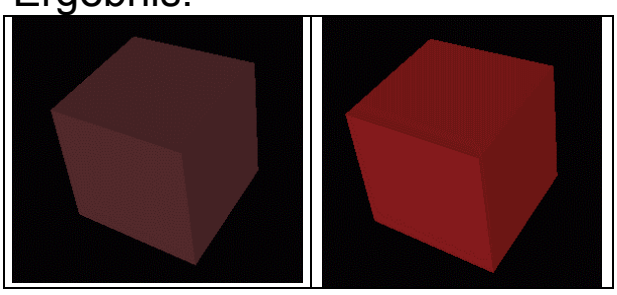


Beispiel eines Blinkers:

```
#VRML V2.0 utf8

Transform
   {
   children
      [
      Shape
         {
         geometry Box { }
         appearance Appearance
            {
            material DEF Blinker Material { }
            }
         }
      ]
   }
DEF Timer TimeSensor
   {
   cycleInterval 5.0  # 5 Sek. Zyklusintervall
   loop TRUE
   }
DEF Farbgeber ColorInterpolator
   {
   key [ 0.0, 1.0 ]
   keyValue [ 0 0 0, 1 0 0 ]
   }
ROUTE Timer.fraction_changed TO
   Farbgeber.set_fraction
ROUTE Farbgeber.value_changed TO
   Blinker.set_diffuseColor
```

Ergebnis:



Interaktion

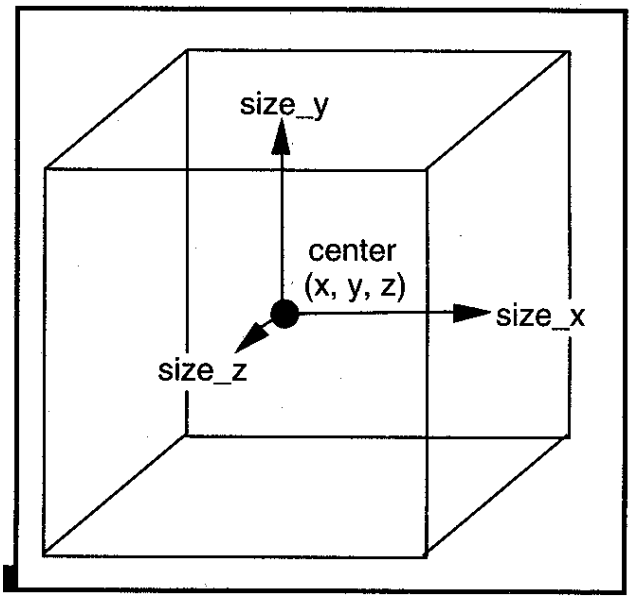
Die Einbeziehung von Benutzer-Eingaben erfordert *Sensorknoten* (von denen der **`TimeSensor`** bereits ein Sonderfall war):

`TouchSensor`	Berührungssensor (wird aktiv bei Anklicken mit der Maus)
`ProximitySensor`	Annäherungssensor, spezifiziert durch umgebende Box
`PlaneSensor`	Abbildung der Mauscursor-Bewegung in eine Ebene parallel zur *xy*-Ebene
`CylinderSensor`	Abb. der Mauscursor-Bewegung auf eine Drehung einer Zylinderfläche mit Zylinderachse parallel zur y-Achse
`SphereSensor`	Abb. der Mauscursor-Bewegung auf eine Drehung einer Kugelfläche
`VisibilitySensor`	Ermittlung der Sichtbarkeit einer Box

`Touch-Sensor`	**`enabled`**	**`SFBool`**	Start/Stop des Sensors
	`hitPoint_changed`	**`SFVec3f`**	)
	`hitNormal_changed`	**`SFVec3f`**	)
	`hitTexCoord_changed`	**`SFVec2f`**	) **`eventOut`**-
	`isActive`	**`SFBool`**	) Felder
	`isOver`	**`SFBool`**	)
	`touchTime`	**`SFTime`**	)
`Proxi-mity-Sensor`	**`center`**	**`SFVec3f`**	Mittelpkt. der Box
	`size`	**`SFVec3f`**	Ausdehnung
	`enabled`	**`SFBool`**	Start/Stop
	`position_changed`	**`SFVec3f`**	)
	`orientation_changed`	**`SFRota-tion`**	) **`eventOut`**-) Felder
	`enterTime`	**`SFTime`**	)
	`exitTime`	**`SFTime`**	)
	`isActive`	**`SFBool`**	)

Plane-Sensor	**autoOffset** **enabled** **offset** **maxPosition** **minPosition** **trackPoint_changed** **translation_changed** **isActive**	**SFBool** **SFBool** **SFVec3f** **SFVec2f** **SFVec2f** **SFVec3f** **SFVec3f** **SFBool**	Flag für Additivität der Bewegungskoordinaten Start/Stop Offset zur Koord.-ausg. Einschränkung der Sensoraktivität) **eventOut**-) Felder)
Cylinder-Sensor	**autoOffset** **enabled** **diskAngle** **offset** **maxAngle** **minAngle** **trackPoint_changed** **rotation_changed** **isActive**	**SFBool** **SFBool** **SFFloat** **SFFloat** **SFFloat** **SFFloat** **SFVec3f** **SFRotation** **SFBool**	analog zum **Plane-Sensor**-Knoten
Sphere-Sensor	**autoOffset** **enabled** **offset** **trackPoint_changed** **rotation_changed** **isActive**	**SFBool** **SFBool** **SFRotation** **SFVec3f** **SFRotation** **SFBool**	analog zum **Plane-Sensor**-Knoten
Visibility-Sensor	**center** **size** **enabled** **enterTime** **exitTime** **isActive**	**SFVec3f** **SFVec3f** **SFBool** **SFTime** **SFTime** **SFBool**	Mittelpkt. Box Ausdehnung Start/Stop)) **eventOut**)

Skizze zur Bedeutung der Felder des **ProximitySensor**-Knotens:



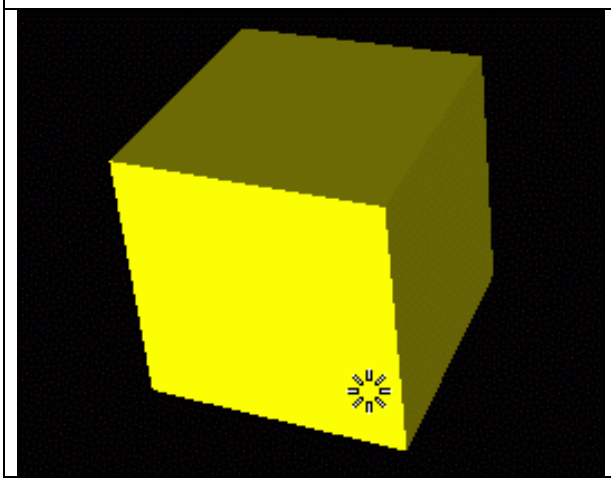
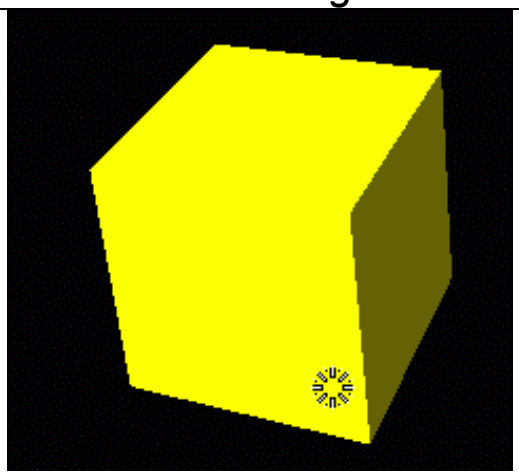
Beispiel:
Einschalten einer Lichtquelle mit dem **TouchSensor**

```
#VRML V2.0 utf8
# Lichtschalter

PointLight
   { location 0 0 10 }
DEF Licht2 PointLight
   {
   location 0 2 0
   on FALSE
   }
Transform
   {
   children
      [
      Shape
         {
         geometry Box { }
         appearance Appearance
```

```
            {
            material Material
               { diffuseColor 1 1 0 }
            }
         }
      DEF Schalter TouchSensor { }
      ]
   }
ROUTE Schalter.isActive TO Licht2.on
```

Ergebnis:

ohne Benutzereingriff (Licht von oben ist ausgeschaltet):	*Benutzer hat durch Anklicken des Würfels mit der Maus das Licht von oben eingeschaltet:*
	

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.

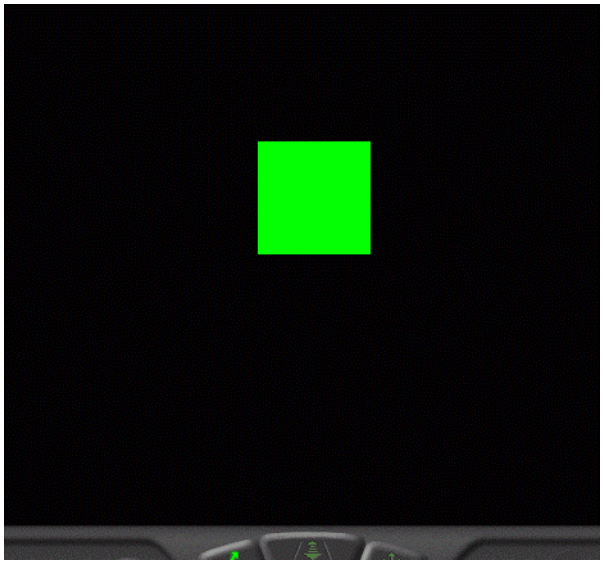
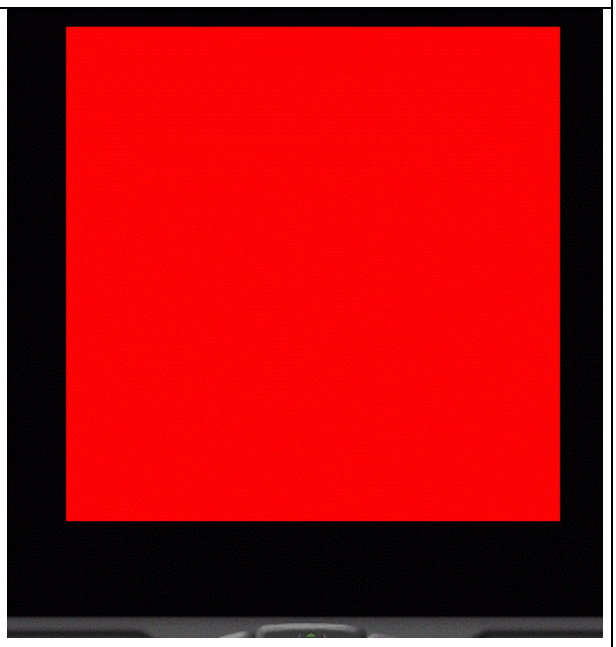
Beispiel:
Ein Würfel wird von einem größeren **proxy**-Würfel umgeben. Bei Kollision mit dem unsichtbaren, äußeren Würfel ändert der innere Würfel seine Farbe von Grün nach Rot:

```
#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:

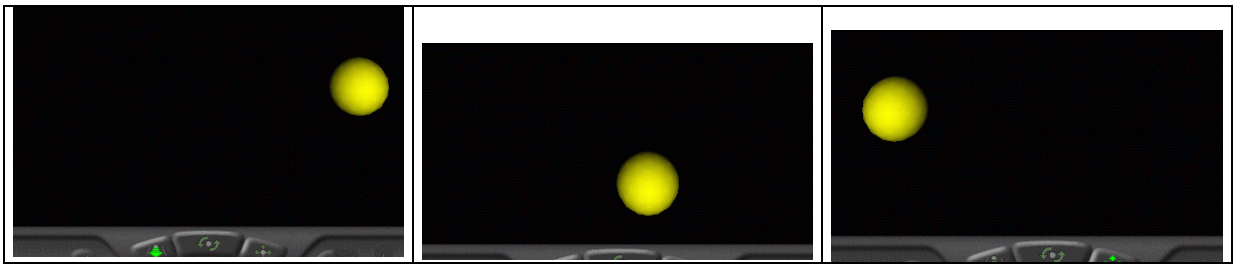
```
#VRML V2.0 utf8

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):