

11. Computergrafik

11.1 Grundlagen der 2D-Computergrafik

11.2 2D-Vektorgrafik mit XML: SVG

11.3 Grundlagen der 3D-Computergrafik

11.4 3D-Computergrafik: VRML (Fortsetzung)



Szenegraphen: Group- und Transform-Knoten

- Ein *Szenegraph* ist eine Baumstruktur (genauer: DAG), die alle in einer 3-dimensionalen virtuellen Welt enthaltenen Objekte mit ihren Eigenschaften enthält
- Wurzel des Szenegraphen: **Group-Knoten**
 - enthält Liste von Objekten im **children**-Feld
- Darstellung an anderer Stelle als im Ursprung durch **Transform-Knoten**
 - Anwendung von Transformationen *in folgender Reihenfolge*
 - **children**-Feld gibt Knoten an, die transformiert werden
 - **center**-Feld: Definition eines neuen Mittelpunkts
 - **rotation**-Feld: Drehung um Winkel
 - » Angabe in rad
 - » (Klassisch: Tripel: x-Achse (*pitch*), y-Achse (*yaw*), z-Achse (*roll*))
 - » In VRML: Rotationsachse (Tripel) + Winkel
 - » Positives Vorzeichen bedeutet Rechtsdrehung
 - **scale**-Feld: Maßstäblich veränderte Darstellung
 - **translation**-Feld: Verschiebung um Vektor

Beispiel: Einfacher Szenegraph

```
Group {
  children [
    Transform {
      children [
        Shape {
          appearance Appearance {
            material Material {
              diffuseColor 1.0 0 0
            }
          }
          geometry Box {
            size 2.0 2.0 2.0
          }
        }
      ]
      translation 2.0 0 0
    }
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0 0 1.0
        }
      }
      geometry Sphere {
        radius 1.0
      }
    }
  ]
}
... (rechte Spalte)

...
Transform {
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0 1.0 0
        }
      }
      geometry Box {
        size 2.0 2.0 2.0
      }
    }
  ]
  translation -2.0 0 0
}
]
NavigationInfo {
  type "EXAMINE"
}
```

Animation von 3D-Grafik

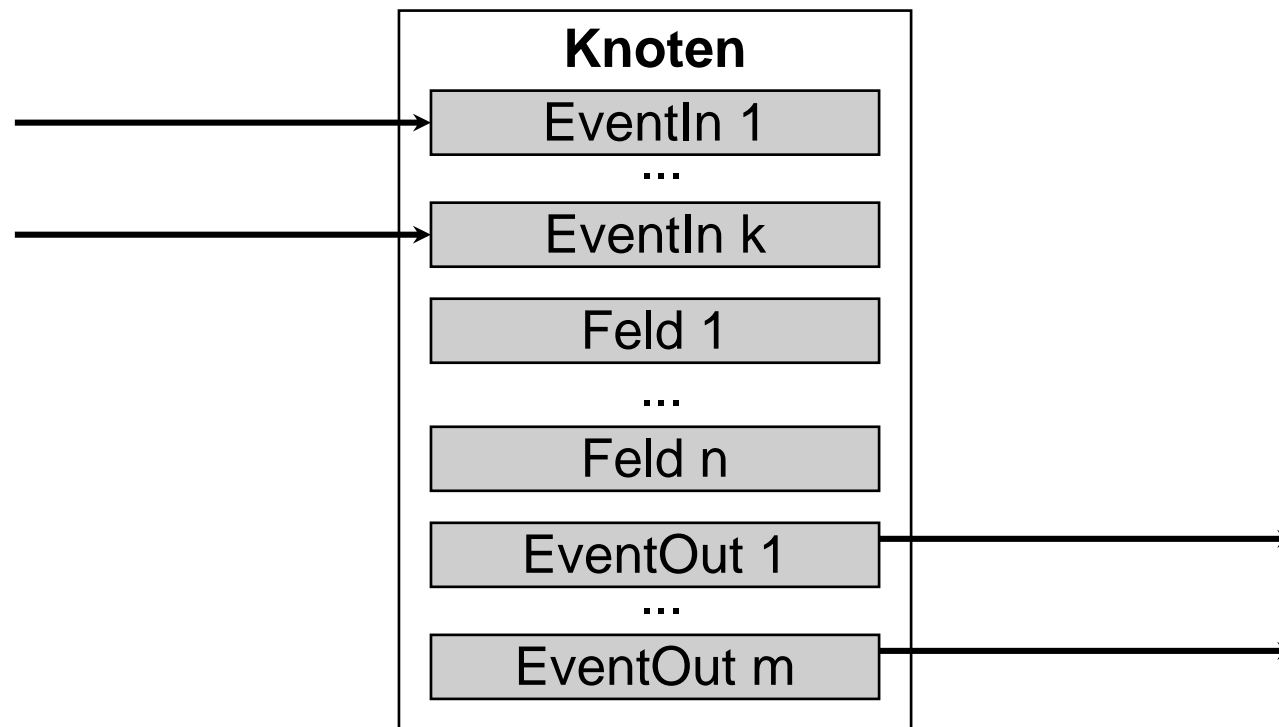
- *Dynamik* in der Darstellung von 3D-Szenen - drei mögliche Ursachen:
 - Veränderung der Betrachterposition
 - » auch in *statischen* 3D-Szenen möglich
 - Automatische, d.h. von selbst ablaufende, Veränderungen innerhalb der 3D-Szene
 - » z.B. Bewegung oder Veränderung von Farbe/Beleuchtung
 - » *Animation*
 - Vom Benutzer oder anderen systemexternen Quellen gesteuerte Veränderungen innerhalb der 3D-Szene
 - » *Interaktion*
- Wesentlich sowohl für Animation als auch Interaktion:
 - Veränderung des Modells als Reaktion auf *Ereignisse* (*dynamische* Szenen)
 - » Zeitereignisse und externe Ereignisse
 - » Ausschliesslich Zeitereignisse und deren Folgeereignisse = Animation

Ereignisse in VRML

- Ereignisentstehung:
 - *Sensoren*, eine spezielle Art von VRML-Objekten, erzeugen Ereignisse zu bestimmten Zeitpunkten
- Für Animation wichtig:
 - Zeitsensoren (Taktgeber)
- Für Interaktion wichtig:
 - Sensoren für Benutzerinteraktion (z.B. TouchSensor)
- Ereignisverarbeitung:
 - Ereignisse können an beliebige Objekte weitergeleitet (*routed*) werden
 - In einem empfangenden Objekt
 - » können Veränderungen von Attributen ausgelöst werden
 - » können erneut Ereignisse ausgesendet werden (*Ereigniskaskade*)
- Spezialobjekte zur Ereignisumsetzung:
 - insbesondere *Interpolatoren* zur drastischen Reduzierung der zu betrachtenden Ereignisanzahl

Knotenattribute in VRML

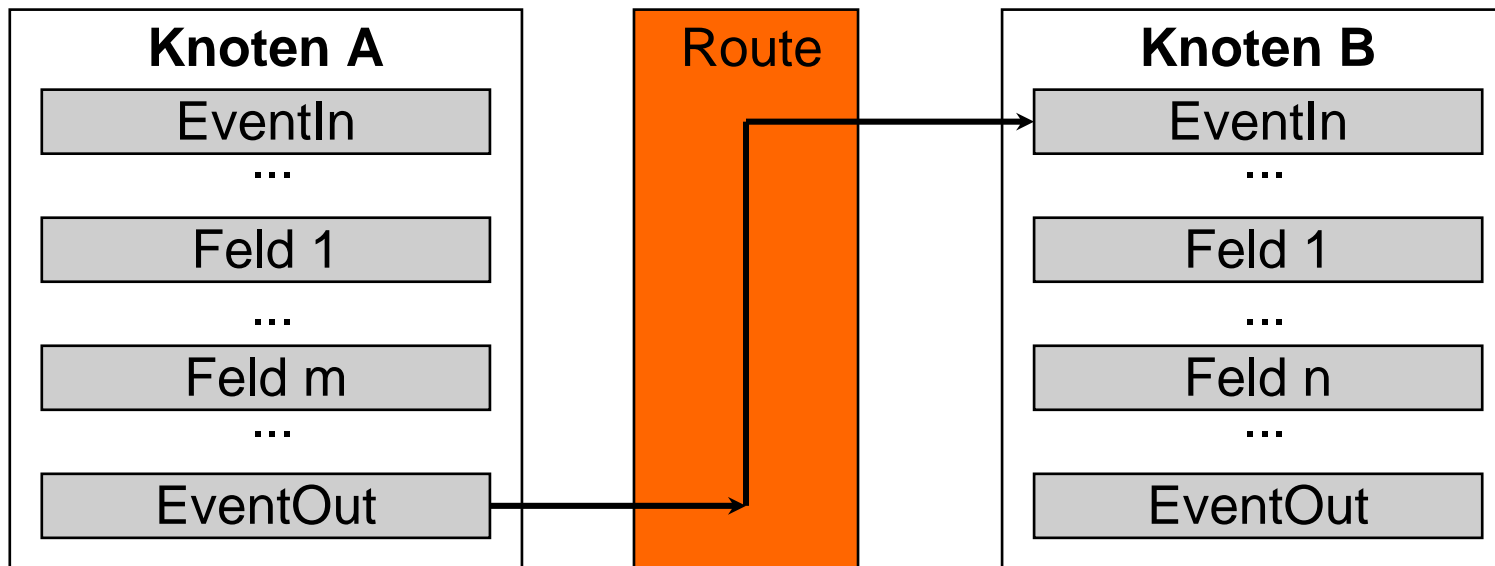
- Knoten können drei Arten von Attributen haben:
 - Felder zur statischen Festlegung von Eigenschaften
 - *EventIn*-Attribute zum Empfang von Ereignissen
 - *EventOut*-Attribute zum Senden von Ereignissen



Ereignisweitergabe auf Routen

- Ereignisse werden an spezifische Zielobjekte weitergegeben
 - Benennung von Knoten mit *DEF Bezeichner Knoten*
- Das Zielobjekt wird nicht beim Erzeugen des Ereignisses spezifiziert, sondern in einem speziellen Sprachkonstrukt: *Route*

ROUTE KnotenA.EventOut TO KnotenB.EventIn



Fan-Out (mehrere Empfänger für gleiches Ereignis) unproblematisch
Fan-In (mehrere Ereignisse gleichzeitig an ein Objekt) problematisch

Ereignisempfang

- Die meisten Knotentypen unterstützen (**EventIn-**)Ereignisse der Art *set_Feldwert*
- Beispiele für Verwendung:
 - Setzen der absoluten Position (translation) in einem **Transform**-Knoten
 - Setzen von Rotationswerten in einem **Transform**-Knoten
 - Setzen von Farbwerten

Zeitsensoren

- Zeitsensor:
 - Uhr, die regelmäßig Zeitereignisse generiert (Taktgeber)
- Zwei Verwendungsarten:
 - Absolute Zeit (normale Uhr):
 - » Gibt verstrichene Zeit seit Referenzzeitpunkt an (1. Januar 1970, 0:00 Uhr GMT)
 - Relative Zeit (Stopp-Uhr): *wesentlich häufiger verwendet!*
 - » Gesamtdauer des Ablaufs festgelegt
 - » Relative Zeit beginnt bei Start mit 0 und überschreitet nie die Gesamtdauer des Ablaufs
 - » Automatische Wiederholung (= Rücksprung der relativen Zeit zu 0) möglich
 - » Relative Zeiten werden in VRML als Bruchteile der Gesamtdauer (Reelle Zahlen zwischen 0 und 1) angegeben

TimeSensor-Knoten

- Knotentyp **TimeSensor**
 - Erlaubt diverse Feldtypen
 - Erzeugt Ereignisse
- Feldtyp **enabled**: Uhr ein/aus
- Feldtyp **startTime**: Startzeit, default 0.0
- Feldtyp **stopTime**: Endzeit, default 0.0
- Feldtyp **cycleInterval**: Zeitintervall für relative Zeitmessung
- Feldtyp **loop**: Wiederholung ein/aus, default **FALSE**
 - Endlosschleife möglich durch **stopTime = startTime**
- Ereignis (**EventOut**) **time**: absolute Zeit
- Ereignis (**EventOut**) **fraction_changed**: relative Zeit
 - wichtig zur Steuerung von Animationen

Beispiele für (relative) Zeitsensoren in VRML

- Einmaliger Ablauf von Dauer 6 Sekunden

```
DEF Clock TimeSensor {  
    cycleInterval 6.0  
}
```

- Endlosschleife mit Periode 6 Sekunden

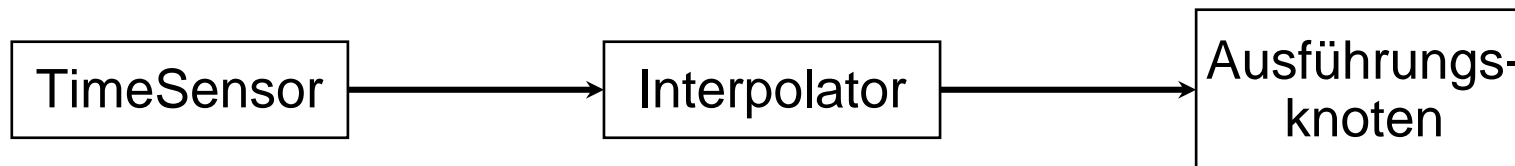
```
DEF Clock TimeSensor {  
    cycleInterval 6.0  
    loop TRUE  
}
```

- Vier Durchläufe mit Periode 6 Sekunden, insgesamt 24 Sekunden

```
DEF Clock TimeSensor {  
    cycleInterval 6.0  
    loop TRUE  
    stopTime 24.0  
}
```

Interpolatoren

- *Interpolatoren* dienen zur Schlüsselwert-orientierten Definition von Animationen
 - Vollständige Angabe von Animationen zu umfangreich
 - Schlüsselwert: Definierter Wert (*key value*) zu einem bestimmten Zeitpunkt (*key*)
- Interpolator berechnet durch lineare Interpolation alle Zwischenwerte zwischen den gegebenen Schlüssel-/Wert-Paaren
- Typische Anwendung von Interpolatoren in Ereignisverarbeitung:



Genereller Aufbau von Interpolatoren

- Alle Interpolatoren in VRML haben folgende Elemente
- Feldtyp **key**
 - Liste mit Zeitwerten, zu denen Schlüsselwerte festgelegt werden sollen
 - Müssen den Gesamtzeitraum *nicht* linear aufteilen
 - z.B.: `key [0.0, 0.15, 1.0]`
- Feldtyp **keyValue**
 - Liste mit Schlüsselwerteeinstellungen für die angegebenen Zeitpunkte
 - Sollten genau das Format aufweisen, das der empfangende Ausführungsknoten erwartet
 - Sinnvoll: Gleich viele Werte wie Zeitwerte im zugehörigen **key**-Feld
 - z.B.: `keyValue [0 1 0 0.00, 0 1 0 1.57, 0 1 0 3.14]`
- Eingabeereignis (**EventIn**) `set_fraction`
 - Passend zu den Ausgabeereignissen von Zeitsensoren
- Ausgabeereignis (**EventOut**) `value_changed`
 - Zur Weitergabe von Feldwertänderungen an Ausführungsknoten

OrientationInterpolator

- Zweck:
 - Drehung von Objekten in VRML-Animationen
- Schlüsselwerte:
 - Entsprechend der Konventionen von `rotation`-Feldern in `Transform`-Knoten
 - D.h.:
 - » 3 Werte für Rotationsachse
 - » 1 Wert für Rotationswinkel
 - Beispiel:

```
keyValue [  
    0 1.0 0 0.00,  
    0 1.0 0 1.57,           =  $\pi/2$   
    0 1.0 0 2.36,           =  $3\pi/4$   
    0 1.0 0 3.14  
]
```

Beispiel: Würfeldrehung

```
DEF RotCube Transform {
  children [
    Shape {
      appearance Appearance {
        texture ImageTexture {
          url "textur0.gif"
        }
      }
      geometry Box {
        size 2.0 2.0 2.0
      }
    }
  ]
}
```

```
DEF Clock TimeSensor {
  cycleInterval 6.0
  loop TRUE
}
```

```
DEF Interpolator OrientationInterpolator {
  key [0.0, 1.0]
  keyValue [
    0 1.0 0 0.00,
    0 1.0 0 3.14
  ]
} ...nächste Spalte
```

```
...
NavigationInfo {
  type "EXAMINE"
}

ROUTE Clock.fraction_changed
  TO Interpolator.set_fraction
ROUTE Interpolator.value_changed
  TO RotCube.set_rotation
```

Animation - Fortsetzung

Beispiel: Nichtlineare Geschwindigkeit

...

```
DEF Interpolator OrientationInterpolator {  
  key [0.0, 0.15, 0.85, 1.0]  
  keyValue [  
    0 1.0 0 0.00,  
    0 1.0 0 1.57,  
    0 1.0 0 2.36,  
    0 1.0 0 3.14  
  ]  
}  
...
```


NavigationInfo

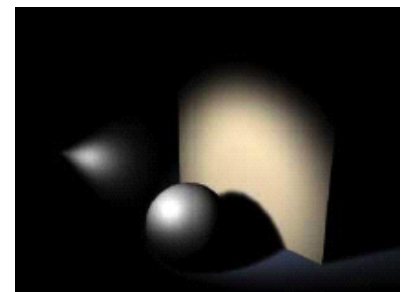
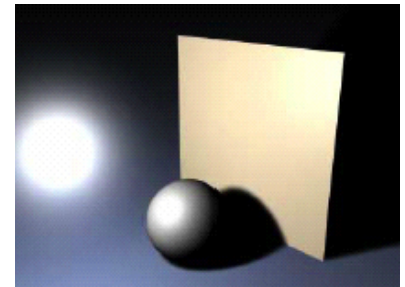
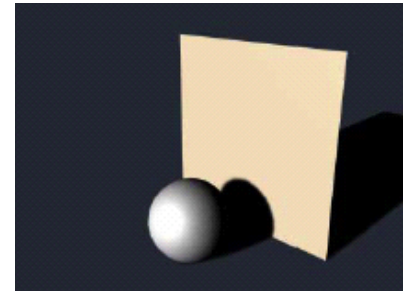
- Gibt globale Zusatzinformation für das Rendering an:
 - Z.B. Standardmodus (Walk, Fly, Examine)
 - Z.B. Standardgeschwindigkeit
- Headlight:
 - Standardlichtquelle (directional) aus Betrachtersicht
 - kann in NavigationInfo ausgeschaltetet werden

- Beispiel:

```
NavigationInfo {  
    type "EXAMINE"  
    headlight FALSE  
}
```

Szenenbeleuchtung: Lichttypen

- Drei Lichttypen werden in VRML unterstützt (eigene Knotentypen)
- **Directional Light:**
 - parallel gerichtetes Licht einer unendlich weit entfernten Quelle
 - keine Abschwächung mit der Entfernung
- **PointLight:**
 - Licht breitet sich gleichmäßig von punktförmiger Quelle aus (z.B. Glühlampe)
 - Abschwächung mit der Entfernung
- **SpotLight:**
 - Licht breitet sich kegelförmig von punktförmiger Quelle aus (z.B. Taschenlampe)
 - Abschwächung mit der Entfernung und mit Winkel
- Wichtigste Feldtypen:
 - **direction**-Feld: Richtungsvektor
 - **ambientIntensity**-Feld: Stärke des Einflusses auf Objekte
 - **color**-Feld: Lichtfarbe
 - **location**-Feld: Position der Lichtquelle



Beispiel: Szene mit Beleuchtung

```
DirectionalLight {
  direction 0 -1.0 0
  ambientIntensity 0.7
  color 1.0 1.0 1.0
}

SpotLight {
  location -5.0 3.0 0
  direction 0.5 -0.5 0.0
  ambientIntensity 0.4
  color 1.0 1.0 1.0
}

Group {
  ... wie früheres Beispiel zu Gruppen
}

NavigationInfo {
  headlight FALSE
}
```

PositionInterpolator

- Zweck:
 - Bewegung von Objekten in VRML-Animationen
- Schlüsselwerte:
 - Entsprechend der Konventionen von `translation`-Feldern in `Transform`-Knoten
 - D.h.:
 - » 3 Werte für aktuelle Position
 - Beispiel:

```
keyValue [  
  0 3 0,  
  0 0.5 0,  
  0 1.5 0,  
  0 2 0,  
  0 1.5 0,  
  0 0.5 0,  
  0 1 0,  
  0 0.5 0,  
  0 3 0  
]
```

Beispiel: Fallende Kugel

```
DEF Ball Transform {
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0.5 0 0.8
        }
      }
      geometry Sphere {
        radius 0.5
      }
    }
  ]
}
```

```
DEF Plane Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0 1 0
    }
  }
  geometry Box {
    size 4 0.1 4
  }
}
```

```
DEF Clock TimeSensor {
  cycleInterval 6.0
  loop TRUE
}

DEF Interpolator
  PositionInterpolator {
    key [0.0, 0.05, 0.1, 0.15,
         0.2, 0.3, 0.4, 0.5, 1.0]
    keyValue [
      0 3 0,
      0 0.5 0,
      0 1.5 0,
      0 2 0,
      0 1.5 0,
      0 0.5 0,
      0 1 0,
      0 0.5 0,
      0 3 0
    ]
  }
}

ROUTE Clock.fraction_changed TO
  Interpolator.set_fraction
ROUTE Interpolator.value_changed
  TO Ball.set_translation
```

ColorInterpolator

- Zweck:
 - Veränderung der Farbe von Objekten in VRML-Animationen
- Schlüsselwerte:
 - Entsprechend der Konventionen zur Darstellung von RGB-Farben
 - » 3 Werte für aktuellen Farbton
 - Beispiel:

```
keyValue [  
    0.5 0 0.8, #violett  
    0.5 0 0.8,  
    0.5 0 0.8,  
    0.5 0 0.8,  
    0.5 0 0.8,  
    0.5 0 0.8,  
    0.5 0 0.8,  
    0.5 0 0.8,  
    1 0 0, #rot  
    1 0 0,  
    0.5 0 0.8  
]
```

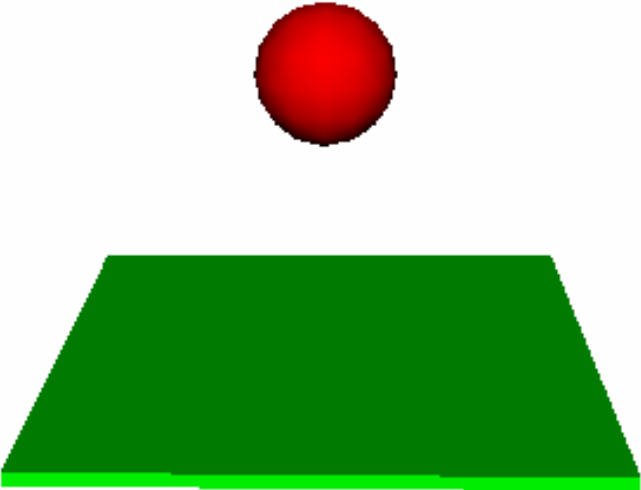
Beispiel: Kugelverfärbung (1)

```
...
DEF Ball Transform {
  children [
    Shape {
      appearance Appearance {
        material DEF Color Material {
        }
      }
      geometry Sphere {
        radius 0.5
      }
    }
  ]
}
...
DEF PInterpolator PositionInterpolator {
  ...
}
...
```

Beispiel: Kugelverfärbung (2)

```
...
DEF CInterpolator ColorInterpolator {
  key [0.0, 0.05, 0.1, 0.15, 0.2, 0.3,
       0.4, 0.5, 0.55, 0.95, 1.0]
  keyValue [
    0.5 0 0.8,
    0.5 0 0.8,
    0.5 0 0.8,
    0.5 0 0.8,
    0.5 0 0.8,
    0.5 0 0.8,
    0.5 0 0.8,
    0.5 0 0.8,
    1 0 0,
    1 0 0,
    0.5 0 0.8
  ]
}
ROUTE Clock.fraction_changed TO PInterpolator.set_fraction
ROUTE PInterpolator.value_changed TO Ball.set_translation

ROUTE Clock.fraction_changed TO CInterpolator.set_fraction
ROUTE CInterpolator.value_changed TO Color.set_diffuseColor
```



ScalarInterpolator

- Universeller Interpolator
- Berechnet beliebige reelle Zahlenwerte abhängig von gegebenen Schlüsselwerten
- Anwendungsbeispiele:
 - Animation der Grösse von Objekten
 - Animation der Transparenz von Objekten

Sensoren für Interaktionen in VRML

- TimeSensor: Uhr, Zeitgeber
- TouchSensor: Berührung mit dem Mauszeiger
- PlaneSensor: Verschiebung von Objekten in einer Ebene
- SphereSensor: Freie Rotation eines Objektes
- CylinderSensor: Rotation eines Objektes um eine Achse
- ProximitySensor: Nähe des Beobachters
- Collision: Kollision eines Beobachters mit dem Objekt
- LOD (=Level of Detail): Entfernung des Beobachters zum Objekt
- Anchor: Hyperlink

TouchSensor-Knoten

- Knotentyp **TouchSensor**
 - Kann an verschiedenen Stellen in Objekthierarchie eingebaut werden
- Ereignis (**EventOut**) **isOver**:
 - Erzeugt Ereignis, wenn sich Mauszeiger im Objekt befindet
- Ereignis (**EventOut**) **isActive**:
 - Erzeugt Ereignis, wenn Maus im Objekt gedrückt ist

Beispiel: Interaktion in Ball-Animation

```
DEF Ball Transform {  
  ...  
}  
  
Group {  
  children [  
    DEF Sensor TouchSensor {}  
    DEF Plane Shape {  
      ...  
    }  
  ]  
}  
  
DEF Clock TimeSensor {...}  
DEF PInterpolator PositionInterpolator {...}  
DEF CInterpolator ColorInterpolator {...}  
  
ROUTE ...  
ROUTE Sensor.isOver TO Clock.set_enabled
```

Beispiel: Kollisionserkennung

```
Collision {
  children [
    Transform {
      children [
        Shape {
          appearance Appearance {
            material Material {
              diffuseColor 1 0 0
            }
          }
          geometry Sphere {
            radius 0.5
          }
        }
      ]
      translation 0 0 -5
    }
  ]
  collide TRUE # Verhindert Eindringen in Objekt
}
```

Benutzerdefinierte Formen

- Beliebige Formen können über Koordinatenwerte definiert werden
 - Knotentyp `Coordinate`, Feldtyp `point`, Werte 3er-Gruppen von reellen Zahlen
- Bildung von Objekten mit `IndexedLineSet` bzw. `IndexedFaceSet`:
 - `IndexedLineSet` erzeugt Gittermodell, `IndexedFaceSet` Flächenmodell
 - Feld `coord` enthält die beteiligten Punkte
 - » Implizit werden die Punkte, mit 0 beginnend, nummeriert (je drei Zahlen = 1 Punkt)
 - Feld `coordIndex` enthält die einzelnen anzuzeigenden Linien bzw. Flächen
 - » Als Indizes in der Punktliste
 - » Jedes Element (Linie bzw. Fläche) mit -1 abgeschlossen

Beispiel: Würfel selbstdefiniert (Drahtgitter)

```
Shape {
  appearance ...
  geometry IndexedLineSet {
    coord Coordinate {
      point [
        -1.0 1.0 1.0, # Punkt 0: links oben vorn
        -1.0 -1.0 1.0, # Punkt 1: links unten vorn
        1.0 -1.0 1.0, # usw.
        1.0 1.0 1.0,
        -1.0, 1.0, -1.0,
        -1.0, -1.0, -1.0,
        1.0, -1.0, -1.0,
        1.0, 1.0, -1.0
      ]
    }
    coordIndex [
      0, 1, 2, 3, 0, -1, # vorderes Quadrat
      4, 5, 6, 7, 4, -1,
      0, 4, -1,
      1, 5, -1,
      2, 6, -1,
      3, 7
    ]
  }
}
```

Beispiel: Würfel selbstdefiniert (Flächen)

```
Shape {
  appearance ..
  geometry IndexedFaceSet {
    solid FALSE
    coord Coordinate {
      point [... wie oben ...]
    }
    coordIndex [
      0, 1, 2, 3, 0, -1,
      4, 5, 6, 7, 4, -1,
      0, 3, 7, 4, 0, -1,
      1, 2, 6, 5, 1, -1,
      3, 2, 6, 7, 3, -1,
      0, 1, 5, 4, 0
    ]
  }
}
```


Programmeinbindung in VRML: Script-Knoten

- Der Knotentyp `script` ermöglicht die Einbindung eines Programmskripts
 - Meist JavaScript, VRMLScript (spezialisierte Teilsprache von JavaScript) oder Java
- Skript kann (nach entsprechender Deklaration)
 - Feldwerte lesen und verändern
 - Eingabeereignisse verarbeiten
 - Ausgabeereignisse erzeugen

External Authoring Interface

- Ermöglicht Datenaustausch zwischen einer VRML-Welt und einem Java-Applet
 - Voraussetzung: Applet und VRML-Welt in gleiche Webseite integriert
- Beispiel einer Anwendung:
 - Komplexe Visualisierung, z.B. von geographischen Daten, in mehreren Fenstern
 - Fenster 1: Dreidimensionale Ansicht (VRML-Welt)
 - Fenster 2: Navigations- oder Übersichtsfenster, z.B. Landkarte mit aktuellem Ausschnitt oder Standort
- Ermessensentscheidung beim Programmwurf:
 - VRML-Dokument mit hohem Programmlogik-Anteil oder
 - Programm mit Einbindung von VRML-Dokumenten (z.B. Java 3D)

Aktuell: X3D

- Neuer Standard (Web3D-Konsortium):
 - X3D (eXtensible3D-Graphics)
- XML-Anwendung
 - Dateien folgen XML-Syntax (alternativ VRML-Encoding)
- Enge Integration mit anderen Web-Technologien
- Allgemeine XML-Werkzeuge anwendbar
 - Z.B. Navigation mit XPath
 - Z.B. Transformation mit XSLT
- Entwicklungsstand:
 - Sommer 2003: Draft International Standard ISO 19775
 - Industrielle Unterstützung derzeit allerdings vorwiegend durch kleine Nischenfirmen (z.B. „Media Machines“, Tony Parisi)
- Übergang von VRML zu X3D:
 - X3D-Viewer geben im Allgemeinen auch VRML wieder
 - Transformationsprogramme