11. Computergrafik und Virtuelle Realität

11.1 Grundlagen der 2D-Computergrafik
11.2 2D-Vektorgrafik mit XML: SVG
11.3 Grundlagen der 3D-Computergrafik 
11.4 3D-Computergrafik: VRML

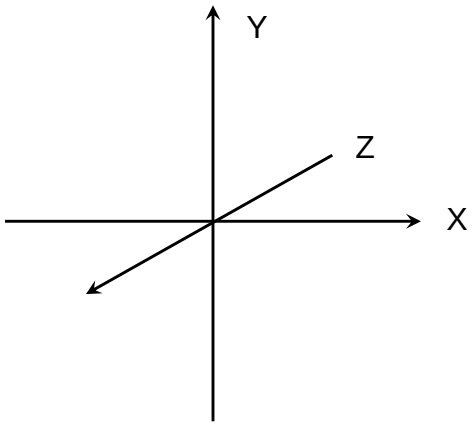
Literatur:

Alan Watt: 3D Computergrafik, 3. Auflage, Pearson Studium 2002

Dreidimensionale Darstellung

- Dimensionenkonflikt:
  - Die reale Welt ist dreidimensional
  - Bilddarstellungen (wie bisher betrachtet) sind zweidimensional
    - » Verdeckte Ansichten und Details
- Dreidimensionale Darstellung:
  - Setzt Modell mit den Informationen in allen drei Dimensionen voraus
    - » Alle möglichen Ansichten ohne Informationsverlust
- Anwendungsbereiche für dreidimensionale Darstellung:
  - Virtuelle Welten, „Cyberspace“
  - Ingenieuranwendungen:
    - » CAD (z.B. Maschinenbau)
    - » Designmodelle von Produkten
    - » (Interaktive) Architekturmodelle
  - Produktpräsentation
  - Geovisualisierung
  - Animation im Film (Trickfiguren in Spielfilm, Vollanimation)

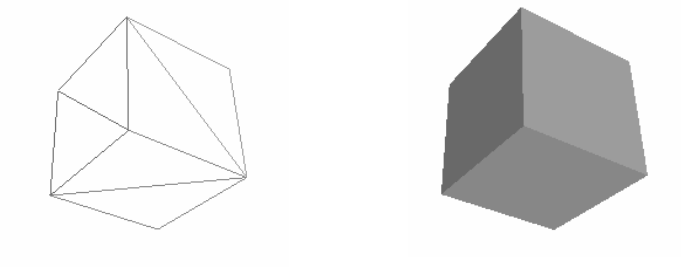
3D-Koordinatensystem



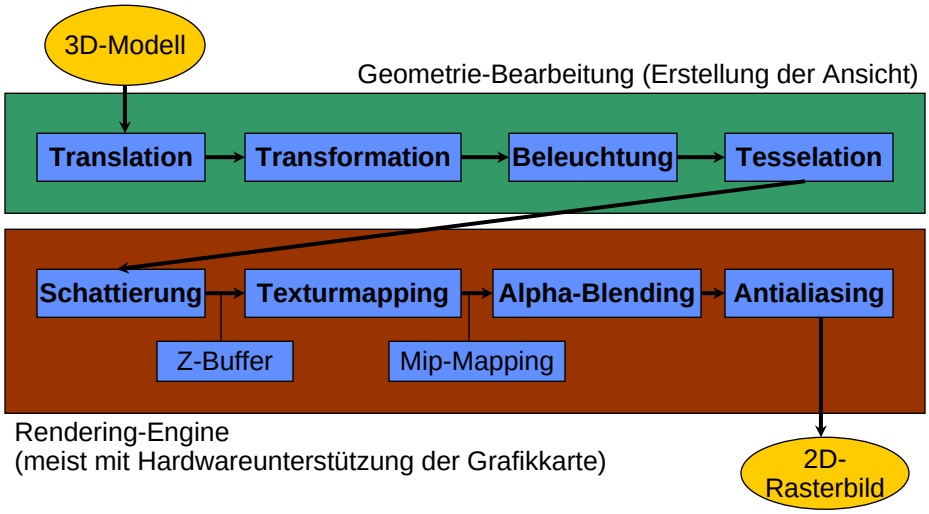
Kartesisches Koordinatensystem
Merkhilfe: „Rechte-Hand-Regel“

Grundidee der 3D-Modellierung

- Gegenstände:
  - Punktwolken im 3-dimensionalen Raum
  - Zusatzinformationen z.B. zur Oberflächenstruktur
- Verbindung der Punkte in definierter Weise:
  - » Rendering als *Drahtmodell*
- Anpassung des Rendering an visuelle Wahrnehmung:
  - Perspektive, Verdeckung



3D-Rendering-Pipeline



Translation

- Übersetzung der Modellkoordinaten in den zum Rendering benutzten Koordinatenraum
  - Integration von Modellen aus verschiedenen Quellen, z.B. verschiedenen Entwicklungssystemen
  - Häufig: *Weltkoordinatensystem*
- *Clipping*
  - Abschneiden von Objektteilen außerhalb des Blickwinkels des Beobachters

Transformation

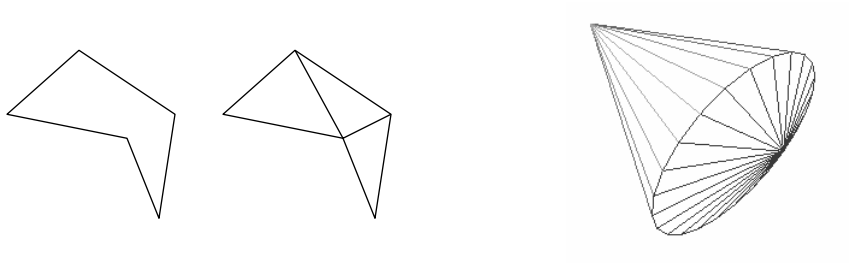
- Änderung der Objektposition
  - Verschieben (oft auch *translation* genannt)
  - In 3 Freiheitsgraden
- Änderung der Objektausrichtung
  - Rotation
  - In 3 Freiheitsgraden
- Änderung der Objektgröße
  - Skalierung
  - 1 Freiheitsgrad bei Erhaltung der Proportionen
  - 3 Freiheitsgrade bei Verzerrung
- Bewegung der Betrachterposition in einer virtuellen Welt:
  - Obige Operationen treten (kombiniert) extrem häufig auf
  - Schnelle Implementierung wichtig

Beleuchtung

- Einfluss von Lichtquellen auf das Erscheinungsbild einer 3D-Szene
  - Ganz ohne Lichtquellen: Schwarz!
- Abhängig von:
  - Standort und Art von Lichtquellen
  - Spezialfall einer Standard-Lichtquelle:
    - » „Headlight“ aus der Richtung des Betrachters
  - Standpunkt und Blickrichtung des Betrachters

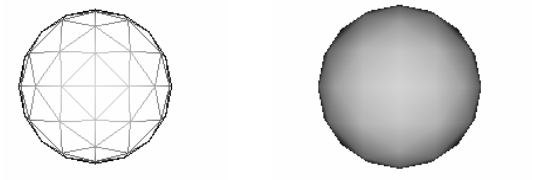
Tesselation

- Durch 3D-Rendering nur Polygone darstellbar!
  - Komplexe Szenen aus extrem vielen (Millionen) von Polygonen zusammengesetzt
- Darzustellendes Objekt wird in einfache Polygone (meist Dreiecke) zerlegt
- Tesselation = Zerlegung komplexer Polygone in Dreiecke



Schattierung *(shading)*

- *Flat-Shading:*
  - Berechnet für jedes Flächenelement (Polygon) der 3D-Szene einen Helligkeitswert
  - Bestimmt sich aus der Winkeldifferenz zwischen einfallendem Licht und dem Normalenvektor des Polygons
  - Einfach zu berechnen
  - Nachteil: Homogener Farbwert je Polygon
- Verfeinerte Schattierungsverfahren:
  - Z.B. Gouraud-Shading, Phong-Shading
  - Interpolation der Farbwerte mit benachbarten Polygonen
  - Erreicht relativ gute optische „Glättung“



Z-Buffer

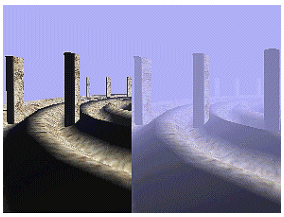
- Z-Buffer speichert für jeden Bildpunkt des 2D-Bildes die niedrigste Entfernung zu einem Objekt
- Beschleunigung des Rendering:
  - Offensichtlich verdeckte Objekte bzw. Objektteile müssen nicht berechnet werden
- Größere Wahlfreiheit bei der Abarbeitung des Rendering
  - Hintergrundbild muss nicht unbedingt zeitlich vor den Vordergrundobjekten gerendert werden
- Hardwareunterstützung in Grafikkarten

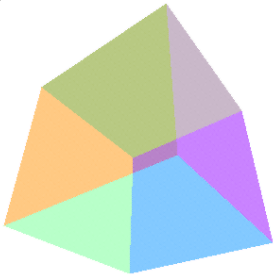
Textur-Mapping

- Textur:
  - Muster oder Bild, das auf Oberflächen von 3D-Objekten gelegt wird
  - Kann oft Anzahl der benötigten Polygone drastisch reduzieren
- Textur-Mapping:
  - Darstellung der Flächen eines Objekts mit Textur
  - Erweiterung: Perspektivische Korrektur der Textur
- Mip-Mapping:
  - Textur in mehreren Auflösungen verfügbar (und automatisch passende Fassung ausgewählt)

Alpha-Blending

- Kontrolle der Transparenz eines Objekts
  - Analog zu 2D-Rendering
- Bei 3D-Grafik:
  - Tiefeneindruck durch „Verwischen“ von Details bei grösserer Entfernung
  - Nebeleffekt *(fogging)*
  - „Depth cueing“ (Überblendung ins Schwarze)





11. Computergrafik

Literatur:

Henning, Taschenbuch Multimedia, Kap. 13
Rolf Däßler: VRML - 3D-Welten im Internet, bhv Verlag 2002
http://www.web3d.org
Mark Pesce: Vrml - Browsing and Building Cyberspace, New Rider 1995

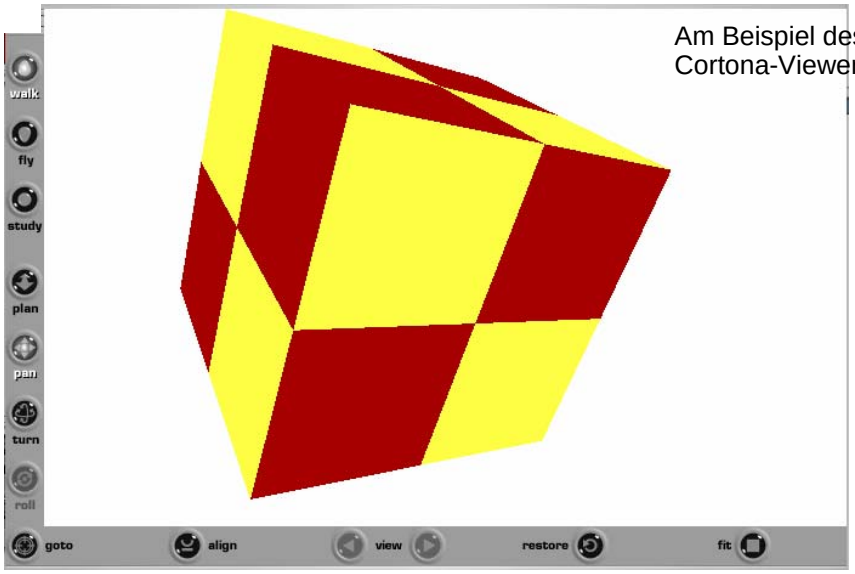
Virtual Reality Modeling Language VRML

- Skriptsprache und Austauschformat zur Beschreibung von 3D-Welten
  - Auf den Einsatz im Internet ausgelegt
  - Vektor-Grafikformat
- VRML hat *keine* XML-Syntax!
  - Nachfolger von VRML: „X3D“ ist XML-Sprache
- Geschichte:
  - Basiert auf Grafikstandard „OpenInventor“ von Silicon Graphics
  - Marc Pesce, Toni Parisi, 1994: Erster 3D-Browser, Entwurf VRML 1.0
  - April 1995: VRML Version 1.0 verabschiedet
  - 1996: Internet-Abstimmung über konkurrierende Vorschläge für VRML 2.0, gewonnen von *MovingWorlds*-Standard (Silicon Graphics & Sony), VRML 2.0 verabschiedet
  - 1997: VRML wird Internationaler Standard ISO-14772
    - » Meist als „VRML 97“ bezeichnet, weitgehend identisch zu VRML 2.0
- Dateiextension:
  - .wrl (wie „world“)

Softwarewerkzeuge für VRML

- Anzeigeprogramme *(viewer)*
  - Meist als „Plug-In“ für Web-Browser
  - Bekannte Produkte:
    - » *CosmoPlayer* (Silicon Graphics)
    - » *Cortona* (Parallel Graphics)
    - » *FreeWRL* (OpenSource-Aktivität)
- Autorenwerkzeuge:
  - Einfache syntaxunterstützende Editoren (z.B. VrmlPad)
  - Spezielle 3D-Editoren
  - Aufwändige 3D-Modellierungs- und Animationswerkzeuge mit VRML-Exportfunktion
    - » Z.B. 3d studio max, SoftImage, Maya, Cinema4D

Bedienungselemente eines VRML-Viewers



Am Beispiel des Cortona-Viewers

Navigationsmodi

- Grundmodi:
  - Walk: Bewegung des Betrachters nur in der horizontalen Ebene
  - Fly: Bewegung des Betrachters auch in der vertikalen Ebene
  - Study: Bewegung der Welt um das Zentrum des Objekts
- Optionen (in Kombination mit den Grundmodi):
  - Plan: Bewegungseingaben beziehen sich auf Verschiebung in der horizontalen Ebene
  - Pan: Bewegungseingaben beziehen sich auf Verschiebung in der vertikalen Ebene
  - Turn: Bewegungseingaben beziehen sich auf Drehung in der horizontalen Ebene
  - Roll: Bewegungseingaben beziehen sich auf Drehung in der vertikalen Ebene
- Bewegungseingaben erfolgen z.B. durch Pfeiltasten oder Mausgesten

Syntax von VRML

- Bezeichner empfindlich gegen Gross- und Kleinschreibung!
- Knoten:
  - *Knotentypbezeichner* { *Felder* }
  - Knotentypbezeichner beginnt immer mit Grossbuchstaben
  - Z.B. **`Sphere { radius 1.0 }`**
- Felder:
  - Folgen von Paaren *Feldtypbezeichner Feldwert*
  - Feldtypbezeichner beginnt immer mit Kleinbuchstaben
  - Z.B. **`radius 1.0`**, z.B. **`skyColor 1.0 1.0 1.0`**
- Wert: Tupel aus mehreren Einzelwerten, meist 1 bis 3
- Listen von Werten:
  - Z.B. **`[0, 1, 2, 3, 4`**], z.B. **`[-1.0 1.0 1.0, -1.0 -1.0 1.0]`**
- Datentypen:
  - Ganze Zahlen, reelle Zahlen, Zeichenketten, Boolesche Werte u.v.a.
- Einheiten:
  - VRML-Einheiten müssen extern interpretiert werden, z.B. Längeneinheit = Meter, Winkeleinheit = rad, Zeiteinheit = Sekunde

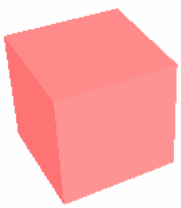
VRML-Beispiel: box0.wrl

```
#VRML V2.0 utf8
Background { skyColor 1.0 1.0 1.0 }

Shape {

  appearance Appearance {
    material Material {
      emissiveColor 1.0 0 0
    }
  }

  geometry Box {
    size 2.0 2.0 2.0
  }
}
```



Shape-Knoten

- Knotentyp **`Shape`**
  - Benötigt Felder **`appearance`** und **`geometry`**
- Feldtyp **`appearance`**
  - Enthält in der Regel einen Knoten vom Typ **`Appearance`**
    - » Angabe diverser Materialeigenschaften (Farbe, Schattierung, ...)
- Feldtyp **`geometry`**
  - Enthält Geometrieknoten
- Übersicht wichtiger Geometrieknotentypen:
  - **`Box:`** Quader **`(size)`**
  - **`Cone:`** Kegel **`(bottomRadius, height)`**
  - **`Cylinder:`** Zylinder **`(radius, height)`**
  - **`Sphere:`** Kugel **`(radius)`**
  - **`Text:`** 3D-Text
  - ...

Hintergrund

- Grundkonzept:
  - Kreis mit unendlichem Radius als Boden *(ground)*
  - Halbkugel mit unendlichem Radius als Himmel *(sky)*
- **`Background`**-Knoten:
  - Spezifikation der Boden- und Himmelfarben
    - » **`groundColor, skyColor`**
  - Möglichkeit der Beschreibung von Abstufungen
    - » Liste von Farben und Winkel, in denen sie angewandt werden
  - Möglichkeit der Einbindung von Texturen

Appearance- und Material-Knoten

- Knotentyp **`Appearance`**
  - Optionale Felder **`material, texture`** und **`textureTransform`**
- Feldtyp **`material`**
  - Enthält in der Regel einen Knoten vom Typ **`Material`**
- Feldtyp **`texture`**
  - Enthält einen Texturknoten (siehe unten)
- Feldtypen im Materialknoten:
  - (Werte immer zwischen 0.0 und 1.0)
  - **`ambientIntensity:`** Reflexion für Umgebungslicht
  - **`diffuseColor:`** Reflektierende (nicht leuchtende) Farbe
  - **`emissiveColor:`** Selbstleuchtende Farbe
  - **`shininess:`** Stärke von Glanzlichtern
  - **`specularColor:`** Farbe von Glanzlichtern
  - **`transparency:`** Durchsichtigkeit
- Spezifikation von Farben:
  - als RGB-Wert (Zahlentripel)

Beispiel: Materialeigenschaften

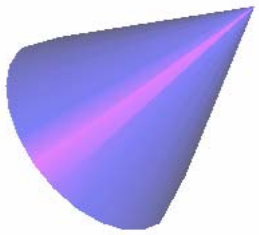
```
#VRML V2.0 utf8
Background { skyColor 1.0 1.0 1.0 }

Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0.2 0.2 1.0
      shininess 1.0
      specularColor 1.0 0 0
      transparency 0.3
    }

  }

  geometry Cone {
    bottomRadius 1.0
    height 2.0
  }

}
```



Texturen

- Knotentyp **`ImageTexture`**
  - Benötigt Feldtyp **`url`** zur Angabe einer Datei mit 2D-Grafik (JPEG, PNG, GIF)
  - Achsen des Texturbildes mit S (horizontal) und T (vertikal) bezeichnet
  - Feldtypen **`repeatT, repeatS`** (Boolean) zur Steuerung der Wiederholung
- Knotentyp **`PixelTexture`**
  - Direkte Angabe einer Textur als Pixelfeld in VRML
- Knotentyp **`MovieTexture`**
  - Analog zu **`ImageTexture`**, aber mit Bewegtbild (MPEG-1)
  - Zusätzliche Feldtypen: **`loop, speed, startTime, stopTime`**

Beispiel: Quader mit Textur

```
#VRML V2.0 utf8

Background { skyColor 1.0 1.0 1.0 }

Shape {

  appearance Appearance {
    texture ImageTexture {
      url "textur0.gif"
    }
  }

  geometry Box { }

}
```

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

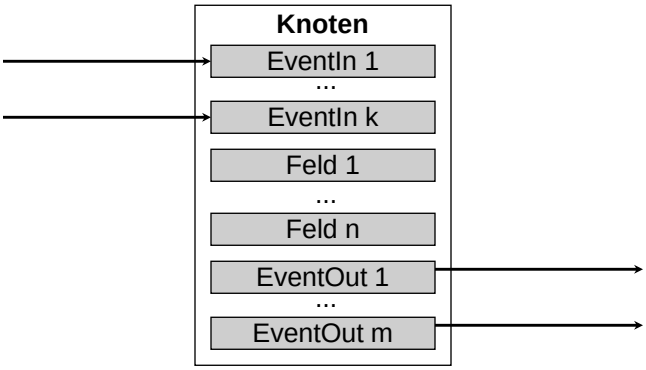
- *Dynamik* in der Darstellung von 3D-Szenen - drei mögliche Ursachen:
  - Veränderung der Betrachterposition
    - » auch in *statischen* 3D-Szenen möglich
  - Automatische, d.h. von selbst ablaufende, Veränderungen innerhalb der der 3D-Szene
    - » z.B. Bewegung oder Veränderung von Farbe/Beleuchtung
    - » *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

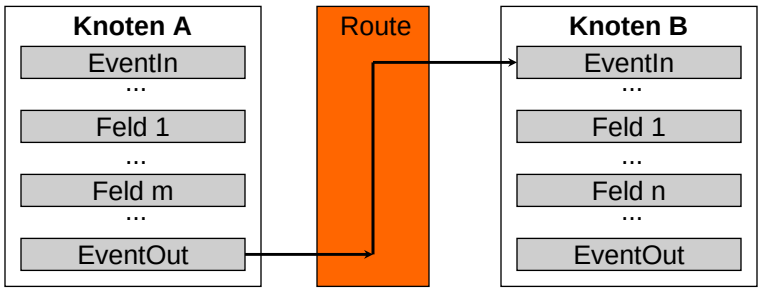
- Knoten können drei Arten von Attributen haben:
  - Felder zur statischen Festlegung von Eigenschaften
  - *EventIn*-Attribute zum Empfang von Ereignissen
  - *EventOut*-Attribute zum Senden von Ereignissen



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
}
```

- Endlosscheife mit Periode 6 Sekunden

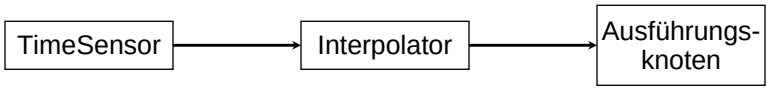
```
DEF Clock TimeSensor {
  cycleInterval 6.0
  loop TRUE
}
```

- 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 Schlüsselwerteinstellungen 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,          = π/2
    0 1.0 0 2.36,          = 3π/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
  ]
}
...
```