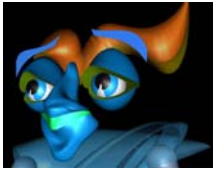
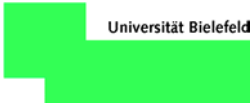


Realtime 3D Computer Graphics Virtual Reality

Marc Erich Latoschik
AI & VR Lab
Artificial Intelligence Group
University of Bielefeld



VRML and X3D

Virtual Reality Modeling Language
Extensible 3D



VRML – X3D

- X3D is a considerably more mature refined standard than VRML (taken from www.web3d.org)
- VRML compatible
 - "Classic VRML" encoding which can play most non-scripted VRML 2 worlds with only minor changes.
 - None of the technology has been lost; instead, it has evolved into X3D.
 - X3D has made a very large effort to maintain as much compatibility with VRML as possible while still solving incompatibility problems that directly lead to non-interoperability of environments between players.
- XML encoding to integrate smoothly with other applications
 - [XML](#) is fast becoming a prerequisite for including information in corporate and government data bases.
 - Having an XML encoding simply makes it easier to manage, control, validate, and exchange information. The XML encoding of X3D plays nicely in this world.
- X3D scenes and environments operate predictably between different players
 - A major problem with VRML is that it is difficult to develop VRML environments that play on all conformant browsers/players. This is due to lack of adequate specification of VRML behavior in the VRML standard. A concerted effort has been made to provide adequate specification of X3D behavior in such a way that scenes and environments can interoperate between browsers.
- X3D is componentized
 - X3D is componentized which allows for the specification of profiles tailored to a particular large market segment (e.g., CAD, Medical, Visualization). It also allows cleaner introduction of new technology as the industry develops.
- X3D authoring for any player is consistent and simpler
 - X3D Scene Authoring Interface provides consistent functionality for all scripting languages both internal and external (not true of VRML where Java and ECMAScript have widely different programming models).
 - X3D SAI solves all of this by specifying a unified set of abstract services that can then be mapped to any programming/scripting language (environments play consistently regardless of programming language).
 - Language bindings have been provided for Java and ECMAScript. This makes authoring X3D much simpler.

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

VRML – X3D

- X3D is more feature rich
 - A large number of features requested for VRML have been provided in X3D in a manner that is completely integrated into the architecture (as well as being standardized).
 - Thus, many ad hoc solutions that are vendor-specific have been avoided. You can think of X3D as "VRML3".
- X3D is continually being enhanced and updated
 - X3D is growing in functionality. [Proposed Draft Amendment 1 specification](#) adds such things as 3D textures and shading languages is available.
 - This also corrects some identified anomalies in the original X3D specification.
 - Structure of X3D makes it much easier to update on a regular basis. It is also easier to add new features that adapt to the changing graphics and commercial markets.
- X3D applications can be certified as reliable and predictable
 - An X3D conformance program is being developed by the Web3D Consortium to provide service marks for conformant X3D software. This means authoring and playback applications (browsers/players) will be reliable and predictable.
- An X3D open source conformant application is available as a developer resource
 - An open source implementation of nearly all of X3D ([X3D](#)) is available and proprietary conformant browsers such as [Flux](#) are also being developed.
 - Unlike with VRML scenes, X3D scenes will play consistently in each conformance certified player.
- X3D binary format offers encryption (i.e. security) and compression (i.e. speed)
 - A [Compressed Binary encoding](#) is under development that allows encryption for model security and very high compression (significantly more compressed than VRML's gzip) of X3D environments.
 - Scene parsing and loading speedups of 300-500% are commonplace.
 - Note that it is easy for all browsers to support all encodings since the only real difference in an implementation between them is that a different parser is required. Thus, the encodings can be intermixed in a world provided that the browser supports all of the used encodings. Current X3D browser developers plan to support all of the encodings.

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

1. VRML information

- **The VRML repository:**
<http://www.web3d.org/vrml/vrml.htm>
- **Browsers and plug-ins:** z.B. cosmo player
<http://www.cai.com/cosmo/home.htm>
- **Books (example):**
VRML 2.0 Source Book, Andrea L. Ames, David R. Nadeau and John L. Moreland, 2nd Edition, Wiley & Sons, 1996. ISBN 0-471-16507-7

VRML-models in the www

- <http://www.web3d.org/vrml/oblib.htm>
- <http://www.rdservice.de/german/produkte/VRCreator/library/vrlibrary.htm>
- <http://home.t-online.de/home/kiwano6/models.htm>
- <http://www.rccad.com/Gallery.htm>

VRML Files

ASCII Files: `<my_world>.wrl`

- **VRML header:** `#VRML V2.0 utf8`
- Comments: `#`
- **Prototypes**
- **Nodes:** Shapes, transformations, timers, interpolators, sensors, scripts, ...
- **Routes**

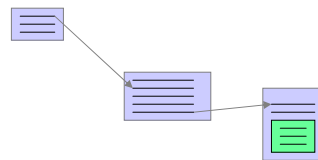
utf8 is an international character set standard. It stands for: UCS (Universal Character Set) Transformation Format, 8-bit. Encodes 24,000+ characters for many languages, ASCII is a subset

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Object description in VRML

Nodes (in an acyclic graph) with

- **Fields and Field Values**
of predefined **Field Value Types**
 - **Inputs and Outputs**
 - **Bounding Boxes**



Examples:

- Shape nodes
`Shape {appearance SFNode, geometry SFNode}`
- Geometry nodes
`Cylinder {height SFFloat, radius SFFloat}`
- Appearance nodes
`Appearance {material SSFNode, texture ...}`

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Minimal VRML-worlds

```
#VRML V2.0 utf8
```

```
# generic cylinder
```

```
Shape {
```

```
  geometry Cylinder {
```

```
  } # Cylinder
```

```
} # Shape
```

```
#VRML V2.0 utf8
```

```
# cylinder of variable  
size
```

```
Shape {
```

```
  geometry Cylinder {
```

```
    height 4.0
```

```
    radius 2.0
```

```
  } # geometry Cylinder
```

```
} # Shape
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Minimal VRML-worlds

```
#VRML V2.0 utf8
```

```
# colored cylinder of variable size
```

```
# with missing pieces
```

```
Shape {
```

```
  appearance Appearance {
```

```
    material Material {
```

```
      diffuseColor 0.5 0.5 0.0
```

```
    } # material Material
```

```
  } # appearance Appearance
```

```
  geometry Cylinder {
```

```
    height 4.0
```

```
    radius 1.5
```

```
    side TRUE
```

```
    top FALSE
```

```
    bottom TRUE
```

```
  } # geometry Cylinder
```

```
} # Shape
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

More geometry nodes

- Box {size ...}
- Cone {bottomRadius ..., height ..., ...}
- Sphere {radius ...}
- Text {string ["...", "...", ...], fontStyle ..., ...}

3. combination of objects

- Inlining of files
`Inline {url [...], ...}`
- geometric Transformationen
`Transform {
 children [...]
 translation
 Rotation
 Scale
}`

Example: sphere and cylinder

```
#VRML V2.0 utf8

# 1. Object
Inline {url "sphere.wrl"} # Inline sphere

# 2. Object
Transform {
    translation 0.0 2.0 0.0
    rotation 0.0 0.0 1.0 -0.524 # radians; 30
    degrees
    children [ Inline {url "cylinder-c.wrl"} ]
} # Transform
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Example: wooden parts (Baufix)

```
#VRML V2.0 utf8
Transform {
    translation -6.0 0.5 0.0
    children [ Inline {url "yellow-box1.wrl"} ]
} # Transform
Transform {
    translation -4.5 1.0 0.0
    children [ Inline {url "blue-box2.wrl"} ]
} # Transform
...
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

4. Reuse of models

multiple use of objects

- Inlining
- DEF und USE

```
DEF Green-Box3
  Inline {url "../blocks-
    world/green-box3.wrl"
  } # Inline cylinder
...
USE Green-Box3
```

multiple use of object
groups

- Groups

```
Group {
  children [
    ...
  ]
  ...
}
```

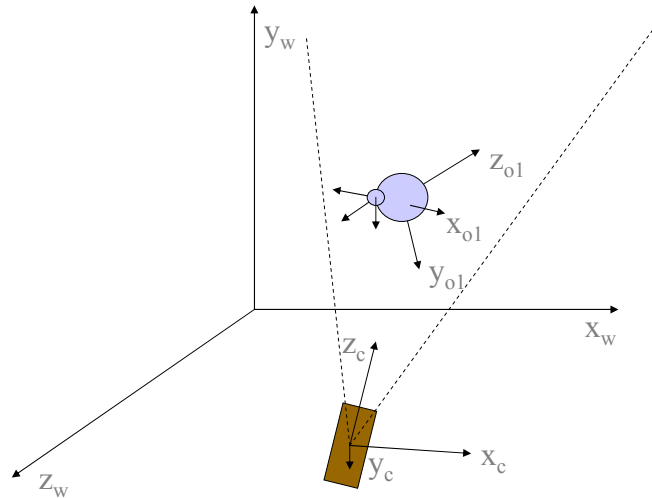
Example: replicated model

```
DEF Rep-Structure Group {
  children [
    DEF Baufix-Structure Inline {url "structure-group.wrl"} # 1.
    structure
    Transform {
    # replicated structure
      translation 0.0 0.0 -2.0
      children [USE Baufix-Structure]
    } # Transform 2. Structure
    # ... Further structures ...
  ] # children
} # group
Transform {
  translation 5.0 0.0 0.0
  children [USE Rep-Structure]
  # replicated group
}
```

Examples

CS in virtual worlds

- world CS
- object(s)
- camera(s)



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

5. Viewpoints

- Position of virtual camera with respect to the embedding CS

```
Viewpoint {  
    description "initial camera position"  
    position 0.0 0.0 25.0  
    orientation 0.0 0.0 1.0 0.0  
    fieldOfView 0.785 # 45 degrees  
}
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

avatar navigation model

- movement types

- Walk
- Fly
- Examine
- NONE

- Speed

- Size

- Headlight

- Frustum

```
NavigationInfo {  
    type "EXAMINE"  
    headlight TRUE  
    visibilityLimit  
    0.0  
    avatarSize [0.25  
    1.6 0.75]  
}
```

6. Animation in dynamic worlds

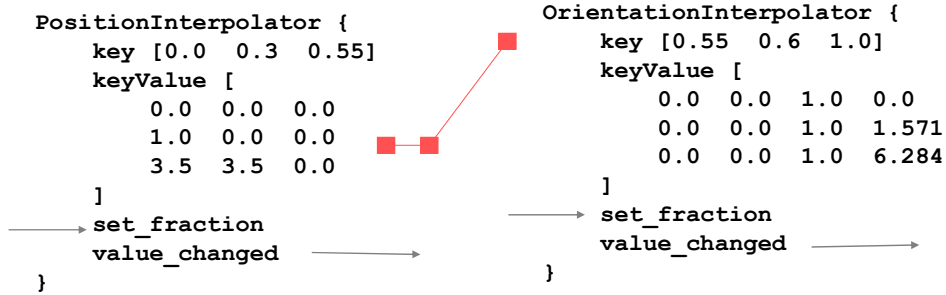
- Sensors: “Sources” of events

- TimeSensor
- Interactive sensors
- Collision detection

```
TimeSensor {  
    → enabled TRUE  
    startTime 0.0  
    stopTime 0.0  
    cycleInterval  
    1.0  
    loop TRUE  
    fraction_changed →  
    time →  
}
```

Movement Interpolationen (keyframe animation)

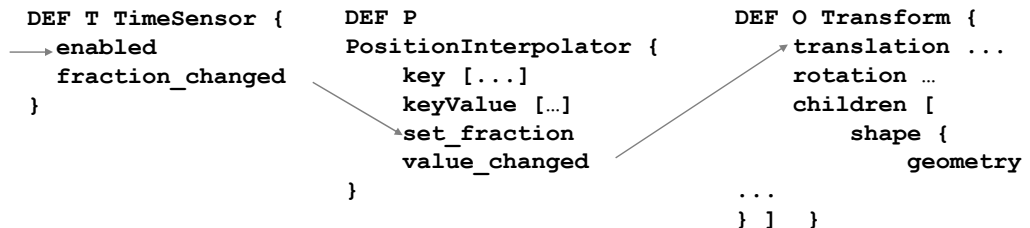
- VRML: linear interpolation between n **key frames**



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

7. Routing between fields

- Route: connection between object node fields to propagate events



ROUTE T.fraction_changed TO P.set_fraction
 ROUTE P.value_changed TO O.set_translation

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

8. Anchor points

- Connection between two virtual worlds are established by anchor points
- (e.g., dungeon worlds, game levels)

```
Anchor {  
    url ["...", ...]  
    children [...Box{}...]  
    description "..."  
    ...  
}
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Anchors in AR-applications

- “hot-spots”
 - in 3D Menus (e.g., beside the object “at the wall”) for a dynamic, time dependent help
 - in virtual models to augment specific areas with information
 - in virtual models to switch between alternative object views
 - in reality to visit alternative camera positions

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

9. Sensors

- Sensors for different interaction types:

- Pointer position (TOUCH): `isOver`, `isActive`
`DEF Touch-Green TouchSensor{`
`ROUTE Touch-Green.touchTime TO Clock.set_startTime`
- Pointer motion (MOTION)
 - CylinderSensor
 - PlaneSensor
 - SphereSensor
`DEF Spin-Cylinder SphereSensor{`
`ROUTE Spin-Cylinder.rotation_changed TO Red-Cylinder2-`
`b.set_rotation`
- If sensors are embedded, the most inner wins
- Different incrementation modes (offsets)

10. Scripts

- Connection VRML -> Java to enable more complex simulations
- Movement paths of objects (e.g., to realize physical object behaviors)
 - Intelligent object responses
 - Communication in distributed systems (e.g., games with multiple players)

Scripts

- Example: Movement interpolator

VRML:

```
DEF Mover Script {
  url "move1.js"
  eventIn SFFloat set_fraction
  eventOut SFVec3f value_changed
}
...
ROUTE Clock.fraction_changed TO Mover.set_fraction
ROUTE Mover.value_changed TO Red-Cylinder2-b.set_translation
```

Java file (move1.js):

```
// move a shape in a sinusoidal path
function set_fraction (fraction, eventTime) {
  value_changed[0] = 6.0 + Math.sin (fraction * 6.28); // x component
  value_changed[1] = -2.0; // y component
  value_changed[2] = -2 + fraction*4; // z component
}
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Universität Bielefeld

VRML

Interactive demonstration

Examples

Sensor nodes

- ... sense *changes*
- ... watch attributes
- ... react to changes
- ... trigger an control animations

Types of sensor nodes (using VRML)

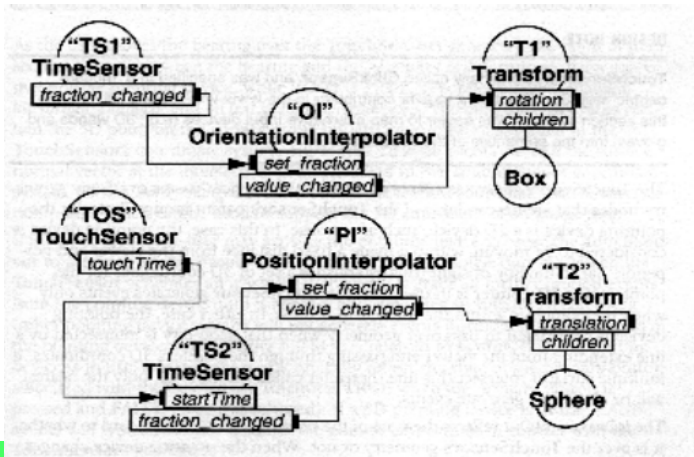
environment sensors

- Proximity Sensor
- Time Sensor
- Visibility Sensor
- *Collision*

input sensors

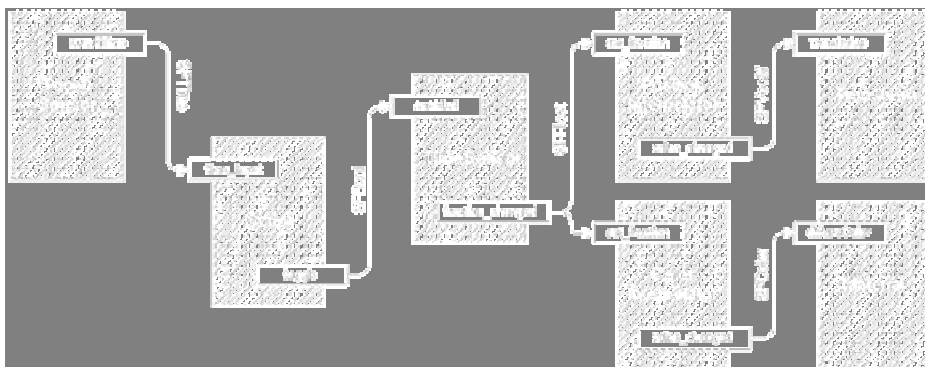
- Anchor
- Cylinder sensor
- Sphere sensor
- Touch sensor
- Plane sensor

sensor-event-routing

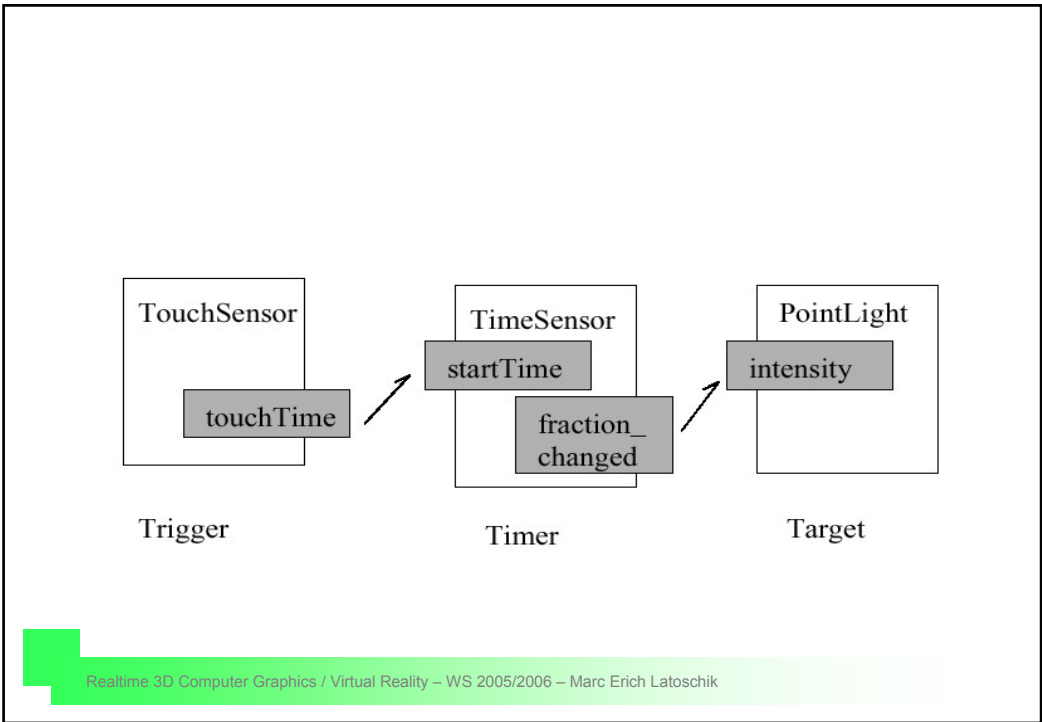
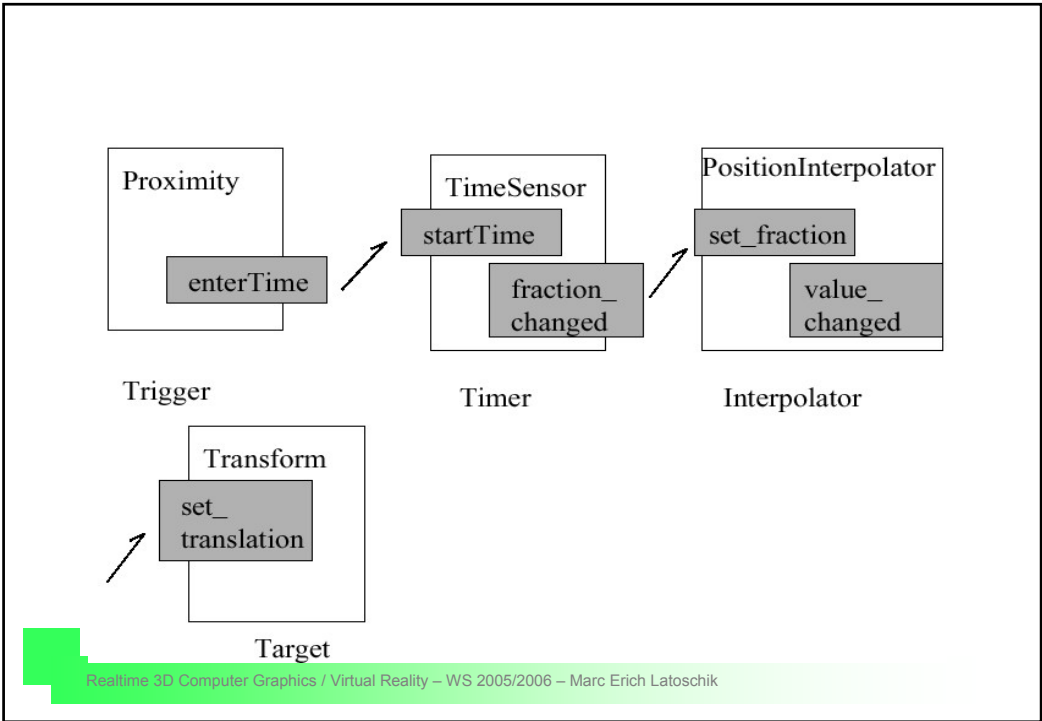


Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

sensor-event-routing



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik



time sensor

- abstract *wall clock*
- no position in the virtual scene
- generation of repeating (or unique) events
- e.g., to drive an interpolation



This concept is the basis for animations!

time sensor (VRML)

```
TimeSensor {  
  exposedField STime      cycleInterval 1  
  exposedField SBool     enabled      TRUE  
  exposedField SBool     loop        FALSE  
  exposedField STime     startTime    0  
  exposedField STime     stopTime     0  
  eventOut      STime     cycleTime  
  eventOut      SFloat   fraction_changed  
  eventOut      SBool    isActive  
  eventOut      STime    time  
}
```

visibility sensor

- activates if it comes into view (almost like an interaction sensor)
- can activate certain simulation parts to avoid unnecessary calculations

visibility sensor (VRML)

```
VisibilitySensor {  
  exposedField SFVec3f    center      0 0 0  
  exposedField SFBool     enabled    TRUE  
  exposedField SFVec3f    size       0 0 0  
  eventOut      SFTime     enterTime  
  eventOut      SFTime     exitTime  
  eventOut      SFBool     isActive  
}
```

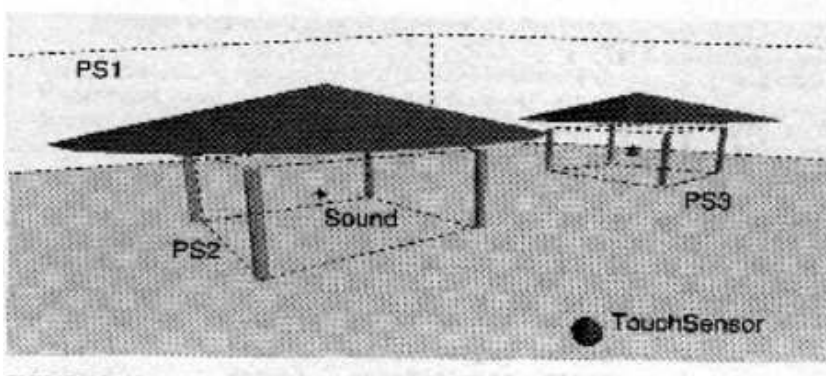
proximity sensor

- activates if the user (e.g., the view platform) reaches the sensors proximity
- is activated during the whole time something is in its proximity
- Can test for different proximity shapes (e.g., a box in VRML)

proximity sensor (VRML)

```
ProximitySensor {  
  exposedField SFVec3f center 0 0 0  
  exposedField SFVec3f size 0 0 0  
  exposedField SFBool enabled TRUE  
  eventOut SFBool isActive  
  eventOut SFVec3f position_changed  
  eventOut SFRotation orientation_changed  
  eventOut SFTime enterTime  
  eventOut SFTime exitTime  
}
```

proximity sensor (Box, VRML)



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

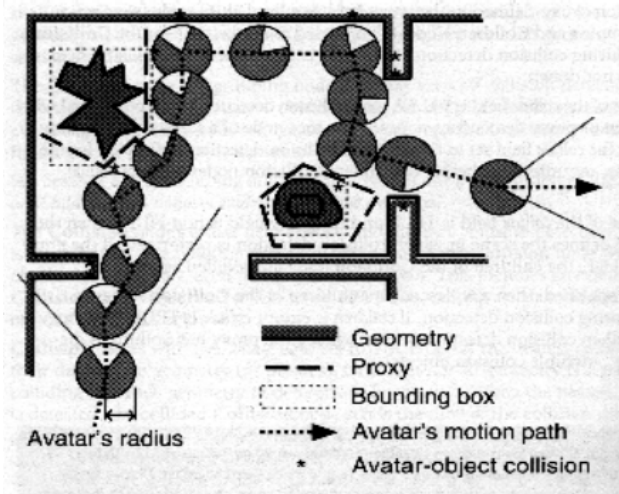
collision (VRML)

- tests for collision with the user (the view platform)
- is implemented as a group node
- has additional alternative test geometry (e.g., to speed up calculation)

↓ How would you implement an arbitrary collision engine?

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

collision (VRML)



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

collision (VRML)

```
Collision {
  eventIn      MFNode      addChildren
  eventIn      MFNode      removeChildren
  exposedField MFNode      children      []
  exposedField SFBool     collide        TRUE
  field        SFVec3f     bboxCenter    0 0 0
  field        SFVec3f     bboxSize      -1 -1 -1
  field        SFNode      proxy         NULL
  eventOut     SFTIME      collideTime
}
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

interaction sensors

- map data from input devices to scene changes
- input devices: mouse, trackball, stylus, gloves or gesture events
- detects **isOver** and **isActive** as events

Touch Sensor (VRML)

- Detektiert Mausinteraktion mit *benachbarter* Geometrie
- Liefert Touchtime (Zeit zwischen button down und button up)
- Liefert Normale, Punkt und Texturkoordinate am Auftreffpunkt (während **isOver**)

Touch Sensor (VRML)

```
TouchSensor {  
  exposedField SFBool    enabled          TRUE  
  eventOut    SFVec3f    hitNormal_changed  
  eventOut    SFVec3f    hitPoint_changed  
  eventOut    SFVec2f    hitTexCoord_changed  
  eventOut    SFBool     isActive  
  eventOut    SFBool     isOver  
  eventOut    SFTIME     touchTime  
}
```

↓ Frage: Wie ist das Touch Sensor Konzept auf immersive VR Anwendungen zu erweitern?

Drag Sensoren

- Spezielle Interaktionssensoren
- *Mappen* Pointing-Device
Bewegungsinformationen auf
Objektänderungen
- Siehe dazu z.B. auch die OpenInventor
Klassenbibliothek

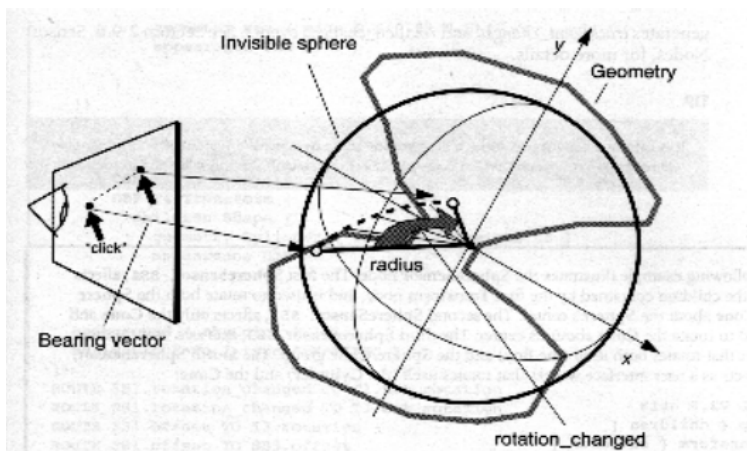
Sphere Sensor (VRML)

- Rotiert benachbarte Geometrie um den Nullpunkt des Sensors

```
SphereSensor {  
  exposedField SFBool    autoOffset    TRUE  
  exposedField SFBool    enabled      TRUE  
  exposedField SFRotation offset      0 1 0 0  
  eventOut     SFBool    isActive  
  eventOut     SFRotation rotation_changed  
  eventOut     SFVec3f   trackPoint_changed  
}
```

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Sphere Sensor (VRML)



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

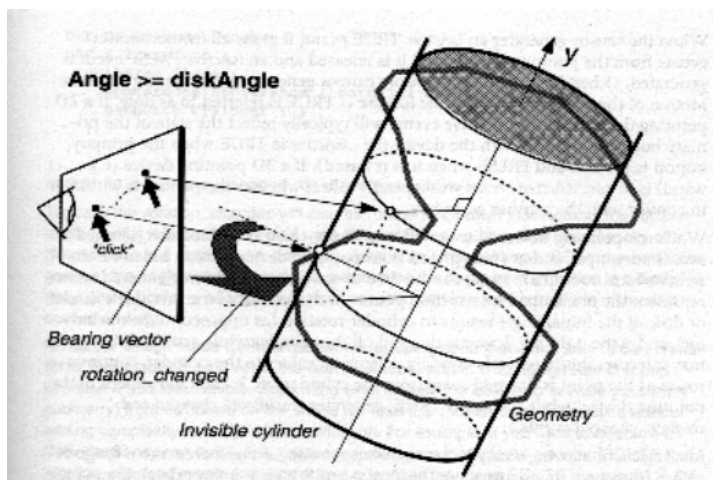
Cylinder Sensor (VRML)

- Rotiert benachbarte Geometrie in zwei Arten um die y-Achse des Sensors

```
CylinderSensor {  
  exposedField SFBool    autoOffset    TRUE  
  exposedField SFFloat   diskAngle    0.262  
  exposedField SFBool    enabled      TRUE  
  exposedField SFFloat   maxAngle     -1  
  exposedField SFFloat   minAngle     0  
  exposedField SFFloat   offset       0  
  eventOut     SFBool    isActive  
  eventOut     SFRotation rotation_changed  
  eventOut     SFVec3f   trackPoint_changed  
}
```

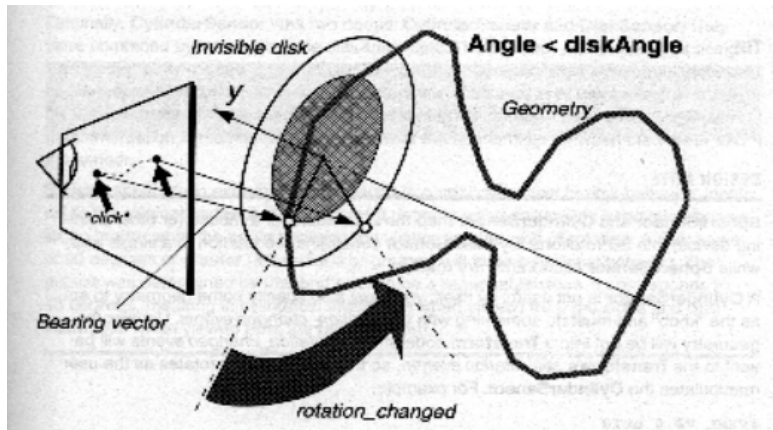
Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Cylinder Sensor (VRML)



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Cylinder Sensor (VRML)



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Plane Sensor (VRML)

- Translatiert in Sensor-lokaler xy-Ebene

```
PlaneSensor {
  exposedField SFBool    autoOffset    TRUE
  exposedField SFBool    enabled       TRUE
  exposedField SFVec2f   maxPosition   -1 -1
  exposedField SFVec2f   minPosition   0 0
  exposedField SFVec3f   offset        0 0 0
  eventOut     SFBool    isActive
  eventOut     SFVec3f   trackPoint_changed
  eventOut     SFVec3f   translation_changed
}
```

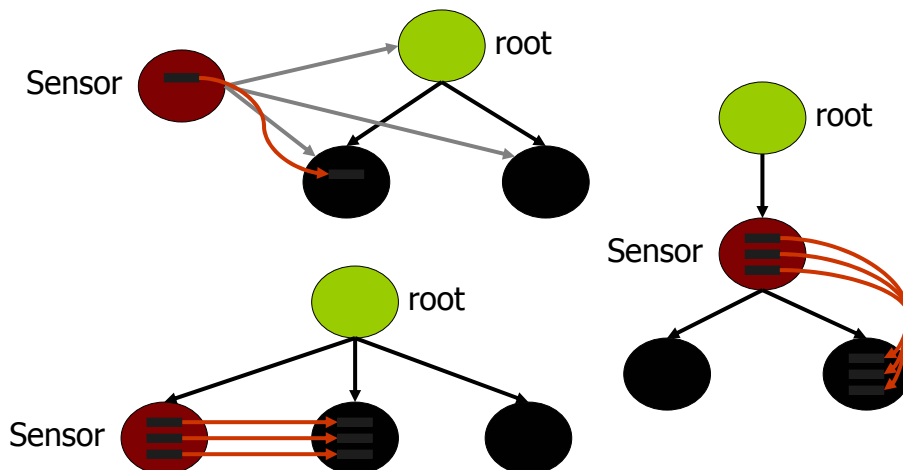
Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Designüberlegungen

- Übertragbarkeit auf andere VR-Anwendungen bzw. VR-Designsysteme
- Homogene Einbettung in existierende Datenstrukturen (Szenengraph)
- Abstraktion von aktuell gegebenen Datenlieferanten; siehe dazu den Transfer von Maus zu Glove etc.
- Performanz im Hinblick auf ständige Triggerung

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Prinzipielle Sensorenlage im Szenengraph



Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Prinzipielle Sensorenlage im Szenengraph

- Wird ein Knotenzugriff auf die beeinflussten Knoten benötigt?
- Wenn ja, Knotenzugriff per Traversierungsfunktionen (get-children, get-parent ...) oder per registrierter Referenz?
- Ist eine Position des Sensors von funktionaler Bedeutung (siehe Annäherung vs. Kollisionsengine)?

Datenaustausch zwischen den Komponenten (Field vs. Referenz)

- Fieldconnections durch das Scripting steuerbar aber bei großem Datenaustausch evt. langsam
- Funktion durch Knotenreferenz ist performant aber weniger flexibel und setzt nicht auf die eventuelle Kommunikationsfähigkeiten der Fields

Aufgaben

1. Entwickle zwei verschiedene Proximitysensoren, einen mit Box- und einen mit Sphereumgebung. Wie könnte man eine Staffelung verschiedener Umgebungen in einer Instanz umsetzen? -> Wer kann setzt dieses um!
2. Entwickle einen Touchsensor für mögliche Pointing-Devices.
3. Kopple den Touchsensor an einen Interpolator, so dass nach `touch` eine Interpolationsanimation ausgeführt wird.
4. Entwerfe ein eigenes Architekturkonzept für eine Kollisionsengine.
5. Stelle die Vor- und Nachteile unterschiedlicher Grapheinbettungen zusammen.

Realtime 3D Computer Graphics / Virtual Reality – WS 2005/2006 – Marc Erich Latoschik

Universität Bielefeld

Animation durch Interpolation

Animation durch Interpolation

- „Animation verstehen wir als die Veränderung eines Attributs (einer Eigenschaft) im Zeitverlauf.“
- Eine Animation benötigt zwei Elemente:
 - 1. Einen Zeitgeber, um die Wiedergabe der Animation zu steuern
 - 2. Eine Beschreibung der Form der Veränderung während der Animation

Aufgaben

- Für die folgenden Aufgaben kann der bereits entwickelte Viewer benutzt werden. Alternativ dazu kann ebenfalls der aview (`$(AV_HOME)/bin/aview.sh`) verwendet werden.
- 1. Lade ein beliebiges Objekt und rotiere es unter Einsatz der Interpolatoren um eine Achse deiner Wahl.
- 2. Lade ein Objekt mit impliziten Ausrichtungen (vorne, oben, etc., also ein Auto oder eine humanoide Figur...) und positioniere es so über einem ebenfalls zu ladenden Boden (`floor.iv`) der virtuellen Welt, dass es realistisch Bodenkontakt erhält. Interpoliere die Position des Objekts in Form eines Vierecks.
- 3. s.2. aber lade nun ebenfalls ein statisches Hindernis (Baum, Tisch...) und umfahre das Hindernis durch Interpolation der Position und der Orientierung. Dabei soll „vorne“ immer der Bewegungsrichtung entsprechen.
- 4. Zusatz: Teste beliebige andere Interpolatoren (für Skalierung, Farbe, etc.).



and that's it...have fun...