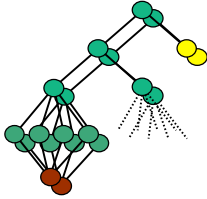




Computer-Graphik II

Scenegraphs / VRML

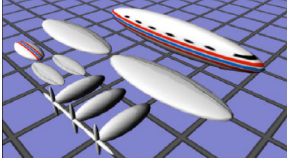
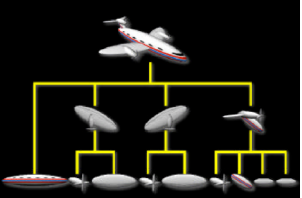


G. Zachmann
 Clausthal University, Germany
cg.in.tu-clausthal.de

Motivation

- Immediate mode vs. retained mode:
 - Immediate mode = OpenGL / Direc3D = App. schickt Pgone / State-Befehle an die Grafik = flexibler
 - Retained mode = Scenegraph = App. legt vordefinierte Datenstrukturen an, die Pgone und States speichern = bequemer und evtl. effizienter
- Flach vs. hierarchische Datenstrukturen:



- Code re-use und Know-how re-use!
- Descriptive, not imperative (cv. C vs. Prolog)
 - Thinking objects ... not rendering processes

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 2

Struktur eines Szenegraphen

- Gerichteter, azyklischer Graph, i.A. ein echter Baum
- Heterogene Knoten
- Beispiel:

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 3

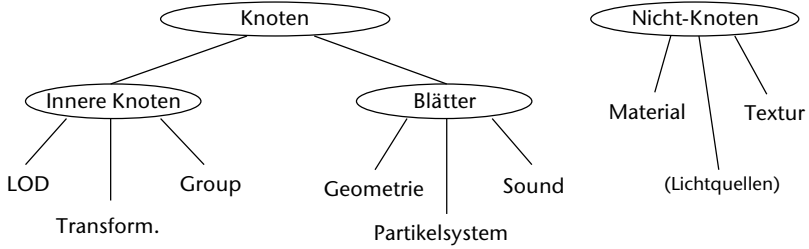
Semantik

- Semantik der Knoten:
 - Wurzel = "Universum"
 - Blätter = "*content*" (Geometrie, Sound, ...)
 - Innere Knoten = Gruppierung, State(-änderungen), und nicht-geometrische Funktionalität (z.B. Transf.)
- Gruppierung: nach welchen Kriterien? Bleibt Applikation überlassen:
 - Geometrische Nähe? (Scenegraph induziert BV-Hierarchie!)
 - Nach Material? (state changes kosten Performance!)
 - Nach log. Bedeutung? (alle Wasserleitungen, alle Kabel, alle Sitze, ...)
- Semantik der Kanten = Vererbung des "State"
 - Transformation
 - Material
 - Lichtquellen

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 4

Knotenarten

- 2 Hierarchien: Szenegraph-Hierarchie + Klassenhierarchie
- Die Mächtigkeit und Flexibilität eines Szenegraphen hängt von der Menge der zur Verfügung stehenden Knotenklassen ab!
- Etliche Klassen sind nicht Teil des Szenegraphen, aber doch Teil der Szene



```

graph TD
    K((Knoten)) --- IK((Innere Knoten))
    K --- B((Blätter))
    IK --- LOD[LOD]
    IK --- T[Transform.]
    IK --- G[Group]
    B --- Geometrie[Geometrie]
    B --- PS[Partikelsystem]
    B --- Sound[Sound]
    NK((Nicht-Knoten)) --- M[Material]
    NK --- L["(Lichtquellen)"]
    NK --- T2[Textur]
  
```

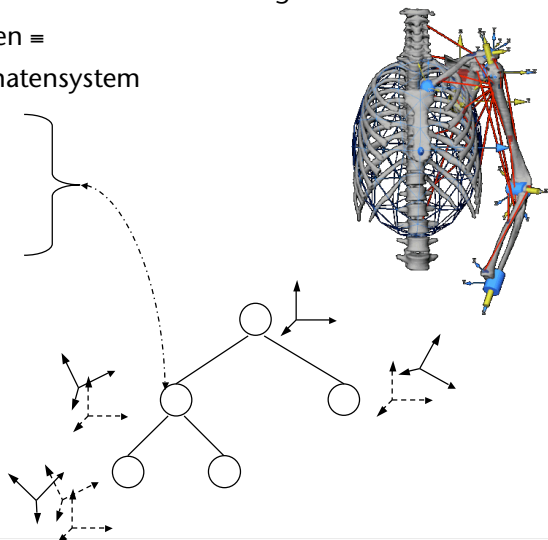
G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 5

Transformationssemantik

- Alle Scenegraphs behandeln diese Semantik gleich
- Transformationsknoten ≡ neues lokales Koordinatensystem
- ```

pushMatrix();
multMatrix(...);
traverse sub-tree
popMatrix();

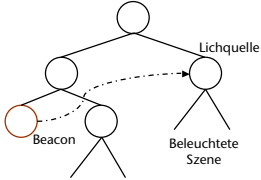
```



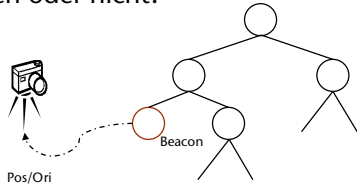
G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 6

## Issues

- **Lichtquellen:**
  - Teil des Szenengraphen (meistens)
  - Semantik (OpenSG):
    - beleuchtet Teilbaum darunter
    - Pos./Ori kommt von **Beacon**
  - Je nach Art (directional, point, spot) wird verschiedener Anteil der Transformation verwendet



- **Kamera: Knoten im Szenengraphen oder nicht?**  
(gibt beide Varianten)
  - Ja: Kamera ist ein Knotentyp
  - Nein: Beacon-Konzept




G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 7

## Material

- Property eines Knotens
- Vererbung: top-down
  - Pfad von Wurzel zu Blatt muß mindestens 1 Material haben
  - Folge:
    - Jedes Blatt wird mit eindeutig definiertem Material gerendert
    - Dieses läßt sich leicht bestimmen
- Schlechte Idee (Inventor): Vererbung left-to-right!

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 8

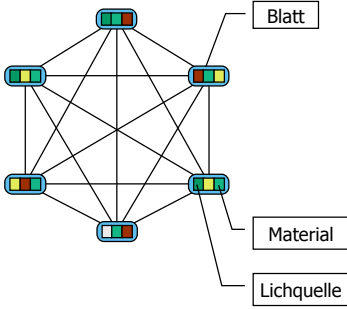
## State Sorting

- State = Zustand
  - Zusammenfassung aller Attribute
  - Attribut-Bsp.e: Farbe, Material, Lighting-Param., Textur, Blend-Fkt., Shader-Programm, etc.
  - Jedes Attribut hat zu jedem Zeitpunkt genau 1 Wert aus einer endlichen Menge
- State-Wechsel sind einer der Performance-Killer
- Kosten: 
  - Matrix-Stack-Modifikation
  - Lighting-Modifikation
  - Textur-Modifikation
  - Shader-Programm-Modifikation
- Ziel: kompletten Szenengraphen mit minimaler Anzahl State-Wechsel rendern
- "Lösung": Pre-Sorting

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 9

## State Sorting

- Problem: die optimale Lösung ist NP-vollständig
- Denn:
  - Jedes Blatt ist ein Knoten in einem vollständigen Graphen,
  - Kosten einer Kante = Kosten des/der State-Wechsels (verschiedene State-Wechsel kosten verschieden viel, z.B. sind neue Trafos relativ billig),
  - Gesucht: kürzester Weg  
→ *Traveling Salesman Problem*
- Weiteres Problem: klappt nicht bei dynamischen Szenen und Occlusion Culling



G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 10

## Der Sorting-Buffer

- Idee & Abstraktion:
  - Betrachte nur ein Attribut ("Farbe")
  - Schalte Buffer zwischen App. und Graphikkarte
  - Buffer enthält Elemente mit verschiedenen Farben
  - Pro Schritt eine von drei Operationen:
    - Element direkt an Graphikkarte weiterreichen
    - Element lesen und in Buffer schreiben
    - Teilmenge aus Buffer löschen und an Graphikkarte schicken

Polygonsequenz      Sortierpuffer      Graphik-Hardware

G. Zachmann    Computer-Graphik 2 - SS 07
Scenegraphs    11

- Zwei Algorithmen-Klassen:
  - **"Online"-Algorithmen:** Algo kennt *nicht* zukünftige Elemente!
  - **"Offline"-Algorithmen:** Algo kennt *alle* Elemente, muß aber trotzdem Buffer verwenden
- Betrachte nur "lazy" online-Strategie:
  - Elemente werden nur bei Buffer-Overflow aus Buffer entfernt
  - Jede nicht-lazy Online-Strategie läßt sich in eine lazy Strategie mit gleichen Kosten umwandeln
- Frage: welche Elemente muß man bei Buffer-Overflow auswählen, damit minimale Anzahl Farbwechsel auftritt?

G. Zachmann    Computer-Graphik 2 - SS 07
Scenegraphs    12

## Competitive Analysis


- Definition *c-competitive* :
  - Sei  $C_{\text{off}}(k)$  die Kosten (Anzahl Farbwechsel) der optimalen Offline-Strategie,  $k = \text{Buffer-Größe}$ .
  - Sei  $C_{\text{on}}(k)$  die Kosten der Online-Strategie.
  - Dann heißt diese Strategie "*c-competitive*" gdw.
 
$$C_{\text{on}}(k) = c \cdot C_{\text{off}}(k) + a$$
 wobei  $a$  von  $k$  unabhängig ist.
- Gesucht: Online-Strategie mit möglichst kleinem  $c$  (im worst-case, und – wichtiger noch – im average case)

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 13

## Beispiel: LRU (least-recently used)

- Strategie:
  - Pro Farbe ein Timestamp (**nicht pro Element!**)
  - Element wird in Buffer geschrieben  $\rightarrow$  Timestamp seiner Farbe wird auf aktuelle Zeit gesetzt
    - Achtung: dabei können die Timestamps andere Elemente im Buffer auch aktualisiert werden
  - Buffer-Overflow: entferne Elemente, deren Farbe ältesten Timestamp hat
- Untere Schranke für die Competitive-Ratio:  $\Omega(\sqrt{k})$
- Beweis durch Beispiel:
  - Setze  $m = \sqrt{k - 1}$ , oBdA  $m$  gerade
  - Wähle die Eingabe  $(c_1 \dots c_m x^k c_1 \dots c_m y^k)^{m/2}$
  - Kosten der **Online**-LRU-Strategie:  $(m + 1) \cdot 2 \cdot \frac{m}{2} > k$  Farbwechsel
  - Kosten der **Offline**-Strategie: Ausgabe ist  $(x^k y^k)^{m/2} c_1^m \dots c_m^m$   
 $\rightarrow 2m$  Farbwechsel


G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 14



## Bounded-Waste- & Random-Choice-Strategie

- Idee: zähle Platzbedarf jeder Farbe im Buffer über die gesamte bisherige Zeit aufsummiert
- Führe Waste-Zähler  $W(c)$  ein:
  - Bei Farbwechsel: erhöhe  $W(c)$  um Anzahl Elemente im Buffer mit Farbe  $c$
- Bounded-Waste-Strategie:
  - Bei Buffer-Overflow entferne alle Elemente mit Farbe  $c'$ , mit  $W(c')$  maximal
- Competitive Ratio (o.Bew.):  $O(\log^2 k)$
- Random-Choice-Strategie:
  - Randomisierte Version von BW
  - Wähle uniform zufälliges Element aus Buffer, entferne alle Elemente mit derselben Farbe
  - Damit: häufige Farbe wird häufiger ausgewählt, über die Zeit ergibt sich gerade  $W(c)$

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 15



## Round-Robin-Strategie

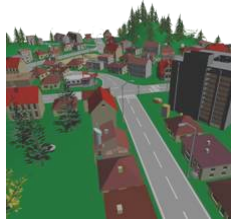
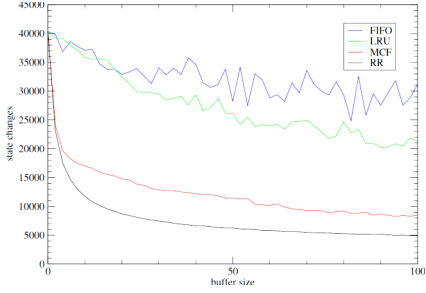
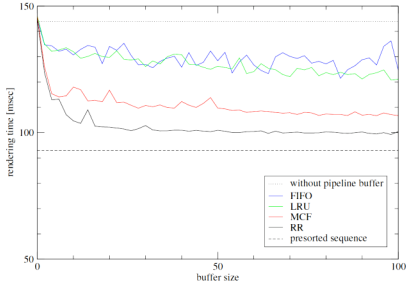
- Problem: Generierung guter Zufallszahlen dauert rel. lange
- RR-Strategie:
  - Variante von Random-Choice
  - Wähle nicht zufälligen Slot im Buffer, sondern den gemäß eines Pointers
  - Pointer wird in Round-Robin-Manier weitergeschaltet

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 16



## Vergleich

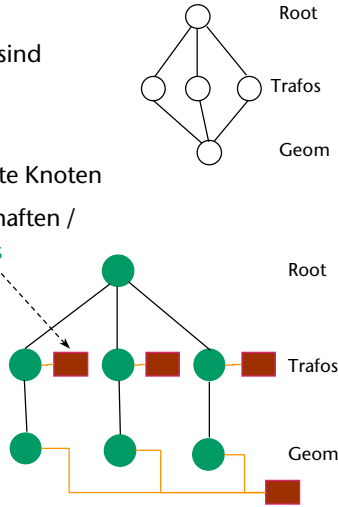
- Fazit:
  - Round-Robin ist sehr gut (obwohl sehr einfach)
  - Worst-Case sagt sehr wenig über prakt. Perf.

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 17

## Sharing von Geometrie

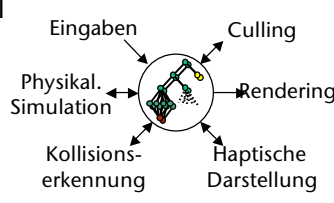
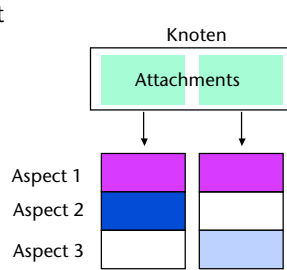
- Problem: große Szenen mit viel identischer Geometrie
- Idee: DAG (statt Baum)
  - Problem: Zeiger/Namen von Knoten sind **nicht mehr eindeutig!**
- Lösung: trenne Struktur von Inhalt
  - Baum besteht nur noch aus **einer** Sorte Knoten
  - Knoten bekommen **spezielle** Eigenschaften / Inhalt durch **Attachments / Properties**
  - Vorteile
    - alles wird share-bar
    - Viele Szenengraphen zur selben Szene möglich
    - Ein Knoten kann viele Attachments (= Eigenschaften) bekommen



G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 18

## Multi-threading / Thread-safety

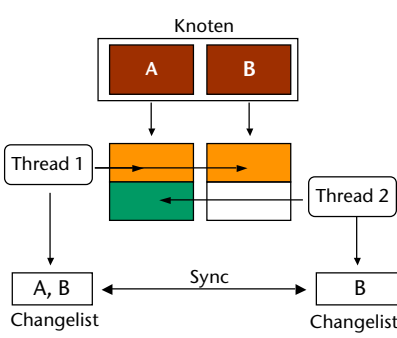
- Performer: App / Cull / Draw – Modell
- Idee: mehrere Szenengraphen
- Problem: Speicheraufwand
- Lösung:
  - "Aspects" und *Copy-on-Write* der Attachments
  - Jeder Thread "sieht" einen eigenen Aspect
  - Problem: einfacher Zugriff über Pointers  
`geom->vertex[0]`  
geht nicht mehr
  - Lösung:
    - Smart Pointers
    - Pro Klasse eine Pointerklasse. Bsp.:  
`geomptr = Geometry::create(...);`  
`geomptr->vertex[0] ...`

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 19

## Synchronisation: Changelists

- Synchronisation: **Changelists**



- Distributed Rendering:
  - Wunsch: Rendern auf einem Cluster
  - Problem: Änderung des Szenengraphen propagieren
  - Lösung: Changelists übertragen
    - Enthalten IDs von geänderten Knoten / Properties
    - Werden bei Update übertragen

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 20

## Erweiterbarkeit

- Wunsch:
  - Neue Knoten als SOs/DLLs
  - Soll auch der Loader verstehen können (ohne Neu-Compilieren)
  - Alle Traversierungen sollen funktionieren
- Lösung:
  - Design Patterns (Factory, Visitor, ...)

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 21

## Anwendungskriterien für Szenegraphen

- Wann soll man Scenegraphs verwenden:
  - Komplexe Szenen: viele verschiedene Materialien, viel Geometrie, oft ist nur ein Teil zu sehen, komplexe Transformationshierarchien
  - Relativ statische Geometrie
  - Spezifische Features, die ein Scenegraph bietet (Partikel, Clustering, ...)
- Wann man einen Scenegraph **nicht** verwenden soll:
  - Einfache Szenen (ein Objekt in der Mitte)
  - Hochgradig dynamische Geometrie

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 22

## Einige (ehem.) populäre Scenegraphs

- SGI Performer (<http://www.sgi.com/software/performer/>)
- Java3D (<http://java.sun.com/products/java-media/3D/>)
- Inventor/Coin (<http://oss.sgi.com/projects/inventor/> ,  
<http://www.coin3d.org/>)
- VRML (<http://www.vrml.org/>)
- OpenSG (<http://www.opensg.org/>) !
- Viele andere (siehe [www.sf.net](http://www.sf.net) , "Game Engines List" , ...)

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 23

## Geschichte

- Zusammen mit OO Programming
- Davor eher "flache" Strukturen (GKS, Starbase)
- Inzwischen nur noch open-source Scenegraphs und Game-Engines

| Scenegraph    | Start Year | End Year |
|---------------|------------|----------|
| Open Inventor | 1992       | 1995     |
| Performer     | 1992       | 1995     |
| Y             | 1994       | 1997     |
| Cosmo3d       | 1995       | 1997     |
| Optimizer     | 1996       | 1997     |
| DirectModel   | 1996       | 1997     |
| OpenGL++      | 1997       | 1998     |
| Java3d        | 1997       | 1999     |
| Fahrenheit    | 1998       | 1999     |
| OpenSG        | 1999       | 2005     |

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 24

## VRML

- Was ist VRML?
  - Scenegraph & File-Format, plus ...
  - Multimedia-Support
  - Hyperlinks
  - Verhalten und Animationen
- Achtung: VRML  $\neq$  VR
- Varianten:
  - VRML 1.0 (1995) (= Inventor, also kein VRML)
  - VRML 2.0 (1996)
  - VRML97 (1997) – ISO Standard, praktisch identisch zu VRML2

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 25

## Knoten und Felder

- Knoten:
  - Die üblichen (Verdächtigen):
    - Geometry, Transform, Group, Lights, LODs, ...
  - Sensors – stellen Kontakt zwischen GUI und Szene her
    - TouchSensor, CollisionSensor,
  - Interpolators – für Animationen
  - Skript-Knoten (Javascript, Java, ...)
- Felder:
  - Knoten = Menge von Feldern
  - "Single Fields" (SF...) und "Multi Fields" (MF...)
    - Bsp.: MFNode, SFVec3f, MFVec3f, ...
- Feldarten:
  - field = Daten im File
  - eventIn, eventOut = s.u., werden nicht gespeichert
  - exposedField = Kombination der drei (xxx, set\_xxx, xxx\_changed)

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 26

Spec eines Knotens legt Felder fest

- Bsp.: Cone
 

```
Cone {
 field SFFloat bottomRadius 1
 field SFFloat height 2
 field SFBool side TRUE
 field SFBool bottom TRUE
}
```

fieldart    Feldtyp    Feldname    Default
- Verwendung:
 

```
DEF Kegel Shape {
 geometry Cone { bottomRadius 2 height 1 }
 appearance Appearance {
 material Material { diffuseColor 1 1 1 }
 }
}
```

Name des Knotens    Name eines Feldes eines Shape-Knotens    Wert des Feldes geometry

G. Zachmann    Computer-Graphik 2 - SS 07    Scenegraphs    27

### Authoring, Routing, Events

- In VRML:
  - actions & objects = Knoten im selben Scenegraph
  - Events sind flüchtige "Ereignisse", haben keine zugreifbare Repr.
  - Ist "Middleware" für Implementierung von Anwendungen!

G. Zachmann    Computer-Graphik 2 - SS 07    Scenegraphs    28

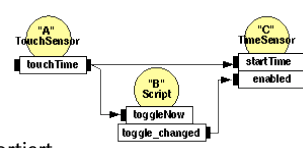
## Execution Model

- **Route:**
  - Verbindung zwischen zwei Knoten (von eventOut nach eventIn)
    - ROUTE Sender.field\_changed TO Receiver.set\_field
  - Fan-in, fan-out
  - Routes ergeben Route-Graph, impliziert Abhängigkeit der Knoten
- **Definition Event:**
  - Wird generiert durch Änderung eines eventOut Feldes
  - Datenwert wird kopiert von eventOut nach eventIn
  - Timestamp
- **Event Cascade:**
  - Initialer Event (von Script, Sensor, oder Timer)
  - Propagieren an alle angeschlossenen eventIn's
  - Knoten können als Folge weitere Events generieren über eventOut's
  - Alle Teil derselben Cascade, alle denselben Timestamp

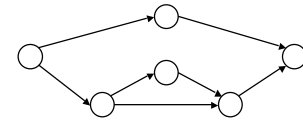
G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 29

## Events induzieren eine Abhängigkeit der Knoten:

- Propagiere in der "richtigen" Reihenfolge
- Algo:
  - Breadth-first traversal
  - Aktuelle Front wird gemäß Abhängigkeiten sortiert



- **Zyklen:**
  - Sind erlaubt (manchmal sogar sinnvoll)
  - Abbruchregel bei Propagation:
    - Keine zwei Events vom selben eventOut oder zum selben eventIn mit demselben Timestamp



G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 30

### Bsp. für Routes/Events

```

DEF Frame1 Transform {
 translation 0.0 0.0 -0.5
 children [Shape { ... }]
}
DEF Frame2 Transform {
 translation 0.0 0.0 +0.5
 children [Shape { ... }]
}

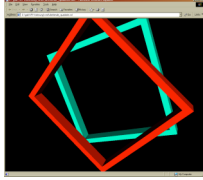
DEF Rot1 OrientationInterpolator {
 key [0.0, 0.5, 1.0]
 keyValue [0.0 0.0 1.0 0.0, 0.0 0.0 1.0 3.14, 0.0 0.0 1.0 6.28]
}
DEF Rot2 OrientationInterpolator {
 key [0.0, 0.5, 1.0]
 keyValue [0.0 0.0 1.0 0.0, 0.0 0.0 1.0 3.14, 0.0 0.0 1.0 6.28]
}

DEF Timer1 TimeSensor { cycleInterval 10.0 loop TRUE startTime -1 }
DEF Timer2 TimeSensor { cycleInterval 11.0 loop TRUE startTime -1 }

ROUTE Timer1.fraction_changed TO Rot1.set_fraction
ROUTE Timer2.fraction_changed TO Rot2.set_fraction
ROUTE Rot1.value_changed TO Frame1.set_rotation
ROUTE Rot2.value_changed TO Frame2.set_rotation

```

Scenegraph



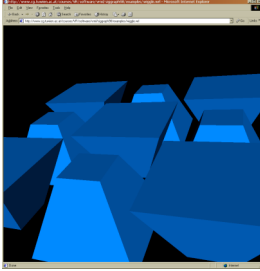
Keyframe Animation (später)

Routes

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 31

### Animationen

- Komponenten:
  - Timer (**TimeSensor**)
  - Interpolatoren (interpoliert zwischen Keyframes)
  - Routes: Timer → Interpolator → graph. Attribut
- Interpolatoren:
  - PositionInterpolator
  - OrientationInterpolator
  - ColorInterpolator
  - ScalarInterpolator
  - CoordinateInterpolator
    - NormalInterpolator
- Bsp.



G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 32



## Prototypes

- Definition neuer Knotenarten; faßt zusammen:
  - Knoten (Shapes, Sensors, Interpolators, etc.)
  - Script-Knoten
  - Routes
- Bsp:
 

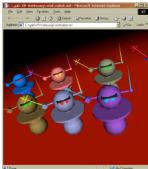
```
PROTO Robot
[field SFCOLOR eyeColor 1.0 0.0 0.0
 ...
]{
 Shape { appearance Appearance {
 material Material {
 diffuseColor IS eyeColor
 }
 }
 ...
}
```

} Interface (Felder & Events)

} Body (Implementierung)

Zuweisung des Interfaces

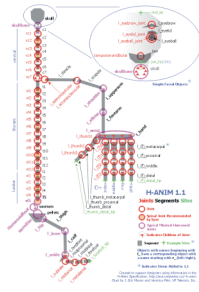
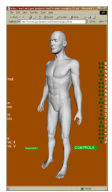
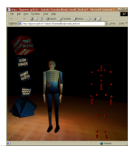
Name des neuen Knoten



G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 33

## Weiterentwicklungen

- X3D:
  - VRML mit Hilfe von XML spezifiziert (plus kleine Verbesserungen)
- H|Anim:
  - Standard zur Spezifizierung von "humanoiden" Figuren in VRML97
  - Joints = Baumstruktur
  - Segments = Geometrie
  - Sites = "Handles" (ausgez. Punkte im Koord.system des Joints)

```

graph TD
 Joint1((Joint)) --- Joint2((Joint))
 Joint1 --- Joint3((Joint))
 Joint1 --- Ellipse((Segment))
 Joint1 --- Sites((Sites))

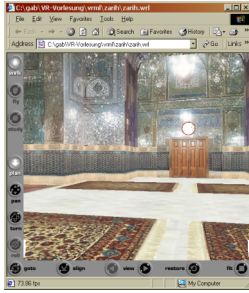
```

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 34

## Beispiele



VRMLCity  
(each time different)



Zarih



3D Chats?




Edutainment

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 35


## Support

- Verbreitung:
  - Als brauchbares, nicht-proprietäres Geometrie-Format (Inventor war zu ungeschickt)
  - Für "interaktive" Animationen
  - Nicht zum Geld machen (sogar blaxxun ist insolvent, März 2002)
  - VR?
- Browsers:
  - Blaxxun: <http://developer.blaxxun.com> , Plugin, nur Windows
  - Cortona: <http://www.parallelgraphics.com/products/cortona/> (dito)
  - OpenVRML: <http://openvrml.org> , cross-platform, noch recht unreif
  - FreeWRL: [www.crc.ca/en/html/FreeWRL/home/home](http://www.crc.ca/en/html/FreeWRL/home/home) , Unix, VRML & X3D

G. Zachmann Computer-Graphik 2 - SS 07 Scenegraphs 36



## Literatur, References



- Sites:
  - <http://home.hiwaay.net/~crispen/vrmlworks/>
  - [www.vrml.org](http://www.vrml.org)
- Bücher:
  - Andrea L. Ames, David R. Nadeau, and John L. Moreland:  
*The VRML 2.0 Sourcebook*. John Wiley & Sons, 1996.
  - Hartman, Jed, and Wernecke:  
*The VRML 2.0 Handbook*. Addison-Wesley, 1996.

G. Zachmann Computer-Graphik 2 - SS 07

Scenegraphs 37