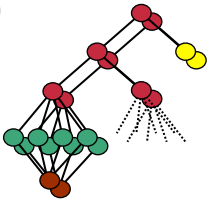
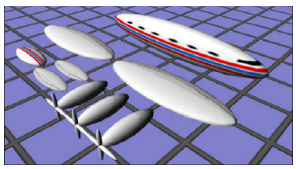
Virtuelle Realität ...
Scenegraphs
(Game Engines)

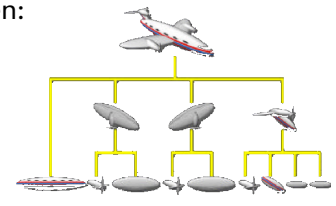


G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

Motivation

- Immediate mode vs. retained mode:
  - Immediate mode = OpenGL / Direc3D = Applikation schickt Polygone / 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:

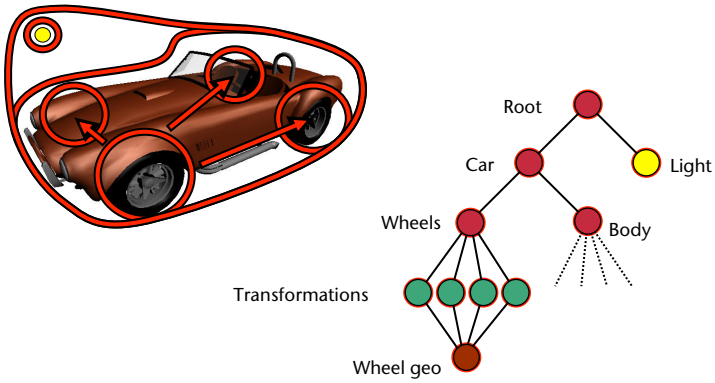




- *Code re-use* und *Know-how re-use*!
- Descriptive, not imperative (cv. C vs. Prolog)
  - Thinking objects ... not rendering processes

Struktur eines Szenegraphen

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

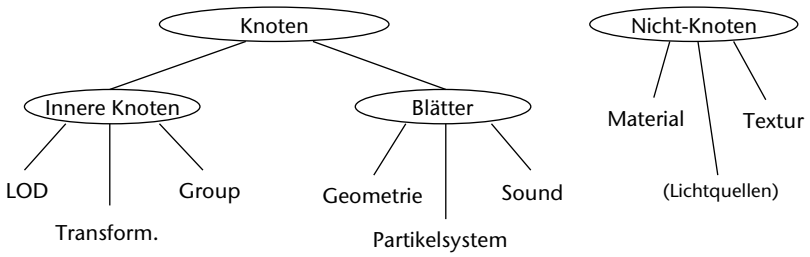


Semantik

- 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 der 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

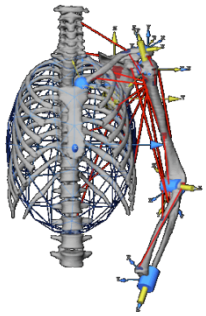
Knotenarten

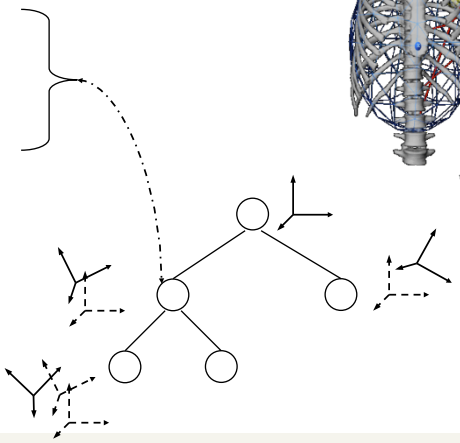
- 2 Hierarchien: Scenegraph-Hierarchie + Klassenhierarchie
- Die Mächtigkeit und Flexibilität eines Szenengraphen hängt von der Menge der zur Verfügung stehenden Knotenklassen ab!
- Etliche Klassen sind nicht Teil des Scenegraphen, aber doch Teil der Szene



Transformationssemantik

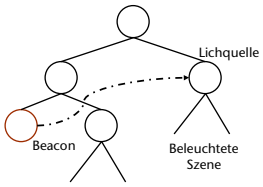
- Alle Scenegraphs behandeln diese Semantik gleich
- Transformationsknoten ≡ neues lokales Koordinatensystem
- `pushMatrix();`
  `multMatrix( … );`
  traverse sub-tree
  `popMatrix();`



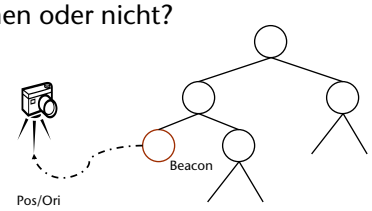


Issues

- 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

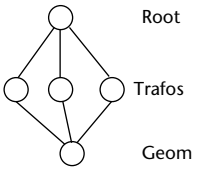


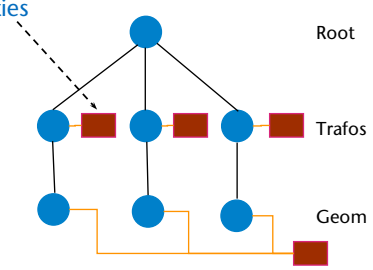
Material

- 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!

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





Multi-threading / Thread-safety

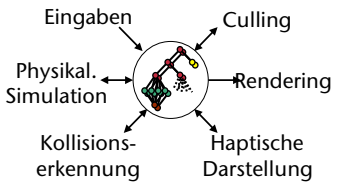
- Idee: mehrere Szenengraphen
- Problem: Speicheraufwand
- Lösung:
  - *Copy-on-Write* der Attachments
    → *"Aspects"*
  - Jeder Thread "sieht" einen eigenen Aspect
  - Problem: einfacher Zugriff über Pointers

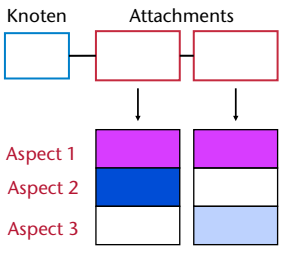
    `geom->vertex[0]`

    geht nicht mehr
  - Lösung (in C++):
    - "Smart Pointers"

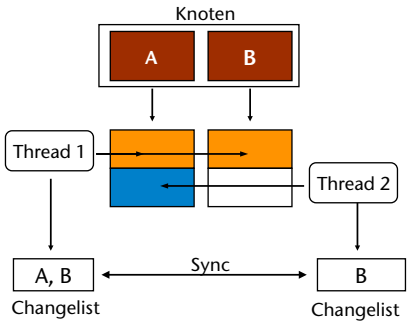
    ➢Pro Klasse eine Pointerklasse. Bsp.:

```
geomptr = Geometry::create(…);
geomptr->vertex[0] ...
```





- 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

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, ...)

Anwendungskriterien für Szenengraphen

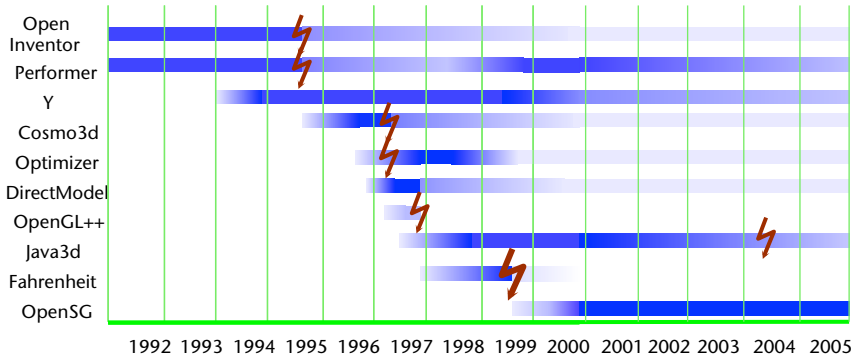
- Wann soll man Scenegraphs verwenden:
  - Kompexe Szenen: viele verschiedene Materialien, viel Geometrie, oft ist nur ein Teil zu sehen, komplexe Transformationshierarchien
  - Relativ statische Geometrie
  - Spezifische Features, die ein Scenegraph bietet (Partikel, Clustering, ...)
- Wann man einen Scenegraph nicht verwenden soll:
  - Einfache Szenen (ein Objekt in der Mitte)
  - Visualisierung von technisch-wissenschaftlichen Daten
    - Z.B. CT/MRI-Volumen-Daten
  - Hochgradig dynamische Geometrie

Einige (ehem.) populäre Scenegraphs

- SGI Performer
- Java3D (http://www.java3d.org/)
- Inventor/Coin (https://bitbucket.org/Coin3D/coin/wiki/Home)
- VRML & X3D
- OpenSG (http://www.opensg.org/) !
- Open Scene Graph
- Ogre3D
- Castle Engine (für VRML, http://castle-engine.sourceforge.net)
- Viele andere (siehe "Game Engines List": http://en.wikipedia.org/wiki/List_of_game_engines

Geschichte

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



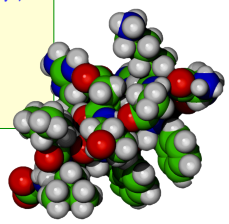
Exkurs: Memory-Layout für schnelles Rendering

- Häufiges Problem: elegante Strukturierung der Daten (aus Sicht des Software-Engineerings) ist ungünstig für schnelles Rendering
- Terminologie: "Array of Structs (AoS)" vs. "Struct of Arrays (SoA)"
- Zur Illustration: Beispiel Molekül-Visualisierung
  - Sauberes Software-Engineering würde folgende Klasse enthalten

```
class Atom
{
public:
    Atom( uint atom_number, Vec3 position, ... );
private:
    Vec3 position_;
    uint atom_number_;
    Atom bonds_[max_num_bonds];
    ...
};
```

- Damit wäre dann eine Molekül folgendermaßen zu definieren:

```
class Molekule
{
public:
    Molekule( const std::vector<Atom> & atoms );
private:
    std::vector<Atom> atoms_;
    ...
};
```



- Memory Layout eines AoS:

pos	num	bonds	pos	num	bonds	pos	num	bonds	

- Problem dabei: Speicher-Transfer ist sehr langsam
- Alternative: Struct of Arrays