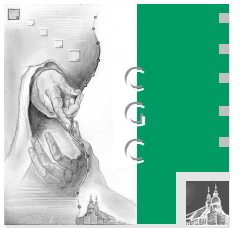
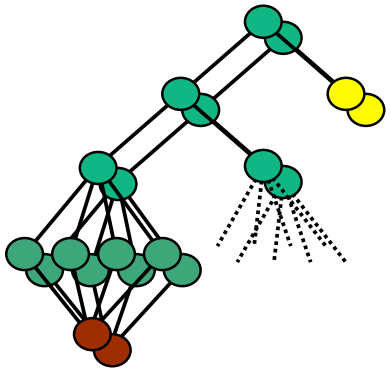


Virtuelle Realität
Scenegraphs



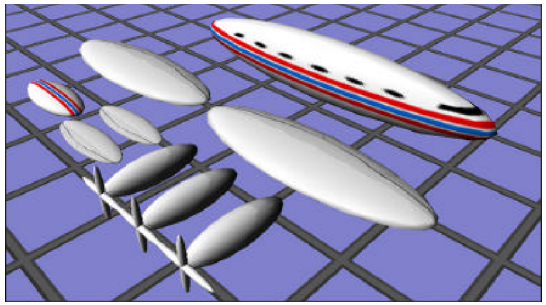
G. Zachmann

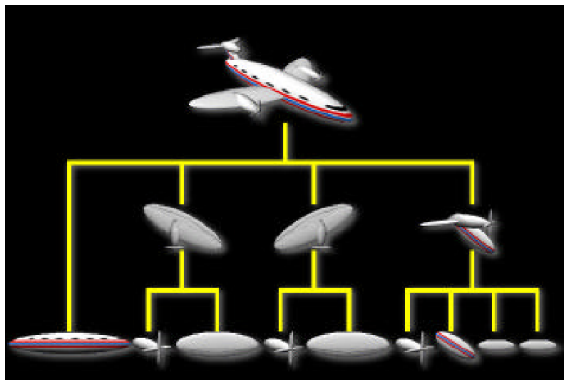
Clausthal University, Germany

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:

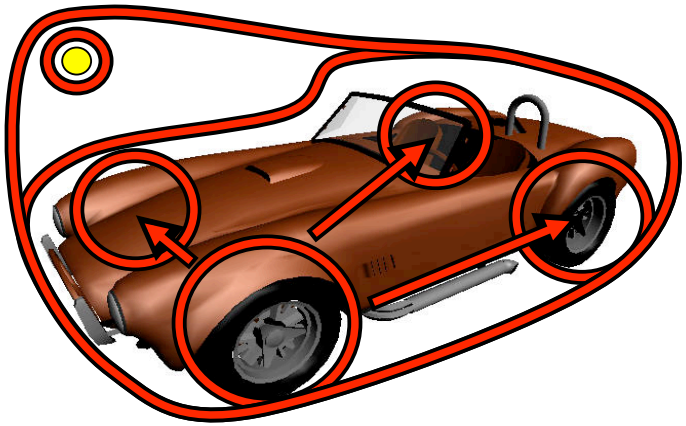


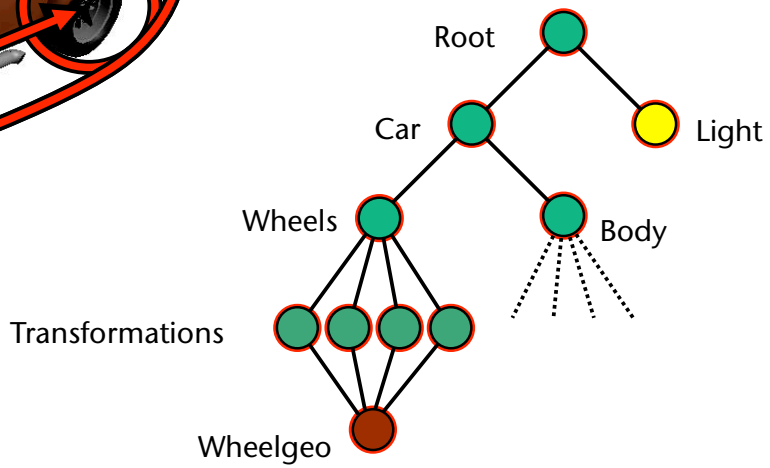


- *Code re-use* und *Know-how re-use*!
- Descriptive, not imperative (cv. C vs. Prolog)
  - Thinking objects ... not rendering processes

Struktur eines Szenegraphen

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



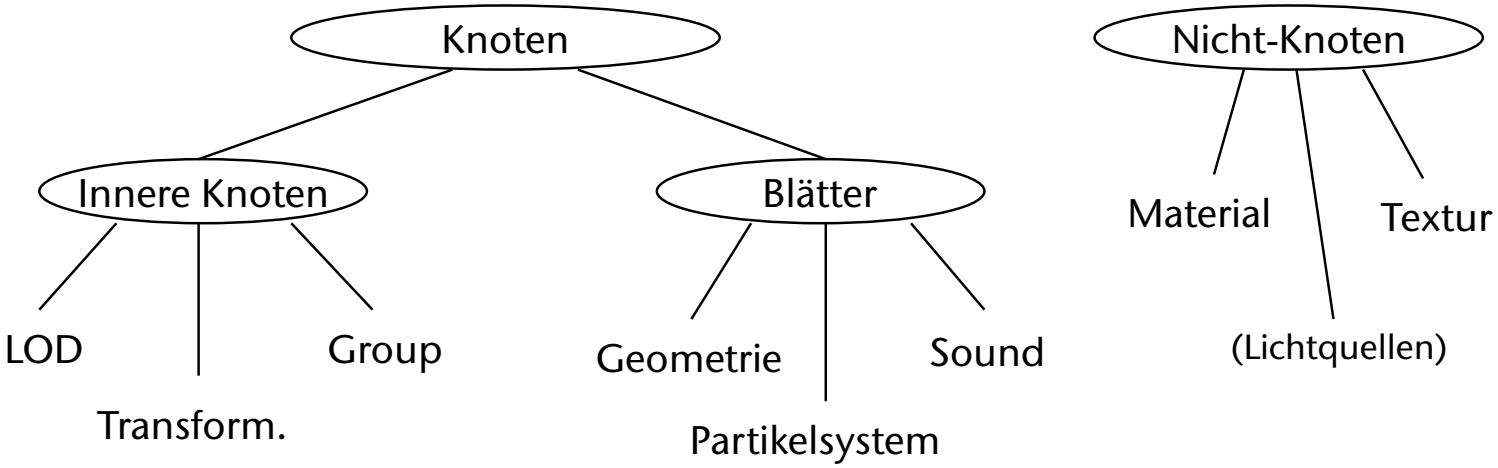


Semantik

- 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 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

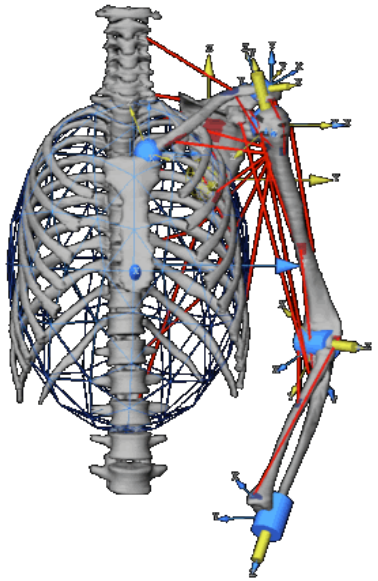
Knotenarten

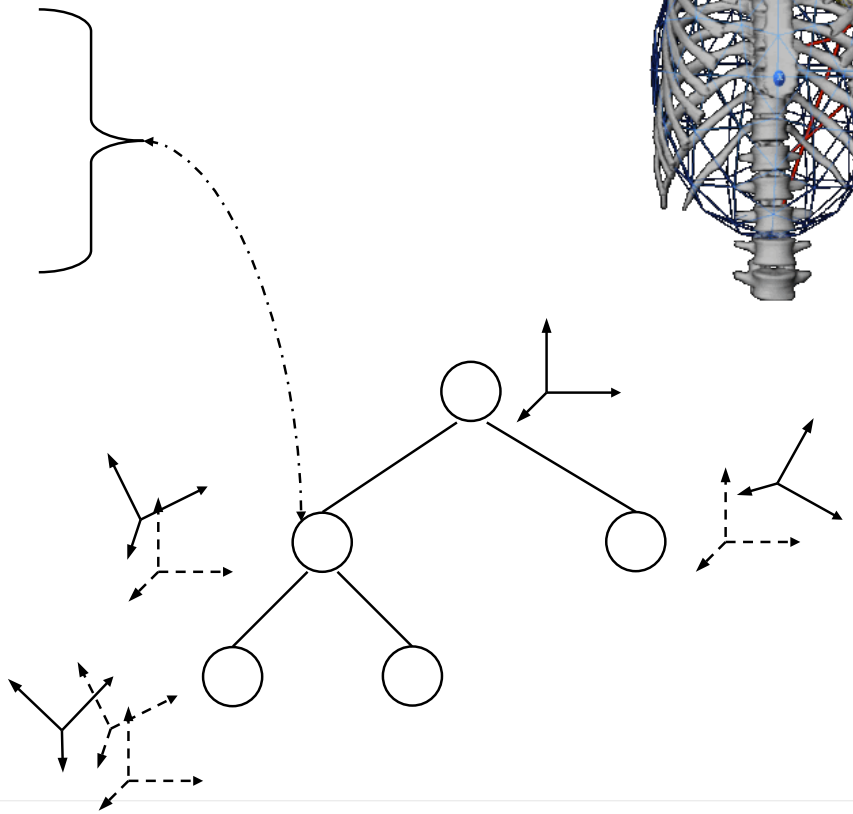
- 2 Hierarchien: Szenegraph-Hierarchie + Klassenhierarchie
- Die Mächtigkeit und Flexibilität eines Szenegraphen hängt von der Menge der zur Verfügung stehenden Knotenklassen ab!
- Etliche Klassen sind nicht Teil des Scenegraphen, aber doch Teil der Szene



Transformationssemantik

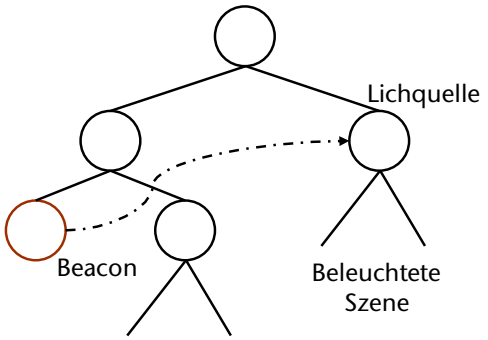
- Alle Scenegraphs behandeln diese Semantik gleich
- Transformationsknoten ≡
  neues lokales Koordinatensystem
- `pushMatrix();`
  `multMatrix( … );`
  traverse sub-tree
  `popMatrix();`



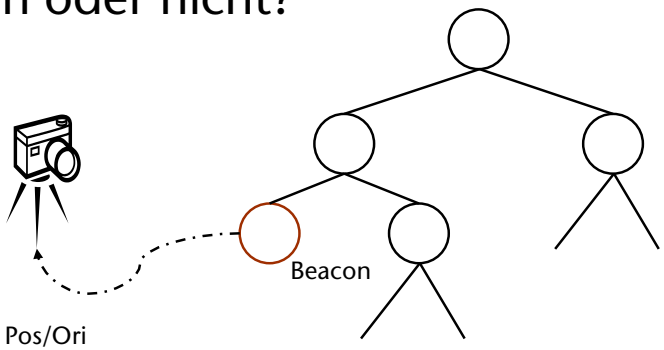


Issues

- 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

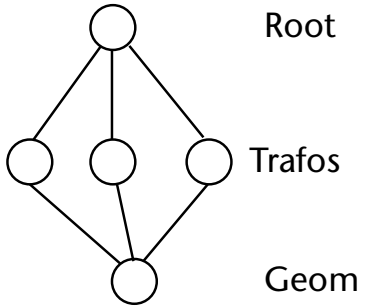


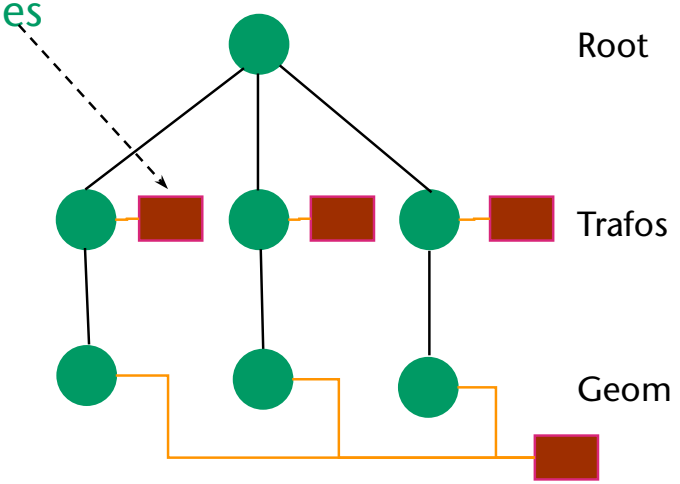
Material

- 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

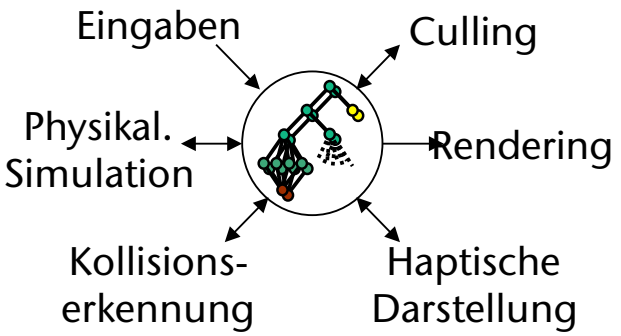
- Problem: große Szenen mit viel identischer Geometrie
- Idee: DAG (statt Baum)
  - Problem: Zeiger/Namen von Knoten sind nicht mehr eindeutig!
- Lösung: trenne Struktur von Inhalt
  - Baum besteht nur noch aus einer Sorte Knoten
  - Knoten bekommen spezielle Eigenschaften / Inhalt durch Attachments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





Multi-threading / Thread-safety

- Performer: App / Cull / Draw – Modell
- Idee: mehrere Szenengraphen
- Problem: Speicheraufwand

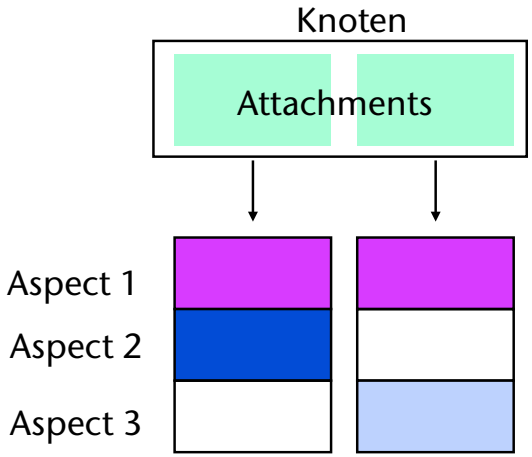


- Lösung:
  - *"Aspects"* und *Copy-on-Write* der Attachments
  - Jeder Thread "sieht" einen eigenen Aspect
  - Problem: einfacher Zugriff über Pointers

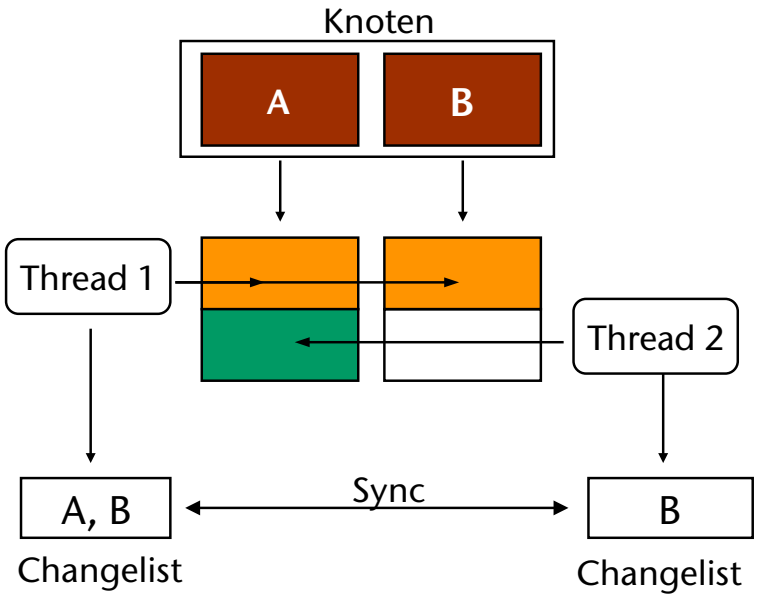
    ```
    geom->vertex[0]
    ```

    geht nicht mehr
  - Lösung:
    - Smart Pointers
    - Pro Klasse eine Pointerklasse. Bsp.:

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



- 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 Szenegraphen

- 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)
  - Hochgradig dynamische Geometrie

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 & X3D
- OpenSG (http://www.opensg.org/) !
- Open Scene Graph
- Viele andere (siehe www.sf.net , "Game Engines List", ...)

Geschichte

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