Computergraphik II

Prof. Dr. Andreas Kolb

Fachgruppe Computergraphik und Multimediasysteme

Universität Siegen – Fachbereich 12

Version: 7. Januar 2005

Vergleich OpenGL vs. OpenInventor

Sprachkonzept OpenGL: imperativ (C), OpenInventor: Objekt-orientiert (C++)

Graphik-Zugriff

- OpenGL: direkt auf Framebuffer
- OpenInventor: Scene-Database

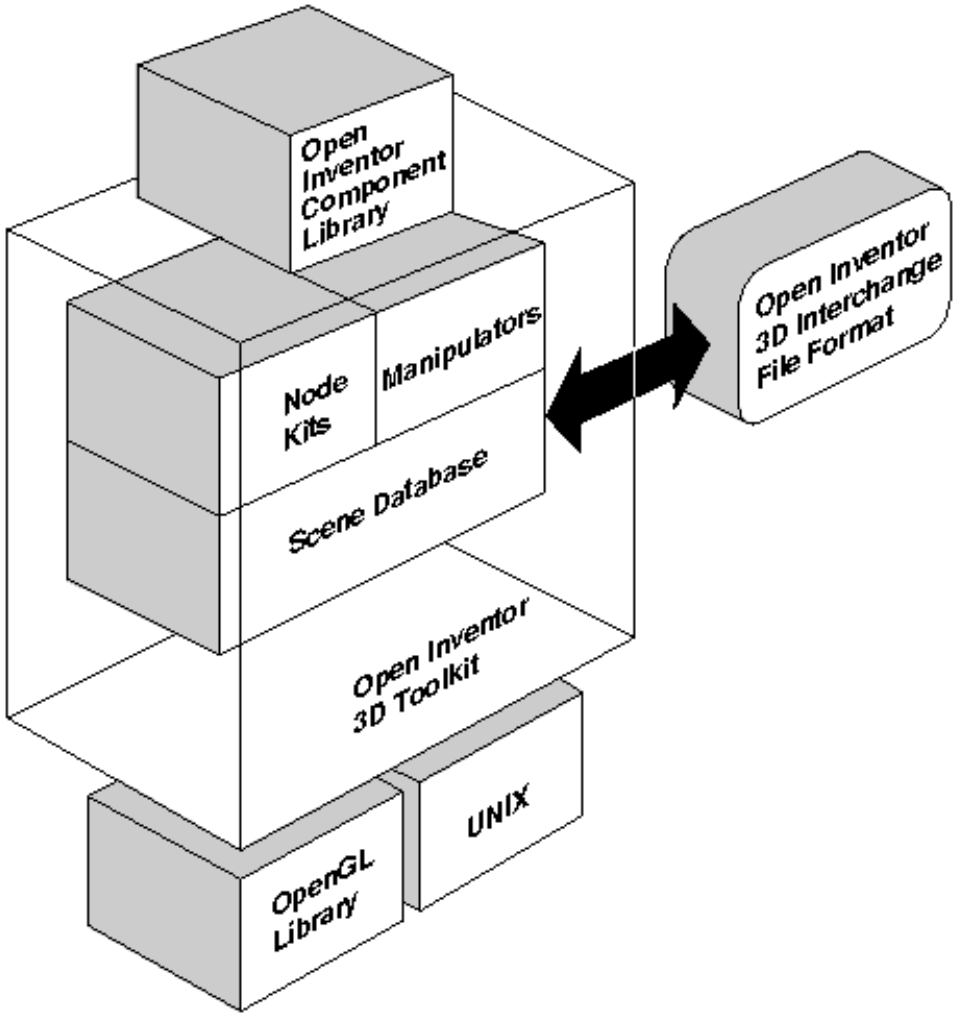
Interaktion/Animation

- OpenGL: Grundelemente
- OpenInventor: Manipulatoren, Engines

File-Format

- OpenGL: keines
- OpenInventor: iv-Format

OpenInventor bietet Szenenoptimierung und Komponenten wie Viewer, Editoren, etc.



Literatur

Josie Wernecke: *Inventor Mentor*, Addison-Wesley, 1994

Josie Wernecke, OpenInv. Architecture Group: *Inventor Toolmaker*, Addison-Wesley, 1994

Bezugs-Quellen

Coin3D `www.coin3d.org`**:** Open Source Clone des SGI Open Inventor

`oss.sgi.com/projects/inventor` weitere Programme zum Thema Inventor, z.B.:

Demos und Beispiele: u.a. für den „Inventor Mentor“ und den „Inventor Toolmaker“

Tools wie

- `ivview` zum Darstellen eines Inventor-Files
- `gview` zum parallelen Darstellen eines Inventor-Files und dessen Szenengraph

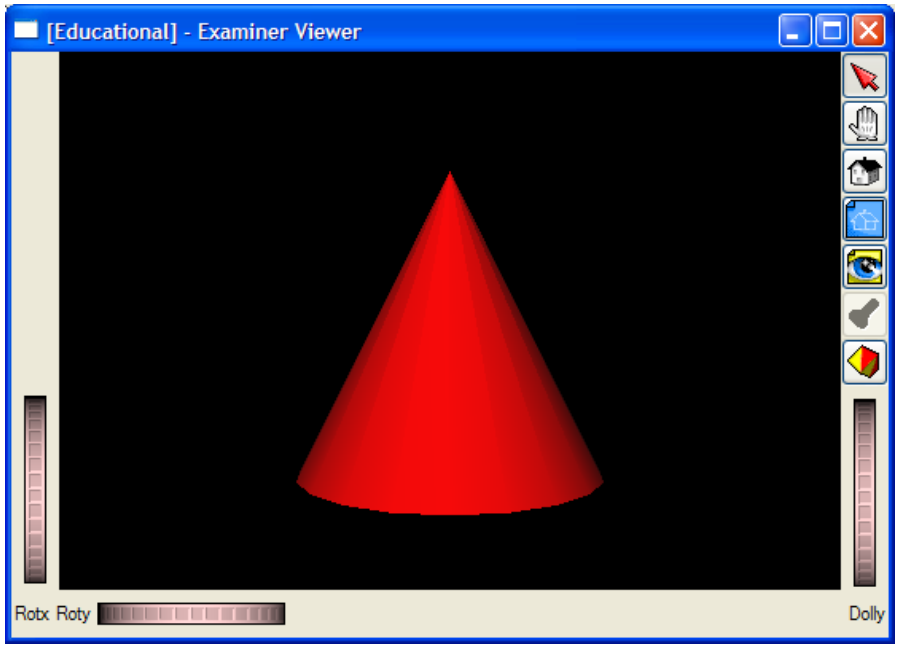
Converter zum Transformieren von Inventor-Files, z.B.

- `ivinfo` gibt statistische Informationen zu Inventor-Files aus
- `ivfix` optimiert Inventor-Files zum schnelleren Rendern
- `ivcat` zeigt (insb. binäre) Inventor-Files in ASCII an

Beispiel: „Hello Cone“

```
// Initialize Inventor
Widget myWindow = SoXt::init(argv[0]);
if (myWindow == NULL) exit(1);

// Make a scene containing a red cone
SoSeparator *root = new SoSeparator;
SoPerspectiveCamera *myCam
                  = new SoPerspectiveCamera;
SoMaterial *myMaterial = new SoMaterial;
myMaterial->diffuseColor.setValue(1, 0, 0);

root->ref();
root->addChild(myCam);
root->addChild(new SoDirectionalLight);
root->addChild(myMaterial);
root->addChild(new SoCone);

// Create a renderArea to show the scene
SoXtRenderArea *myRA
               = new SoXtRenderArea(myWindow);

// Make myCamera see everything.
myCam->viewAll(root,
               myRA->getViewportRegion());
```

```
// Put our scene in myRA, change the title
myRA->setSceneGraph(root);
myRA->setTitle("Hello Cone");
myRA->show();

SoXt::show(myWindow);  // Display main win.
SoXt::mainLoop();      // Main event loop
```



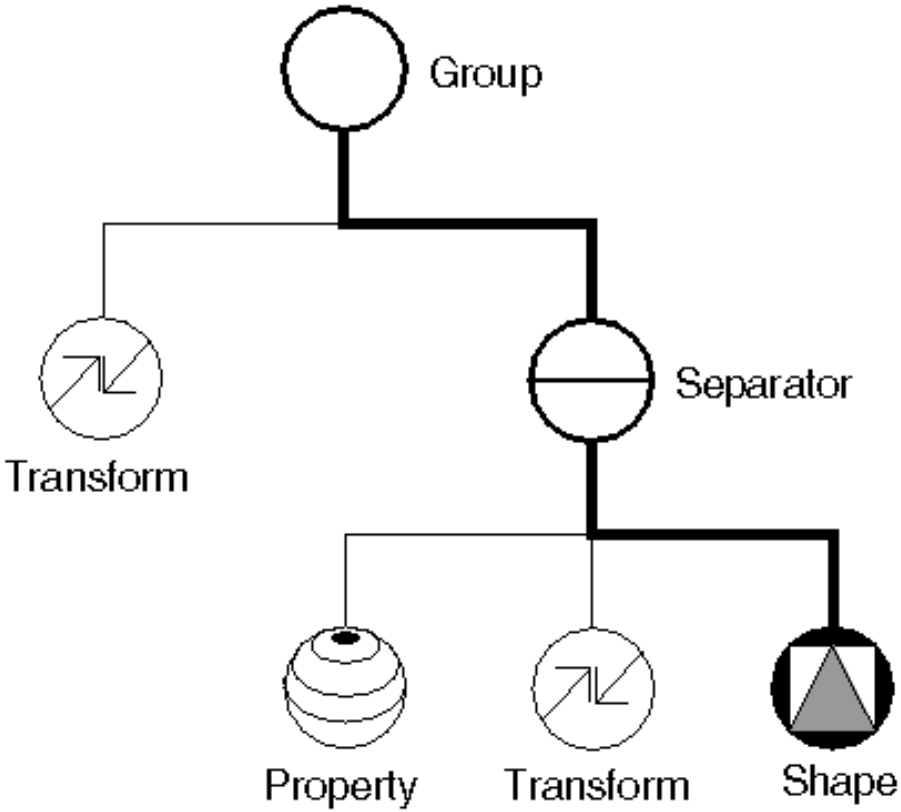
„Hello World“-Beispiel

11.1 Scene-Database / Szenengraph

Nodes/Knoten: Alle Informationen werden hier abgespeichert bzw. definiert

Grundlegende Node-Klassen:

- *shape nodes:* Primitive wie *sphere, cube, cylinder* oder *quad-mesh*
- *property nodes:* Eigenschaften wie Licht und Material
- *group nodes:* Hierarchie-Aufbau



Traversierung:

- Pfade definieren Objekte
- Top-Down und Links-Recht $\Longrightarrow$ Reihenfolge wichtig
- Separatoren verwalten stack-artig verschiedene Zustände

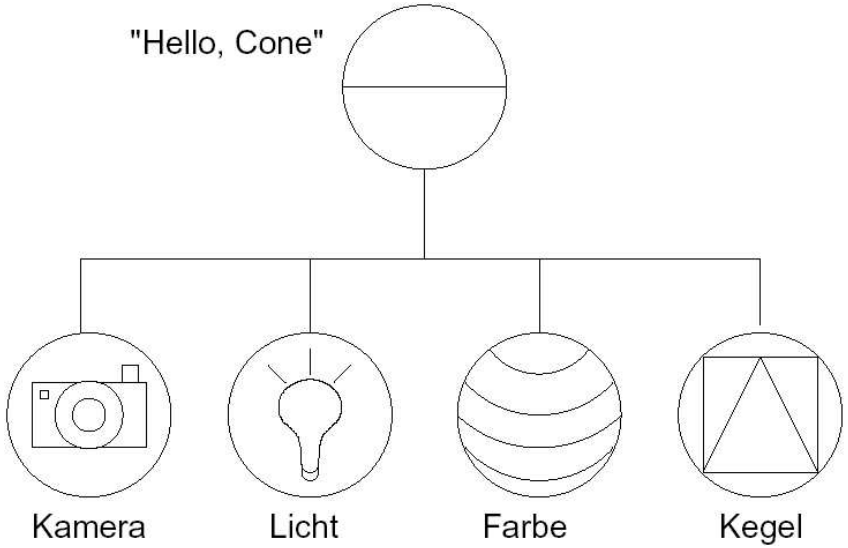
Traversal-State $\hat{=}$ aktueller Zustand

Zum „Hello Cone“-Beispiel

Programm-Auszug:

```
// Make a scene containing a red cone
SoSeparator *root = new SoSeparator;
SoPerspectiveCamera *myCam
               = new SoPerspectiveCamera;
SoMaterial *myMaterial = new SoMaterial;
myMaterial->diffuseColor.setValue(1,0,0);

root->ref();
root->addChild(myCam);
root->addChild(new SoDirectionalLight);
root->addChild(myMaterial);
root->addChild(new SoCone);
```

Scene-Graph:



Einfluß:

- Material beeinflußt cone-Objekt
- Material beeinflußt nicht evtl. weitere Objekte rechts vom Seperator

Hinweis: Im folgenden werden Licht und Camera im Scene-Graph weggelassen

Group-Nodes und Seperatoren

Grundsätzlich: Nur Group-Nodes können Kind-Knoten (*Child-Nodes*) besitzen

Gruppierung mit unterschiedlicher Zielsetzung, z.B.:

`SoGroup`: Einfache Zusammenfassung (Basisklasse)

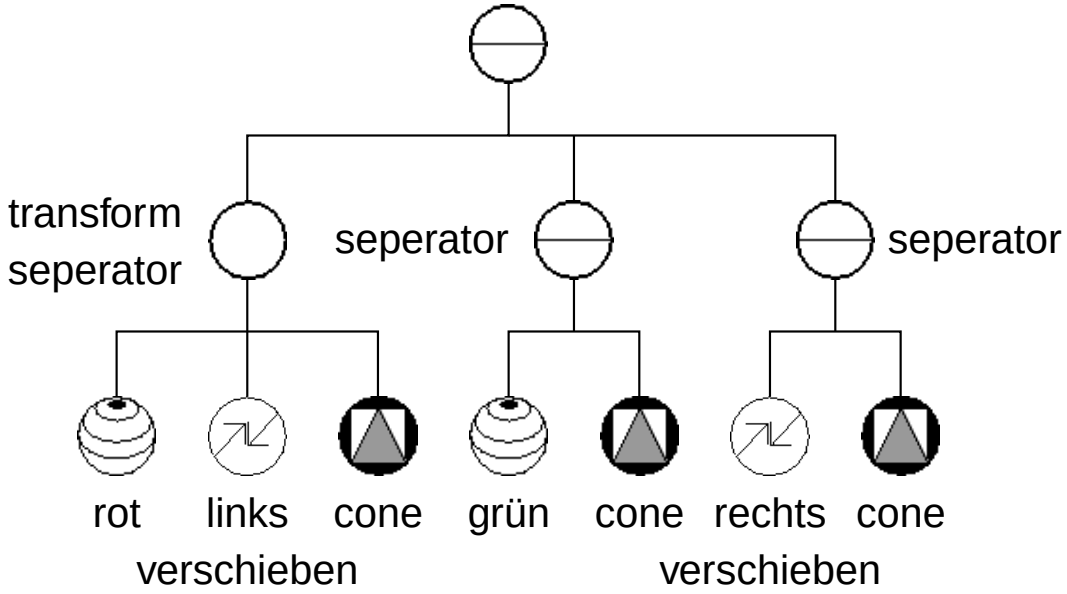
`SoSeparator`: Zusätzliches Stacking des Traversal-States

`SoTransformSeparator`: Spezialfall von `SoSeparator`; stacked nur Transformations-Status (analog zum Matrix-Stack in OpenGL)

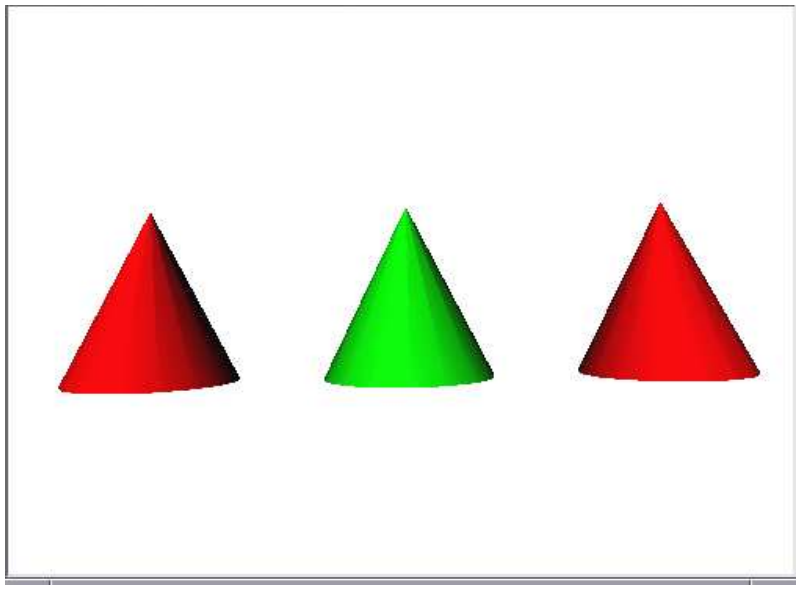
`SoSwitch`: „Schalter“, nur ein Child wird traversiert

`SoLevelOfDetail`: Schaltet Child abhängig von Beobachter-Entfernung aktiv

Beispiel: Einfluß von Seperatoren



Scene-Graph



Szene

- das rote Material wirkt sich global aus, das grüne Material nur lokal
- alle Transformationen wirken sich lokal aus

Allgemeines zu Nodes

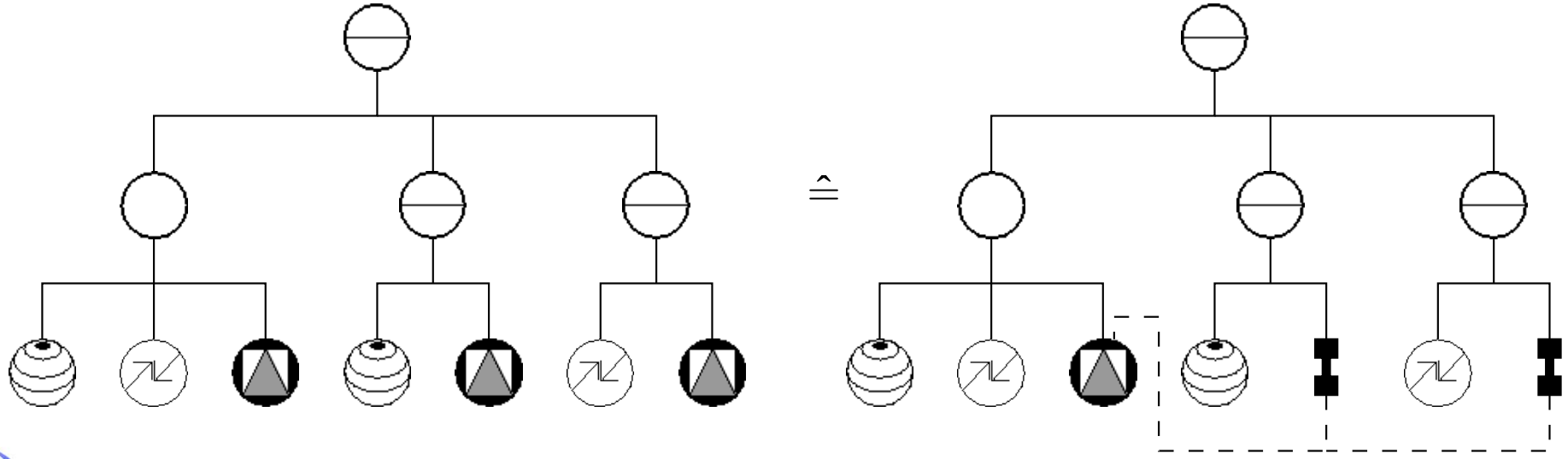
Fields sind die Datenfelder von Nodes

Unterscheide *single value field* und *multiple value field*

Action wird auf Scene-Graph angewendet (⟶ traversal) und von Nodes spezifisch umgesetzt

Path-Objekt beschreibt den Pfad im Kontext der Traversierung ⇒ Zugriff auf Vorgänger-Nodes, Manipulation des Scene-Graph

DEF/USE: Nodes können benannt und wiederverwendet werden (⇒ DAG)



Shape-Nodes

Einfache Basistypen: `SoCone`, `SoCube`, `SoSphere`, `SoCylinder`, ...

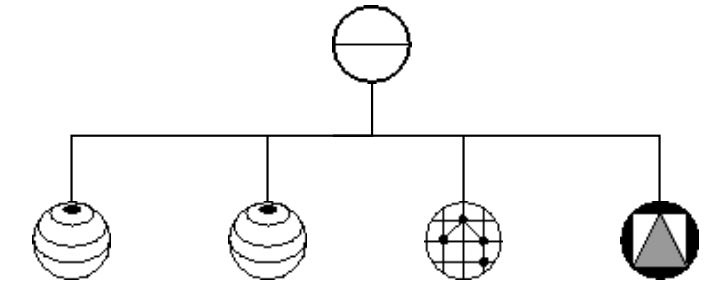
Freiformkurven und -flächen: NURBS

Polygon-Geometrien: Unterscheide zwischen

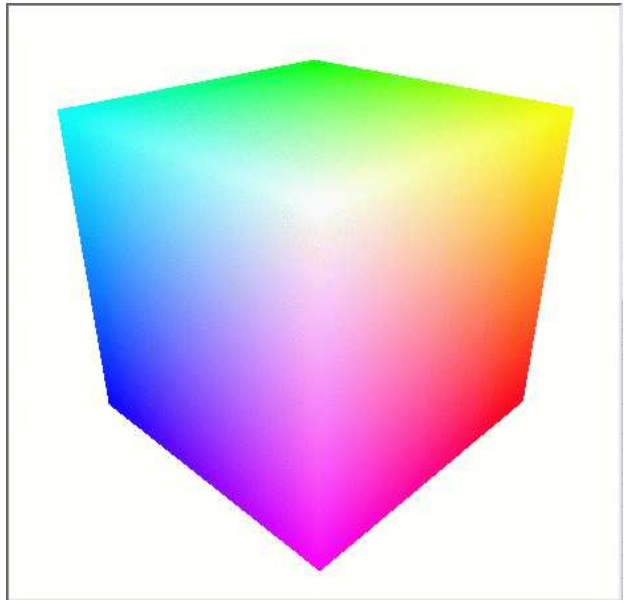
1. nicht indizierten Geometrien: `SoFaceSet`, `SoQuadMesh`, `SoTriangleStripSet`, ...
2. indizierten Geometrien: `SoIndexedFaceSet`, `SoIndexedTriangleStripSet`, ...

Beispiel: RGB-Farbwürfel als `SoIndexedFaceSet`:

- Material legt hier diffuse Farbe fest
- Property-Node `MaterialBinding` bindet hier Farben an Vertices



Scene-Graph



RGB-Cube

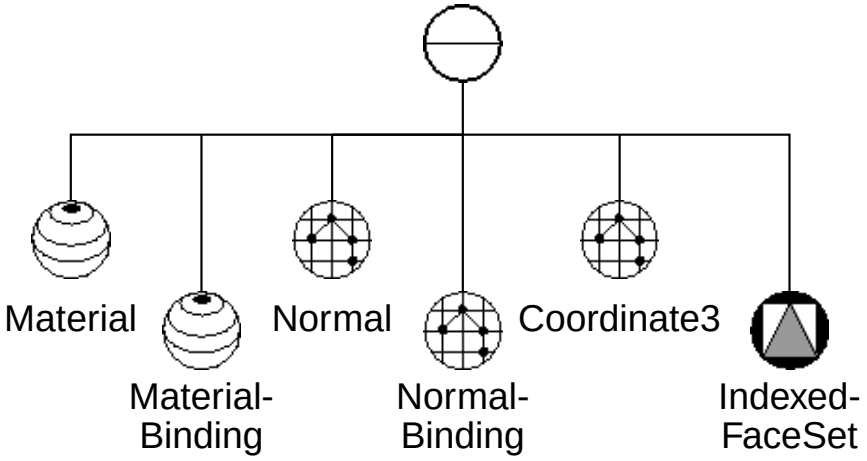
Property-Nodes

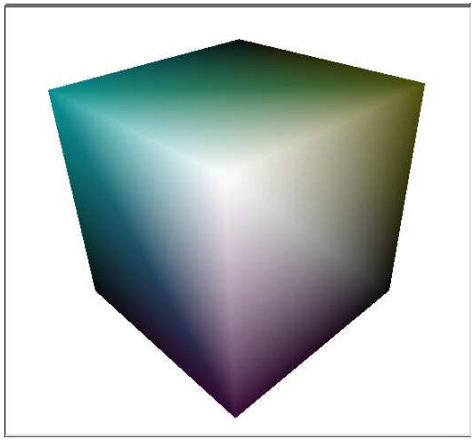
Transformationen: `SoTransform`, `SoRotation`, `SoScale`, `SoTranslation`, ...

Appearance:
- Materialparameter: `SoMaterial` (Phong-Parameter), `SoMaterialBinding`
- Darstellungsart: `SoDrawStyle` (Punkte, Gitter, Polygon) und `SoLightModel`
- Komplexität `SoComplexity`: Detailgrad der Darstellung (Object- u. Screen-Space)

Geometrie-Daten: `SoCoordinate` (Vertex-Koordinaten), `SoNormal` (Normalen), `SoNormalBinding`

Beispiel: RGB-Cube mit Per-Vertex-Normalbinding:

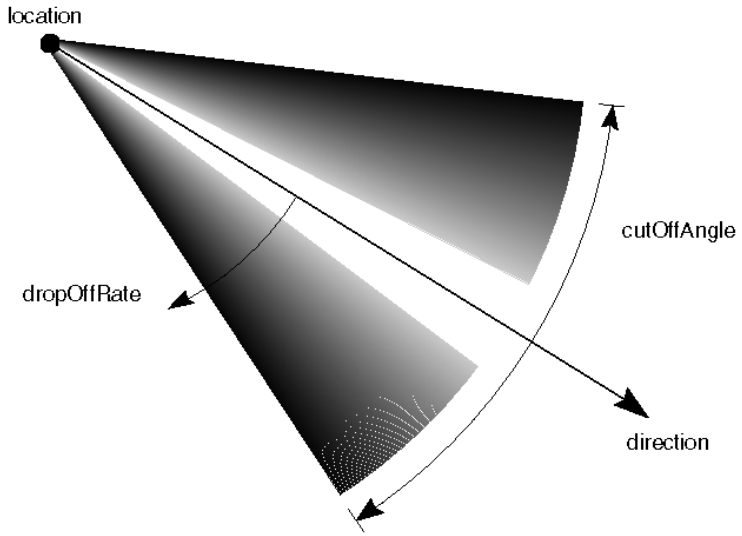


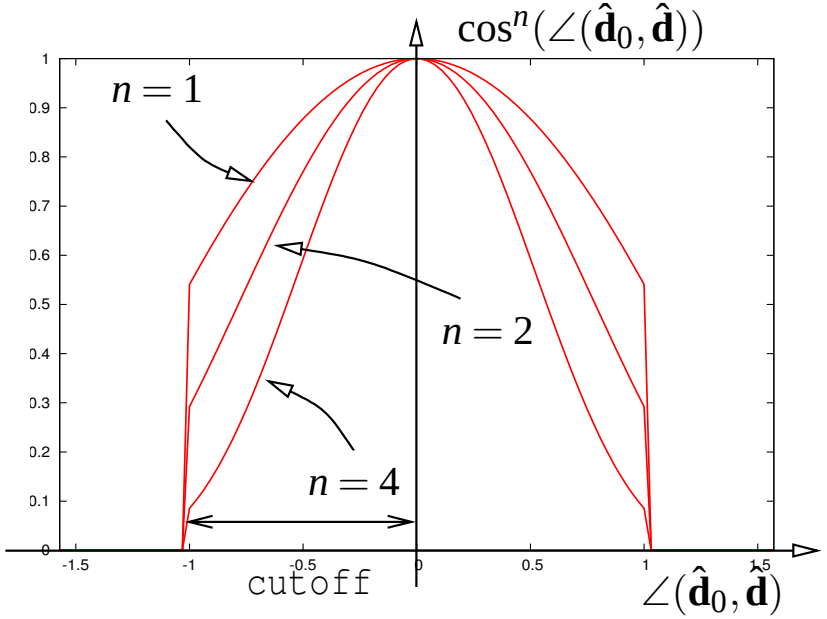


Licht und Camera

Camera:
- orthographisch & perspektivisch
- Kamera muß *vor* allen Objekten der Szene im Graph stehen

Lichtquellen:
- beleuchten nur *nachfolgende* Objekte im Graph
- Position und Orientierung entsprechend der aktuellen Transformationsmatrix
- `SoPointLight`: Punktlicht, in alle Richtungen gleichförmig
- `SoDirectionalLight`: Richtungslicht/paralleles Licht
- `SoSpotLight`: Punktlicht mit Hauptausstrahlrichtung





Intensitätsverteilung Spot-Light

11.3 Actions

Vorgehen:
- Actions werden auf Scene-Graph (oder Teil-Graph) angewendet
  - Action wird für jeden Node ausgeführt

Beispiele für Actions:

- `SoGLRenderAction`: Darstellen der Szene entsprechend der Kamera
- `SoWriteAction`: Scene-Graph in Datei schreiben
- `SoSearchAction`: Sucht Pfade zu spezifischen Nodes
- `SoRayPickAction`: Picking

Anwendung in einem OpenInventor-Programm:

1. Instanziierung eines Action-Objektes
2. Setzen Action-spezifischer Parameter
3. Anwenden der Action unter Angabe eines Nodes (i.a. root oder Teil-Graph)
4. Auswerten des Ergebnisses (Action-spezifisch)

11.4 Sensoren

Ziel: Reaktion auf Zustandsänderungen im Scene-Graph oder auf Zeit-Ereignisse

Sensoren
- *triggern* anwendungsspezifische *callback*-Funktionen
  - werden in *Queues* verwaltet

Daten-Sensoren
- sind einem Field (`SoFieldSensor`), einem Node (`SoNodeSensor`) oder einem Pfad (`SoPathSensor`) zugeordnet
- triggern, sobald sich ein Datum geändert hat

Zeit-Sensoren:
- `SoAlarmSensor`: Einmaliges Triggern zu festgelegtem Zeitpunkt
- `SoTimerSensor`: Triggern in vorgegebenen Abständen, z.B. für Animationen

Beachte: Sensoren sind keine Nodes!

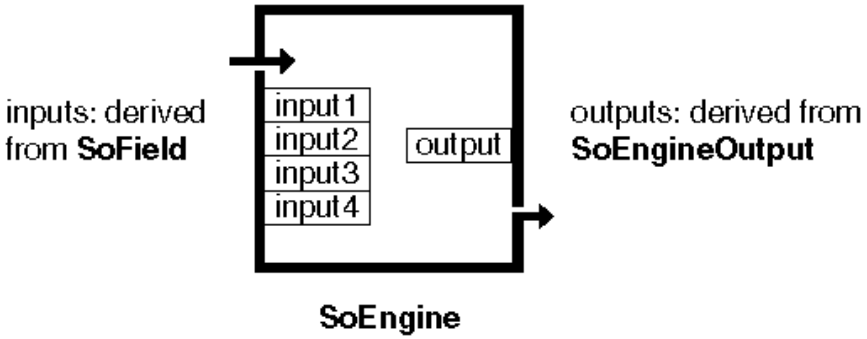
Einsatzzweck: Komplexe Anwendungen mit dynamischem Scene-Graph

11.5 Engines

Ziel: Integration von Bewegung direkt in den Scene-Graph

Beachte: Im Gegensatz zu Sensoren können nur Node-Objekte betroffen sein

Generell erzeugen Engines aus $0 - n$ Eingabedaten ein Ausgabedatum



Engine-Typen: Entsprechend der Zielsetzung werden Engines eingestuft

Arithmetische Engines: Berechnung anhand der Eingabedaten

Animations Engines: Zeitbasierte Verarbeitung

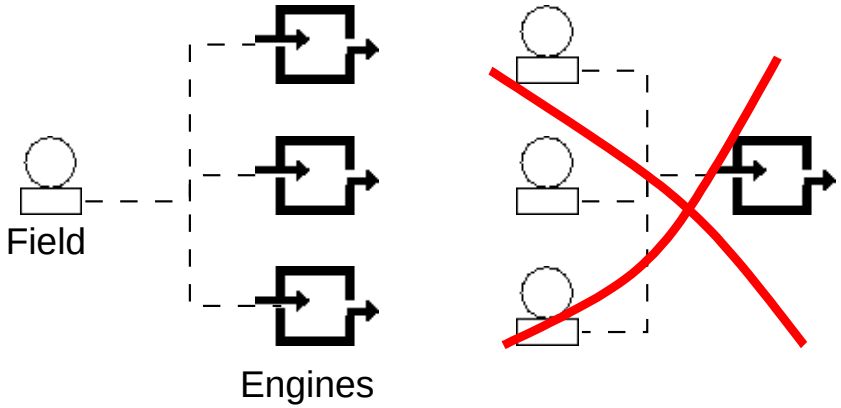
Trigger Engines: Zustandabhängige Ergebnisberechnung

Array Manipulation: Verarbeitung von Arrays

11.5 Engines

Field-Connections ermöglichen die Verknüpfung von Node-Feldern mit Engine-Feldern

Fan-In und Fan-Out:
- Multiple Verknüpfung des Engine-Outputfeldes mit Node-Feldern möglich (Fan-Out)
- Multiple Verknüpfung eines Engine-Inputfeldes mit Node-Feldern nicht erlaubt (Fan-In)



Typ-Konversion: Teilweise können auch Felder unterschiedliches Types verknüpft werden

Beachte: Implizite Konversion u.U. mit falschem Ergebnis

Global Fields können zusätzlich angelegt werden, um Ergebnisse zu speichern; sind nicht einem Node zugeordnet!

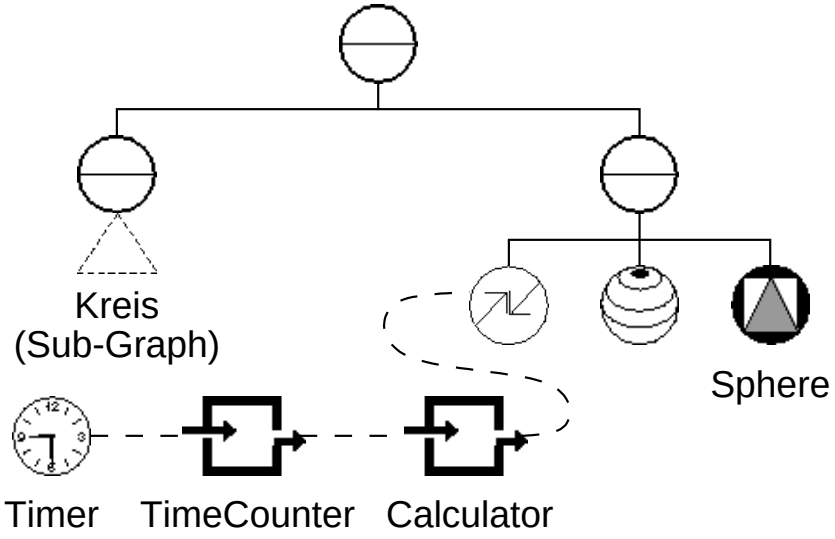
Beispiel: SoTimeCounter und SoCalculator Engine

Ziel: Bewegung einer Kugel auf einer Kreisbahn

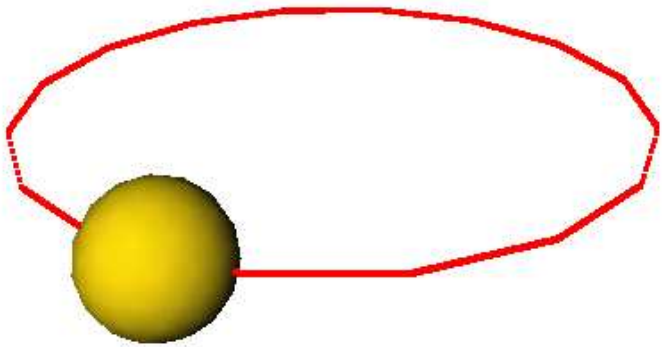
Umsetzung: Zwei Engines,

1. `SoTimeCounter` zählt von 0 auf 359 mit gegebener Geschwindigkeit ($\Rightarrow$ Winkel $\theta(t)$)
2. `SoCalculator` rechnet aus $\theta(t)$ eine Kreisposition aus

   $\Rightarrow$ verwende Kreisposition als Translationsvektor für Kugel



Scene-Graph



Kugel auf Kreisbahn

11.6 Dragger und Manipulatoren

Dragger:
- Node, der auf Interaktion reagiert
  - Interaktion arbeitet mit Picking auf Dragger-Geometrie

Manipulator:
- Subklassen von „statischen“ Nodes
  - verwenden Dragger, um die sonst „statischen“ Felder zu verändern

Beispiel: Manipulator

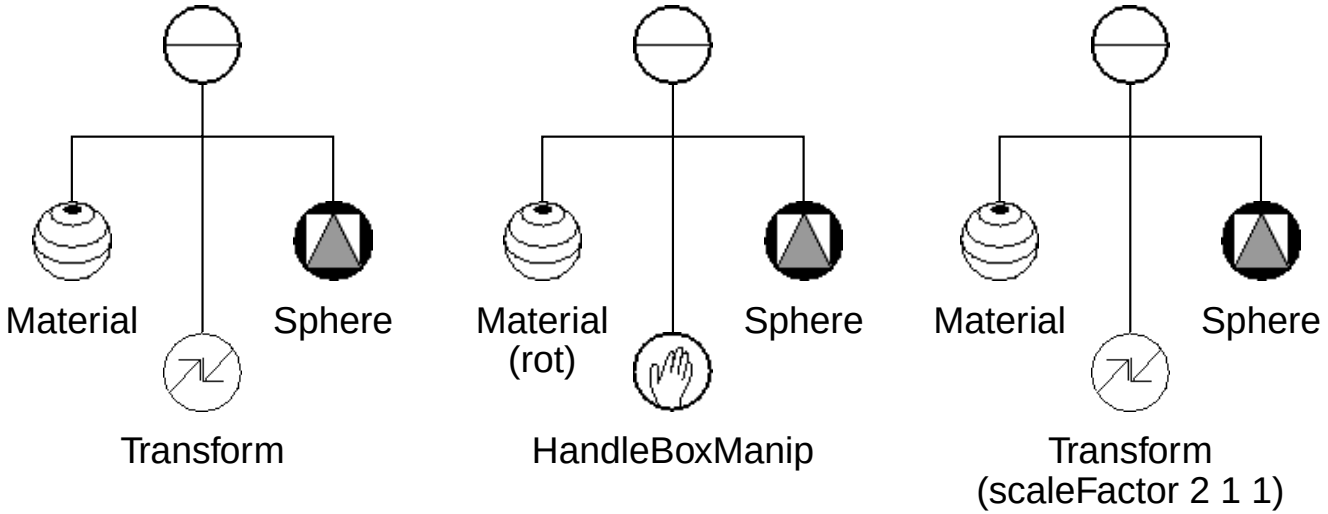
`SoHandleBoxManip` als Subklasse von `SoTransform`

Ablauf in der Anwendung:

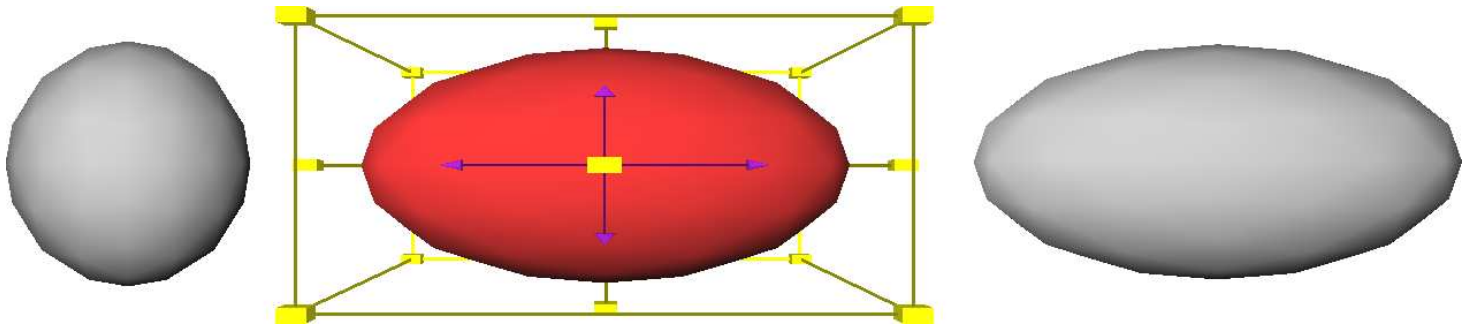
1. statischer Scene-Graph mit Transform-Node und Objekt
2. Picking des Objektes $\Rightarrow$ Transform-Node wird durch Manipulator ersetzt
3. Nutzer-Interaktion mit Dragger verändert Transform-Wert
4. Abschluß: Manipulator wird durch Transform-Node ersetzt (mit korrekten Werten)

Beispiel: Manipulator (Forts.)

Scene-Graphen:



Szenen im zeitlichen Ablauf



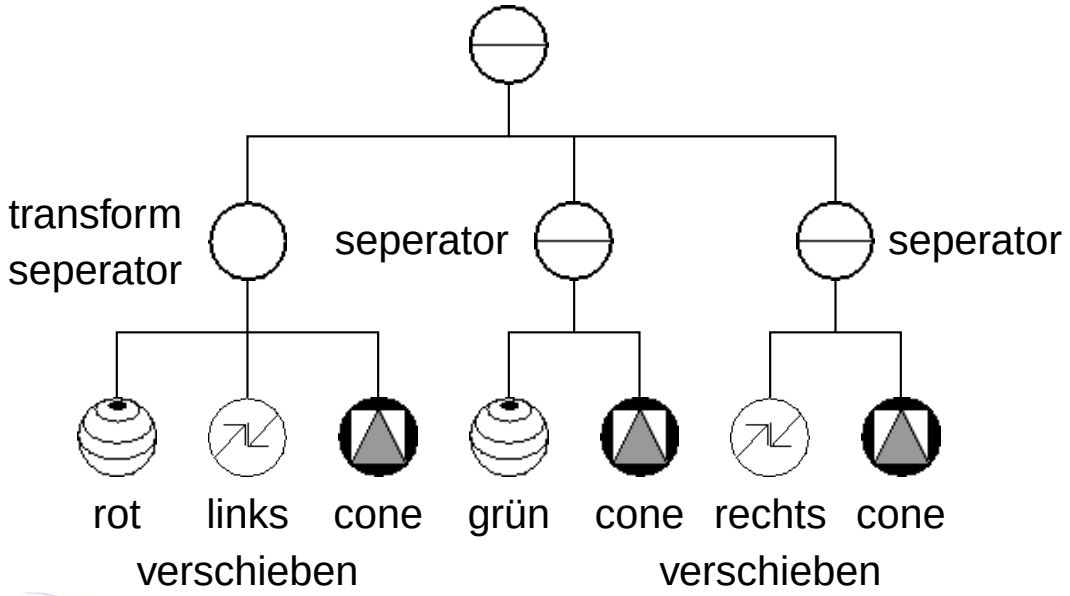
11.7 Datei-Format (.iv)

Ziel: Persistente Abspeicherung eines Scene-Graph

Abspeichern durch eine `SoWriteAction`

Laden über die statische OpenInventor-Database-Methode `SoDB::readAll`

Beispiel für eine Serialisierung:




```
#Inventor V2.1 ascii
Separator {
  TransformSeparator {
    Material {  diffuseColor 1 0 0 }
    Transform { translation -3 0 0 }
    Cone {  }
  }
  Separator {
    Material {  diffuseColor 0 1 0 }
    Cone {  }
  }
  Separator {
    Transform { translation  3 0 0 }
    Cone {  }
  }
}
```

Hinweis: OpenInventor ist State-Machine mit Default-Werten (wie OpenGL)

Speichermanagement

Wichtig für Programmierung: OpenInventor hat ein eigenes Speicher-Management

Referenz-Zähler (Ref-Count): Jeder Node zählt Referenzen zu anderen Nodes

- Initial: Ref-Count auf 0
- Einhängen in die Hierarchie, Löschen von Kindern etc. $\Rightarrow$ geänderter Ref-Count
- Ref-Count auf 0 reduziert $\Rightarrow$ Node-Objekt wird gelöscht

Policy für Bearbeitung von Node-Objekten:

```
SoAnyNodeType *anyNode = getAnyNodeObjectFromSceneGraph();
anyNode->ref();              // increment ref-count
...                          // work with object
anyNode->unref();            // decrement ref-count at the end
```

$\Rightarrow$ `anyNode`-Zeiger während der Bearbeitung immer gültig!

Node-Kits

Ziel: Einfache Erstellung kompletter Sub-Graphen

Node-Kits verwalten die Nodes eines Sub-Graphen

Node-Kit-Catalog: Vollständige Liste aller *möglichen* Nodes eines Node-Kits

Initial umfaßt ein Node-Kit nur absolut notwendige Nodes

Erzeugen und Zugriff auf Nodes, sowie Setzen von Parametern durch Namensreferenzen

```
SoShapeKit  myShape = new SoShapeKit();          // a new shape kit
myShape->set("material", "diffuseColor 1 0 1"); // create material node
                                                 // and set diffuse color
SoTransform *t = (SoTransform*) myShape->getPart("transform", TRUE);
                                                 // create and retrieve
                                                 // transform-node
```

Noch mehr Themen

Spezielles Error- und Event-Handling, Erweiterungen durch eigene Knoten parallele Verwendung von OpenGL