
OpenInventor

Inhalt

1. Einführung
2. Struktur
3. Component Library
4. Kamera
5. Licht
6. Text
7. Formen
8. Kurven & Flächen
9. Oberflächenstruktur

1. Actions
2. Events & Selections
3. Sensoren
4. Engines
5. Draggers & Manipulators
6. FileFormat
7. Amira
8. Zukunft
9. Quellen

1. Einführung

1.1 Bedeutung

- für Grafik-Programmierer und Anwendungsentwickler
- objektorientierter 3D-Toolkit
- Bibliothek von Objekten und Methoden um interaktive 3D Grafik-Anwendungen zu schaffen

1. Einführung

1.2 Basis

- in C++ und teilweise C geschrieben
- basiert auf OpenGL und standardmäßig auf Unix
- nutzt Hardware-Grafikbeschleunigung aus bei relativ geringem Programmieraufwand
- 3D interchange file format um 3D-Szenen zwischen verschiedenen Programmen auszutauschen

1.3 Historie

1.3.1 Open Inventor

- um 1988 initiierten Wei Yen und Rick Carey bei *SGI* das IRIS Inventor Projekt
- sollte das Erstellen von 3D-Objekten mit OpenGL vereinfachen
- nachdem für Dritt-Unternehmen lizensierbar, Namensänderung in Open Inventor
- Abspaltung einer Entwicklergruppe, die fort an Open Performer entwickelte
- seit 2000 Open Source License von *SGI*
- bald erster Release von *Systems in Motion* Coin 3D

1. Einführung

1.3 Historie

1.3.2 Coin 3D

- 1995 das erste Grundgerüst für Coin 3D von *Systems in Motion* geschaffen
- nach Zunahme der Ansprüche wurde Open Inventor als Grundlage gewählt
- als Free Edition freie Alternative zu den kommerziellen *SGI's* und *TGS'* Open Inventor Bibliotheken
- fast vollständig kompatibel zur Open Inventor 2.1 API

2. Struktur

2.1 Bestandteile

- database primitives: z.B. shape, property, groups, engine objects
- manipulators: z.B. handlebox, trackball
- components: z.B. material editor, directional light editor, examiner viewer

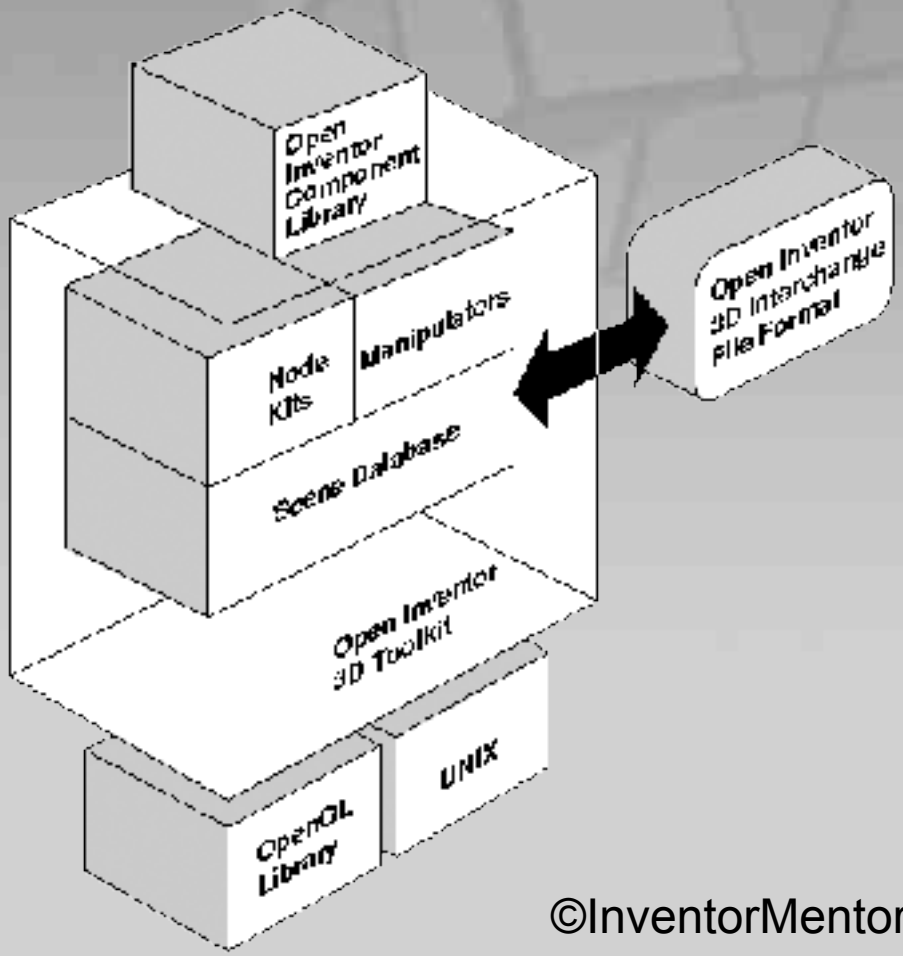
2. Struktur

2.2 Objekt-Funktionen

- Auswahl
- Highlighting
- Manipulierung
- Berechnung der Abgrenzungen
- Laden
- Speichern
- Suche

2. Struktur

2.3 Architektur



©InventorMentor

2. Struktur

2.3 Architektur

- Basis: Open GL Bibliothek und Unix
- darauf aufbauend: Open Inventor 3D Toolkit bestehend aus:
  - Scene Database: enhält Informationen zu allen Objekten wie z.B. Umriß, Größe, Texturen, Farben, Position etc.
  - Node Kits: bieten komfortable Mechanismen um Nodes zu gruppieren
  - Manipulators: Komponenten zum Verändern der Objekte

2.4 SceneGraph

- 2.4.1 SceneGraph und SceneDatabase
  - node ist die Basiseinheit für SceneGraphs (ein root)
  - SceneGraphs werden in SceneDatabase SoDB gespeichert
  - database primitives (alles Knoten des SceneGraphs) sind:
    - shape nodes (Kugel)
    - property nodes (Material)
    - group nodes (Seperator)
    - engines (Rotation)
    - sensors (Ereignisregistrierung)
  - SceneGraphs repräsentiert die komplette 3D-Szene

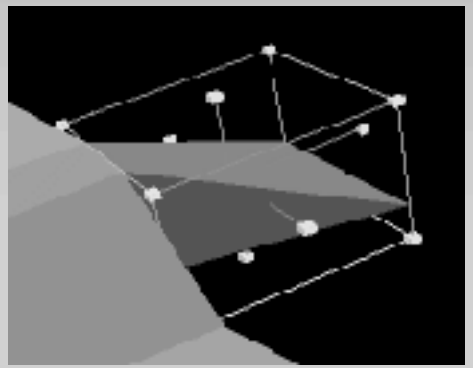
2.4 SceneGraph

- 2.4.2 Node Kits
  - ermöglichen den Aufbau einer strukturierten, konsistenten Datenbank mit SceneGraphs
  - Regulieren die Art und die Platzierung der Nodes über ein Template
  - ermöglicht das Erschaffen anwendungsorientierter Objekte und Semantiken

2. Struktur

2.4 SceneGraph

- 2.4.3 Manipulators
  - o ein Node, der vom Benutzer on-the-run verändert werden kann (Bounding Box transformiert ein Objekt per Mouse-Drag)



2.4 SceneGraph

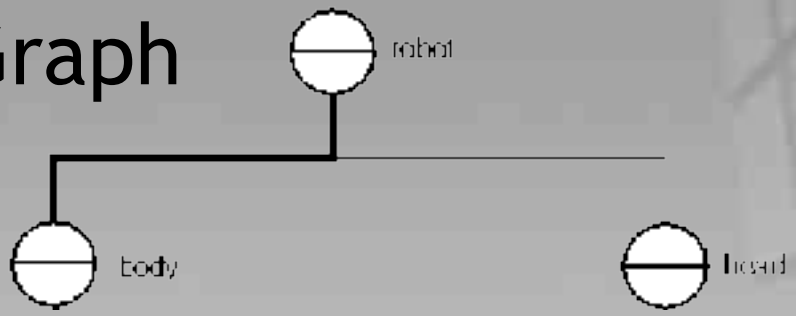
- 2.4.4 Beispiel
  - Pfad, der den linken Fuß des RobotMans repräsentiert
  - enthält auch shared instancing

2. Struktur

2.4 SceneGraph

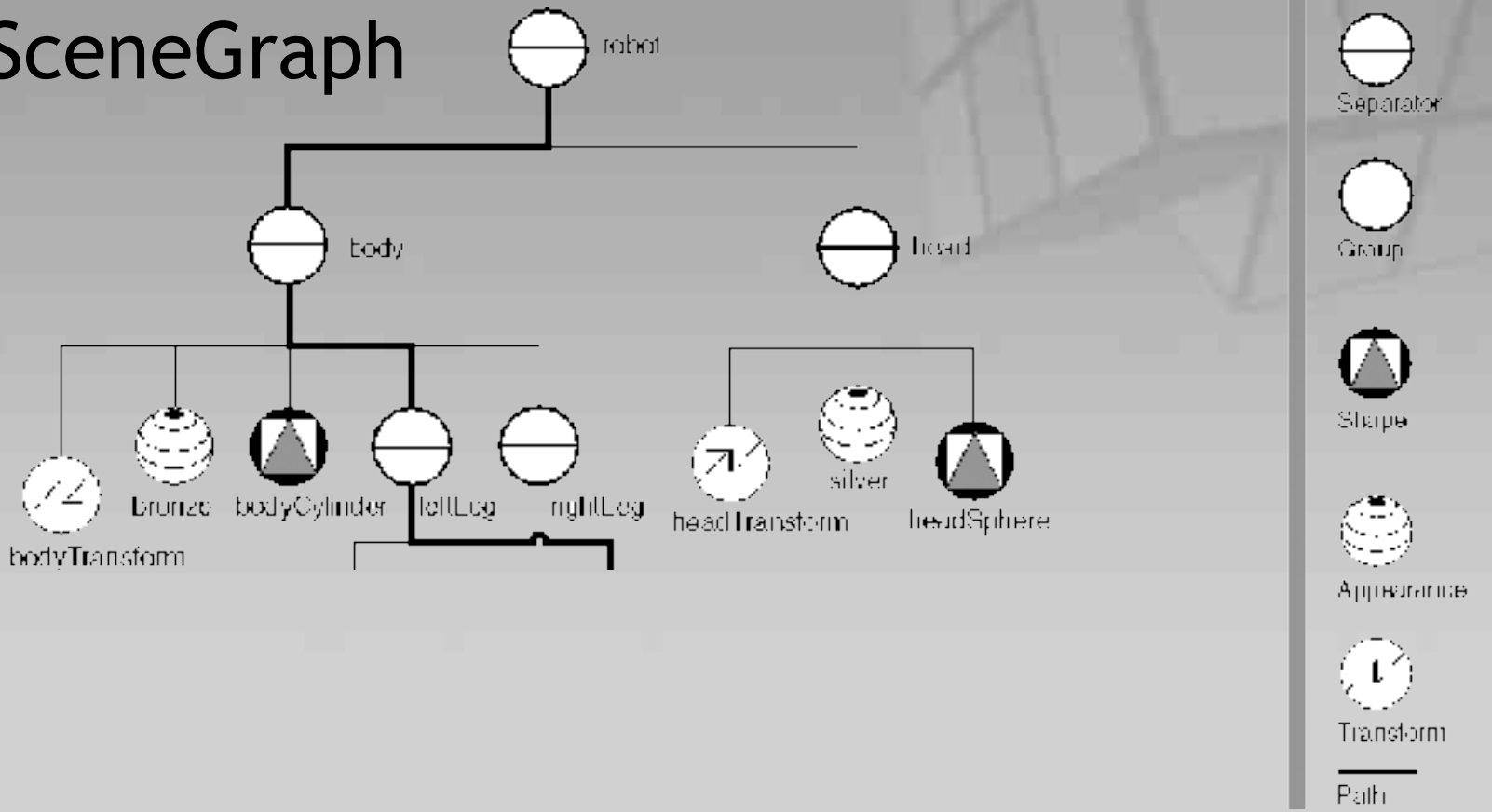
2. Struktur

2.4 SceneGraph



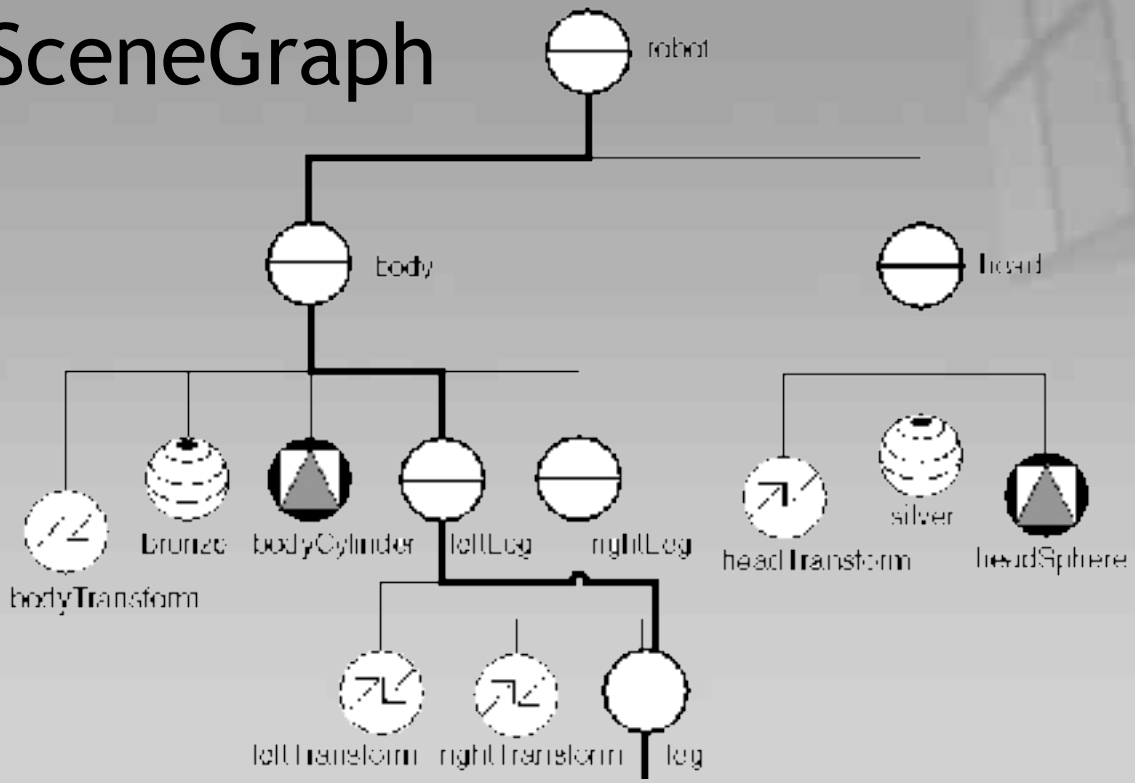
2. Struktur

2.4 SceneGraph



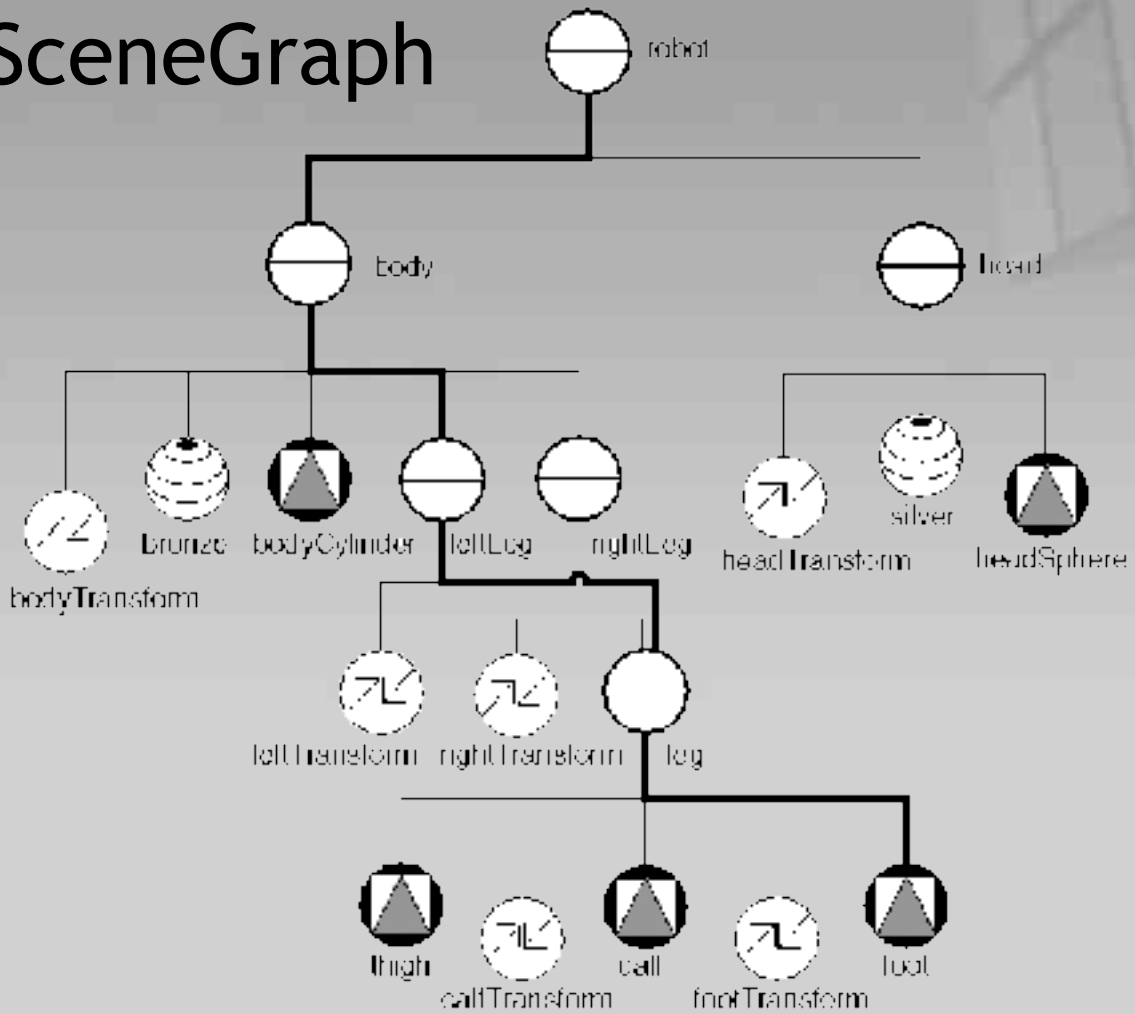
2. Struktur

2.4 SceneGraph



2. Struktur

2.4 SceneGraph



©InventorMentor

2. Struktur

2.5 Implementierung von Open GL

- Wird fürs Rendern benutzt
- in Open GL ist der Rendervorgang normalerweise explizit (framebasiert)
- in Open Inventor ist das Rendervorgang Bestandteil der Objekte
  - o kein direkter Zugriff auf den Frame-Buffer möglich
  - o eine seperate Render-Funktion initiiert den Rendervorgang

3. Component Library

3.1 Bestandteile

- unterstützt unterschiedliche window-systems und integriert das X Window System
- dazu gehören:
  - o main loop und Methoden zur Initialisierung
  - o Viewers
  - o render area (window)
  - o event translator utility
  - o Editoren

3. Component Library

3.2 Aktionsauswertung

- render area empfängt eine Aktion
- render area übersetzt sie für Inventor
- render area leitet die Aktion an Objekte und manipulators weiter

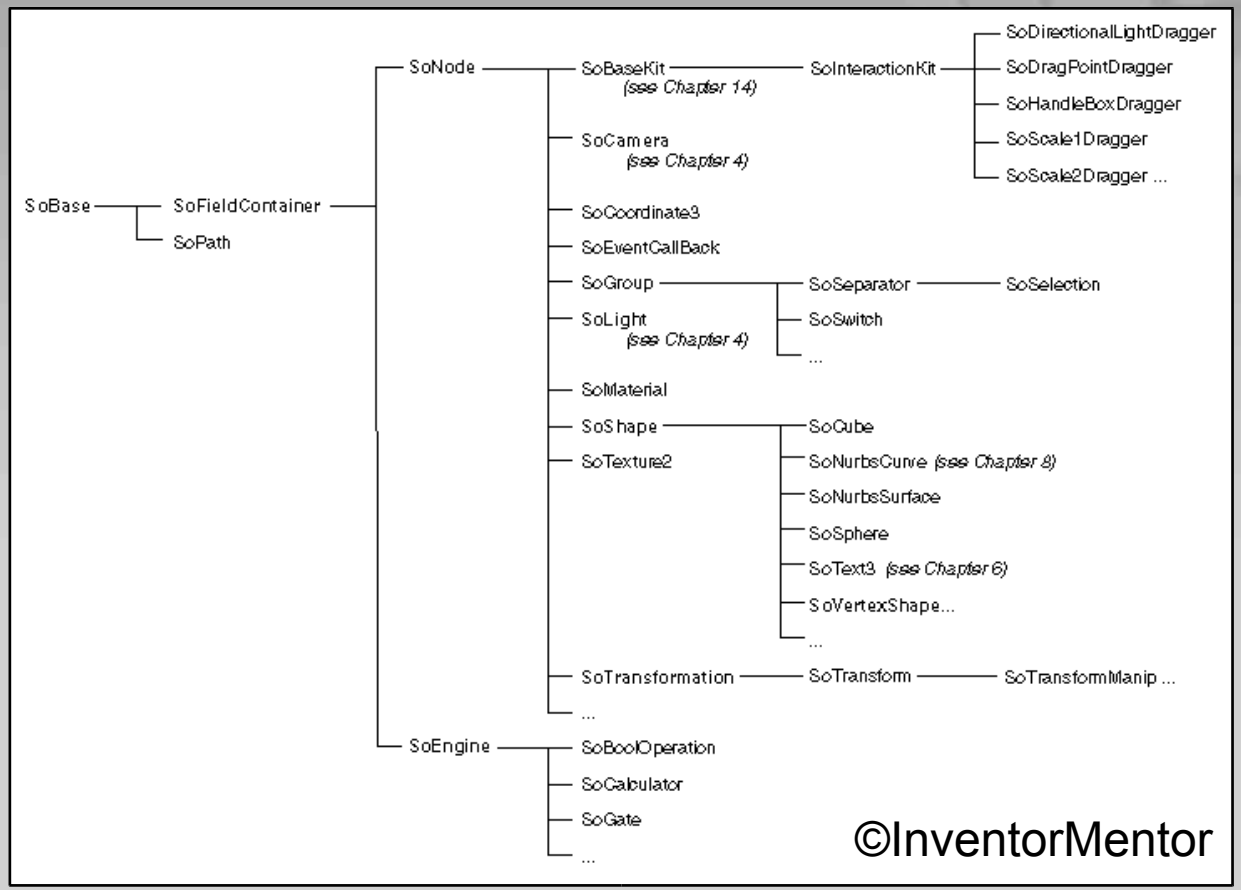
3. Component Library

3.3 Editoren

- die Component Library enthält auch viewers und editors, die sowohl eine render area als auch ein user interface enthalten
- Beispiele: directional light/material editor, fly/examiner viewer

3. Component Library

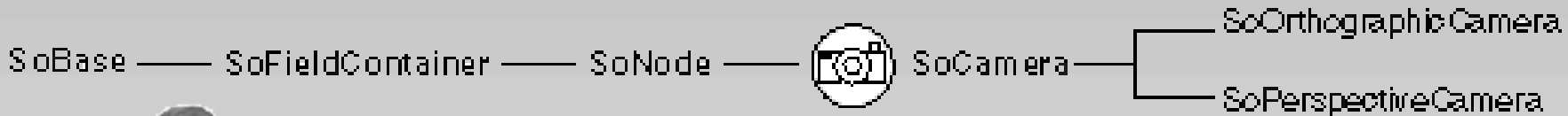
3.4 Class Tree



4. Kamera

4.1 Einführung

- Ohne aktive Kamera ist das Objekt niemals sichtbar
- In einer 3D-Darstellung kann immer nur eine Kamera aktiv sein
- OpenInventor stellt drei verschiedene Kameras zur Verfügung



CameraNodeClasses ©InventorMentor

4. Kamera

4.2 Funktionsweise

I. Während der 3D-Darstellung wird die Kamera im Raum positioniert

II. Abhängig von den Kameraeinstellungen erstellt die Kamera einen Ausschnitt von dem Raum

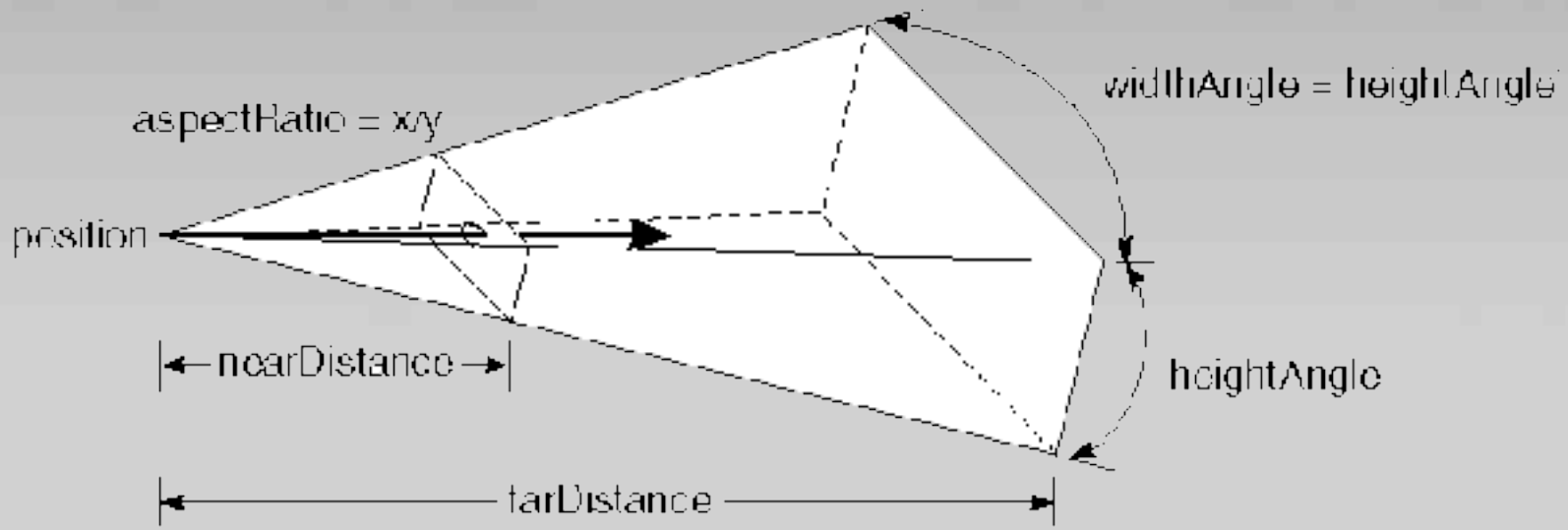
III. Die Kamera erstellt aus der 3D-Darstellung ein 2D-Bild und stellt dieses auf der Graphischen Oberfläche dar

IV. Der Rest des Szenengraphen wird generiert

4. Kamera

4.3 SoPerspectiveCamera

- Simuliert das Menschliche Auge
- Distanzfunktion enthalten

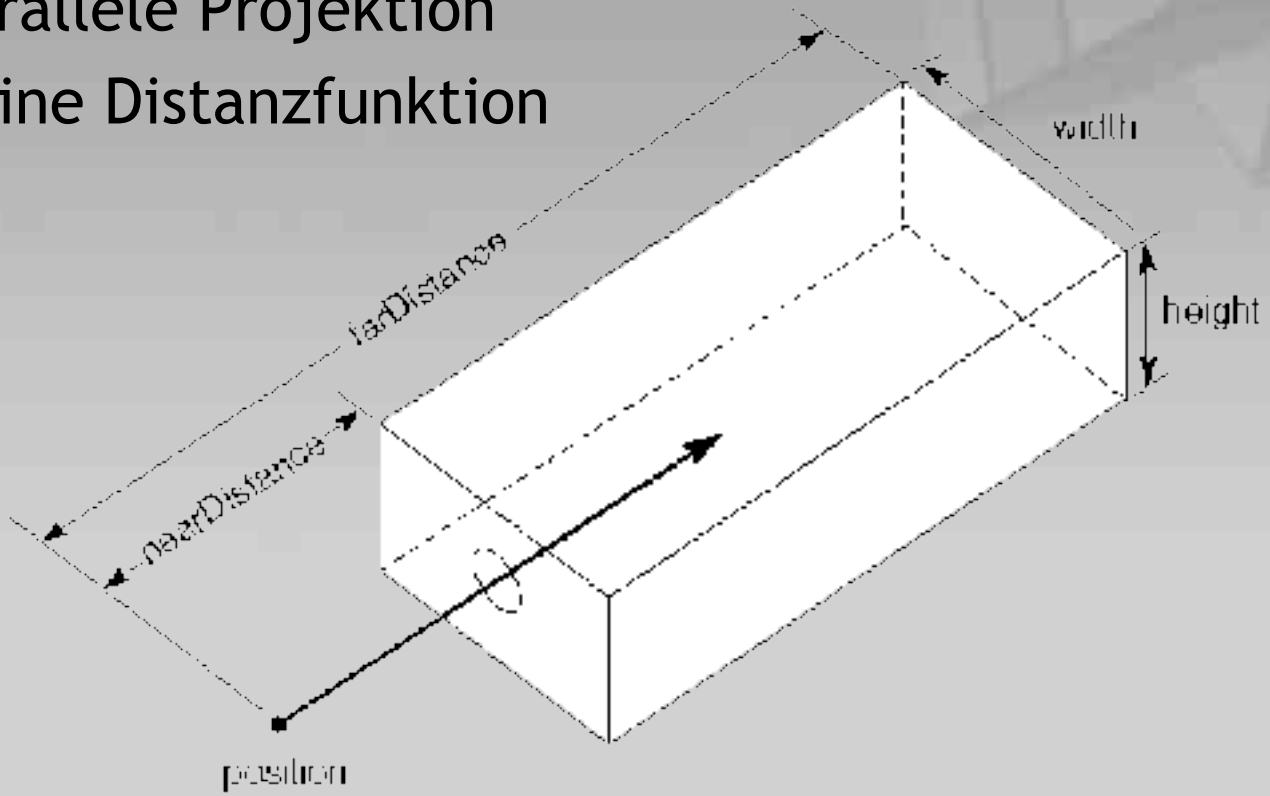


©InventorMentor

4. Kamera

4.4 SoOrthographicCamera

- Parallele Projektion
- Keine Distanzfunktion

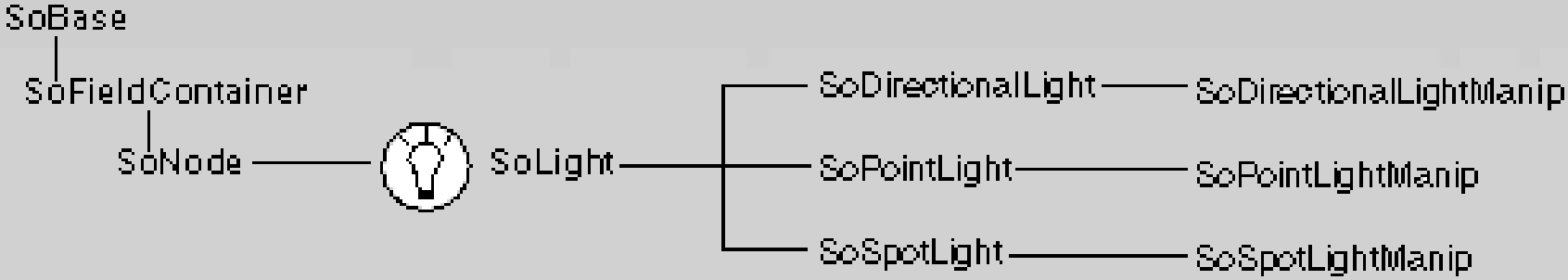


©InventorMentor

5. Licht

5.1 Einführung

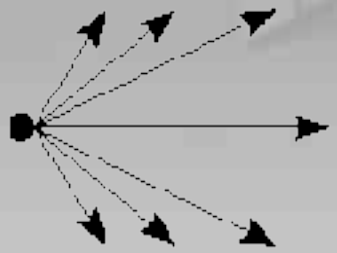
- Mindestens eine Lichtquelle nötig
- Je mehr Licht, desto größer die sichtbare Darstellungsebene
- Maximale Anzahl des Lichts ist abhängig von der OpenGL Implementierung

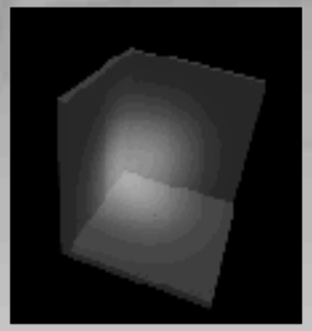


5.2 Lichtarten

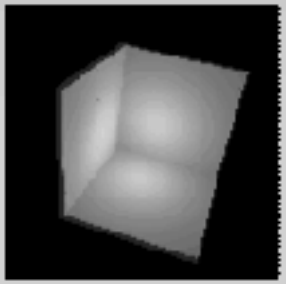
- Drei verschiedene Lichttypen vorhanden

SpotLight

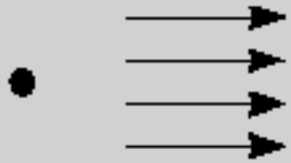


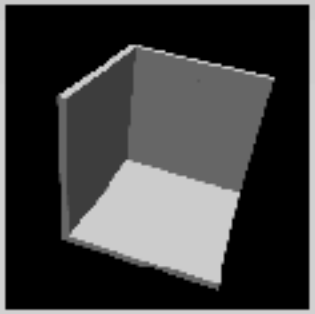


PointLight



DirectionalLight

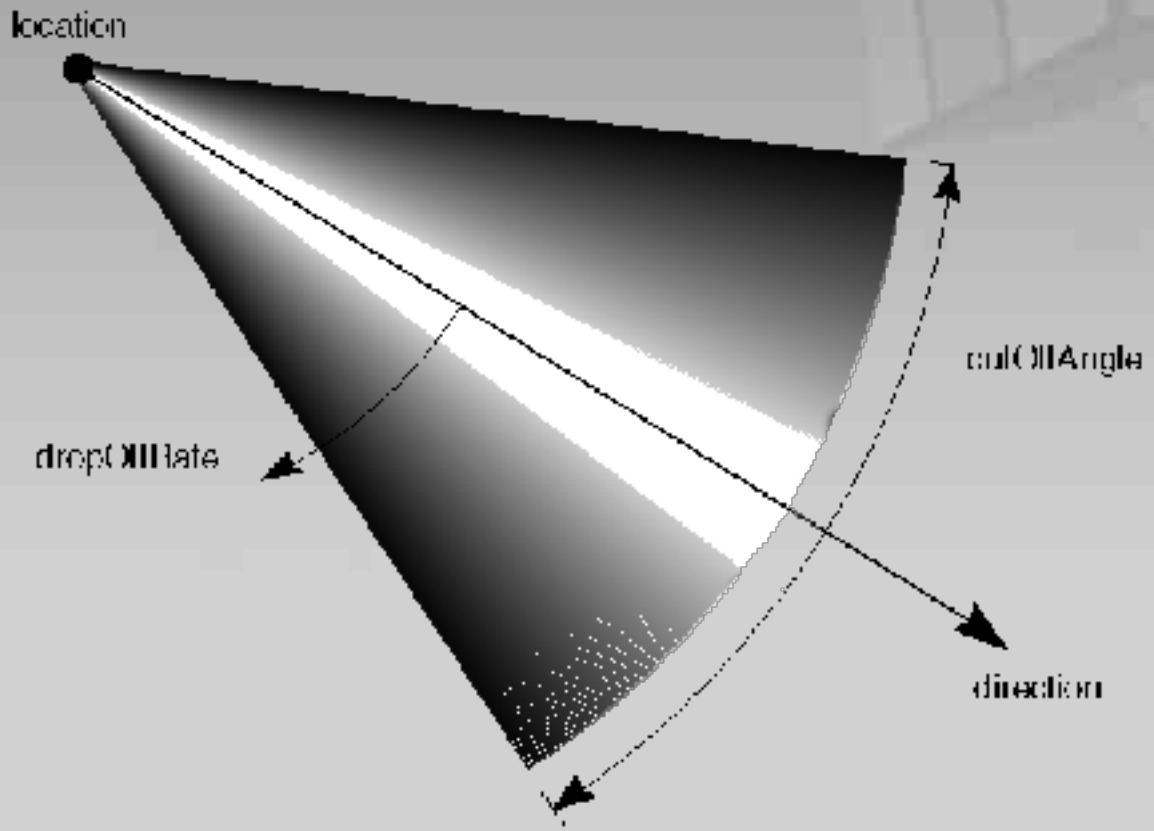




Images ©InventorMentor

5. Licht

5.3 SpotLightAngle



©InventorMentor

6. Text

6.1 Einführung

- Zwei Arten von Textdarstellungen: 2D und 3D
- Schriftart und Größe können durch SoFont geändert werden

Bsp.:

```
SoFont *schriftart = new SoFont;
schriftart->name.setValue(“TimesNewRoman");
schriftart->size.setValue(140);
```

6. Text

6.2 2D - Text

- Bleibt parallel zur Kamera stehen
- Keine Distanzfunktionen
- Schnell zu generieren

Bsp.:

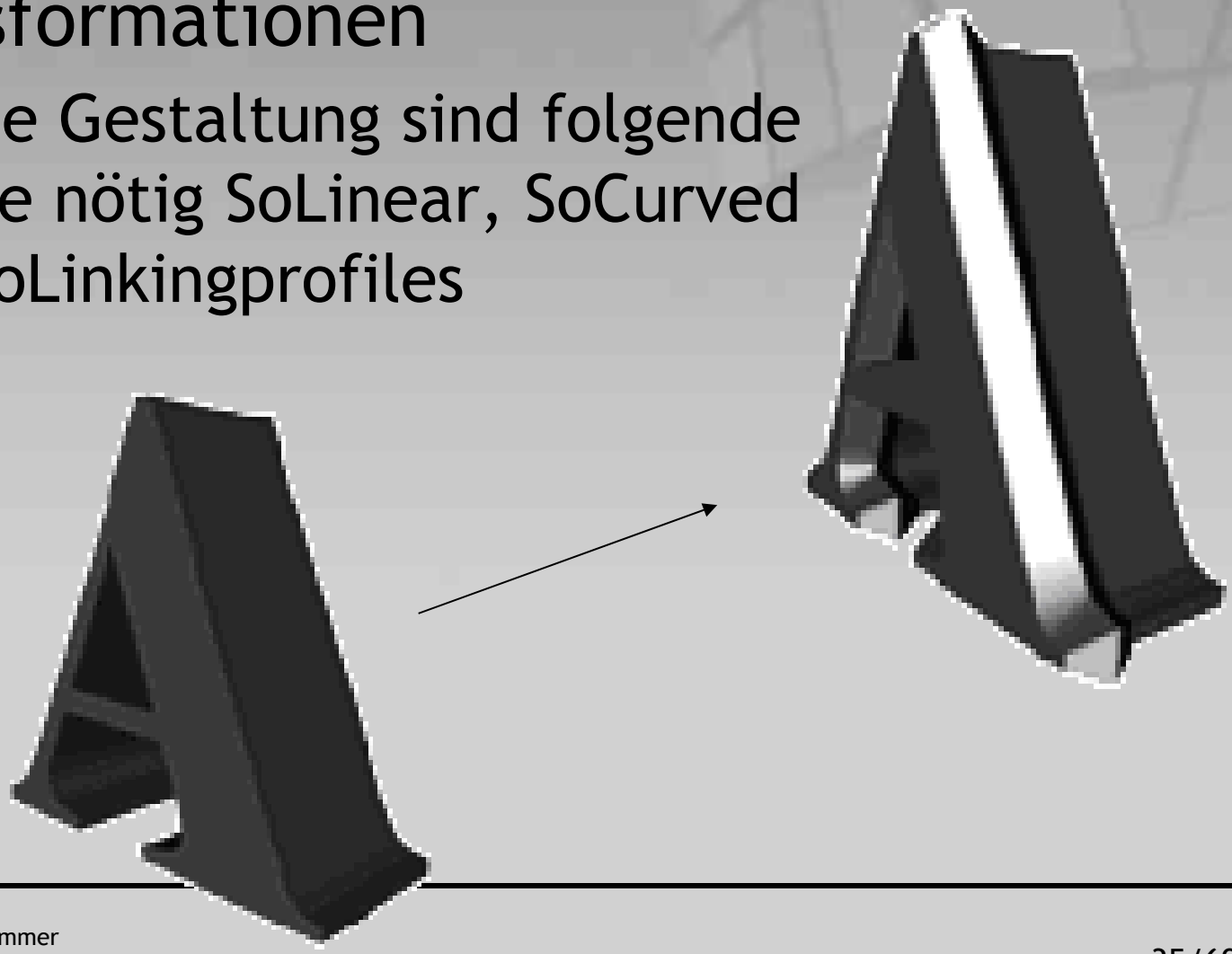
```
SoText2 *2D_text = new SoText2;
2D_text->string = „OpenInventor";
root->addChild(2D_text);
```

6.3 3D - Text

- Dreht sich mit der Kamera
- Hat Distanzfunktionen
- Kann verschiedene Materialien und Oberflächenstrukturen haben
- Transformationen möglich, wie z.B. Höhe, Breite, Tiefe, Gestalt etc.

6.4 Transformationen

- Für die Gestaltung sind folgende Pakete nötig SoLinear, SoCurved und SoLinkingprofiles



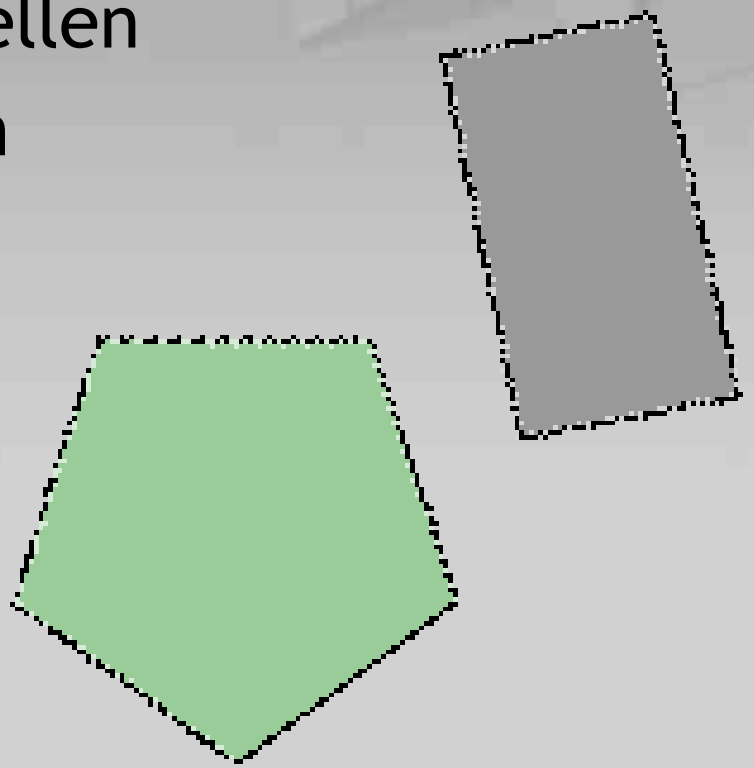
7. Formen

7.1 Einführung

- Alle Formen sind von der SoShape-Klasse abgeleitet
- Folgende simple Forme sind in der Libary vorhanden
  - Rechteck (veränderbar in Höhe, Tiefe, Breite)
  - Kugel (veränderbar beim Radius)
  - Kegel (veränderbar im Radius vom Boden, Höhe)
  - Zylinder (veränderbar in Höhe und Radius)

7.2 Komplexe Formen

- Brauchen zumindest ein Set von Koordinaten um Polygone herzustellen
- Diverse Möglichkeiten
  - o FaceSet
  - o Indexed Face Set
  - o Triangle Strip Set
  - o Quad Mesh



7.3 Beispiel für FaceSet (Obelisk)

- static float vertices[28][3] =

```
{ { 0,30, 0}, {-2,27, 2}, { 2,27, 2}, // vorne
{ 0, 30, 0}, {-2,27,-2}, {-2,27, 2}, //links
{ 0, 30, 0}, { 2,27,-2}, {-2,27,-2}, //rechts
{ 0, 30, 0}, { 2,27, 2}, { 2,27,-2}, //hinten
{-2, 27, 2}, {-4,0, 4}, { 4,0, 4}, { 2,27, 2},
{-2, 27,-2}, {-4,0,-4}, {-4,0, 4}, {-2,27, 2},
{ 2, 27,-2}, { 4,0,-4}, {-4,0,-4}, {-2,27,-2},
 { 2, 27, 2}, { 4,0, 4}, { 4,0,-4}, { 2,27,-2} };
```

7.3 Beispiel

- SoNormal (definiert die Werte die benutzt werden)
- SoNormalBinding (Art der Verbindungen werden definiert)
- SoMaterial (erstellt eine Oberflächentextur für die Form)
- SoCoordinate3 (definiert die Koordinaten für die Eckpunkte)
- SoFaceSet (definiert die Art der Formerstellung)

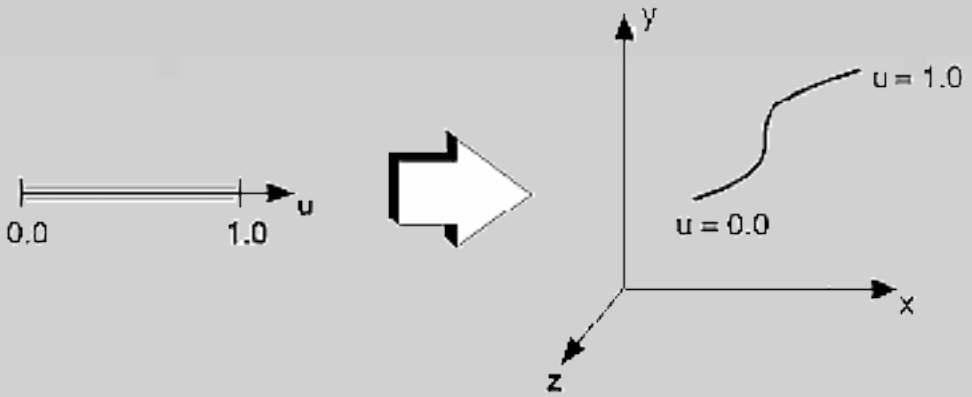
7. Formen

7.4 Eigenschaften

- SoMaterial
- SoDrawStyle
- SoLightModel
- SoEnvironment
- SoShapeHints
- …

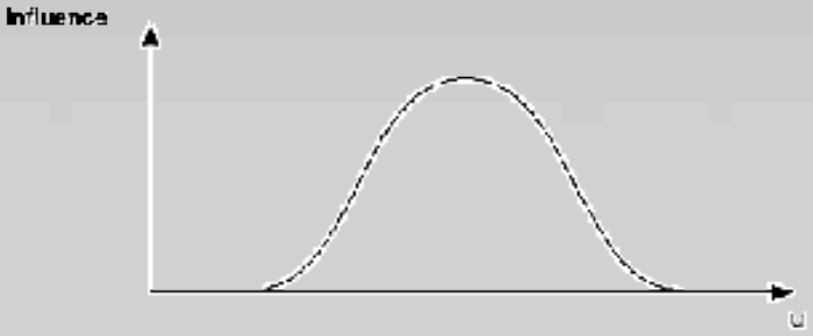
Einführung

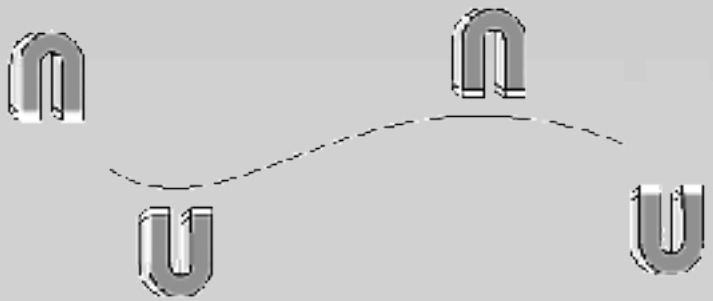
- NURBs sind Non Uniform Rational B-Splines, d.h. Kurven in einem 3D- Raum die stückweise durch rationale Funktionen dargestellt werden können
- Funktionen zur Darstellung der Kurve
  - x = f (u)
  - y = g (u)
  - z = h(u)



Erstellen von Kurven

- Control points - SoCoordinate3, SoCoordinate4
- Kontrollpunkte üben wie ein Magnet Einfluss auf den Verlauf der Kurve aus

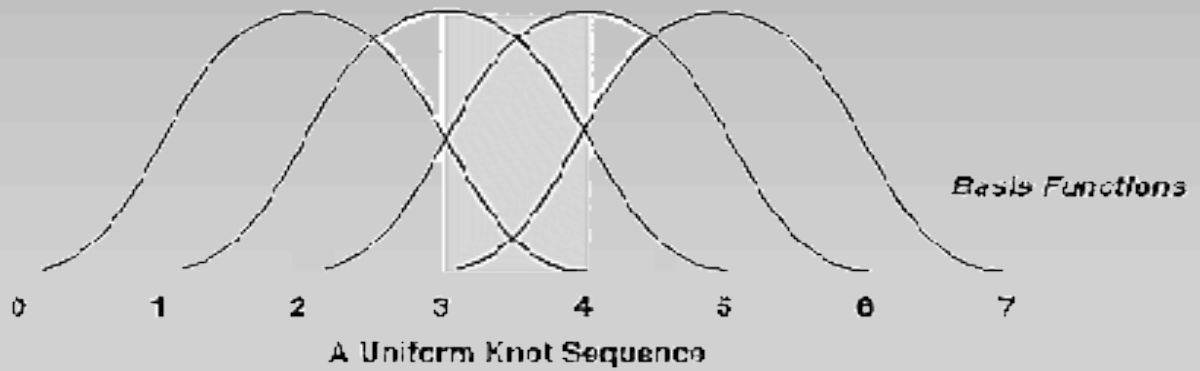


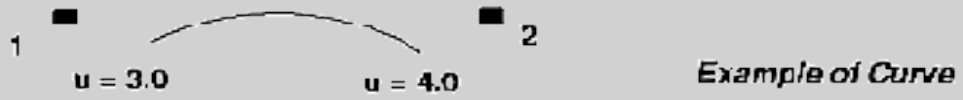


8. Kurven und Flächen

Erstellen von Kurven

- Knot sequence - SoNurbsCurve, SoIndexedNurbsCurve nodes
- Knot sequences definieren, wie die Control points Einfluss auf die Kurve ausüben

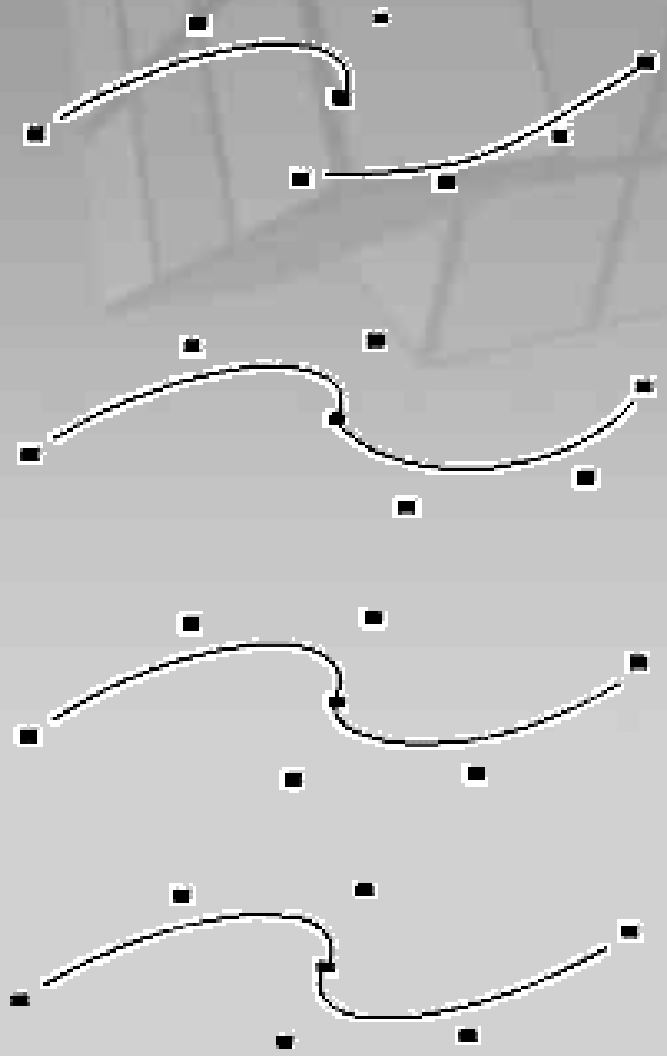




8. Kurven und Flächen

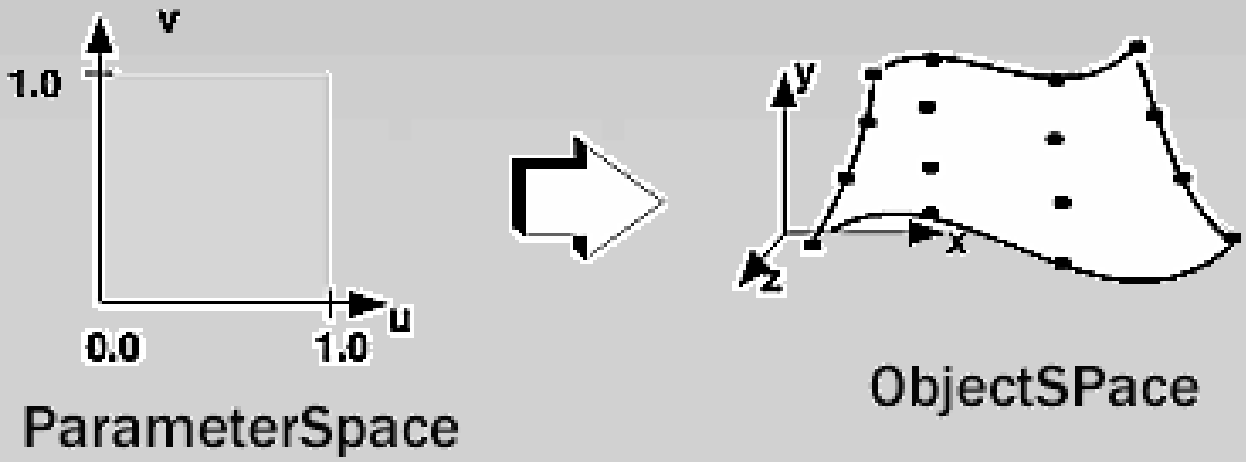
Verbindungen

- Mit Hilfe von breakpoints können Kurven miteinander verbunden werden
- Diverse Möglichkeiten Kurven miteinander zu verbinden



Flächen

- Eine Fläche hat im Gegensatz zur Kurve zwei Richtungen (u und v)
- Ausrichtung und knot sequences müssen für beide Parameter definiert werden



9. Oberflächenstruktur

9.1 Einführung

- Jedes Objekt kann mit einer Oberfläche umhüllt werden
- Eine 2D-Map wird über das Objekt gehüllt, wenn nötig werden Transformationen durchgeführt

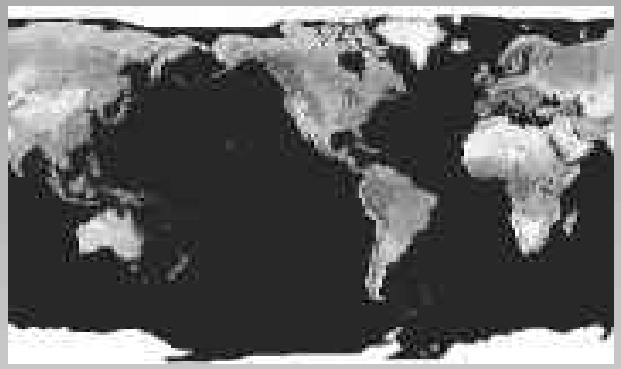
Bsp.:

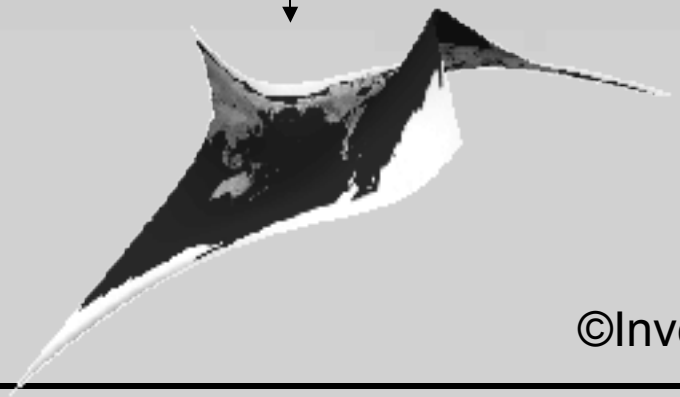
```
SoTexture2 *ziegel = new SoTexture2;
root->addChild(ziegel);
rock->filename.setValue("ziegel_bild.1.rgb");
root->addChild(new SoCube);
```

9. Oberflächenstruktur

9.2 Texture - Beispiele

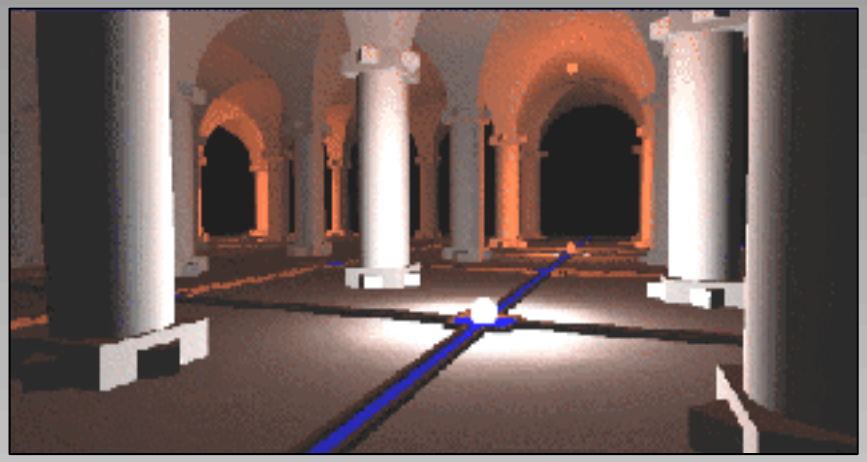
Map



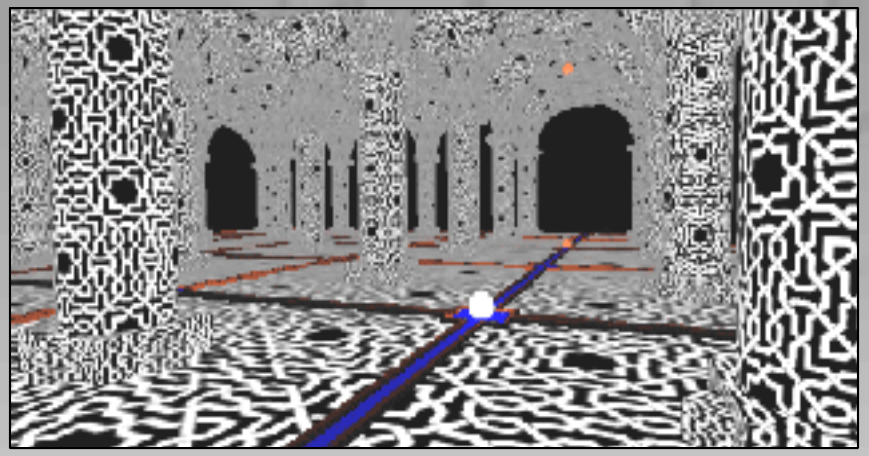


©InventorMentor

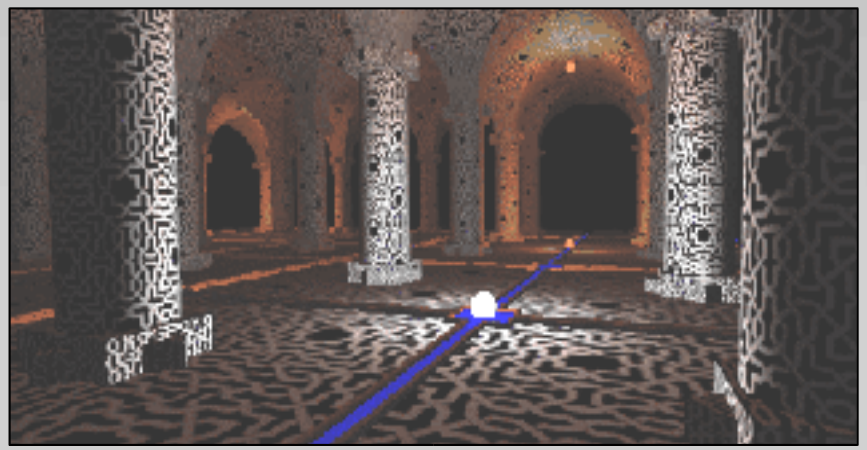
9. Oberflächenstruktur



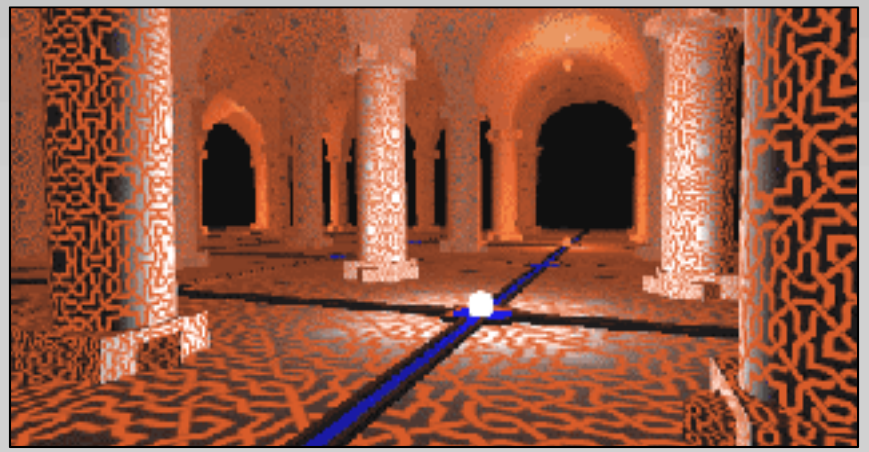
Normal material



Decal



Modulate



Blend

10. Actions

10.1 Einführung

- actions können
  - o Teile oder den ganzen Graphen generieren
  - o Den Graphen als Datei speichern
  - o Nach bestimmten Elementen im Graphen suchen
  - o Informationen über Objekte holen
  - o 3D – Zeichen- Box für Objekte in dem Graphen erstellen
  - o Eigene Actions können erstellt werden
  - o …

10 Actions

10.2 Initialisierung von Actions

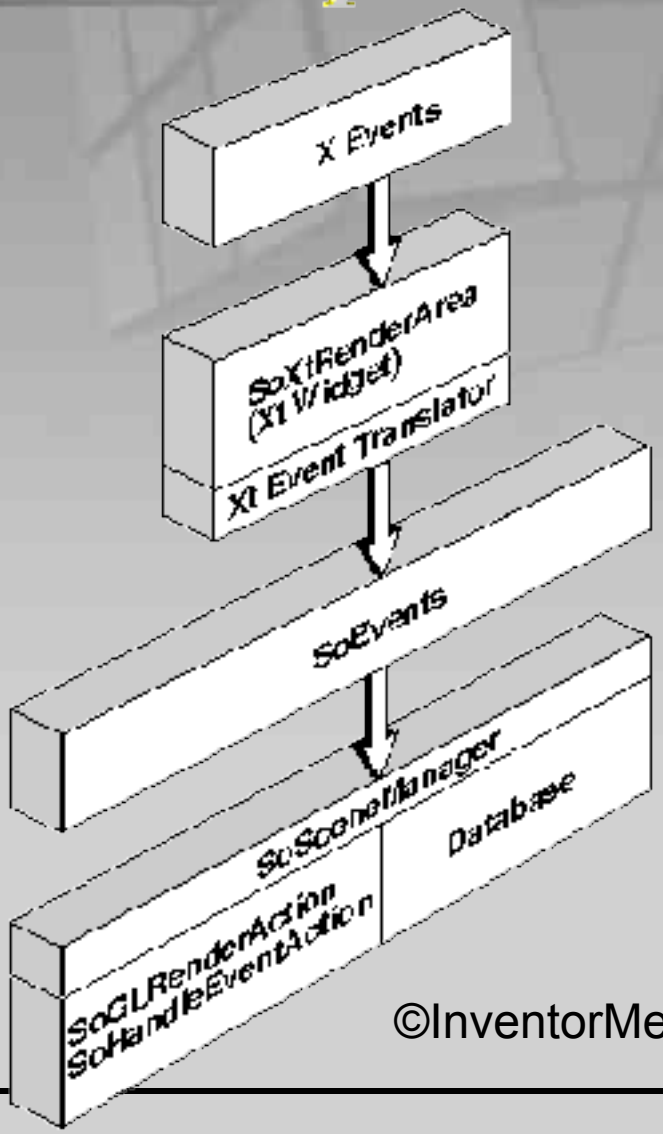
1. Die Actionclass muss Initialisiert werden
2. Setzte Parameter
3. Füge die Actionclass dem gewähltem node hinzu

```
SoWriteAction myAction;

myAction.getOutput()->
openFile("myFile.iv");
myAction.getOutput()->
setBinary(FALSE);

myAction.apply(root);
myAction.getOutput()->
closeFile();
```

11. Events und Selections

11.1 Einführung

- X events sind user interfaces die es ermöglichen den Szenengraphen zu beeinflussen
- RenderArea + Eventtranslator übersetzen das WindowEvent in ein InventorEvent
- SoEvent erkennt die Art des Events
- SoSceneManager übernimmt das Generieren und den Zugriff auf den Graphen



©InventorMentor

11.2 Eigene Events

- Eigene Events können mit Hilfe der CallBack-Funktion generiert werden
- Die eigenen Transformationen oder Funktionen sind im Graphen integriert und werden bei bestimmten Verhalten ausgelöst

  Bsp.:

  SoEventCallback *eventCB = new SoEventCallback;

  eventCB-> addEventCallback
  (SoKeyboardEvent::getClassTypeId(),
  myCallbackFunc, *userData*);

11.3 Selections

- SoSelection ermöglicht
  - Highlighting von Objekten
  - Kann events auslösen bei gewählten Objekten
- SoSelection erstellt eine Liste, in welcher die Verbindungen zu den ausgewählten Objekten gespeichert sind

12. Sensoren

12.1 Einführung

- Spezielle Klasse von Nodes die der Database zugeordnet werden
- Reagieren bei Änderungen in der Database oder bei bestimmten Zeitfunktionen
- Am häufigsten benutzt sind DataSensors und TimerSensors

12.2 DataSensors

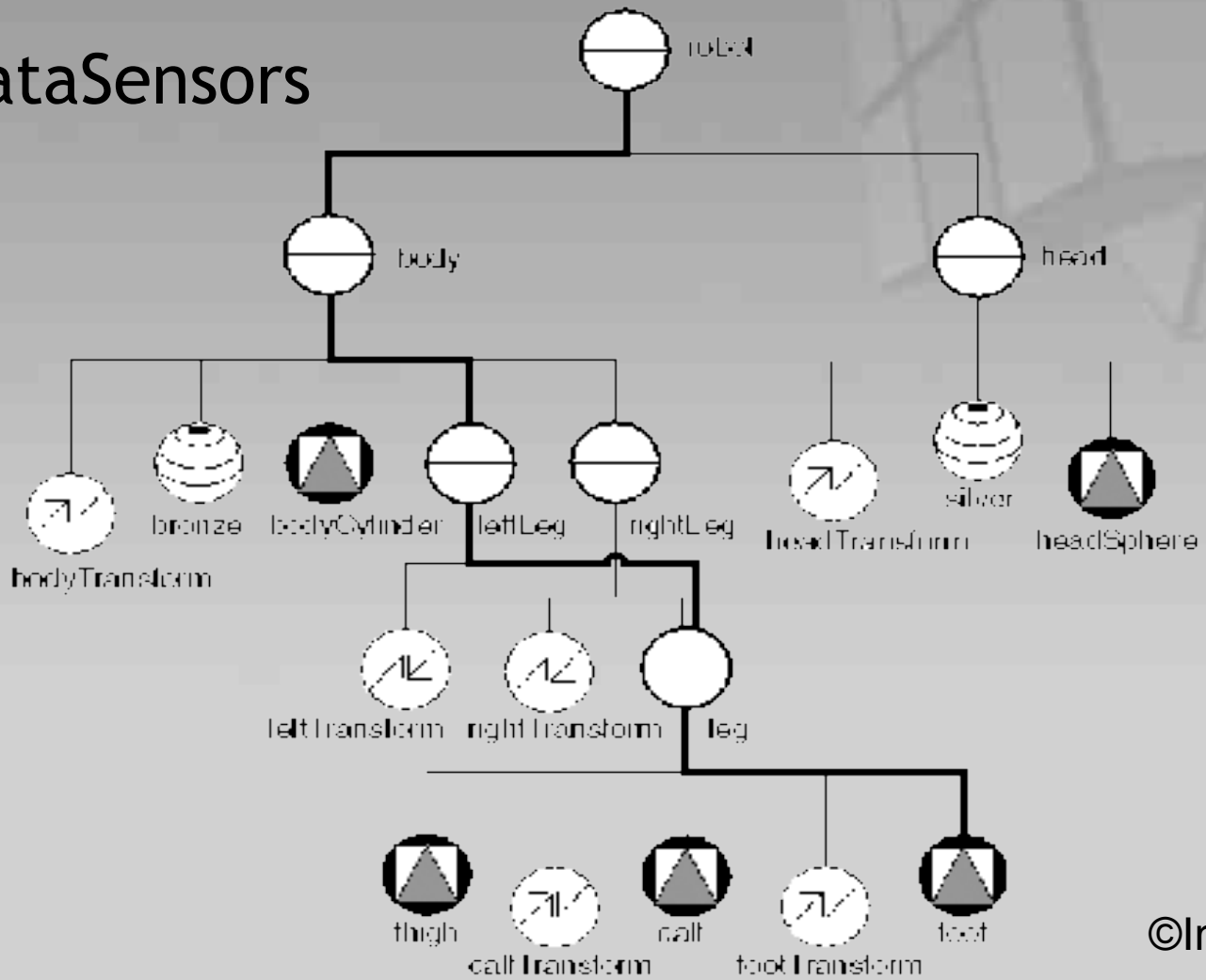
- Drei verschiedene DataSensors
  - **SoFieldSensor**, wird einem Feld zugeordnet
  - **SoNodeSensor**, wird einem Node zugeordnet
  - **SoPathSensor**, wird einer Verbindung zugeordnet

Bsp.:

```
SoFieldSensor *mySensor = new SoFieldSensor
(cameraChangedCB, camera); mySensor-
>attach(&camera->position);
```

12. Sensoren

12.2 DataSensors



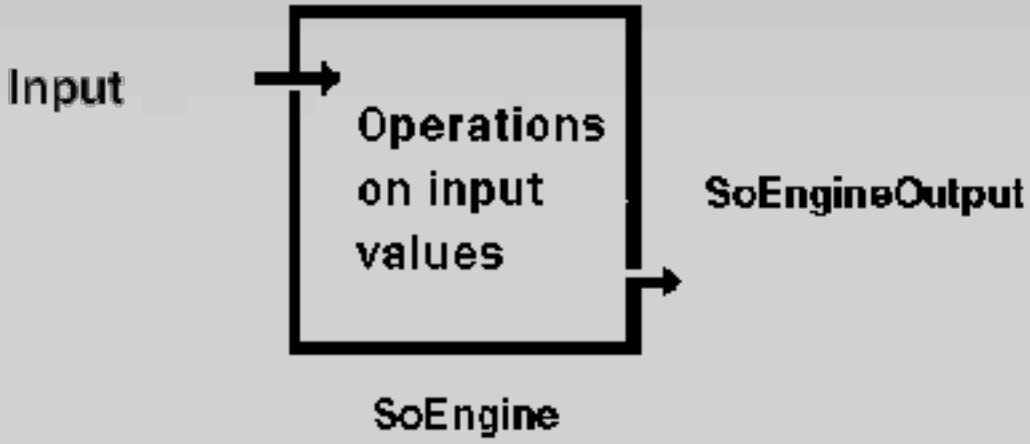
13. Engines

13.1 Einführung

- Engines können Bewegungen in einem Graphen hervorrufen
- Engines können mit anderen Engines verbunden sein, um zu interagieren oder Events auszulösen
- Geometrie und Verhalten der einzelnen Ereignisse werden im Engine-Node gespeichert
- Engines werden in der iv.-Datei gespeichert

13.2 Anwendungen

- Teile vom Szenengraphen zu animieren
- Verschieden Teile des Szenengraphen in Relation zu setzen
- Bei verschiedenen Inputs kann OpenInventor die values für die Operationen umwandeln



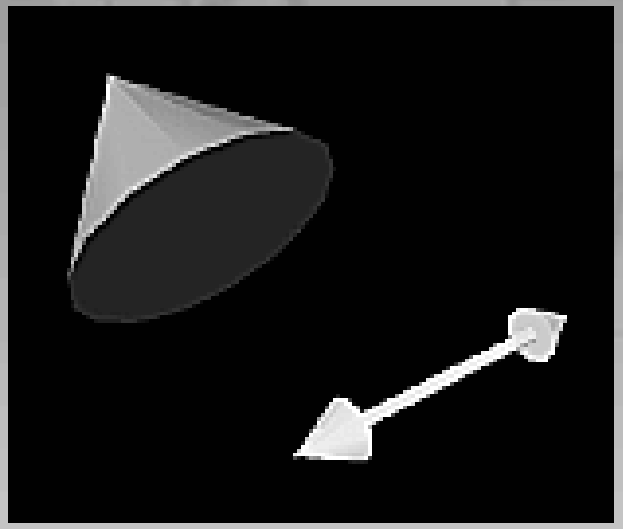
13.3 Engine – Arten

- Arithmetische Engines (Entwerfen, Interpolieren, Boolsche Operationen, Manipulation)
- Animations Engines (Bewegungen die mit Zeitfunktionen verbunden sind)

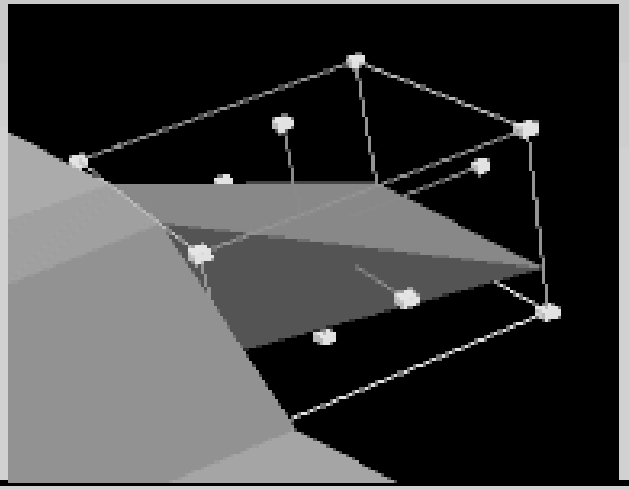
14. Draggers and Manipulators

14.1 Einführung

- Dragger und Manipulator ermöglichen die Transformation von Objekten
- Transformation wird durch ein WindowEvent durchgeführt
- Dragger und Manipulators lösen events aus



©InventorMentor



14. Draggers und Manipulators


```
SoTranslate1Dragger *myDragger = new
SoTranslate1Dragger;
root->addChild(myDragger);
myDragger->translation.setValue(1,0,0);
```

… (Kegelerzeugung)

```
SoDecomposeVec3f *myEngine = new
SoDecomposeVec3f;
myEngine->vector.connectFrom(&myDragger-
>translation);
myCone-> bottomRadius.connectFrom(&myEngine->x);
```

15. File - Format

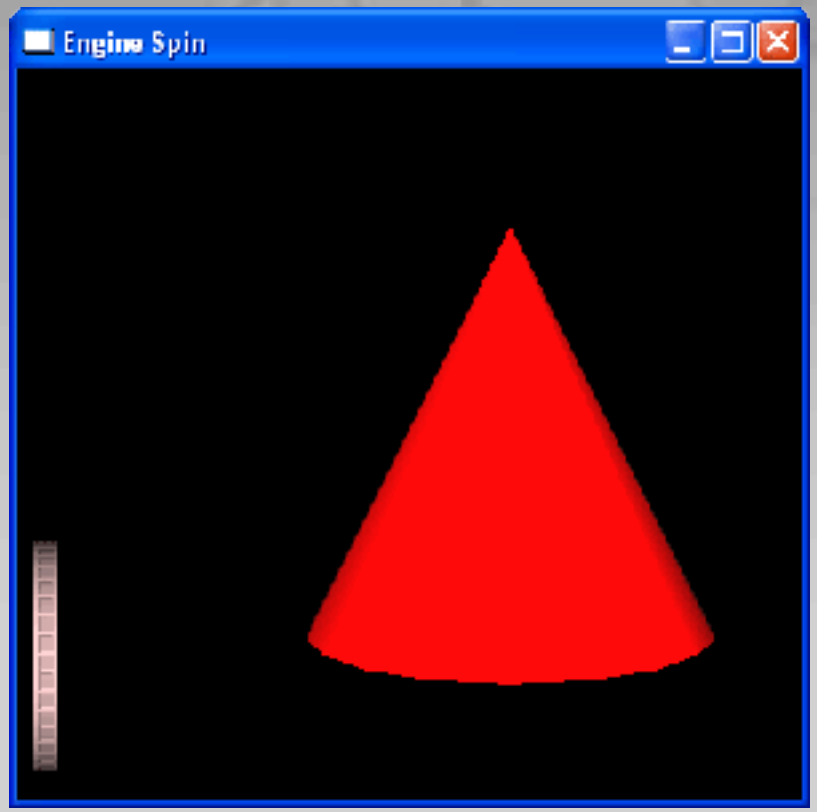
15.1 Einführung

- Der Szenengraph kann als ASCII oder Binäres File-Format (.iv - Format) gespeichert werden
- Theoretisch können alle kompatiblen .iv-Dateien in der OpenInventorDatabase gespeichert und aufgerufen werden
- Das Programmieren der Objekte kann umgangen werden, indem man eine .iv-Datei nach den syntaktischen und semantischen Regeln schreibt

15. File – Format

15.2 Aufbau

```
#Inventor V2.1 ascii

Separator {
  PerspectiveCamera {
    position 0 0 4.3336115
    nearDistance 3.3302779
    farDistance 5.3389449
    focalDistance 4.3336115
  }
  DirectionalLight {
  }
  Material {
    diffuseColor 1 0 0
  }
  Cone {
  }
}
```



16. Amira

16.1 Einsatzgebiete

- modulares, objektorientiertes 3D-Visualisierungs- und Modellierungs-System
- basiert auf *TGS*' Open Inventor bzw. Open Inventor from *Mercury* (Übernahme 2004)
- Einsatz in folgenden wissenschaftlichen Bereichen:
  - Biologie
  - Chemie
  - Maschinenbau
  - Medizin
  - Physik

16.2 Visualisierungen

- basieren auf Polygon-Gittermodellen (drei- oder viereckige Gittermodelle)
- echtzeit-nahes Volume-Rendering
- interaktiver 3D-Viewer
- baumstrukturähnliche Visualisierung der Objektabhängigkeiten
- 3D-Stereo-View

16. Amira

16.3 Features

- Editor für 3D-Modelle
- Oberflächen Reduzierung
- kann aus Bilderstapeln 3D-Modelle erzeugen
- Visualisierungen sind skriptfähig und animierbar
- data sets können simultan, in verschiedenen Viewern und Koordinatensystem in Abhängigkeit zueinander verwaltet werden

17. Zukunft

- *SGI*: seit Release 2.1 keine Weiterentwicklung, kaum Softwarepflege
- *Systems in Motion*: Angleichung an den *SGI* Release 2.1
- *TGS*: Open Inventor *Mercury* Release 5.0

18. Quellen

- The Inventor Mentor, Josie Wernecke, Release 2, 1999, Addison Wesley Publishing Company
- Amira 3.0 User‘s Guide, 2002, Konrad-Zuse-Zentrum für Informationstechnik Berlin
- Open Inventor Manual from TGS
- http://www.computerlexikon.de
- http://www-evasion.imag.fr/Membres/Francois.Faur
- http://oss.sgi.com/projects/inventor/
- http://www.tgs.com
- http://www.coin3d.org

Ende

Danke für Eure Aufmerksamkeit!

Bitte jetzt:

- Aufwachen
- Ab ins Labor
- Ran an die Brillen ;-)


OpenInventor