



---

# VR VO Overview

- Introduction

## Hardware

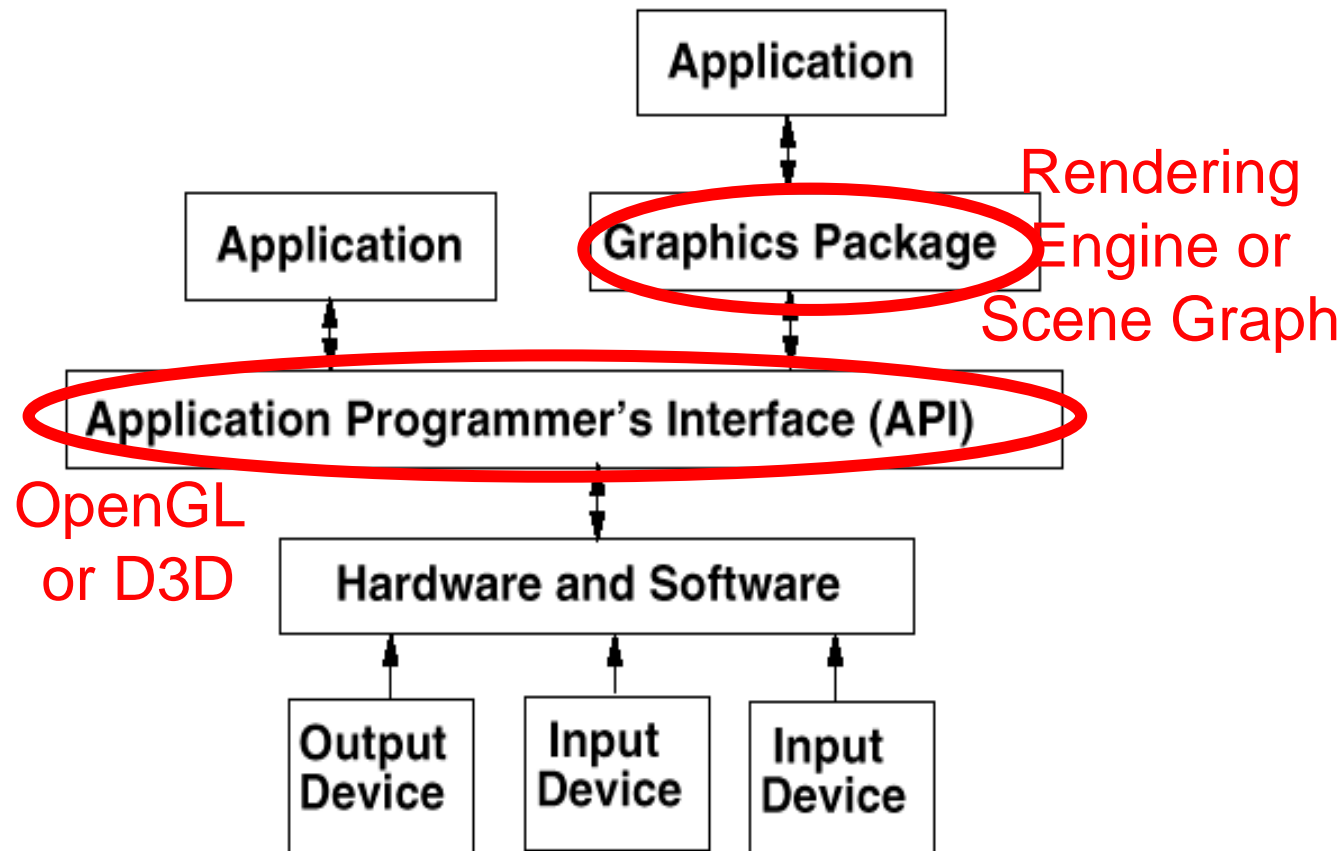
- 3D Graphics Hardware
- Input Devices
- Output Devices

## Software

- High Level Graphics Programming
- Augmented Reality; OpenTracker & Studierstube
- 3D Interaction

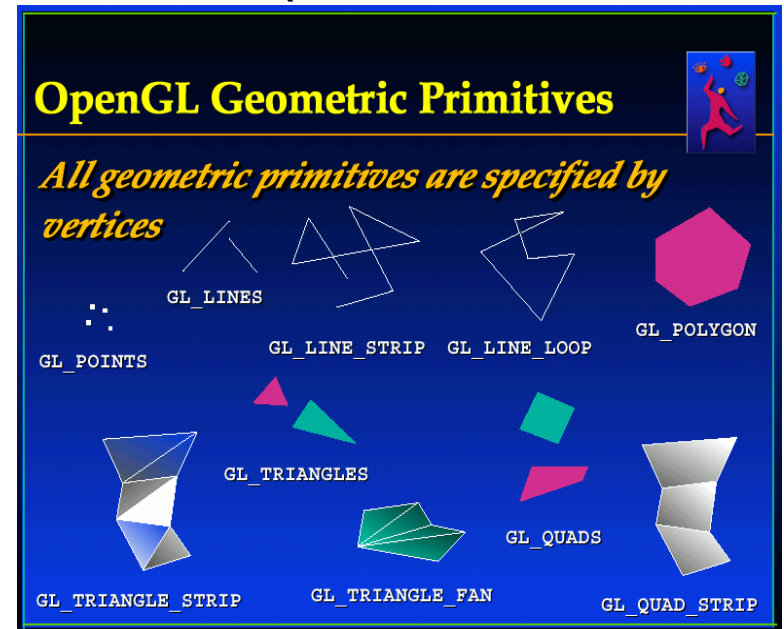
Usability, Evaluations & Psychological Effects

# Application Programmer's View



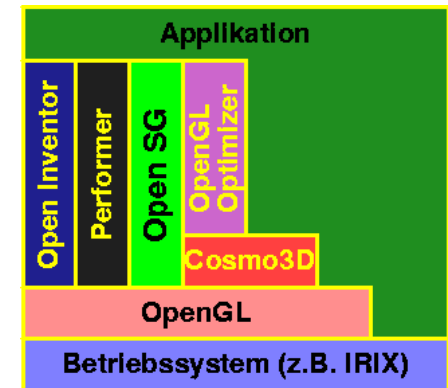
# Low-Level Graphics API

- OpenGL (v 1.0 1992), Direct3D (DirectX 2, 1996)
- Procedural
- Primitives
  - Line, Triangle, ...
  - Color, ...
- Dual Database Problem
  - 1. Representation: Data Objects
  - 2. Representation: Graphical
  - Redundancy, Problem of Inconsistency



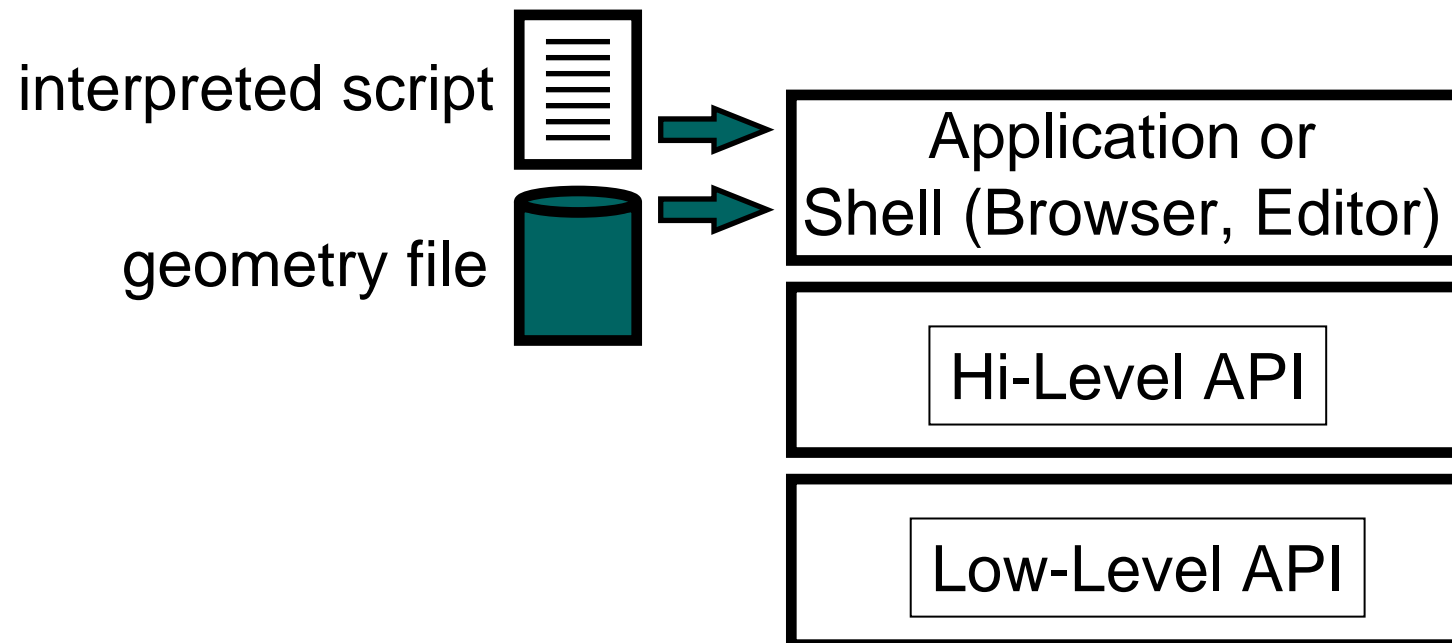
# High-Level Graphics API

- e.g. Inventor, VRML, Java3D, OpenSG, Performer, World Toolkit
- Rendering Engine (e.g. Ogre, Unreal Engine,...) or Scene Graph
- Object oriented
- Scene Objects – “Objects, not Drawings”
  - Not limited to graphical display
- Interactivity: Event-model for 3D scenes
- Software architecture
  - Toolkit-approach, extendible



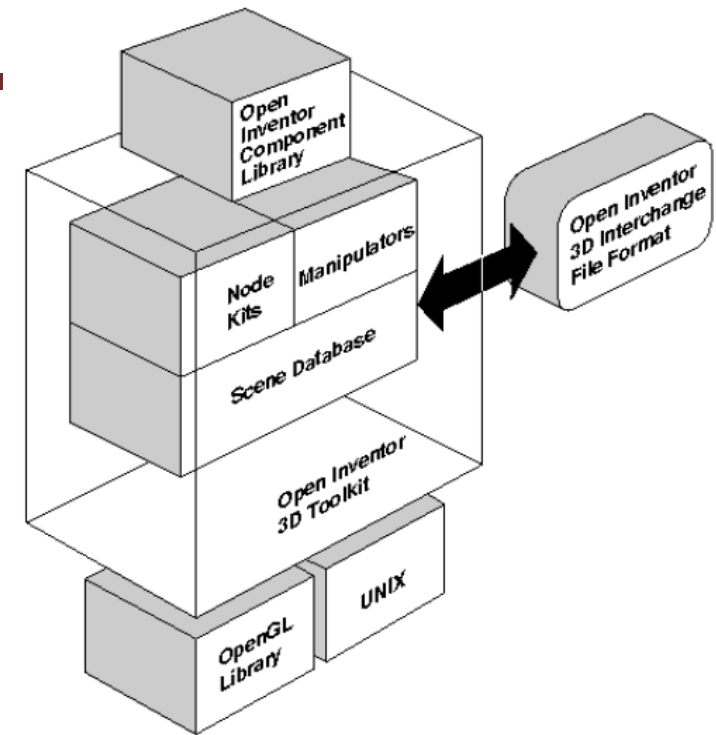
# Why High-Level API?

- Rapid prototyping
- Rapid application development (RAD)



# Open Inventor

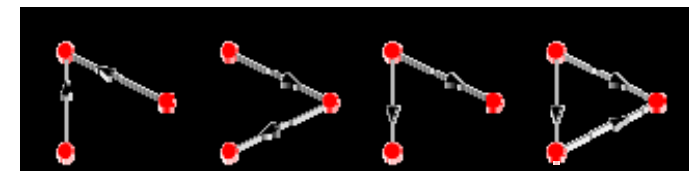
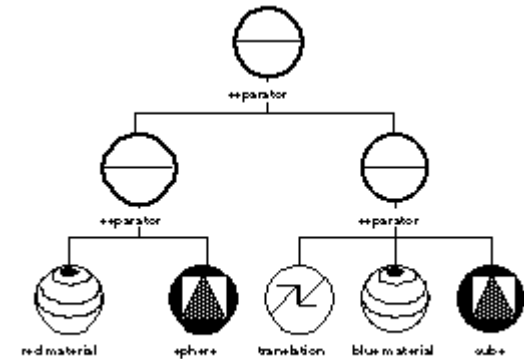
- In our case: Open Inventor API
- Based on C++
- Used in research projects & VRLU
- Window system independant



- 1. Version: SGI Inventor, 1992. Open Source (ver 2.1)
- 2. Version: Mercury (former TGS) Inventor: Continued development of SGI Inventor. Now ver. 6.0
- 3. Version: **Coin** by Systems in Motion (SIM), Re-Engineered API, Open Source; soon ver. 3.0  
<http://www.coin3d.org/>

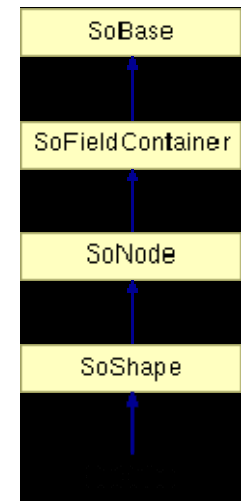
# Scenegraph – Structure

- Graphical data structure = **Scenegraph**
- Scenegraph consists of **Nodes**
- Directed graph! (Head/Tail)  
Directed edges -> **Hierarchy**
- Use of the hierarchy
  - Semantic Hierarchy: e.g. car (parts)
  - Geometric Hierarchy: e.g. puppet / jointed doll
- Usually one tree is sufficient
- General: **Directed Acyclic Graph** (DAG)
  - Multiple parent nodes allowed
  - No directed circles



# Scene graph - Nodes

- Nodes consist of data and methods
- Nodes are of a specific **type**
  - Type determines behavior
  - Behavior = Reaction to certain events
  - Events are generated by the application – by the user -> Interactivity
- Nodes are instances of a **class**
  - Scene hierarchy vs. class hierarchy!
- Flexibility: Choose node(type), compose scene graph with nodes
- Extendible: New nodes can be implemented















# Scene graph - Fields

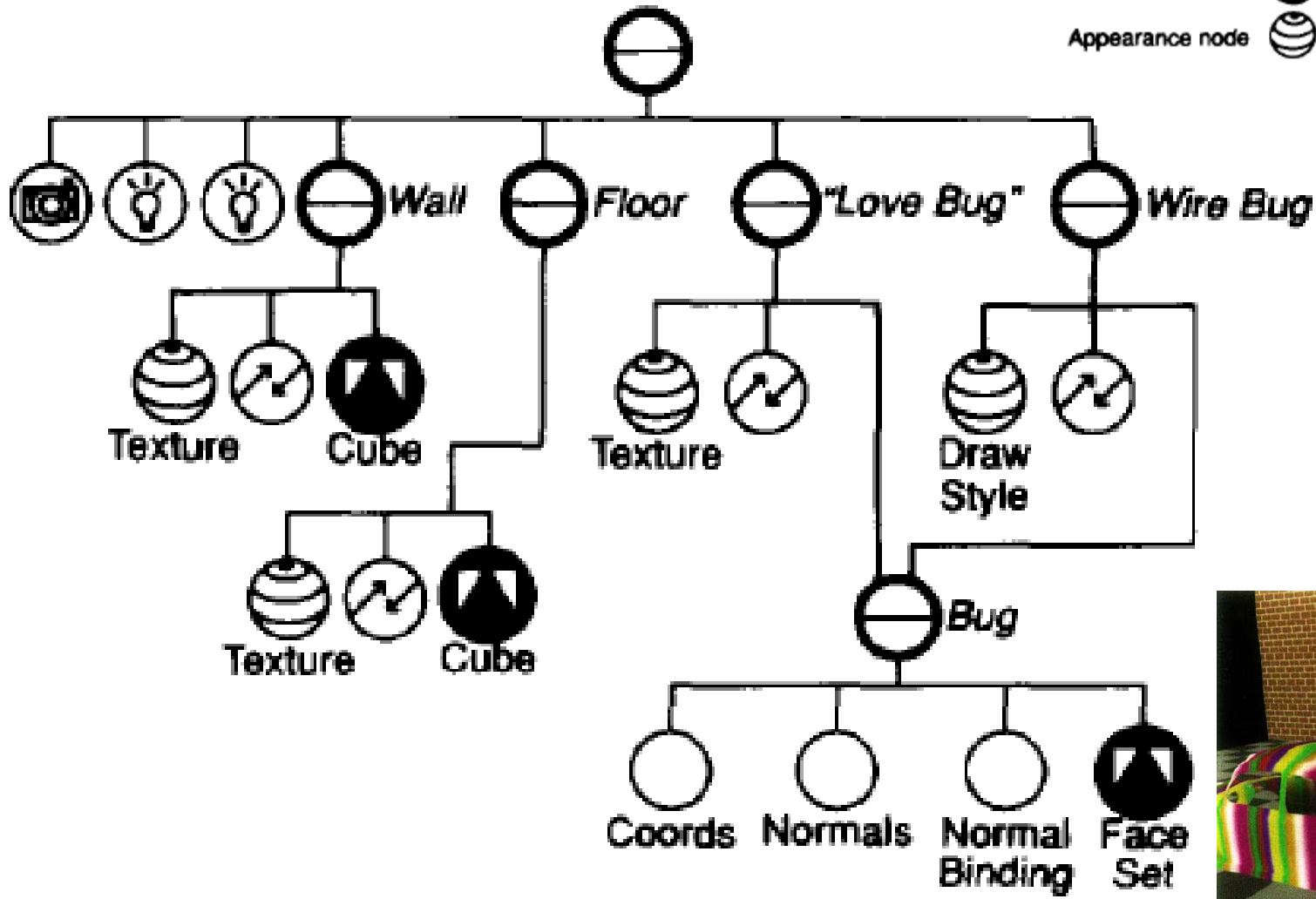
- Attributes (member variables) in nodes are called **fields**
- Fields: set/get, detect changes, connect fields across nodes
- Fields are objects by themselves
  - Float-Object, String-Object etc.
- Advantage: The whole scene graph system is self-describing (reflection)
  - Important property of a database
  - e.g. serialisation, data persistency

## SoMaterial

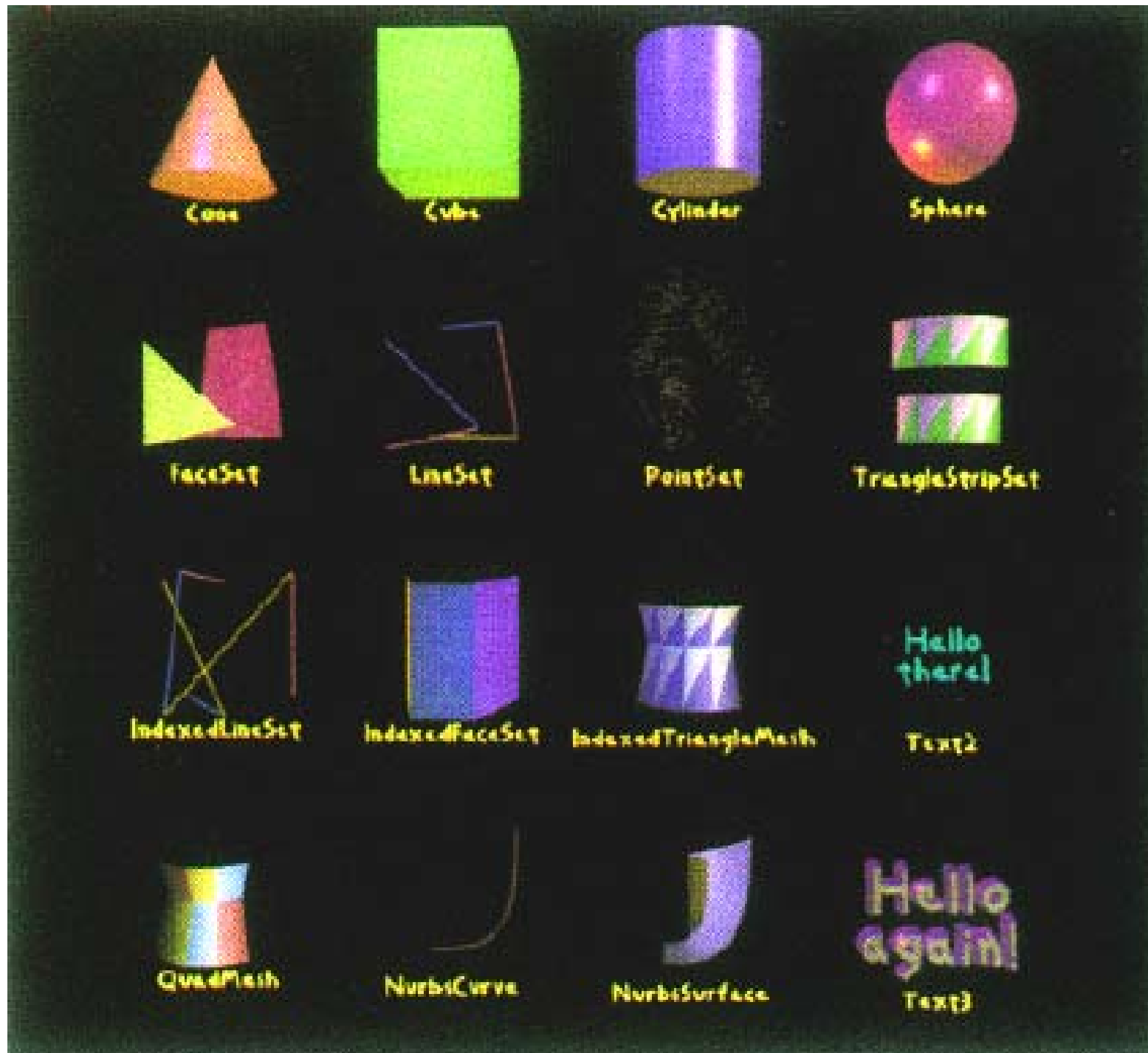
ambientColor  
diffuseColor  
specularColor  
emissiveColor  
shininess  
transparency

# Example

- Separator group node   Transform node
- Other group node   Other property node
- Camera node   Node Kit
- Light node   Manipulator
- Shape node   Sensor
- Appearance node   Subgraph



# Node-Types: Shape Nodes



# Node-Types: Property Nodes



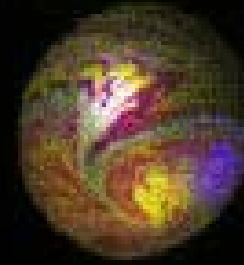
Material



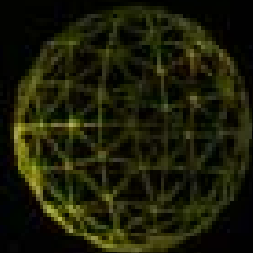
Texture  
Function  
ENVIRONMENT



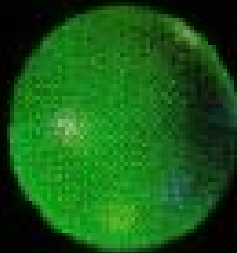
Material  
transparency



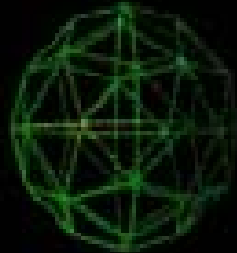
Texture2



DrawStyle  
LINES



Complexity  
0.9



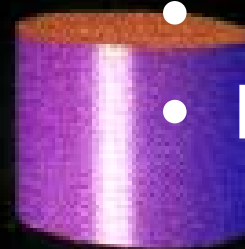
Complexity  
0.3



DrawStyle  
POINTS



LightModel  
BASE\_COLOR



Material  
Binding  
PER\_PART

Additional:

- Transform
- Normal

---

# Node Types: Non-Visuals

## Group Nodes

- Group
- Separator
- Switch
- Selection
- LevelofDetail, LOD
- Array
- MultipleCopy

## Light/Camera Nodes

- OrthographicCamera
- PerspectiveCamera
- (OffAxisCamera)
- DirectionalLight
- PointLight
- SpotLight

others:

File, OffscreenRenderer

---

# Graph Traversal: Basic Idea

- Data structure (scene graph) is processed (=“traversed”)
- For each node a number of methods is implemented:
  - Rendering
  - BoundingBox calculation
  - Transformation matrix calculation
  - Handle Events (e.g. picking)
  - Search nodes
  - Write to file
  - Execute user callback...

---

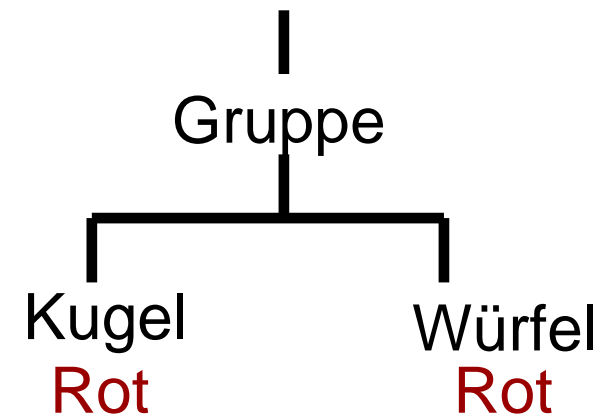
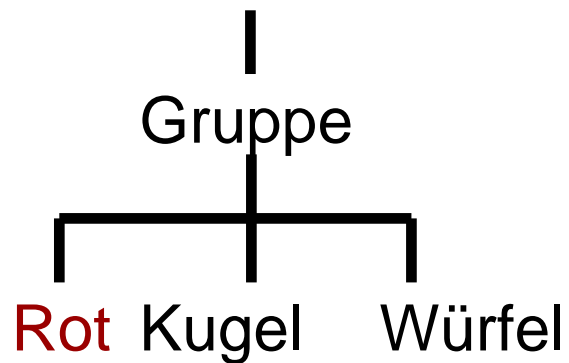
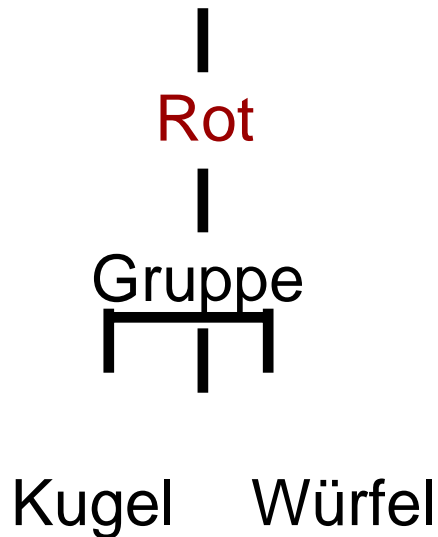
# Graph Traversal Order

- All nodes must be processed
- In general: Depth-First
- Traversal uses a State-Engine
- Difference in Group Nodes
  - Ordered Group
    - State Propagation top->down and left->right
    - e.g. Inventor, VRML
    - Very flexible scene graph generation
  - Unordered Group
    - State Propagation only top->down
    - e.g. Java3D, Performer
    - Parallel Render Traversal possible (Threads, SMP)!

# Graph Traversal

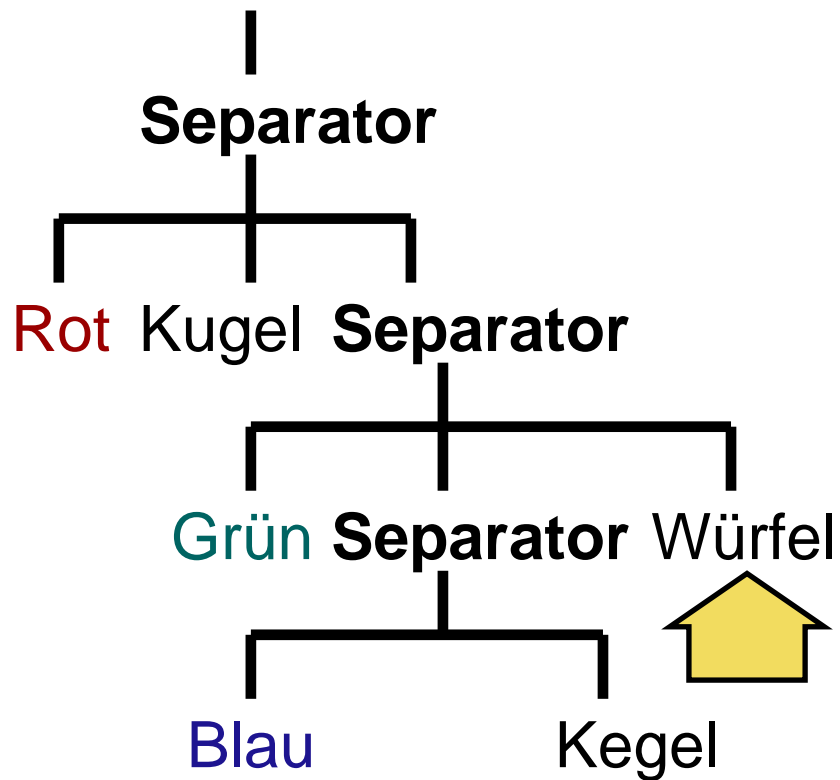
## Modeling Attributes

- In-between, leaves or fields?

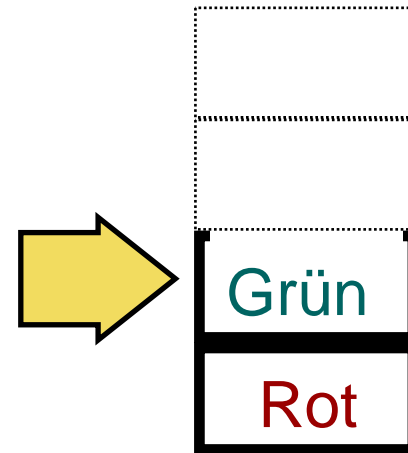


Some toolkits only allow specific structures  
e.g. VRML97 Shape & Appearance combined

# State, Stack & Separator

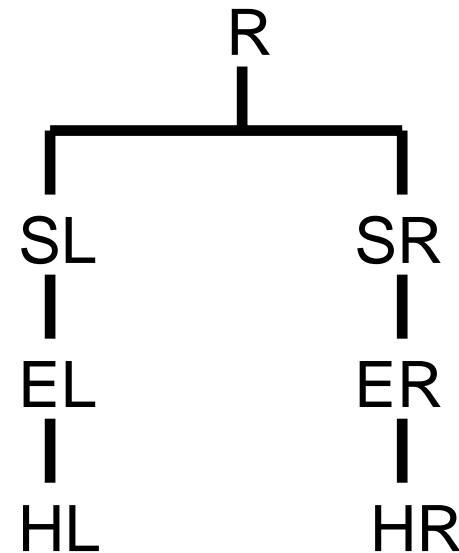
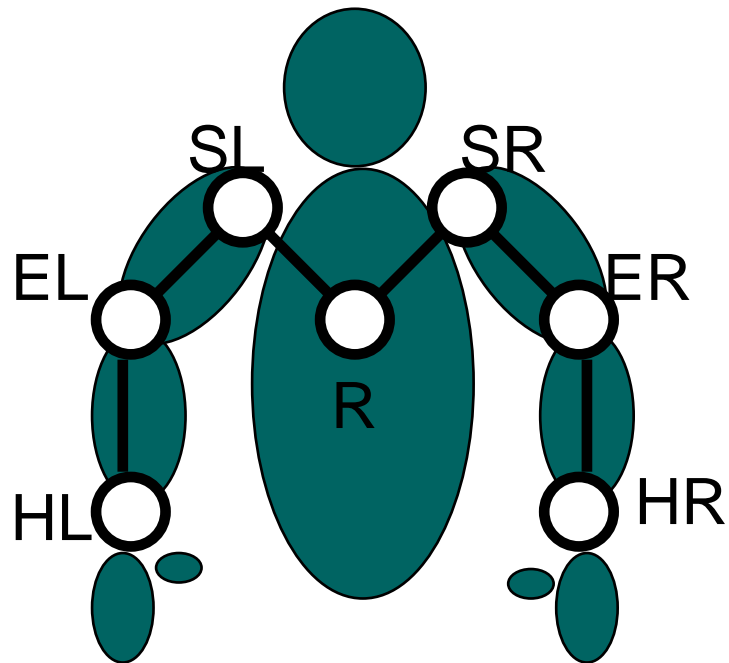


Color state stack



Traversal saves state in Stack

# Transformation-Hierarchy



OpenGL Matrix Stack <--> Transformation hierarchy

---

# Paths

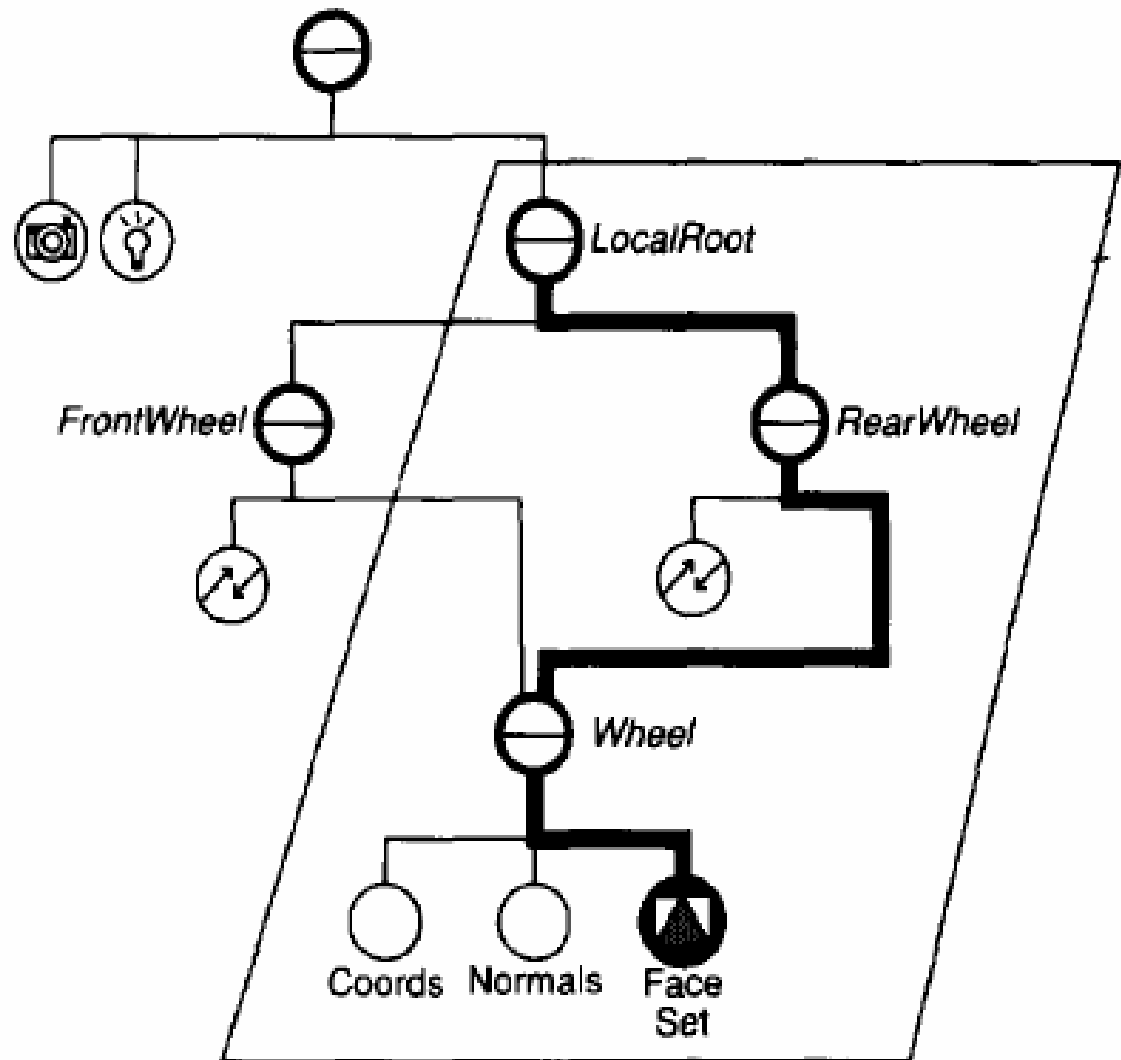
- Structures, that have been defined (DEF), can be reused (USE)
- Objects are simply included at a different position (with different transformation) in scene graph
- With **Paths** objects can be uniquely specified

# Instancing (Re-use)

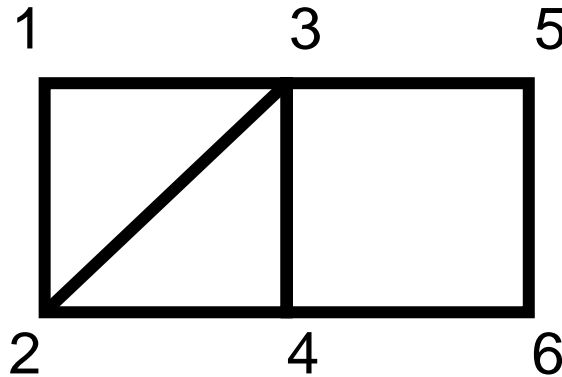
Example: Car

In case of textures:

- Saves texture memory



# Polygonal Shapes: Coordinates



Indexed vs. non-indexed polygon lists (e.g. FaceSet):

**Non-Indexed:**

$V = \{P1=(x1,y1,z1), P2, P3, P2, P3, P4, P3, P4, P5, P6\}$

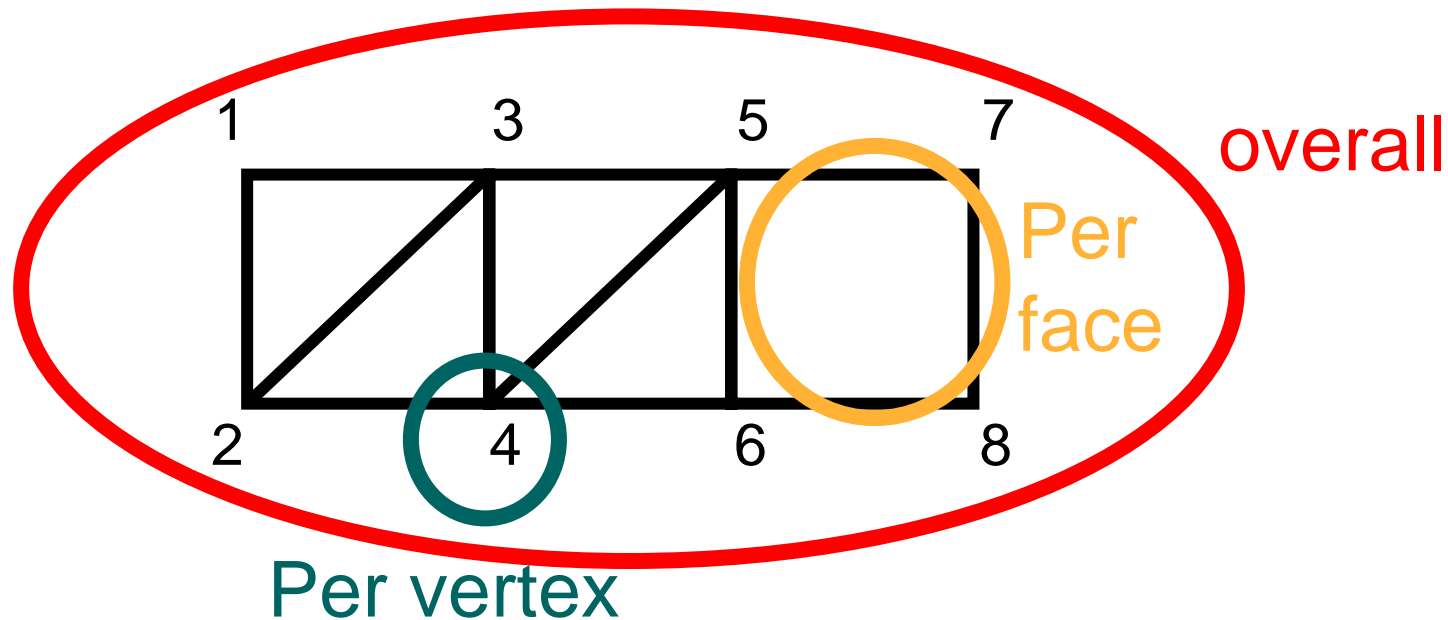
$F = \{3, 3, 4\}$

**Indexed:**

$V = \{P1, P2, P3, P4, P5, P6\}$

$F = \{1, 2, 3, -1, 2, 3, 4, -1, 3, 4, 5, 6, -1\}$

# Polygonal Shapes: Attribute

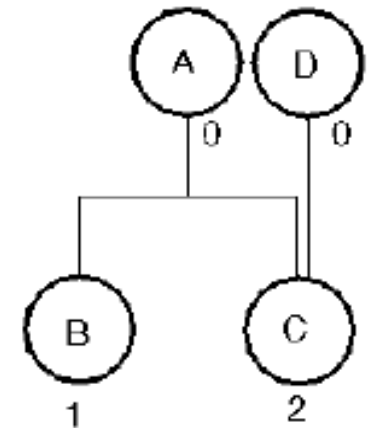


## Bindings of attributes

- for material, normals, texture coordinates
- specifies mapping of attributes to polygons
- Overall object, per face, per vertex

# Reference Counting

- Problem: Deleting a node
  - If multiple references exist
- Solution: Each node counts the references to itself
- Automatic Garbage-Collector
  - Nodes with 0 references are automatically deleted
  - **Never** call delete! (different to removing from scene graph – which is allowed)



---

# Inventor Example: Hello Cone

```
SoSeparator *root = new SoSeparator;  
SoPerspectiveCamera *myCamera =  
    new SoPerspectiveCamera;  
SoMaterial *myMaterial = new SoMaterial;  
root->ref();  
root->addChild(myCamera);  
root->addChild(new SoDirectionalLight);  
myMaterial->diffuseColor.setValue(1.0, 0.0, 0.0); // Red  
root->addChild(myMaterial);  
root->addChild(new SoCone);
```

---

# File format

- Simple ASCII-File format (“\*.iv”)
- **Powerful declarative scripting!**
- DEF Name/USE Name for instancing
- Basic syntax
  - Class\_name {            #Comment
  - field\_name1 <value>
  - field\_name2 <value2>
  - <children if group node>    }

---

# File format - Example

Material { diffuseColor 1 0.5 0 }

Separator {

    Translation { translation 2 4 5 }

    Cone { radius 0.5 height 2 }

}

Separator {

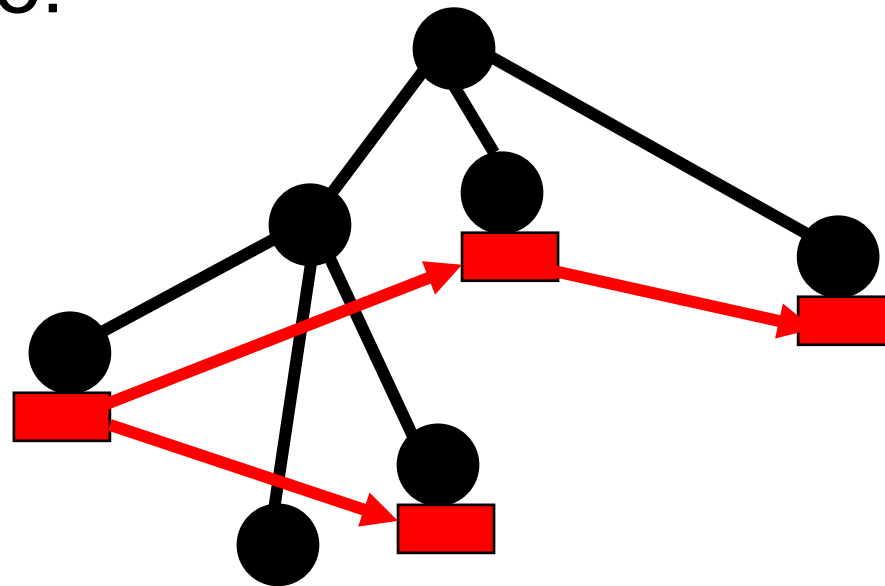
    Translation { translation 0 5 5 }

    Cube {width 1 depth 3 height 2 }

}

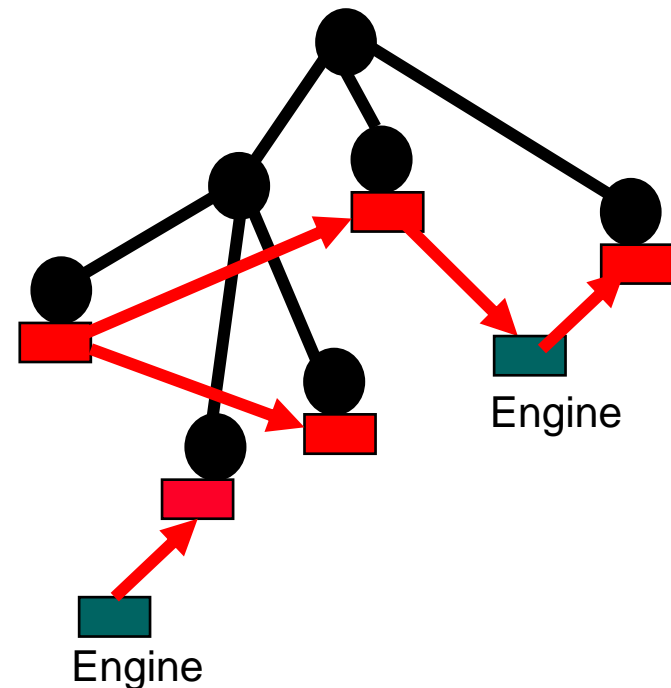
# Dependency Graph

- Two different (overlapping) structures
  - Scene graph
  - Dependency graph (dependent fields)
- “Field connections”: Field types must be compatible!



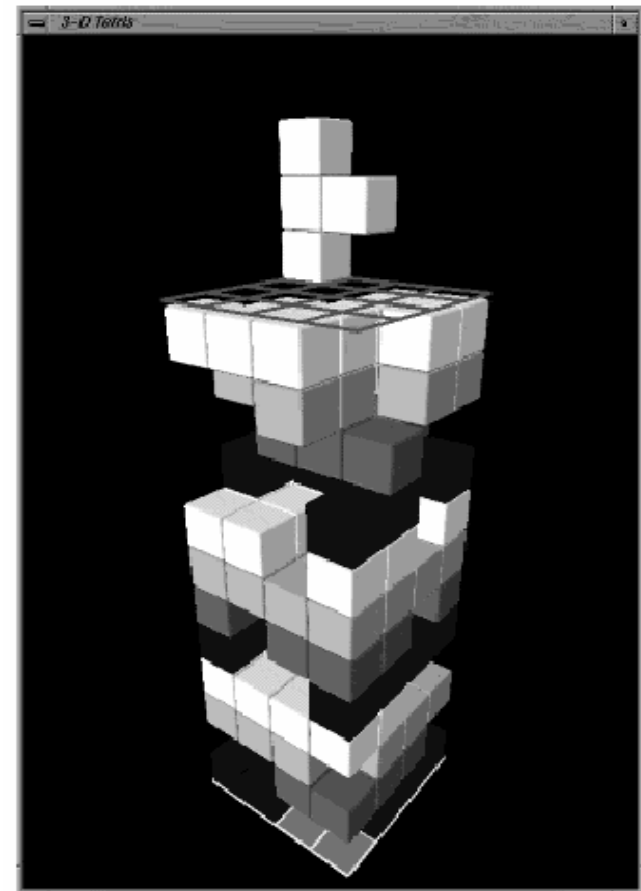
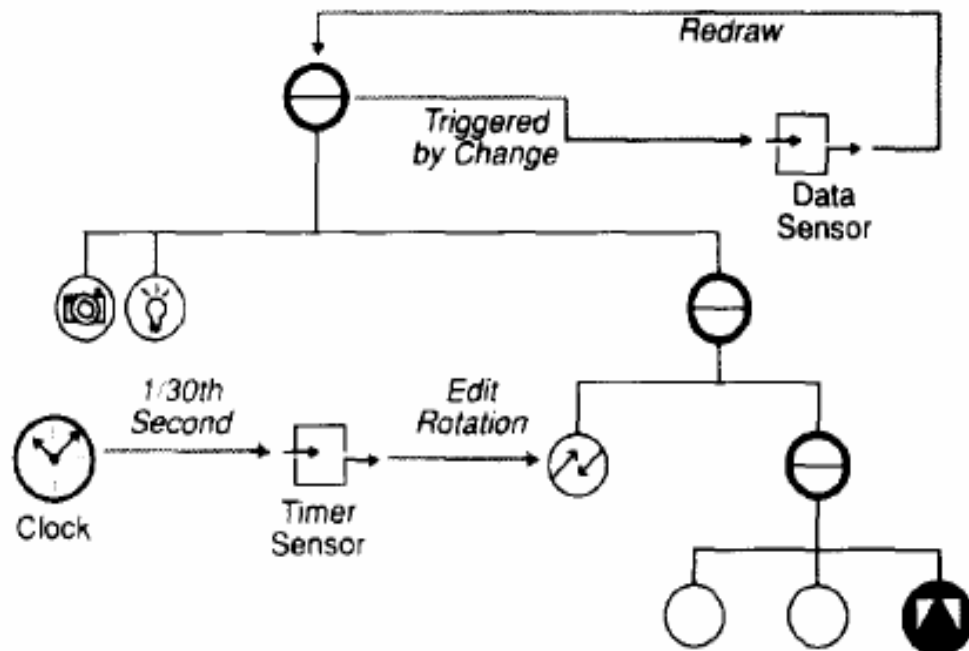
# Engines

- To model complex dependencies in graph
- TargetField := Engine(SourceField)
- Lazy evaluation
- E.g. Calculator, Counter, Type converter, Interpolator, Trigger, *TrakEngine*
- Engine vs. Node
  - No traversal
  - Cumbersome file format
  - Field connections are no objects – cannot be changed on the fly
- Studierstube extensions:  
SoRoute, SoEngineWrapper



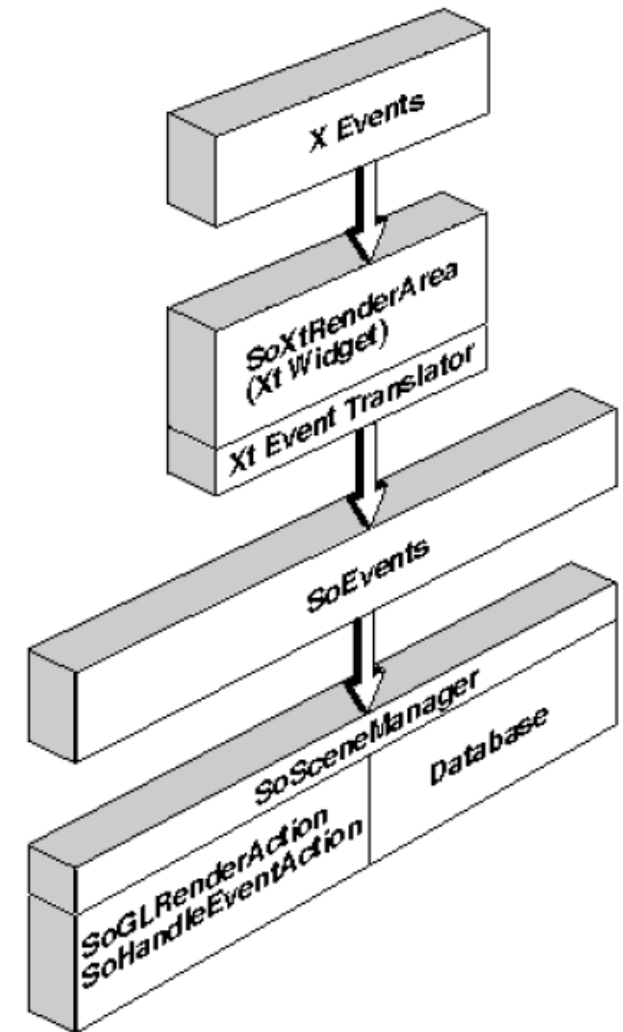
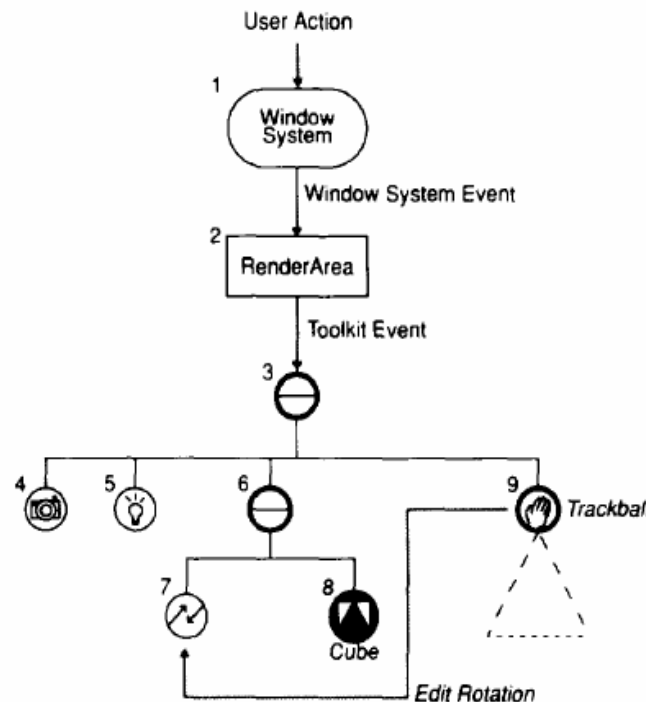
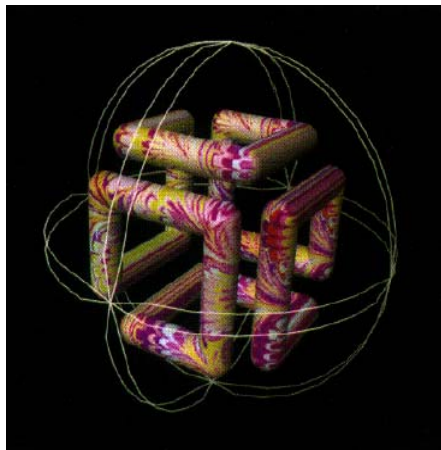
# Sensors

- Callback function called at changes of nodes or fields
- Enables for example
  - Redraw at scene-changes
  - Animations
  - Automatic display of status



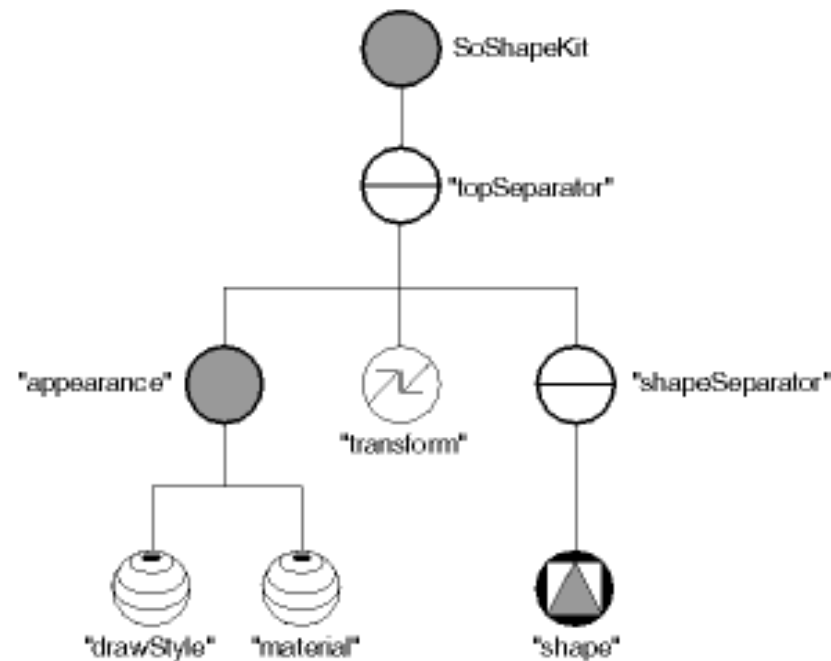
# Events & Manipulators

- Pass user events (mouse, keyboard) on to scene graph
- Group passes event on – each node can handle or grab events
- E.g. Manipulator like virtual trackball



# Node Kits

- Prefabricated sub-scene graphs
  - e.g. transformation, material + shape
  - Simplify the construction of semantically correct scenes



---

# Extensions

- Object-oriented: Subclasses!
- Sub-classes of nodes, actions, node kits, engines, manipulators, draggers, ...
- Dynamic linking of nodes (at run-time)
- Prototyping: Callbacks
  - Callback-Node: call user function on traversal
  - Callback-Action: call user function for every node

---

# Actions & Graph Traversal

- Various Actions:
  - SoGLRenderAction (rendering)
  - SoGetBoundingBoxAction (BBox calculation)
  - SoGetMatrixAction (calculates transf. matrix)
  - SoHandleEventAction (handle events e.g. picking)
  - SoSearchAction (search nodes)
  - SoWriteAction (write to file)
  - SoCallbackAction (execute user callback)
  - SoRayPickAction....

---

# Dual Dispatch

- Action-objects do traversal
- At traversal the corresponding virtual method of each node is called
- Extensibility in C++:
  - Easy: New nodes, the same methods
  - Harder: Nodes with new methods
- New actions solve the problem
  - Action is dependent of node type

---

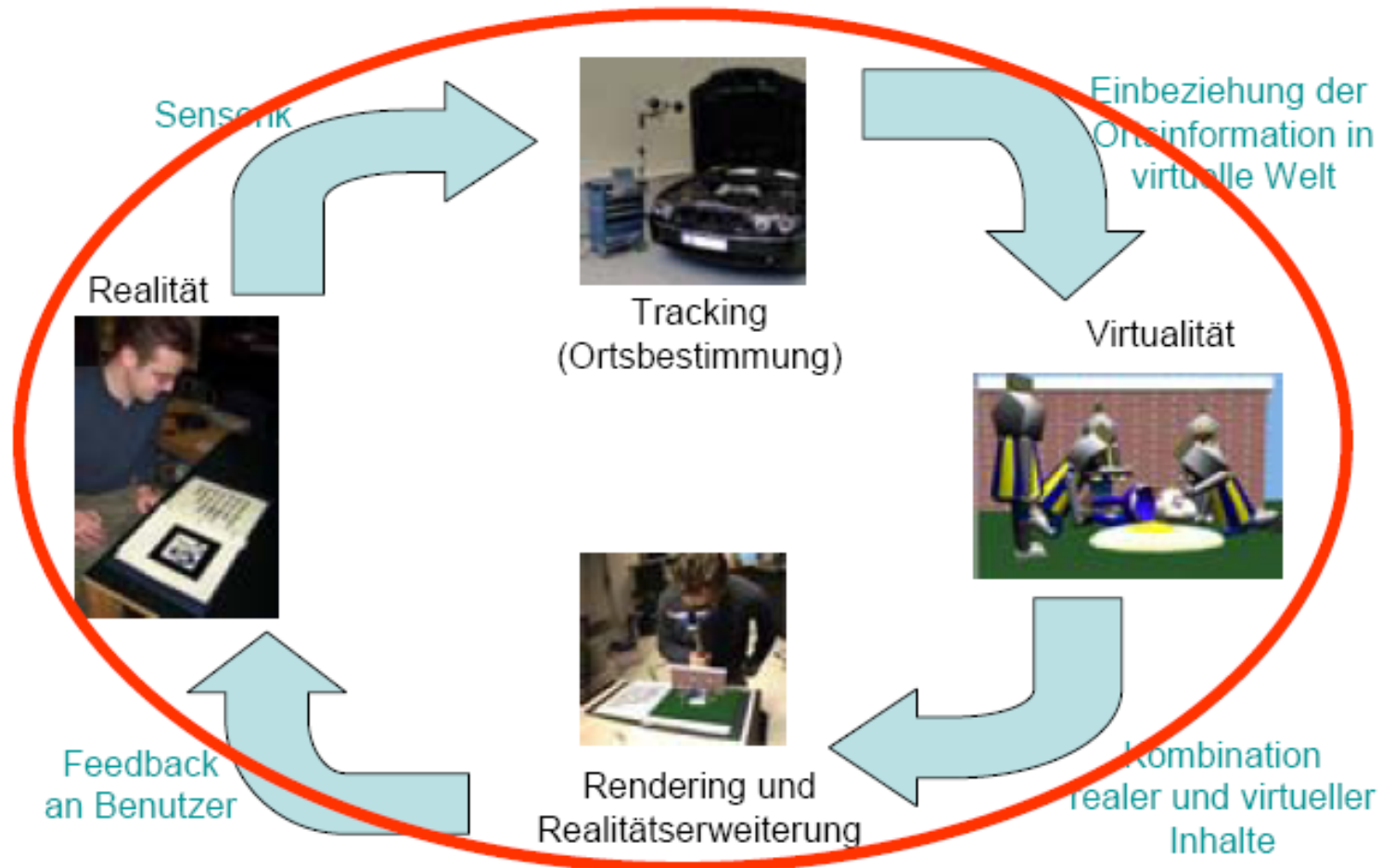
# Literature

- The Inventor Mentor (J. Wernecke)
- The Inventor Toolmaker
  - Addison-Wesley
  - PDF-Files on VRLU-CD (./docs)

---

# Software Design and Components of an AR/VR Toolkit/Framework

# Ein Generisches AR-System



---

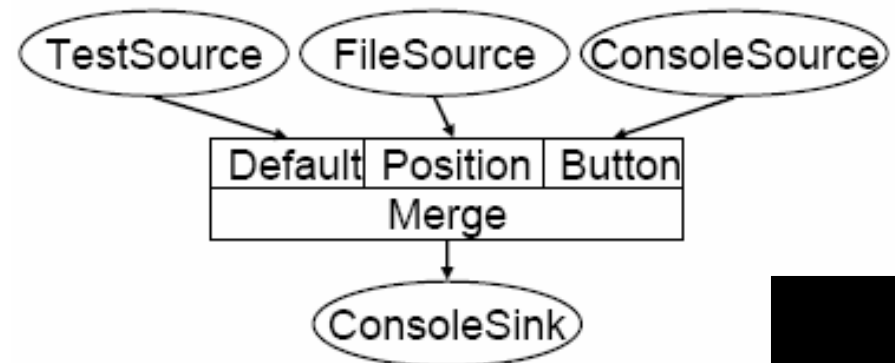
# Requirements & Wishes

- Support multiple input & output devices
  - Input: Interface to tracking middleware (e.g. OpenTracker, VRPN)
  - Output: High level graphics programming, Stereo rendering,...
- Handle user interaction
- Allow flexible 3D user interface
  - widget libraries/middleware
- Support of collaboration
  - multiple users, flexible user configuration, mobile work
- Support distributed work
- Easy application design / authoring

# OpenTracker - Intro

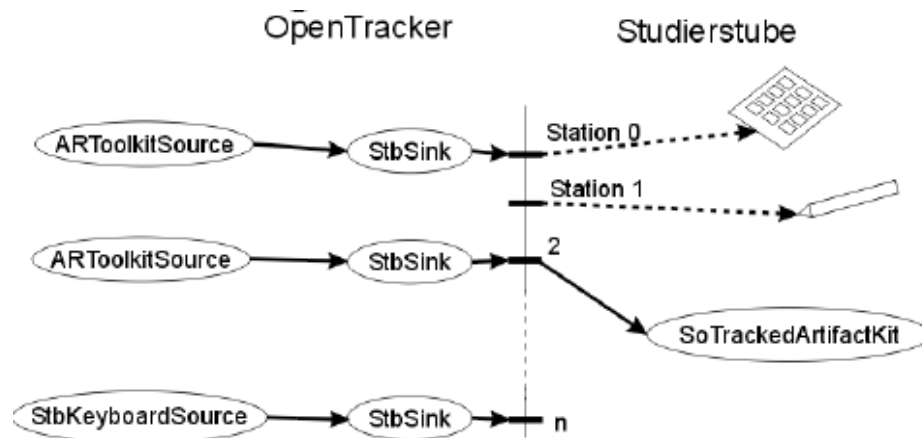
- Software library - Interaction framework
- Implemented as data flow network
- Provides drivers for multiple input devices
- Transformation/Filter of tracking data
- Configuration with XML files
- Flexible distribution of tracking data
- Changes of configuration at run time possible with OT 2.0

[www.studierstube.org/opentracker](http://www.studierstube.org/opentracker)

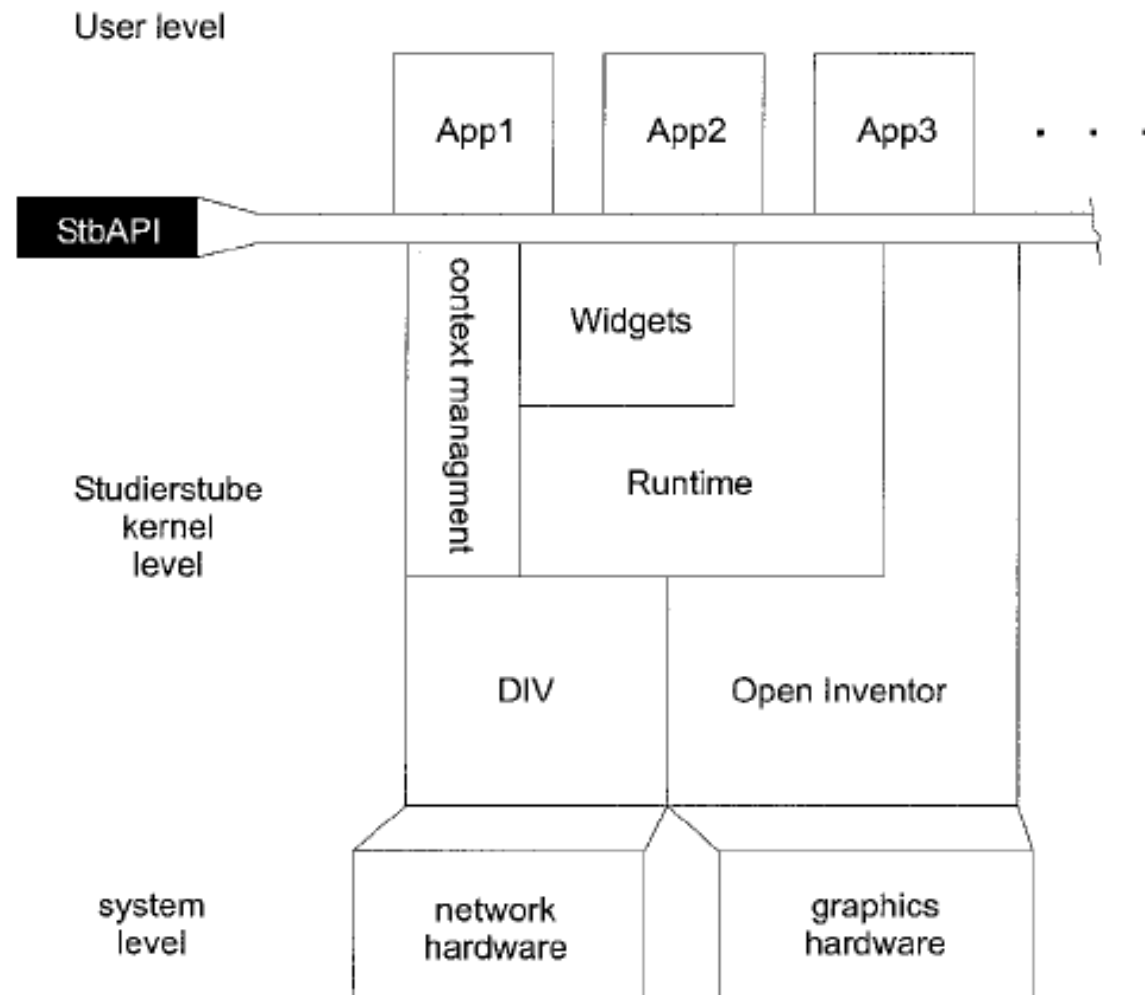


# Studierstube Introduction

- Principle: scene graph library
- Basic idea:
  - Rendering is an integral part of an AR application
- Application logic:
  - is encapsulated **in nodes** of the scene graph.
  - automatically executed at traversal of the scene graph
- Integration of OpenTracker framework



# Studierstube Architecture



---

# Studierstube

- Studierstube Kernel extends Open Inventor scene graph
  - 3D event system (6DOF): Base3D node
  - 3D Widgets
  - Applications in 3D Windows (dynamic loading) -> Multitasking supported
  - Multi-user support
  - Distributed Open Inventor (DIV) – replicate scene graph on multiple machines
    - Multicast vs. TCP/IP network protocol
    - Robust Application replication

---

# Studierstube (cont.)

- Context sensitive scene graph traversal
- Managing collaboration (locales)
- Scripting extensions – new nodes
  - PIVY: Python scripting (<http://pivy.tammura.at>)
- Studierstube 4 (modular design) – still under development