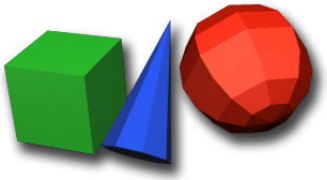
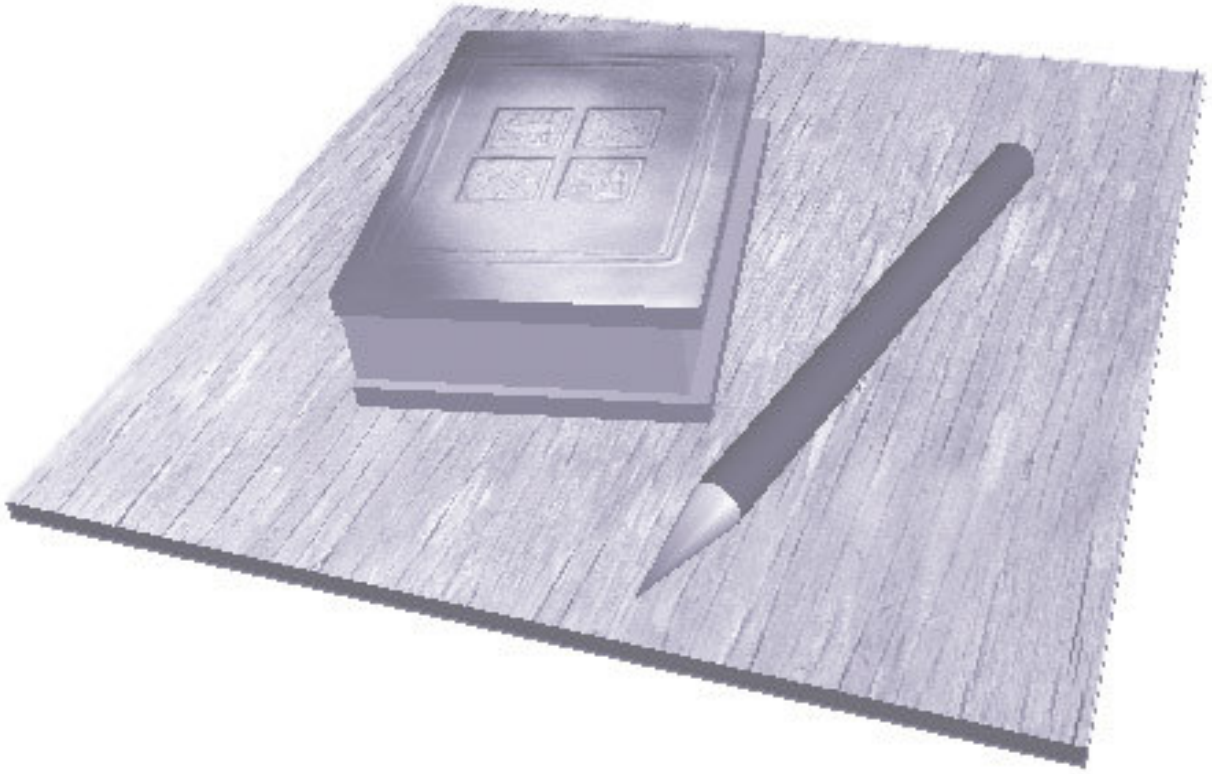


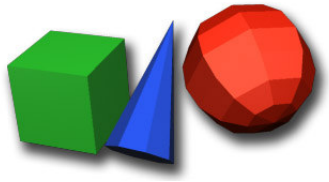
# Java 3D



Seminar Medientechnik LMU München

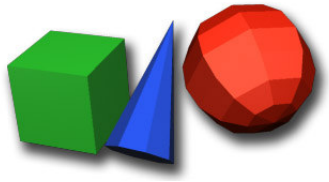
SoSe 2003

# Überblick



- Einführung
- Der Scenegraph
- Transformationen
- 3D Modelle
- Geometrien und Materialien
- Animation
- Interaktion
- Lichter und Texturen
- Programmier-Aufgabe

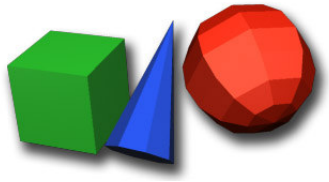
# Einführung



- 3D Grafik API
  - Schnittstelle zur 3D Programmierung
  - OpenGL, Direct3D, Java3D
  - Hardwarebeschleunigung

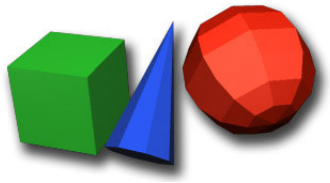
- Java 3D API
  - Spezielle Klassenhierarchie für Java
  - High Level Runtime API
  - Nutzt OpenGL oder Direct3D

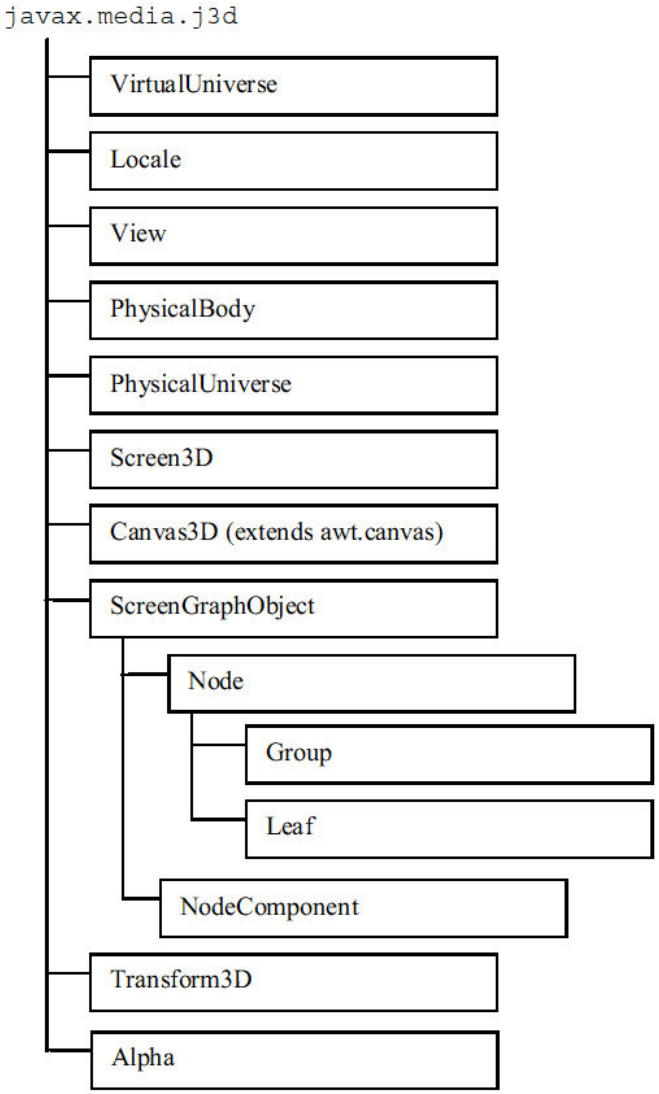
# Vorteile



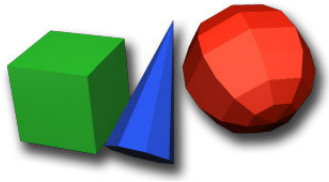
- Relativ einfach zu erlernen
- Gute Integration in Java (Swing, AWT)
- Sehr gut für Web geeignet (Applets)
- Plattformunabhängig
- Automatische Optimierung des Renderingprozesses(!)

# Java3D Klassen



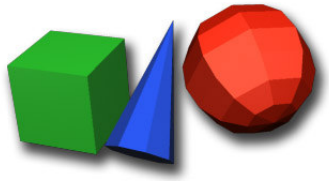


# Scenegraph I



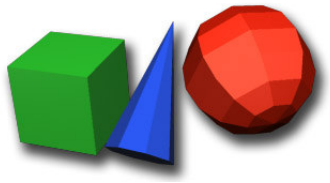
- Enthält alle Daten einer 3D Szene
  - Objekte
  - Transformationen
  - Lichter
  - Interaktionen etc.
- Ist hierarchisch aufgebaut
- Java3D rendert den Scenegraph
- Ermöglicht Optimierung

# Scenegraph II

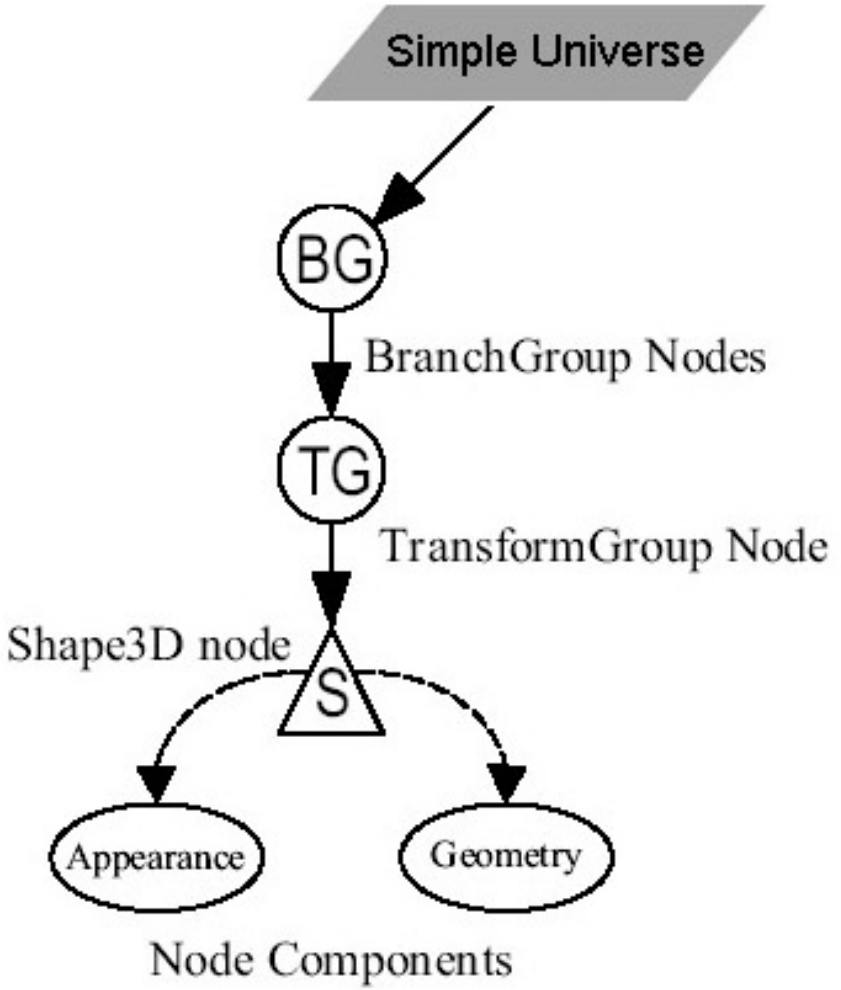


- Besteht aus Knoten, Komponenten und Kanten
  - Knoten
    - Group Nodes: Transformationen u.a.
    - Leaf Nodes: Geometrie, Lichter u.a.
  - Komponenten: definieren Eigenschaften von Knoten
  - Kanten: Verbindung zwischen Elementen
    - Parent-Child Beziehung
    - Referenz

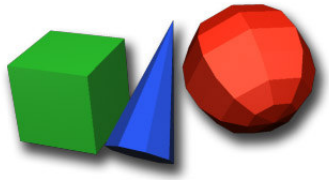
# Scenegraph III



- Virtual Universe
- Content Branch
  - Geometrie, Beleuchtung
- View Branch
  - Projektion, Kamera
- Simple Universe
- Nur ein Vater pro Kind



# Beispiel

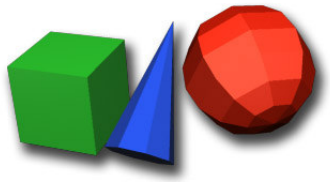


```
public class Beispiel3D extends javax.swing.JFrame {
   public Beispiel3D() {
      getContentPane.setLayout(new BorderLayout()
      GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
      Canvas3D canvas3D = new Canvas3D(config)
      getContentPane().add("Center", canvas3D);
      SimpleUniverse.getViewingPlatform().setNominalViewingTransform();
      BranchGroup scene = createSceneGraph();
      SimpleUniverse.addBranchGraph(scene);
      scene.compile(); }

   private Branchgroup createSceneGraph() {...}

   public static void main(String[] args) {
      new Buch3D().show();
}}
```

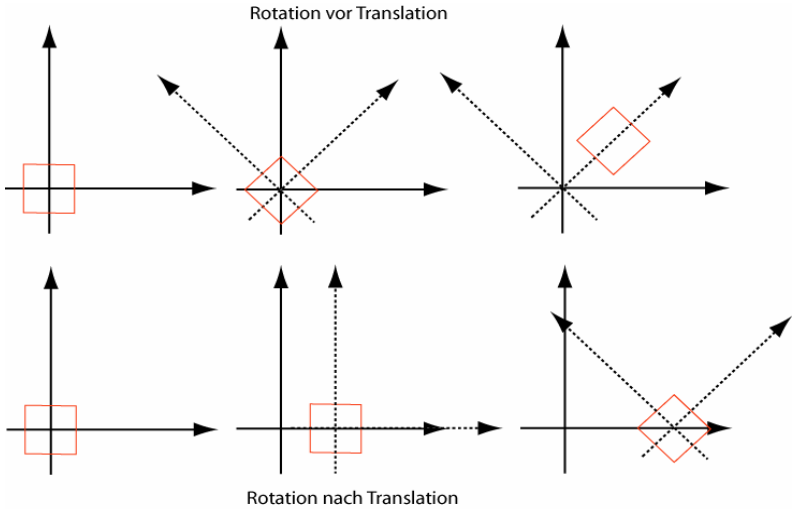
# Transformationen



- Translation, Rotation, Skalierung
- Bsp.: Skalierungsmatrix

$$MS*p = \begin{vmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} = \begin{vmatrix} (sx*x) + (0*y) + (0*z) + (0*1) \\ (0*x) + (sy*y) + (0*z) + (0*1) \\ (0*x) + (0*y) + (sz*z) + (0*1) \\ (0*x) + (0*y) + (0*z) + (1*1) \end{vmatrix} = \begin{vmatrix} x*sx \\ y*sy \\ z*sz \\ 1 \end{vmatrix}$$

- Reihenfolge entscheidend

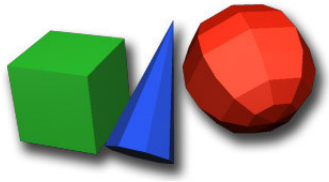


# Transformationen in Java

- Nicht die Objekte werden transformiert, sondern ihre Koordinatensysteme
- TransformGroup ermöglicht ein neues lokales Koordinatensystem für ein Objekt
- Transform3D beschreibt eine Transformation einer TransformGroup
- Beispiel:

```
Transform3D TF = new Transform3D();
TF.set(new Vector3f(-1.0f,0.0f,0.2f));
TransformGroup TG = new TransformGroup(TF);
TG.addChild(new ColorCube());
```

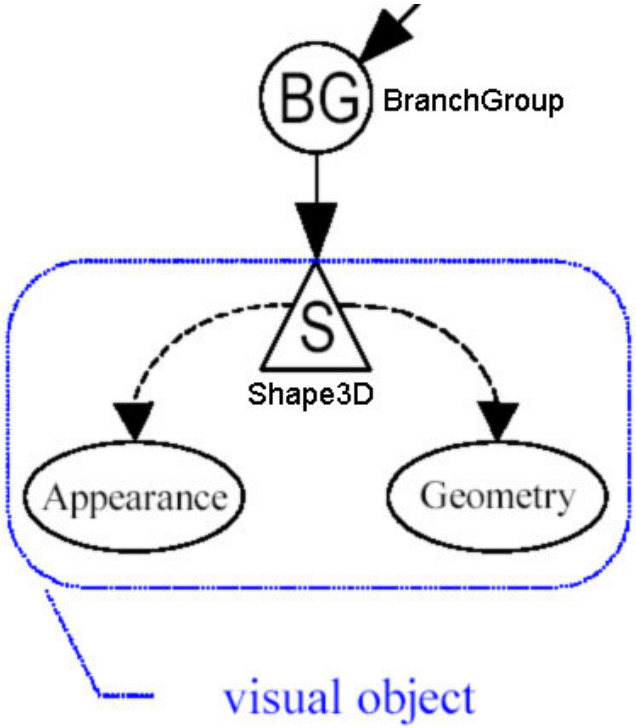
# 3D Modelle



- Es gibt drei Arten um 3D Modelle  zu erzeugen
  - Nutzen von Geometrischen Basisformen
  - Selbst mit Polygon bzw. Punkten aufbauen
  - Spezielle Loader verwenden um Objekte zu laden zum Beispiel .obj oder .3ds

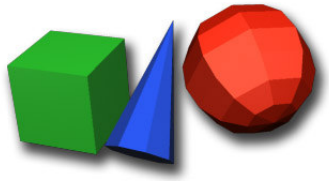
# Grundlage für 3D Modelle

- Shape3D Klasse
- Geometry
  - Koordinaten
  - Normalen
  - Textur Koordinaten
- Appearance (Erscheinung)
  - Farbe
  - Beleuchtungsmodell
  - Transparenz
  - Vieles mehr




```
Shape3D shp = new Shape3D();
shp.setGeometry( Geometry geo );
sph.setAppearance( Appearance appear);
```

# Basiskörper



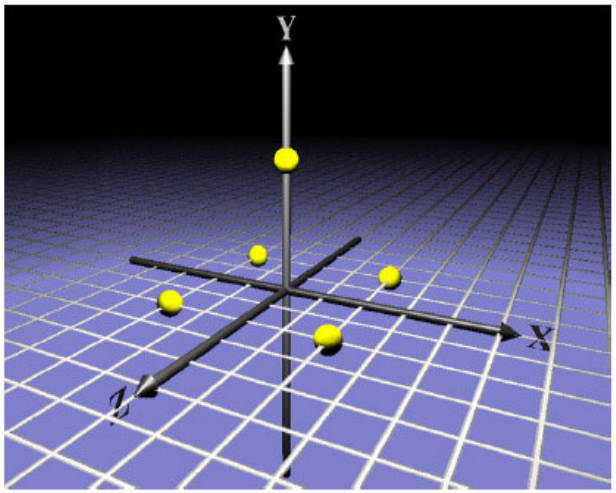
- Zu finden in *com.sun.j3d.utils.geometry.Primitive*
- Box
- Cone
- Cylinder
- Sphere

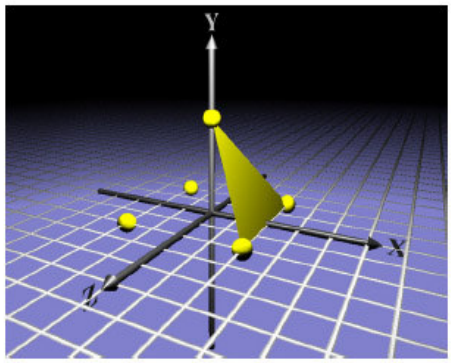
Beispiel:

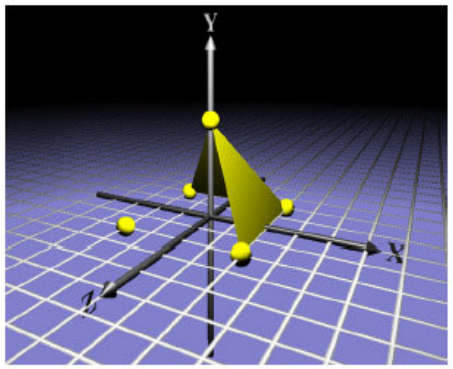
```
Appearance app = new Appearance();
...
Box Tisch = new Box(1.0f,0.2f.2.0f,app)
```

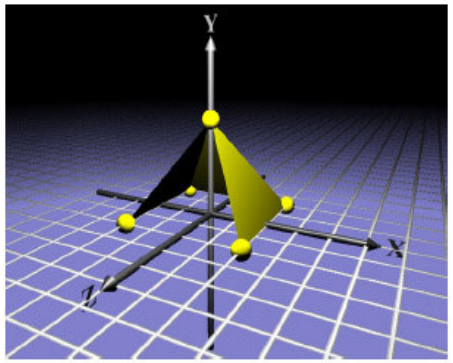
# Selbstdefinierte Polygone

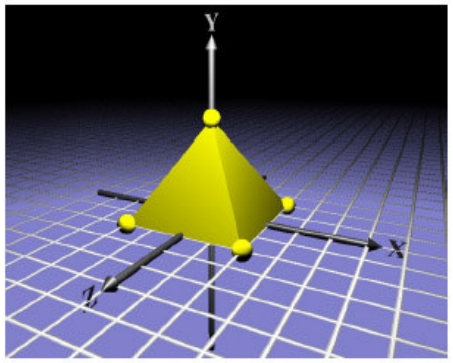
- Alle 3D Daten sind Vertex(Punkt) orientiert
- Polygone werden durch diese definiert (Koordinaten)



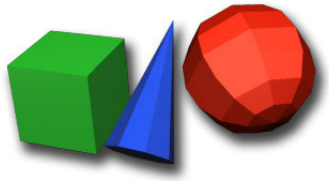



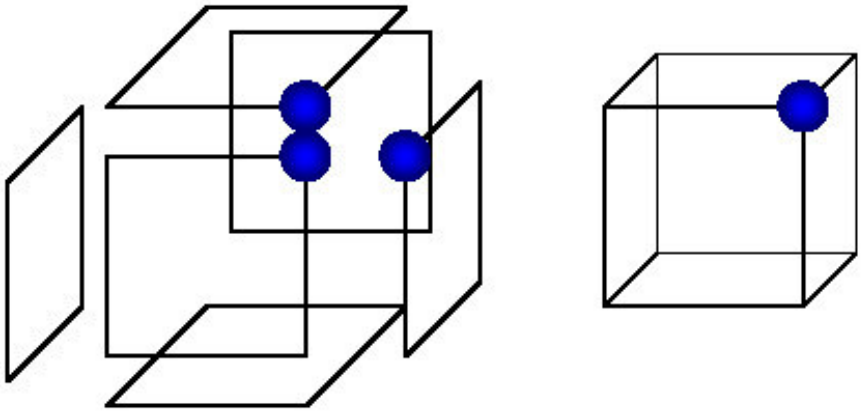



# Geometrie

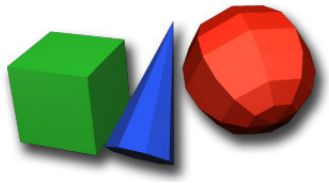


- 3 Hauptarten
  - Non-Index vertex-based (Vertex nur einmal benutzt)
  - Index vertex-based  (Vertex wird mehrmals benutz)
  - Andere Arten (Text3D, Compressed Geometry etc.

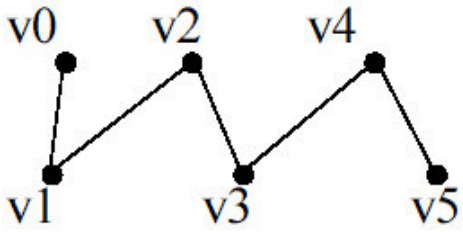
Non-Index vs. Index



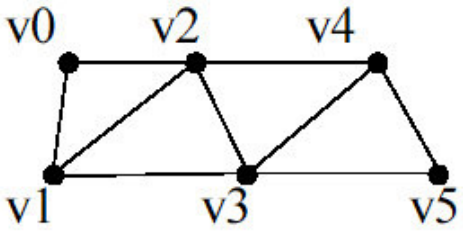
# Geometrie II



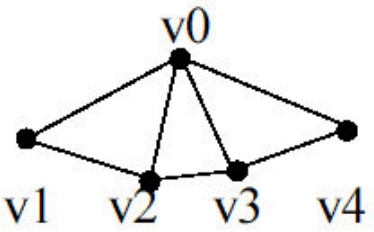
- GeometryArray ist „Container“ für die 3D Daten
- Subklassen für die verschiedenen Arten
  - Linien, Punkte, Polygone
- Klasse GeometryArray ist „non-indexed“
- Deswegen Subklasse GeometryStripArray




LineStripArray

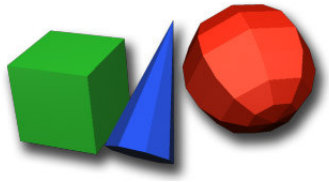



TriangleStripArray



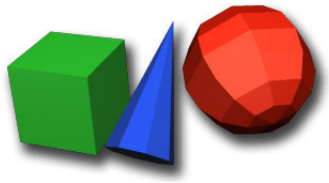

TriangleFanArray

# Appearance

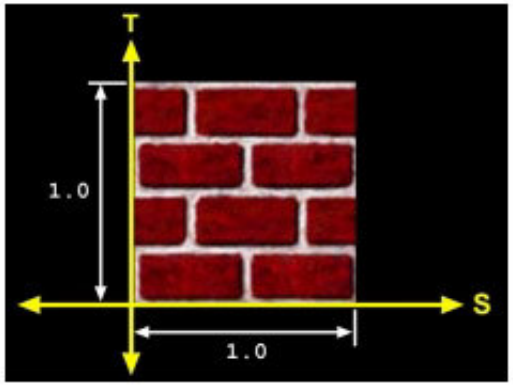


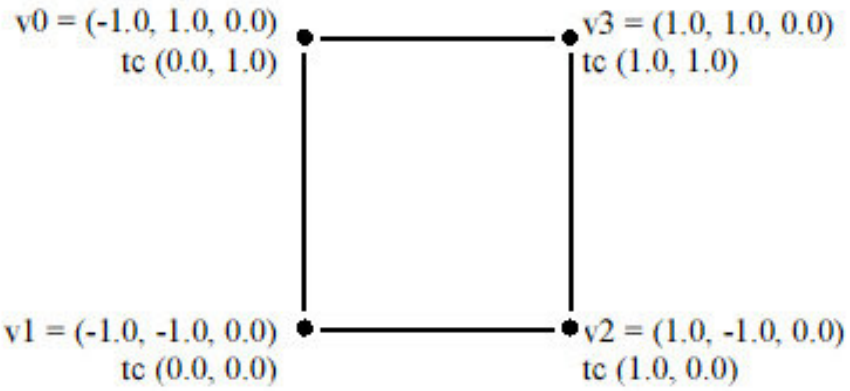
- Appearance dient als Container für alle visuellen Eigenschaften eines 3D-Modells
- Farbe und Transparenz
  - Material (beleuchtetes Modell)
  - ColoringAttributes (unbeleuchtet)
  - TransparencyAttributes
- Rendering
  - PolygonAttributes (Punkt, Linien, Gefüllt, Backface Culling)
  - RenderingAttributes (DepthBuffer, Alpha etc.)
- Textur
  - Textur Bild
  - Texturkoordinaten-Erzeugung

# Texturkoordinaten

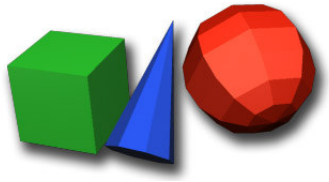


- Vorraussetzung für Textur (sonst Exception!):
  - Breite = $2^n$
  - Höhe = $2^m$
- Polygone brauchen Texturkoordinaten
  - Logische Koordinaten einer Textur
  - Festlegen der entsprechenden Vertices



# Texturkoordinaten

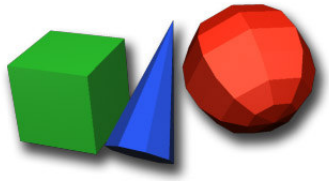


- Automatisch Erzeugung bei den Primitven

  ...new Box(1.0f,1.0f,1.0f,Box.GENERATE_TEXTURE_COORDS | Box.GENERATE_NORMALS, appearance);
- Selbst eingeben (mühsam!)
- Textur auf Modell projizieren durch TexCoordGeneration()
  - Linear
  - Sphärisch
- Ein fertiges Objekt mit Texturkoordinaten laden

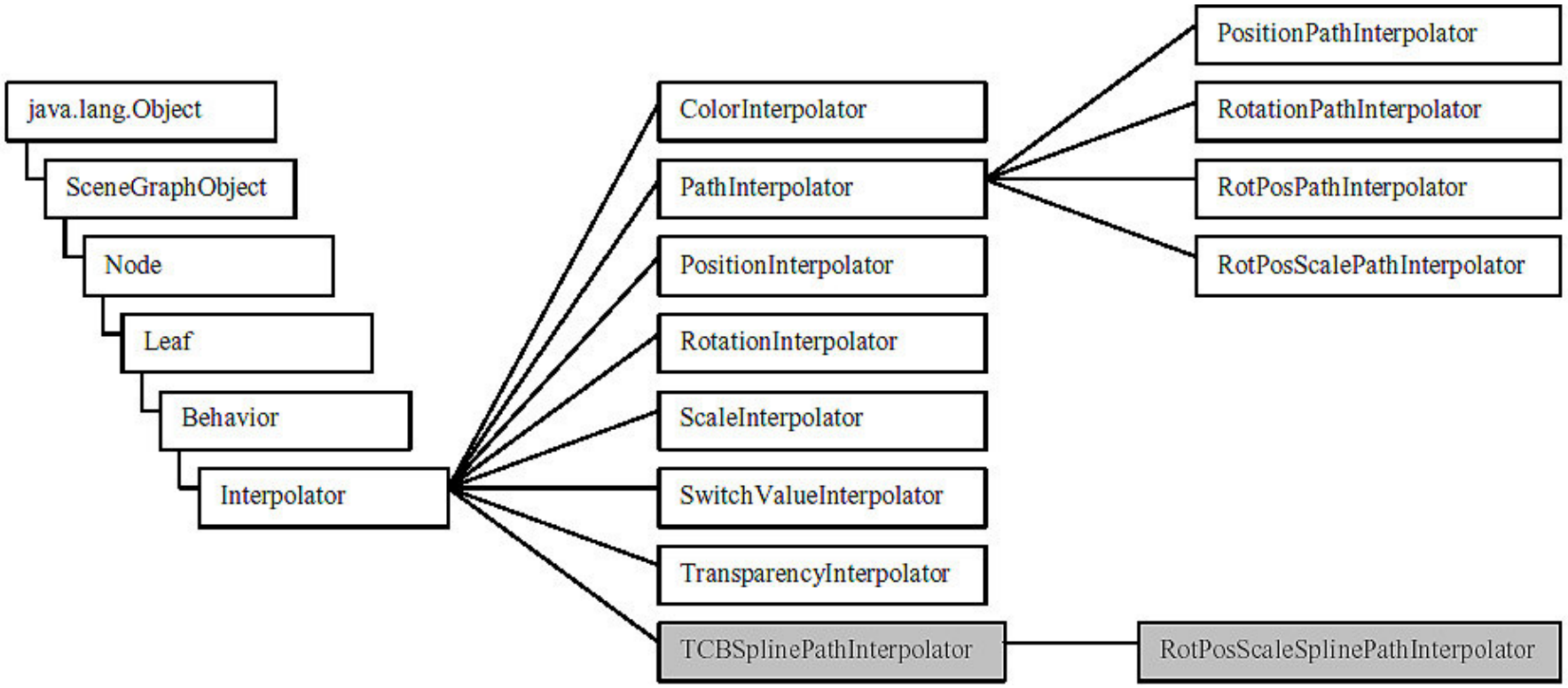
  z.b. mit Right Hemisphere DeepUV erstellt

# Animation

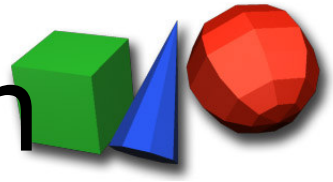


- Animation ist eine automatische Änderung der Szene über einen bestimmten Zeitraum
- In Java3D werden Animationen und auch Interaktionen durch die Behaviors implementiert
- Vorgefertigte Behaviors sind die Interpolators
  - RotateInterpolator, TranslateInterpolator…
  - ColorInterpolator, TransparenyInterpolator
  - Ein Alpha Objekt gibt das Zeitsignal (0.0f-1.0f)
- Es ist möglich seine eigenen Behaviors zu schreiben

# Behavior Klassen



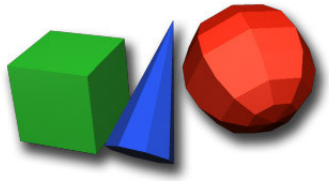
# Rezept für Animationen



- Erzeugen des zu verändernden Objekts
- Capabilites setzen
- Erzeugen des Alpha Objekts
- Erzeugen eines Interpolators mit Referenz auf Alpha und Objekt
- Bestimmen der SchedulingBounds (Aktivierungsbereich)
- Hinzufügen des Interpolators zum Scenegraph

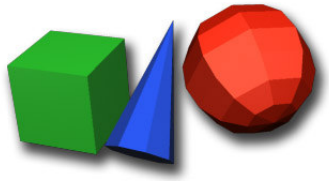
- Animationen sind auch ohne Interpolatoren möglich!

# Eigene Behaviors



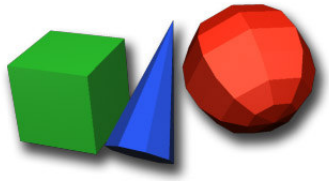
- Behaviors haben zwei wichtige Methoden, die überschrieben werden müssen
- Initialize()
  - WakeupCondition festlegen
    - Z. B. WakeupOnAWTEvent(MouseEvent.MOUSE_CLICKED)
- processStimulus(Enumeration criteria)
  - Code der nach Aktivierung ausgeführt wird
  - Möglichkeit andere Behaviors zu benachrichtigen

# Interaktion



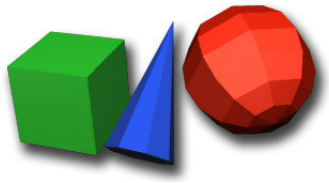
- Interaktion ist die Veränderung der Szene nach einer Benutzereingabe (Maus, Tasten, etc.)
- Möglichkeiten durch Interaktion
  - Navigation in der Szene mit Tastatur und Maus
  - Selektieren oder Manipulieren einzelner Objekte in der Szene (Picking)

# Navigation

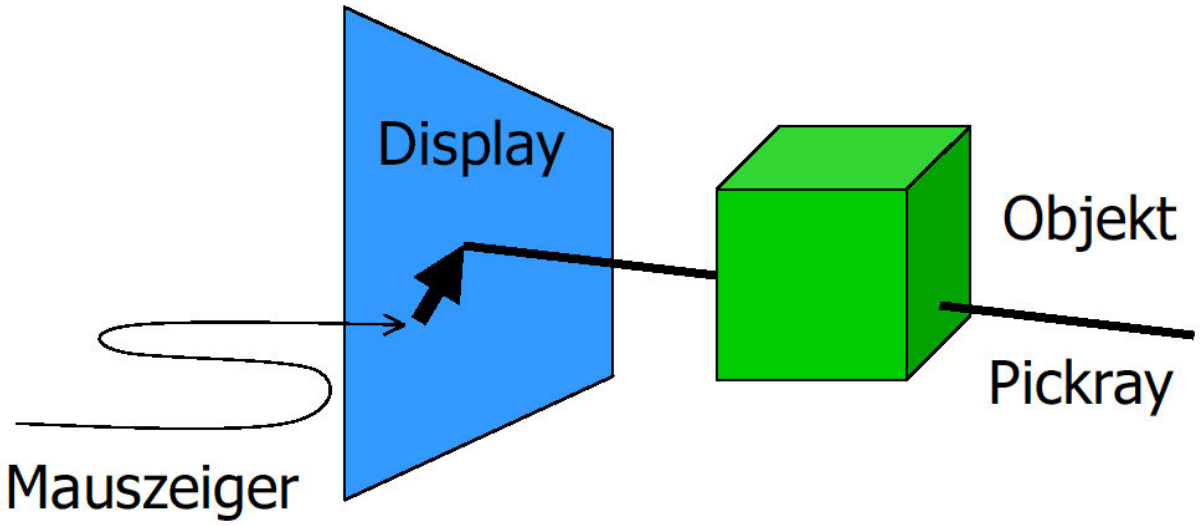


- Vorgefertigte Navigations Behaviors
- MouseBehavior
  - MouseRotate
  - MouseTranslate
  - MouseZoom
- KeyNavigatorBehavior
- Einfach zu Implementieren

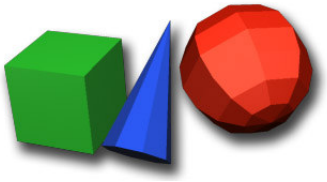
# Picking



- Nach einem Mausklick wird ein Strahl in die Szene geschossen (daneben gibt es noch PickShapes)
- Picking gibt ein SceneGraphPath Objekt zurück das die durchschnittenen Objekte enthält
- Picking ist rechenintensiv, daher ist es möglich die Genauigkeit festzulegen



# Picking-Utility-Klassen

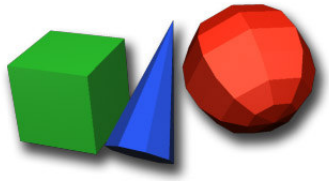


- Es gibt vordefinierte Picking Behaviors
  - PickingRotateBehavior
  - PickingTranslateBehavior
  - PickingZoomBehavior
- Erstellen eines Picking Behavior
  - Picking***Behavior mit Referenz auf Objekt erzeugen
  - Hinzufügen zum Scenegraph
  - Capabilites für das Objekt setzen

# Benutzerdefiniertes Picking

- Implementiert die Selektierbarkeit eines Objekts
- Ablauf
  - Erzeugen eines PickRays vom Auge durch die Mauskoordinaten
  - Selektieren alles durchlaufenen Objekte
  - Rückgabewert: Sortierte Liste (SceneGraphPath[])
  - Zugreifen auf das 1. Objekt
  - Manipulationen

# Beleuchtung

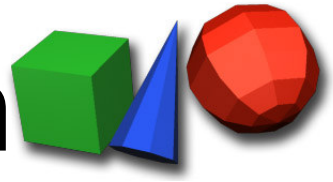


- Java3D bietet mehrere Arten von Lichtern
  - AmbientLight - Grundbeleuchtung
  - DirectionalLight - Paralleles Licht (z.b Sonne)
  - PointLight - Punktlicht
  - SpotLight - Lichtkegel
- Attribute von allen Lichtquellen
  - Status an/aus
  - Farbe
  - Volumen des Einflussbereichs
- Nicht vergessen: Surface-Normalen für Shading

# Programmier-Aufgabe

*„Modellieren Sie mit Hilfe von Java 3D ein Buch und einen Bleistift aus graphischen  Primitiven. Das Buch und der Bleistift sollen im Raum auf einer Ebene liegen. Fügen Sie der Szene zwei unterschiedliche Lichtquellen hinzu. Bei einem Mausklick auf das Buch soll sich dieses in einer Animation öffnen. Bei einem weiteren Klick soll sich das Buch wieder schließen. Auf der aufgeschlagenen Buchseite soll ein Text oder ein Bild zu sehen sein. Verwenden Sie dafür eine geeignete Textur.“*

# Verwendete Techniken

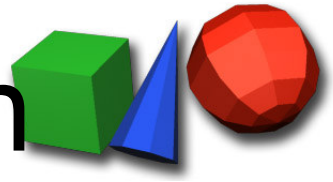


- Maus Navigation
- Transformationen
- Animation
- Picking
- Beleuchtung
- Texturen
- Hintergrund

# Zusammenfassung Java3D

- Vorteile
  - „Einfach“ zu lernen
  - Baut auf Java auf
  - Webfähig
  - High Level API
  - Scenegraph
- Probleme
  - Performance
  - Manche vermeintlich einfache Dinge sind recht kompliziert gelöst

# Weitere Informationen



- Sun Homepage

http://java.sun.com/products/java-media/3D/

http://developer.java.sun.com/developer/onlineTraining/java3d/

- Komplette Java 3D API

http://www.cs.ucsb.edu/~cs290imr/javadoc/overview-summary.html

- Allgemein

http://www.j3d.org

http://www.java3d.org

http://webster.fhs-hagenberg.ac.at/staff/haller/mmp5_20012002/