B6. 3D-Computergrafik mit Java

B6.1 Grundlagen der 3D-Computergrafik

B6.2 Einführung in Java 3D

B6.3 Animation

B6.4 Geometrie, Material, Beleuchtung (Forts.) 

B6.5 Interaktion

Literatur:

D. Selman: Java 3D Programming, Manning 2002

A. E. Walsh, D. Gehringer: Java 3D API Jump-Start, Prentice-Hall 2001

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

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

http://java.sun.com/docs/books/java3d/

http://www.j3d.org, http://www.java3d.org/

Wiederholung: Beispiel zu Material und Lighting

```
public BranchGroup createSceneGraph() {
  BranchGroup objRoot = new BranchGroup();
  ...
  TransformGroup objSpin = new TransformGroup();
  ...
  Appearance app = new Appearance();
  Material mat = new Material();
  mat.setDiffuseColor(new Color3f(1.0f, 0.0f, 0.0f));
  app.setMaterial(mat);
  objSpin.addChild(new Box(0.4f, 0.4f, 0.4f, app));
  ...
  DirectionalLight light1 = new DirectionalLight();
  light1.setDirection(new Vector3f(1.0f, -1.0f, -0.5f));
  light1.setInfluencingBounds(new BoundingSphere());
  objRoot.addChild(light1);

  SpotLight light2 = new SpotLight();
  light2.setDirection(new Vector3f(-1.0f, +1.0f, 0.f));
  light2.setPosition(2.0f, -1.0f, 0.0f);
  light2.setInfluencingBounds(new BoundingSphere());
  objRoot.addChild(light2);

  return objRoot;
}
```

Achtung: Lokale Koordinatensysteme

- Testfrage: Was passiert, wenn man im Beleuchtungsbeispiel statt
  ```
  objRoot.addChild(light1);
  ```
  die folgende Zeile schreibt:
  ```
  objSpin.addChild(light1);
  ```
  ?
- Jeder Knoten des Szenengraphen trägt sein eigenes lokales Koordinatensystem
- Transform-Knoten bilden das lokale Koordinatensystem eines Unterknotens auf das übergeordnete Koordinatensystem ab
  - Die Abbildung kann auch dynamisch, d.h. zeitabhängig sein
- Das Einfügen eines Knotens als Unterknoten eines anderen bedeutet auch die Auswahl eines spezifischen Koordinatensystems!

Vorgefertigte Geometrien

- Die Erstellung von Geometrien ist extrem arbeitsaufwändig
- Prinzipiell sind grafische Werkzeuge besser geeignet für die Erstellung dreidimensionaler Objekte und Szenen als Programmcode
- Ausweg:
  - Externes Erstellen von Szenegraphen oder Teilen davon
  - „Interpreter“ in Java, der externe Datei einliest
  - „*Loader*“
- Standardbasis für Loader-Klassen:

  Paket **`com.sun.j3d.loaders`**

  Enthält u.a. (im Interface *Loader*):

  ```
  Scene load (java.lang.String fileName)
  ```
- Loader-Implementierungen nicht Bestandteil der Distribution
  - Von externer Quelle zu beziehen
  - Viele Loader kostenfrei verfügbar

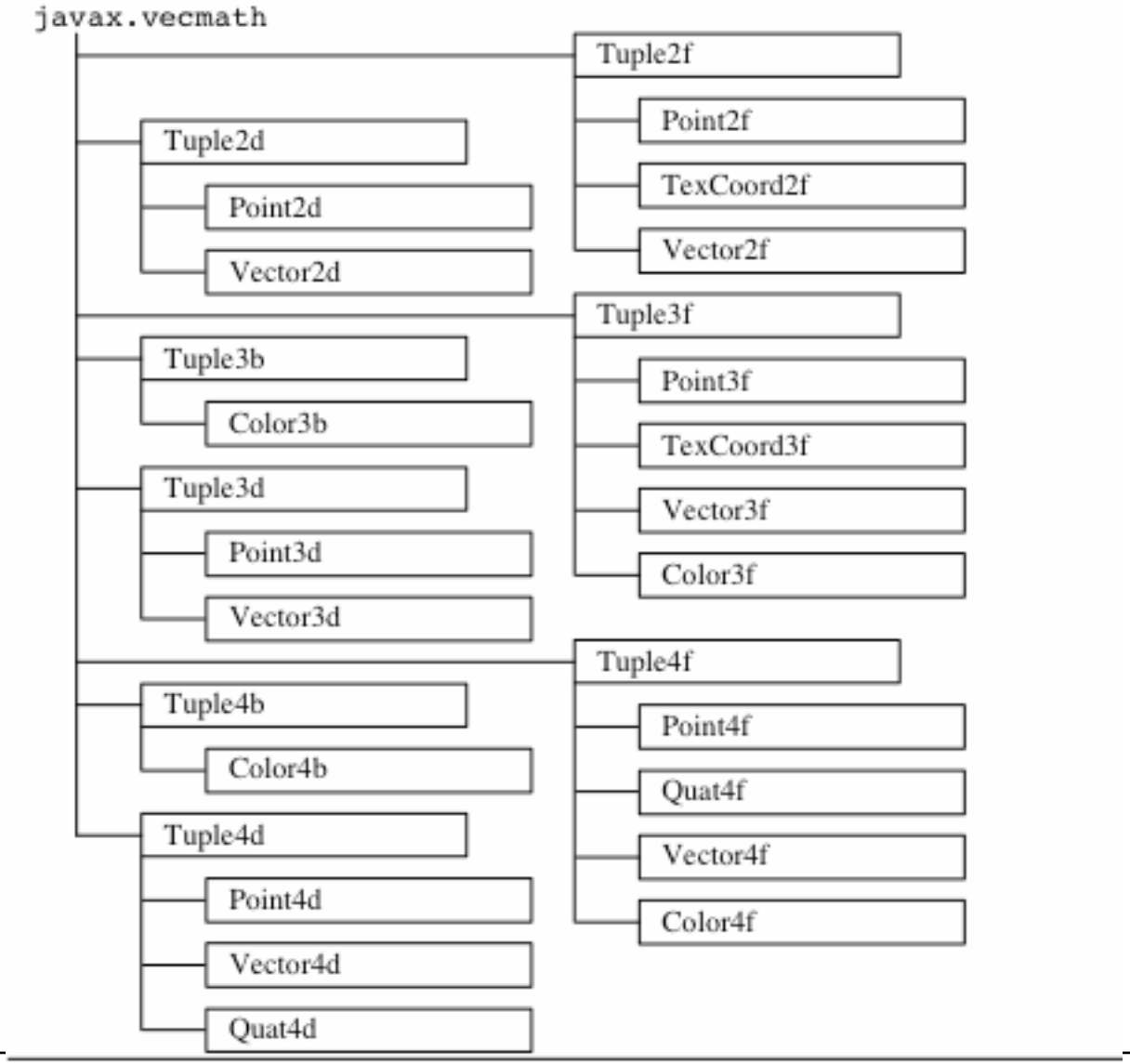
Populäre 3D-Grafikformate mit J3D-Loader

- 3DS 3D Studio Max
- COB Caligari TrueSpace
- DEM Digital Elevation Map
- DXF AutoCAD Austauschformat
- IOB Imagine
- LWS Lightwave Scene Format
- NFF WorldToolKit
- OBJ Alias/Wavefront
- PDB Protein Data Bank
- SLD Solid Works
- VRT Superscape
- VTK Visual Toolkit
- WRL VRML
- X3D X3D

Text in Java 3D

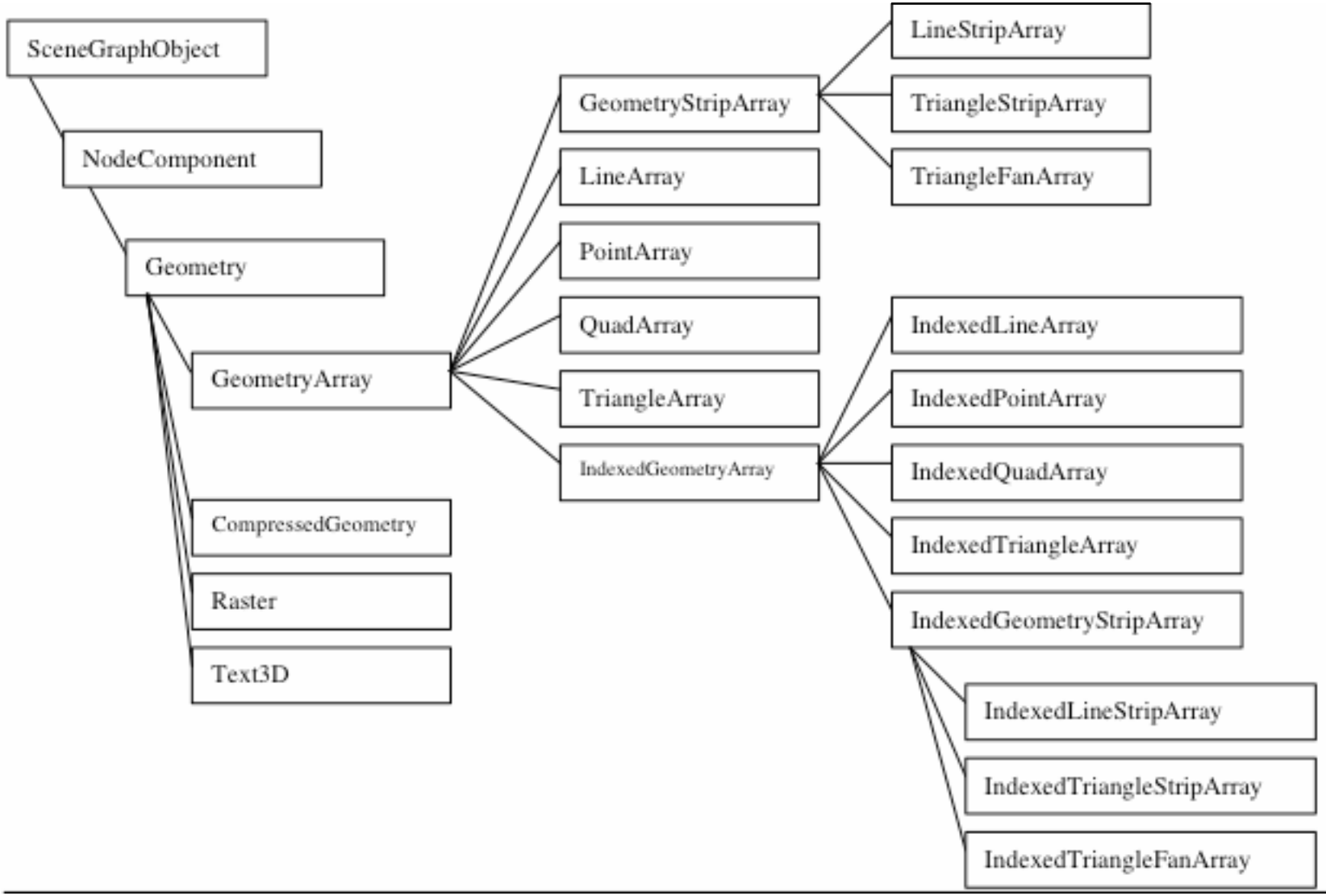
- Text2D
  - Transparente Rechtecke
  - mit durch Textur ausgezeichnetem Text
- Text3D
  - Dreidimensionale Textwiedergabe
  - Verwendet 3D-Versionen gängiger Fonts (z.B. Helvetica)

Vektormathematik-Klassen in Java



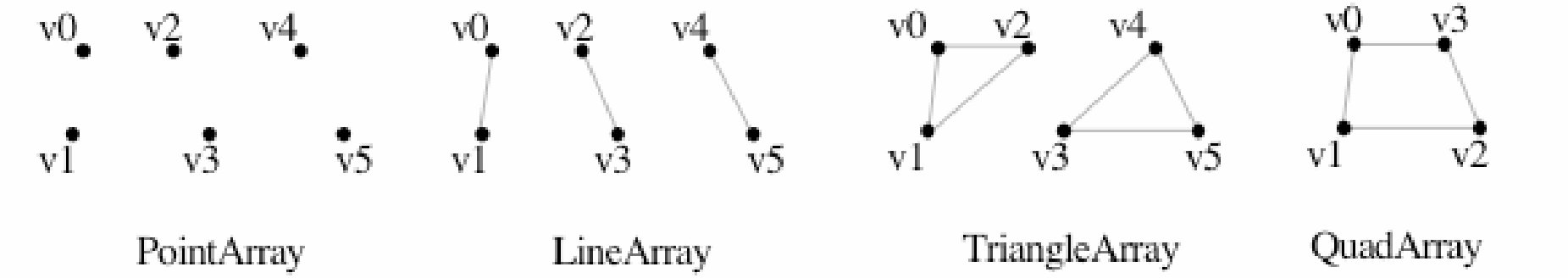
f: float
d: double
b: byte

Geometrie-Klassen in Java 3D



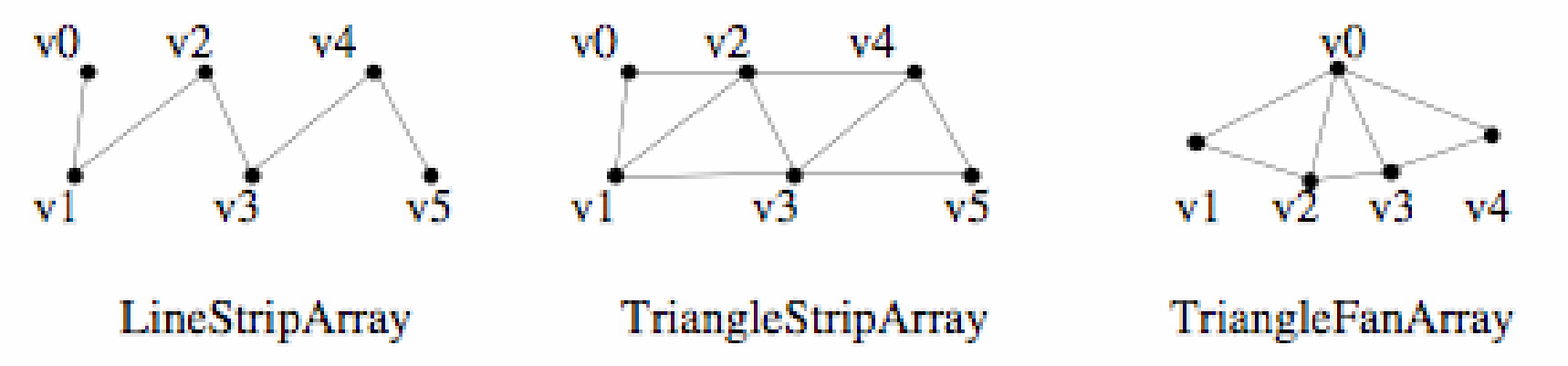
GeometryArray-Klassen

- GeometryArray:
  - enthält Ansammlung von Eckpunkten *(vertices)* als Koordinaten-Daten
  - verschiedene Unterklassen legen verschiedene Arten fest, wie die Punkte zu geometrischen Gebilden verbunden werden
- Zusatzinformationen möglich:
  - Farbe
  - Flächennormale (Vektor orthogonal zur Fläche, zeigt Orientierung)
  - Texturkoordinaten
- Unterklassen:

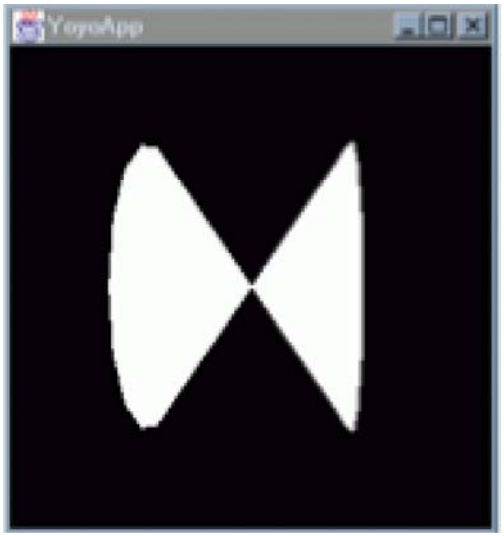


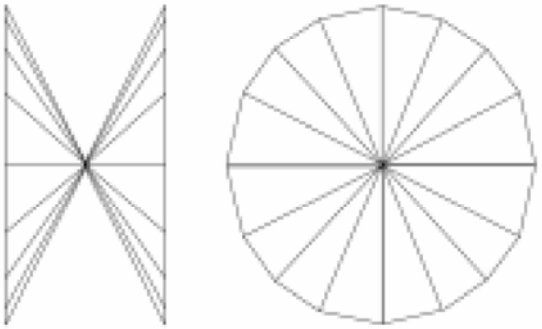
GeometryStripArray-Klassen

- Einbindung desselben Knotens in mehrere geometrische Objekte
  - durchgehende Linien
  - Dreieckszerlegung von Flächen
- Drei Typen (= Unterklassen)
  - Linien: **`LineStripArray`**
  - Dreieckszerlegung: **`TriangleStripArray`**
  - Fächerformige Dreieckszerlegung: **`TriangleFanArray`**



Beispiel 1: Jo-Jo-Geometrie





```
int N = 17;
int totalN = 4*(N+1);
Point3f coords[] = new Point3f[totalN];
int stripCounts[] = {N+1, N+1, N+1, N+1};
float r = 0.6f;
float w = 0.4f;
...
coords[0*(N+1)] = new Point3f(0.0f, 0.0f, w);
coords[1*(N+1)] = new Point3f(0.0f, 0.0f, 0.0f);
coords[2*(N+1)] = new Point3f(0.0f, 0.0f, 0.0f);
coords[3*(N+1)] = new Point3f(0.0f, 0.0f, -w);

for(a = 0,n = 0; n < N;
    a = 2.0*Math.PI/(N-1) * ++n){
  x = (float) (r * Math.cos(a));
  y = (float) (r * Math.sin(a));
  coords[0*(N+1)+n+1] = new Point3f(x, y, w);
  coords[1*(N+1)+N-n] = new Point3f(x, y, w);
  coords[2*(N+1)+n+1] = new Point3f(x, y, -w);
  coords[3*(N+1)+N-n] = new Point3f(x, y, -w);
}

tfa = new TriangleFanArray (totalN,
  TriangleFanArray.COORDINATES, stripCounts);
tfa.setCoordinates(0, coords);
...
```

Beispiel 2: ColorCube (1)

- Auszug aus dem Quelltext von **`com.sun.j3d.utils.geometry.ColorCube`**:

```
private static final float[] verts = {
  // front face
 1.0f, -1.0f,  1.0f,
 1.0f,  1.0f,  1.0f,
-1.0f,  1.0f,  1.0f,
-1.0f, -1.0f,  1.0f,
  // back face
-1.0f, -1.0f, -1.0f,
-1.0f,  1.0f, -1.0f,
 1.0f,  1.0f, -1.0f,
 1.0f, -1.0f, -1.0f,
  // right face
 1.0f, -1.0f, -1.0f,
 1.0f,  1.0f, -1.0f,
 1.0f,  1.0f,  1.0f,
 1.0f, -1.0f,  1.0f,
  // left face
-1.0f, -1.0f,  1.0f,
-1.0f,  1.0f,  1.0f,
-1.0f,  1.0f, -1.0f,
-1.0f, -1.0f, -1.0f,...
```

```
// top face
   1.0f,  1.0f,  1.0f,
   1.0f,  1.0f, -1.0f,
  -1.0f,  1.0f, -1.0f,
  -1.0f,  1.0f,  1.0f,
// bottom face
  -1.0f, -1.0f,  1.0f,
  -1.0f, -1.0f, -1.0f,
   1.0f, -1.0f, -1.0f,
   1.0f, -1.0f,  1.0f,
};...
```

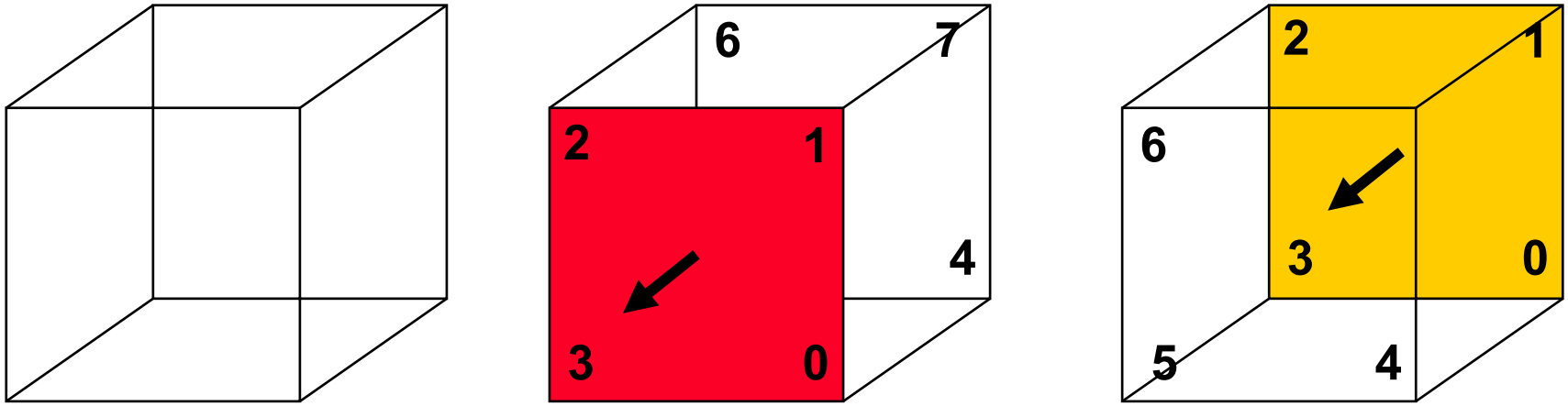
Beispiel 2: ColorCube (2)

```
 private static final float[] colors = {
    // front face (red)
  1.0f, 0.0f, 0.0f,
  1.0f, 0.0f, 0.0f,
  1.0f, 0.0f, 0.0f,
  1.0f, 0.0f, 0.0f,
    // back face (green)
  0.0f, 1.0f, 0.0f,
  0.0f, 1.0f, 0.0f,
  0.0f, 1.0f, 0.0f,
  0.0f, 1.0f, 0.0f,
    // right face (blue)
  0.0f, 0.0f, 1.0f,
  0.0f, 0.0f, 1.0f,
  0.0f, 0.0f, 1.0f,
  0.0f, 0.0f, 1.0f,
... };

public ColorCube() {
   QuadArray cube = new QuadArray(24, QuadArray.COORDINATES |
       QuadArray.COLOR_3);
   cube.setCoordinates(0, verts);
   cube.setColors(0, colors);
   this.setGeometry(cube);
}
```

Oberflächennormalen

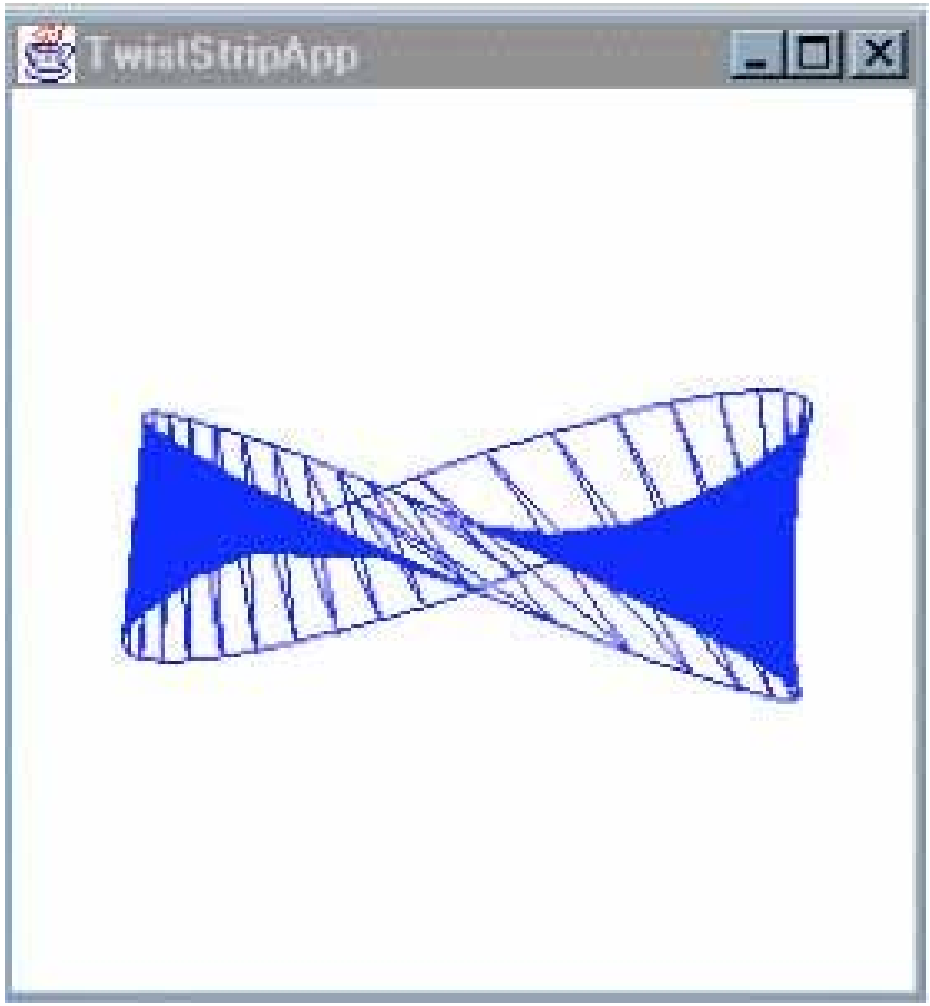
- Wo ist „vorne“ bei einem Polygon?
  - Wichtig, um z.B. verdeckte Rückseiten nicht rendern zu müssen
- Grundregeln in Java 3D:
  - Vorne ist die Seite, zu der der Normalenvektor zeigt
  - Von „vorne“ erscheinen die Ecken in einer Reihenfolge *gegen den Uhrzeigersinn (counter-clockwise)*



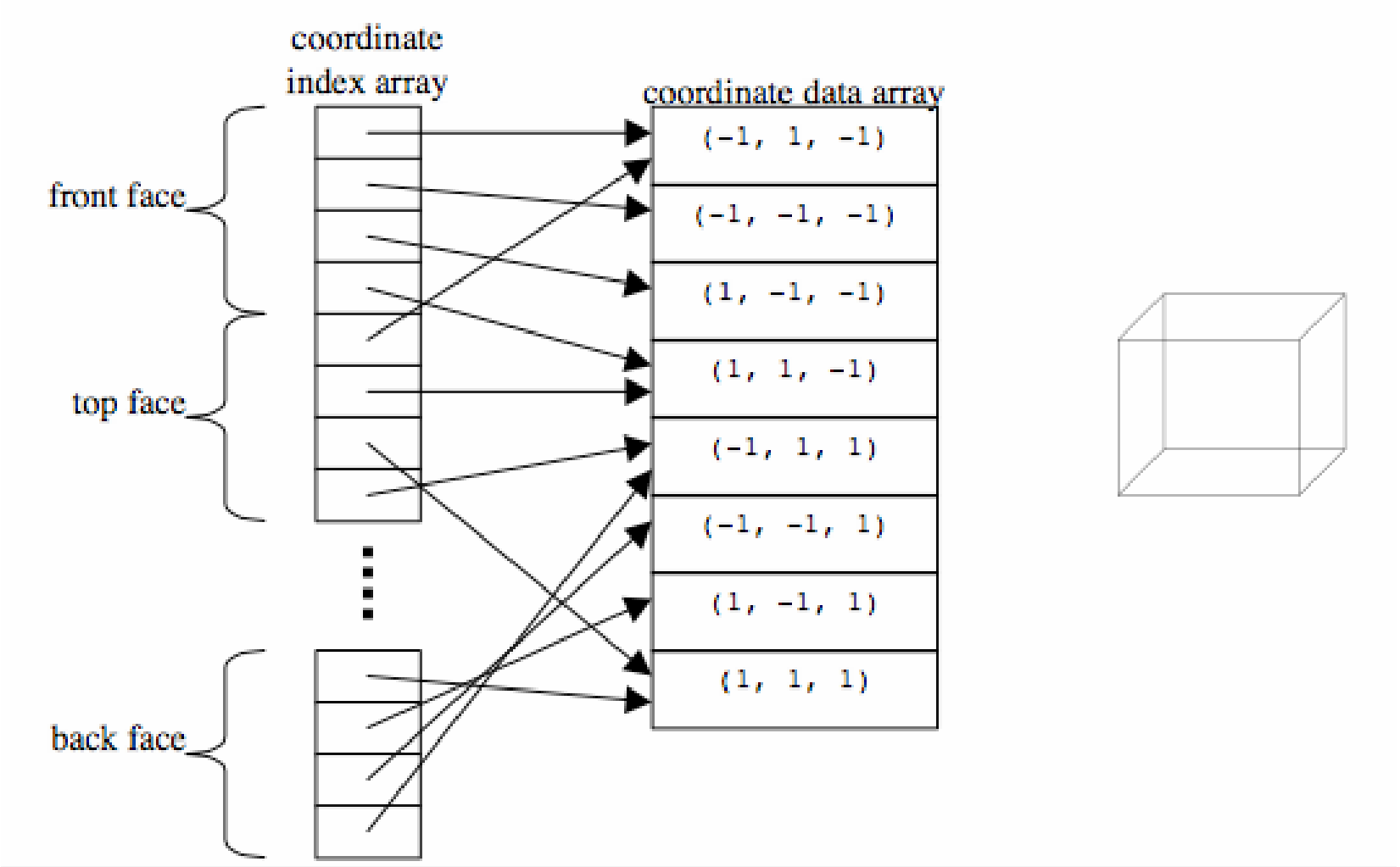
Culling (Verdeckung)

- Im Normalfall müssen verdeckte Seiten nicht gerendert werden
  - Optimierung der Berechnungszeit für das Rendering
- Culling-Strategien:
  - *„Cull Backface“:*
    - » Rückseiten werden nicht gerendert
    - » Standard-Modus
  - *„Cull Frontface“:*
    - » Vorderseiten werden nicht gerendert
  - Culling ausschalten

Beispiel: Möbius-Band

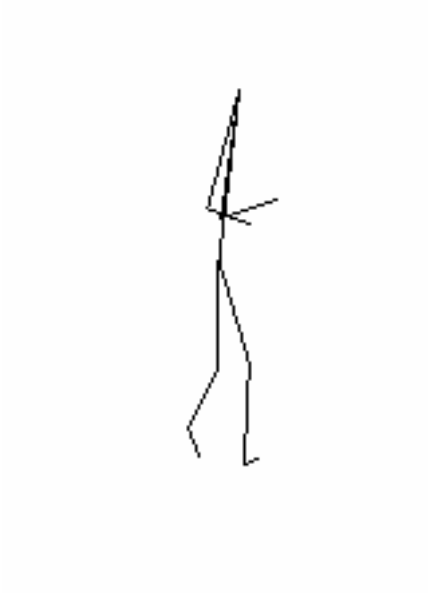


Würfel als IndexedArray



Morphing

- *Morphing* ist die Interpolation von Formen über die Zeit
  - Spezieller Interpolator
  - Benutzt Array von Geometrien
- Zeitgesteuertes Morphing ermöglicht die Animation von Figuren und Bewegungen
  - Getrieben meist über Alpha-Objekte



B6. 3D-Computergrafik mit Java

B6.1 Grundlagen der 3D-Computergrafik

B6.2 Einführung in Java 3D

B6.3 Animation

B6.4 Geometrie, Material, Beleuchtung (Forts.)

B6.5 Interaktion 

Literatur:

D. Selman: Java 3D Programming, Manning 2002

A. E. Walsh, D. Gehringer: Java 3D API Jump-Start, Prentice-Hall 2001

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

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

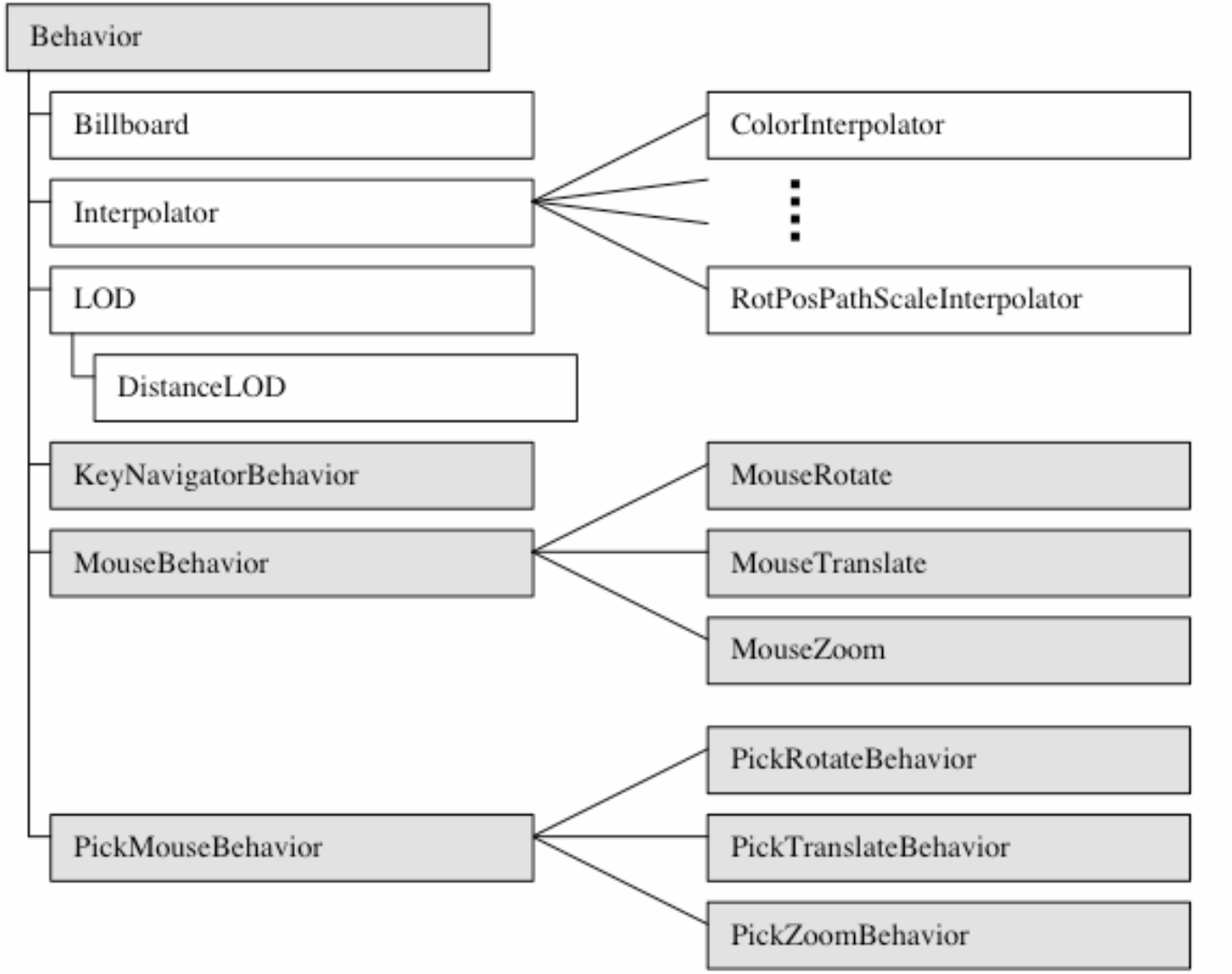
http://java.sun.com/docs/books/java3d/

http://www.j3d.org, http://www.java3d.org/

Verhalten *(behavior):* Überblick

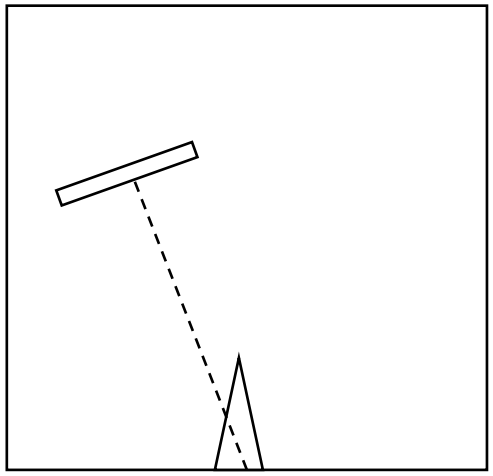
stimulus (reason for change)	**object of change**			
	TransformGroup (visual objects change orientation or location)	Geometry (visual objects change shape or color)	Scene Graph (adding, removing, or switching objects)	View (change viewing location or direction)
user	interaction	application specific	application specific	navigation
collisions	visual objects change orientation or location	visual objects change appearance in collision	visual objects disappear in collision	View changes with collision
time	animation	animation	animation	animation
View location	billboard	level of detail (LOD)	application specific	application specific

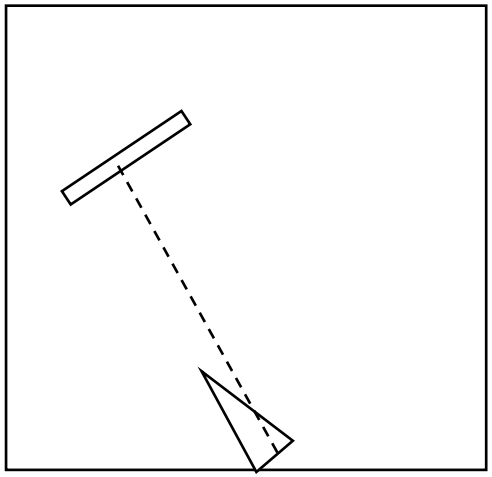
Behavior-Klassen in Java 3D



Billboard

- Ein *Billboard* ist ein graphisches Element, das sich immer nach der Betrachterposition ausrichtet
  - z.B. Textuelle Erläuterungen
  - z.B. Zweidimensionale Gebilde, die nur von einer Seite betrachtbar sind
- Durch Utility-Klassen in Java 3D unterstützt





javax.media.j3d.Behavior

- Abstrakte Klasse zur Definition von benutzer-initiierten Veränderungen in einem Java 3D- Szenengraph
- *Scheduling Region:*
  - Ähnlich zu Animationen, sind Interaktionen *(behaviors)* nur aktiv *(active)*, wenn ein Betrachter in der Nähe ist *(activation volume intersection)*
- *Scheduling Interval:*
  - Dient zur Festlegung der Reihenfolge „gleichzeitig“ ausgelöster Ereignisse
- Abstrakte Methoden der abstrakten Klasse:
  - **`public abstract void initialize()`**
    - » Vor allem zur Definition von Aufwachbedingungen *(wakeup criteria)*
  - **`public abstract void processStimulus(java.util.Enumeration criteria)`**
    - » Zur Ausführung der tatsächlichen Aktionen
    - » Kann Aufwachbedingungen verändern

Aufwachbedingungen (Beispiele)

- Eine ViewingPlatfrom betritt/verlässt eine bestimmte Region
- Auslösen eines bestimmten Ereignisses
- Ablauf einer Zeitbedingung
- Ein bestimmtes Verhalten wird aktiviert/deaktiviert
- Die Transformation einer TransformGroup wird verändert
- Kollisionserkennung/-aufhebung
- Bewegung zwischen dem zugehörigen Geometrieobjekt und einem potentiellen Kollisionsobjekt
- ...

Beispiel: SimpleBehavior

```
public class SimpleBehavior extends Behavior{

  private TransformGroup targetTG;
  private Transform3D rotation = new Transform3D();
  private double angle = 0.0;

  SimpleBehavior(TransformGroup targetTG){
    this.targetTG = targetTG;
  }

  public void initialize(){
    this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
  }

  public void processStimulus(Enumeration criteria){
    angle += 0.1;
    rotation.rotY(angle);
    targetTG.setTransform(rotation);
    this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
  }
}
```

Veränderung von Szene und Betrachter

- Interaktion zur Veränderung der Szene
  - z.B. Auslösen einer Animation
  - z.B. Veränderung der Darstellung eines (komplexen) Objekts
    - » Level of Detail
    - » Einblendung/Ausblendung von Teilen des Objekts
    - » Einblendung/Ausblendung von Zusatzinformation
- Interaktion zur Veränderung des Betrachterstandpunkts
  - Natürlichste Form der Interaktion mit einer 3D-Szene
  - In Dokumentsprachen wie VRML durch Browser/Viewer realisiert
  - In Java 3D: Standardklassen für *Keyboard Navigation*
    - » Ohne weiteres durch spezielle Implementierungen ersetzbar/erweiterbar
- Kombination von Animation und Keyboard Navigation:
  - beliebt zur Programmierung von Spielen

Beispiel: KeyNavigatorApp

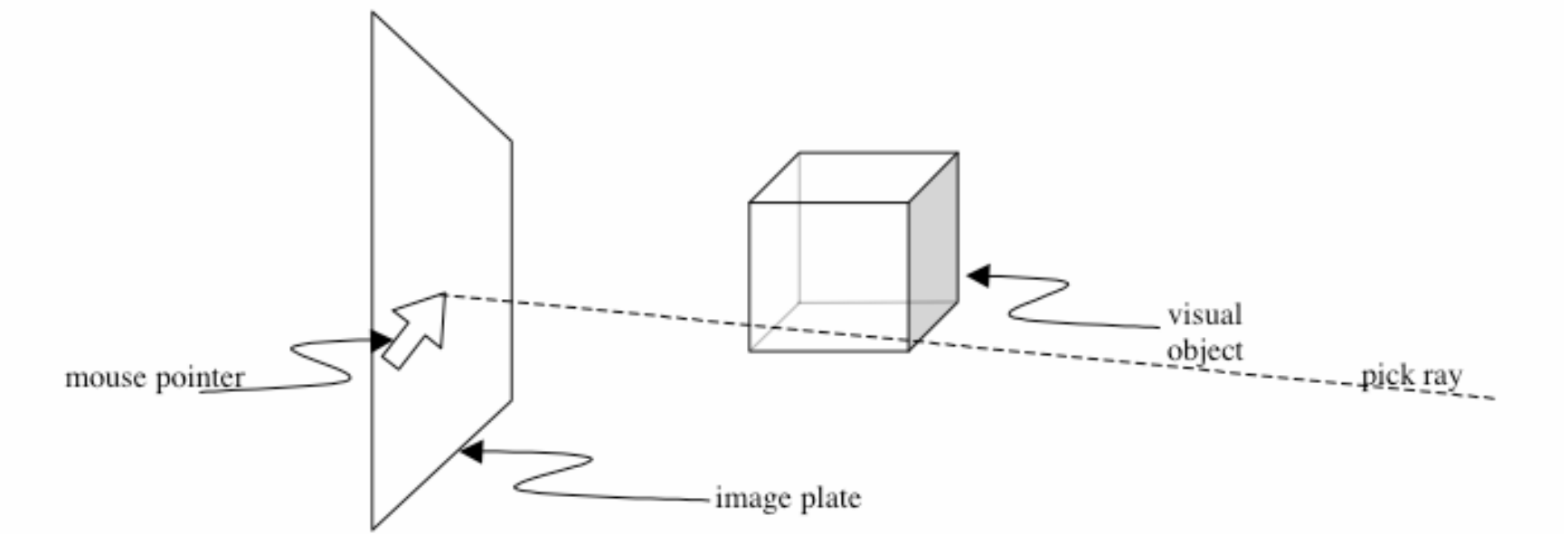
```
public BranchGroup createSceneGraph(SimpleUniverse su) {
  ...
  TransformGroup vpTrans = null;
  Transform3D T3D = new Transform3D();
  Vector3f translate = new Vector3f();
  translate.set( 0.0f, 0.3f, 0.0f);
  ...
  vpTrans =
    su.getViewingPlatform().getViewPlatformTransform();
  T3D.setTranslation(translate);
  vpTrans.setTransform(T3D);
  KeyNavigatorBehavior keyNavBeh =
    new KeyNavigatorBehavior(vpTrans);
  keyNavBeh.setSchedulingBounds
    (new BoundingSphere(new Point3d(),1000.0));
  objRoot.addChild(keyNavBeh);
  ...
}
```

MouseBehavior

- Analog zu KeyboardBehaviour ist es (durch Standardklassen) möglich, auf Mausgesten zu reagieren
  - Bewegen des Standpunkts in allen Freiheitsgraden
  - Bewegen des betrachteten Objekts
  - Zoomen
- Relevante Klassen:
  - MouseRotate
  - MouseTranslate
  - MouseZoom
- Mögliche Reaktion auf Benutzerinteraktion:
  - Standardverhalten gemäß benutztem Behavior-Objekt
  - Spezifische selbstprogrammierte *callback*-Methoden

Auswahl *(picking)*

- Benutzer wählt Objekte der virtuellen Szene durch Mausklick aus
  - Auswahl des nächstgelegenen Objekts innerhalb gewisser Toleranz
- Dreidimensionale Auswahl:
  - Strahl von der Mauszeigerposition parallel zu den aktuellen Projektionslinien zum Betrachter
  - Auswahl des Objekts, das vom Auswahlstrahl geschnitten wird und am nächsten zum Betrachter liegt



Beispiel: MousePickApp.java

```
...
public BranchGroup createSceneGraph(Canvas3D canvas) {
   BranchGroup objRoot = new BranchGroup();
   ...
   PickRotateBehavior pickRotate = null;
   Transform3D transform = new Transform3D();
   BoundingSphere behaveBounds = new BoundingSphere();
   transform.setTranslation(new Vector3f(-0.6f, 0.0f, -0.6f));
   objRotate = new TransformGroup(transform);
   objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
   objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
   objRotate.setCapability(TransformGroup.ENABLE_PICK_REPORTING);
   objRoot.addChild(objRotate);
   objRotate.addChild(new ColorCube(0.4));
   pickRotate =
     new PickRotateBehavior(objRoot, canvas, behaveBounds);
   objRoot.addChild(pickRotate);
   ...
   objRoot.compile();
   return objRoot;
}
```