3D Modellierung

Masterstudium Informatik
Vertiefungsrichtung Computer Graphics
Sommersemester 2008

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

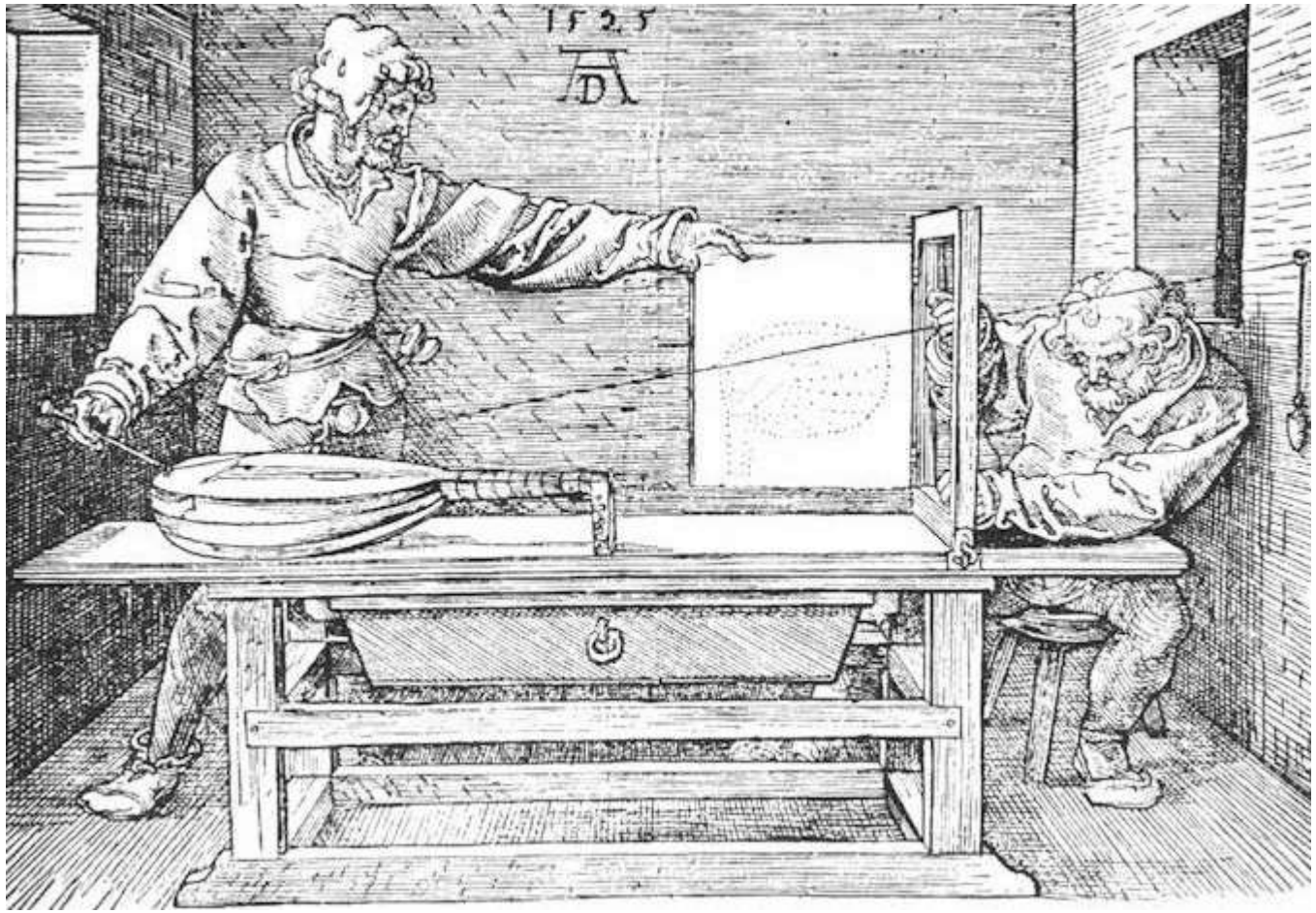


Bild aus der Abhandlung "Underweysung der Messung" von Albrecht Dürer, 1525

… knapp 500 Jahre später

… beschäftigt der Klimawandel auch die Animationsfilme

Übersicht

- Motivation, Ziele, typische Aufgabenstellungen
- Koordinatenräume, Transformationen
- Visualisierungspipeline, Berechungsgrundlagen
- 3D-Primitive, -Datenstrukturen, 3D-API, Szenegraphen
- Implizites Modellieren, Subdivision Surfaces
- Formate, Systeme
- Konstruktion komplexer graphischer Objekte: B-Rep, Sweeping, CSG, Modellierungsüberlegungen, Implementierungskonzept
- Objektidentifikation, -umgebungen: Bounding Box
- Nearest-Neighbour-Problem
- BSP - Objekthierarchien:
  Median-Cut, Popularity, Grid-Datenstruktur, Oct-Trees
- Freiformkurven und –Flächen (Spline, Bézier, de Casteljau)

Literatur

- Foley, vanDam, Feiner, Hughes:
  Computer Graphics - Principles and Practice Second Edition in C.
  Addison-Wesley Longman Publishing Company. ISBN 0-201-84840-6.
- Foley, vanDam, Feiner, Hughes:
  Grundlagen der Computergraphik - Einführung, Methoden, Prinzipien
  (Übersetzung aus dem Amerikanischen)
  Addison-Wesley Longman Verlag. ISBN 3-89319-647-1.
- Encarnacao, Straßer:
  Graphische Datenverarbeitung 1 + 2 (2 Bände). 4. Auflage, Oldenbourg Verlag. ISBN 3-486-23223-1
- Edward Angel:
  Interactive Computer Graphics – A Top-Down Approach Using OpenGL
  Addison-Wesley. ISBN 0-201-77343-0
- Allan Watt, Mark Watt:
  Advanced Animation and Rendering Techniques – Theory and Practice
  Addison-Wesley. ISBN 0-201-54412-1
- Michael Bender, Manfred Brill:
  Computergrafik - Ein anwendungsorientiertes Lehrbuch
  2. Auflage, Hanser Verlag. ISBN 3-446-40434-1

- Dennis J. Bouvier, sun microsystems
  Getting Started with the Java 3D API- A Tutorial for Beginners
- Java3D Slide Set (Siggraph 99):
  http://www.sdsc.edu/~nadeau/Courses/Siggraph99/
- http://www.web3d.org; http://www.j3d.org

Motivation, Ziele

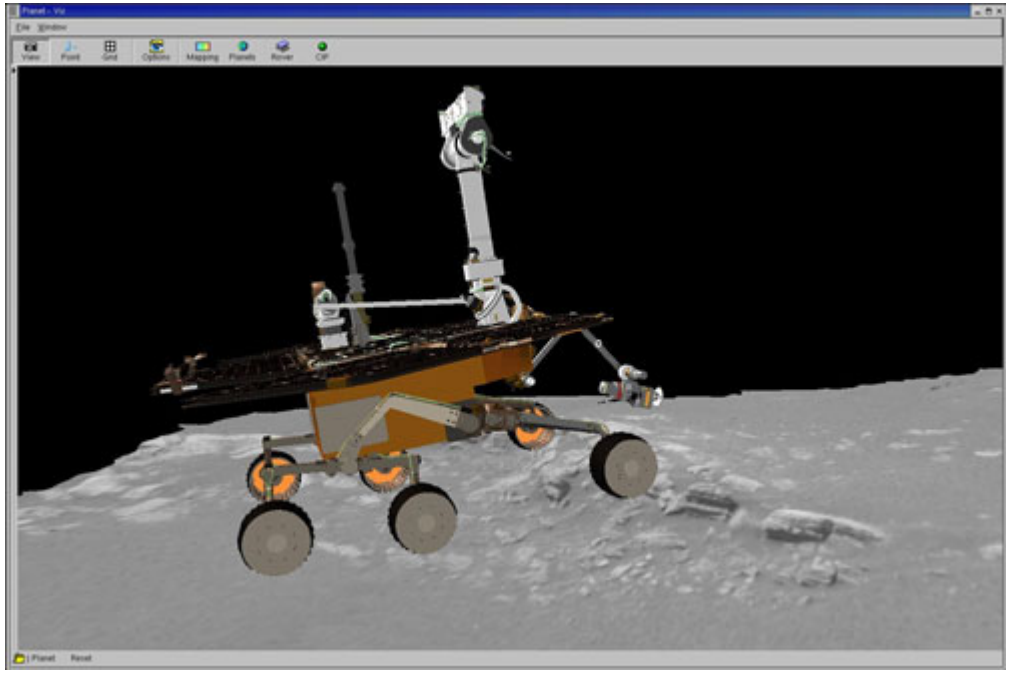
- „The screen is a window through one sees a virtual world. The challenge is to make that world look real, act real, sound real, feel real."
  I. E. Sutherland, 1965
- „Imagine a time far into the future, when all knowledge about our civilization has been lost. Imagine further, that in the course of planting a garden, a fully stocked computer store from the 1980's was unearthed, and that all of the equipment and software was in working order. Now, based on this find, consider what a physical anthropologist might conclude about the physiology of the humans of our era? My best guess is that we would be pictured as having a well-developed eye, a long right arm, a small left arm, uniform length fingers and a 'low fi' ear. But the dominating characteristics would be the prevalence of our visual system over our poorly developed manual dexterity."
  W. Buxton, 1986
- „Modellieren Sie Spaghetti – und vergessen Sie die Soße nicht"
  Jim Blinn, SIGGRAPH ,98, Orlando

Anwendungsgebiete

- Neue Medien: Internet, Intranet, Online-Systeme
- E-commerce, Wertpapiermanagement – allgemein: Aufbereitung großer Datenmengen und zeitnaher Zugang
- Data Mining
- Druckbranche
- Filmindustrie, Entertainment (Spiele) → Augmented Reality
- Scientific Visualisation: Wetterdaten →Virtual Reality
- Simulationen und Tests → Digital Mock Up (DMU) (z.B. CAD-Techniken, Produktdaten, Architektur)
- Prozessvisualisierung techn. Anlagen: Monitoring und Visualisierung
- Bürokommunikation – Fenstersysteme (→ GUI)
- Medizin, Bioinformatik, Drug Design → Visualisierung
- Qualitätskontrolle → Bildverarbeitung
- Geographische Informationssysteme: Katasterdaten, Verkehrsinformationen, Facility Management, ...

Virtueller Marsflug

- Aus einer VR-Umgebung heraus werden die Bewegungen von den Marssonden Spirit und Opportunity simuliert und gesteuert
- Über die Distanz von etwa 200 Millionen km ist Echtzeitkontrolle nicht möglich
- Die Signallaufzeit beträgt 20 Minuten



3D Modellierung

bedeutet in der Umsetzung

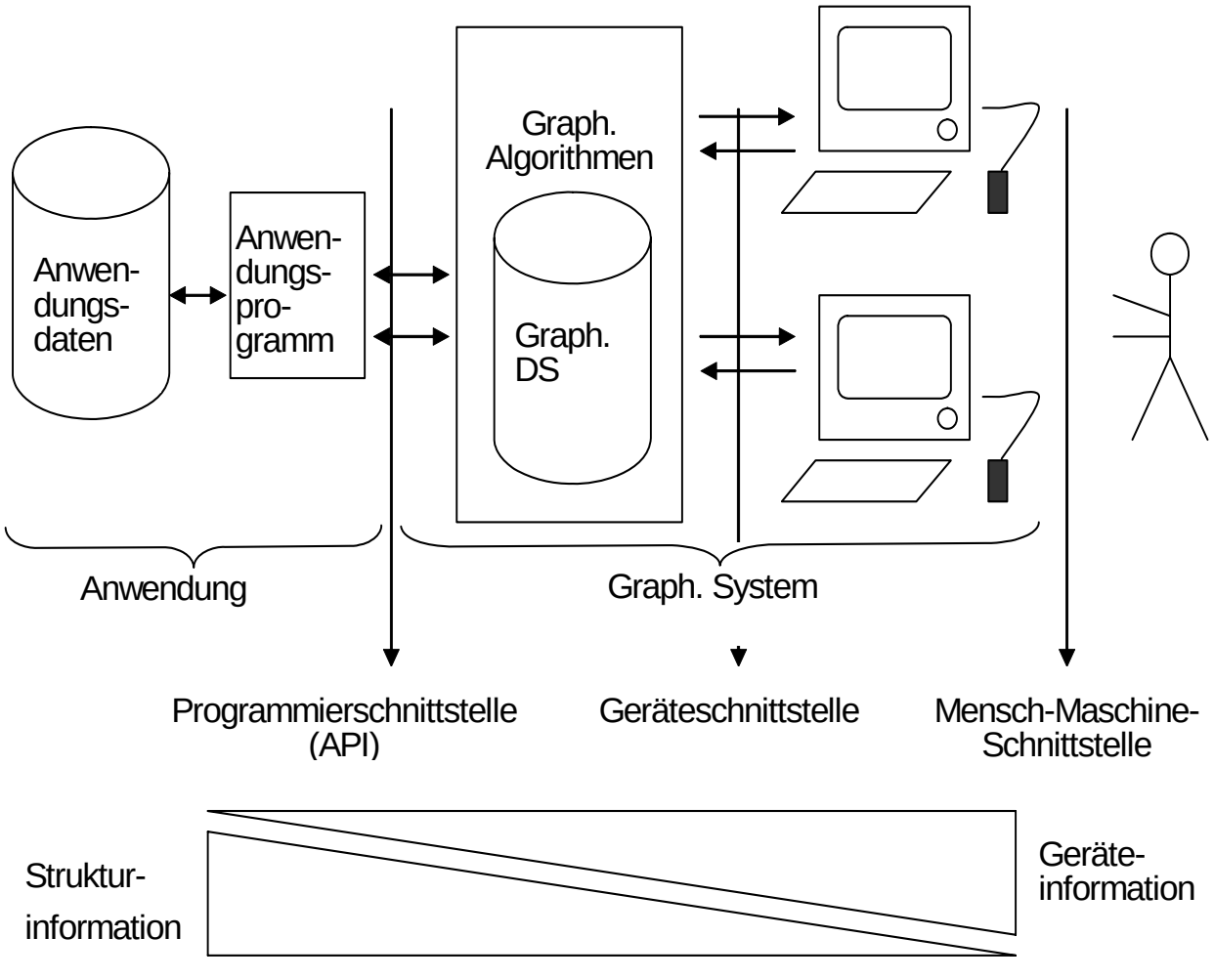
- Auseinandersetzung mit Anwendungswissen, Anwendungsentwicklung und -programmierung
- Beschäftigung mit (Klassen-)Bibliothek Programmierung von Graphikpaketen, Treibern (= Schnittstellenprogrammierung)
- Auseinandersetzung mit Mensch-Maschine-Schnittstelle
  - Modellierung
  - Visualisierung (3D)
  - Interaktion (GUI)

➔ Graphisch-interaktive Systeme

Wie können Experten gute (= intuitiv zu verstehende, leicht zu bedienende) Systeme für jedermann bauen

- Aufbereitung (sehr) großer, komplexer Datenmengen, die nicht unmittelbar zugänglich sind
- Anwendungsdaten sind 3-dimensional!
- Umgang mit 3D-Objekten ist intuitiver, attraktiver, spannender → Power-User, exploratives Arbeiten
- Einsatz von Mathematik

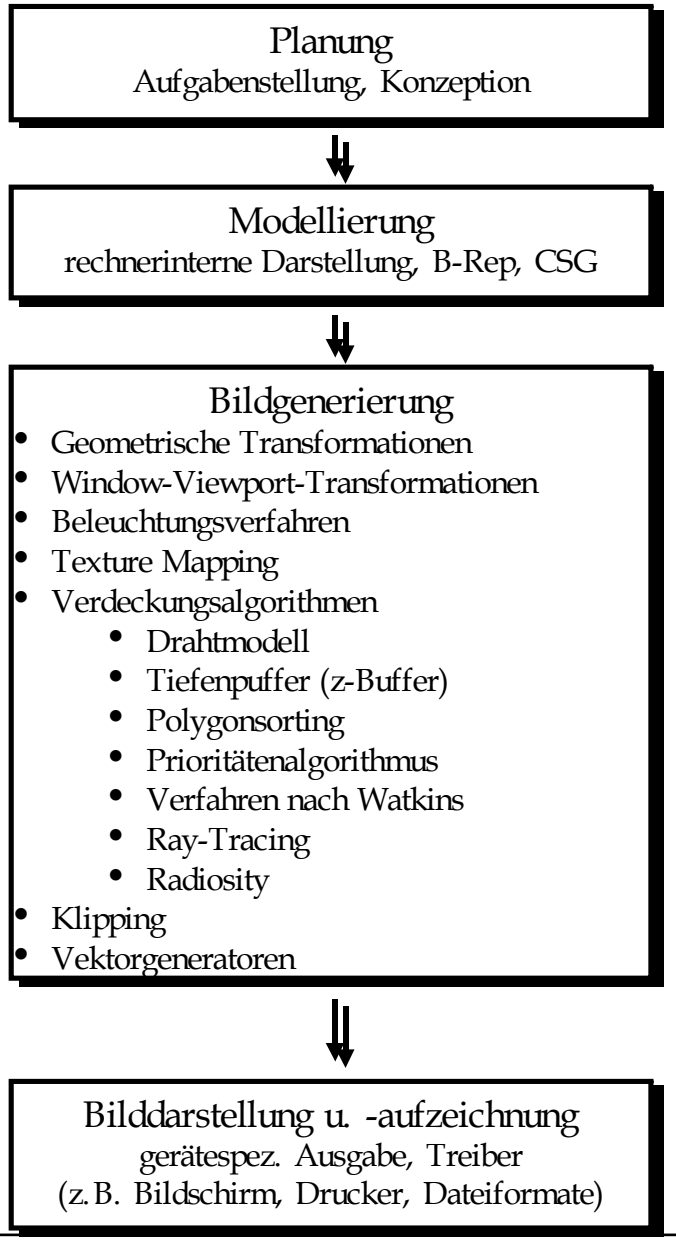
Modell eines Graphischen Systems



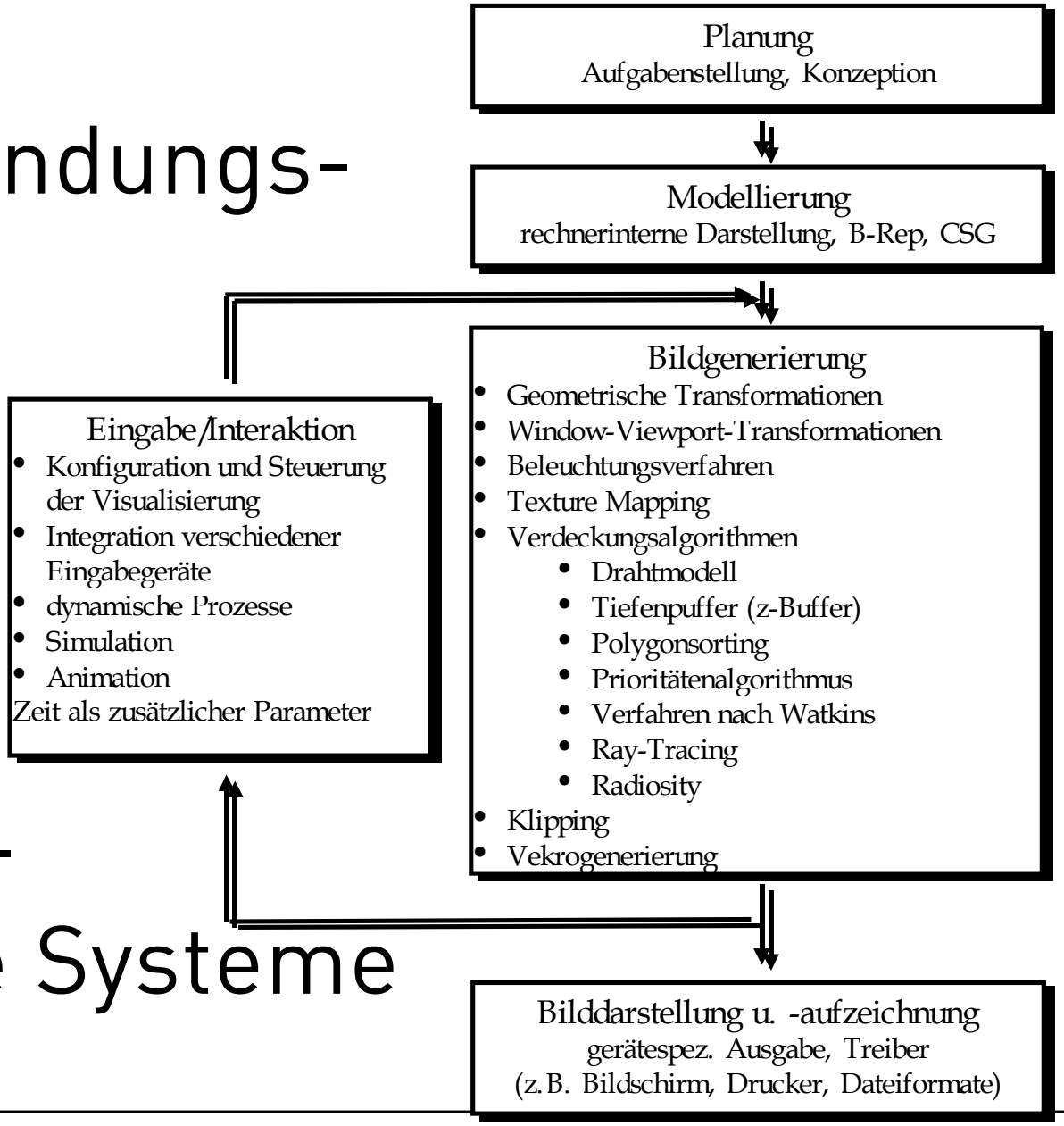
Vom Anwendungsobjekt zum Pixel

- Aufgabe ist nun, dieses rechnerinterne Modell, das in aller Regel als Flächen- oder Kantenmodell gegeben ist, in einfachere Objekte zu zerlegen (Dreiecke oder Scanlines) und auf einem i.d.R. pixelorientierten Ausgabegeräte darzustellen.
  Dazu müssen die geometrischen Eigenschaften eindeutig beschrieben werden.
- Auf dieses Modell wird eine räumliche Transformation angewandt, welche dem Beobachterstandpunkt und der gewünschten Blickrichtung sowie dem darzustellenden Ausschnitt (Szene) Rechnung trägt.
  Anschließend werden Linien und Oberflächen, welche durch andere Objekte verdeckt werden oder aus der aktuellen Blickrichtung nicht sichtbar sind, ermittelt und die verdeckten Teile zumindest „markiert“. Die übrig bleibenden Objekte werden eingefärbt; der Schattenwurf und Schattierung wird berechnet.

Vom Anwendungs-objekt zum Pixel: Ausgabepipeline

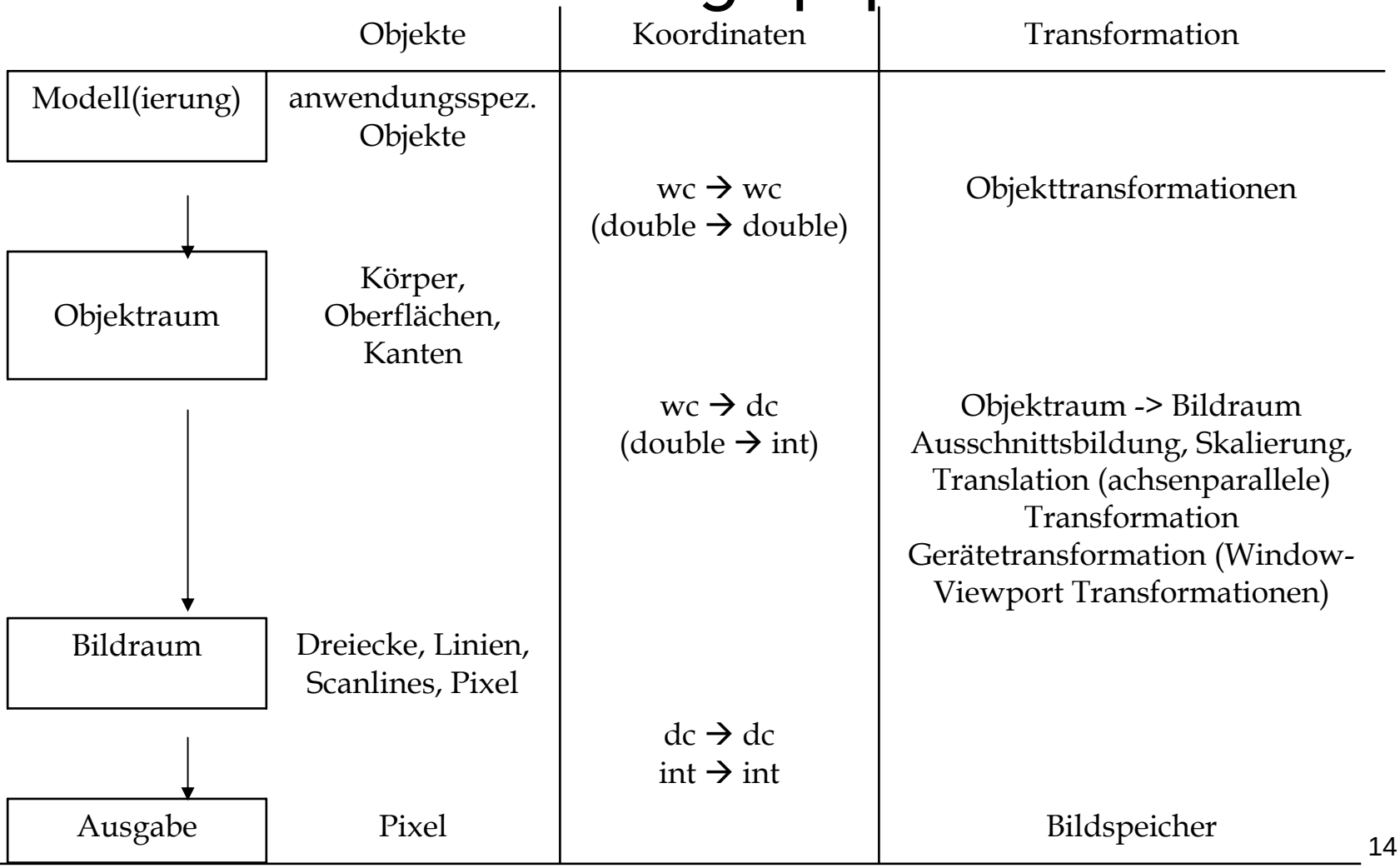


Vom Anwendungs-objekt zum Pixel:



graphisch-interaktive Systeme

Visualisierungspipeline



Grundlegende Überlegungen (I)

- Beschreibung der Objekte anhand markanter Merkmale
- Geometrie
  - Referenzpunkt – Position im Raum
  - Lokales/objektbezogenes Koordinatensystem
  - Orientierung, Ausrichtung
  - Objektdefinition basiert auf der Vektoralgebra (Punkte, Kanten, Flächen)
- Erscheinung
  - Beleuchtung, Farbe, Textur, Transparenz, ...
  - Visualisierungspipeline berechnet Sichtbarkeit/Verdeckung → 2D Eindruck am Bildschirm

Grundlegende Überlegungen (II)

- Alles, was im 2D betrachtet wurde, muss entsprechend auf 3D übertragen werden:
  - Clipping Plane wird Clipping Box
  - Homogene Koordinaten (3D → 4D)
  - Objekttransformationen (3x3 Matrix → 4x4 Matrix)
  - Neben Punkten (point, vertex), Kanten (edge) und Flächen (plane, patch, surface) kommen 3D Objekte (box, cone, cylinder, sphere, ...) hinzu

Berechnungsgrundlagen Mathematische Überlegungen

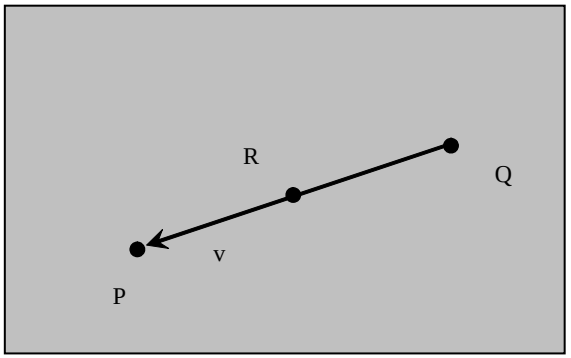
- R = Q + t * v, mit v = P – Q

  → Gerade: $t \in \mathfrak{R}$

  → Strahl: t >= 0

  → Linie/Kante: 0 <= t <= 1

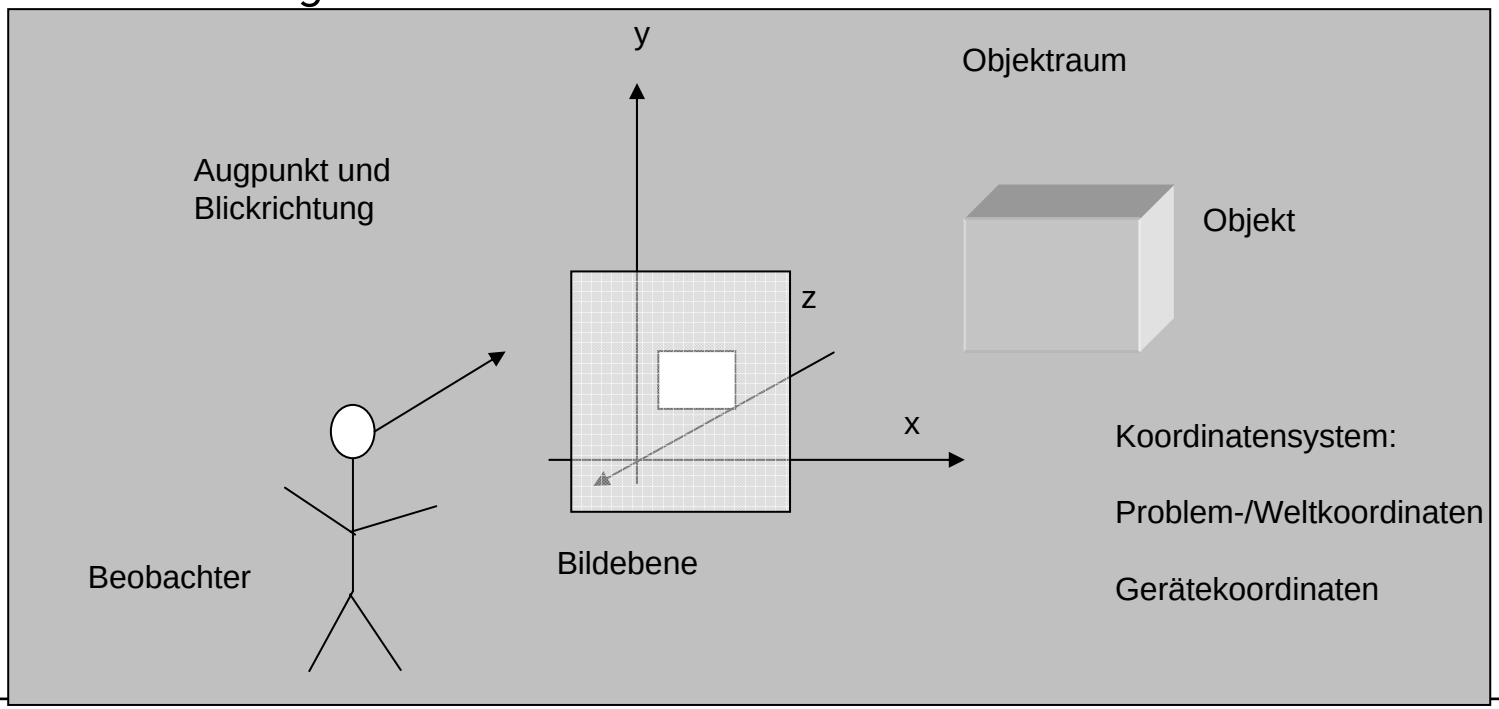
Skizze



- Die Angaben beziehen sich immer auf ein bestimmtes Koordinatensystem (auch wenn die Achsen oder der Ursprung nicht angegeben sind)

Szene

- Menge aller Objekte (Geometrie- und Erscheinungseigenschaften)
- Definition eines Beobachters/einer Kamera (Augpunkt, Blickrichtung)
- Beleuchtungsmerkmale (Lichtquelle, Schattenwurf, Oberflächeneigenschaften)



Affine Transformationen

Wird die Ebene in kartesischen Koordinaten beschrieben, dann wird durch Abbildungsgleichungen der Form

$$x' = a_1 * x + b_1 * y + c_1 * z + t_1$$
$$y' = a_2 * x + b_2 * y + c_2 * z + t_2$$
$$z' = a_3 * x + b_3 * y + c_3 * z + t_3$$

In Matrixschreibweise: $\vec{v}' = \vec{v}^t * M + \vec{T}$

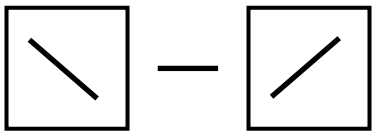
$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}^t * \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

eine affine Transformation festgelegt. Diese ist bijektiv, d.h. umkehrbar, wenn die Determinante von M <> 0. Dies ist insbesondere vorteilhaft, wenn man aus den Zielkoordinaten die Ursprungskoordinaten zurück rechnen muss (etwa bei einer Objektidentifikation).

Determinante

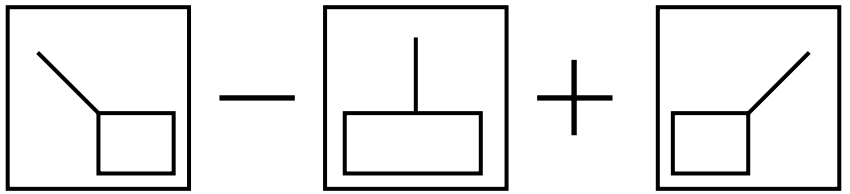
- Bemerkung:
  die Determinante kann wie folgt berechnet werden:
  - im 2D: $\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = a * d - b * c$

    Prinzip:

    
  - im 3D:

$$\det \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = a * \det \begin{pmatrix} e & f \\ h & i \end{pmatrix} - b * \det \begin{pmatrix} d & f \\ g & i \end{pmatrix} + c * \det \begin{pmatrix} d & e \\ g & h \end{pmatrix}$$

    Prinzip:

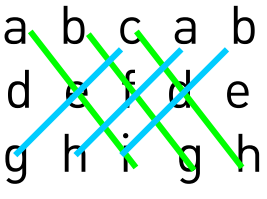
    

Determinante

Alternatives Berechnungsschema:

- Für 3x3 Matrizen:

$$\det \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{array}{ccccc} a & b & c & a & b \\ d & e & f & d & e \\ g & h & i & g & h \end{array}$$



➔ a * e * i + b * f * g + c * d * h – g * e * c – h * f * a – i * d * b

Beispiel Determinanten

✍ Aufgabe:

Berechnen Sie zu den folgenden Matrizen die Determinante und interpretieren Sie das Ergebnis:

a) $\det \begin{pmatrix} 1 & 1 & 1 \\ 1 & 4 & 0 \\ 3 & 1 & 1 \end{pmatrix}$

b) $\det \begin{pmatrix} 1 & 2 & 2 \\ 4 & 1 & 0 \\ 1 & 3 & 2 \end{pmatrix}$

c) $\det \begin{pmatrix} 2 & 4 & 2 \\ 1 & 2 & 0 \\ 3 & 6 & 2 \end{pmatrix}$

Homogene Koordinaten

- Übergang von 3D nach 4D
  3D Transformationen werden im 4D beschrieben und mit Vorteilen für den gesamten Prozess gelöst:

$$v = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

  w hat üblicherweise den Wert 1.0

- Die Einführung von homogenen Koordinaten erlaubt **die Gleichbehandlung der drei Elementartransformationen**
  Translation T, Skalierung S und Rotation R.

Skalierung

Im folgenden betrachten wir die Elementarfunktionen im 3D.

Skalierung/Zoom (Bildausschnitt verändern):

$$x = s_x * x$$
$$y = s_y * y$$
$$z = s_z * z$$
$$w = 1$$

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(x, y, z, 1)^t * S = (x* s_x , y * s_y , z * s_z, 1)$$

Translation

Translation (Verschiebung)

$$x = x + x_t$$
$$y = y + y_t$$
$$z = z + z_t$$
$$w = 1$$

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_t & y_t & z_t & 1 \end{pmatrix}$$

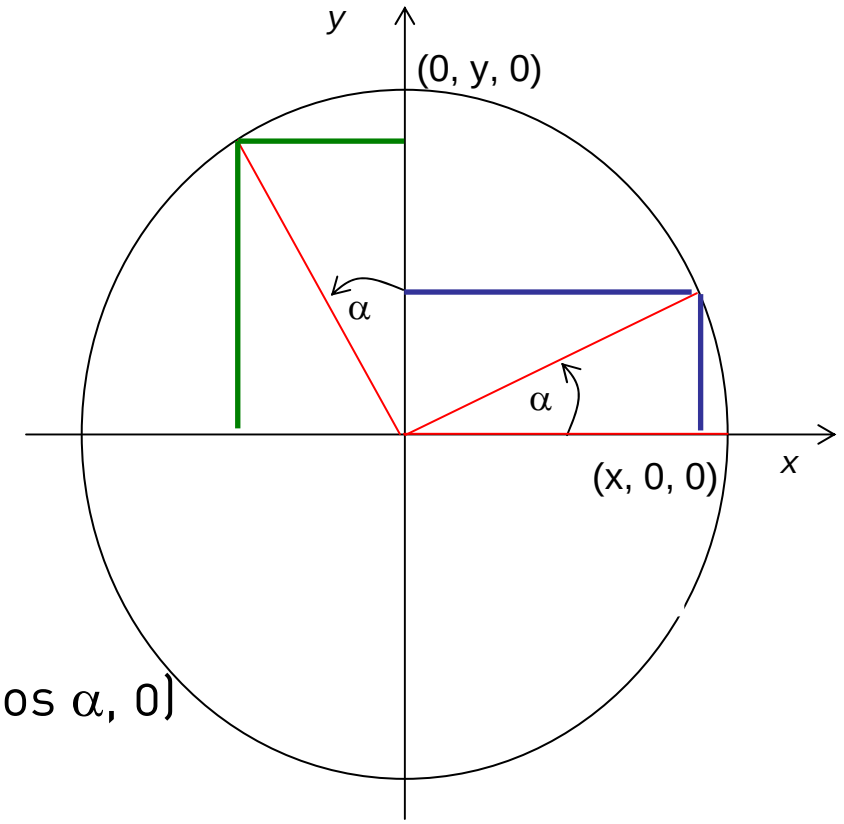
$$(x, y, z, 1)^t * T = (x + x_t, y+y_t, z+z_t, 1)$$

Rotation

Skizze und Überlegungen:

- der Punkt (x, 0, 0) wird überführt in
  (x * cos α, x * sin α, 0)

- für gilt (0, y, 0) gilt:
  (-y * sin α, y * cos α, 0)



→ Koeffizientenvergleich liefert

(x,y,0) →
(x*cos α - y*sin α, x*sin α + y*cos α, 0)

Zur Erinnerung:
sin α = Gegenkatete/Hypotenuse
cos α = Ankatete/Hypotenuse

Rotation um z-Achse

Rotation (Drehung) um den Nullpunkt:

Die z-Achse zeigt zum Beobachter hin

Drehung um einen Winkel bewirkt folgende Veränderung:

$$x = x * \cos\alpha - y * \sin\alpha$$

$$y = x * \sin\alpha + y * \cos\alpha$$

$$z = z$$

$$w = 1$$

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(x, y, z, 1)^t * R_z = (x*\cos\alpha - y*\sin\alpha, x*\sin\alpha + y * \cos\alpha, z, 1)$$

Rotation um y-Achse

Rotation (Drehung) um den Nullpunkt:

- Die y-Achse zeigt zum Beobachter hin
- Drehung um einen Winkel bewirkt folgende Veränderung:

$$R_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um x-Achse

Rotation (Drehung) um den Nullpunkt:

- Die x-Achse zeigt zum Beobachter/zur Kamera hin
- Drehung um einen Winkel bewirkt folgende Veränderung:

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bemerkung: in der Literatur ist oft das sin-Vorzeichen vertauscht. Achten Sie auf die Schreibweisen; die **Matrizen** sind dann **transponiert** dargestellt!

Transformationen

- Das invariante Format erleichtert die Implementierung.
- Komplexe Transformationen können so auf aufeinander folgende Elementartransformationen zurückgeführt werden (Konstruktiv)
- Beispiel:

  Erg = R * T * S

- Die Hintereinanderausführung verschiedener Transformationen (**Concatenation**) erlaubt die Akkumulation in einer Ergebnismatrix. Die Berechnung ist dann nur noch eine einfache Matrixmultiplikation.
- Wichtig dabei allerdings ist, dass dabei die **Reihenfolge eine bedeutende Rolle** spielt.
- Machen Sie sich die Rechengesetze (Distributivgesetz, Assoziativgesetz, Kommutativgesetz) deutlich; überlegen Sie verschiedene Beispiele.
  Es sind allerdings nur ganz bestimmte Transformationen kommutativ!

Beispiel Transformationen

Aufgabe (2D):

Leiten Sie die Transformationsmatrix für folgende Transformationen her:

a) Spiegelung an der x-Achse
b) Rotation um den Winkel α (entgegen dem Uhrzeigersinn)
c) Spiegelung an der Geraden

$$y = x * \sqrt{3} = x * \tan 60°$$

Hinweis: Setzen Sie diese Transformation aus mehreren, nacheinander anzuwendenden Transformationen zusammen

Aufgabe (2D):

Entwickeln Sie die Transformationsmatrix, mit der das Quadrat

Q = ((-1,-1), (+1, -1), (+1, +1), (-1, +1))

in das Rechteck

R = ((2, 4), (4, 2), (8, 6), (6, 8))

überführt werden kann.

3D Primitive, Datenstrukturen

Konstruktion als Prinzip der Modellierung:

- Ein einfacher und doch redundanzfreier Ansatz zur rechnerinternen Beschreibung und Speicherung von Polyedern besteht darin, pro Körper jeweils eine Flächen-, Kanten- und Punkteliste zu halten, in welchen die den Körper bestimmenden Oberflächen, Kanten und Eckpunkte enthalten und somit die Objekte lediglich nur einmal gespeichert sind.

- Jedes Element der Kantenliste enthält eine Referenz auf die beiden Endpunkte; dies ist gerade in einer objektorientierten Programmiersprache gut zu realisieren.

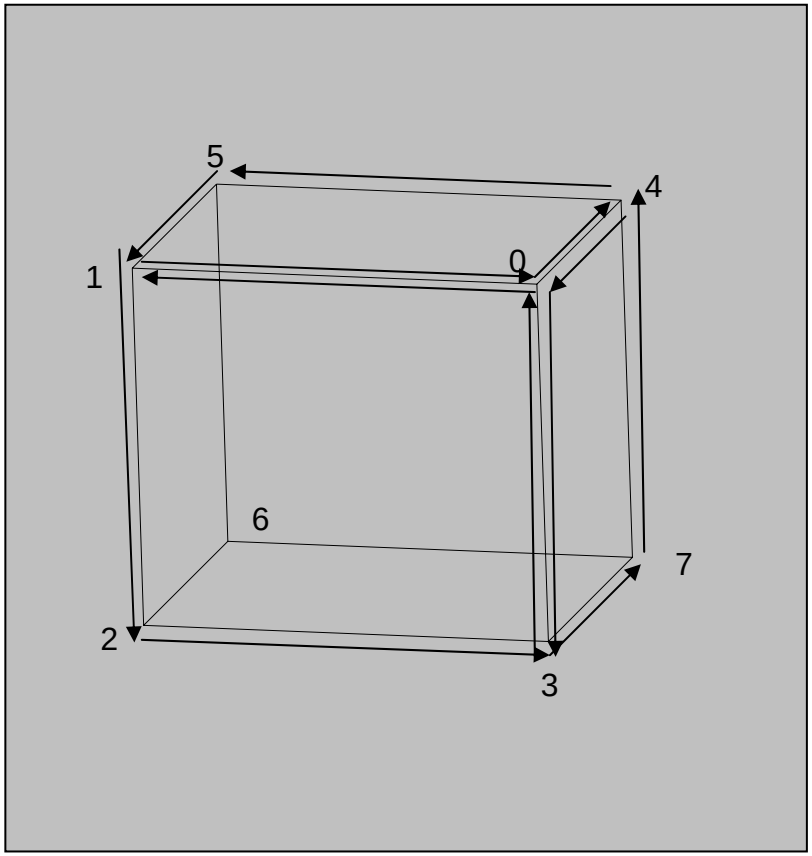
- Wie Bauklötze oder Legosteine können komplexe Objekte aus einfachen zusammengesetzt werden.

Box / Cube (I)

- Als einfaches aber ausführliches Beispiel dient der Würfel:

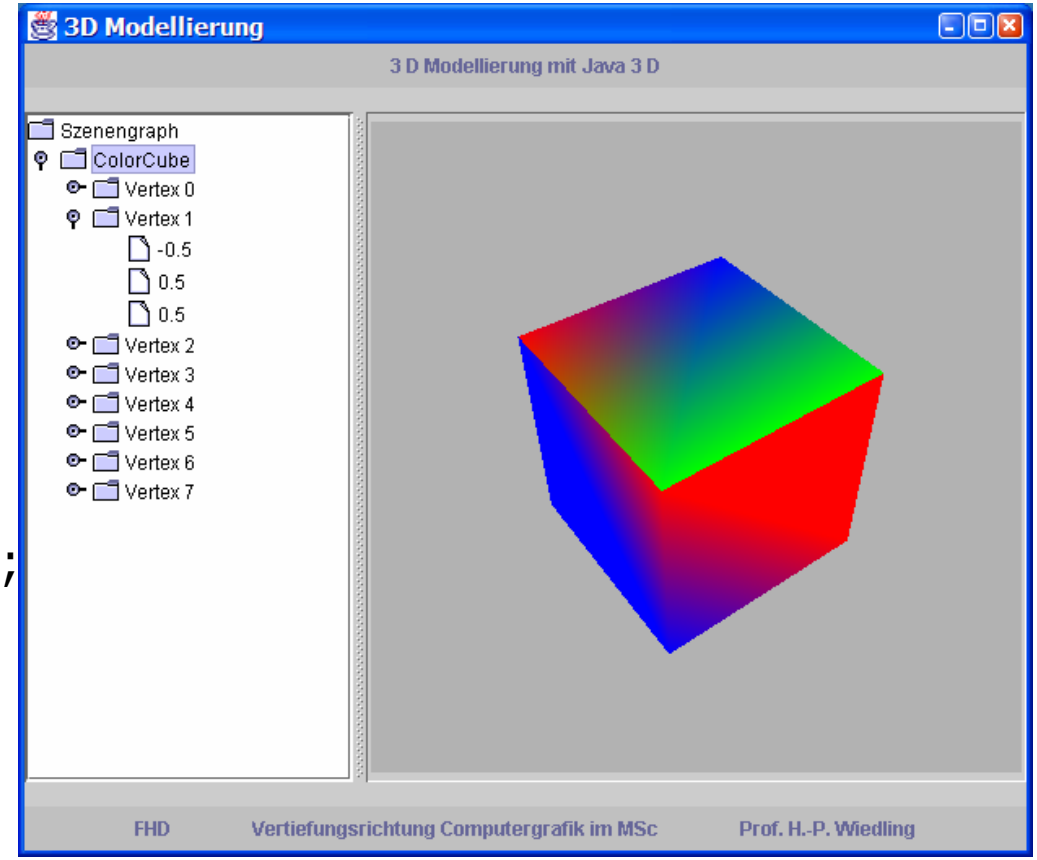
→ Definition von Punkten, Kanten, Flächen

→ Festlegung der Geometrie und der Farbzuordnung



Box / Cube (II)

```
...
RandomColorCube
cc = new
   RandomColorCube ();
Shape3D ashape =
   new Shape3D();

// Geometry and Color
ashape.setGeometry(cc);
// Transformationen
objTrans.addChild ( ashape);

// Szenegraph
bg.addChild(objTrans);
...
```



Box / Cube (III)

```
class RandomColorCube extends QuadArray
{
  RandomColorCube()
  {
    super
   (24, GeometryArray.COORDINATES |GeometryArray.COLOR_3);
    Point3f verts[] = new Point3f[8];
    Color3f colors[] = new Color3f[3];
    verts[0] = new Point3f( 0.5f, 0.5f, 0.5f);
    verts[1] = new Point3f(-0.5f, 0.5f, 0.5f);
    verts[2] = new Point3f(-0.5f,-0.5f, 0.5f);
    verts[3] = new Point3f( 0.5f,-0.5f, 0.5f);
    verts[4] = new Point3f( 0.5f, 0.5f,-0.5f);
    verts[5] = new Point3f(-0.5f, 0.5f,-0.5f);
    verts[6] = new Point3f(-0.5f,-0.5f,-0.5f);
    verts[7] = new Point3f( 0.5f,-0.5f,-0.5f);
```

Box / Cube (IV)

```
colors[0] = new Color3f(1.0f, 0.0f, 0.0f);
colors[1] = new Color3f(0.0f, 1.0f, 0.0f);
colors[2] = new Color3f(0.0f, 0.0f, 1.0f);
Point3f pnts[] = new Point3f[24];
Color3f clrs[] = new Color3f[24];
```

```
// 1. Seite
pnts[0]=verts[0];clrs[0]= colors[(int)(Math.random()*3.0)];
pnts[1]=verts[3];clrs[1]= colors[(int)(Math.random()*3.0)];
pnts[2]=verts[7];clrs[2]= colors[(int)(Math.random()*3.0)];
pnts[3]=verts[4];clrs[3]= colors[(int)(Math.random()*3.0)];
```

Box / Cube (V)

```
// 2. Seite
pnts[4]=verts[1];clrs[4]= colors[(int)(Math.random()*3.0)];
pnts[5]=verts[5];clrs[5]= colors[(int)(Math.random()*3.0)];
pnts[6]=verts[6];clrs[6]= colors[(int)(Math.random()*3.0)];
pnts[7]=verts[2];clrs[7]= colors[(int)(Math.random()*3.0)];
```

```
// 3. Seite
pnts[8]=verts[0];clrs[8]= colors[(int)(Math.random()*3.0)];
pnts[9]=verts[4];clrs[9]= colors[(int)(Math.random()*3.0)];
pnts[10]=verts[5];clrs[10]=colors[(int)(Math.random()*3.0)];
pnts[11]=verts[1];clrs[11]=colors[(int)(Math.random()*3.0)];
```

Box / Cube (VI)

```
// 4. Seite
pnts[12]=verts[3];clrs[12]= colors[(int)(Math.random()*3.0)];
pnts[13]=verts[2];clrs[13]= colors[(int)(Math.random()*3.0)];
pnts[14]=verts[6];clrs[14]= colors[(int)(Math.random()*3.0)];
pnts[15]=verts[7];clrs[15]= colors[(int)(Math.random()*3.0)];
```

```
// 5. Seite
pnts[16]=verts[0];clrs[16]= colors[(int)(Math.random()*3.0)];
pnts[17]=verts[1];clrs[17]= colors[(int)(Math.random()*3.0)];
pnts[18]=verts[2];clrs[18]= colors[(int)(Math.random()*3.0)];
pnts[19]=verts[3];clrs[19]= colors[(int)(Math.random()*3.0)];
```

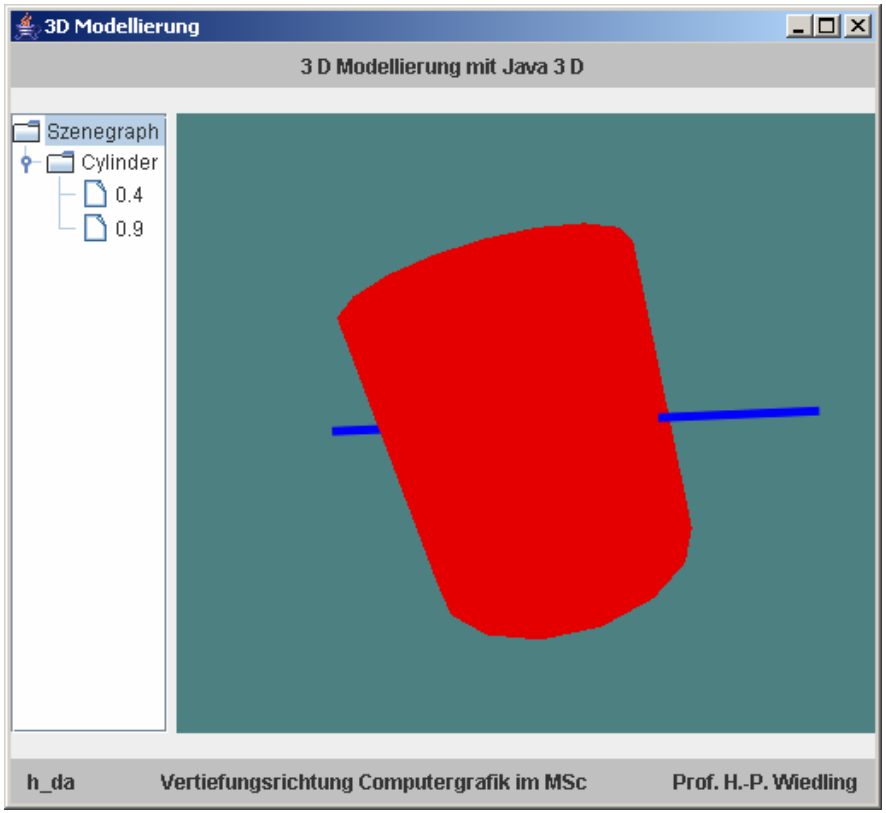
Box / Cube (VII)

```
 // 6. Seite
 pnts[20]=verts[7];clrs[20]=colors[(int)(Math.random()*3.0)];
 pnts[21]=verts[6];clrs[21]=colors[(int)(Math.random()*3.0)];
 pnts[22]=verts[5];clrs[22]=colors[(int)(Math.random()*3.0)];
 pnts[23]=verts[4];clrs[23]=colors[(int)(Math.random()*3.0)];

 setCoordinates(0, pnts);
 setColors(0, clrs);
  }
}
```

Cylinder

- Parameter:
  Grundfläche, Höhe

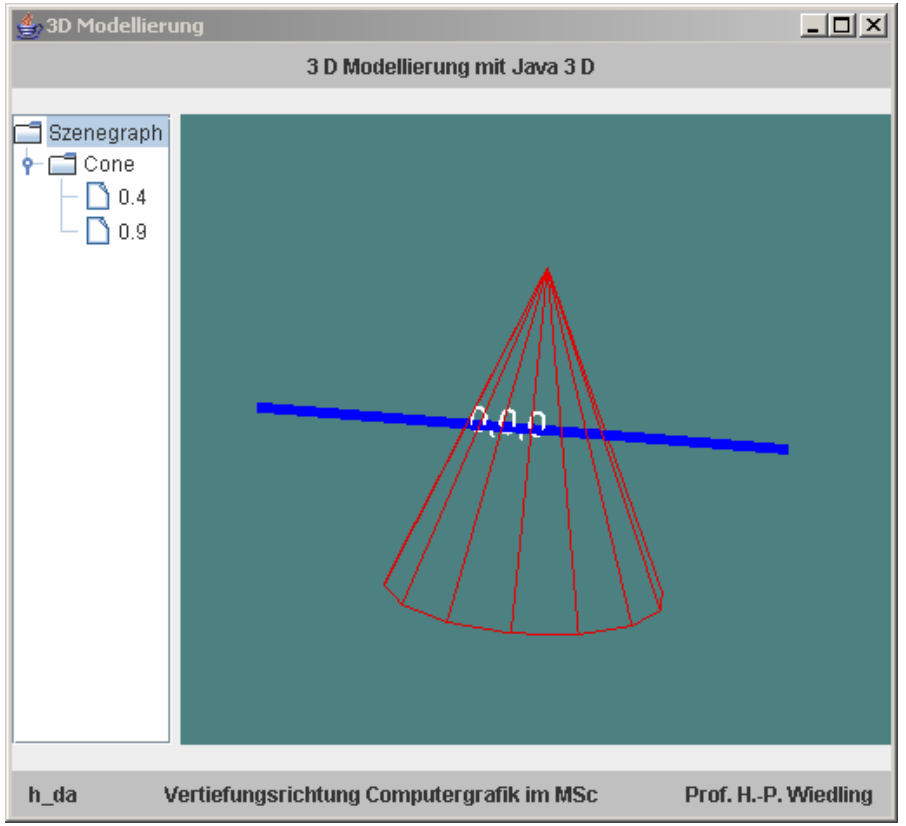
- **Cylinder**
  (float radius,float height)
  constructs a default cylinder of a given radius and height
- float **getHeight**()
  Returns the height of the cylinder
- float **getRadius**()
  returns the radius of the cylinder
- void **setAppearance**(Appearance ap)
  sets appearance of the cylinder



Cone

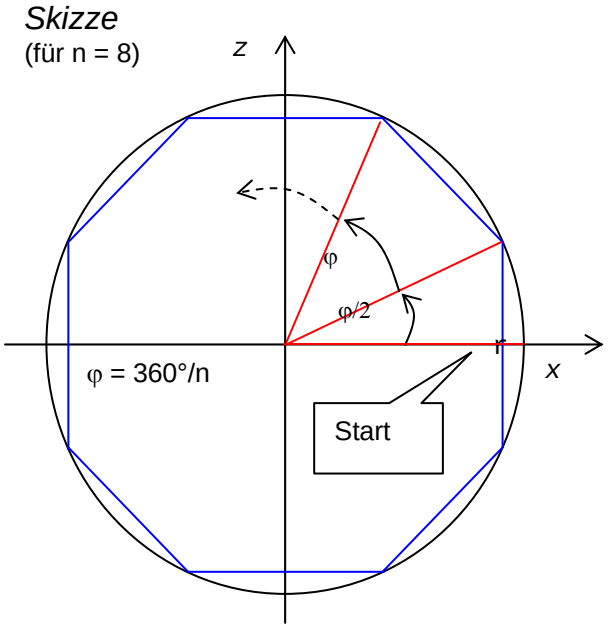
- Parameter:
  Grundfläche, Höhe

- **Cone**
  (float radius,float height)
  constructs a default cone
  of a given radius and height
- float **getHeight**()
  Returns the height
  of the cone
- float **getRadius**()
  returns the radius
  of the cone
- void
  setAppearance(Appearance ap)
  sets appearance of the cone



Sphere (I)

- auf gleichseitigen n – Ecken basierendes Raumprimitiv
- benötigt zur eindeutigen Bestimmung lediglich die Angabe von Mittelpunkt M, Radius r und Eckenzahl n (als Anzahl der die Kugel in horizontaler und vertikaler Richtung approximierenden Oberflächen);
  wobei n allerdings nicht beliebig gewählt werden kann, da für die korrekte Erzeugung der Eckpunkte die Teilbarkeit von n durch vier eine wesentliche Voraussetzung ist!



Sphere (II)

- $x(\theta, \phi) = \sin\theta \cos\phi$
  $z(\theta, \phi) = \cos\theta \cos\phi$
  $y(\theta, \phi) = \sin\phi$

```
c = PI / 180.0;
for (phi=80.0; phi<=80.0; phi-=20.0)
  {
   for (thet=-180; thet<=180;thet+=20)
     {
      // Pi
      x = sin (c*thet) * cos(c*phi);
      z = cos (c*thet)* cos(c*phi);
      y = sin(c*phi);
      // Pi+1
      x = sin (c*thet) * cos(c*phi+20);
      z = cos (c*thet)* cos(c*phi+20);
      y = sin(c*phi+20);
     }
}
```

Bemerkung:

- Die Formel entspricht der Transformation von Polar-koordinaten in kartesische Koordinaten
- Beschreibung als Formel
- Parametrisierung als Prinzip der Modellierung

Sphere (III)

- Die hier vorgestellte Methode zur rechnerinternen Modellierung kugelförmiger Objekte ergibt symmetrische Polyeder.
- Ähnlich wie bei den anderen Körpern wird zunächst ein regelmäßiges, grundsätzlich symmetrisches, n – Eck erzeugt.
- Durch den zusätzlichen Parameter r kann die Größe festgelegt werden.
- Mit Hilfe der eben erzeugten Punkte werden im folgenden Schritt die für die einzelnen Oberflächenelemente benötigten Eckpunkte berechnet.

Sphere (IV)

- Hierzu werden diese Punkte, oder wenn man so will, das durch sie aufgespannte und zuvor um einen gewissen Faktor s skalierte Polygon jeweils um einen bestimmten Betrag t in Richtung der y – Achse nach oben bzw. unten verschoben.

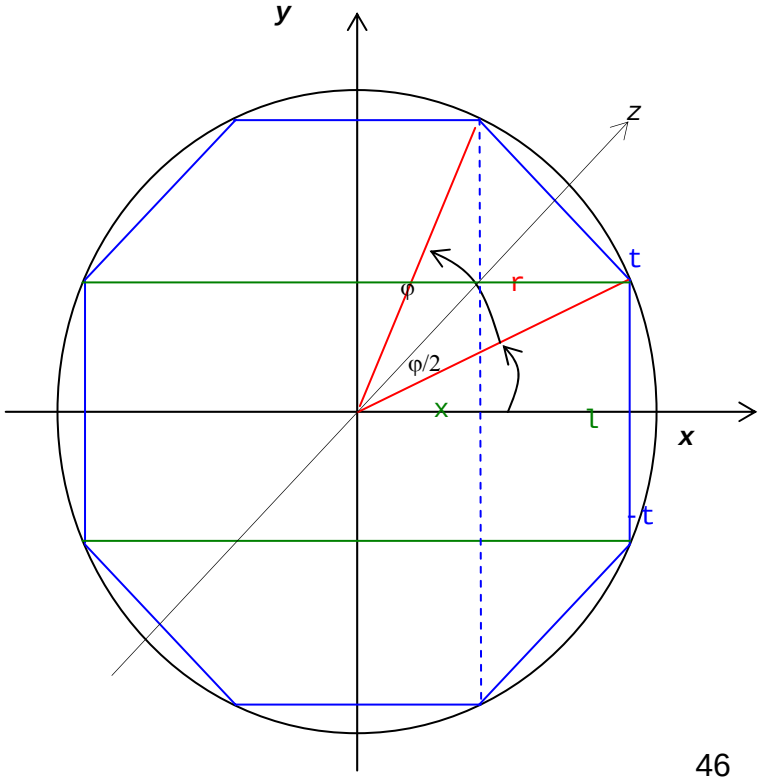
- Enthält ein solcher Polygon – „Ring“ stets n Punkte, die insgesamt n / 2 –mal verschoben werden müssen, folgt daraus, dass die Gesamtzahl E der Eckpunkte $n^2/2$ beträgt.

- Bemerkung:
  Ein Gütekriterium für den Wert n (bzw. für die Anzahl der Patches), kann etwa eine obere Grenze für den Winkel zwischen zwei benachbarten Patches herangezogen werden.

Sphere (V)

- Der Skalierungsfaktor s ergibt sich direkt aus dem Strahlensatz und berechnet sich wie folgt (s = 1 für i = 0):

$$s = \frac{x}{l} = \frac{r \cdot \cos\left(\frac{\varphi}{2} + i \cdot \varphi\right)}{r \cdot \cos\frac{\varphi}{2}} = \frac{\cos\left(\frac{\varphi}{2} + i \cdot \varphi\right)}{\cos\frac{\varphi}{2}}$$



Sphere (VI)

- Wie man sich anhand der Skizze auch für den allgemeinen Fall verdeutlichen kann, beträgt die Zahl der Flächen

$$F = (n/2 - 1) \cdot n + 2 = n^2 / 2 - n + 2$$

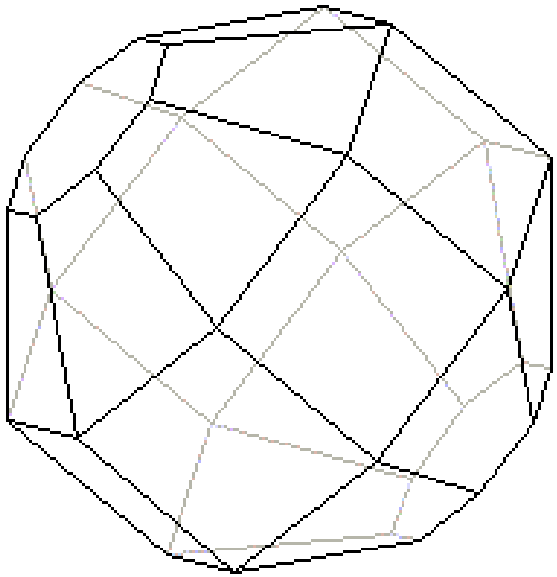
und die der Kanten $K = (n/2 + n/2 - 1) \cdot n = (n - 1) \cdot n = n^2 - n$

- Diese Werte genügen zudem der Eulerschen Formel für konvexe Polyeder, was zwar keinen vollständigen Beweis der Korrektheit darstellt, da es sich bei dieser Formel lediglich um eine notwendige Bedingung für das Vorliegen eines solchen Polyeders handelt, sie bietet aber dennoch eine einfache Möglichkeit der Verifikation

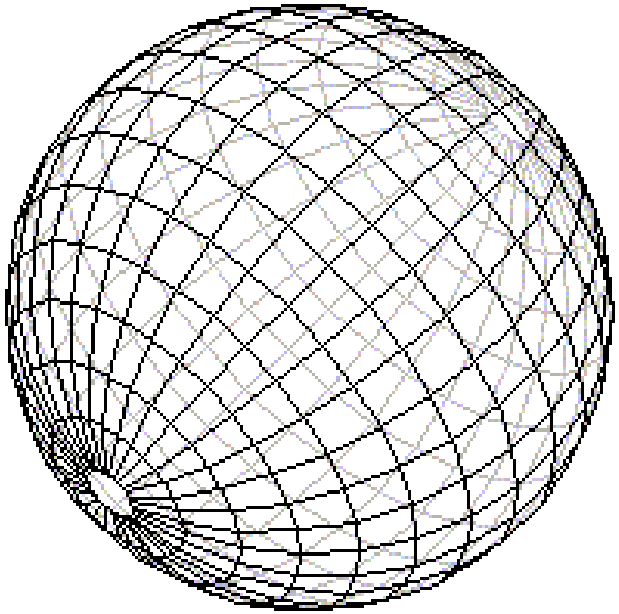
Sphere (VII)

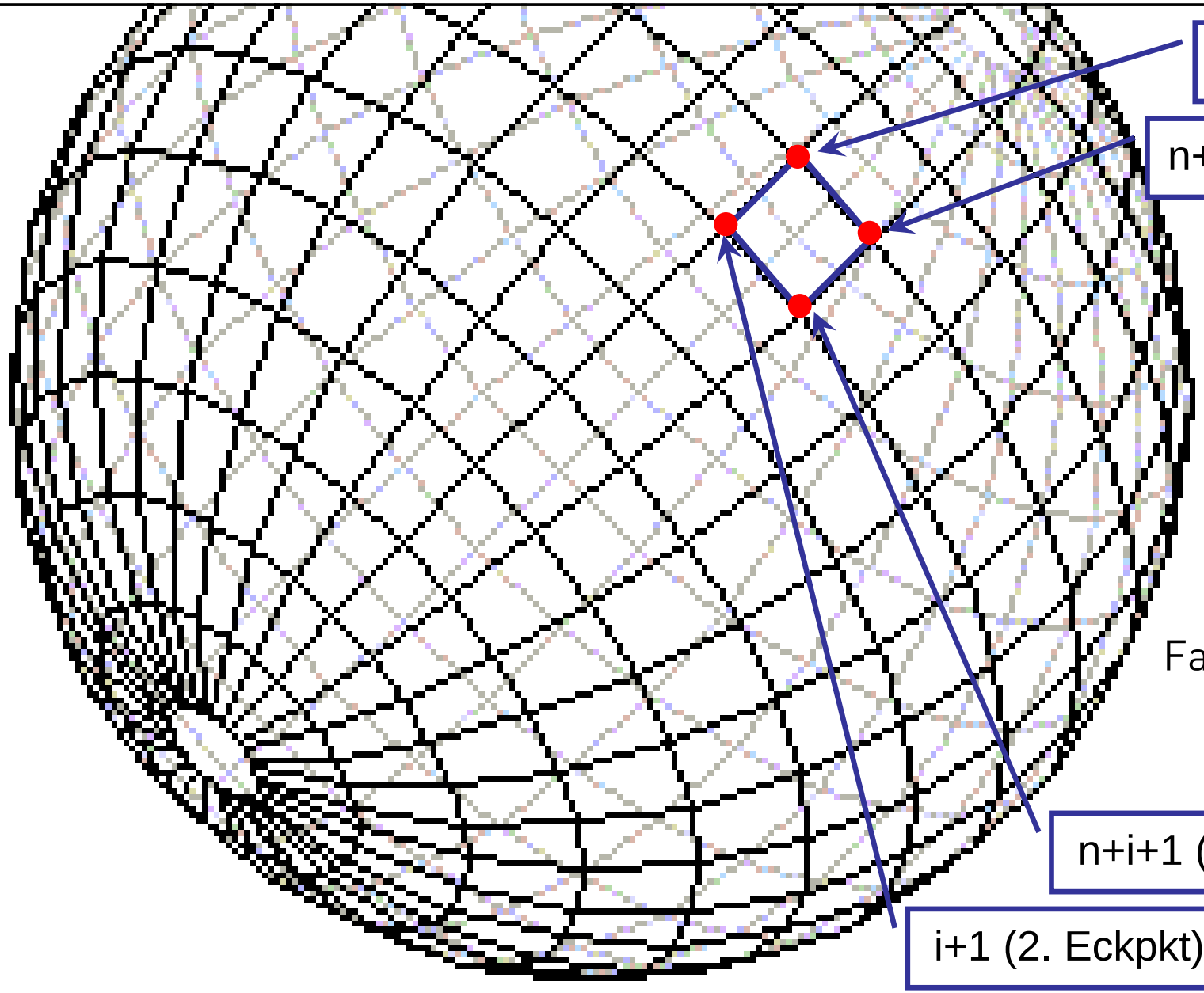
die Erhöhung der Punkte verbessert die Annäherung an eine Kugeloberfläche:

n = 8



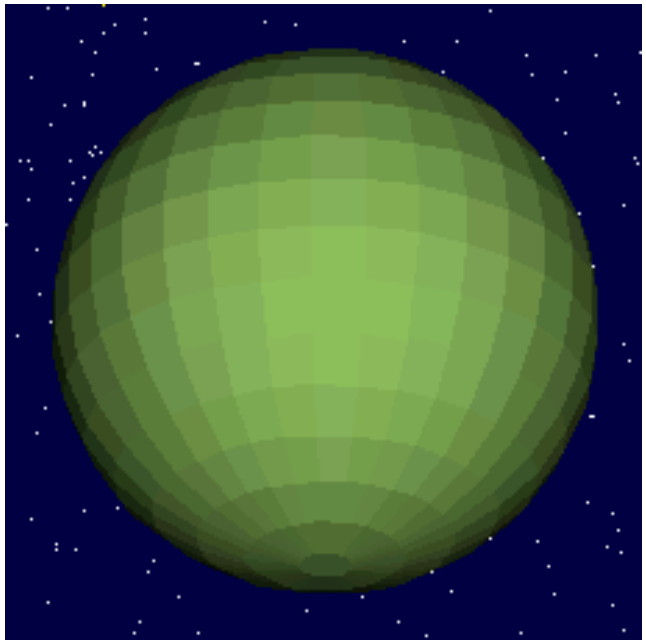
n = 32



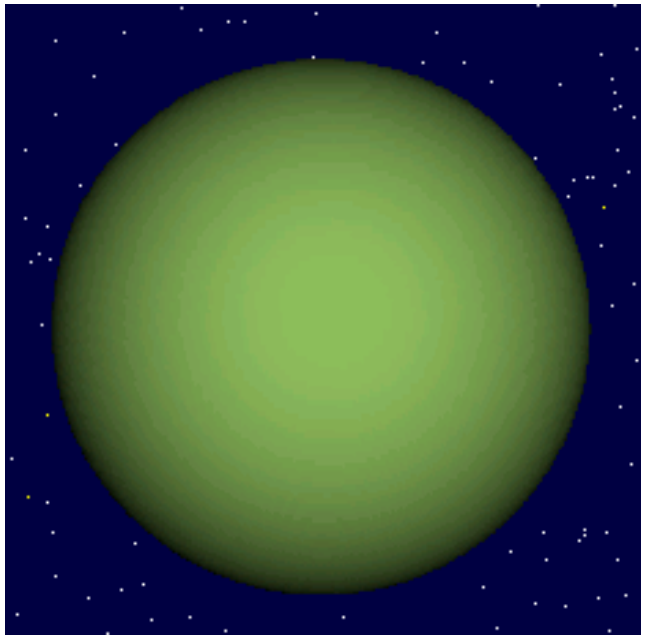

i (1. Eckpkt)
n+i (4. Eckpkt)
Facette, Patch
n+i+1 (3. Eckpkt)
i+1 (2. Eckpkt)

Visualisierung: konstante Schattierung (Flat Shading)

Kugel mit 482 Oberflächenelementen



Kugel mit mit 32514 Polygonen



Euler‘sche Polyederformel

Ecken + Flächen = Kanten + 2

Beispiel Box:

8 – 12 + 6 = 2

Beispiel Sphere:

$$E - K + F = \frac{n^2}{2} - (n^2 - n) + (\frac{n^2}{2} - n + 2) = \frac{n^2}{2} + \frac{n^2}{2} - n^2 + n - n + 2 = 2 \qquad \forall n \geq 4$$

Beispiele

Aufgabe:

Erstellen Sie eine Arbeitsumgebung in Java 3D.
Erzeugen Sie eine Markierung des Koordinatenursprungs.
Orientieren Sie sich dabei stark an dem Beispiel im Java 3D Tutorial und ändern Sie Parameter bzgl. Geometrie und Appearance ab.

- Hinweis zu Java3D:
  Der (Speicher-)Heap der Java VM ist per Default begrenzt.
  Startet man nun eine Applikation mit den VM Argumenten "Xms" (minimaler Heap) und "Xmx„ (maximaler Heap), so kann man die Begrenzung umgehen bzw. erhöhen.
  Beispiel:
  `java MeineKugel -Xms512m -Xmx512`
  dies reserviert 512 MB Heapspeicher

Pyramide, Prisma, Torus, ...

 Aufgabe:

a) Erstellen Sie eine Sphere als Wire Frame und als „Flickenteppich“ (solid surface)

a) Konstruieren Sie eine Pyramide mit einem n-Eck als Grundfläche und bauen Sie ein Objekt in unser Demoprogramm ein

b) Wie würden Sie entsprechend ein Prisma, ein Parallelepiped (Rechteck oder Parallelogramm - die Raumdiagonalen schneiden sich in einem Punkt, mit zusätzlicher Ausdehnung in z) und einen Torus erzeugen?

c) Testen Sie die Eulersche Formel mit Ihren Objektbeschreibungen

Zusammengesetzte Objekte

Aufgabe:

Konstruieren Sie ein (einfaches) zusammengesetztes Objekt und bauen Sie eine Instanz in Ihr Demoprogramm ein

Interaktion: Rotation ...

```
//import com.sun.j3d.utils.behaviors.mouse.*;
Transform3D t = new Transform3D();
TransformGroup tg = new TransformGroup(t);
tg.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
objRoot.addChild(tg);
MouseRotate myMouseRotate = new MouseRotate();
myMouseRotate.setTransformGroup(tg);
myMouseRotate.setSchedulingBounds(new BoundingSphere());
objRoot.addChild(myMouseRotate);
// scene erzeugen
scene = ...
tg.addChild(scene);
...
```

✍ Aufgabe: Wie sieht der entsprechende SceneGraph aus?

Formate und Systeme: Java 3D, ...

- API zur Beschreibung der
  - Geometrie,
  - Topologie und
  - Transformationen
- zusätzlich wird die Erscheinung (Appearance) von Objekten festgelegt:
  - Farbe, Linienattribute (type, width), Flächenattribute (fillmode), Textur, ...

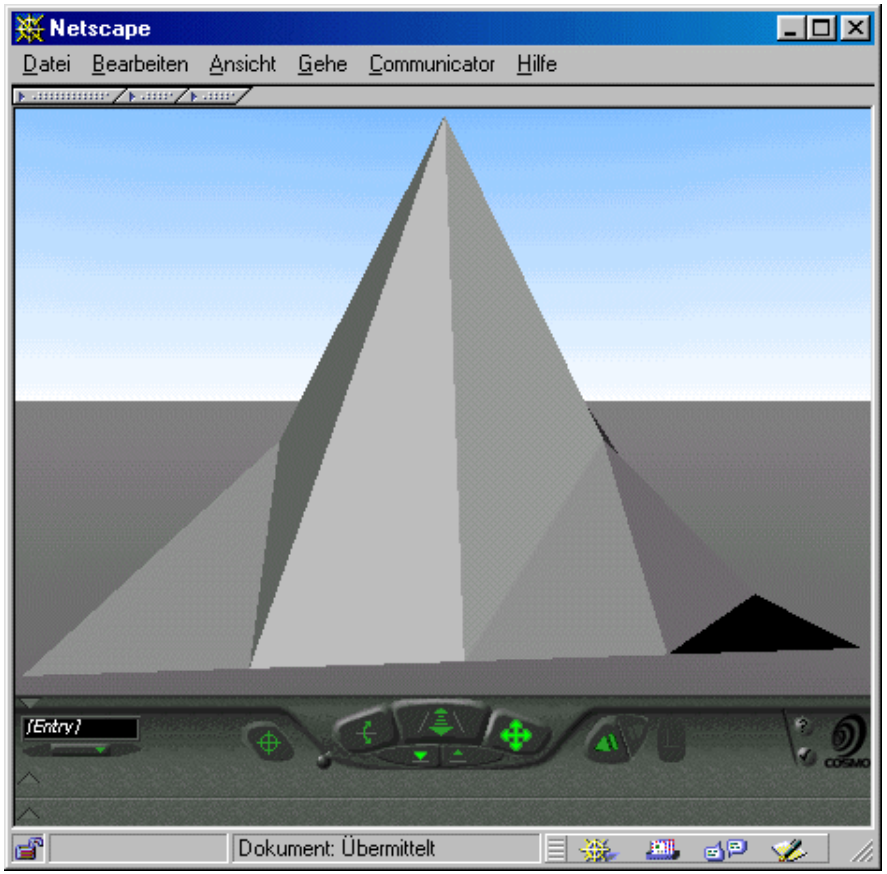
➔ Auseinandersetzung mit entsprechender Klassenbibliothek:

- ➔ Objekte und Attribute: Datenstrukturen
- ➔ Methoden
- ➔ Hierarchie

VRML – Virtual Reality Modelling Languages

- 3D, textuelle Beschreibung eines Szenegraphen
- Skriptsprache
- XML-Format, HTML-Erweiterung
- Beispiel:

```
Shape
{ appearance Appearance
    { material Material {} }
geometry ElevationGrid
{ xDimension 5
  zDimension 5
  xSpacing 1.0 zSpacing 1.0
  height
[ 0.0 0.0 0.0 0.0 0.0,
  0.0 1.0 1.5 1.0 0.0,
  0.0 0.5 2.5 1.5 0.0,
  0.0 1.0 3.0 1.0 0.0,
  0.0 0.0 0.0 0.0 0.0 ] } }
```



VRML bzw. X3D

- Shapes describe the geometrical primitives of a VRML Scene
- Shapes consist of
  - appearance - describes color and texture and
  - geometry - describes the form
- Grouping allows the affection of several
  - objects by one transformation
  - make a complex scene manageable
  - structuring of a whole scene in a hierarchy
  - transformations and animations in hierarchical system

Open GL, Direct 3D

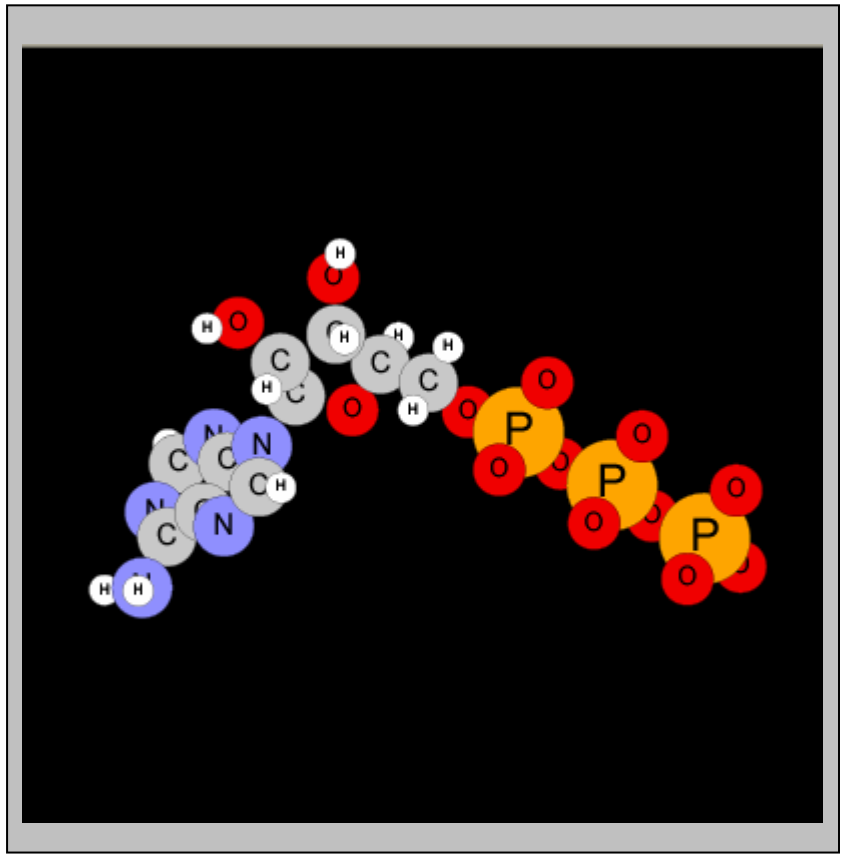
- objektorientiertes API
- hinter Direct 3D steht Microsoft: de facto Standard (http://www.microsoft.com/directx)
- hinter Open GL steht ein Firmenkonsortium → Standardisierungsbestrebungen (http://www.opengl.org)

SVG – Scalable Vector Graphics

- XML-Format, direkt in HTML nutzbar, 2D
- Beispiel:

```
<svg viewBox="-9.187 -6.32
19.406 12.1640005"> <rect
x="-10000" y="-10000"
width="20000" height="20000"
fill="black"/> <circle cx="-
5.649" cy="0.29" r="0.37"
fill="rgb(255,255,255)"
stroke="black" stroke-
width="0.01"/> <text x="-
5.649" y="0.3825"
style="fill:black;stroke:non
e;text-anchor:middle;font-
family:Arial;font-
size:0.37">H</text>
...
```

- Geometry und Appearance als Attribute
- Topologie ist Strukturinformation

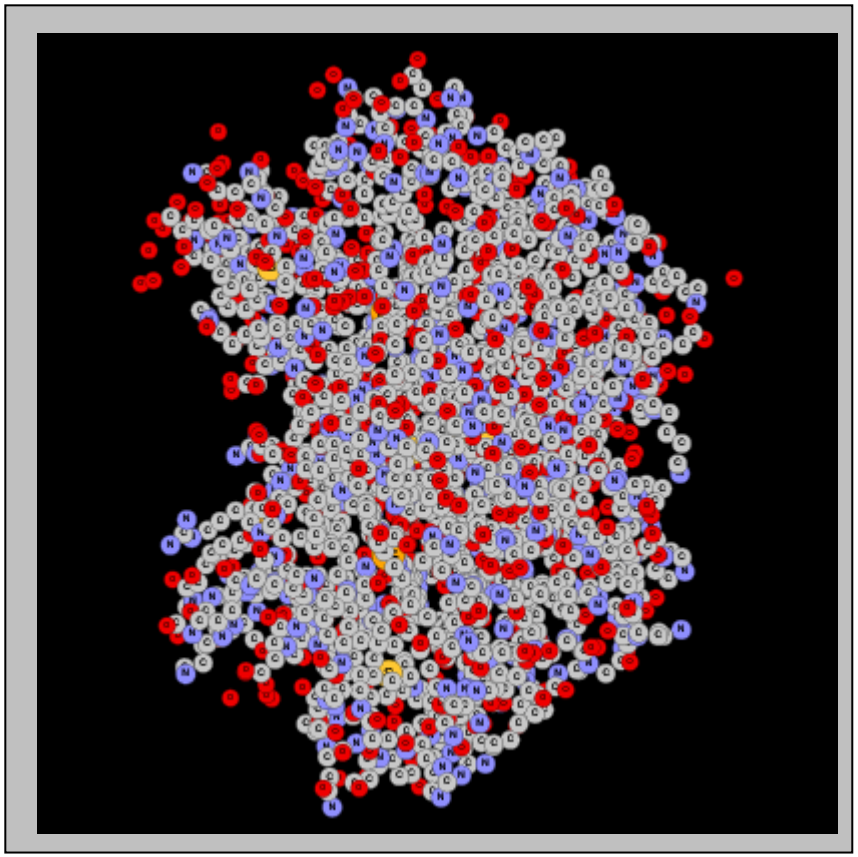


SVG – Scalable Vector Graphics

- Unmittelbar in HTML nutzbar
- Vektorformat
- Mit JavaScript kombinierbar:

```
<svg viewBox="0 0 60 60">
<script type="text/javascript">
<![CDATA[
function click_blue() {
zeit = new Date();
text = zeit.getHours() + ":" +
zeit.getMinutes();
alert("Uhrzeit: " + text); } ]]>
</script>
<rect x="5" y="5" width="20"
height="20" fill="blue"
onclick="click_blue();"/></svg>
```

- Beispiel aus der Bioinformatik Hämoglobin



konkret: Java 3D

- streng objektorientiert
- Szenegraph basierend
- → java 3D Tutorial:
- In 3D computer graphics, everything from the simplest triangle to the most complicated jumbo jet model is modeled and rendered with vertex-based data.
  - With Java 3D, each Shape3D object should call its method setGeometry() to reference one and only one Geometry object.
  - To be more precise, Geometry is an abstract superclass, so the referenced object is an instance of a subclass of Geometry.

  - Transformationen sind Knoten in einer Szene
  - Die entsprechenden Objekte (Shape3D) sind Kindknoten der entsprechenden Transformationen

Shape3D

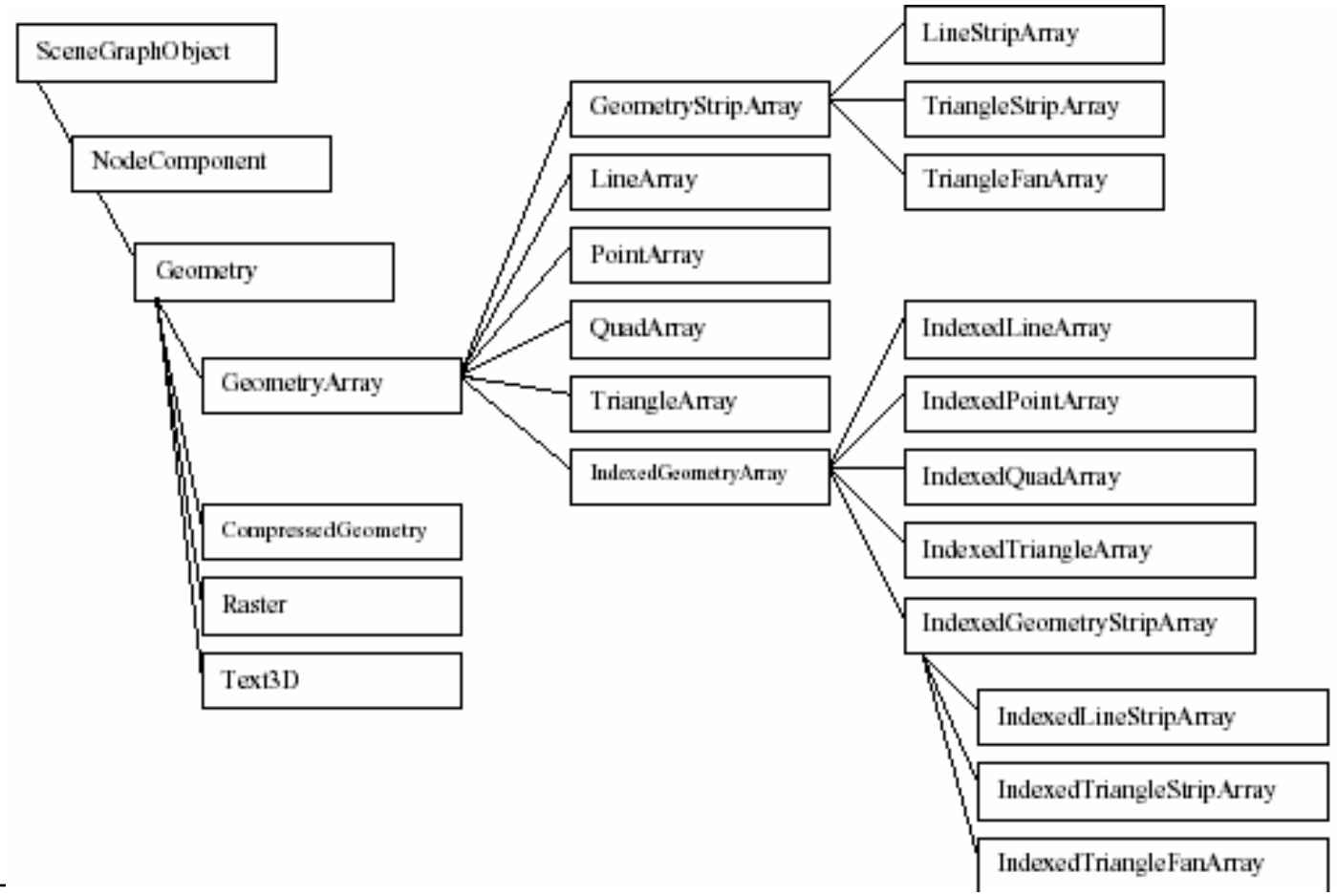
- A shape leaf node consisting of geometery and appearance properties.

- void setAppearance (Appearance appearance)
  Sets the appearance component of this Shape3D node.
- Appearance getAppearance ()
  Retrieves the appearance component of this shape node.
- void setGeometry (Geometry geometry)
  Sets the geometry component of the Shape3D node.
- Geometry getGeometry ()
  Retrieves the geometry component of this Shape3D node.
- boolean intersect (SceneGraphPath path, PickRay pickRay, double[] dist)
  check if the geometry component of this shape node under path intersects with the pickShape.
- boolean intersect (SceneGraphPath path, PickShape pickShape)
  Check if the geometry component of this shape node under path intersects with the pickShape.
- void setCollisionBounds (Bounds bounds)
  Sets the collision bounds of a node.
- Bounds getCollisionBounds ()
  Returns the collision bounding object of this node.

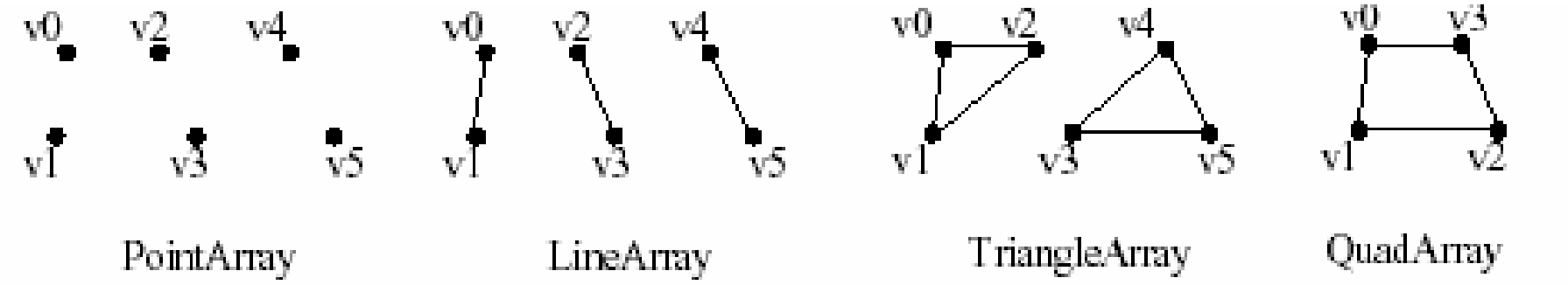
Geometry Class Hierarchy

- As you may deduce from the class names, the Geometry subclasses may be used to specify points, lines, and filled polygons (triangles and quadrilaterals). These vertex-based primitives are subclasses of the GeometryArray abstract class, which indicates that each has arrays that maintain data per vertex.
- For example, if a GeometryArray object is used to specify one triangle, a three-element array is defined:
  - one element for each vertex. Each element of this array maintains the coordinate location for its vertex (which can be defined with a Point* object or similar data).
  - In addition to the coordinate location, three more arrays may be optionally defined to store
    - the color for each vertex (color array),
    - the surface normal (surface normals), and
    - texture coordinate data.

Geometry Class Hierarchy

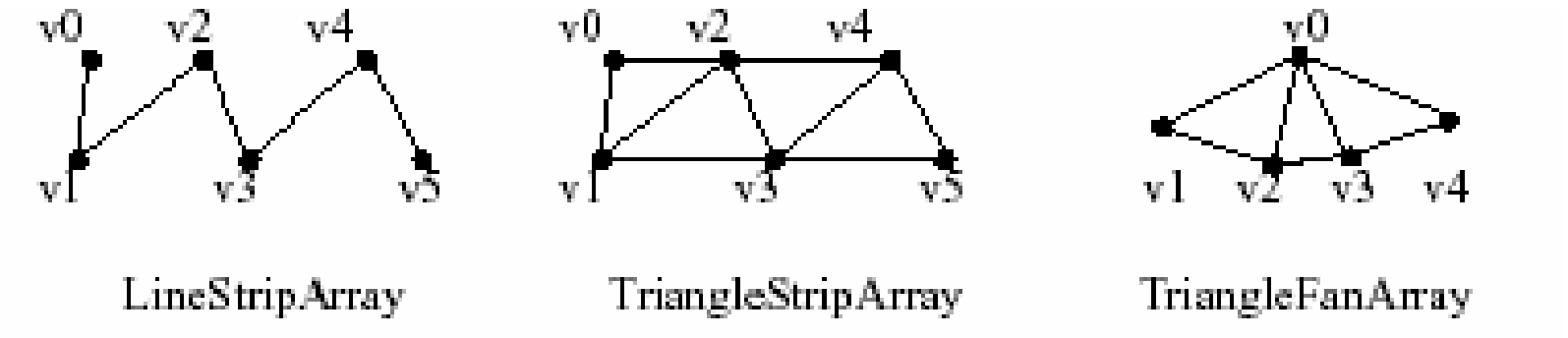


Geometry Array Subclasses



- GeometryArray Subclass Constructors:
  Constructs an empty object with the specified number of vertices and the vertex format. The format is one or more individual flags bitwise "OR"ed together to describe the per-vertex data. The format flags are the same as defined in the GeometryArray superclass.
- `PointArray(int vertexCount, int vertexFormat)`
- `LineArray(int vertexCount, int vertexFormat)`
- `TriangleArray(int vertexCount, int vertexFormat)`
- `QuadArray(int vertexCount, int vertexFormat)`

Subclasses of GeometryStripArray



- Note that Java 3D does not support filled primitives with more than four sides. The programmer is responsible for using tessellators to break down more complex polygons into Java 3D objects, such as triangle strips or fans. The Triangulator utility class converts complex polygons into triangles.
- In the TriangleStrip, for example, each vertex is combined with the previous two vertices to define a new triangle, starting with vertex v2

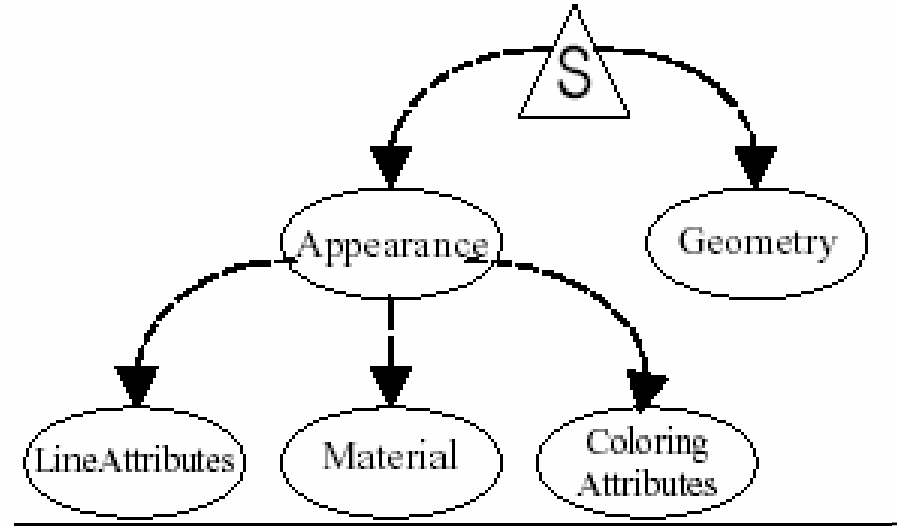
Subclasses of GeometryStripArray

- GeometryStripArray Subclass Constructors
- Constructs an empty object with the specified number of vertices, the vertex format, and an array of vertex counts per strip. The format is one or more individual flags bitwise "OR"ed together to describe the pervertex data. The format flags are the same as defined in the GeometryArray superclass. Multiple strips are supported. The sum of the vertex counts for all strips (from the stripVertexCounts array) must equal the total count of all vertices (vtxCount).
- **`LineStripArray(int vtxCount, int vertexFormat, int stripVertexCounts[])`**
- **`TriangleStripArray(int vtxCount, int vertexFormat, int stripVertexCounts[]))`**
- **`TriangleFanArray(int vtxCount, int vertexFormat, int stripVertexCounts[]))`**

Appearance

An Appearance object can refer to several different NodeComponent subclasses called appearance attribute objects, including:

- PointAttributes
- LineAttributes
- PolygonAttributes
- ColoringAttributes
- TransparencyAttributes
- RenderingAttributes
- Material
- TextureAttributes
- Texture
- TexCoordGeneration



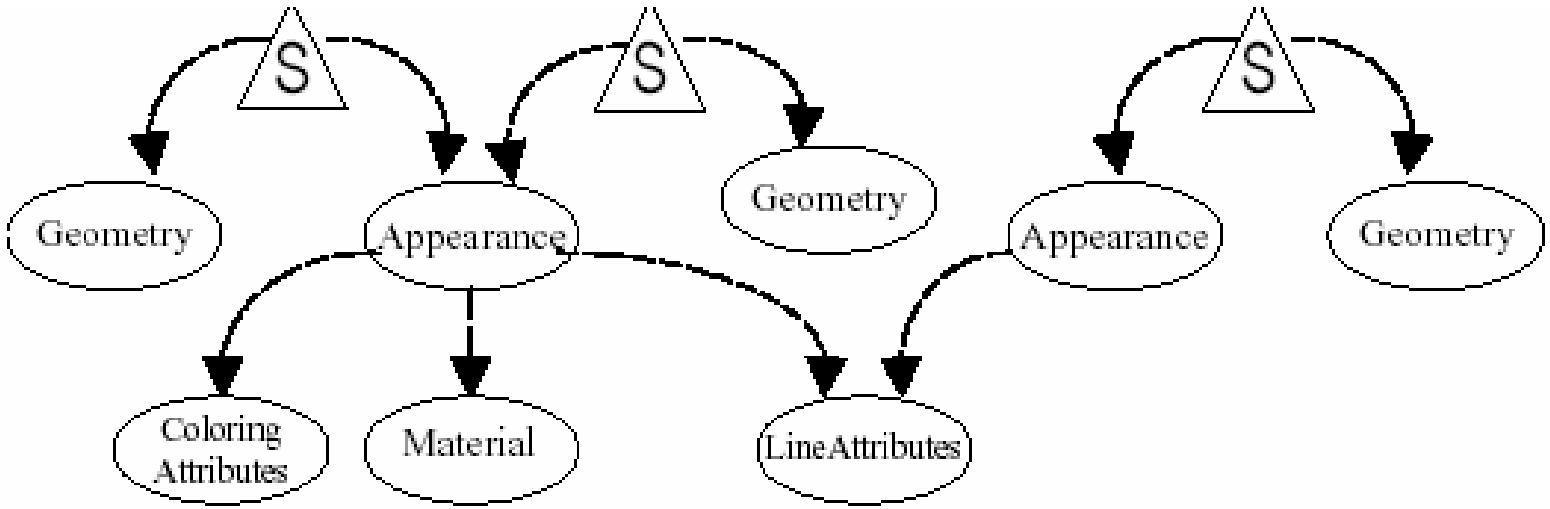
Appearance Methods (excluding lighting and texturing)

- Appearance () Konstruktor
- Each method sets its corresponding NodeComponent object to be part of the current Appearance bundle:

```
- void setPointAttributes
      (PointAttributes pointAttributes)
- void setLineAttributes (LineAttributes lineAttributes)
- void setPolygonAttributes
      (PolygonAttributes polygonAttributes)
- Void setColoringAttributes
      (ColoringAttributes coloringAttributes)
- void setTransparencyAttributes
      (TransparencyAttributes transparencyAttributes)
- void setRenderingAttributes
      (RenderingAttributes renderingAttributes)
```

Appearance-Knoten im Szenegraphen

- Folgende Hierarchie ist erlaubt (und auch sinnvoll):



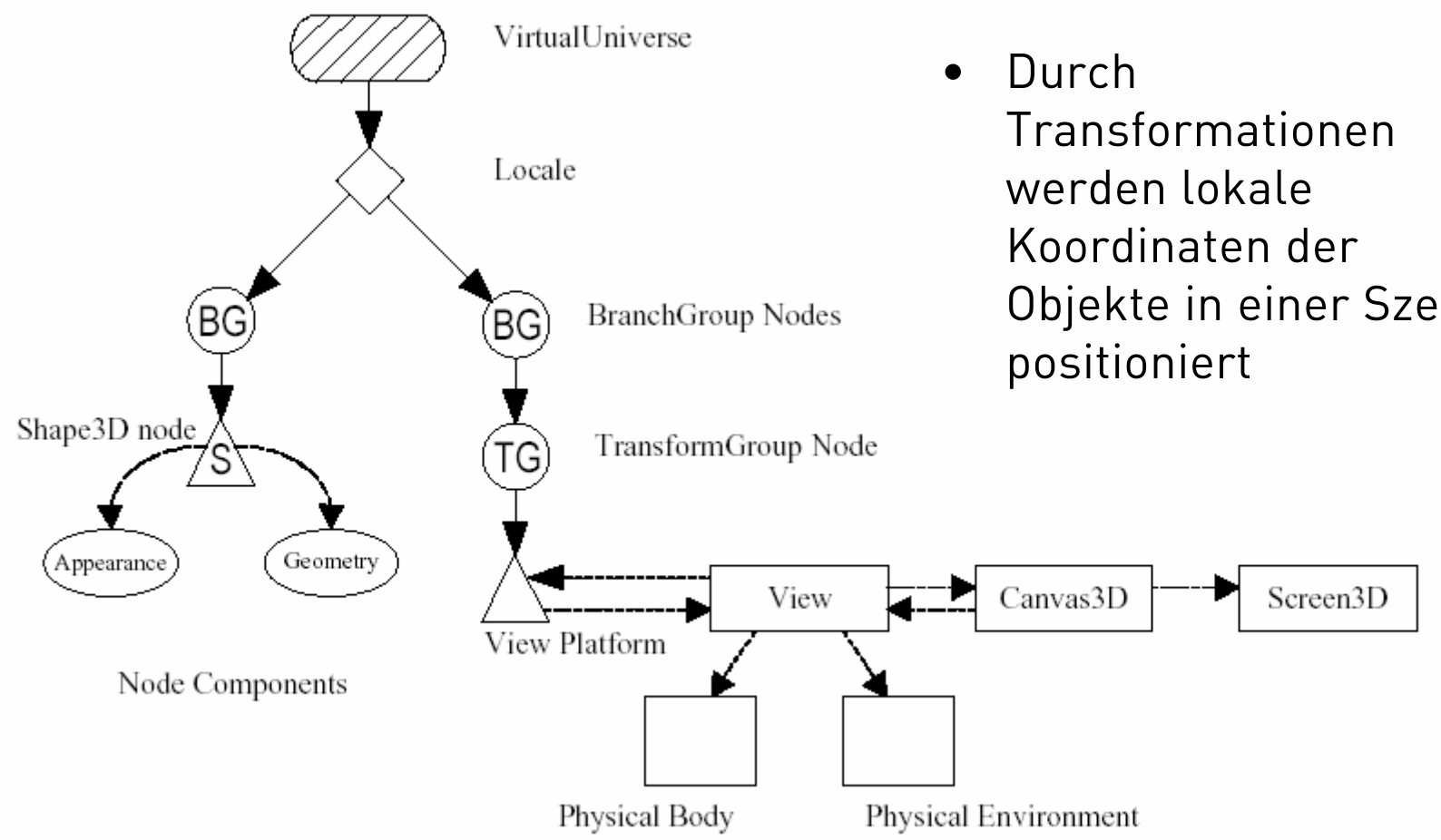
Transform3D Methods (partial list)

- Transform3D objects represent geometric transformations
  Transform3D is one of the few classes not directly used in any scene graph. The transformations represented by a Transform3D object are used to create TransformGroup objects that are used in scene graphs:
  - **void rotX(double angle)**
    Sets the value of this transform to a counter clockwise rotation about the x-axis. The angle is specified in radians.
  - **void rotY(double angle)**
    Sets the value of this transform to a counter clockwise rotation about the y-axis. The angle is specified in radians.
  - **void rotZ(double angle)**
    Sets the value of this transform to a counter clockwise rotation about the z-axis. The angle is specified in radians.

Transform3D Methods (partial list)

- void set (Vector3f translate)
  Sets the translational value of this matrix to the Vector3f parameter values, and sets the other components of the matrix as if this transform were an identity matrix.
- void setScale (double scale)
  Sets the scale component of the current transform; any existing scale is first factored out of the existing transform before the new scale is applied.
- void setScale (Vector3d scale)
  Sets the possibly non-uniform scale component of the current transform; any existing scale is first factored out of the existing transform before the new scale is applied.

Szenegraphen



- Durch Transformationen werden lokale Koordinaten der Objekte in einer Szene positioniert

Szenegraphen

- hierarchische Anordnung von Transformationen
  (TG - TransformGroup-Knoten und
  Objekten/Shape3D
  (S - Shape3D-Knoten)
- Erzeugen/Verwaltung der Objekte in einer Szene ergibt sich aufgrund der Topologie
- einfache Kombination von lokalen und globalen Koordinaten

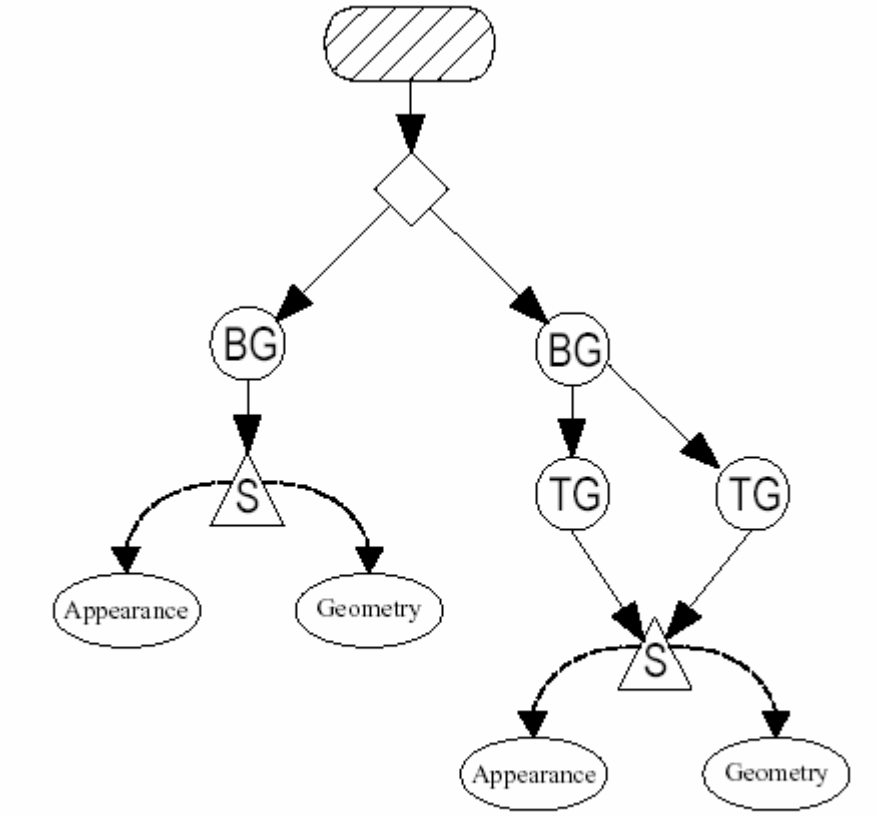


Figure 1-3 Example of an Illegal Scene Graph

verbesserter Szenegraph

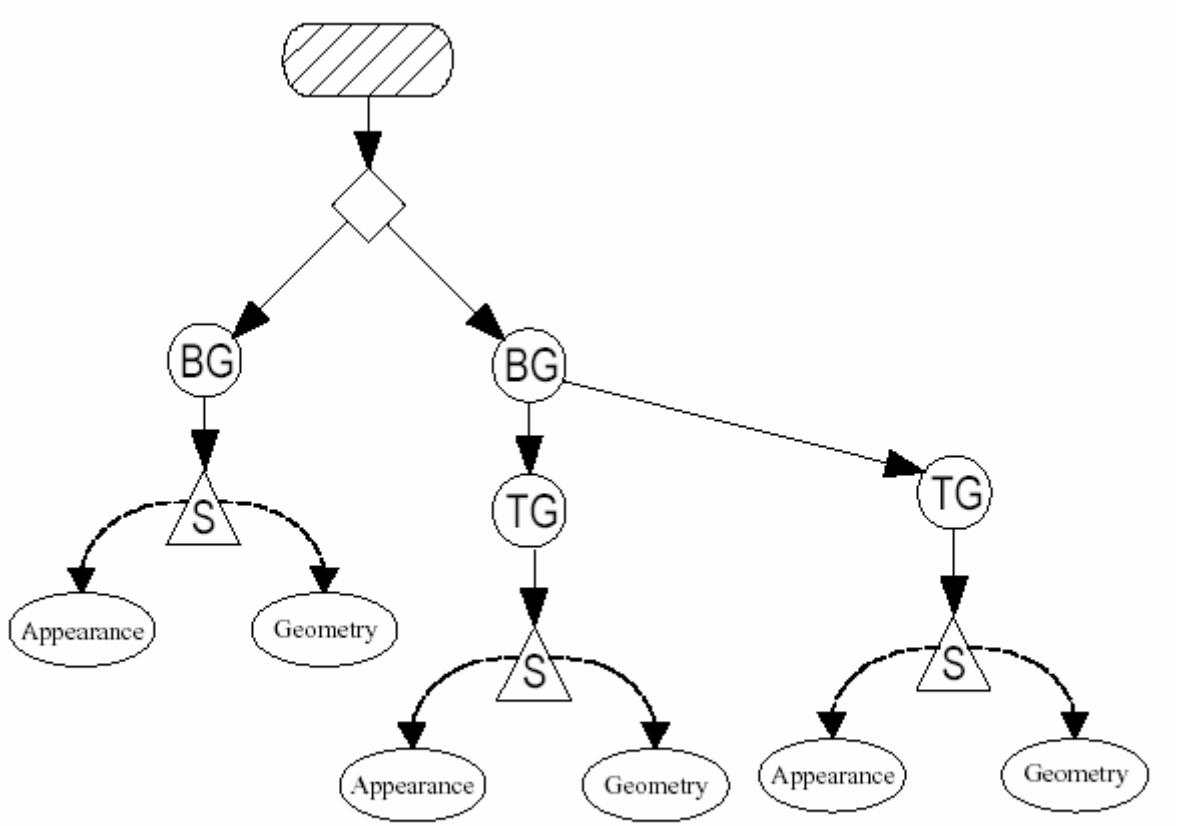


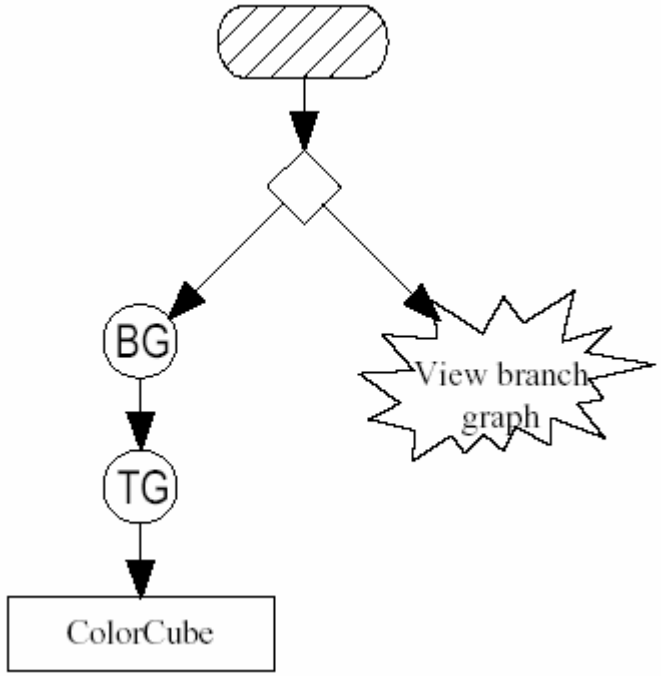
Figure 1-4 One Possible Fix for Illegal Scene Graph of Figure 1-3

... createSceneGraph ...

```
 ...
 public BranchGroup createSceneGraph() {
  // Create the root of the branch graph
  BranchGroup objRoot = new BranchGroup();

  // Create a simple shape leaf node, add it to the scene
  // graph. ColorCube is a Convenience Utility class
  objRoot.addChild(new ColorCube(0.4));

  return objRoot;
 } // end of createSceneGraph method (application related)
 ...
}
```

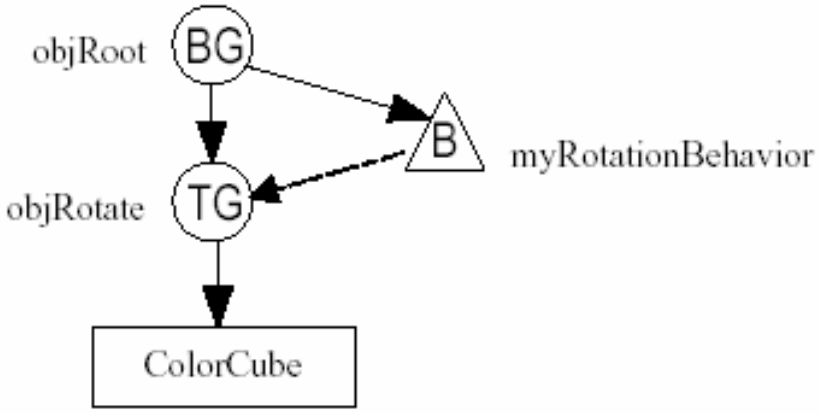
... Transform 3D ...

```
public BranchGroup createSceneGraph() {
// Create the root of the branch graph
BranchGroup objRoot = new BranchGroup();

// rotate object has composite
// transformation matrix
Transform3D rotate = new Transform3D();
Transform3D tempRotate = new Transform3D()

rotate.rotX(Math.PI/4.0d);
tempRotate.rotY(Math.PI/5.0d);
rotate.mul(tempRotate);
TransformGroup objRotate =
      new TransformGroup(rotate);

objRotate.addChild(new ColorCube(0.4));
objRoot.addChild(objRotate);
return objRoot;
```



... Behavior ...

```
// Transformation festlegen und in Szenegraph einbauen
Transform3D t = new Transform3D();
TransformGroup tg = new TransformGroup(t);
// Neuschreiben der Transformation erlauben
tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

// Verhalten definieren
SimpleBehavior sb =
   new SimpleBehavior(tg);
sb.setSchedulingBounds
   ( new BoundingSphere());
objRoot.addChild(sb);
```



SimpleBehavior (I)

```
import java.awt.*;
import java.awt.event.*;
import javax.media.j3d.*;
import java.util.*;

class SimpleBehavior extends Behavior{

private TransformGroup targetTG;
private Transform3D rotation = new Transform3D();
private double angle = 0.0;
// create SimpleBehavior
SimpleBehavior(TransformGroup targetTG){
     this.targetTG = targetTG;
     System.out.println ("constructor");
   }
```

SimpleBehavior (II)

```
// initialize the Behavior
// set initial wakeup condition; called when behavior becomes
   live
public void initialize(){
// set initial wakeup condition
   this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
  }
// behave: called by Java 3D when appropriate stimulus occures
public void processStimulus(Enumeration criteria){
// decode event; do what is necessary
   angle += 0.1;
   rotation.rotY(angle);
   targetTG.setTransform(rotation);
   this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
  }
} // end of class SimpleBehavior
```

... setCapability (...) ...

- auf entsprechende Art kann nicht nur auf Ereignisse reagiert werden sondern auch der SceneGraph verändert werden:

```
tg.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
tg.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
tg.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND); // Anfügen
...
BranchGroup objRoot = new BranchGroup();
objRoot.addChild(new ColorCube (0.15));
// darf aus parent Knoten entfernt werden
objRoot.setCapability(BranchGroup.ALLOW_DETACH);
...
if (targetTG.numChildren()==1) targetTG.addChild( objRoot );
else                           targetTG.removeChild(1);
...
```

... Background ...

```
// Colored Background
Background backg = new Background(0.5f, 1.0f,1.0f);
backg.setApplicationBounds(new BoundingSphere());

// create BranchGroup as Background
// backg.setGeometry (createBackGraph());

// backg.setImage (ImageComponent2D image);

// Create the root of the branch graph
BranchGroup objRoot = new BranchGroup();
objRoot.addChild(backg);
```

... Background Image ...

```
Background backg = new Background(0.5f, 1.0f,1.0f);
backg.setApplicationBounds(new BoundingSphere());
Image bild;
bild = Toolkit.getDefaultToolkit().getImage (
  "C:/darmstadt/vorlesungen/Modellierung/marker/.../teil4.gif");
MediaTracker tracker = new MediaTracker (this);
tracker.addImage (bild, 0);
try {tracker.waitForID(0);}
catch (Exception e){System.out.println("Fehler (MediaTracker): "+e);}
BufferedImage img=new BufferedImage(500,385,BufferedImage.TYPE_INT_RGB);
Graphics2D g = img.createGraphics();
g.drawImage( bild, 0,0, this);
image = new ImageComponent2D (ImageComponent2D.FORMAT_RGB, img);
backg.setImage (image);
objRoot.addChild(backg);
```

Light(I)

```
...
// Shape3D mit Flächennormalen erzeugen
new quadArray(n, GeometryArray.COORDINATES |
   GeometryArray.COLOR_3 | GeometryArray.NORMALS );
// Flächennormale setzen
this.setNormals(j*4, normals);
// Materialeigenschaften verwenden
 Material material = new Material();
  material.setDiffuseColor(new Color3f(.8f, .1f, .1f));
  material.setAmbientColor(new Color3f(.1f, .1f, .8f));
  material.setEmissiveColor(new Color3f(0.f, .5f, 0.5f));
 appear.setMaterial(material);
...
```

Light(II)

```
...
// Lichtobjekt(e) erzeugen
 SpotLight spotlight = new SpotLight();
 spotlight.setEnable(true);    //light on
 spotlight.setColor(new Color3f(1.f, 1.f, 1.f));
 spotlight.setPosition(0.f, 0.f, 50.f);
 spotlight.setDirection(.0f, .0f, -1f);
 spotlight.setSpreadAngle((float)(Math.PI/100));
// Dämpfungsparamter siehe Tutorial Seite 6-14
 spotlight.setAttenuation(1f, 0f, 0f);
 spotlight.setConcentration(0.9f);
 spotlight.setInfluencingBounds(new BoundingSphere());

 objScene.addChild(spotlight);
```

Ein Kochrezept

1. Create a Canvas3D object
2. Create a VirtualUniverse object
3. Create a Locale object, attaching it to the VirtualUniverse object
4. Construct a view branch graph
   a) Create a View object
   b) Create a ViewPlatform object
   c) Create a PhysicalBody object
   d) Create a PhysicalEnvironment object
   e) Attach ViewPlatform, PhysicalBody, PhysicalEnvironment, and Canvas3D objects to View object
5. **Construct content branch graph(s): `createSceneGraph()`**
6. Compile branch graph(s)
7. Insert subgraphs into the Locale