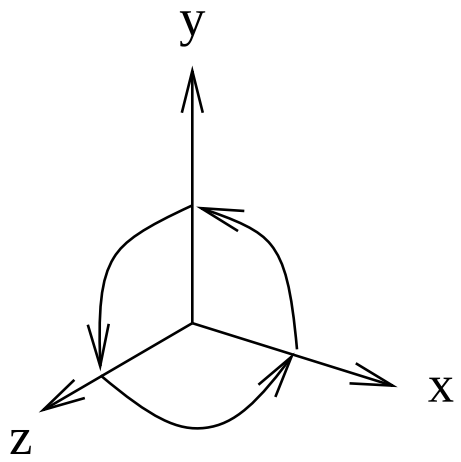
Koordinatensystemorientierung

rechtshändiges Koordinatensystem: eine $90°$-Drehung gegen den Uhrzeigersinn überführt die positiven Achsen ineinander.



Verwendet man den Daumen der rechten Hand als x-Achse, den Zeigefinger als y-Achse und den Mittelfinger als z-Achse ergibt sich das entsprechende Koordinatensystem.

Koordinatensystemorientierung

Drehungsrichtung bzgl. einer orientierten Achse: positiv bedeutet entgegen dem Uhrzeigersinn, wenn die Achse auf den Betrachter gerichtet ist.

Rechte-Hand-Regel: Zeigt der Daumen der rechten Hand in die positive Richtung einer (orientierten) Achse, dann zeigen die Finger die positive Drehrichtung.

Homogene Koordinaten

$(\tilde{x}, \tilde{y}, \tilde{z}, w)$ mit $w \neq 0$

repräsentiert den Punkt

$$\left(\frac{\tilde{x}}{w}, \frac{\tilde{y}}{w}, \frac{\tilde{z}}{w}\right) \in \mathbb{R}^3$$

$(x, y, z) \in \mathbb{R}^3$ wird durch $(x, y, z, 1)$ oder allgemeiner durch $(x \cdot w, y \cdot w, z \cdot w, w)$ dargestellt ($w \neq 0$).

Geometrische Transformationen

Translation um den Vektor $(d_x, d_y, d_z)^\top$:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{pmatrix}$$

Translationsmatrix: $$T(d_x, d_y, d_z) = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Geometrische Transformationen

Skalierung um die Faktoren s_x, s_y, s_z:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} s_x \cdot x \\ s_y \cdot y \\ s_z \cdot z \\ 1 \end{pmatrix}$$

Skalierungsmatrix: $$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Geometrische Transformationen

Rotation um die z-Achse um den Winkel θ:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

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

Geometrische Transformationen

Rotation um die x-Achse um den Winkel θ:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

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

Geometrische Transformationen

Rotation um die y-Achse um den Winkel θ:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

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

Geometrische Transformationen

Rotation um eine beliebige Achse um einen Winkel θ:

- Verschiebung der Rotationsachse in den Koordinatenursprung
- Rotation um die z-Achse, so dass die Rotationsachse in der y/z-Ebene liegt
- Rotation um die x-Achse, die die Rotationsachse auf die z-Achse abbildet
- Rotation um den Winkel θ um die z-Achse
- Umkehrung der drei vorhergehenden Transformationen

$$T(-d_x, -d_y, -d_z) \circ R_z(-\theta_z) \circ R_x(-\theta_x) \circ R_z(\theta) \circ R_x(\theta_x) \circ R_z(\theta_z) \circ T(d_x, d_y, d_z)$$

Geometrische Transformationen

Wie im 2D-Fall kann die Hintereinanderschaltung von geometrischen Transformationen mittels Matrixmultiplikation gelöst werden.

Für alle oben spezifizierten Matrizen lautet die letzte Zeile $(0, 0, 0, 1)$. Diese Eigenschaft bleibt auch bei der Matrixmultiplikation erhalten.

Im 2D-Fall gibt es genau eine Transformationsmatrix, die drei nichtkollineare Punkte auf drei andere nichtkollineare Punkte abbildet.

Im 3D-Fall gibt es genau eine Transformationsmatrix, die vier nichtkoplanare Punkte auf vier andere nichtkoplanare Punkte abbildet.

Geometrische Transformationen

Sind vier Punkte (Vektoren) $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$, die nicht in einer Ebene liegen in 3D-Koordinaten bzw. die den gesamten vierdimensionalen Raum aufspannen in homogenen Koordinaten, und deren neue Koordinaten $\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3, \mathbf{p}'_4$ gegeben, lässt sich die Transformationsmatrix auch durch Lösung eines linearen Gleichungssystems berechnen.

$$\mathbf{p}'_i = M \cdot \mathbf{p}_i \qquad (i = 1, 2, 3, 4)$$

(in homogenen Koordinaten) mit

Geometrische Transformationen

$$M = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In diesem Sinne kann eine Transformation auch als Wechsel des Koordinatensystems aufgefasst werden.

Java 3D Transformationen

Die Klasse `Transform3D` speichert analog zum 2D-Fall 3D-Transformationen als Matrix (bzgl. homogener Koordinaten).

- `Transform3D tf = new Transform3D();` erzeugt die Identität.
- `tf.rotX(theta);` definiert `tf` als Rotation um den Winkel `theta` um die x-Achse. Entsprechend die Methoden `rotY` und `rotZ`.
- Mittels

  `tf.set(new AxisAngle4d(x,y,z,theta));`

  wird eine Rotation um den Winkel `theta` um die Achse in Richtung des `float`-Vektors $(x, y, z)^\top$ definiert.

Java 3D Transformationen

- Mittels

  `tf.setTranslation(new Vector3f(x,y,z));`

  wird eine Translation um den `float`-Vektor $(x, y, z)^\top$ definiert.
- Mittels

  `tf.setScale(new Vector3f(x,y,z));`

  wird die Skalierung $S(x, y, z)$ definiert.

  `tf.setScale(factor)` entspricht einer Skalierung um den Faktor `factor` in x-, y- und z-Richtung.

Java 3D Transformationen

- Mittels `tf.set(matrix);` kann eine beliebige Transformation definiert werden.

  Dabei ist `matrix` ein (eindimensionales) `double`-Array mit 16 Werten, die die Matrixeinträge definieren.
- `tf.get(matrix);` speichert die der Transformation `tf` zugeordnete Matrix in dem (eindimensionalen) `double`-Array `matrix`.
- Hintereinanderschaltung (Matrixmultiplikation) von Transformationen:
  - `tf.mul(tf1,tf2);`
  - `tf1.mul(tf2);`

Java 3D Koordinatensystem

Sofern nicht spezielle Transformationen angewendet wurden oder der Betrachterstandpunkt geändert wurde, zeigt

- die x-Achse im Darstellungsfenster nach rechts,
- die y-Achse nach oben und
- die z-Achse nach vorn.

Elementare geometrische Objekte

Objektoberflächen kann eine Farbe (oder auch eine Textur) zugeordnet werden.

Die alleinige Angabe einer Farbe genügt nicht.

Es müssen zusätzlich Reflexionseigenschaften (z.B. der Glanz der Oberfläche) mit angegeben werden.

Daher muss für eine Oberfläche immer eine `Appearance` definiert werden.

Die Eigenschaften von `Appearance` werden im Detail im Zusammenhang mit Beleuchtung erläutert.

Elementare geometrische Objekte

An dieser Stelle wird eine `Appearance` folgendermaßen vereinfacht generiert:

```
Appearance myApp = new Appearance();
setToMyDefaultAppearance(myApp,
                         new Color3f(r,g,b));
```

mit drei `float`-Werten `r,g,b` unter Verwendung der selbst erstellten Methode `setToMyDefaultAppearance`.

Die drei `float`-Werte `r,g,b`$\in [0,1]$ geben entsprechend den Rot-, Grün- und Blauanteil der Farbe an.

Elementare geometrische Objekte

Sofern nicht anders angegeben, sind bei Fließkommawerten im Folgenden `float`-Werte zu verwenden.

Quader: Mittels

```
Box xyzBox = new Box(x,y,z,myApp);
```

wird ein Quader der Größe $(2x) \times (2y) \times (2z)$ erzeugt, dessen Mittelpunkt sich im Koordinatenursprung befindet.

Kugeln: Mittels

```
Sphere rSphere = new Sphere(r,myApp);
```

wird eine Kugel mit Radius `r` mit Mittelpunkt im Koordinatenursprung erzeugt.

Elementare geometrische Objekte

Zylinder: Mittels

```
Cylinder rhCylinder =
         new Cylinder(r,h,myApp);
```

wird ein Zylinder mit Radius `r` und Höhe h erzeugt, dessen Mittelpunkt sich im Koordinatenursprung befindet.

Der Zylinder dehnt sich jeweils um h/2 Einheiten nach oben und unten (in y-Richtung) aus.

Elementare geometrische Objekte

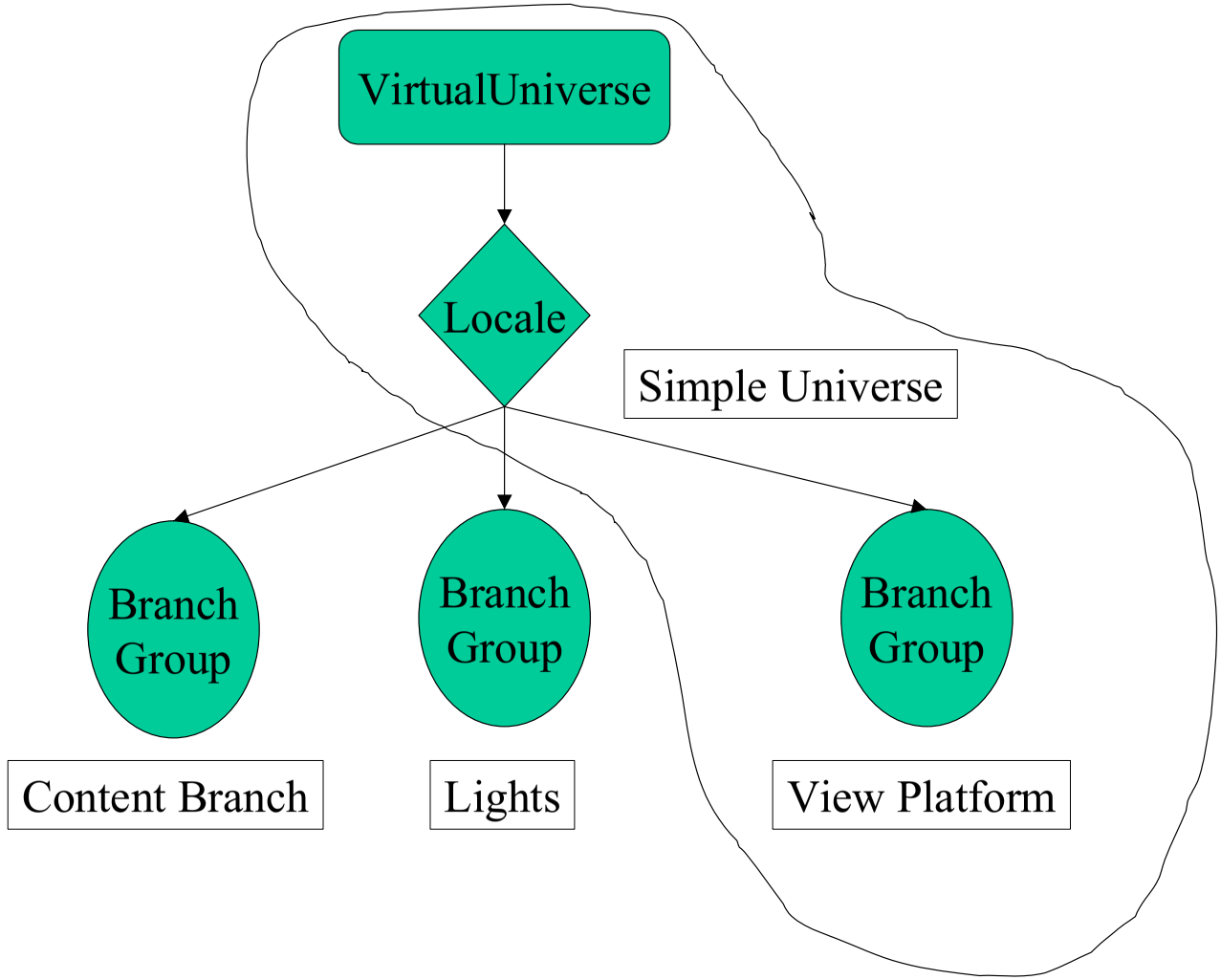
Kegel: Mittels

```
Cone rhCone = new Cone(r,h,myApp);
```

wird ein Kegel mit Radius `r` und Höhe `h` erzeugt.

Der Mittelpunkt des Grundkreises des Kegels befindet sich h/2 Einheiten unterhalb des Koordinatenursprungs.

Der Java 3D Szenengraph



Der Java 3D Szenengraph

- Die geometrischen Objekte der Szene (inklusive eventueller Bewegungen der Objekte) befinden sich im Content Branch.
- Die View Platform (View Branch) definiert den Betrachterstandpunkt und die Art der Projektion.
- In Lights wird definiert, wie die Szene zu beleuchten ist.

Das Simple Universe stellt eine einfache View Platform automatisch zur Verfügung.

Struktur eines Java 3D Progamms

```
import ...

public class MyJava3DClass extends JFrame
{
  public Canvas3D myCanvas3D;

  public MyJava3DClass()
  {
    ...
  }

  public static void main(String[] args)
  {
     MyJava3DClass myJava3D = new MyJava3DClass();
  }
}
```

public MyJava3DClass()

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
myCanvas3D = new Canvas3D(
        SimpleUniverse.getPreferredConfiguration());
SimpleUniverse simpUniv =
                        new SimpleUniverse(myCanvas3D);
simpUniv.getViewingPlatform(
                        ).setNominalViewingTransform();
createSceneGraph(simpUniv);
addLight(simpUniv);
setTitle("Ueberschrift");
setSize(700,700);
getContentPane().add("Center", myCanvas3D);
setVisible(true);
```

Struktur eines Java 3D Progamms

In der Methode `createSceneGraph` werden die Objekte der Szene inklusive eventueller Bewegungen definiert.

In der Methode `addLight` wird die Beleuchtung der Szene definiert.

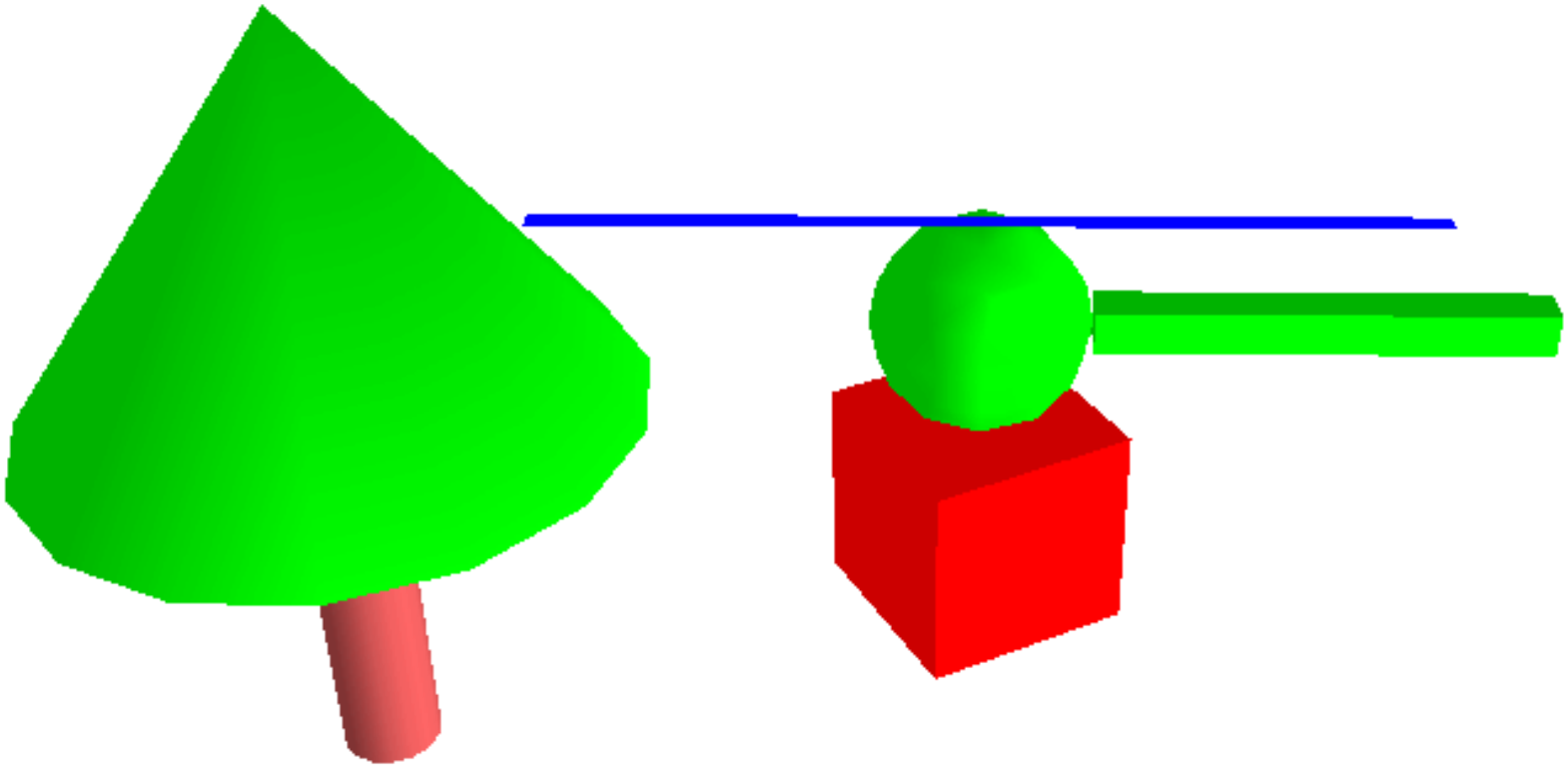
Statischer Content Branch

- Der die Objekte enthaltende Content Branch ist baumartig aufgebaut.
- Die Wurzel des Baums wird üblicherweise durch

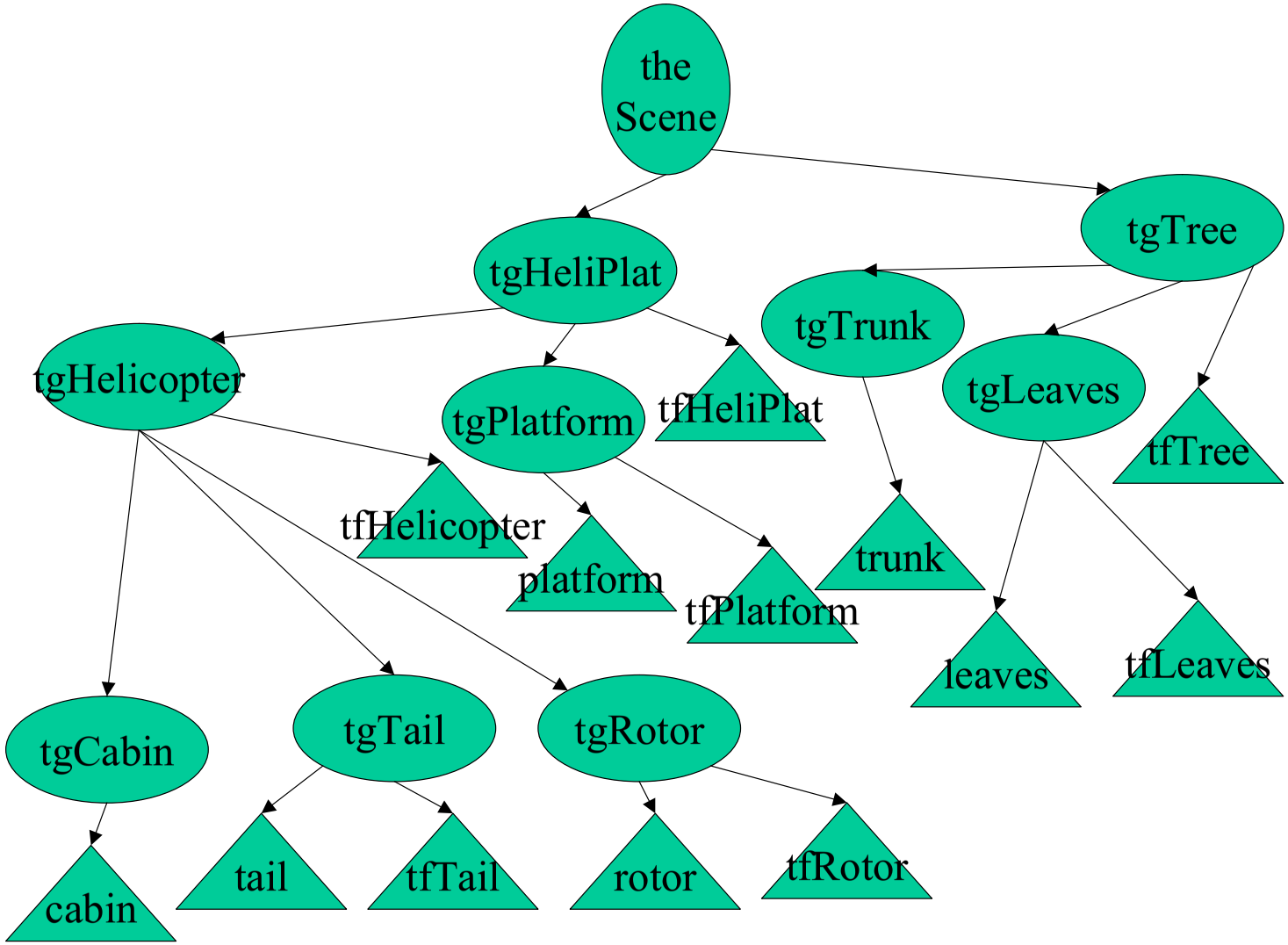
```
BranchGroup theScene =
                    new BranchGroup();
```

  erzeugt.
- Mittels `theScene.addChild(...)` werden an die Wurzel Knoten – meistens in Form von Transformationsgruppen – gehängt.

StaticSceneExample.java



Der Szenengraph



Der Szenengraph

- Blattknoten sind entweder
  - elementare geometrische Objekte (cabin, tail, rotor, platform, trunk, leaves) oder
  - Transformationen (alle Knoten, die mit `tf` beginnen), die zur korrekten Platzierung der Objekte verwendet werden.
- Alle inneren Knoten sind Transformationsgruppen (im Namen durch die ersten beiden Buchstaben `tg` gekennzeichnet).

Der Szenengraph

Transformationsgruppen dienen dazu,

- elementare geometrische Objekte mit Transformationen zu verknüpfen, um sie zu positionieren, oder
- andere Transformationsgruppen, die bereits komplexere geometrische Objekte repräsentieren, zusammenzufassen und als Ganzes zu positionieren.

Auf diese Art kann man beispielsweise den Hubschrauber im Ursprung konstruieren, ihn dann auf die Plattform setzen und danach den Hubschrauber gemeinsam mit der Plattform an eine beliebige Stelle positionieren.

Navigation

Durch Hinzufügen von

```
OrbitBehavior ob =
          new OrbitBehavior(myCanvas3D);
ob.setSchedulingBounds(
          new BoundingSphere(
               new Point3d(0.0,0.0,0.0),
                    Double.MAX_VALUE));
simpUniv.getViewingPlatform(
          ).setViewPlatformBehavior(ob);
```

kann man mit der Maus durch die Szene navigieren.

Animation (Bewegung)

Einer Transformationsgruppe kann anstelle einer statischen Transformation zur Positionierung auch eine Bewegung oder andersartige Veränderung zugeordnet werden.

Um kontinuierlich von einem Startzustand eines Objektes (oder einer Transformationsgruppe) in einen anderen Endzustand zu gelangen, werden Interpolatoren benötigt. Die wichtigsten sind:

PositionInterpolator: Lineare Interpolation zwischen einem Anfangs- und einem Endpunkt. (Translation als Animation)

Interpolatoren

RotationInterpolator: Interpolation einer Drehbewegung. (Rotation als Animation)

ScaleInterpolator: Kontinuierliche Skalierung

ColorInterpolator: Kontinuierliche Änderung der Farbe

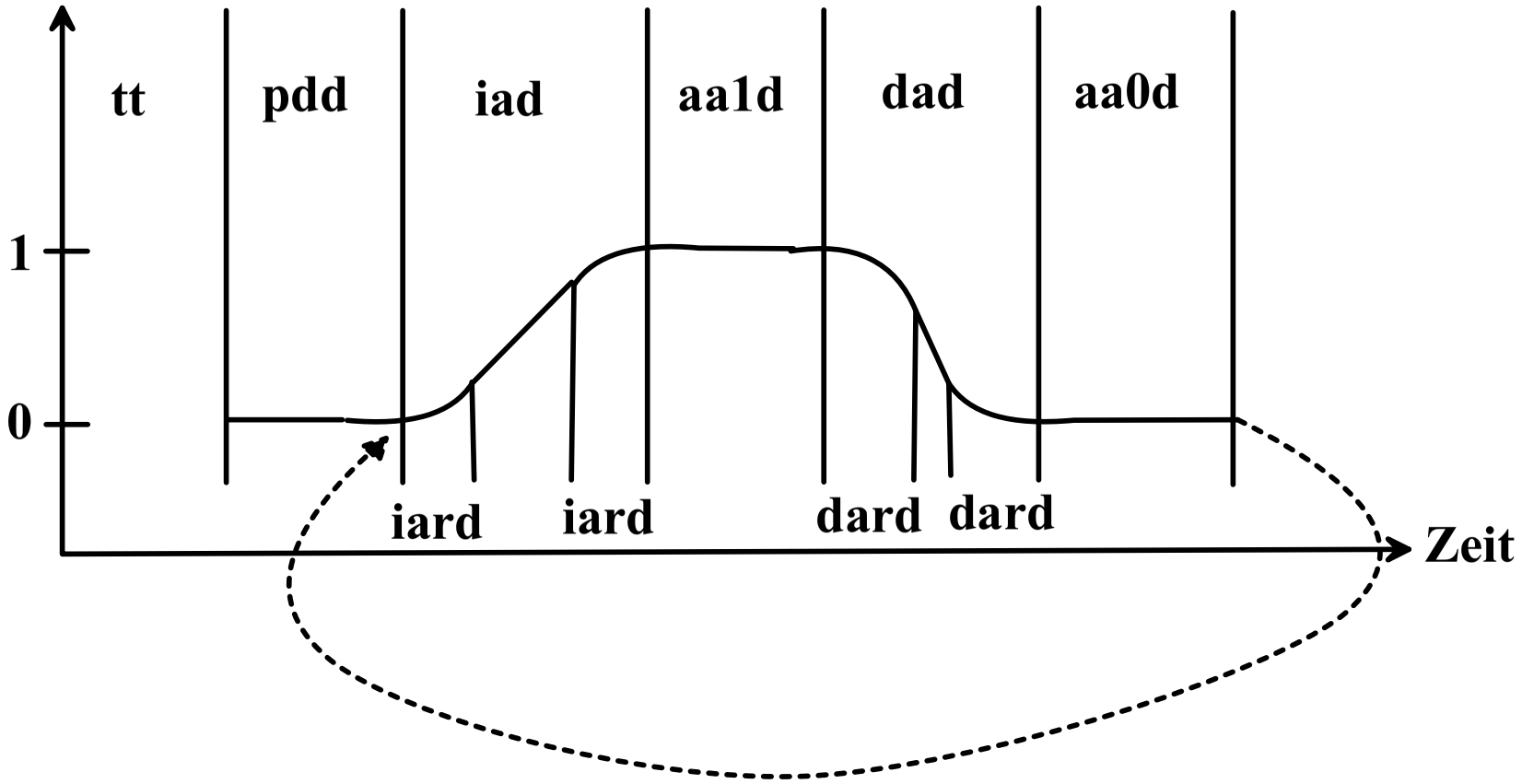
PathInterpolator: Interpolation entlang mehrerer Transformationen interpolieren. Z.B. ermöglicht der `PositionPathInterpolator` Bewegungen entlang eines Polygonzuges.

Alpha-Werte

- Wann soll der Interpolationsvorgang starten?
- Soll er nur vorwärts oder auch rückwärts ausgeführt werden?
- Wie lange dauert der Übergang vom Zustand 0 in den Zustand 1?
- Soll die Animation mit konstanter Geschwindigkeit vom Zustand 0 in den Zustand 1 übergehen oder soll man langsam aus dem Zustand 0 beschleunigen und wieder sanft abbremsen, wenn man sich dem Zustand 1 nähert?
- Soll der Interpolationsvorgang nur einmal oder mehrfach ausgeführt werden?

Alpha-Werte

```
Alpha a = new Alpha(lc,id,tt,pdd,iad,
                    iard,aa1d,dad,dard,aa0d);
```



Alpha-Werte

- `lc` definiert das Attribut `loopCount`: Anzahl der Wiederholungen (bei -1 wird der Interpolator endlos wiederholt)
- `id` definiert das Attribut `mode`:
  - `id=Alpha.INCREASING_ENABLE`: nur Interpolation von 0 nach 1
  - `id=Alpha.DECREASING_ENABLE`: nur Interpolation von 1 nach 0
  - `id=Alpha.INCREASING_ENABLE+ Alpha.DECREASING_ENABLE`: abwechselnd Interpolation von 0 nach 1 und von 1 zurück zu 0

Alpha-Werte

Alle weiteren Parameter sind vom Typ `long` und spezifizieren Zeiten in Millisekunden.

- `tt` definiert das Attribut `triggerTime`: nach wie vielen Millisekunden nach dem Start des Programms `Alpha` Werte liefern soll.
- `pdd` definiert das Attribut `phaseDelayDuration`: `Alpha` bleibt nach dem Ablauf der `triggerTime` für `phaseDelayDuration` Millisekunden beim Wert `0`.

Alpha-Werte

- `iad` definiert das Attribut `increasingAlphaDuration`: Dauer des kontinuierlichen Übergangs vom Zustand 0 in den Zustand 1
- `iard` definiert das Attribut `increasingAlphaRampDuration`: Dauer der linearen Beschleunigung bis zum Erreichen der konstanten Maximalgeschwindigkeit.
  Dieselbe Zeit `iard` wird verwendet, um die Bewegung am Ende beim Erreichen des Zustands 1 von der Maximalgeschwindigkeit linear auf die Geschwindigkeit 0 abzubremsen.

Alpha-Werte

- `aa1d` definiert das Attribut `alphaAtOneDuration`: wie lange im Zustand 1 verharrt werden soll
- `dad` und `dard` bestimmen die Attribute `decreasingAlphaDuration` bzw. `decreasingAlphaRampDuration` und haben die gleiche Bedeutung wie `increasingAlphaDuration` bzw. `increasingAlphaRampDuration`, nur für den Übergang von 1 nach 0
- `aa0d` definiert das Attribut `alphaAtZeroDuration`: Zeit, die im Zustand 0 verbracht werden soll.

PositionInterpolator

- Definition der Achse, auf der Start- und Endpunkt liegen:

  `Transform3D axis = new Transform3D();`

  definiert die x-Achse.

  Soll eine andere Achse verwendet werden, muss auf `axis` noch eine Transformation angewendet werden, die die x-Achse in die gewünschte Achse überführt.
- Definition des gewünschten `Alpha alpha`

PositionInterpolator

- Definition des `PositionInterpolator`

```
PositionInterpolator pi =
  new PositionInterpolator(
        alpha,transformgroup,axis,
        startingPoint,endPoint);
```

  - `transformgroup` ist die Transformationsgruppe, der `pi` zugeordnet werden soll.
  - `startingPoint` und `endPoint` spezifieren den Anfangs- und Endpunkt für die Interpolation auf der Achse `axis` an.

PositionInterpolator

- Es muss ein räumlicher Wirkungsbereich des Interpolators festgelegt werden:

```
BoundingSphere bs =
   new BoundingSphere(
            new Point3d(0.0,0.0,0.0),
            Double.MAX_VALUE);

pi.setSchedulingBounds(bs);
```

- Für `transformgroup` muss Bewegung (Veränderung) zugelassen werden:

```
transformgroup.setCapability(
TransformGroup.ALLOW_TRANSFORM_WRITE);
```

Interpolatoren

- Abschließend muss `pi` der Transformationsgruppe `transformgroup` angefügt werden:

```
tg.addChild(pi);
```

Analog wird ein `RotationsInterpolator` definiert:

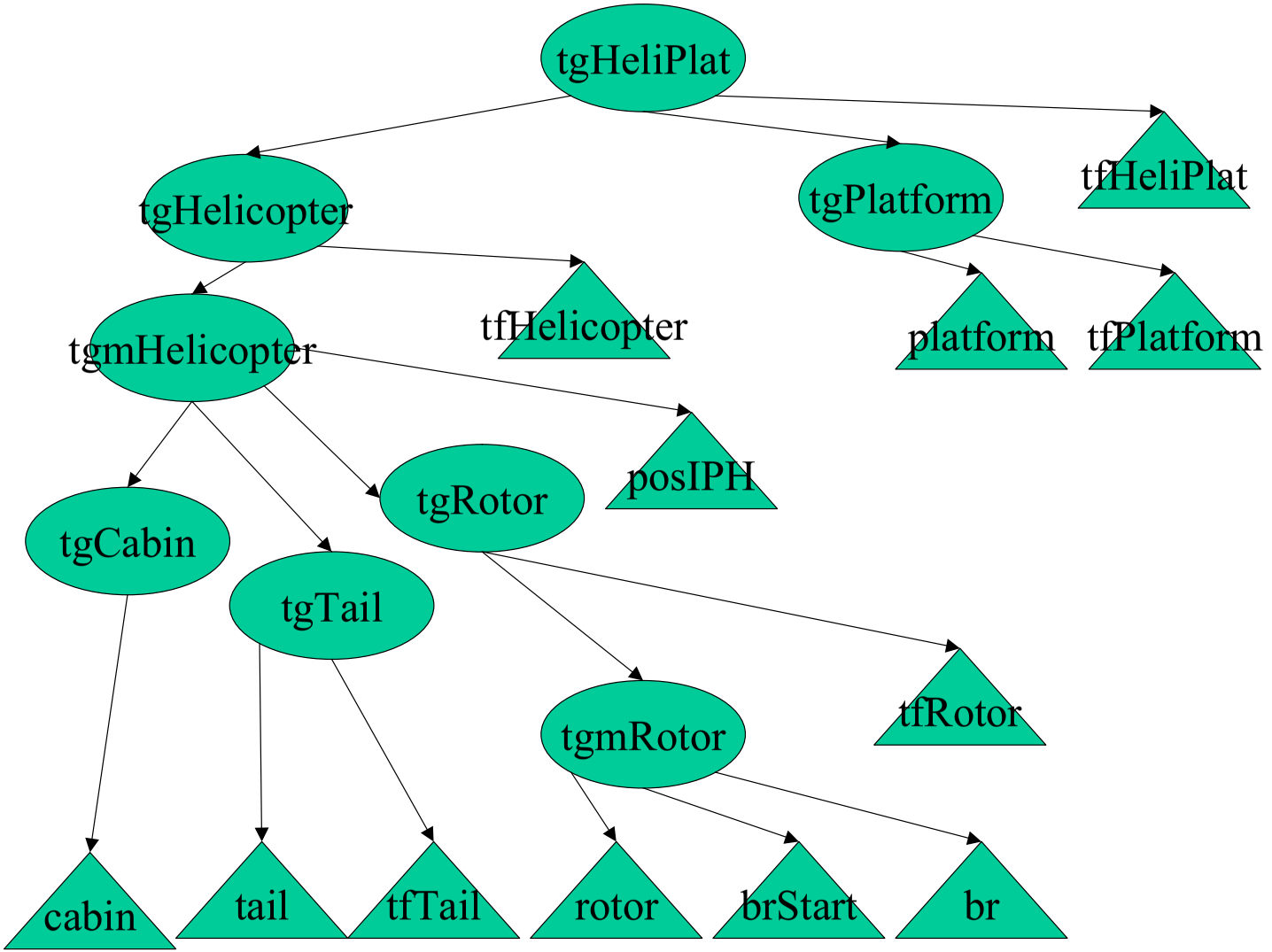
```
RotationInterpolator ri =
  new RotationInterpolator(
        alpha,transformgroup,axis,
        startAngle,endAngle);
```

Interpolatoren

```
ScaleInterpolator si =
  new ScaleInterpolator(
        alpha,transformgroup,axis,
        minScale,maxScale);
```

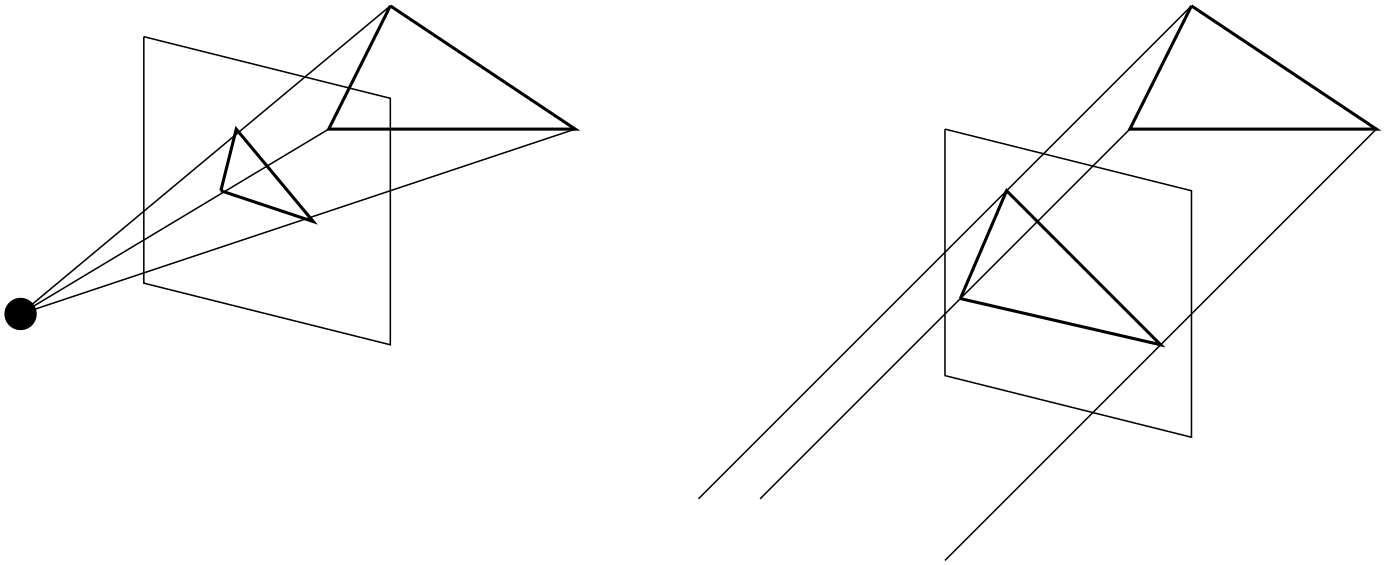
definiert einen `ScaleInterpolator`, der eine kontinuierliche Skalierung vornimmt, angefangen beim Skalierungsfaktor `minScale` bis zum Skalierungsfaktor `maxScale`.

Szenengraphausschnitt



Projektion

Die **3D→2D-Projektion** erhält man, indem man von einem Projektionszentrum (das auch im Unendlichen liegen darf) ausgehende Strahlen mit den Punkten des zu projizierenden Objekts verbindet und die Auftreffpunkte in einer dazwischen liegenden Projektionsebene berechnet.



perspektivische Projektion Parallelprojektion

Projektion

Liegt das Projektionszentrum im Unendlichen spricht man von **Parallelprojektion**, ansonsten von **perspektivischer Projektion**.

Das Projektionszentrum repräsentiert den Standort des Betrachters.

In Java 3D wird standardmäßig die perspektivische Projektion verwendet.

Umschaltung auf Parallelprojektion:

```
simpUniv.getViewer().getView().
setProjectionPolicy(
View.PARALLEL_PROJECTION);
```

(vgl. `ViewParallelProjection.java`)

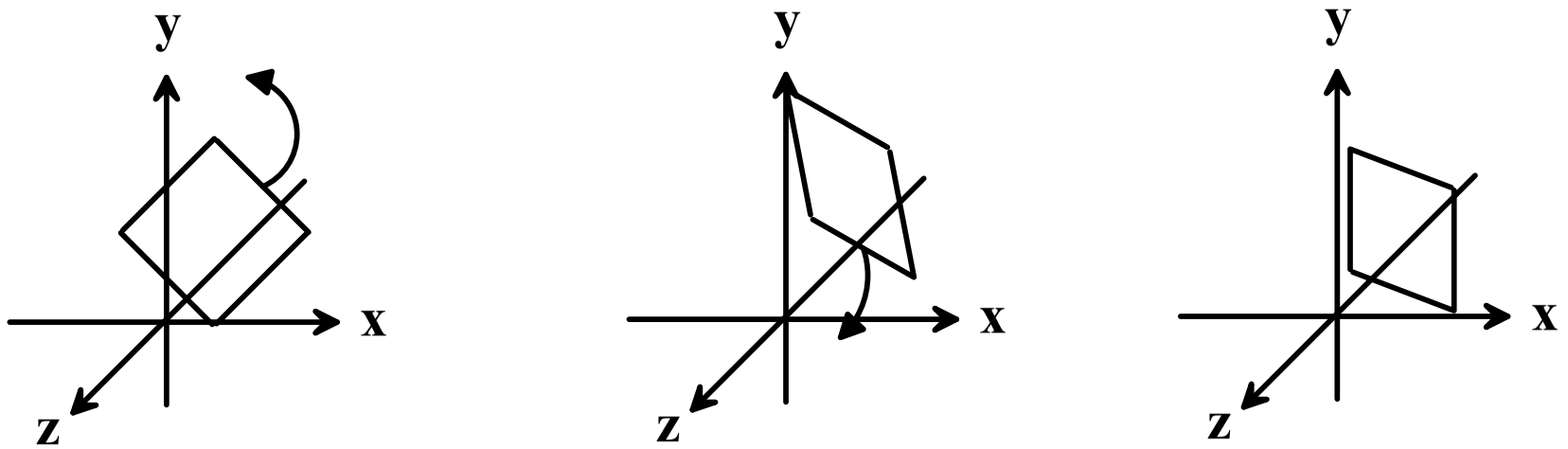
Parallelprojektion in hom. Koord.

Beispiel: Projektion auf die zur (x, y)-Ebene parallele Ebene $z = z_0$ in homogenen Koordinaten:

$$\begin{pmatrix} x \\ y \\ z_0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Parallelprojektion in hom. Koord.

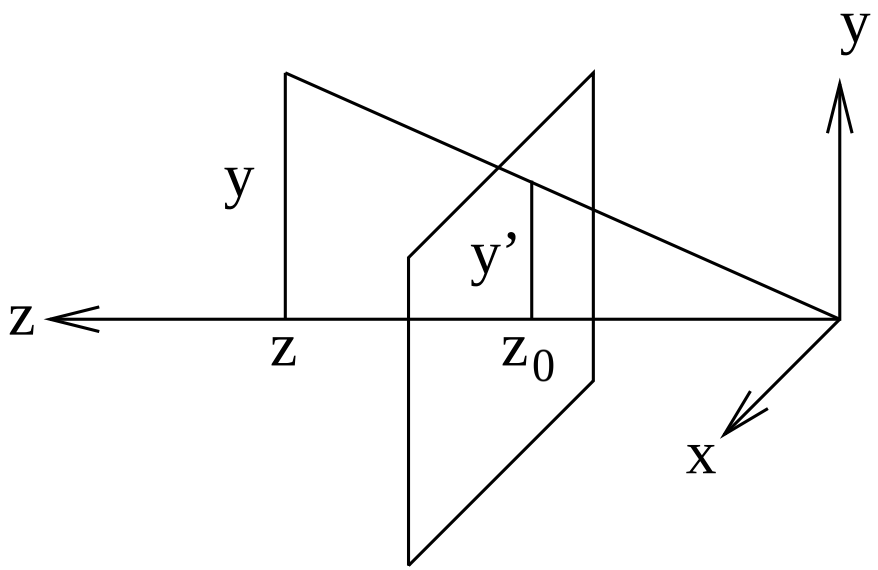
Gegebenenfalls kann das Weltkoordinatensystem immer so transformiert werden, dass diese Ebene die Projektionsebene ist.



Persp. Projektion in hom. Koord.

Beispiel: perspektivische Projektion mit Projektionszentrum im Koordinatenursprung und zur (x, y)-Ebene parallele Projektionsfläche $z = z_0$:

Für die Bildschirmkoordinaten ergibt sich mit dem Strahlensatz:



Persp. Projektion in hom. Koord.

$$\frac{x'}{x} = \frac{z_0}{z} \qquad \text{und} \qquad \frac{y'}{y} = \frac{z_0}{z}$$

bzw.

$$x' = \frac{z_0}{z} \cdot x \qquad \text{und} \qquad y' = \frac{z_0}{z} \cdot y$$

$$\begin{pmatrix} x \\ y \\ z \\ \frac{z}{z_0} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_0} & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

mit dem Ergebnis in **nicht-normierten** homogenen Koordinaten

Persp. Projektion in hom. Koord.

Legt man den Koordinatenursprung in die Projektionsebene und das Projektionszentrum nach $-z_0$, ergibt sich:

$$x' = \frac{z_0}{z_0 + z} \cdot x = \frac{x}{1 + \frac{z}{z_0}} \quad \text{und} \quad y' = \frac{z_0}{z_0 + z} \cdot y = \frac{y}{1 + \frac{z}{z_0}}$$

In Matrixform:

$$\begin{pmatrix} x \\ y \\ 0 \\ 1 + \frac{z}{z_0} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{z_0} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Persp. Projektion in hom. Koord.

perspektivische Projektion als Hintereinanderschaltung einer Transformation und einer Parallelprojektion:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{z_0} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_0} & 1 \end{pmatrix}$$

Diese Transformation (rechte Matrix) lässt Punkte in der Ebene $z = 0$ unverändert.

Persp. Projektion in hom. Koord.

Jede perspektivische Projektion kann als eine Transformation und anschließende Parallelprojektion auf die x/y-Ebene aufgefasst werden:

Durch eine Translation um einen Vektor der Form $(0, 0, z_0)$ lässt sich eine perspektivische Projektion mit Projektionszentrum im Koordinatenursprung mit Projektionsebene parallel zur x/y-Ebene auf eine perspektivische Projektion auf die x/z-Ebene mit Projektionszentrum im Punkt $(0, 0, -z_0)$ zurückführen.

Persp. Projektion in hom. Koord.

Jede beliebige perspektivische Projektion kann wiederum auf eine perspektivische Projektion mit Projektionszentrum im Koordinatenursprung zurückgeführt werden.

Dazu verschiebt man mit Hilfe einer Transformation das Projektionszentrum in den Koordinatenursprung.

Anschließend lässt sich genau wie bei der Parallelprojektion die Projektionsebene durch zwei Drehungen auf eine zur x/y-Ebene parallele Ebene abbilden.

Persp. Projektion in hom. Koord.

Parallel- und perspektivische Projektionen lassen sich durch eine geeignete vorgeschaltete Transformation auf eine Parallelprojektion auf die x/y-Ebene zurückführen.

Änderung der Betrachterposition: Änderung der vorgeschalteten Transformation

Persp. Projektion in hom. Koord.

Eigenschaften der Transformation

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_0} & 1 \end{pmatrix}$$

Betrachte den Punkt $\left(0, 0, \frac{1}{w}\right)$.

Repräsentation in homogenen Koordinaten: $(0, 0, 1, w)$

Persp. Projektion in hom. Koord.

$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ \frac{1}{z_0} + w \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_0} & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ w \end{pmatrix}$$

in kartesischen Koordinaten:

$$\left(0, 0, \frac{1}{w}\right) \mapsto \left(0, 0, \frac{z_0}{1 + z_0 \cdot w}\right)$$

$w \to 0 :$ $\qquad (0, 0, \infty) \mapsto (0, 0, z_0)$

Persp. Projektion in hom. Koord.

Transformation der Geraden, die durch den $\left(0, 0, \frac{1}{w}\right)$ verlaufen:

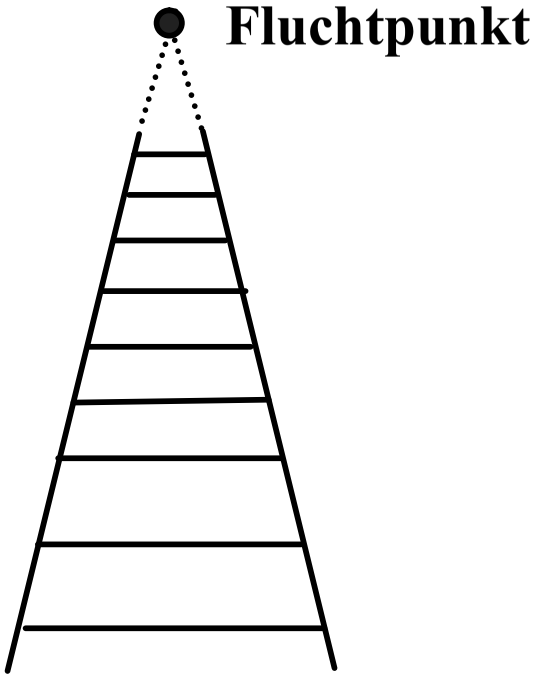
Die transformierten Geraden schneiden sich im Punkt $\left(0, 0, \frac{z_0}{1+z_0 \cdot w}\right)$.

$w \to 0$: Geradenschar geht in zur z-Achse parallele Geraden über.

Die transformierten Geraden schneiden sich jedoch alle im Punkt $(0, 0, z_0)$.

Dieser Punkt wird auch **Fluchtpunkt** genannt.

Persp. Projektion in hom. Koord.



Persp. Projektion in hom. Koord.

Schneidet die Projektionsebene zwei oder drei Koordinatenachsen, spricht man von **Zwei-** bzw. **Dreipunktperspektive**.