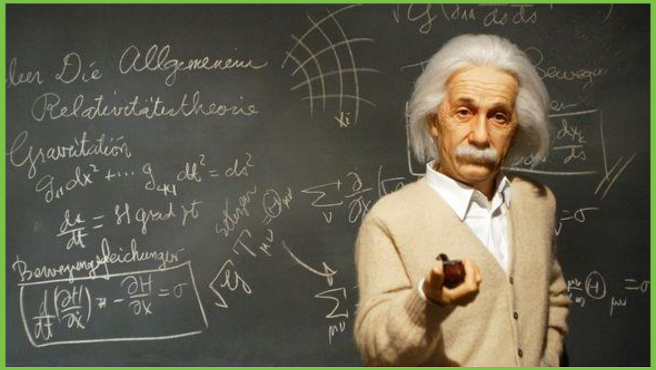
Visual Computing – 2D-/3D-Transformationen

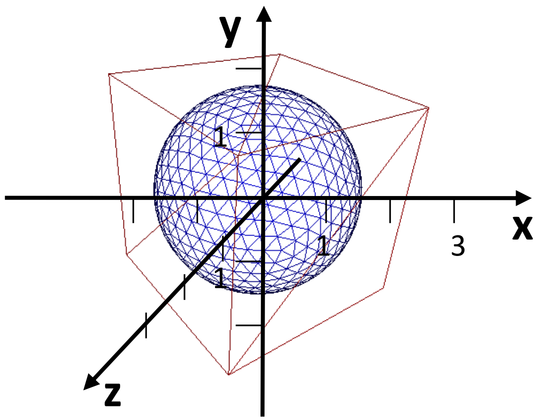


Yvonne Jung

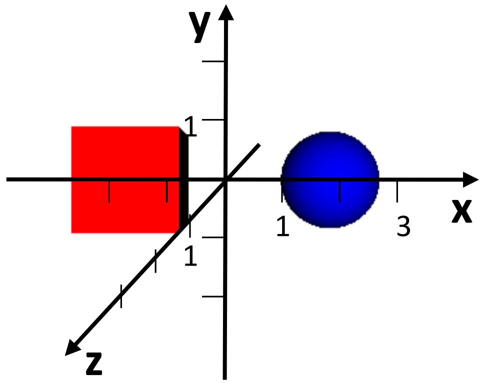
Problemstellung

- Typische Probleme bei Szenenerstellung / Modellierung
  1. Graphische Objekte in 2D- oder 3D-Szene platzieren
  2. Beziehungen zwischen virtuellen Objekten modellieren
- Lösung
  1. Transformationen für 2D- bzw. 3D-Modellierung
     - Repräsentiert durch Abbildungsmatrizen
       - Demo zum interaktiven Ausprobieren: https://www.realtimerendering.com/udacity/transforms.html
  2. Hierarchische Modellierung räumlicher Beziehungen
     - Mittels spezieller Datenstruktur: Szenengraph
     - Knoten beschreiben durch ihre Ausprägung 3D-Szene

Lokal im Ursprung erstellt …



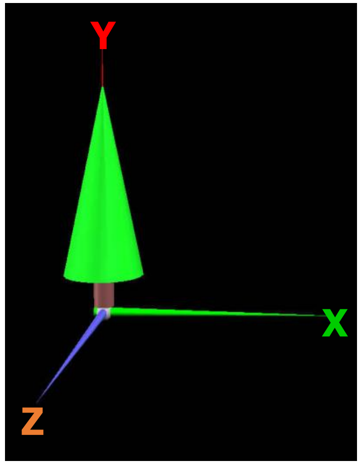
… dann in 3D-Szene platziert



Anordnung der Objekte im Raum

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

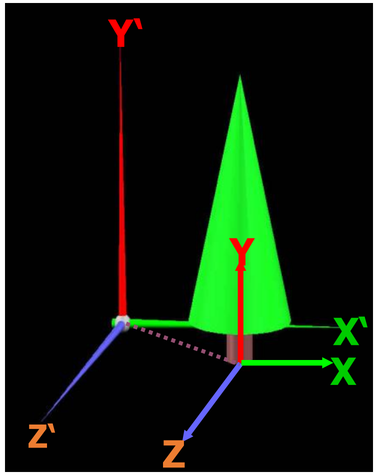
1.

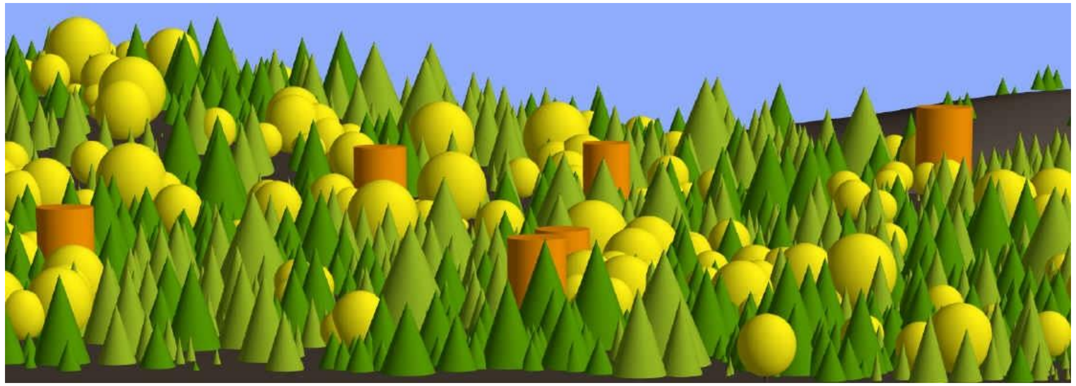


Erstellung in lokalem Koordinatensystem

Transformation des 3D-Objektes in sog. Weltkoordinatensystem

2.

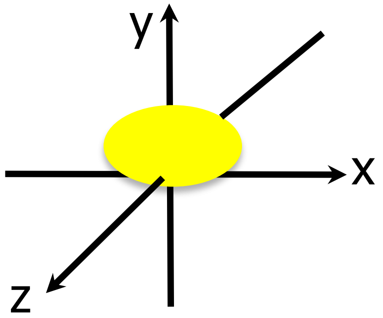




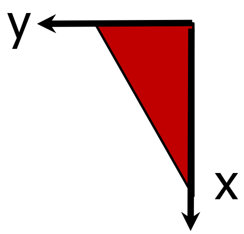
Beispiel: Erzeugen eines 3D-Objekts

„Pinguin“ im Welt-
koordinatensystem

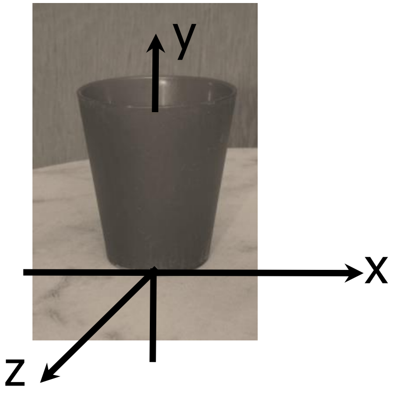
Pinguinkopf im lokalen
Koordinatensystem



Dreiecke für Flügel
und Füße im lokalen
Koordinatensystem



Becher im lokalen
Koordinatensystem



Beispiel: Erzeugen eines 3D-Objekts

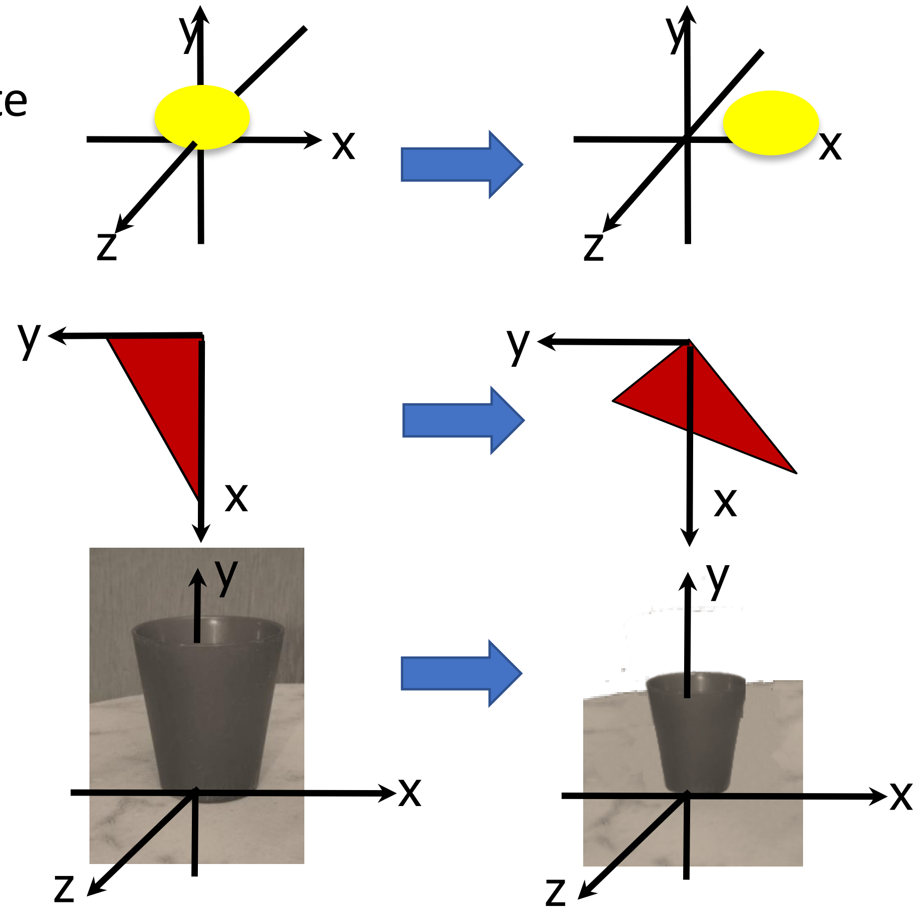
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Was fehlt noch zur vollständigen Modellierung des Pinguins?

- Neben geometrischen Größen müssen noch grafischen Attribute zugeordnet werden – diese beziehen sich auf das „Aussehen"
- Beispiele für graphische Attribute sind etwa Farbe und Textur (z.B. JPEG-Bild) auf Fläche gemappt (abgebildet, „geklebt")

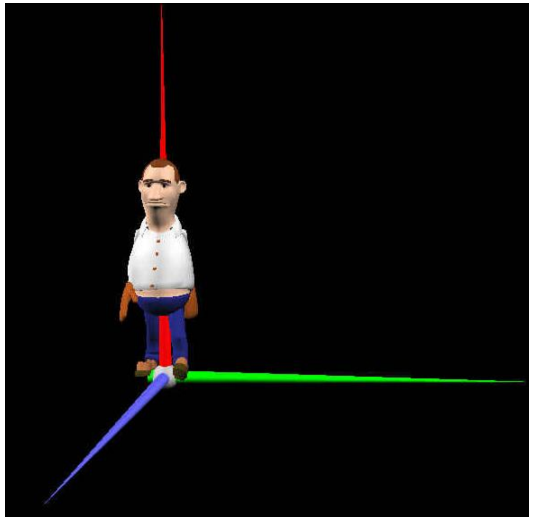
Wie kann man die Primitive im lokalen Koordinatensystem „manipulieren", um einen Pinguin zu konstruieren?

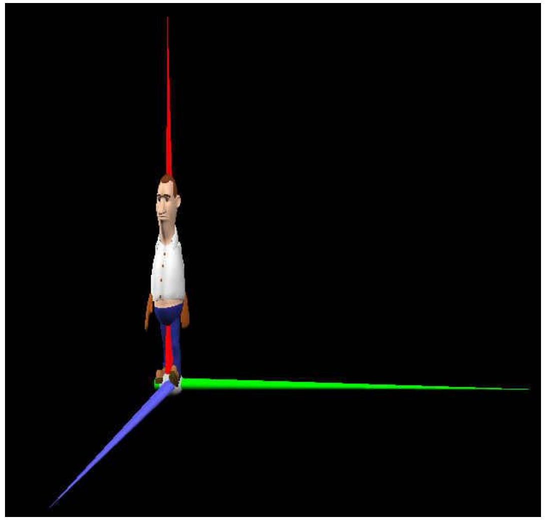
- Man kann sie verschieben (translieren),
- …drehen (rotieren) sowie
- …vergrößern und verkleinern (skalieren)
- Diese Manipulationen nennt man Transformationen

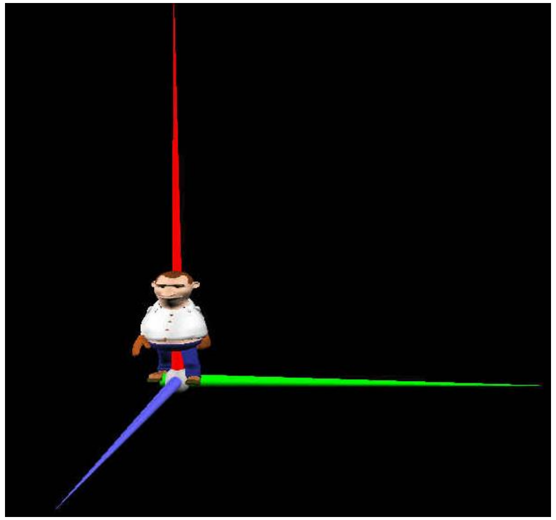


Transformationen

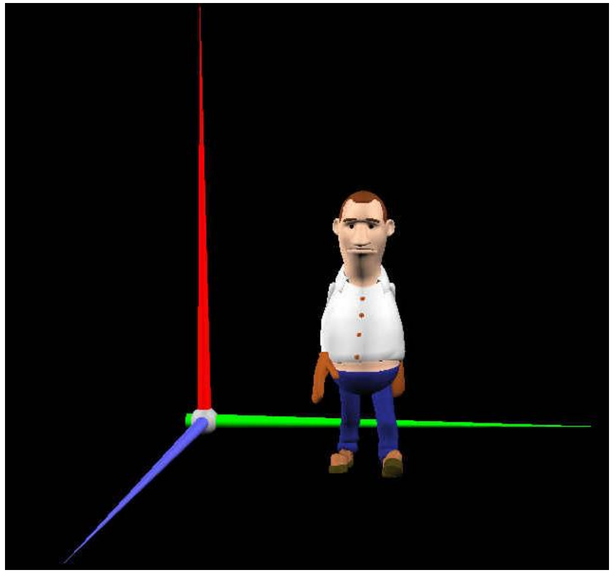
Skalierung





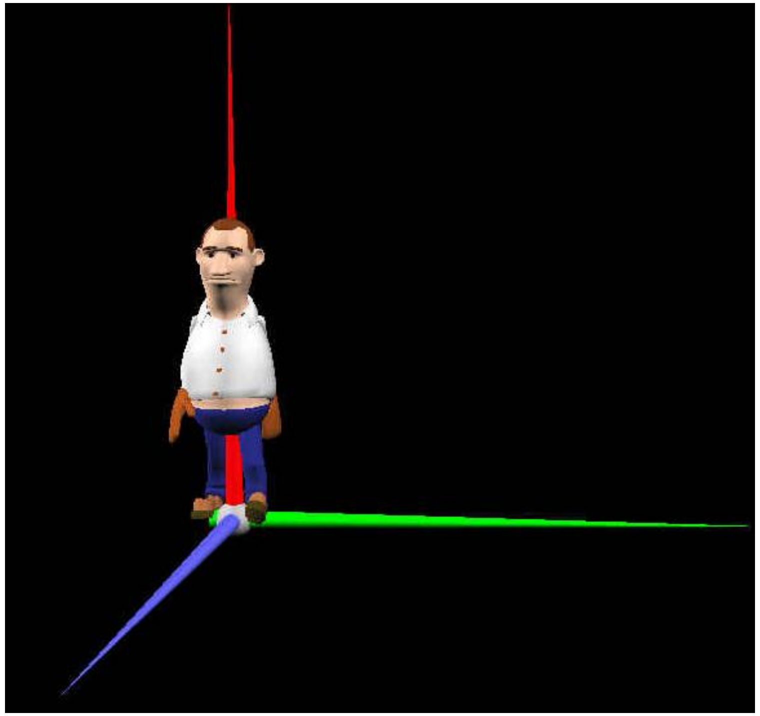


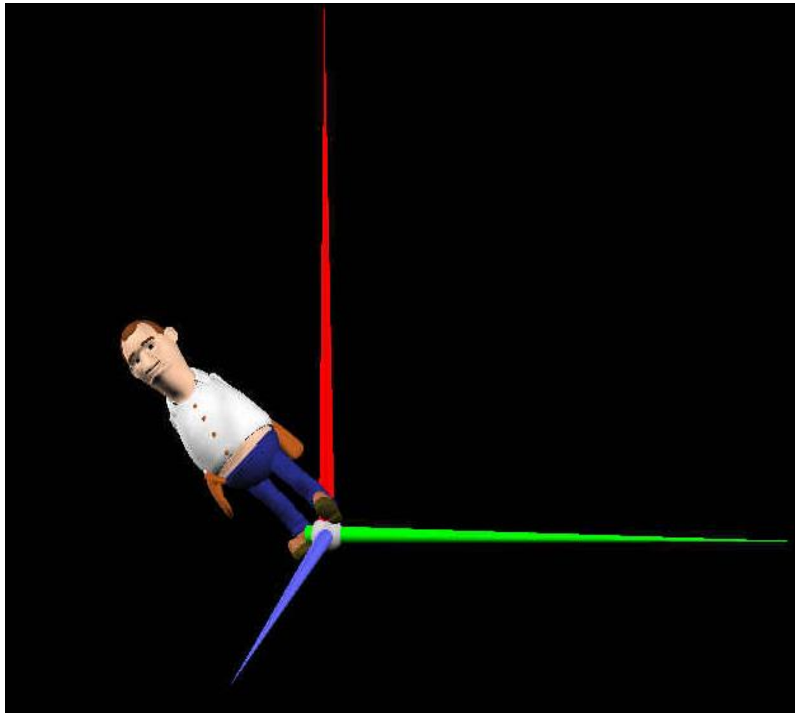
Translation

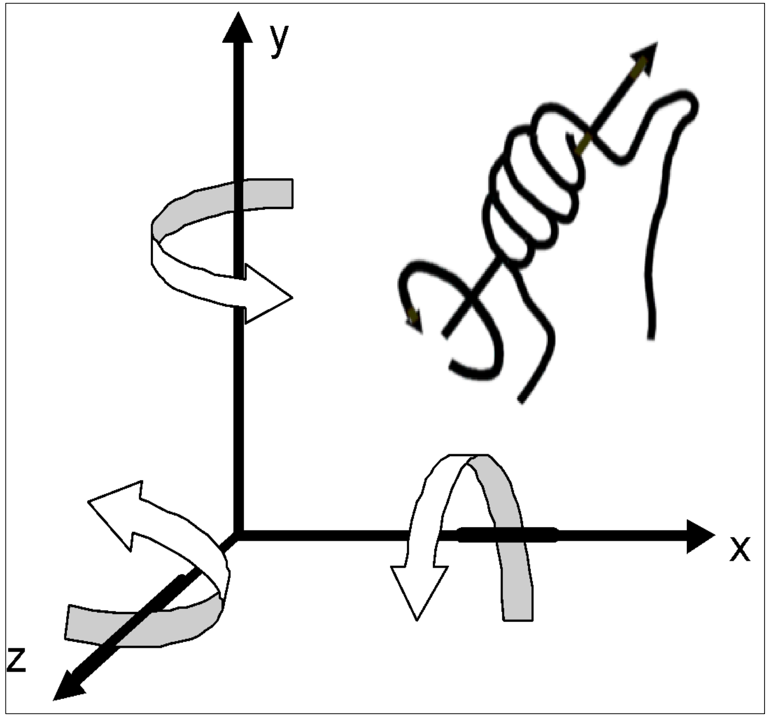


Transformationen lassen sich auch miteinander kombinieren…

Transformationen: Rotation



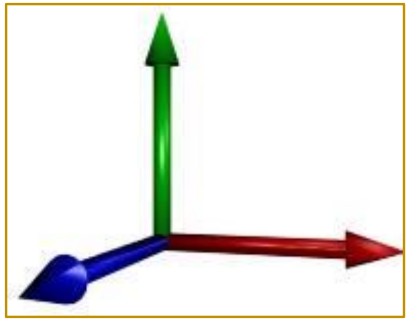




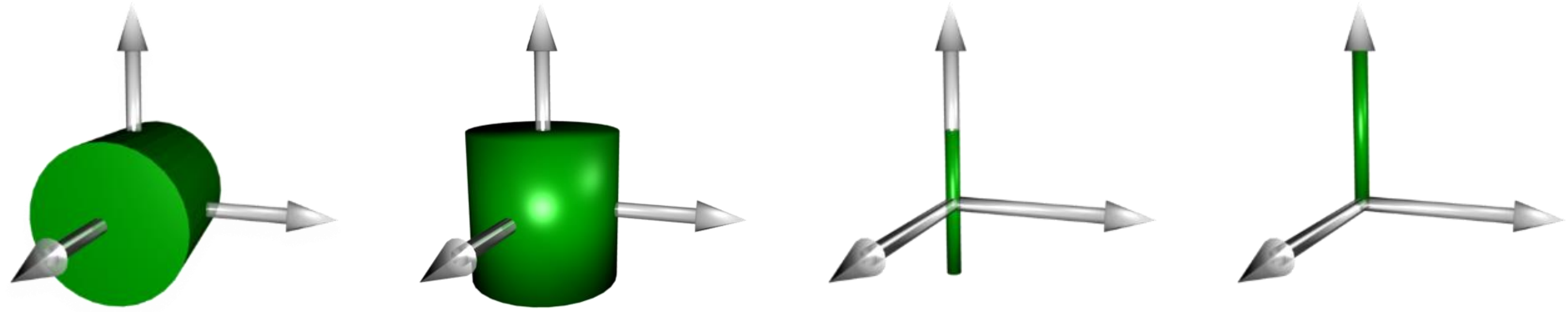
Wenn der Daumen der rechten Hand die Koordinatenachse bildet, zeigen die angewinkelten Finger die Drehrichtung an

Um welche Achse wird die Geometrie bei dieser Rotation gedreht?

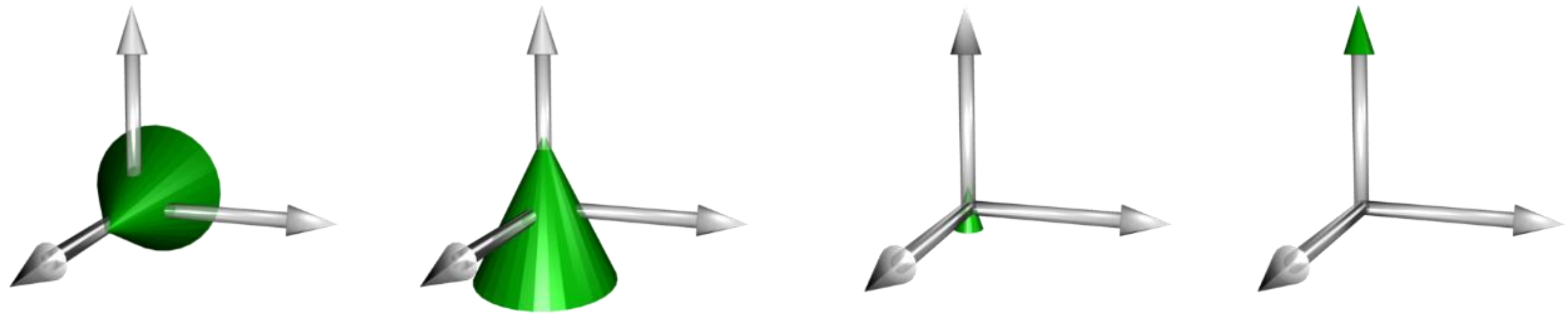
Zusammensetzen eines Modells



- Aufbau eines Koordinatenkreuzes durch Transformation von je zwei Grundprimitiven (Zylinder und Kegel/Cone)

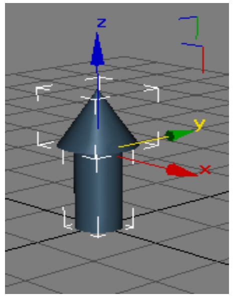


Rotation Skalierung Translation

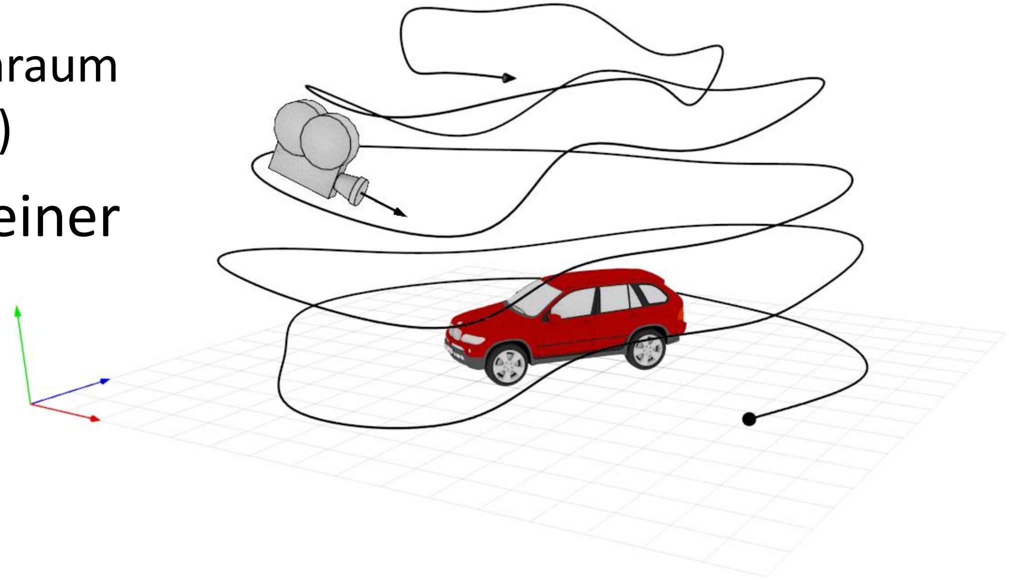


Ausgangszustand *Schritt 1* *Schritt 2* *Schritt 3*

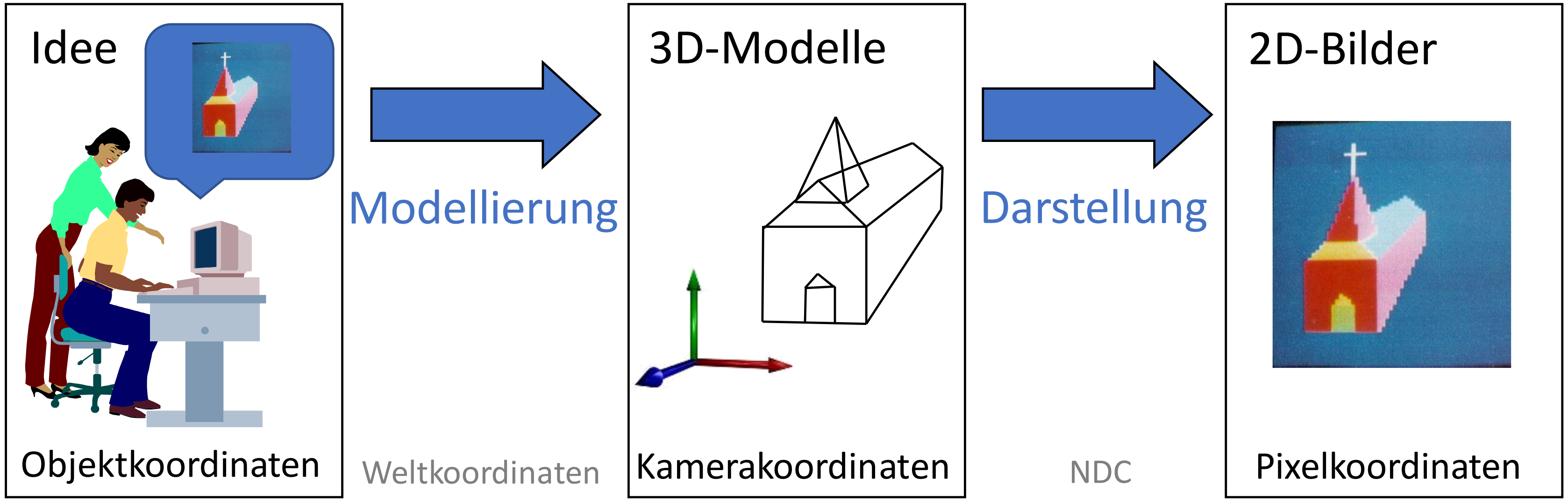
Geometrische Transformationen



- Operationen, die auf die geometrische Beschreibung virtueller Objekte angewendet werden, um Position, Orientierung oder Größe zu ändern
  - Beispiel: Animationsdesigner erstellt Videosequenz durch Bewegen der Kameraposition entlang eines Pfades
  - Transformation ***M*** bildet Punkt von einem Koordinatenraum auf Punkt ***p'*** in einem anderen ab: (x', y', z') = ***M*** (x, y, z)
- Transformationen werden mittels Multiplikation einer Matrix mit einem Vektor umgesetzt
  - $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

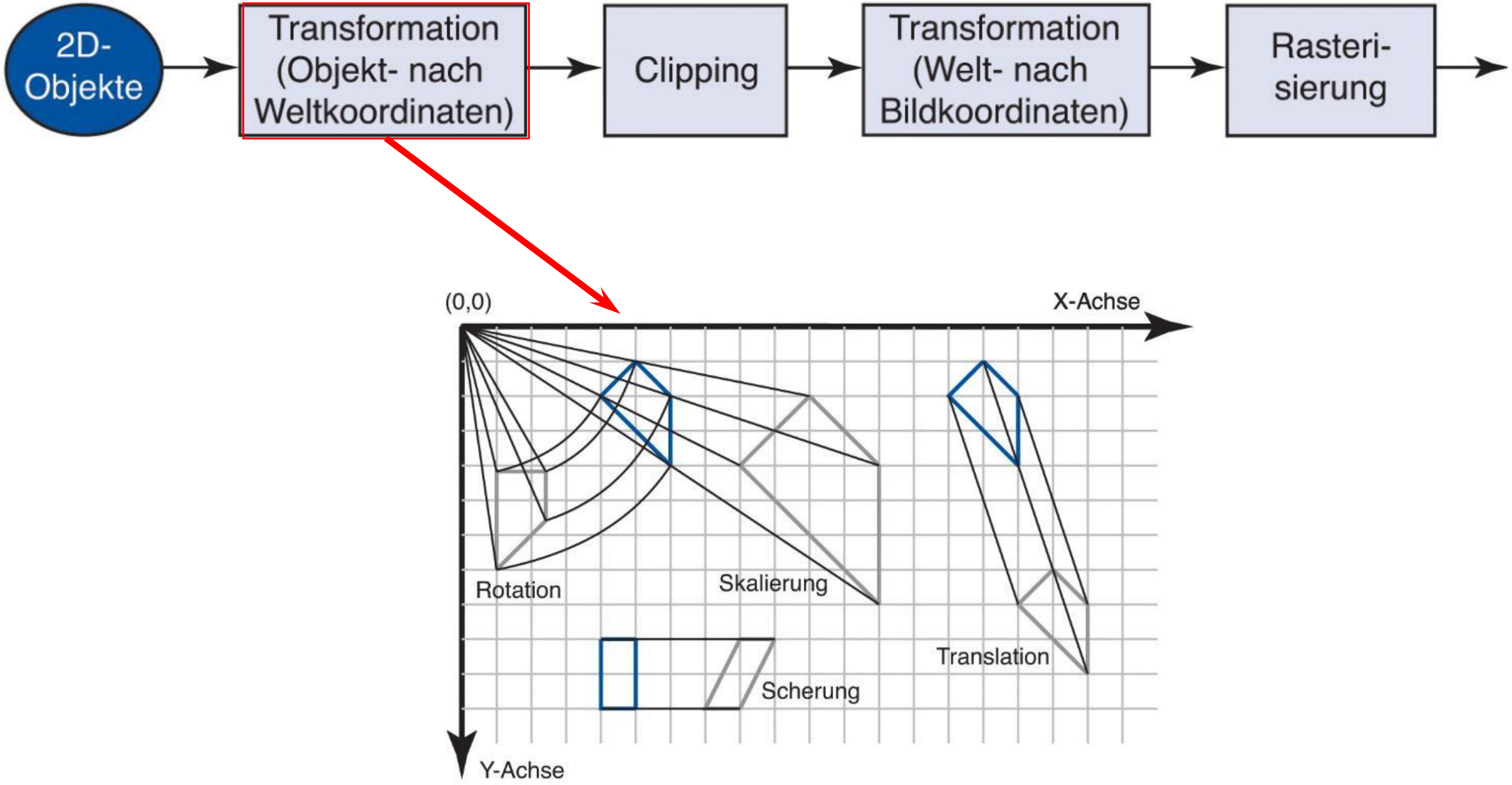


Ablauf Bilderzeugung bei 3D-Graphik



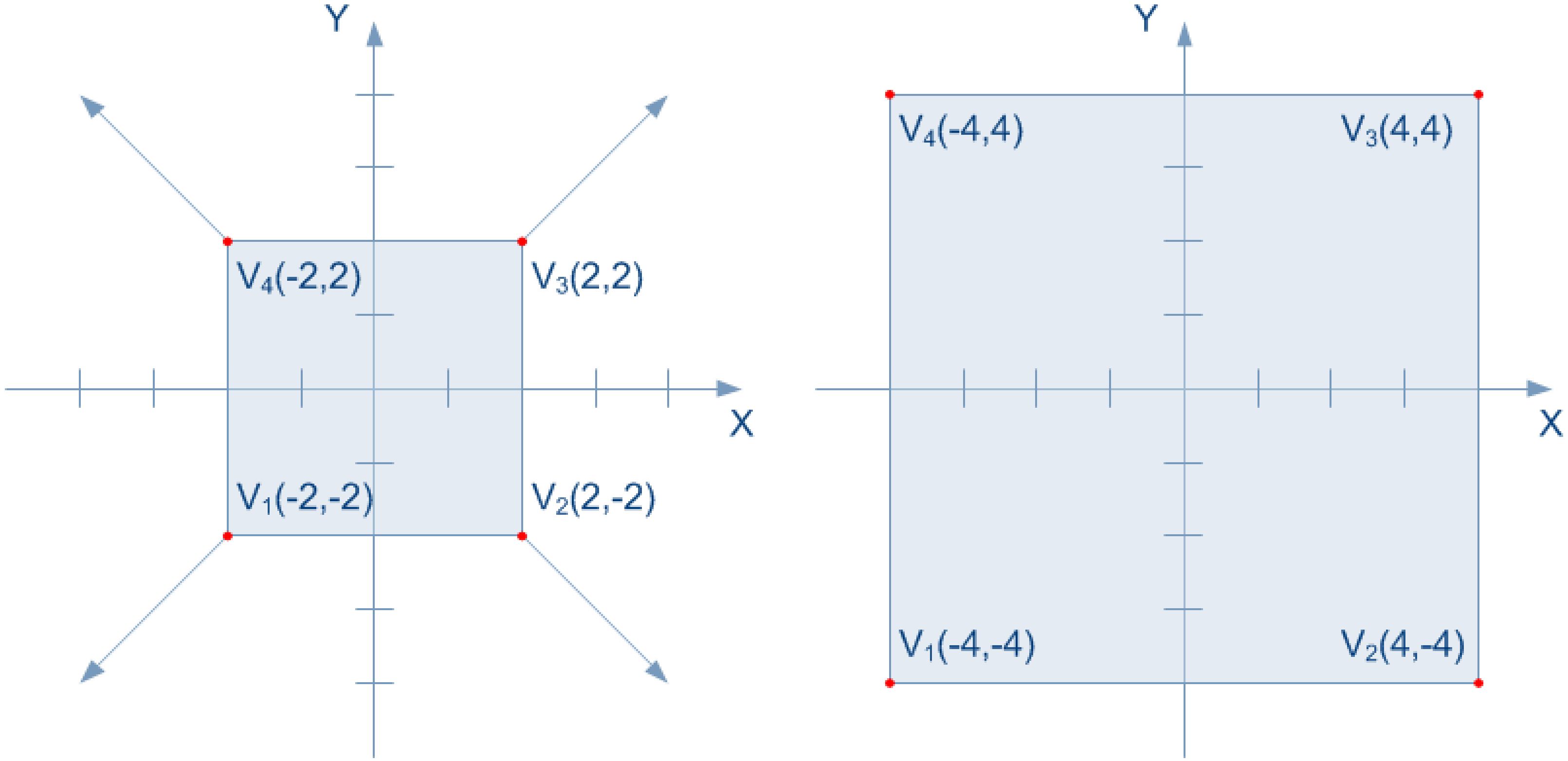
- Transformationen für **Modellierung**
  - …der Eckpunkte $\vec{p}$ eines Objekts von Objekt- in Weltkoordinaten, mit $\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$
  - …dann von Welt- in Kamerakoordinaten
- Transformationen für **Darstellung**

2D-Transformationen



Skalierung

- Stauchen oder Strecken der Eckpunkte eines Objekts entlang der Koordinatenachsen mit Faktor *s*



Skalierung

- Bei uniformer Skalierung ist *s* für alle Achsen gleich
  - Bei nicht-uniformen Skalierungen ist Faktor $s_x \neq s_y$
- Berechnung möglich über komponentenweise Multiplikation mit Vektor ***s***
  - Für alle Eckpunkte: $\vec{v}' = \vec{s} \otimes \vec{v}$ bzw. $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x \\ s_y \end{pmatrix} \otimes \begin{pmatrix} x \\ y \end{pmatrix}$

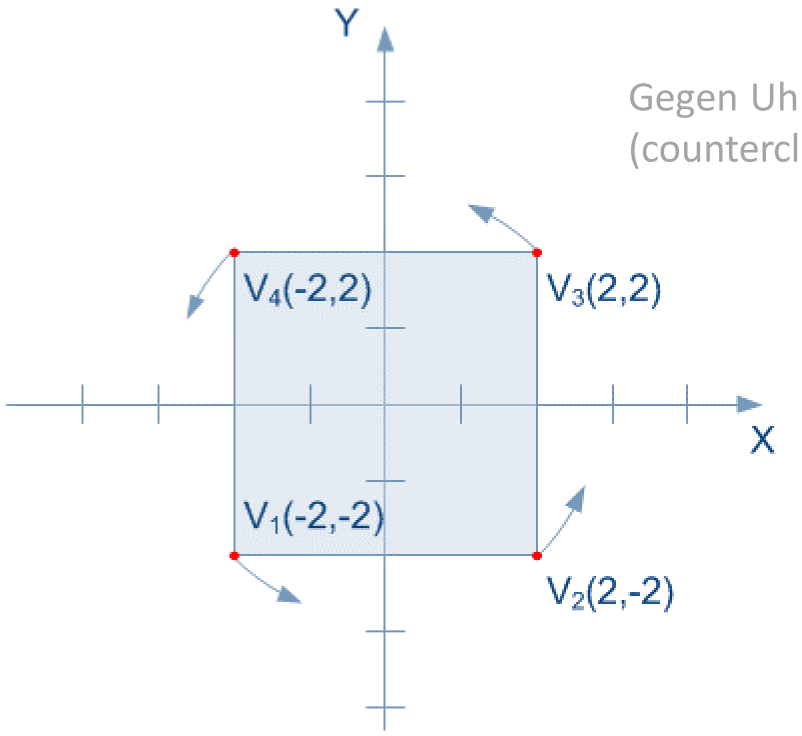
    $\otimes$ komponentenweise Multiplikation
    - Es gilt also: $x' = s_x \cdot x$ und $y' = s_y \cdot y$
    - Bzw. für Spezialfall uniformer Skalierung: $\vec{v}' = s \cdot \vec{v}$
- Allgemein wird aber Matrixschreibweise genutzt
  - $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$

Skalierung nur bzgl. Nullvektor invariant

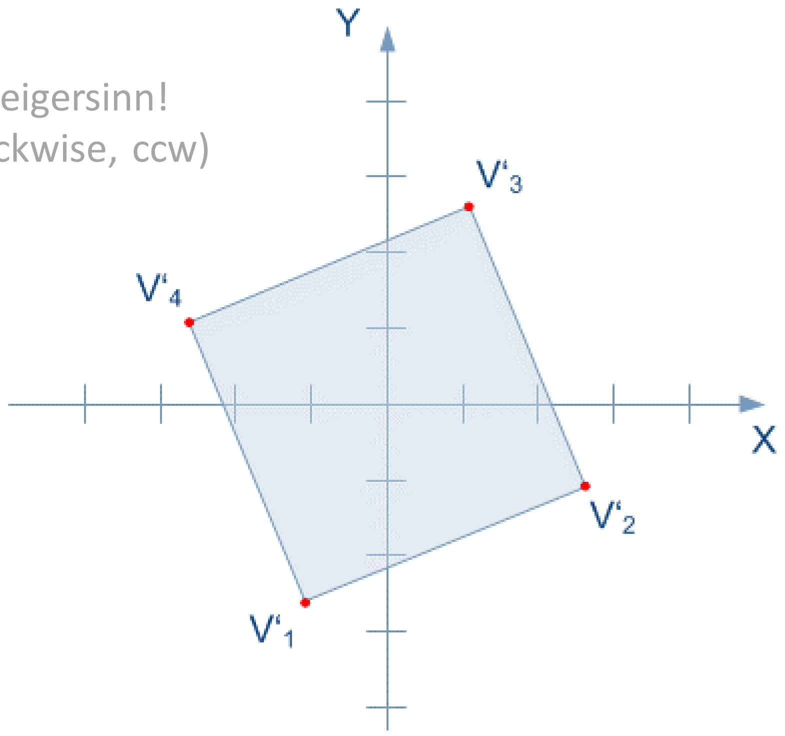
(Nur der Ursprung wird für $s_x, s_y \neq 1$ auf sich selbst abgebildet)

Rotation

- Drehung der Eckpunkte um Winkel α (um Ursprung)
  - Entspricht bei Erweiterung auf 3D der Drehung um *z*-Achse
  - Allgemein im 3D: Drehung um eine der Koordinatenachsen



Gegen Uhrzeigersinn!
(counterclockwise, ccw)



Rotation nur bzgl. Nullvektor invariant

(Nur der Ursprung wird für $\alpha \neq 0°$ auf sich selbst abgebildet)

Rotation

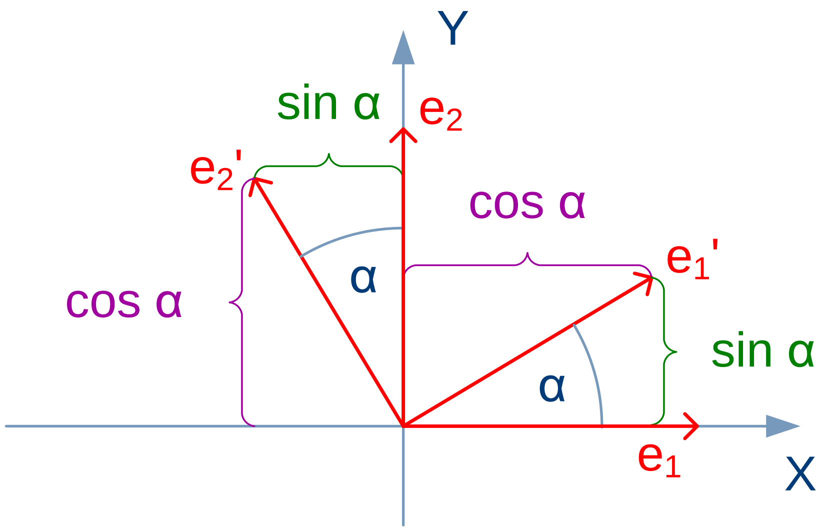
- Rotieren aller Eckpunkte eines 2D-Objekts um Winkel α (um gedachte *z*-Achse) gegen Uhrzeigersinn
  - 1. Einheitsvektor e_1 drehen:
    $e_1' = R_\alpha ((1, 0)^T) = (\cos \alpha, \sin \alpha)^T$
  - 2. Einheitsvektor e_2 drehen:
    $e_2' = R_\alpha ((0, 1)^T) = (-\sin \alpha, \cos \alpha)^T$
  - Allgemein für $\vec{v} = (x, y)^T$:

$$x' = x \cdot \cos\alpha - y \cdot \sin\alpha$$

$$y' = x \cdot \sin\alpha + y \cdot \cos\alpha$$

- Matrixschreibweise

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$



Neue Spaltenvektoren sind Bilder der ursprünglichen Basisvektoren

Übung 1

- Gegeben sei ein Quadrat mit den Eckpunkten A(-1, -1), B(1, -1), C(1, 1) und D(-1, 1). Tragen Sie das Quadrat in ein Koordinatensystem ein. Multiplizieren Sie alle vier Eckpunkte mit der Matrix $S = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$ und tragen selbige ebenfalls in das Koordinatensystem ein (inkl. Kanten).
- Geg. sei die Matrix $R_\alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$. Setzen Sie für α in 60°-Schritten Werte zwischen 0° und 360° ein und multiplizieren Sie die sich je ergebende Matrix mit dem Ortsvektor $\vec{p} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

  Zeichnen Sie das Ergebnis $\overrightarrow{p_\alpha{}'}$ wieder in ein Koordinatensystem ein und verbinden Sie je aufeinanderfolgende Ergebnispunkte. Was fällt Ihnen auf?

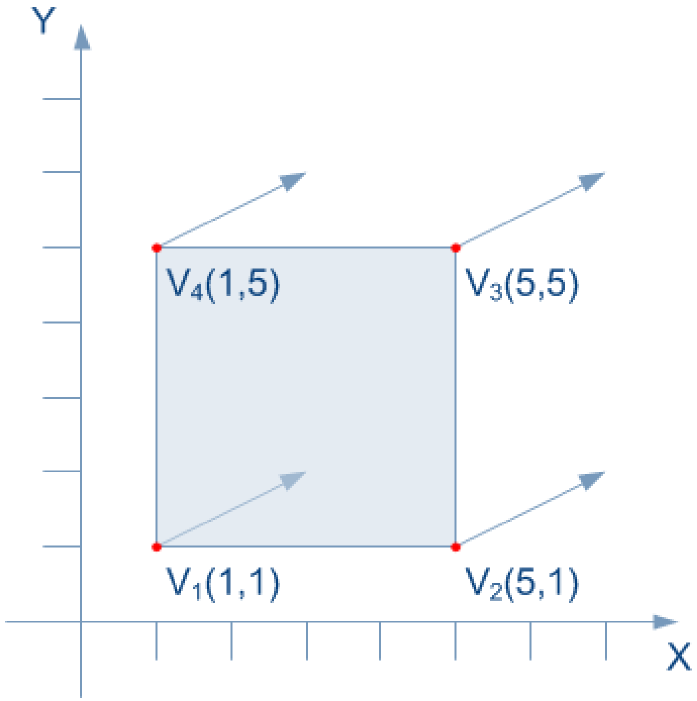
Übung 2

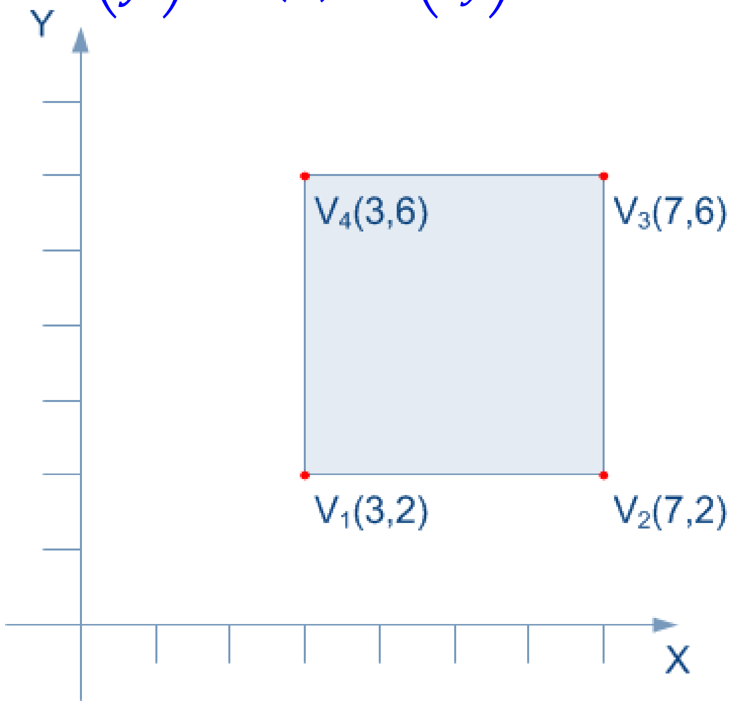
- Bestimmen Sie folgende Transformationen in 2D
  - Spiegelung an x-Achse
  - Rotation um Winkel α entgegen Uhrzeigersinn
  - Spiegelung an Gerade $y = \sqrt{3} \cdot x = \tan(60°) \cdot x$
    - Setzen Sie Gesamttransformation aus mehreren, nacheinander anzuwendenden Transformationen zusammen
    - Zur Erinnerung: m = dy / dx = (y2 – y1) / (x2 – x1) = tan(α)
  - Geben Sie je die Transformationsmatrix **M** an
- Hilfreiche Mathe-Regeln
  - sin(-x) = -sin(x), cos(-x) = cos(x), sin(60°) = $\frac{\sqrt{3}}{2}$, cos(60°) = $\frac{1}{2}$

Translation

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

- Verschiebung aller Eckpunkte um Richtungsvektor ***t***
- Berechnung über komponentenweise Addition von ***t***
  - Für alle Eckpunkte: $\vec{v}' = \vec{v} + \vec{t}$ bzw. $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$





Translation verändert Nullvektor!

(Ursprung verschoben)

2D-Transformationen

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

- Translation:

Nicht als 2x2-Matrix darstellbar

$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} x_{alt} + t_x \\ y_{alt} + t_y \end{pmatrix}$$

Lineare Abbildungen

- Rotation um α:

$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} \cos\alpha x_{alt} - \sin\alpha y_{alt} \\ \sin\alpha x_{alt} + \cos\alpha y_{alt} \end{pmatrix}$$

- Uniforme Skalierung:

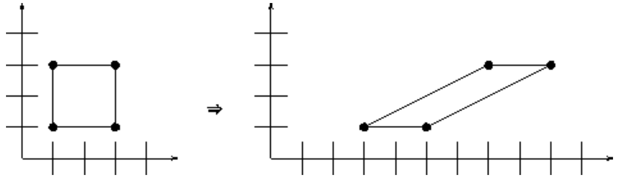
$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = s \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} s x_{alt} \\ s y_{alt} \end{pmatrix}$$

- Allgemeine Skalierung:

$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} s_x x_{alt} \\ s_y y_{alt} \end{pmatrix}$$

- Scherung (entlang x):

$$\begin{pmatrix} x_{neu} \\ y_{neu} \end{pmatrix} = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \end{pmatrix} = \begin{pmatrix} x_{alt} + m y_{alt} \\ y_{alt} \end{pmatrix}$$



Affine Beschreibung

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

- Beschreibung von Skalierung, Rotation (und Scherung) über 2×2-Matrix
  - Skalierung, Rotation (und Scherung) belegen die gleichen Koeffizienten der Matrix
- Wie kann nun 2D-Translation beschrieben werden?
  - Addition problematisch: Bei linearen Abbildungen $\mathrm{M}: V \rightarrow W$ wird Nullvektor von V abgebildet auf Nullvektor von Vektorraum W (Nullvektor ist invariant)

$$\vec{v}' = M \cdot \vec{v} + \vec{t}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Es existiert keine 2×2 Matrix mit $T \cdot \vec{v} = \vec{v}'$

Matrix $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ beschreibt lineare Abbildung (Skalierung, Rotation, Scherung)

Vektor $\begin{pmatrix} t_x \\ t_y \end{pmatrix}$ beschreibt Translation; Bild des Nullvektors dadurch kein Nullvektor mehr

Homogene Beschreibung

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

- Affine Beschreibung von Transformationen erfordert unnötigen Rechenaufwand
  - Matrix-Vektor-Multiplikation plus Vektor-Vektor-Addition bedeutet schlechtere Performance bei Kombination mehrerer Transformationen
- Beschreibung der Translation in 2x2-Matrix M nicht möglich
  - Zudem Vektoren und Punkte nicht voneinander unterscheidbar
- Lösung: Erweiterung von Vektoren und Matrizen um eine weitere Dimension

$$\vec{v}' = M \cdot \vec{v} \qquad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

immer 1 in der untere punkt und die translation gilt nur für punkte !

  - Homogene Beschreibung aller Transformationen mittels einer einzigen Matrix möglich
    - Vektor-Vektor-Addition der affinen Beschreibung entfällt ☺
  - Dazu Beschreibung der Vektoren durch sog. homogene Koordinaten
    - Zusätzliche Komponente der "Vektoren" i.d.R. **1** (→ für Punkte im Raum)
    - Achtung: Richtungsvektoren haben als letzte Komponente **0** (statt 1)

Homogene Koordinaten

- Punkt im $\mathbb{R}^2$ repräsentiert als homogener Vektor im $\mathbb{R}^3$
  - D.h., 2D → 3D (und analog 3D → 4D)
  - $P_{\Re^2} = \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow P_{\Re^3} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \wedge \quad P_{\Re^3} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow P_{\Re^4} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}_w$
- Warum homogene Koordinaten?
  - Alle Elementartransformationen (Rotation, Skalierung, Translation) können somit gleichartig behandelt werden: vereinfacht Implementierung ☺
    - Orts- und Richtungsvektoren können gleichartig behandelt, aber unterschieden werden
  - Komplexe Transformationen durch Konkatenation von Elementartransformationen
    - Zusammenfassen von Transformationen via Matrixmultiplikation: M = A · B
    - Multiplikationsreihenfolge entsprechend Reihenfolge anzuwendender Transformationen

Translation in homogenen Koordinaten

- Affine Beschreibung

$$p' = M \quad p + t$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

- Homogene Beschreibung

$$p' = M \quad p$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Lineare Abbildung
- Skalierung
- Rotation

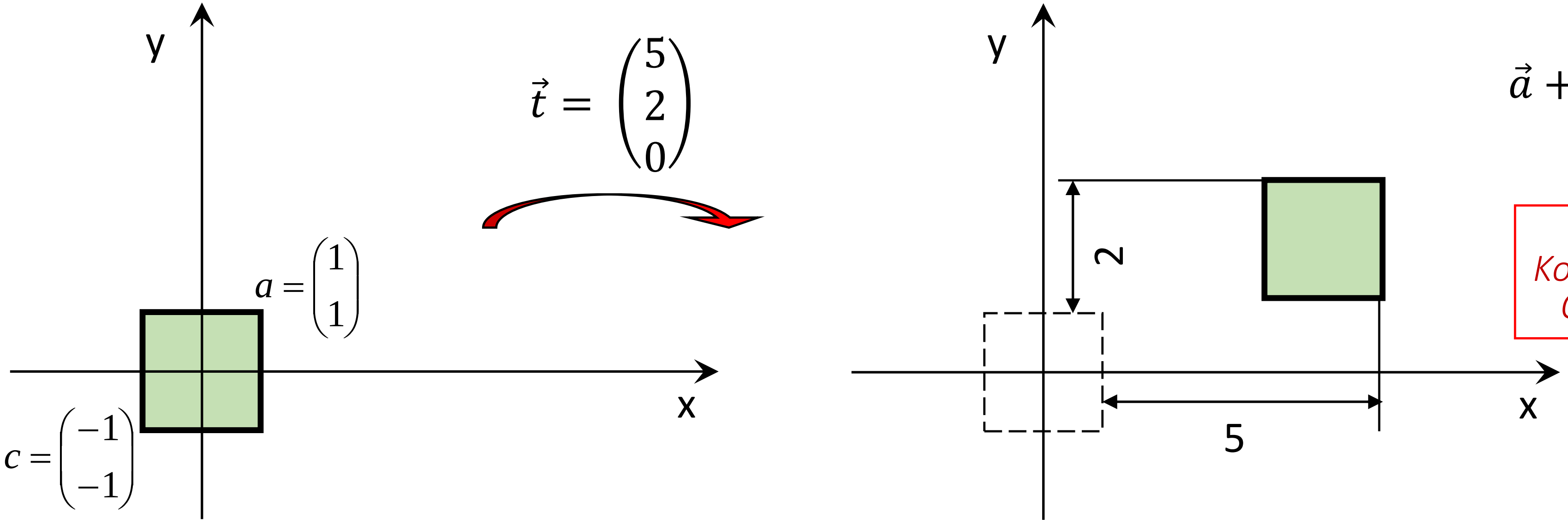
Translation

- Reine 2D-Translation:

$$\begin{pmatrix} x_{neu} \\ y_{neu} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{alt} \\ y_{alt} \\ 1 \end{pmatrix} = \begin{pmatrix} x_{alt} + t_x \\ y_{alt} + t_y \\ 1 \end{pmatrix}$$

Übung 3

- Ein Quadrat soll um $\begin{pmatrix} 5 \\ 2 \end{pmatrix}$ verschoben werden



$$\vec{a} + \vec{t} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 5 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \\ 1 \end{pmatrix} = \vec{a'}$$

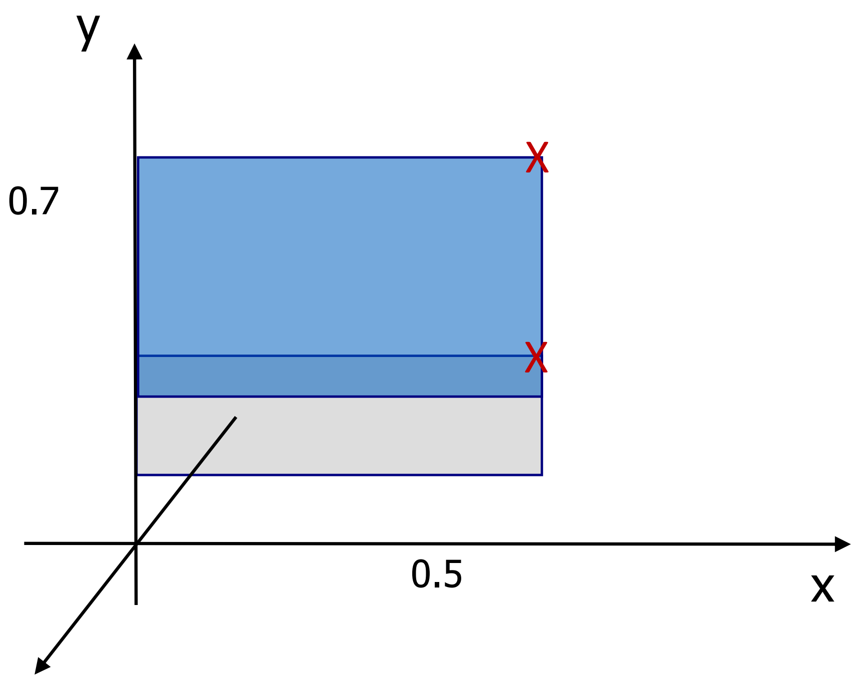
Homogene Koordinate 1 bei Ortsvektoren

Homogene Koordinate 0 bei Richtungsvektoren (neutrales Element der Addition ist 0)

1 0 5
M=0 1 2
0 0 1

- Wie lauten die transformierten Punkte *a'* und *c'*?
- Wie sieht die Matrix *M* aus, um die Punkte *a* und *c* zu transformieren?

Weiteres Beispiel Skalierung



Skalierung um Faktor 2 um y-Achse

- Beispielhaft für obere rechte Ecke bei P(0.6, 0.4)

$$M \cdot \vec{p} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.6 \\ 0.4 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.8 \\ 1 \end{bmatrix} = \vec{p'}$$

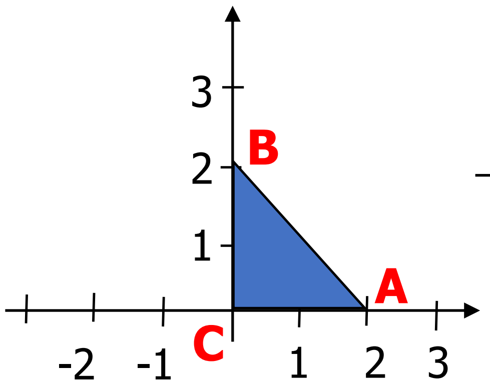
Neutrales Element der Multiplikation 1

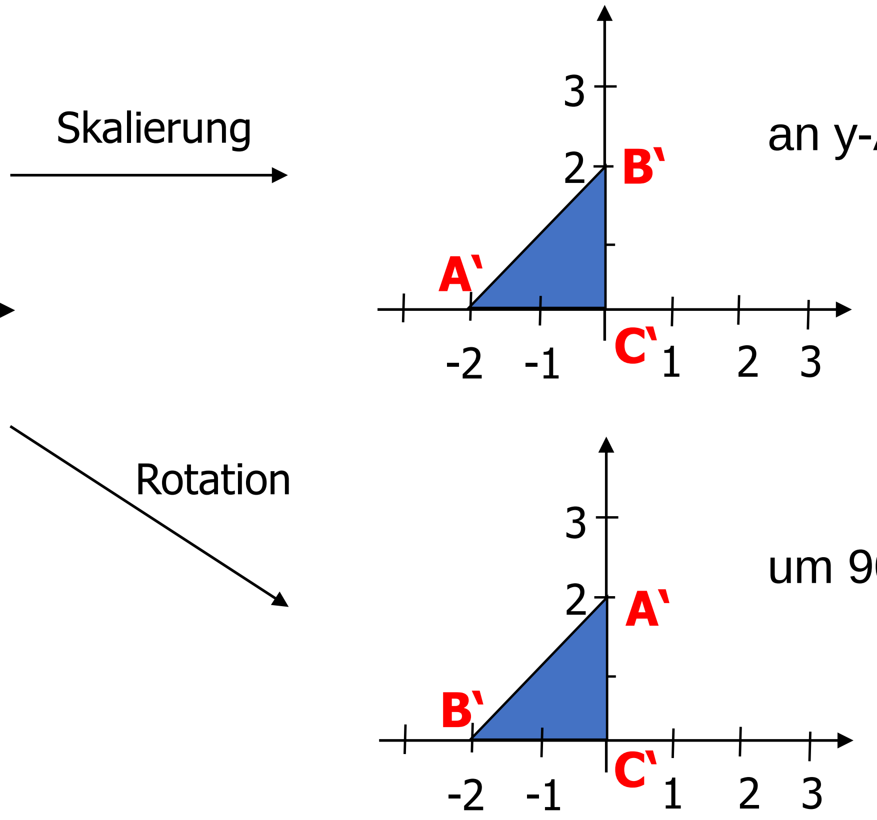
Homogene Koordinate 1 bei Punkten/Ortsvektoren

- Das Objekt wird nicht nur in *y*-Richtung skaliert, sondern verändert hier auch seine Position
  - Abstand zum Nullpunkt wird mit skaliert
  - Skalierung ist nur gegenüber Nullpunkt invariant
    - Gilt analog auch für die Rotation
    - Drehung und Skalierung immer relativ zum Ursprung!

Eindeutigkeit von Transformationen

- Welche Transformation wird gesucht?





an y-Achse gespiegelt

um 90° um Ursprung gedreht

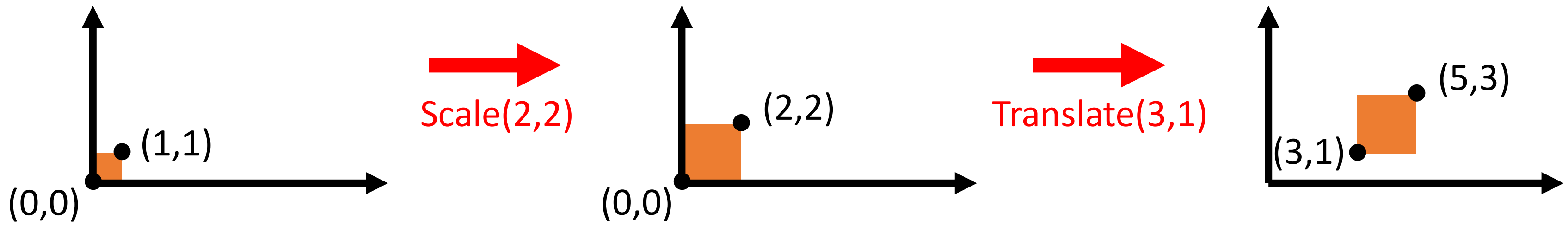
  - Ohne Beschriftung der Eckpunkte gibt es hier mehrere Lösungen

Übung 4

- Entwickeln Sie eine Abbildungsvorschrift, mit der das Quadrat
  Q = ((-1,-1), (1,-1), (1,1), (-1,1)) übergeht in das Rechteck
  R = ((2,4), (4,2), (8,6), (6,8))
  - Tipp: Tragen Sie zur Anschauung Q und R zeichnerisch in ein Koordinatensystem ein
  - Welche Einzeltransformationen sind nötig?
  - Wie lautet die Gesamttransformation M?
  - Wie ändert sich Nullvektor? Zur Erinnerung (Berechnung Mittelpunkt einer Box):
    $\overrightarrow{mid} = \overrightarrow{min} + {}^{1}/_{2}\,(\overrightarrow{max} - \overrightarrow{min}) = {}^{1}/_{2}\,(\overrightarrow{min} + \overrightarrow{max})$
  - Auch hilfreich: $\sin(45°) = \frac{1}{\sqrt{2}}$, $\cos(45°) = \frac{1}{\sqrt{2}}$

Transformationen kombinieren

Bsp.: Skalierung und Translation



Matrixmult.: $p' = T\,(\,S\,p\,) = (\,T\,S\,)\,p = M\,p$

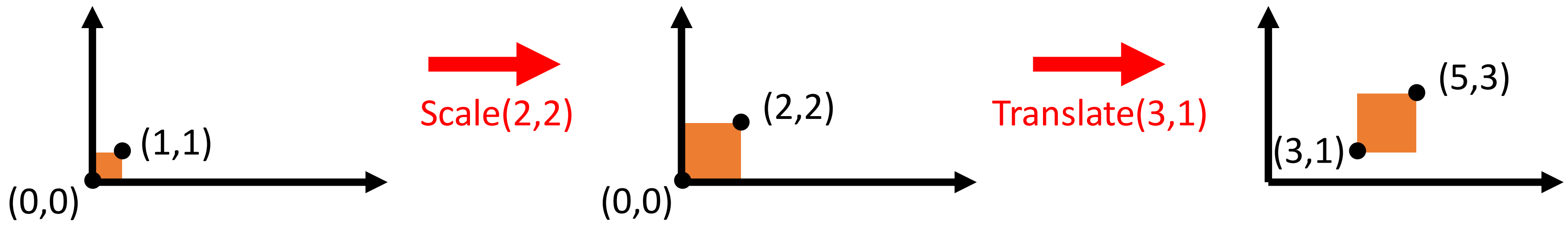
$$T\,S = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

translation ist immer zum schluss

Achtung: Matrixmultiplikation nicht kommutativ

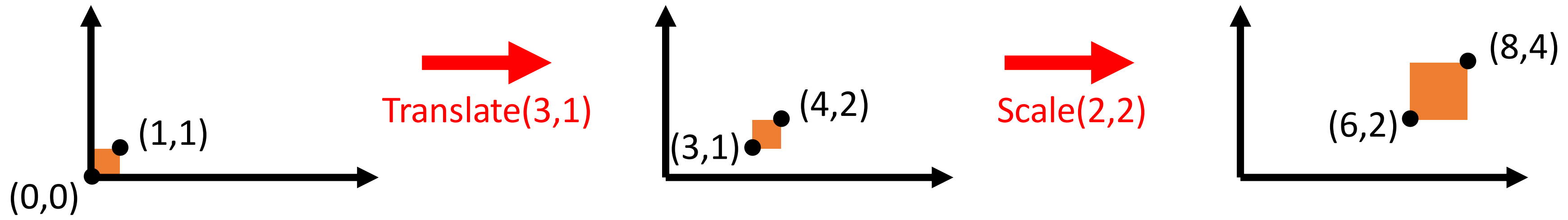
Nicht kommutative Kombination

Skalierung zuerst: p' = T (S p) = (T S) p



assoziativ

Translation zuerst: p' = S (T p) = (S T) p



Nicht kommutative Kombination

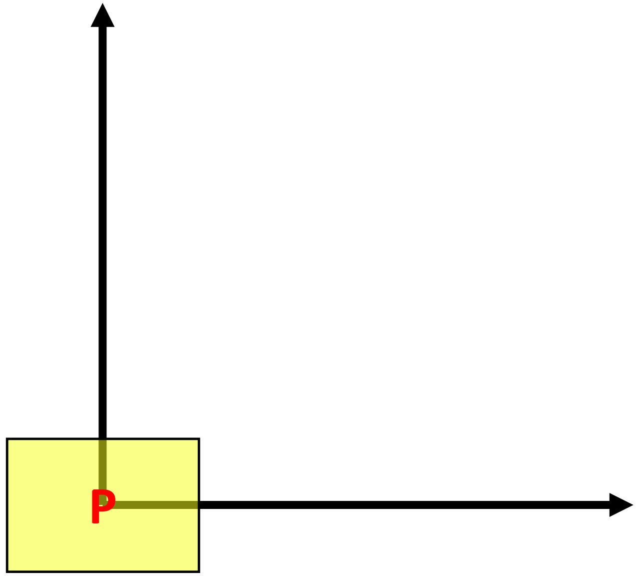
Skalierung zuerst: p' = T (S p) = T S p

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Translation zuerst: p' = S (T p) = S T p

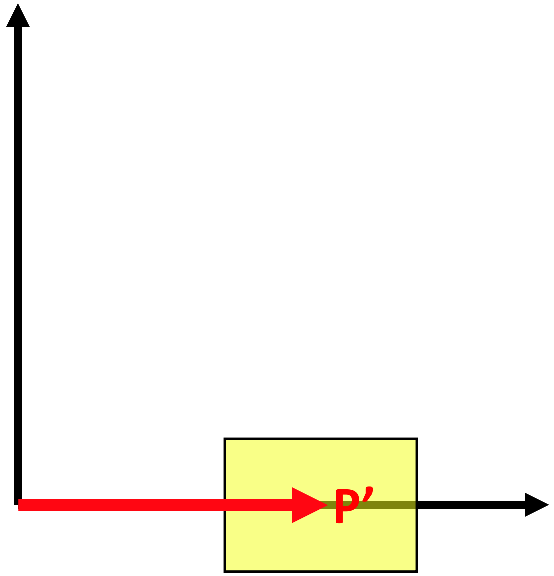
$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Transformationsreihenfolge



Gelbes Rechteck soll verschoben und gedreht werden

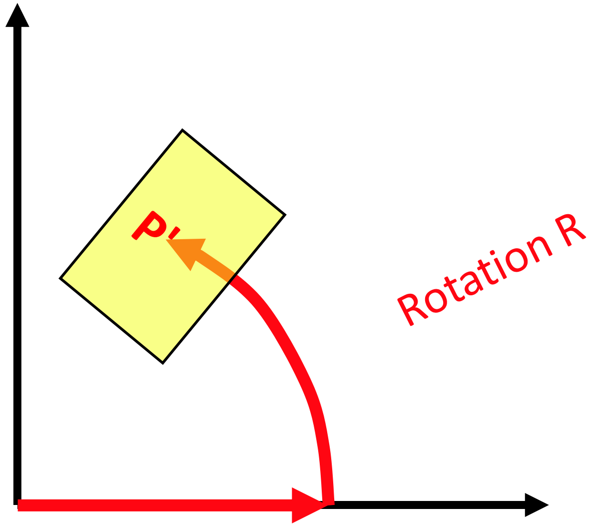
Transformationsreihenfolge



Translation T

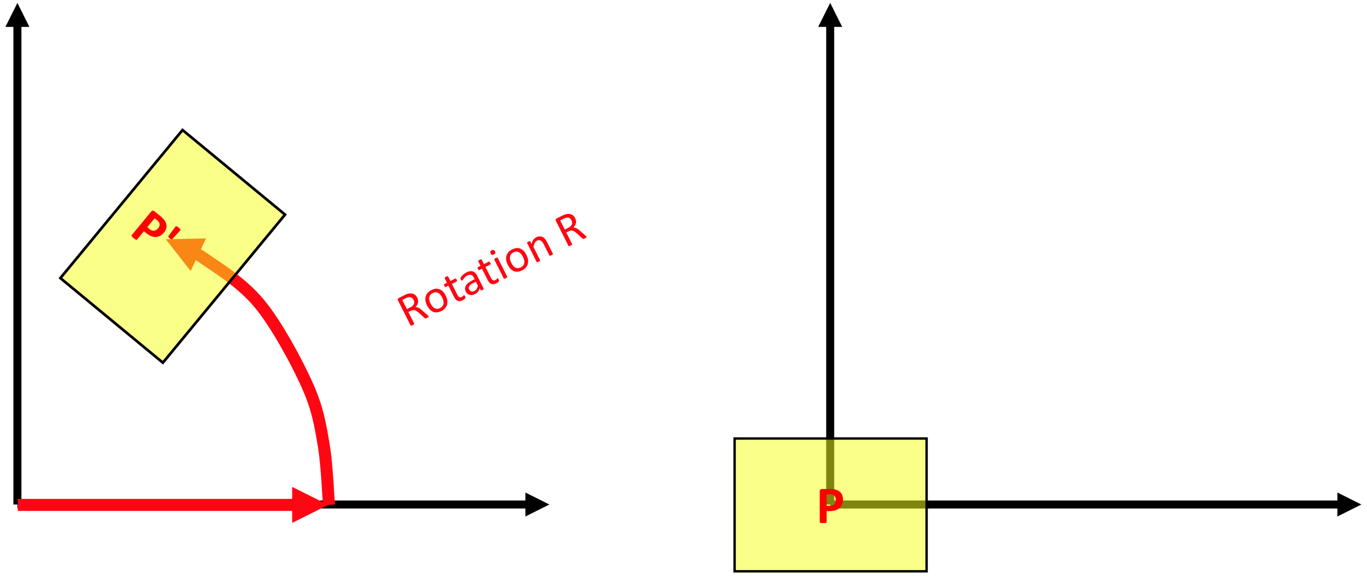
p' = T · p

Transformationsreihenfolge



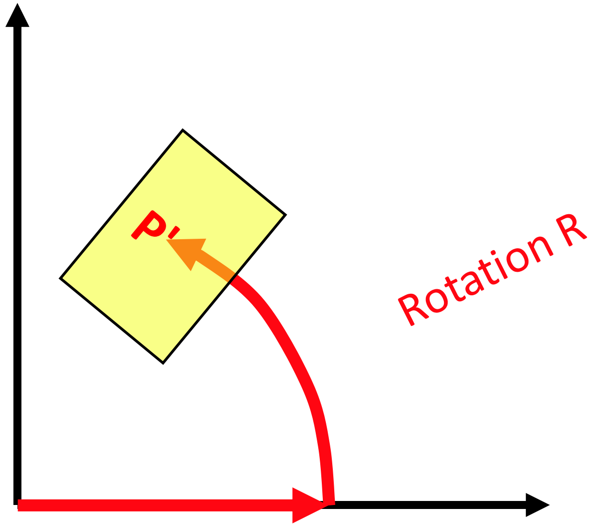
p’ = R · T · p

Transformationsreihenfolge

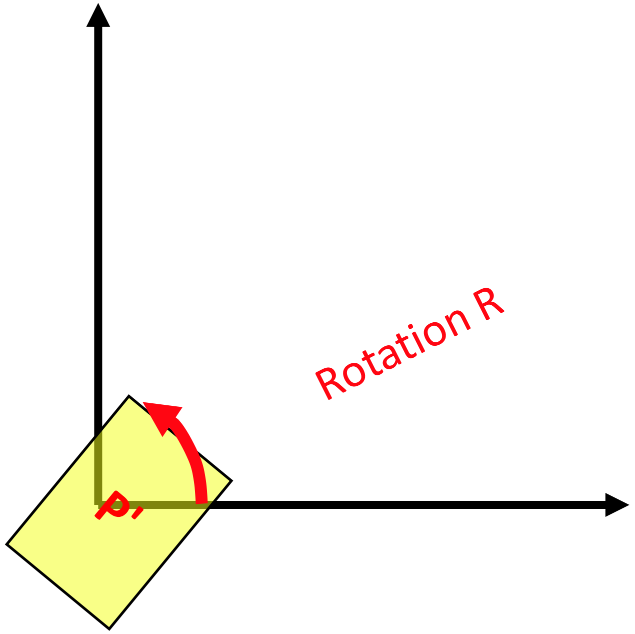


p’ = R · T · p

Transformationsreihenfolge

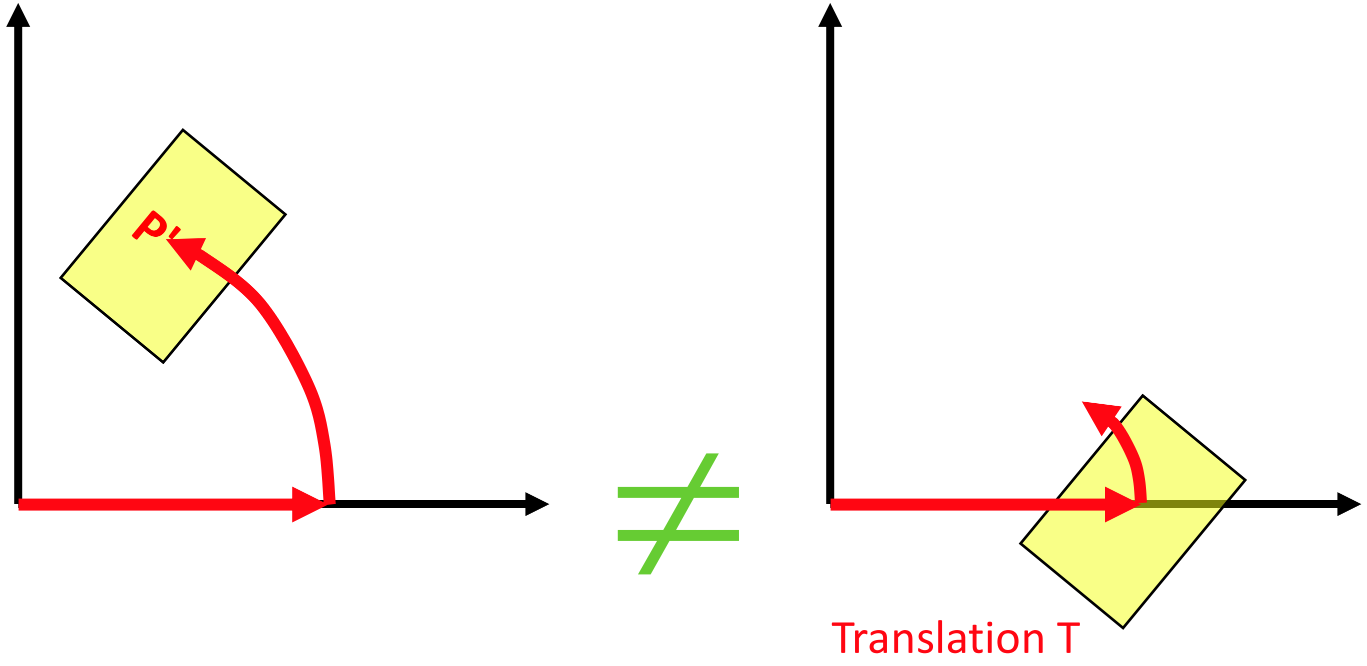


$p' = R \cdot T \cdot p$



$p'' = R \cdot p$

Transformationsreihenfolge



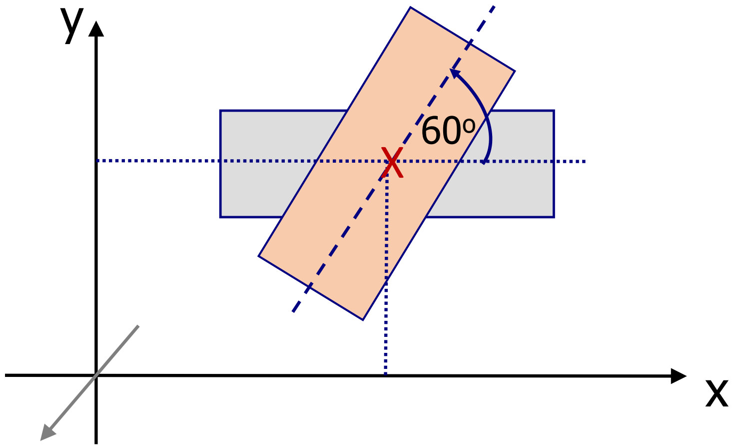
$p' = R \cdot T \cdot p$

$p'' = T \cdot R \cdot p$

Komplexe Transformationen

- Welche Transformationen sind nötig, um Rechteck, wie vorgegeben, zu drehen?

Reihenfolge der Transformationen ist entscheidend

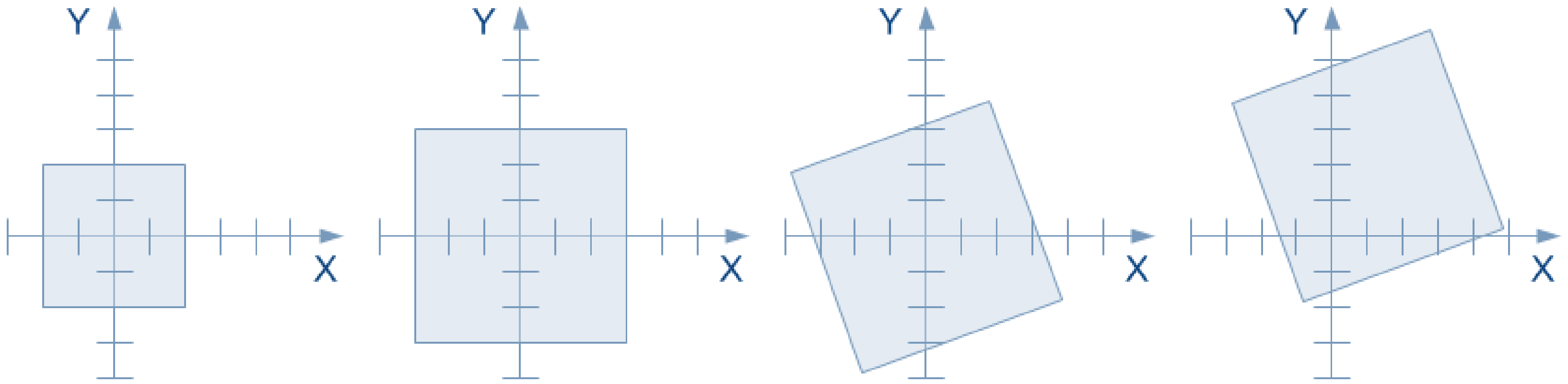


Drehen u. Skalieren mit Referenzpunkt ist dreistufiger Algorithmus:

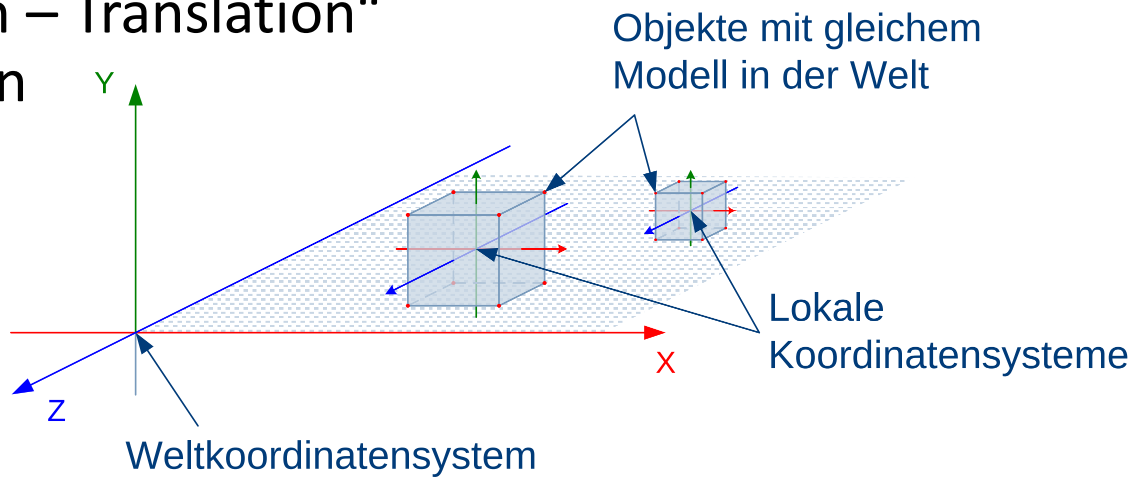
1. Verschieben von Referenzpunkt in Ursprung
2. Rotation um 60° (um z-Achse)
3. Zurückschieben von Referenzpunkt auf ursprüngliche Position

- Verkettung von Transformationen notwendig
  - Aber Anwendung einer Transformation nach der anderen auf jeden Eckpunkt dauert lange und führt zu Rundungsfehlern
  - Daher Einzeltransformationen zu akkumulierter Transformationsmatrix zusammenfassen und nur diese auf alle ursprünglichen Eckpunkte anwenden

Transformationsreihenfolge



- Reihenfolge „Skalierung – Rotation – Translation“ erlaubt intuitive Positionierung von Objekten in 2D- oder 3D-Szene
- Berechnung: $p' = T \cdot R \cdot S \cdot p$
  - Für alle Eckpunkte p



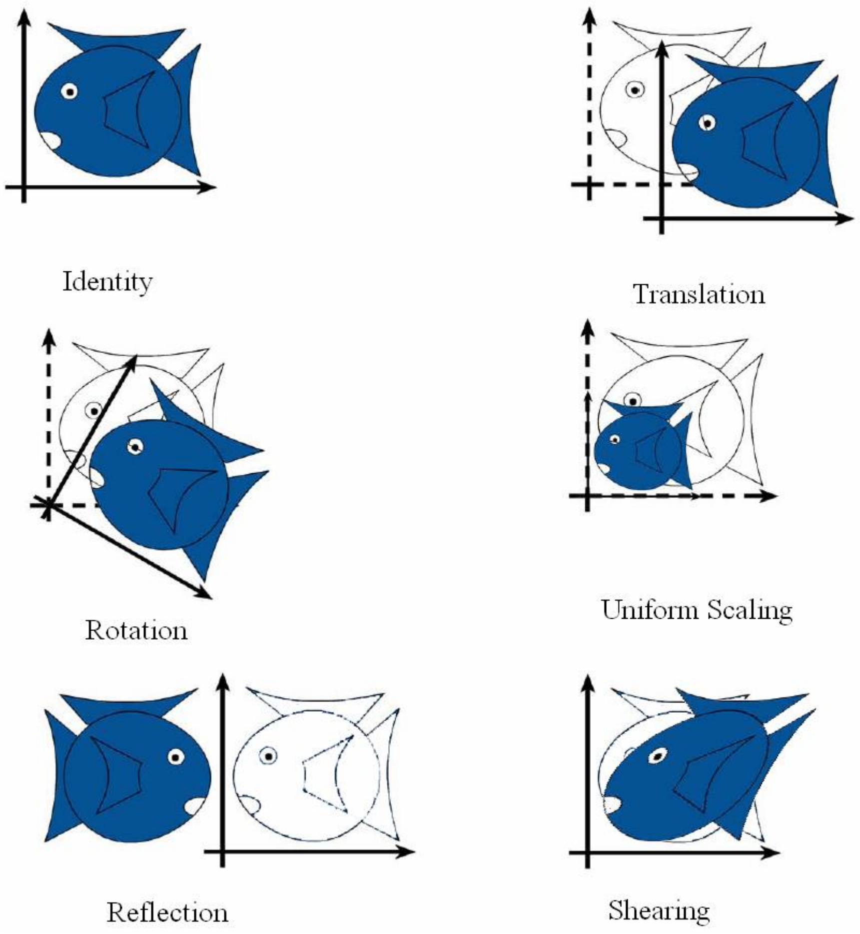
Affine Transformationen in der CG

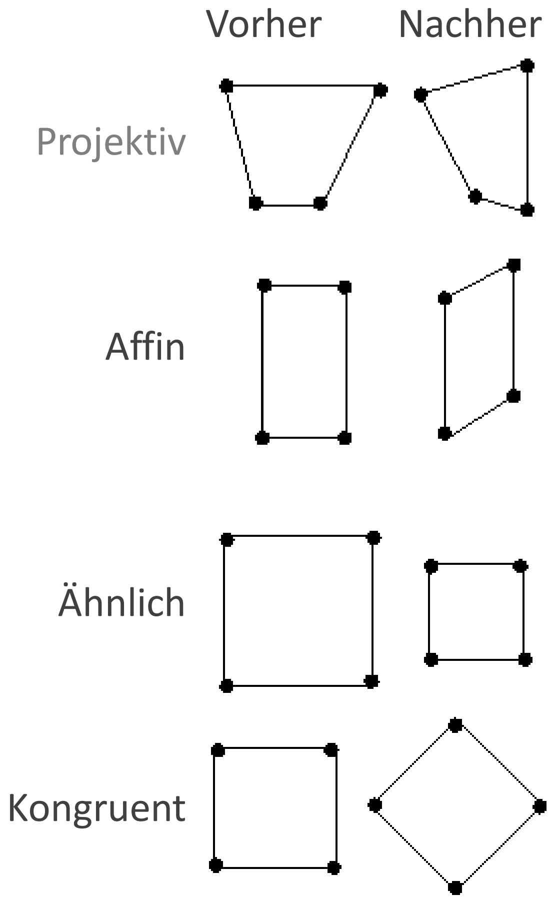
Starrkörper-Transformationen

- Identität
- Translation
- Rotation
- Spiegelung
- Skalierung
  - Uniform
  - Nicht-uniform
- Scherung
  - → selten genutzt

Ähnlichkeitsabbildungen

Affin



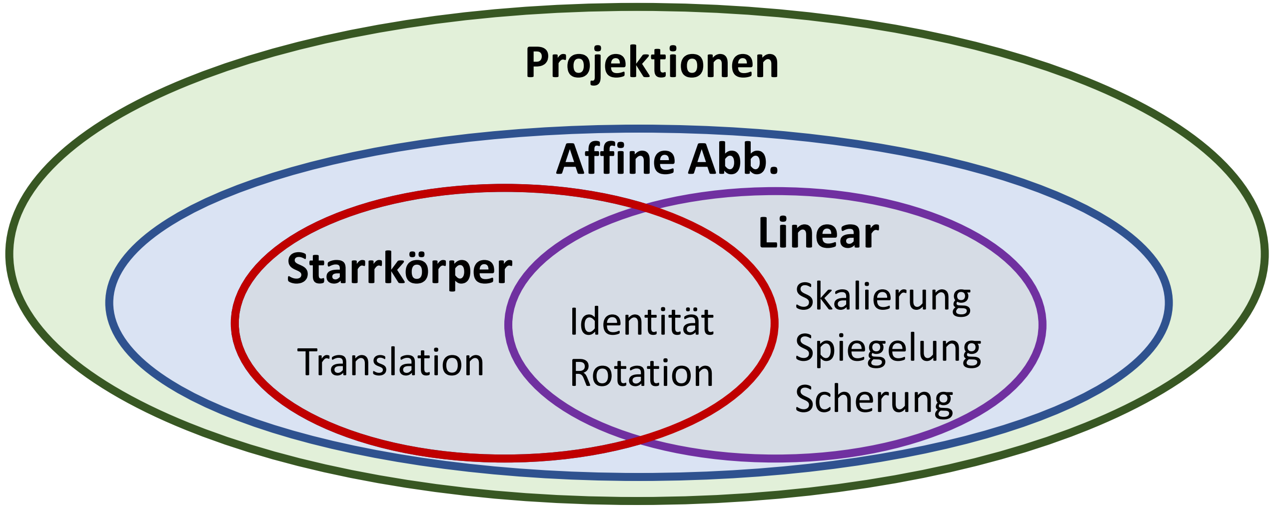


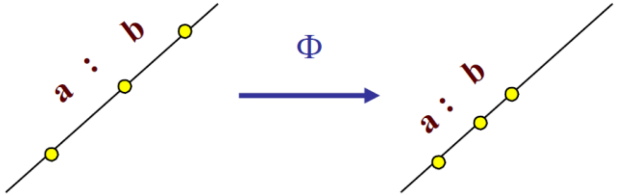
Affine Transformationen

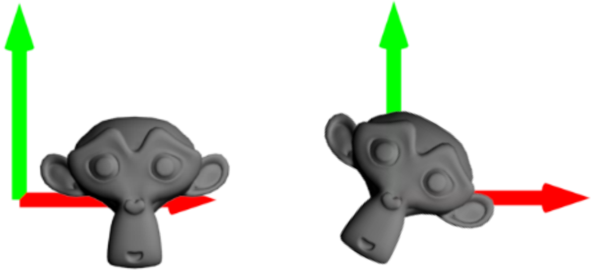
- Abbildung $\Phi: \mathbb{R}^n \rightarrow \mathbb{R}^n$ heißt affine Abbildung (Transformation),
  - …wenn Φ in der Form $\Phi(\vec{v}) = A \cdot \vec{v} + \vec{b}$ darstellbar ist,
  - …wobei A lineare Abbildung ist und $\vec{v}, \vec{b} \in \mathbb{R}^n$
    - A lineare Abb., wenn $\forall \vec{u}, \vec{v} \in \mathbb{R}^n \ \wedge \ \forall \lambda, \mu \in \mathbb{R}$ gilt: $A(\lambda\,\vec{u} + \mu\,\vec{v}) = \lambda\,A(\vec{u}) + \mu\,A(\vec{v})$
- Affine Abbildung besteht aus linearer Abbildung (multiplikativer Teil) und Translation (additiver Teil – ist Parallelverschiebung)
  - Inhomogene 3D-Koordinaten: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}$
  - Kompaktere Darstellung durch homogene Koordinaten

Klassifikation von Transformationen

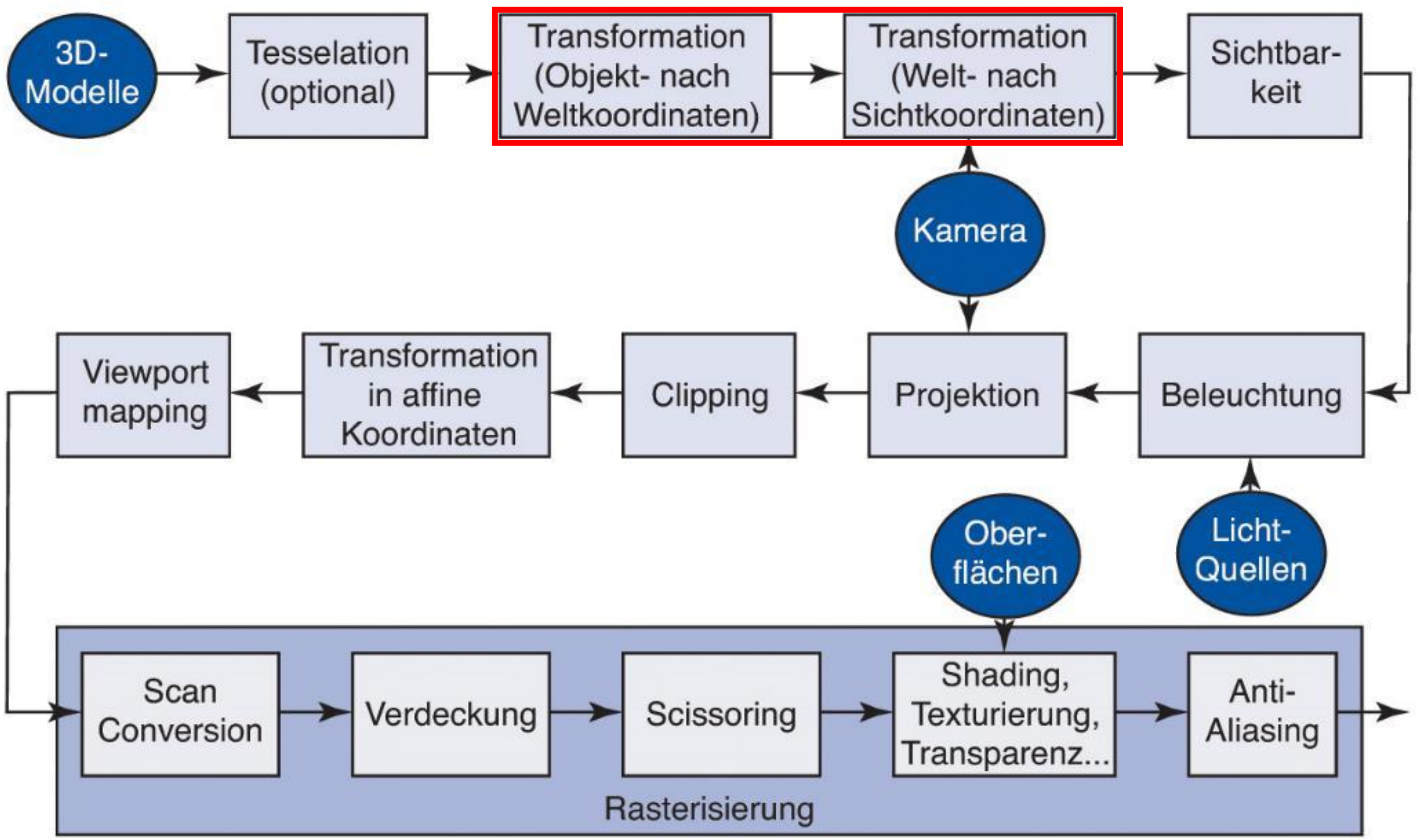
- Transformationen starrer Körper
  - Abstände bleiben erhalten
  - Winkel bleiben erhalten
- Affine Transformationen
  - Geraden bleiben Geraden
  - Parallele Objekte bleiben parallel
  - Längenverhältnisse bleiben erhalten







3D-Transformationen



Affine Transformations in 3D

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

- Transformation can be represented by Matrix **M**
- Transforming a vertex (i.e., a 3d point): p’ = **M** · p

Matrix invertible,
iff det (**M**) ≠ 0

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & k & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

?

Projective
Transformations

Rigid Body
Transformations

rotation translation

uniform scale

Similarity
Transformations

non-uniform scale, shear

Affine
Transformations

Translation in homogenen Koordinaten

- Affine Beschreibung

$$p' = M \quad p + t$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

- Homogene Beschreibung

$$p_h' = M_h \quad p_h$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- Homogene Beschreibung von Vektoren
  - Bei Punkten/Ortsvektoren ist homogene Koordinate $w = 1$ (bei Richtungsvektoren 0)

3D Translation

- We can translate points in space to new positions by adding offsets to their coordinates:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Difference between vector and point:
  - Point has homogenous coordinate $w = 1$
  - Vector has homogenous coordinate $w = 0$
    - No translation of direction vectors!

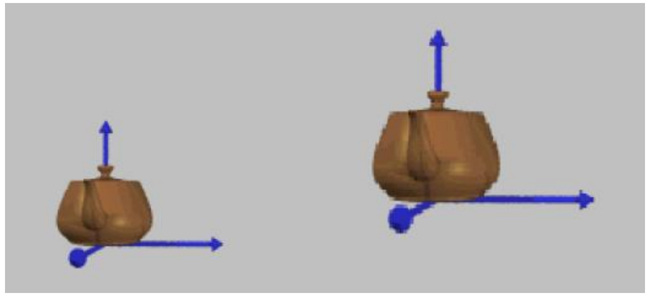
$$\begin{pmatrix} x \\ y \\ z \\ \boldsymbol{w} \end{pmatrix}$$

Homogenen Punkt wieder zurückführen in euklidischen Raum durch Teilen von x, y, z durch w-Koordinate ($w \neq 0$)

3D Scaling

- Objects can be scaled to different sizes
  - If scaling is uniform, the shape is preserved
  - Scaling relative to origin:

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

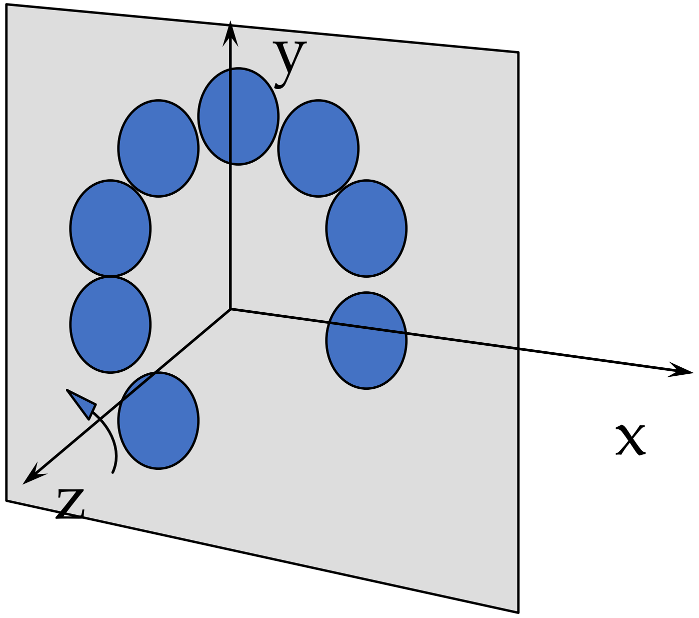


Nicht mit 0 skalieren, denn dann
ist Matrix nicht mehr invertierbar

3D Rotation about z-axis

$$\begin{bmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z does not change



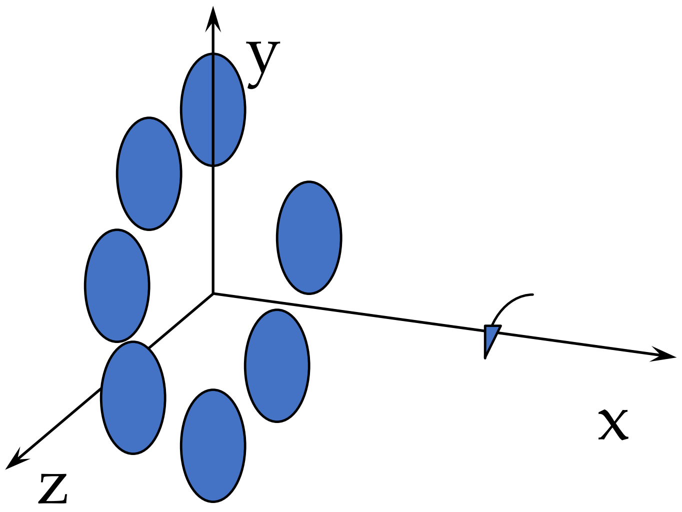
Rotation is assumed positive in a right-hand sense (counterclockwise from x to y)

$$x' = x\cos\varphi - y\sin\varphi$$
$$y' = x\sin\varphi + y\cos\varphi$$

3D Rotation about x-axis

X does not change

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



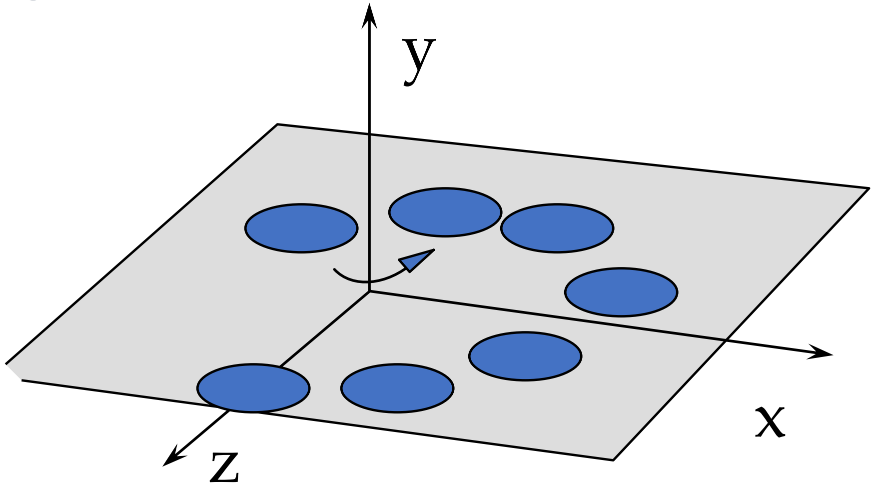
Rotation is assumed positive in a right-hand sense (counterclockwise from y to z)

$$\begin{aligned} y' &= y\cos\varphi - z\sin\varphi \\ z' &= y\sin\varphi + z\cos\varphi \end{aligned}$$

3D Rotation about y-axis

y does not change

$$\begin{bmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

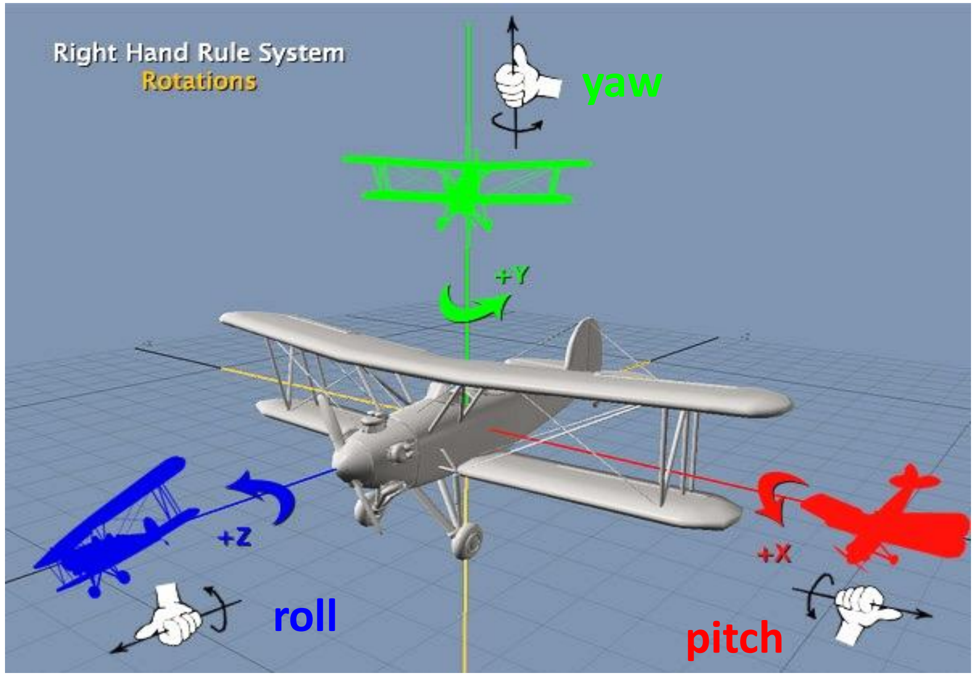


Note: signs of sine terms reversed to maintain positive right-hand rule convention (counterclockwise from z to x; with original matrix written for x to z)

$$z' = z\cos\varphi - x\sin\varphi$$
$$x' = z\sin\varphi + x\cos\varphi$$

Elementare 3D-Rotationen

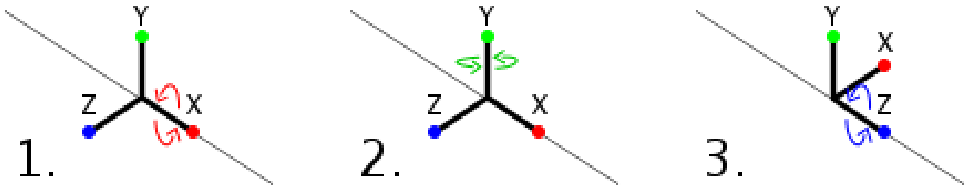
- 3D-Rotationen um Koordinatenachsen
  - X-Achse: dreht nur y und z Koordinaten
  - Y-Achse: dreht nur z und x Koordinaten
  - Z-Achse: dreht nur x und y Koordinaten
- Räumliche Orientierung von 3D-Objekten kann damit auch angegeben werden durch drei Winkel (→ Eulerwinkel)
  - Beschreiben, um wieviel Grad sich Objekt um jeweils x-, y- und z-Achse dreht
  - Multiplikation dreier Matrizen für Gesamtdrehung: $R = R_z(r)\ R_y(y)\ R_x(p)$
  - Sehr anschaulich, hat aber sog. "Gimbal-Lock-Problem"

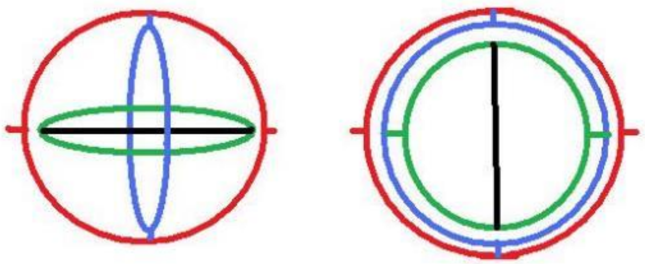


Orientierung geg. durch die Winkel *r*oll, *p*itch u. *y*aw

Ausblick: Matrizen vs. Quaternionen

- Problem bei Rotationen: „Gimbal Lock“
  - Verlust eines Freiheitsgrades (eine Achse rotiert auf andere Achse)





- Lösung: Quaternionen statt Matrizen
  - Haben wie komplexe Zahlen Real- und Imaginärteil

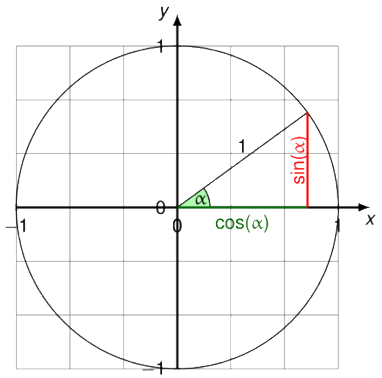
    $\boldsymbol{q} = i\,v_x + j\,v_y + k\,v_z + w = (v_x, v_y, v_z, w) = (\mathbf{v}, w)$
  - Einheitsquaternionen beschreiben Drehungen im $\mathbb{R}^3$
    - Lassen sich darstellen durch $q = \left(\cos\frac{\alpha}{2}, \vec{r}\sin\frac{\alpha}{2}\right)$
    - ***q*** entspricht Rotation um Winkel α mit normierter Drehachse ***r***

Anwendung bei Interaktion und Animation (muss für Rendering in Matrix umgerechnet werden)

Einschub: Basistransformation

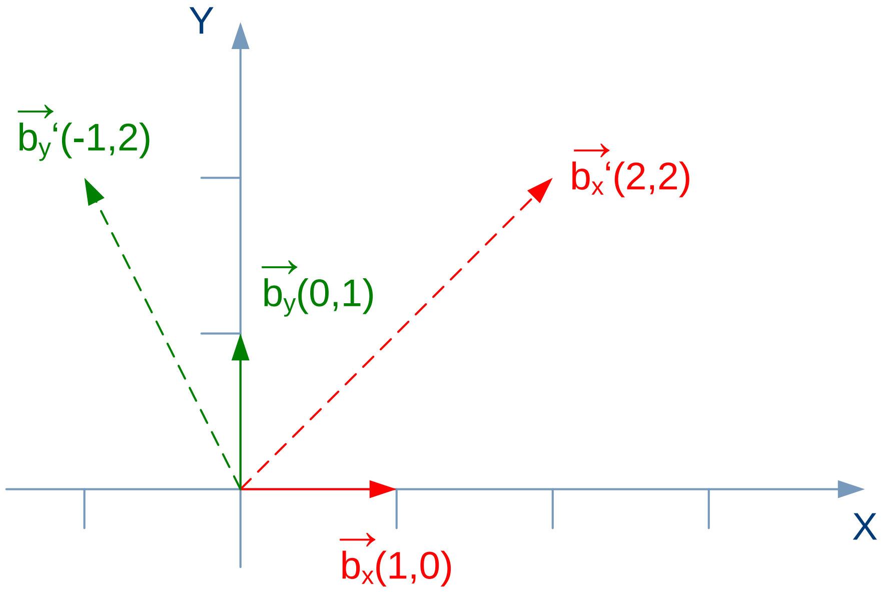
- Multipliziert man Einheitsvektoren $\mathbf{e_1}$, $\mathbf{e_2}$, $\mathbf{e_3}$ des $\mathbb{R}^3$ mit Matrix A, stehen Bilder der Basisvektoren bzgl. der linearen Abbildung A in den Spalten der A beschreibenden Matrix
  - X-Achse: $$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix}$$
  - Y-Achse: $$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}$$
  - Z-Achse: $$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}}_{A}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{13} \\ a_{23} \\ a_{33} \end{bmatrix}$$

Neue Spaltenvektoren sind Bilder der ursprünglichen Basisvektoren



Einschub: Basistransformation

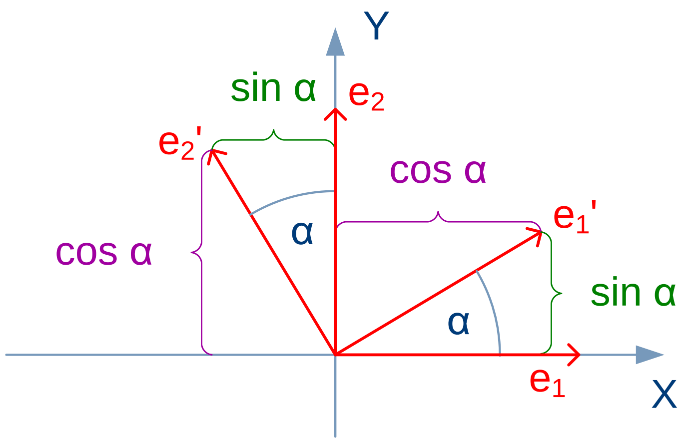
- Beispiel im 2D:
  - Ergibt Transformation der 2D-Basisvektoren $\overrightarrow{b_x} = \begin{pmatrix}1\\0\end{pmatrix}$ u. $\overrightarrow{b_y} = \begin{pmatrix}0\\1\end{pmatrix}$ z.B. die Bilder $\overrightarrow{b_x'} = \begin{pmatrix}2\\2\end{pmatrix}$ und $\overrightarrow{b_y'} = \begin{pmatrix}-1\\2\end{pmatrix}$, dann wird zugehörige lineare Abbildung beschrieben durch Matrix $\mathrm{B} = \begin{pmatrix}2 & -1\\2 & 2\end{pmatrix}$



Herleitung 3D-Rotation

- Übertragen des Beispiels auf Rotation um z-Achse?
  - Basisvektor x-Achse: $\begin{pmatrix} \cos\alpha \\ \sin\alpha \\ 0 \end{pmatrix}$
  - Basisvektor y-Achse: $\begin{pmatrix} -\sin\alpha \\ \cos\alpha \\ 0 \end{pmatrix}$
  - z-Achse wird bei Rotation um sich selbst nicht verändert!
- Eintragen der Bilder der Basisvektoren des $\mathbb{R}^3$ in Spalten der Matrix liefert:

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



Hinweise zum Rechnen mit Matrizen

- Einheitsmatrix E ist neutrales Element der Matrixmultiplikation
- Rotationsmatrizen sind orthonormal:
  wenn R^{-1} Inverse einer 3x3-Matrix R ist, dann ist R^{-1} gleich der Transponierten R^T
  - Transponierte viel schneller zu berechnen als allgemeine Berechnung der Inversen einer Matrix

$$R = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow R^{-1} = R^T = \begin{pmatrix} a_{11} & a_{21} & a_{31} & 0 \\ a_{12} & a_{22} & a_{32} & 0 \\ a_{13} & a_{23} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Inverse der Translationsmatrix T um Vektor ***t*** ist Matrix T^{-1} mit Translation um ***-t***:

$$\mathrm{T}^{-1} = \mathrm{T}_{-t} = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

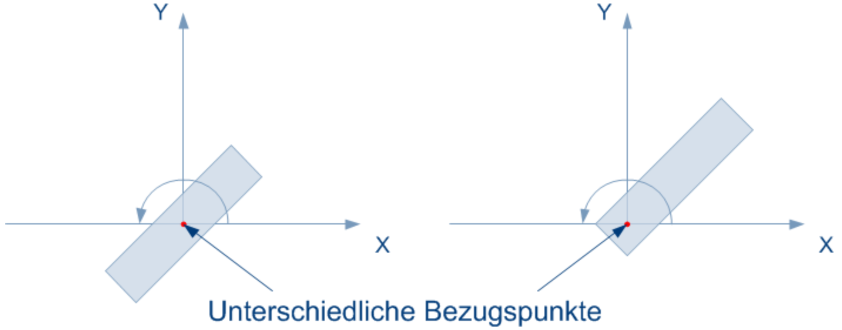
- Inverse bei Multiplikation: $(T \cdot R)^{-1} = R^{-1} \cdot T^{-1} \overset{\text{R orthogonal}}{=} R^T \cdot (T(t))^{-1} = R^T \cdot T(-t)$

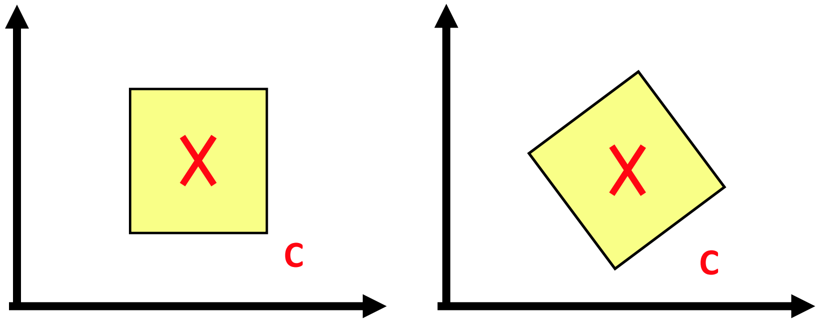
Übung 5

- Gegeben ist Rotation **R** von α=30° um y-Achse und Translation **T** entlang Vektor $\vec{t} = (1, -2,\ 1/2\ )^T$
  - Wie lautet Gesamttransformation **M** = **T** · **R**?
  - Geben Sie erst **R** und **T** an!
    - Hinweis: $\sin 30° = 1/2$ und $\cos 30° = \sqrt{3}/2$
- Sie möchten ein 3D-Objekt erst um den lokalen Ursprung drehen und es dann verschieben
  - Begründen Sie, welche der beiden Matrizen, **N** = **R** · **T** oder **M** = **T** · **R**, Sie nehmen würden, oder ist es egal?
- Geben Sie zu **R** und **T** die inverse Matrix $\mathbf{R}^{-1}$ und $\mathbf{T}^{-1}$ an
  - Wie lautet die Inverse von **M** = **T** · **R**?

Rotation um beliebigen Punkt

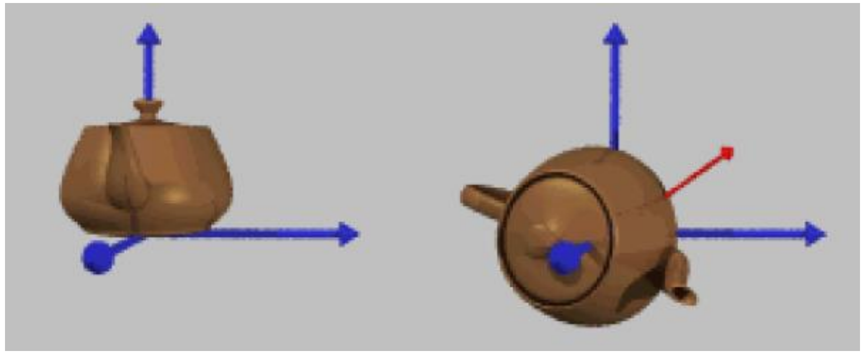
- Auch bei Elementarrotationen ist Ursprung Rotationszentrum
  - Rotationen lassen Ursprung fest (ist invariant bzgl. Rotation)
- Allgemeiner Fall: beliebiges Rotationszentrum (Pivot-Punkt)
  - Transliere Objekt so, dass Pivot-Punkt ***c*** in Ursprung liegt
  - Rotiere Objekt um Ursprung (als neuem Rotationszentrum)
  - Undo der Translation (d.h., transliere alles zurück zu Pivot ***c***)
- Sei **C** Translationsmatrix zu Pivot: $p' = T \cdot \mathbf{C} \cdot \mathbf{R} \cdot S \cdot \mathbf{C}^{-1} \cdot p$

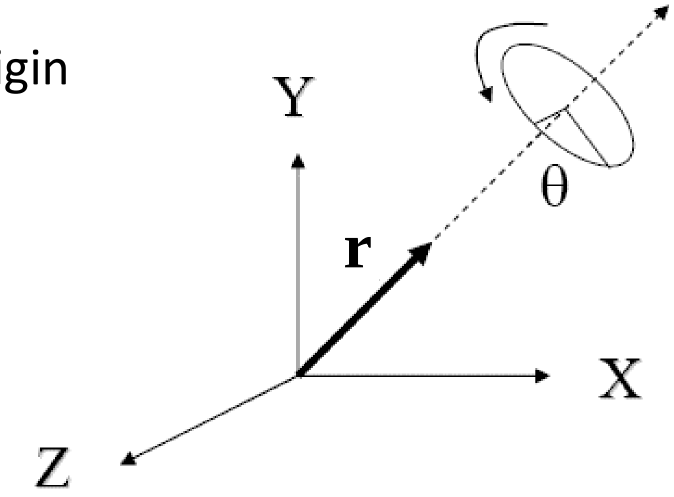




Rotation um beliebige Achse

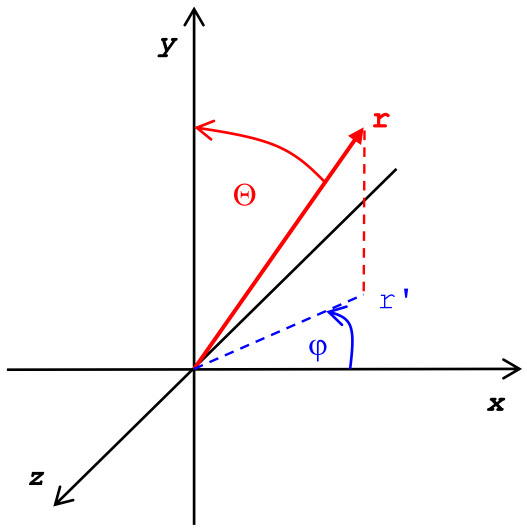
- Rotation about arbitrary axis by multiplying basic rotations
- General 3D rotation often defined by rotation angle and axis
  - Unit vector **r** indicates axis of rotation (points on axis remain unchanged)
    - Scalar θ indicates angle of rotation
  - We assume that rotation axis passes through origin
    - Otherwise, first translate such that **r** is passing through origin

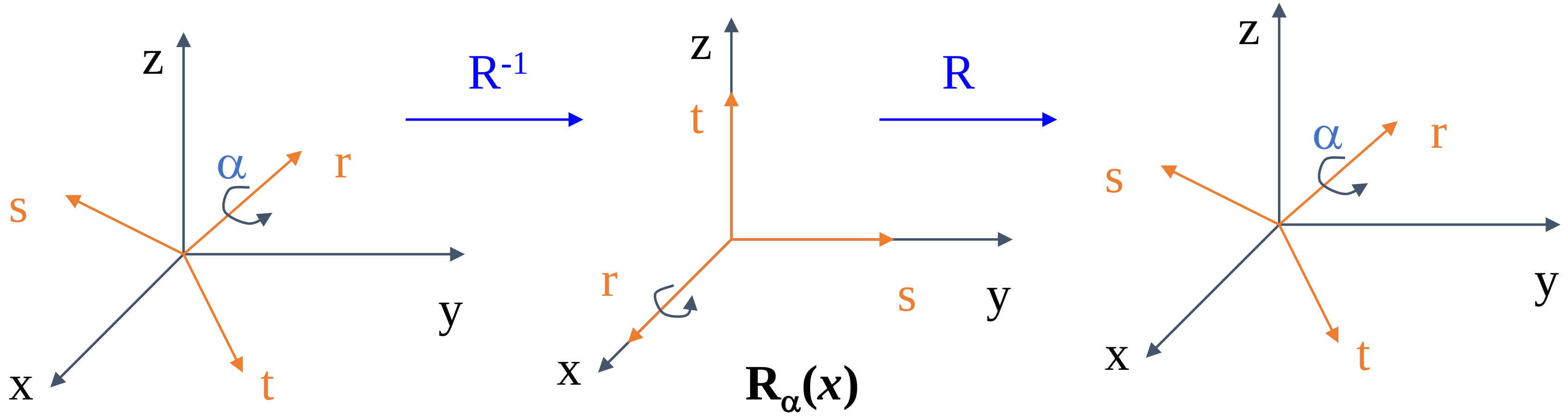




Rotation um beliebige Achse

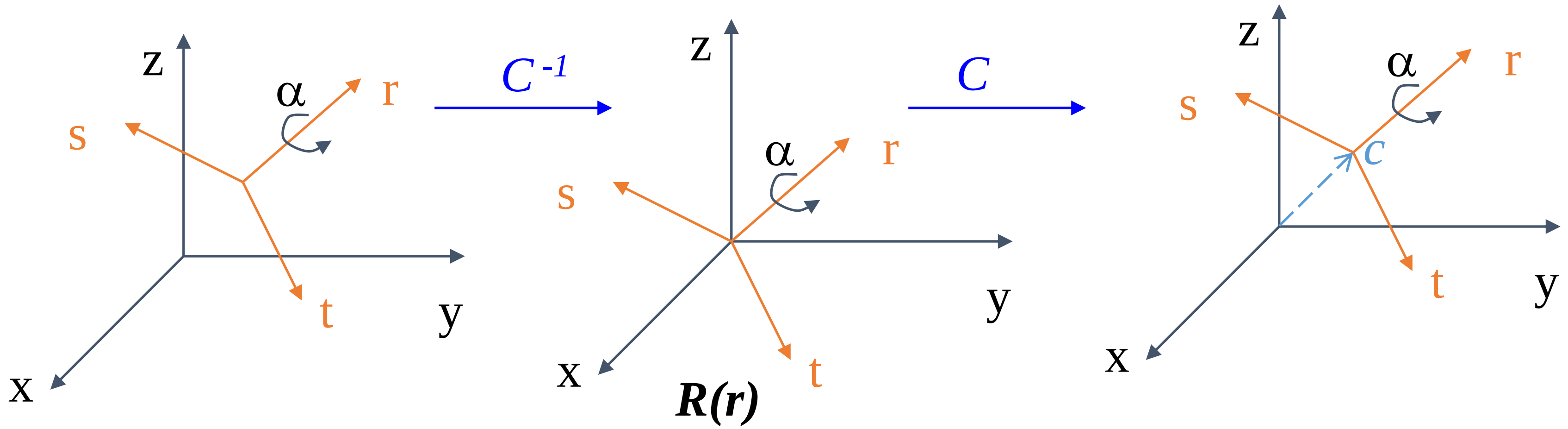
- Drehung $R_\alpha(\mathbf{r})$ um Winkel α um beliebige Achse, gegeben durch normierten Vektor $\mathbf{r} = (x, y, z)^T$
  - Drehachse **r** mit Matrix R^{-1} auf (z-, y- oder) x-Achse drehen
  - Um Winkel α um (z-, y- oder) x-Achse rotieren
  - Von x- (oder z- bzw. y-) Achse mit Matrix R zurück auf **r** drehen
- $p' = R \cdot \mathbf{R}_\alpha(x) \cdot R^{-1} \cdot \mathbf{p} = R_\alpha(\mathbf{r}) \cdot \mathbf{p}$





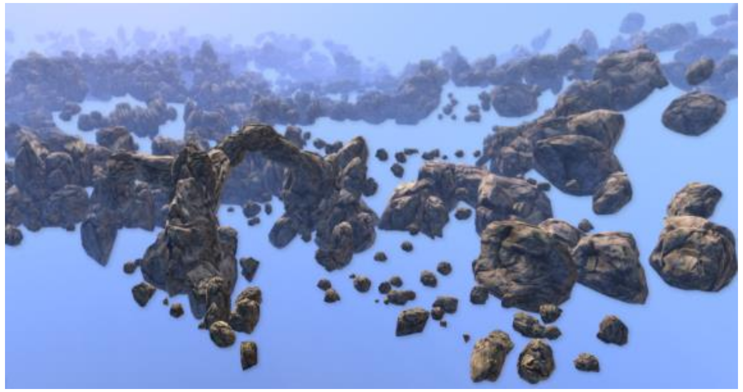
Rotation um beliebige Achse u. Punkt

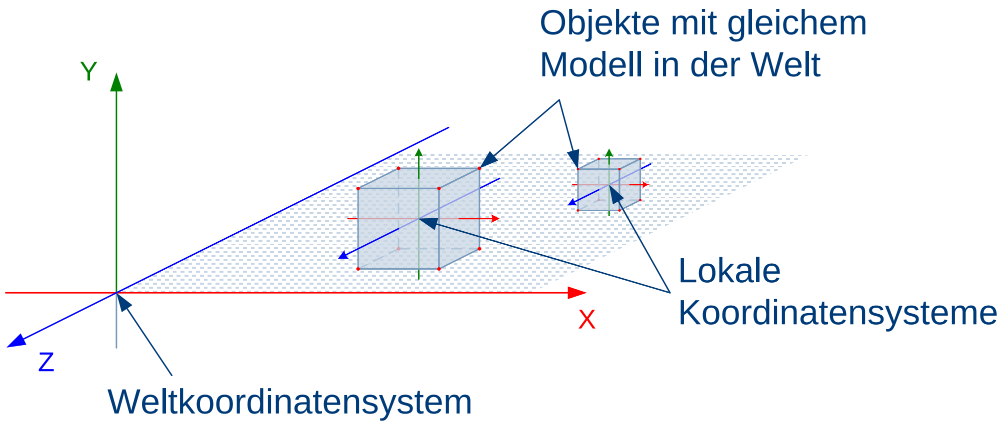
- Rotationsachse ***r*** durch beliebigen Pivot-Punkt
  - Verschiebung des Rotationszentrums ***c*** in Ursprung
  - Anschließende Rotation $R_\alpha(\mathbf{r})$ wie zuvor beschrieben
  - Zurückverschiebung mit Matrix C um Rotationszentrum ***c***
- $\boldsymbol{p}' = C \cdot R_\alpha(r) \cdot C^{-1} \cdot \boldsymbol{p}$



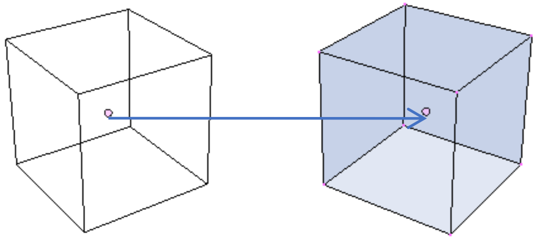
Model (bzw. World) Matrix

- Beschreibung der Transformationen eines 3D-Modells in 3D-Welt durch sog. Model-Matrix *M*
  - Kombinationen von Position, Orientierung und Skalierung
    - Typische Reihenfolge bei Rotationszentrum ***c*** ungleich Nullvektor: $p' = T \cdot \mathbf{C} \cdot \mathbf{R} \cdot S \cdot \mathbf{C}^{-1} \cdot p$
  - Matrix *M* überführt lokale Koordinaten in Weltkoordinaten
    - Berechnung für alle Eckpunkte *p* erfolgt normalerweise auf GPU
- Verschiedene Modell-Instanzen mit unterschiedlichen Transformationen möglich



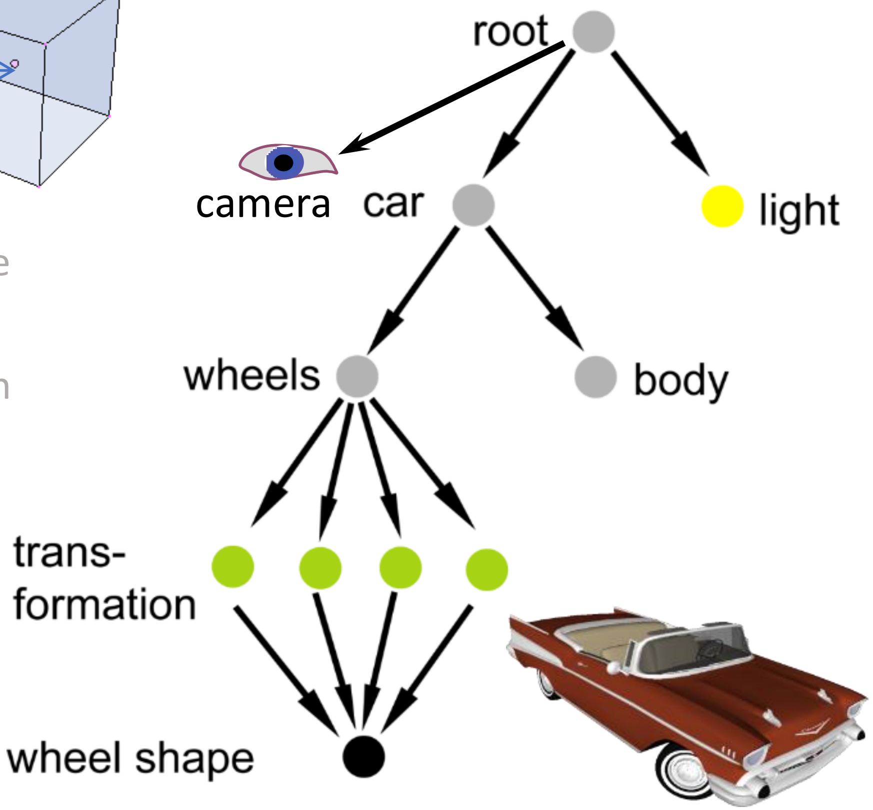


Koordinaten-Transformationen in 3D

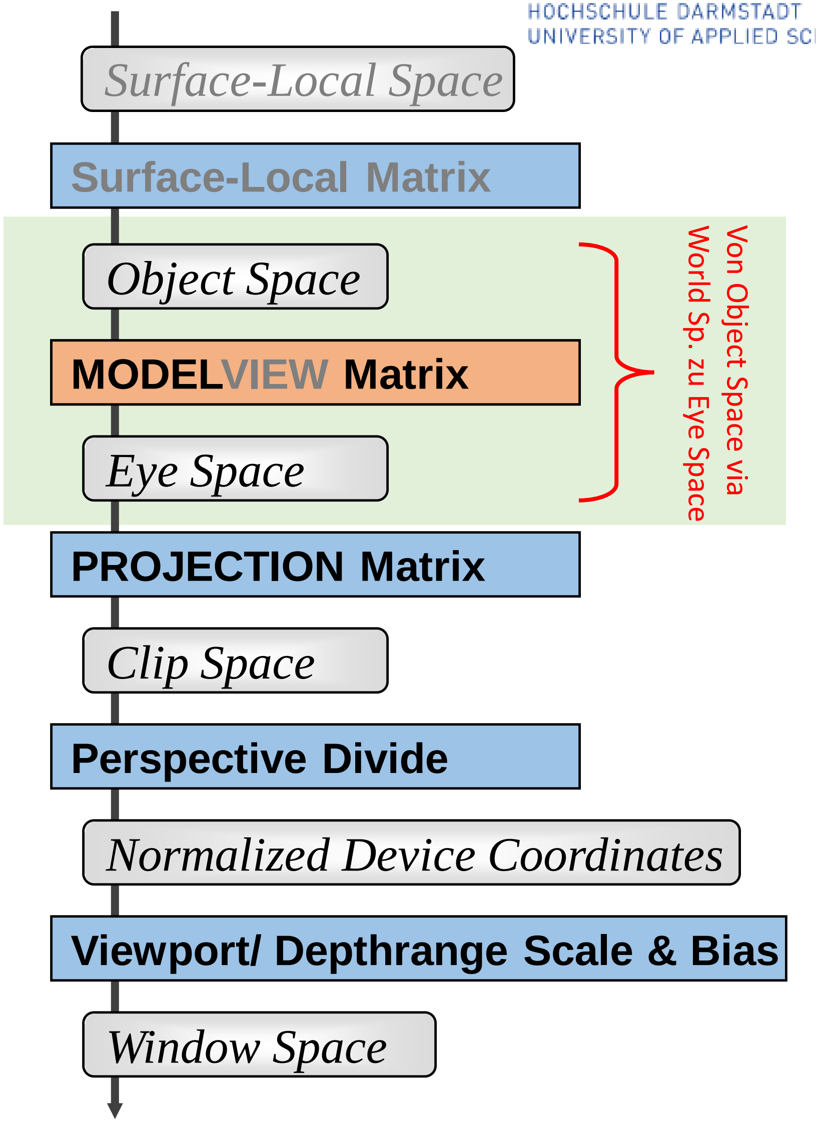


Jetzt: von Object Space zu World Space

Später: Umrechnung in Kamerakoordinaten

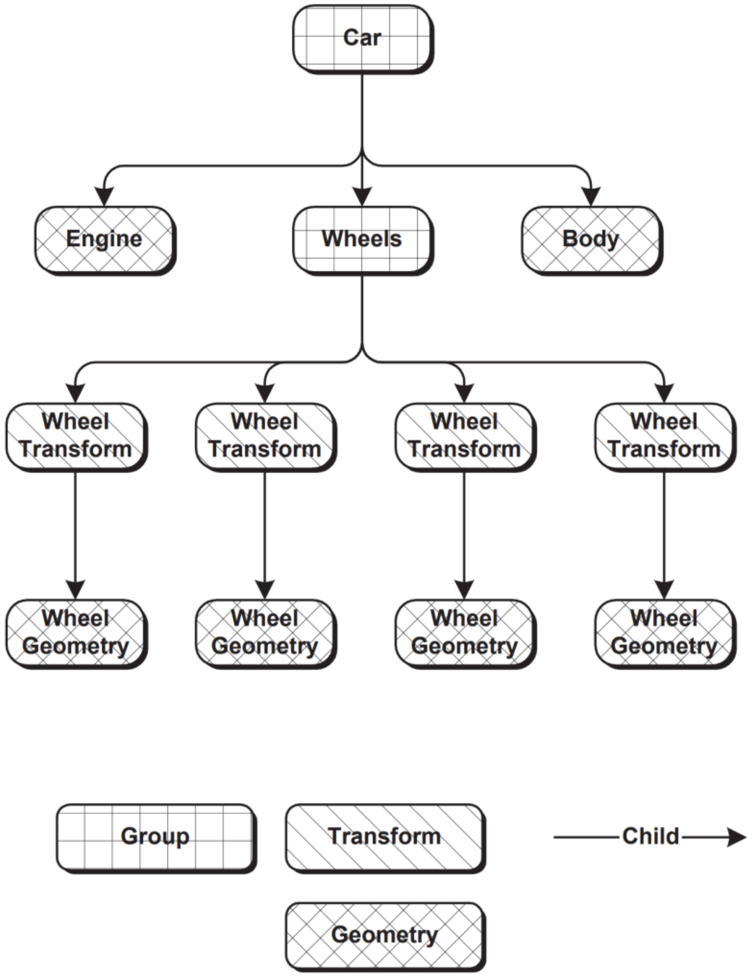


Objekte normalerweise in "lokalem" Ursprung erzeugt

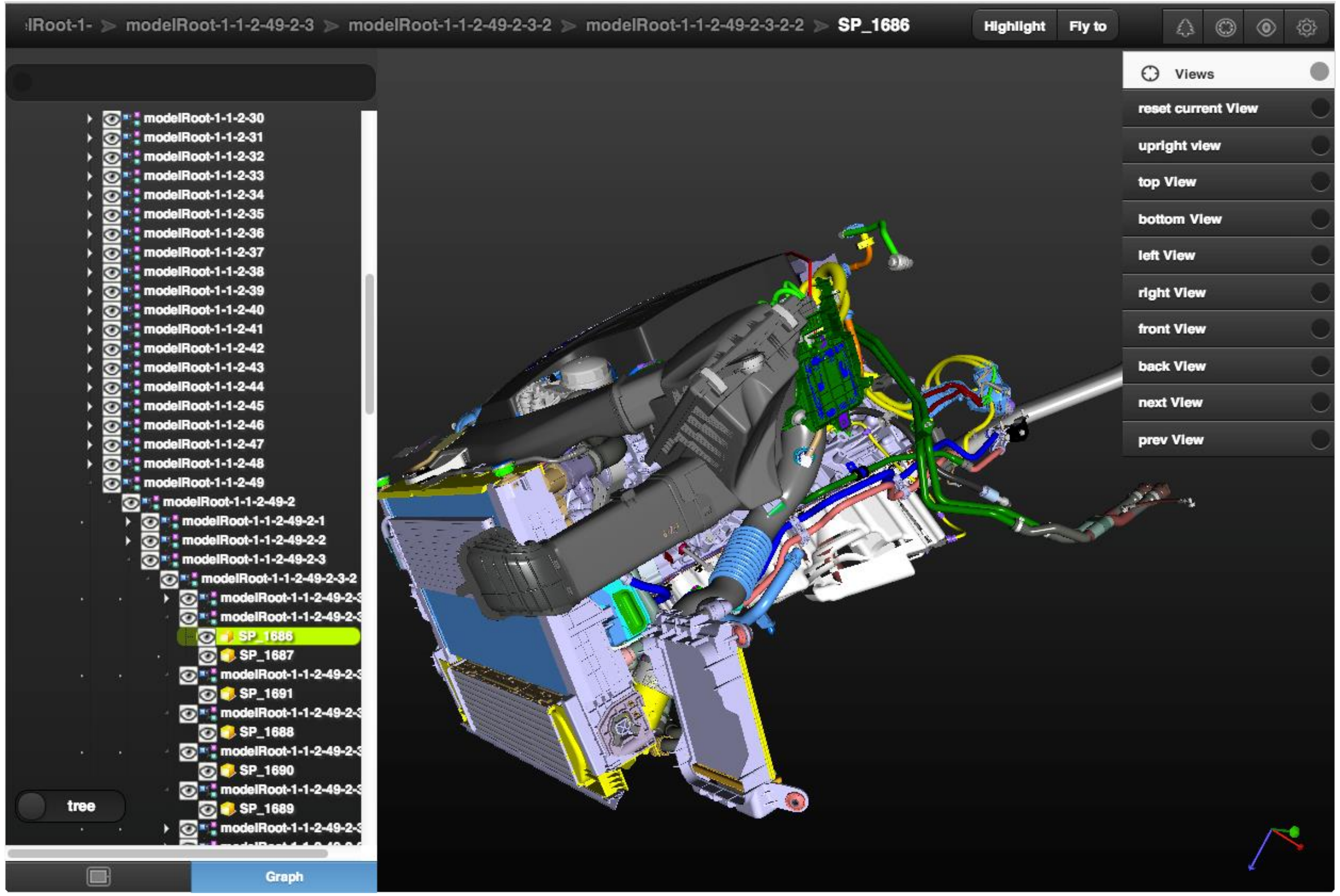


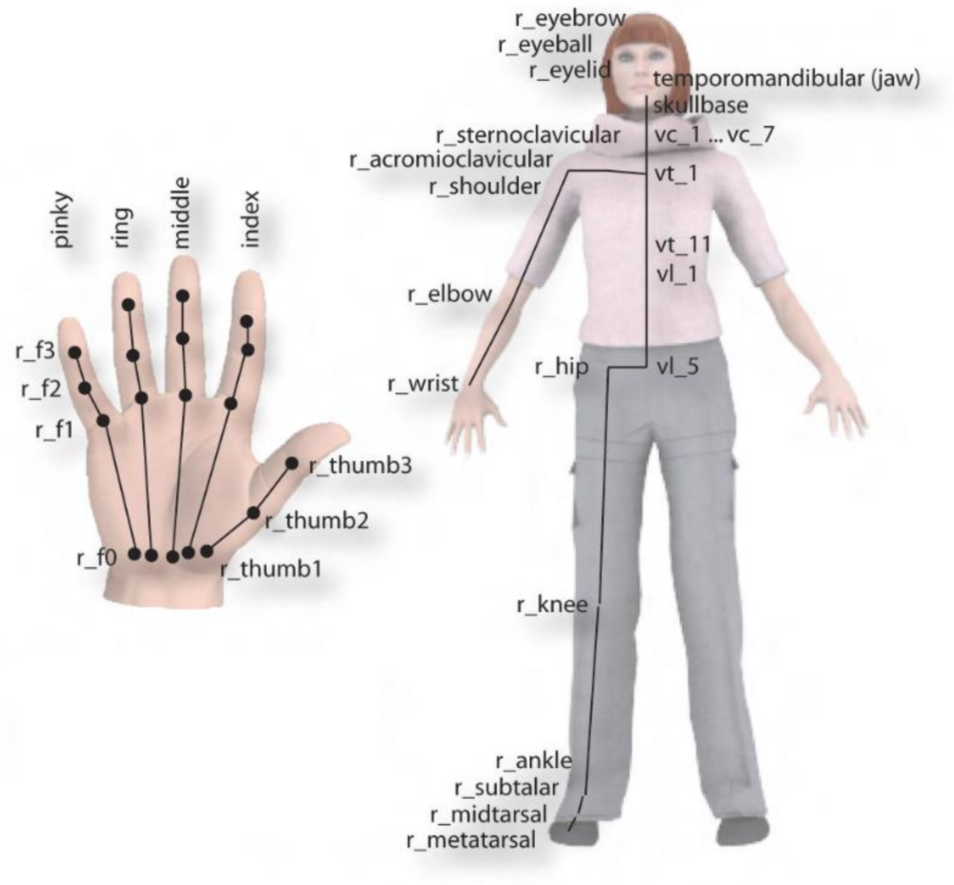
Szenengraphen

Aufbau und Traversierung



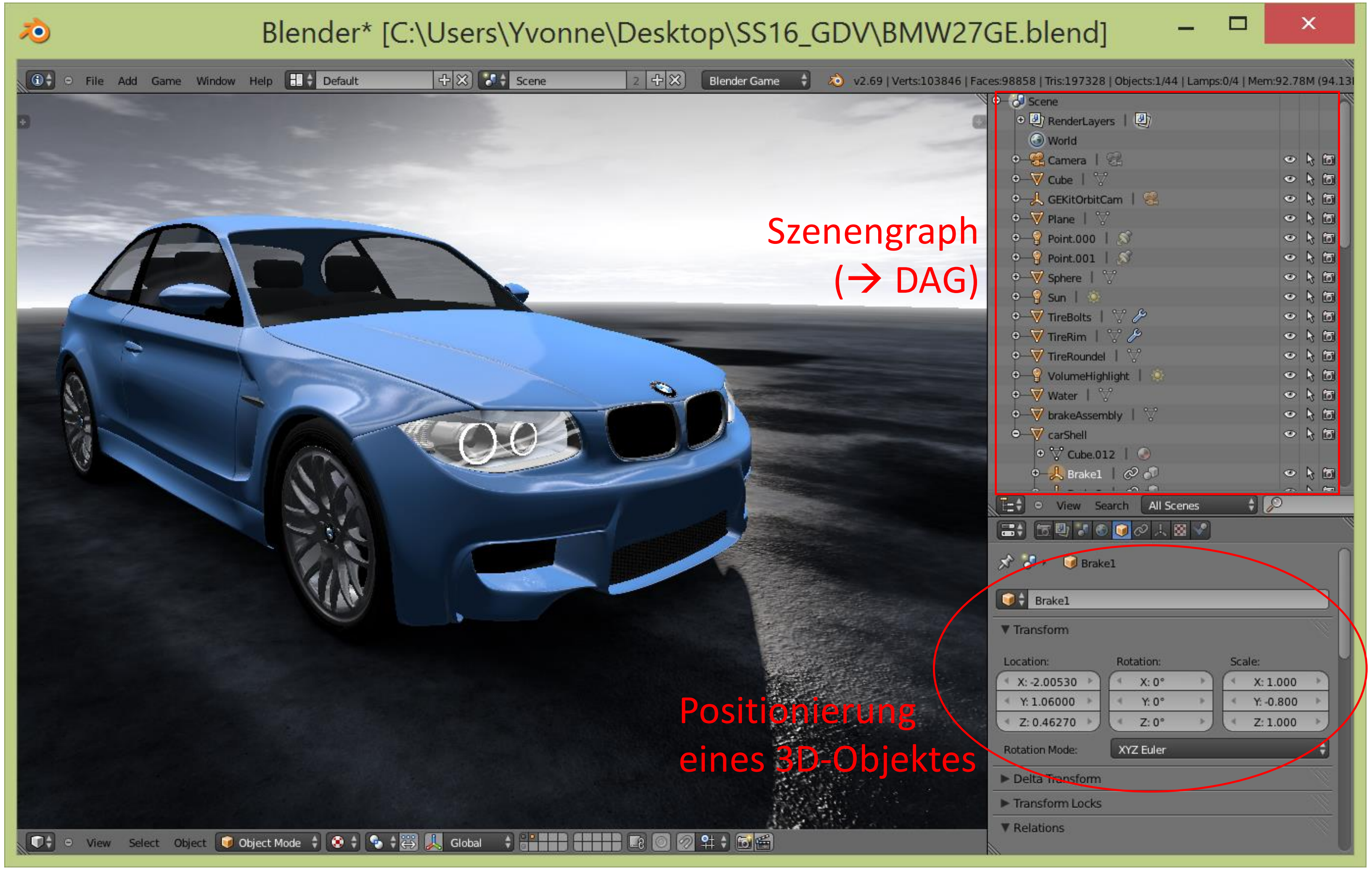
Zwei Beispiele für Szenengraphen

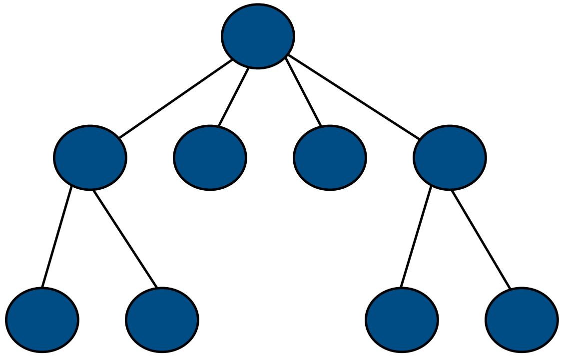




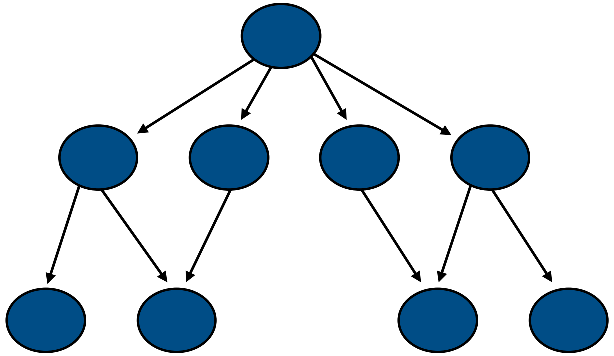
Anwendung bei Modellierung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES





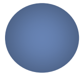
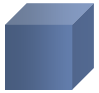
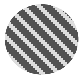
Baum

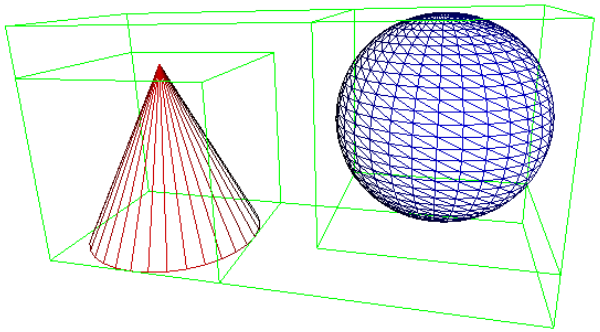


Gerichteter zyklenfreier Graph

Szenengraph (SG)

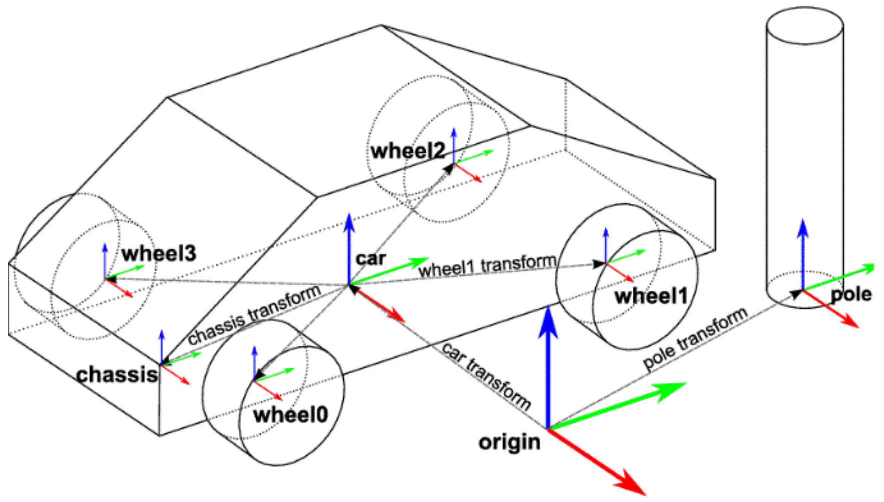
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

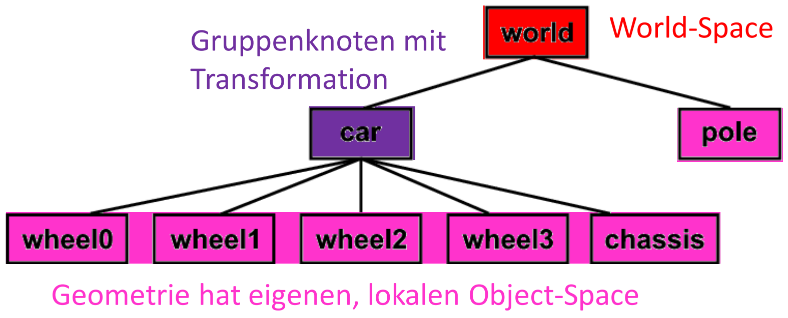
- Hierarchische Modellierung einer 3D-Szene
  - Durch gerichteten azyklischen Graphen (DAG) mit Eltern-Kind Beziehungen
  - Modelliert räumliche Beziehung zwischen (Teil-)Objekten
  - Erlaubt z.B. hierarchisches Frustum Culling
- Besteht aus mindestens drei Knotentypen (darstellbare Primitive in Blättern)
  -  Gruppen
  -  Geometrien (inkl. Materialeigenschaften)
  -  Transformationen
- Dient zur Verwaltung einer komplexeren Szene
  - Gruppierung von Geometrien zu Gruppen
  - Gruppierung von Gruppen zu Gruppen
  - Gruppierung von Gruppen zu einer Szene



Szenengraph traversieren

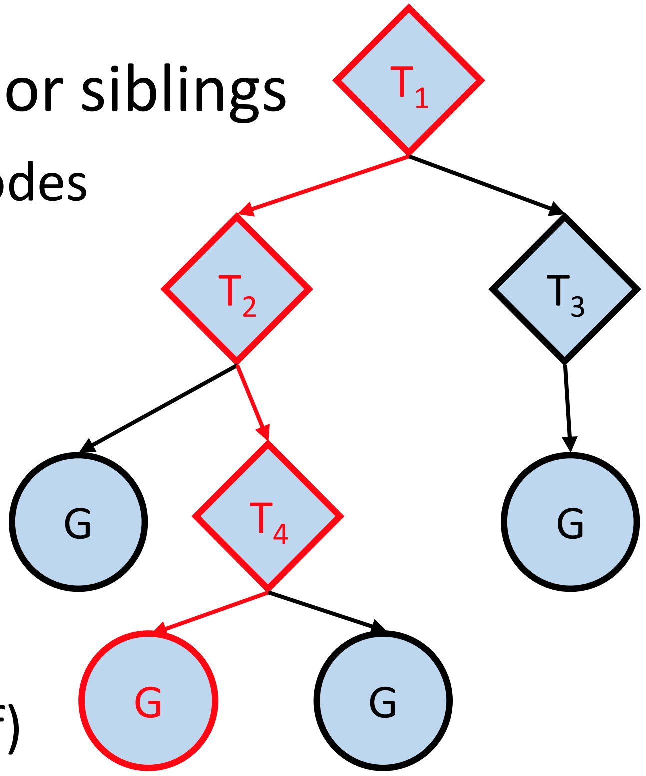
- Den Knoten sind Koordinatensysteme zugeordnet
  - Innere (Transformations-)Knoten halten Matrizen M_i, gruppieren Objekte (Teilgraph)
- Gesamttransformation M durch Traversierung je von Wurzel bis Blattknoten
  - Transformationsmatrizen werden beim Depth First Traversal aufmultipliziert
  - Blätter halten Geometrie je in lokalem Objektraum





Order of Transformations in SG

- Starting from root other elements are inserted as children or siblings
  - Geometry in leaf nodes, with inheritance of attributes in inner nodes
  - Each SG node inherits transformation of its parent and so on
- Transform nodes help to group and reposition objects
  - Translation, rotation, scale most important properties
  - Transformations are specified in reverse order as applied
- Core idea of SG: transformation hierarchy
  - Accumulated matrix obtained by depth first traversal (root to leaf)
  - Transformations applied to each vertex from leaf to root node
  - Transforms geometry to world space



$p' = T_1 \cdot T_2 \cdot T_4 \cdot p = M \cdot p$

T = Transform
G = Geometry
→ = Child of

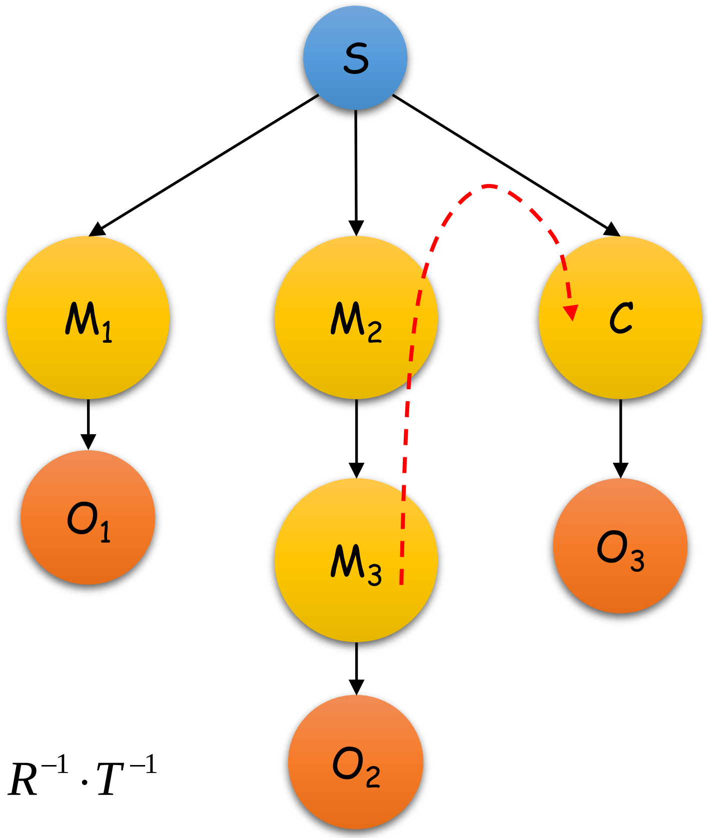
Coordinate Transformations

- Let *S* world and O_1, O_2, O_3 geometric objects (with transformation matrices M_1, M_2, M_3, C)
- Then it holds for world coordinates:
  - $O_{1_{wc}} = M_1 \cdot O_1$
  - $O_{2_{wc}} = M_2 \cdot M_3 \cdot O_2$
  - $O_{3_{wc}} = C \cdot O_3$

Model Matrix M

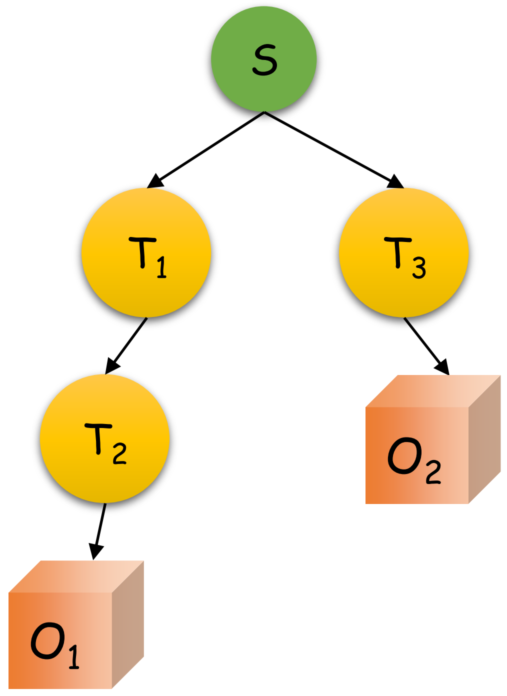
- And for local coordinates in C:
  - $O_{1_{lc}} = C^{-1} \cdot O_{1_{wc}} = C^{-1} \cdot M_1 \cdot O_1$
  - $O_{2_{lc}} = C^{-1} \cdot O_{2_{wc}} = C^{-1} \cdot M_2 \cdot M_3 \cdot O_2$
- → Transformation in O_3's local space:
  - $v_l = C^{-1} \cdot M \cdot v$
  - Where M is accumulated matrix

$$C = T \cdot R \Rightarrow$$
$$C^{-1} = (T \cdot R)^{-1} = R^{-1} \cdot T^{-1}$$
$$\overset{!}{=} R^T \cdot T(-t)$$

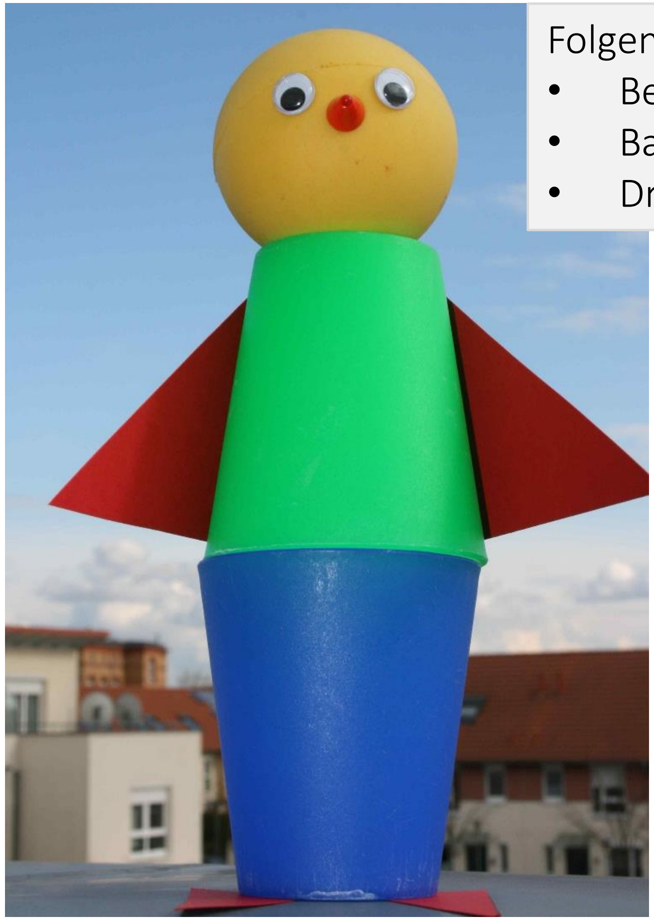


Übung 6

- Was sind die für die Computergraphik wichtigen Typen affiner Abbildungen?
  - Wie lassen sich diese kompakt darstellen?
- Gegeben ist der skizzierte Szenengraph
  - Wie werden die Transformationen auf alle Eckpunkte *p* von Geometrie O_1 angewendet?
  - Wie transformieren Sie die zu O_1 gehörigen Eckpunkte ins Koordinatensystem von O_2?
  - Sei T_2 Rotation um Winkel α um x-Achse und T_1, T_3 Translationen
    - Wie lautet die Gesamtmatrix M?

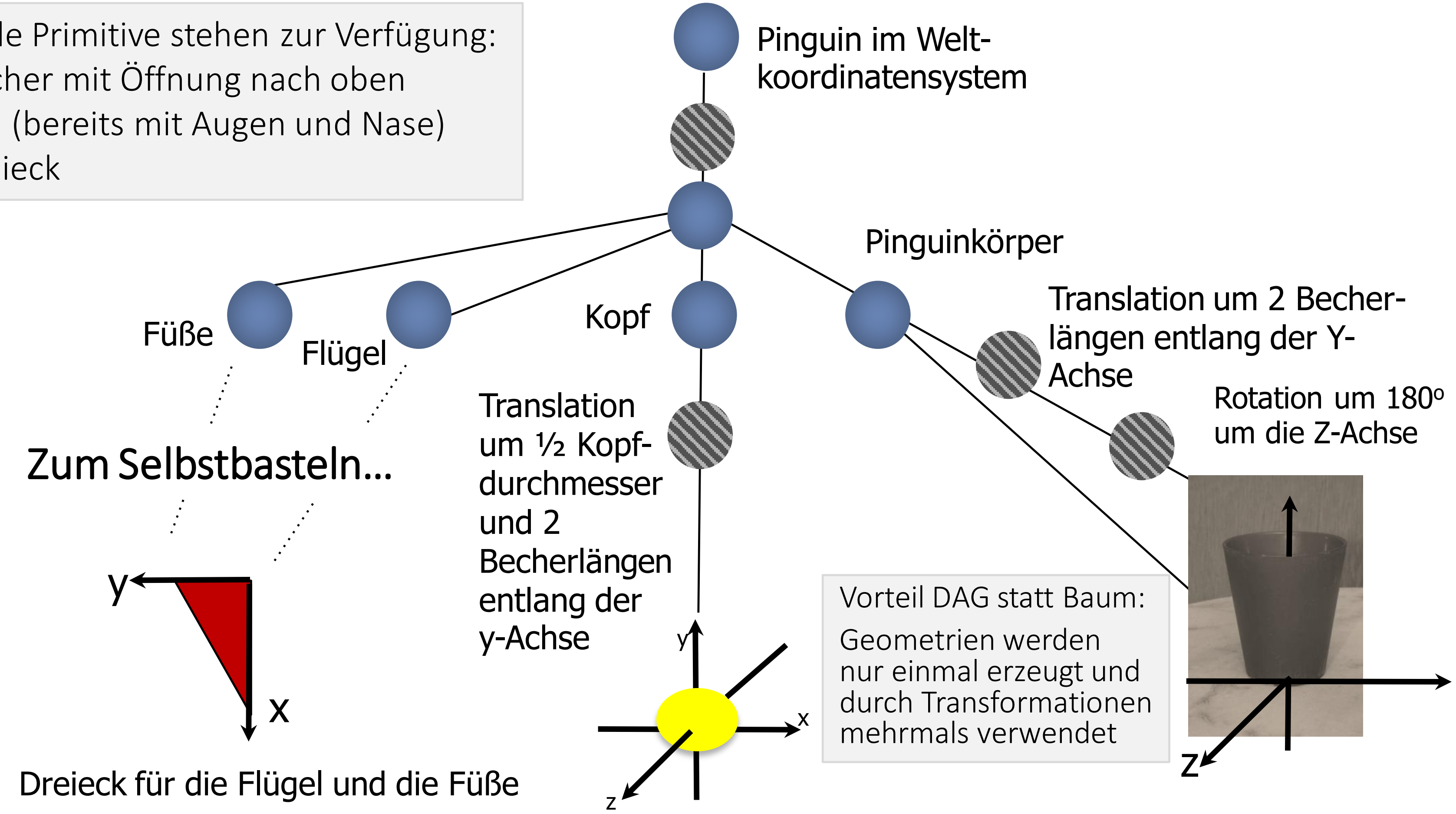


Pinguin mittels Szenengraph erzeugen



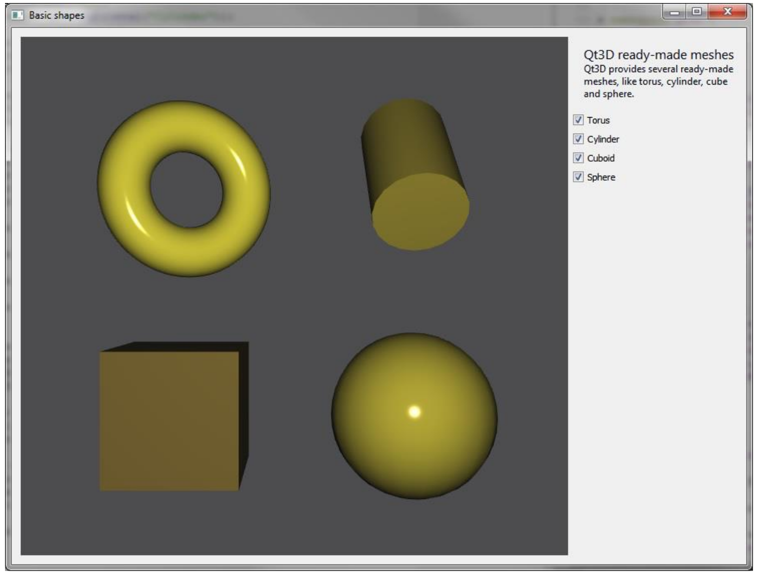
Folgende Primitive stehen zur Verfügung:

- Becher mit Öffnung nach oben
- Ball (bereits mit Augen und Nase)
- Dreieck



Vorteil DAG statt Baum:

Geometrien werden nur einmal erzeugt und durch Transformationen mehrmals verwendet



HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Einführung in Qt 3D

Szenengraphen mit Qt erstellen

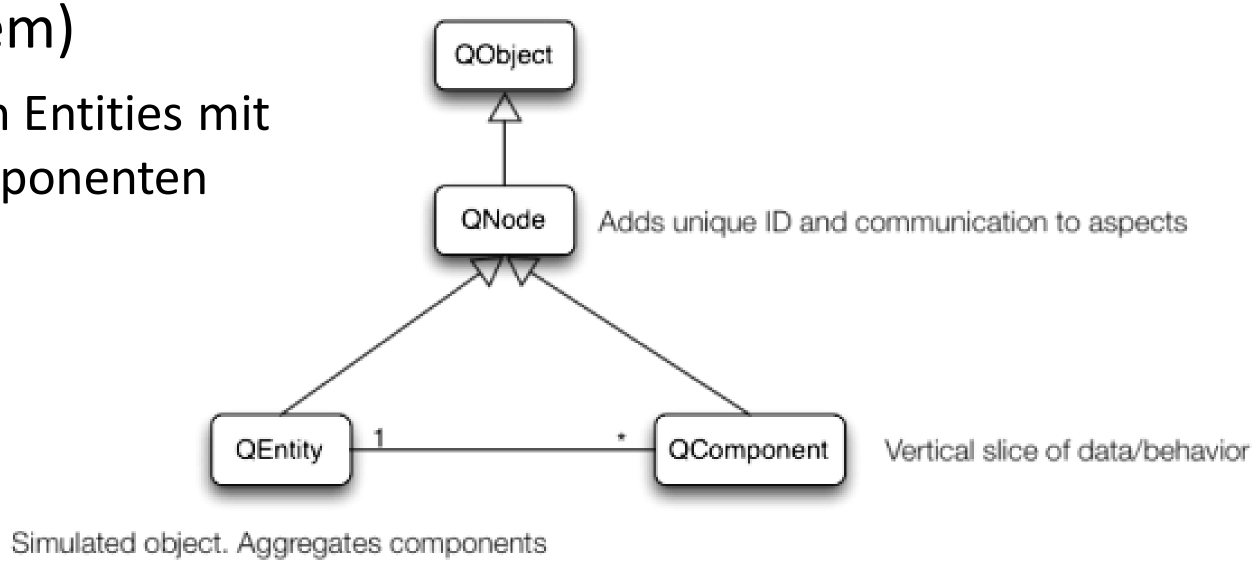
Qt 3D Modul

- Teilmodule (Namespaces):
  - Qt3DCore, Qt3DRender, Qt3DExtras, …
- Dokumentation (C++-Teil beachten)
  - https://doc.qt.io/qt-6/qt3d-index.html
  - https://doc.qt.io/qt-6/qt3d-overview.html
  - Kameraverhalten: https://doc.qt.io/qt-6/qt3dextras-qorbitcameracontroller.html#details
- C++-Beispiele
  - https://doc.qt.io/qt-6/qt3d-examples.html (unten)
- Mathe-Klassen
  - Matrix: https://doc.qt.io/qt-6/qmatrix4x4.html
  - Vector3D: https://doc.qt.io/qt-6/qvector3d.html

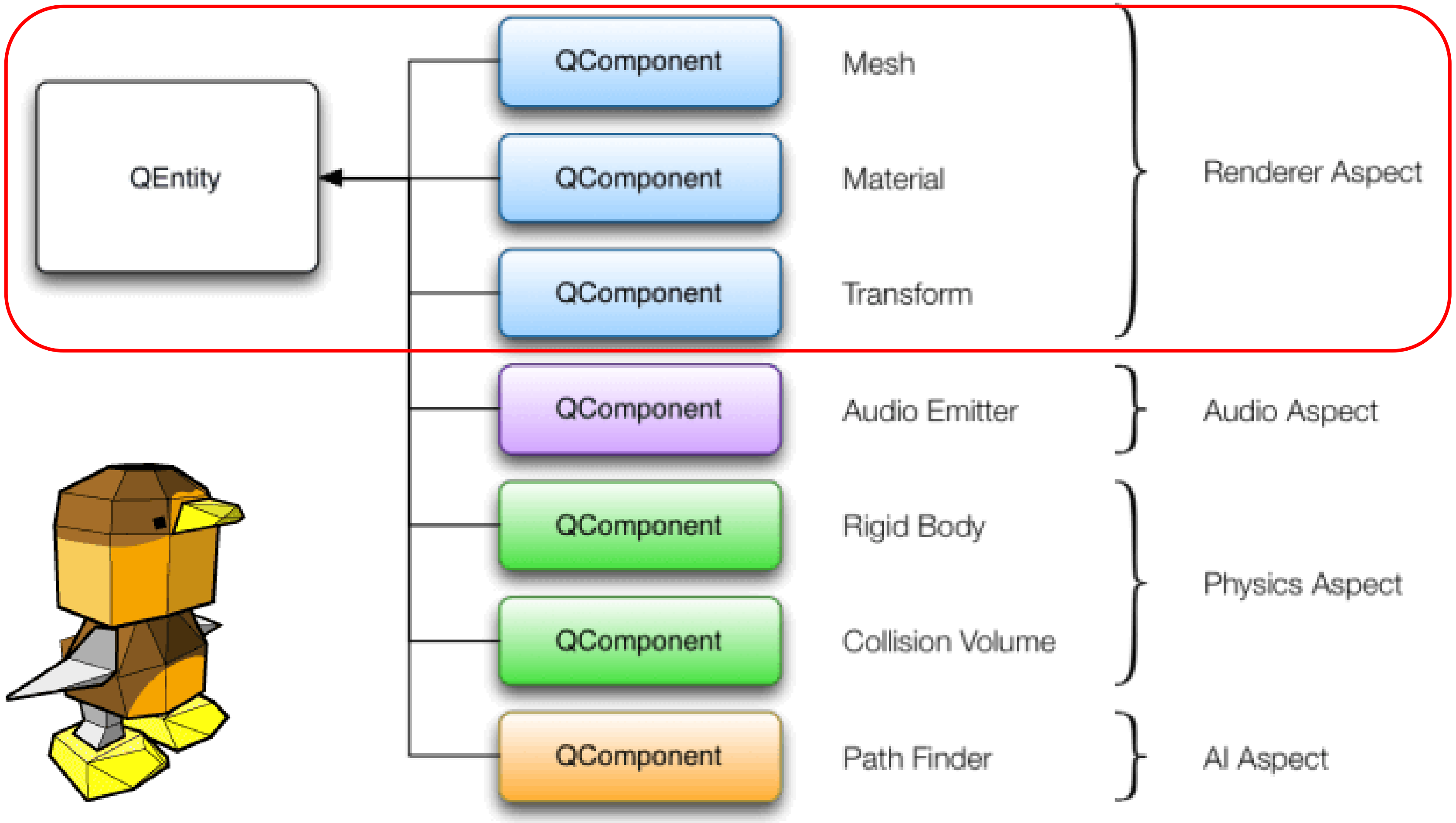
$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Qt 3D Architektur

- Nutzt primär Aggregation statt Vererbung, um dynamisch Objekteigenschaften festzulegen
  - QEntity (repräsentiert Objekt) hat Komponenten (QComponent)
  - Statt DAG Baum aus QEntity Knoten (parametriert durch QComponent)
- Sog. ECS (Entity Component System)
  - Renderer sucht für Darstellung nach Entities mit Mesh, Material und Transform Komponenten

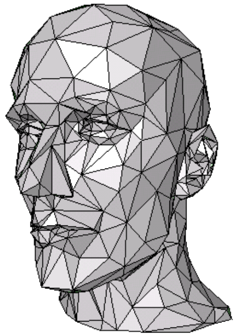


Entity Component System



Aufbauen des Szenengraphen (1)

- Hierarchie von Szenengraph-Knoten
- Gebildet durch Angabe des Parent-Knotens
  - Qt3DCore::QEntity *node = new Qt3DCore::QEntity(parent);
  - Wurzelknoten hat keinen Elternknoten
    - Qt3DCore::QEntity *root = new Qt3DCore::QEntity();



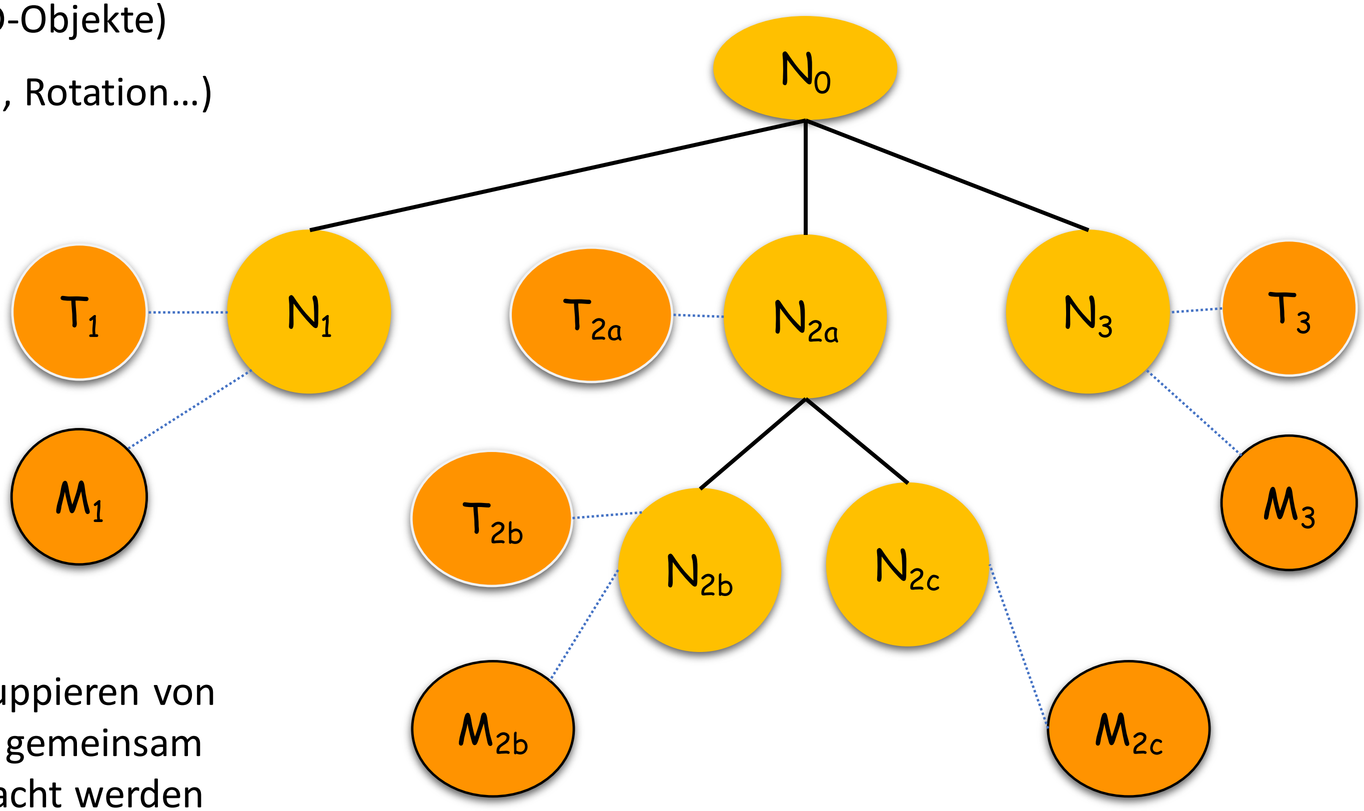
- „Parametrierung“ durch Hinzufügen von Komponenten
  - Mesh: Qt3DExtras::QSphereMesh *mesh = new Qt3DExtras::QSphereMesh();
  - Material: Qt3DExtras::QPhongMaterial *mat = new Qt3DExtras::QPhongMaterial();
  - Transform: Qt3DCore::QTransform *trafo = new Qt3DCore::QTransform();
  - Hinzufügen: node->addComponent(trafo);
    - Rest analog, z.B.: node->addComponent(mesh);

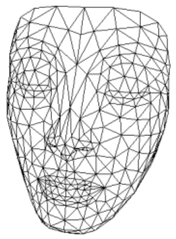
Aufbauen des Szenengraphen (2)

N_i: Szenengraphknoten (Qt3DCore::QEntity)

M_i: Meshes (Polygonnetze für 3D-Objekte)

T_i: Transformationen (Translation, Rotation…)





QEntity insbes. geeignet zum Gruppieren von 3D-Objekten: können so z.B. alle gemeinsam transformiert oder sichtbar gemacht werden

Transformieren mit Qt (Variante 1)

```
Qt3DExtras::QCuboidMesh *cubeMesh = new Qt3DExtras::QCuboidMesh();
cubeMesh->setXExtent(2);
cubeMesh->setYExtent(2);
cubeMesh->setZExtent(2);

Qt3DExtras::QPhongMaterial *mat = new Qt3DExtras::QPhongMaterial();
mat->setDiffuse(QColor(255, 0, 0));

Qt3DCore::QTransform *cubeTransform = new Qt3DCore::QTransform();
cubeTransform->setMatrix(QMatrix4x4(1,  0,  0,  0,
                                    0,  0,  1, -3,
                                    0, -1,  0,  0,
                                    0,  0,  0,  1));

Qt3DCore::QEntity *node = new Qt3DCore::QEntity(root);
node->addComponent(cubeMesh);
node->addComponent(mat);
node->addComponent(cubeTransform);
```

Transformieren mit Qt (Variante 2)

```
Qt3DExtras::QCuboidMesh *cubeMesh = new Qt3DExtras::QCuboidMesh();
cubeMesh->setXExtent(2);
cubeMesh->setYExtent(2);
cubeMesh->setZExtent(2);

Qt3DExtras::QPhongMaterial *mat = new Qt3DExtras::QPhongMaterial();
mat->setDiffuse(QColor(255, 0, 0));

Qt3DCore::QTransform *cubeTransform = new Qt3DCore::QTransform();
cubeTransform->setTranslation(QVector3D(0, -3, 0));
cubeTransform->setRotationX(90);

Qt3DCore::QEntity *node = new Qt3DCore::QEntity(root);
node->addComponent(cubeMesh);
node->addComponent(mat);
node->addComponent(cubeTransform);
```

Sonderfall für **M = T · R · S**
→ interne Transformations-Reihenfolge fest eingebaut, also erst Skalierung, dann Rotation, zuletzt Translation (neu Setzen von z.B. Rotation überschreibt nur alten Wert)

Transformieren mit Qt (Variante 3)

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

```
Qt3DExtras::QCuboidMesh *cubeMesh = new Qt3DExtras::QCuboidMesh();
cubeMesh->setXExtent(2);
cubeMesh->setYExtent(2);
cubeMesh->setZExtent(2);

Qt3DExtras::QPhongMaterial *mat = new Qt3DExtras::QPhongMaterial();
mat->setDiffuse(QColor(255, 0, 0));

Qt3DCore::QTransform *cubeTransform = new Qt3DCore::QTransform();
cubeTransform->setTranslation(QVector3D(0, -3, 0));
Qt3DCore::QEntity *node = new Qt3DCore::QEntity(root);
node->addComponent(cubeTransform);

Qt3DCore::QTransform *trafo = new Qt3DCore::QTransform();
trafo->setRotationX(90);
Qt3DCore::QEntity *child = new Qt3DCore::QEntity(node);
child->addComponent(cubeMesh);
child->addComponent(mat);
child->addComponent(trafo);
```

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Vielen Dank!

Noch Fragen?