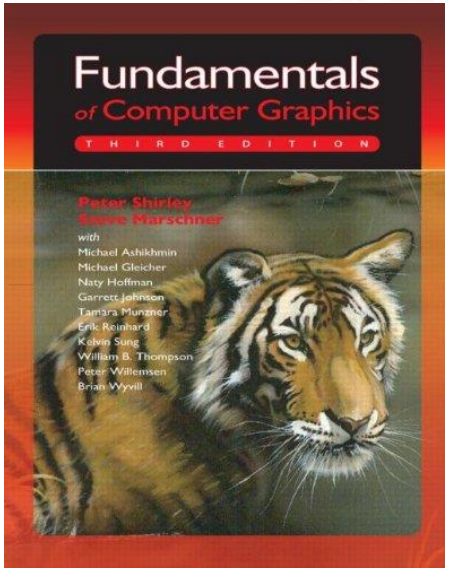
Computergrafik

Vorlesung im Wintersemester 2024/25
Kapitel 3: Transformationen und homogene Koordinaten

Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie

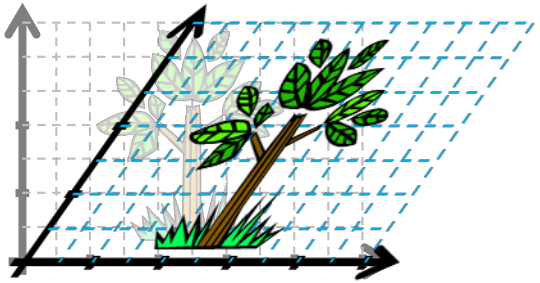
Literatur

- **Fundamentals of Computer Graphics**,
  P. Shirley, S. Marschner, 3rd Edition, AK Peters
  → Kapitel 5, 6 und 7

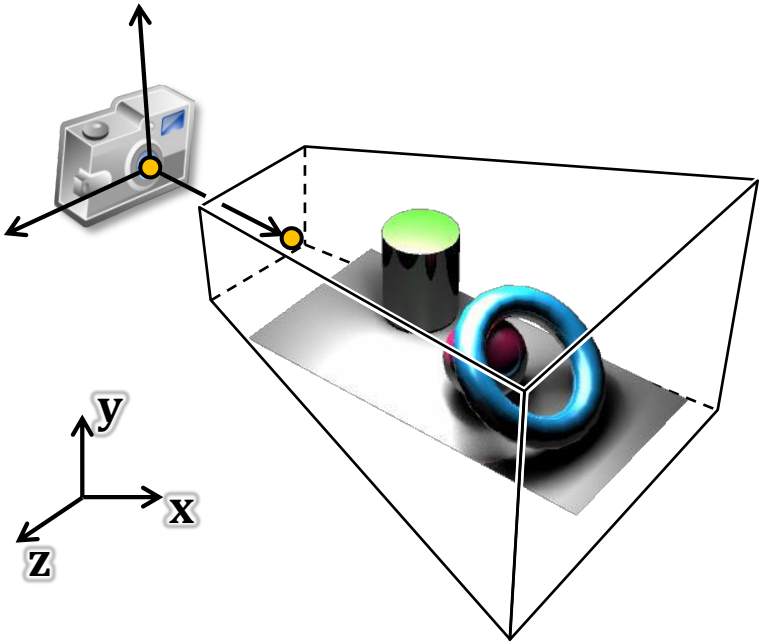


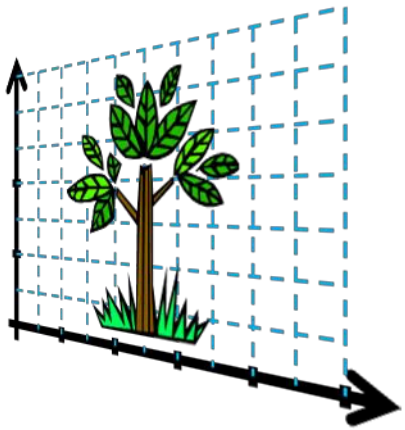
Inhalt

- 2D Transformationen
- 3D Transformationen
- Affine Abbildungen
- Homogene Koordinaten
- Transformation von Normalenvektoren
- Koordinatensysteme in der CG
  und hierarchische Modellierung



$$\begin{pmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{pmatrix}$$



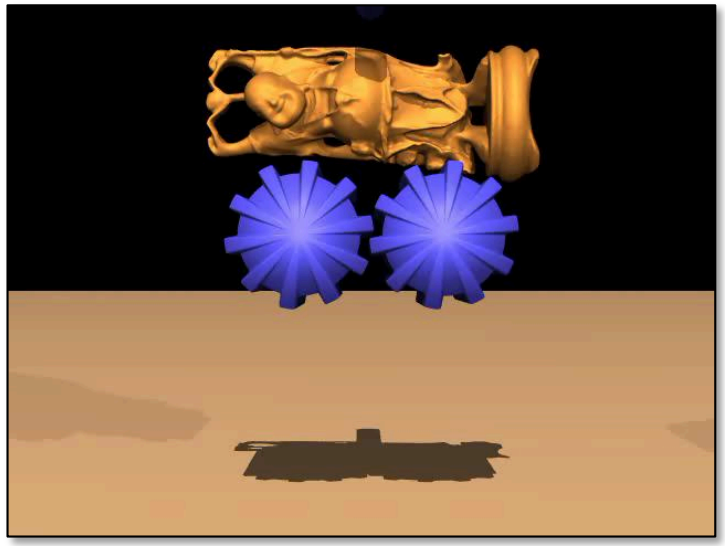


Was ist eine Transformation

- was ist eine Transformation?
  - eine Transformation bildet einen Punkt $\mathbf{x}$ auf einen Punkt $\mathbf{x}'$ ab

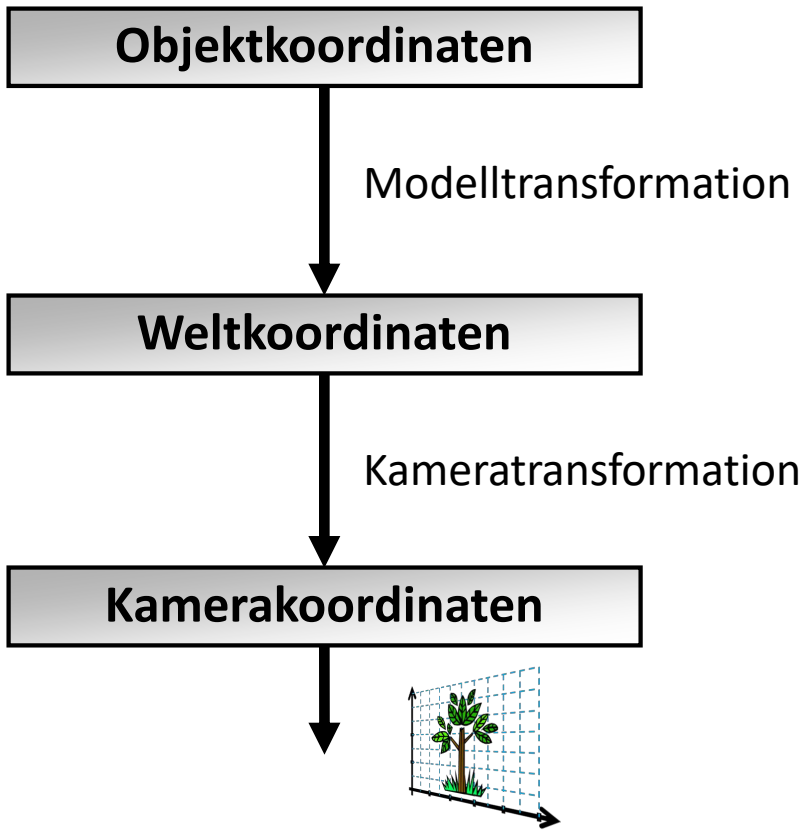
- eine lineare Transformation T ist eine Abbildung, die einen Punkt $\mathbf{x} \in \mathbb{R}^n$ auf einen Punkt $\mathbf{x}' \in \mathbb{R}^m$ abbildet: $\mathbf{x}' = T(\mathbf{x}) = \mathbf{A}\mathbf{x}$
  - $\mathbf{A} \in \mathbb{R}^{m \times n}$ ist die Transformationsmatrix von T
  - $\mathbf{A}$ hat m Zeilen und n Spalten

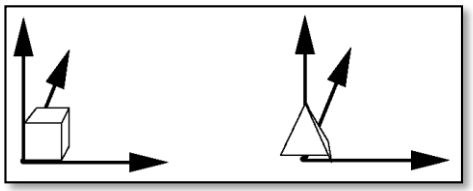
- Beispiele
  - Platzierung von Objekten in einer Szene
  - Animation und Deformation
  - Kameratransformation und Projektion
  - aber auch:
    - Echtzeit-Schattenverfahren
    - Texturierung
    - Spiegelungen usw.

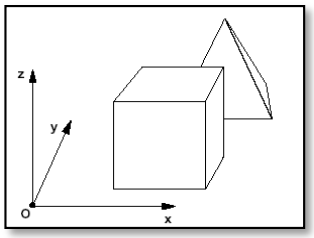


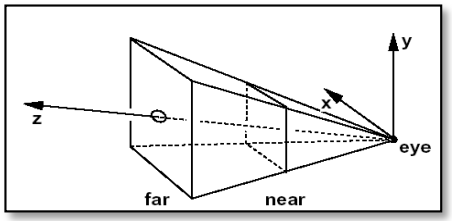
Koordinatensysteme in der CG

- Objekte in einer Szene werden zur Modellierung (Beschreibung) in ihrem eigenen **Objekt- oder Modell-Koordinatensystem** angegeben
- die Platzierung der Objekte im **Weltkoordinatensystem** erfolgt dann durch Translation, Rotation, Skalierung etc.
- dann erfolgt – beim Raytracing implizit – die Transformation in das **Kamerakoordinatensystem** oder der **Sichtstrahlen in Weltkoordinaten**



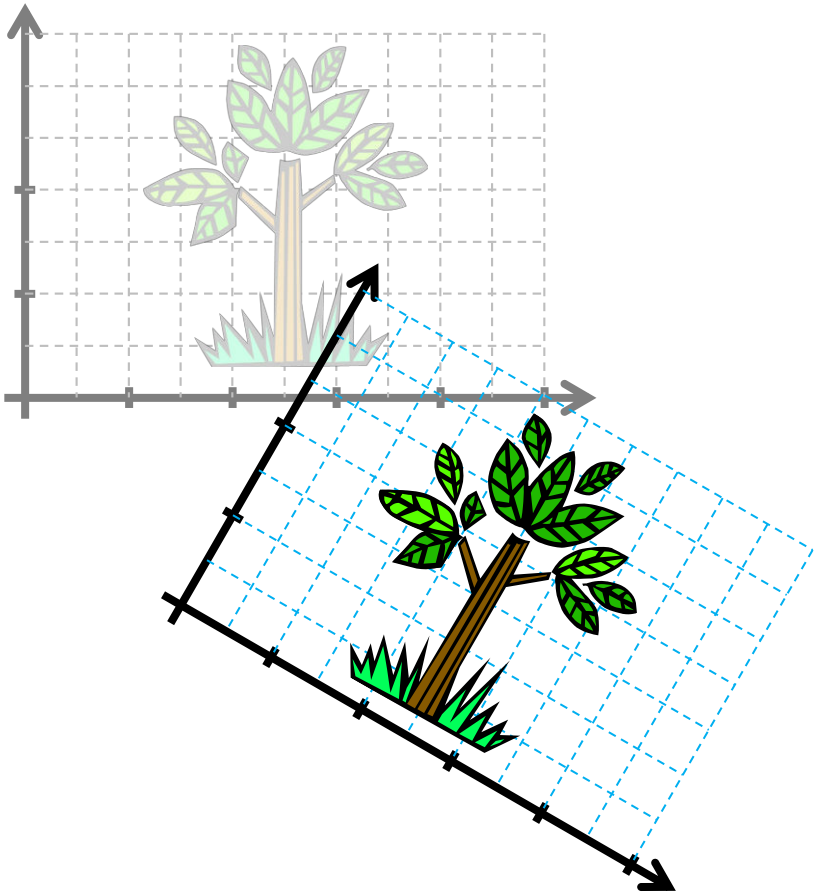


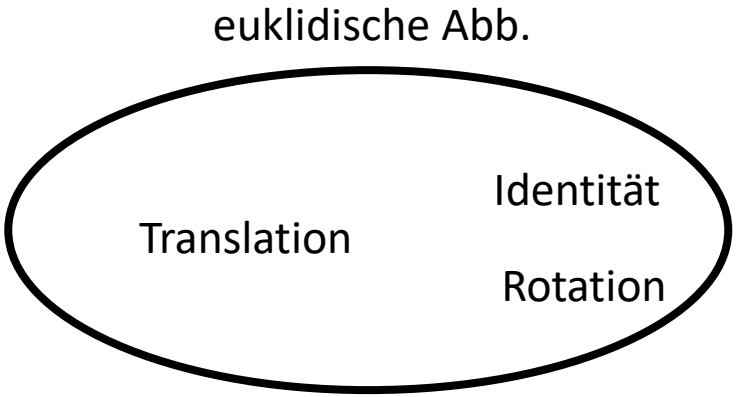




Transformationsgruppen

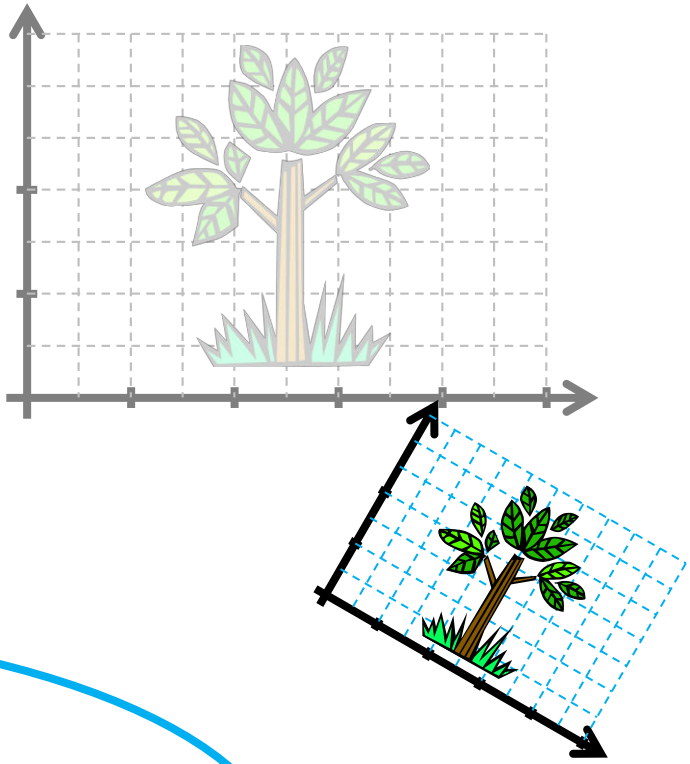
- euklidische/rigide Transformationen
  - erhalten Abstände und Inhaltsgrößen (Volumina, …)
  - erhalten Winkel

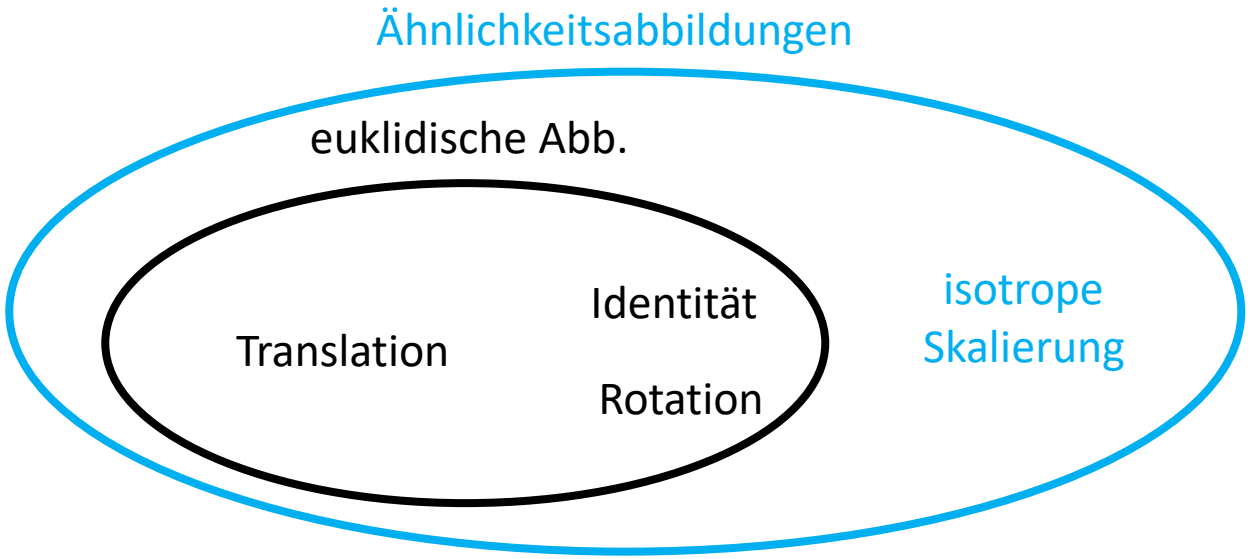




Transformationsgruppen

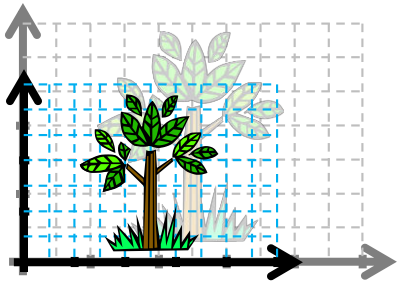
- Ähnlichkeitsabbildungen
  - erhalten Winkel



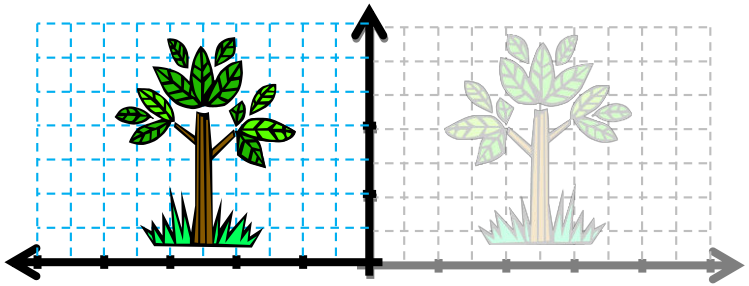


Transformationsgruppen

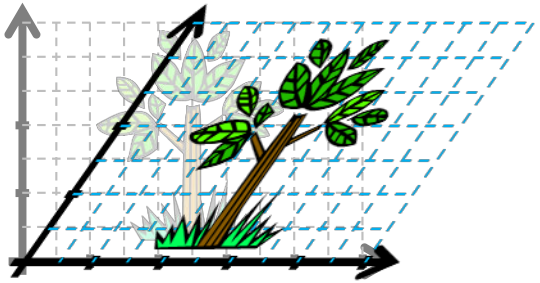
- lineare Abbildungen



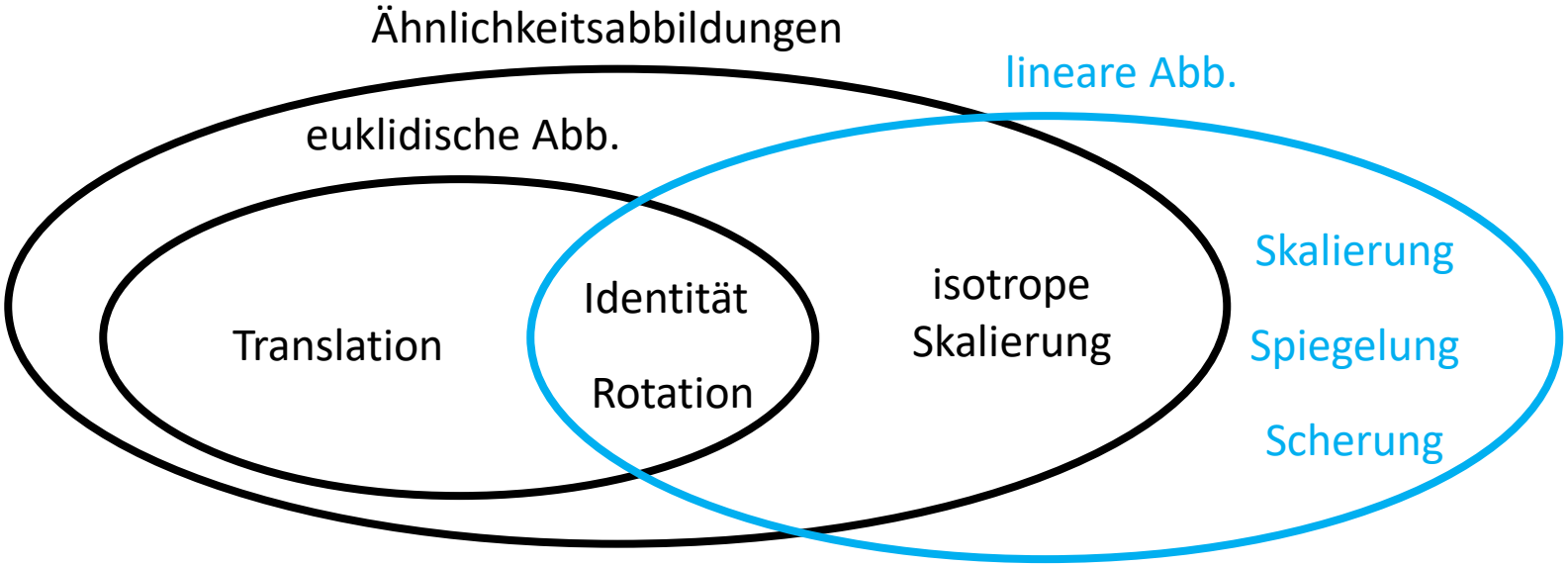
Skalierung



Spiegelung

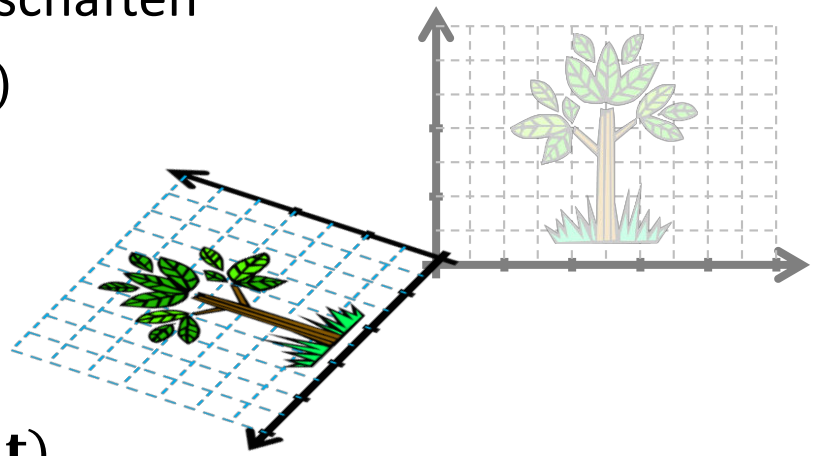


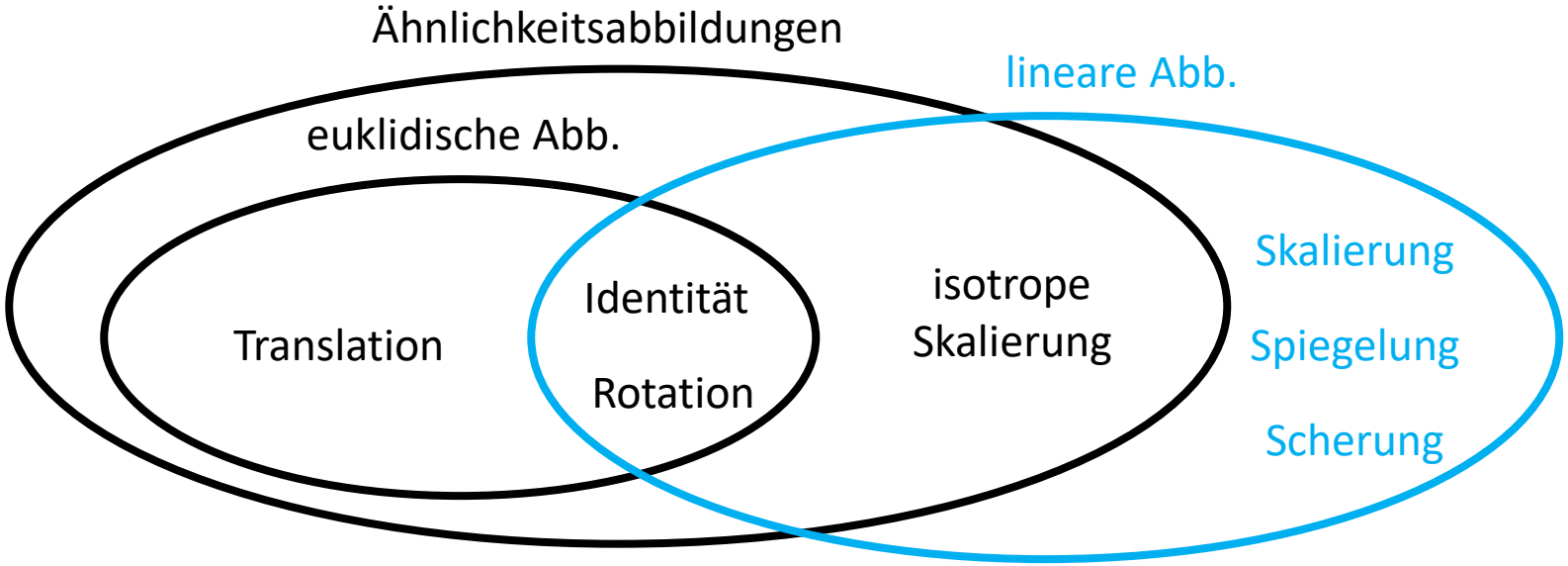
Scherung



Transformationsgruppen

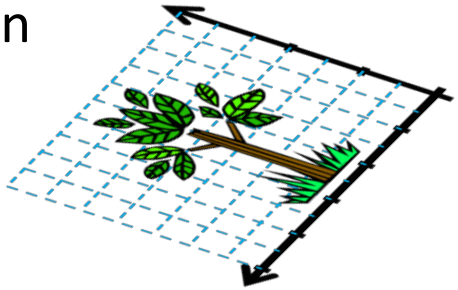
- lineare Abbildungen haben zwei Eigenschaften
  - additiv: $T(\mathbf{p}+\mathbf{q}) = T(\mathbf{p}) + T(\mathbf{q})$
  - homogen: $T(a\mathbf{p}) = aT(\mathbf{p})$
- eine Translation ist nicht linear:
  - sei $T(\mathbf{p}) = \mathbf{p} + \mathbf{t}$
  - $T(a\mathbf{p}) = a\mathbf{p} + \mathbf{t} \neq aT(\mathbf{p}) = a(\mathbf{p} + \mathbf{t})$

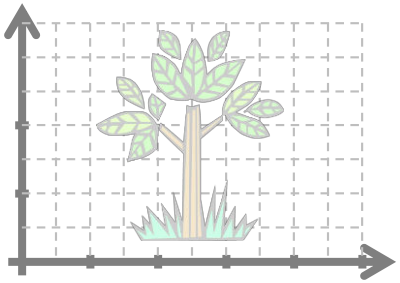


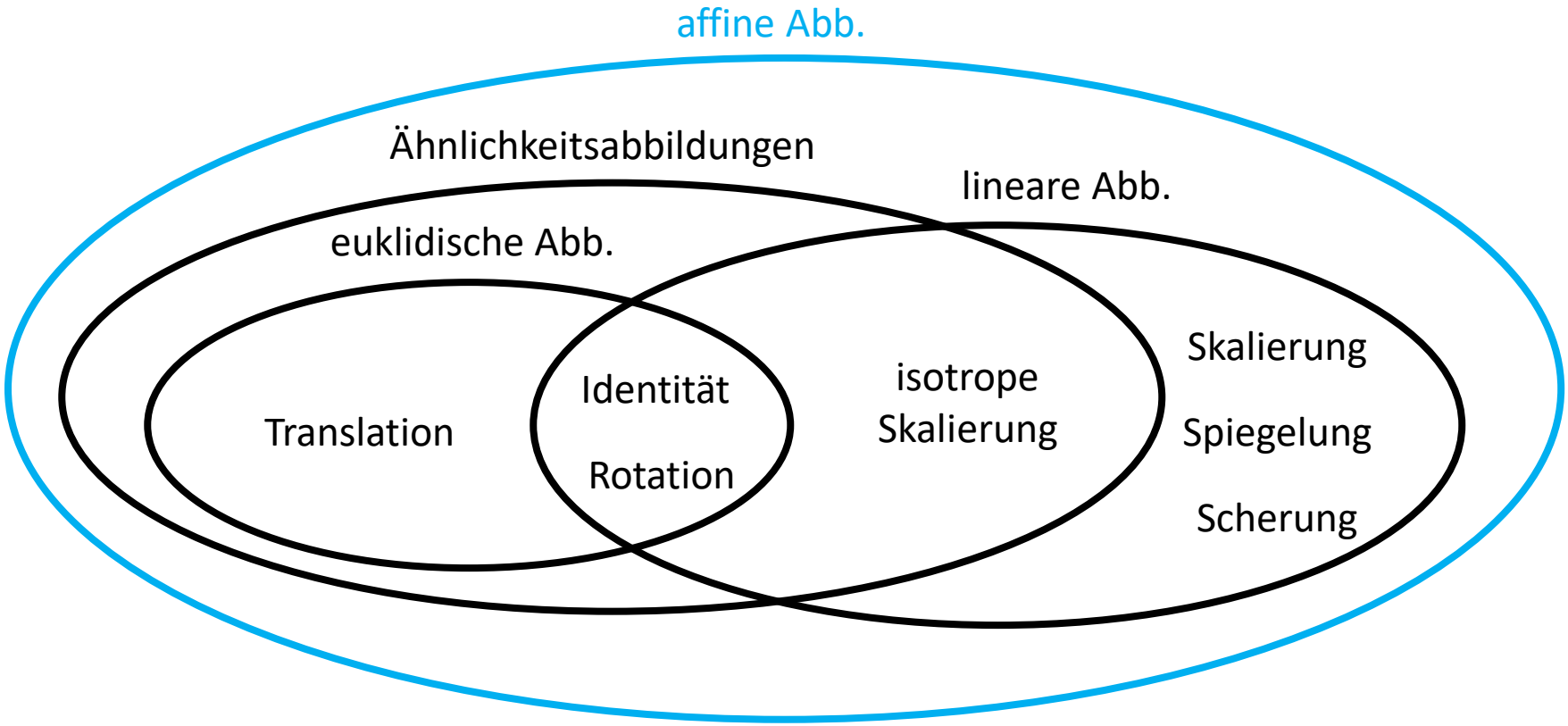


Transformationsgruppen

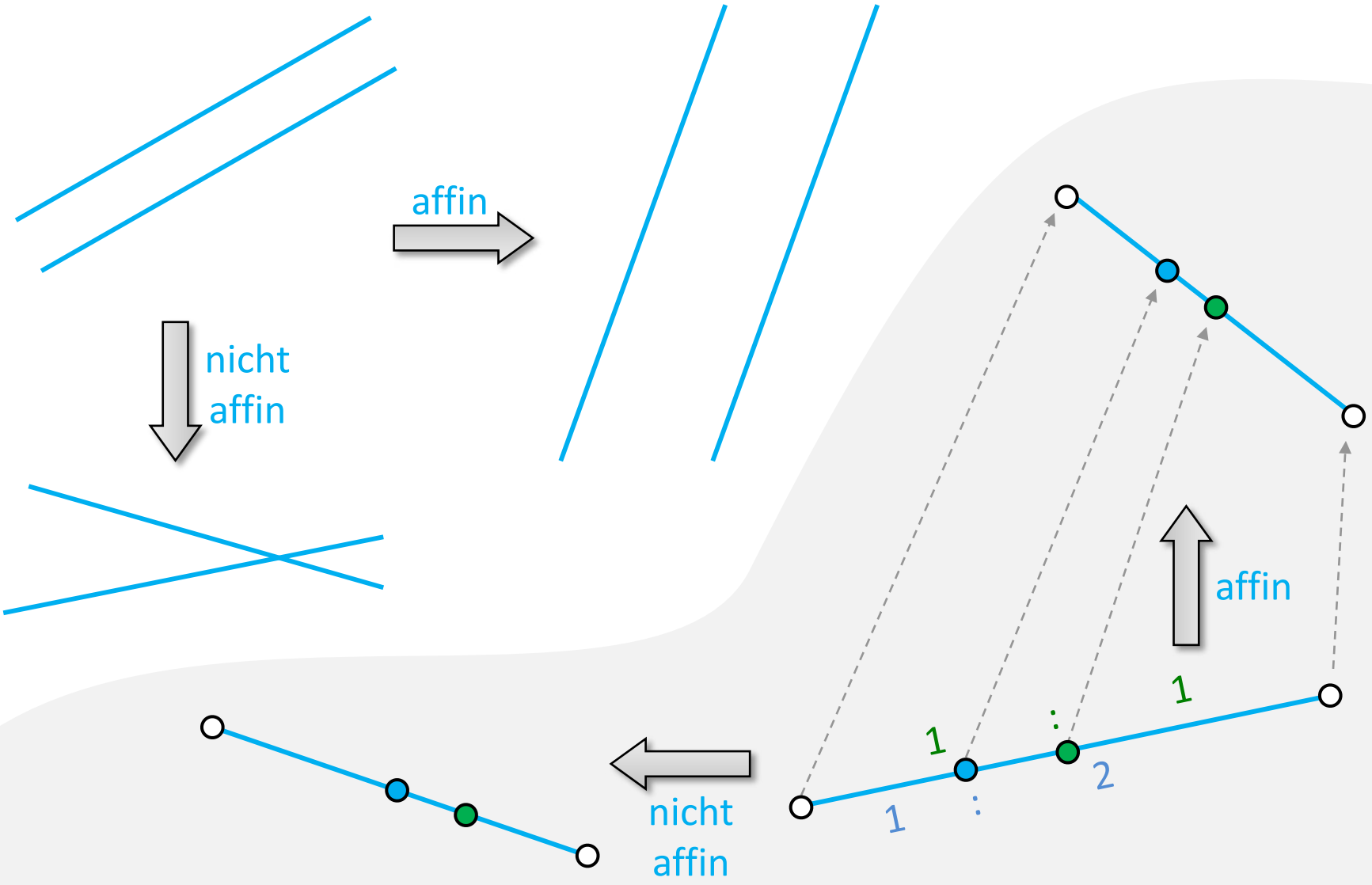
- affine Abbildungen
  - parallele Linien werden erhalten
  - teilverhältnistreu





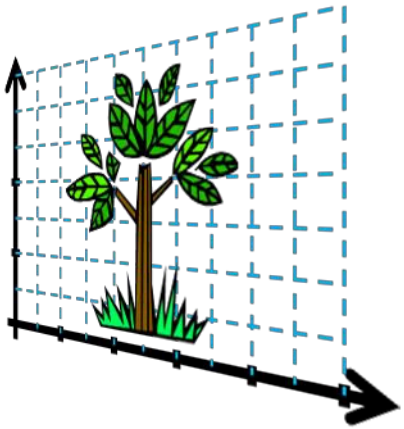


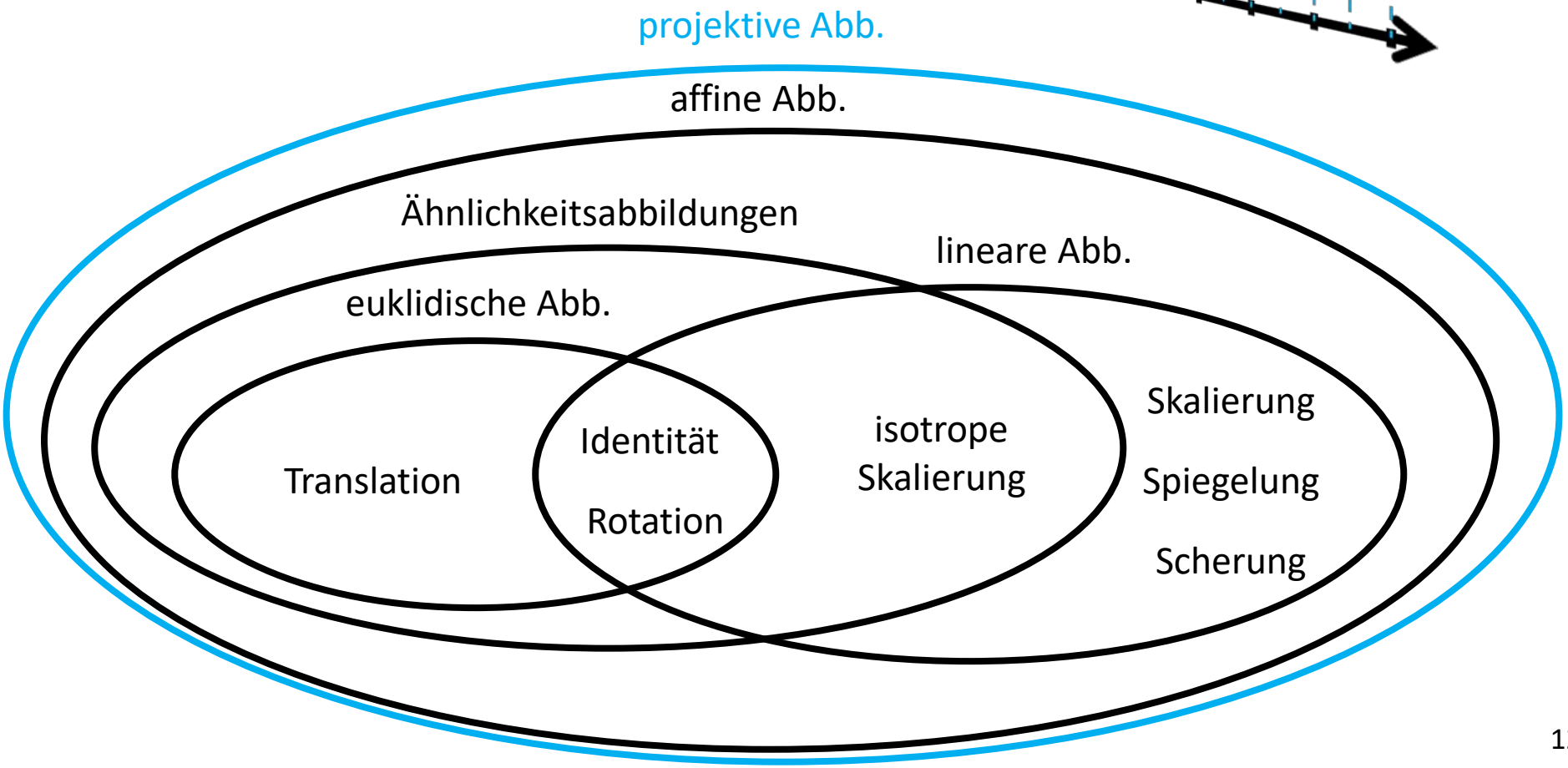
Affine Abbildungen



Transformationsgruppen

- projektive Abbildungen
  - Geraden werden auf Geraden abgebildet
  - Bsp. perspektivische Abbildung





2D Transformationen

Grundlegende lineare 2D Transformationen

- Beschreibung mittels Vektor-Matrix-Multiplikationen
  - Skalierung
  - Scherung
  - Spiegelung
  - Rotation
  - später: zusammengesetzte Transformationen, z.B. Spiegelung an einer Geraden/Ebene, Rotation um einen Punkt/Achse, …

- Beispiel: Matrizen für diese Transformationen haben die Form

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

2D Transformationen

Skalierung

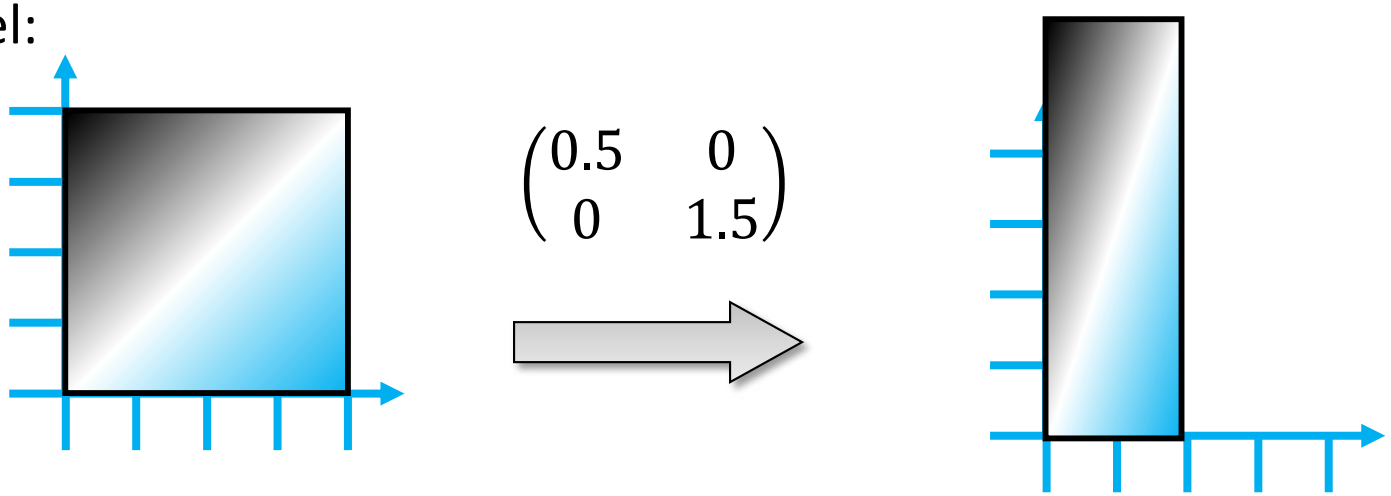
- ändert Längen, Winkel bei nicht-uniformer Skalierung (wenn $s_x \neq s_y$)

$$scale(s_x, s_y) = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$$

- Transformation eines Vektors

$$\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \end{pmatrix}$$

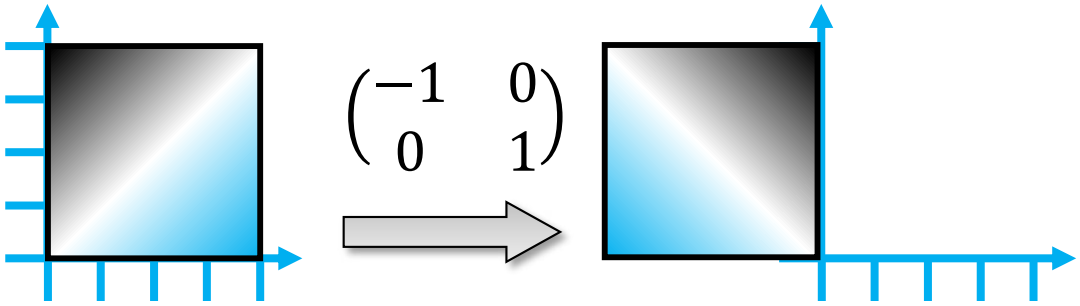
- Beispiel:



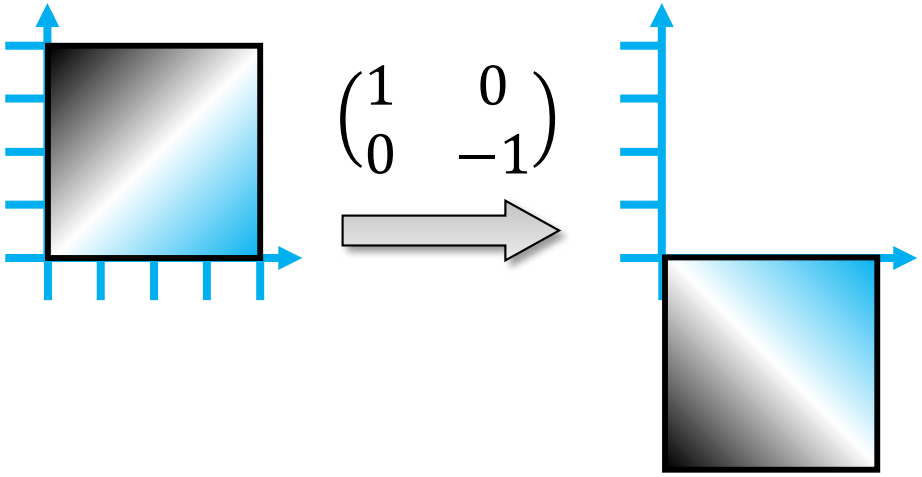
2D Transformationen

Spiegelung

- Spiegeln eines Vektors an einer der Koordinatenachsen
  - Spiegelung ist eine negative Skalierung
- Spiegelung an der y-Achse (Multiplikation der x-Koordinate mit -1)



- … an der x-Achse (Multiplizieren der y-Koordinate mit -1)

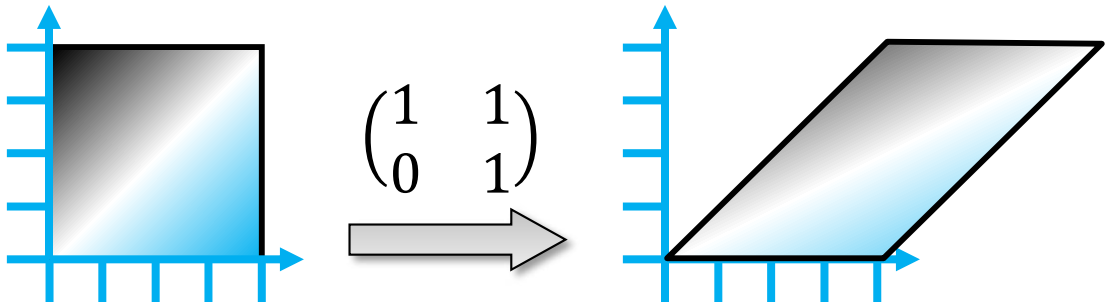


2D Transformationen

Scherung (Transvektion)

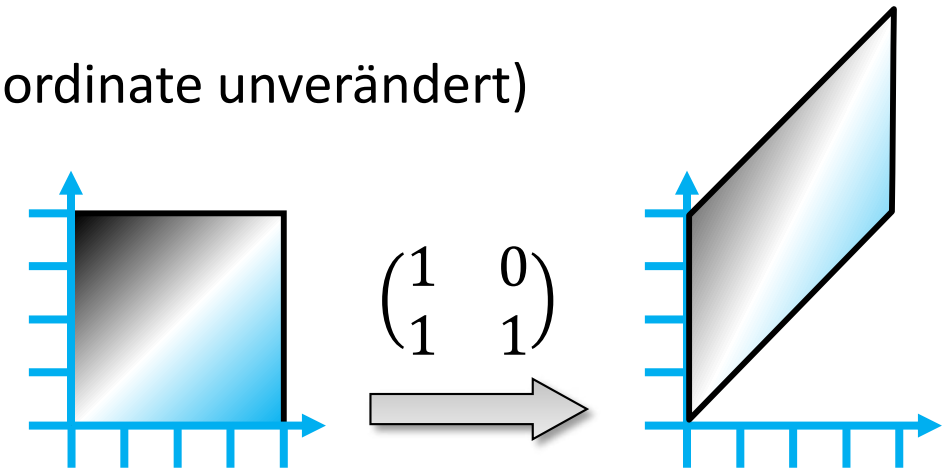
- Verschiebung parallel zu einer Achse (Flächeninhalt bleibt erhalten)
- Länge des Verschiebungsvektors proportional zum Abstand von der Achse
- Beispiele
  - horizontale Scherung (y-Koordinate bleibt unverändert)

$$shear_x(s) = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix}$$



  - vertikale Scherung (x-Koordinate unverändert)

$$shear_y(s) = \begin{pmatrix} 1 & 0 \\ s & 1 \end{pmatrix}$$

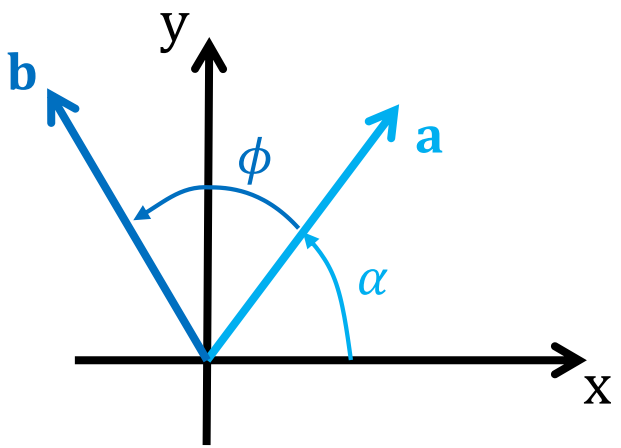


2D Transformationen

Rotation

- Beispiel
  - geg. ein Vektor $\mathbf{a} = (a_x, a_y)$ mit Winkel α zur x-Achse
  - Länge $r := |\mathbf{a}| = \sqrt{a_x^2 + a_y^2}$
  - Polarkoordinaten

$$a_x = r \cdot \cos\alpha$$
$$a_y = r \cdot \sin\alpha$$



  - Rotation um den Winkel ϕ (gegen den Uhrzeigersinn)

$$b_x = r\cos(\alpha + \phi) = r\cos\alpha\cos\phi - r\sin\alpha\sin\phi$$
$$b_y = r\sin(\alpha + \phi) = r\sin\alpha\cos\phi + r\cos\alpha\sin\phi$$

(Additionstheoreme)

2D Transformationen

Rotation

- Rotation um den Winkel ϕ (gegen den Uhrzeigersinn)

$$b_x = r\cos(\alpha + \phi) = r\cos\alpha\cos\phi - r\sin\alpha\sin\phi$$
$$b_y = r\sin(\alpha + \phi) = r\sin\alpha\cos\phi + r\cos\alpha\sin\phi$$

- Substitution ($a_x = r \cdot \cos\alpha$ und $a_y = r \cdot \sin\alpha$)

$$b_x = a_x\cos\phi - a_y\sin\phi$$
$$b_y = a_y\cos\phi + a_x\sin\phi$$

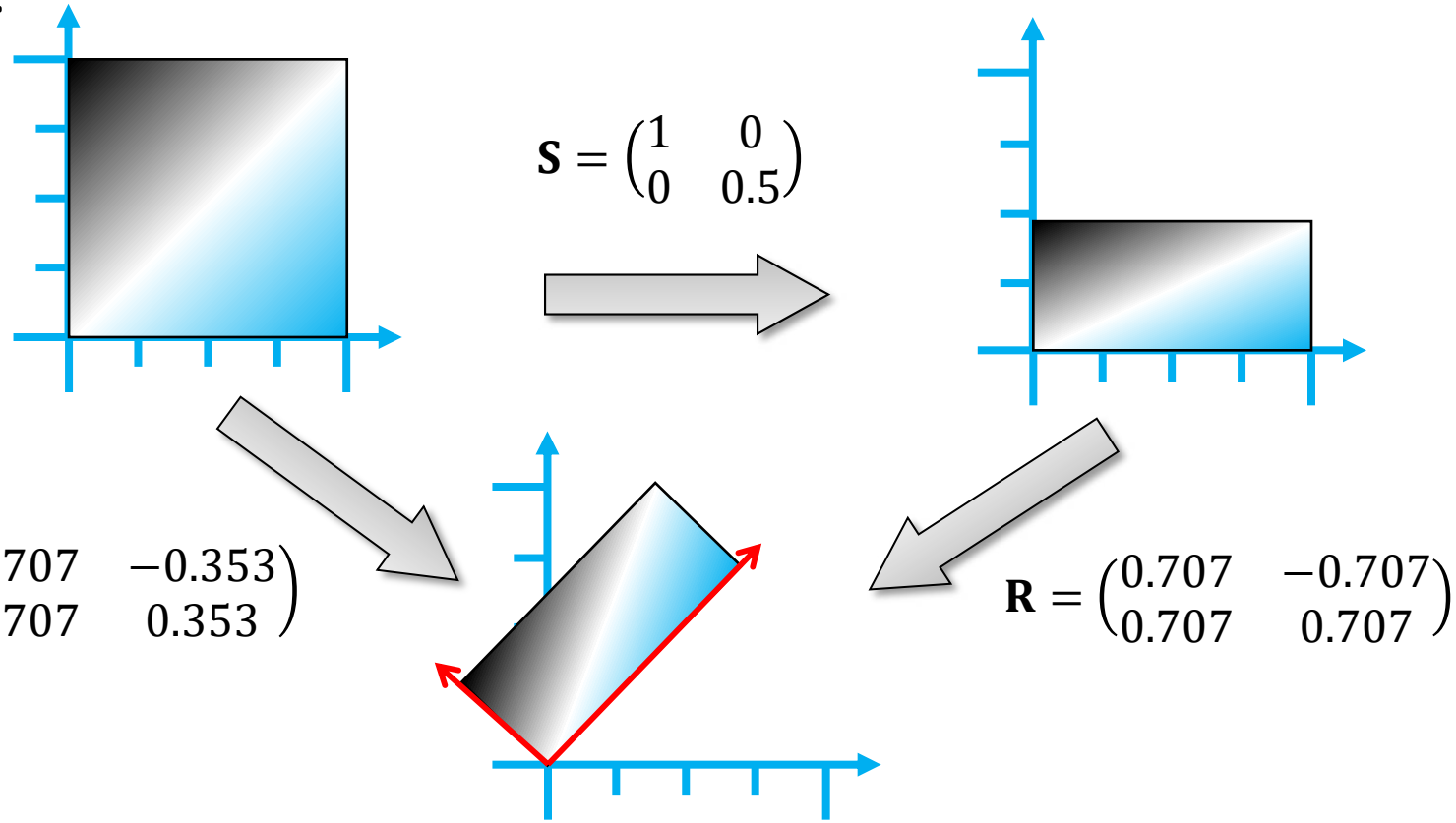
- Matrix: Abb. von $\mathbf{a}$ auf $\mathbf{b}$ bzw. Rotation um ϕ gegen den Uhrzeigersinn

$$rotate(\phi) = \mathbf{R}(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

Zusammengesetzte 2D Transformationen

Hintereinanderausführung von Transformationen

- häufig in der Computergrafik,
  Hintereinanderausführung = Matrixmultiplikation
  - erste Transformation: $\mathbf{v_2} = \mathbf{S}\mathbf{v_1}$
  - zweite Transformation: $\mathbf{v_3} = \mathbf{R}\mathbf{v_2}$ bzw. $\mathbf{v_3} = \mathbf{R}(\mathbf{S}\mathbf{v_1}) = (\mathbf{RS})\mathbf{v_1}$
- Beispiel:



Zusammengesetzte 2D Transformationen

Hintereinanderausführung von Transformationen

- Effekt zweier (allg. mehrerer) Transformationen durch eine Matrix

$$\mathbf{v_3} = (\mathbf{RS})\mathbf{v_1} = \mathbf{M}\mathbf{v_1}$$

  - Transformationen werden von rechts angewendet
  - hier: erst $\mathbf{S}$, dann $\mathbf{R}$

- Matrix-Multiplikation
  - $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times p}$, $\mathbf{C} = \mathbf{AB}$ $(\in \mathbb{R}^{m \times p})$ mit $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$
    c_{ij} ist Skalarprodukt der i-ten Zeile von $\mathbf{A}$ und der j-ten Spalte von $\mathbf{B}$
  - Eigenschaften
    - assoziativ $(\mathbf{RS})\mathbf{T} = \mathbf{R}(\mathbf{ST})$
    - **nicht** kommutativ $\mathbf{RS} \neq \mathbf{SR}$
      → die Reihenfolge der Transformationen ist entscheidend

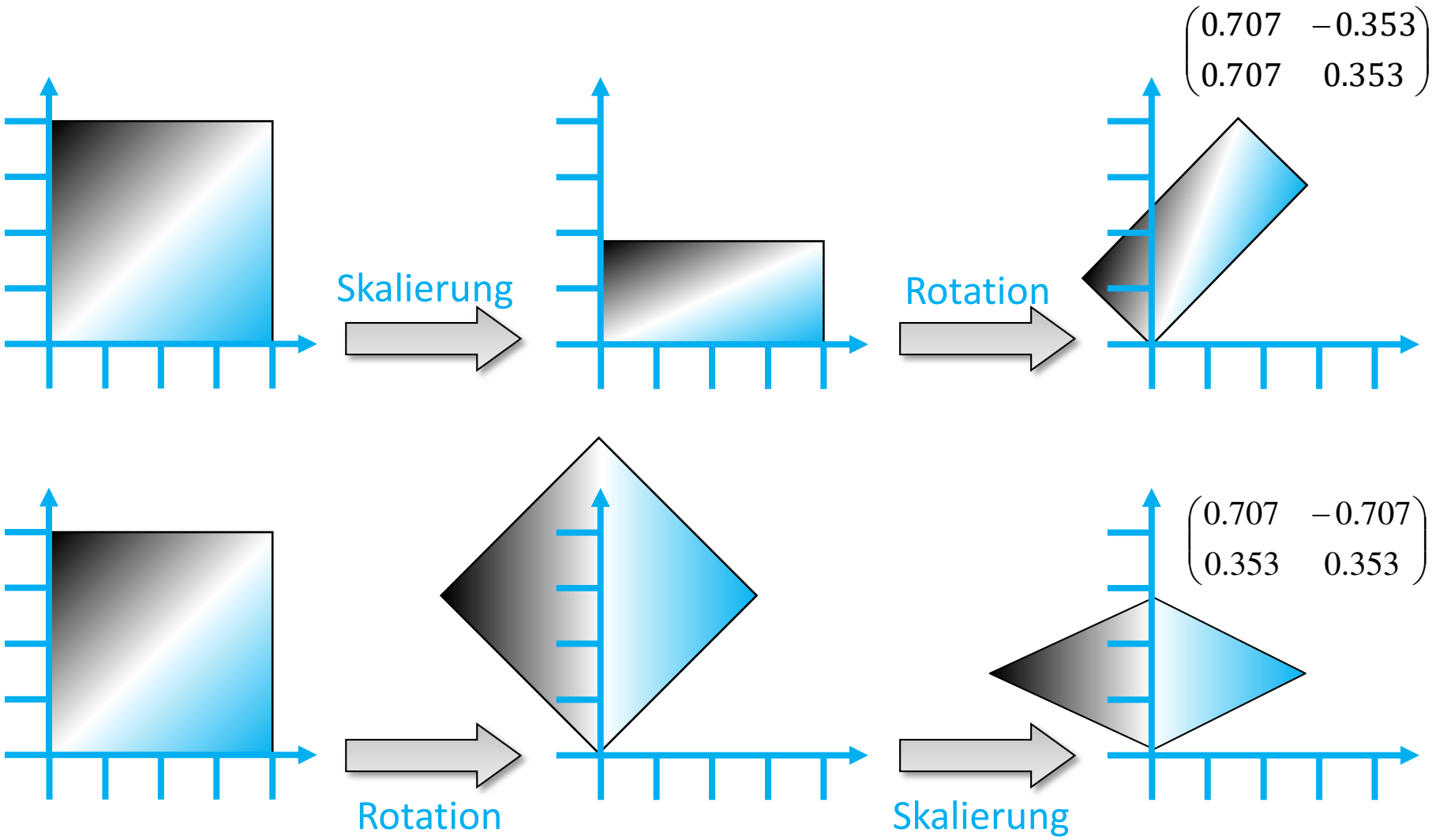
Matrixmultiplikation

- Beispiel: Falksches Schema
  - $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times p}$, $\mathbf{C} = \mathbf{AB}$ ($\in \mathbb{R}^{m \times p}$) mit $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$
  - Element c_{32} entsteht aus dem Skalarprodukt der 3. Zeile von $\mathbf{A}$ und der 2. Spalte von $\mathbf{B}$

					2	2	3	**B**
					-2	-1	2	
					3	4	1	
					-3	4	0	
					1	-3	4	
A	2	-1	0	3	-4	-7	29	-12	$\mathbf{C} = \mathbf{AB}$
	3	-2	2	1	0	13	20	7	
	-1	3	-1	-2	-4	-9	-5	-14	
	3	3	-4	2	-2	-20	1	3	

Zusammengesetzte 2D Transformationen

Reihenfolge der Transformationen ist entscheidend



Repräsentationen von Rotationen in 3D

Verschiedene Darstellungen/Beschreibungen

- orthogonale Matrizen
- Rotationsachse und -winkel
- Euler Rotationen:
  - $\mathbf{R_z} \rightarrow \mathbf{R_x} \rightarrow \mathbf{R_z}$
  - $\mathbf{R_z} \rightarrow \mathbf{R_y} \rightarrow \mathbf{R_z}$
  - $\mathbf{R_x} \rightarrow \mathbf{R_y} \rightarrow \mathbf{R_z}$
- (Einheits-)Quaternionen $x_0 + x_1 i + x_2 j + x_3 k$
- 2 planare Spiegelungen
- …

3D Transformationen

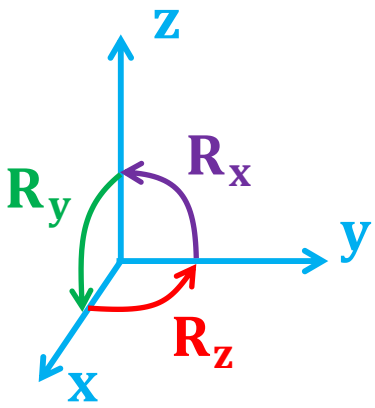
Rotation um die x-, y- bzw. z-Achse

- $\mathbf{R_x}$ dreht die y-Achse in Richtung z-Achse,
  $\mathbf{R_y}$ dreht die z-Achse in Richtung x-Achse und
  $\mathbf{R_z}$ dreht die x-Achse in Richtung y-Achse

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

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

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



3D Transformationen: Rotationen

Rotationen allgemein

- repräsentiert durch **orthogonale** Matrizen (orientierungserhaltend, Zeilen- und Spaltenvektoren **paarweise orthonormal**)
- eine quadratische, reelle Matrix ist orthogonal, wenn gilt:

$$\mathbf{M}^T \cdot \mathbf{M} = \mathbf{M} \cdot \mathbf{M}^T = I_{n\times n} \text{ und } |\det(\mathbf{M})| = 1$$

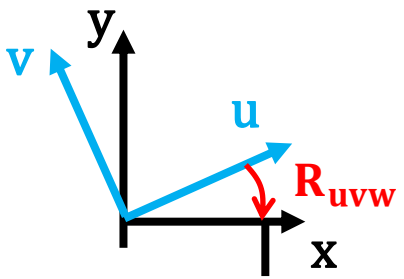
  - es gilt also: $\mathbf{M}^{-1} = \mathbf{M}^T$

- die algebraische Inverse ist auch die geometrische Inverse: auch $\mathbf{M}^{-1}$ bzw. $\mathbf{M}^T$ sind Rotationsmatrizen

3D Transformationen: Rotationen

Rotation des Koordinatensystems u, v, w auf das kartesische KoSys

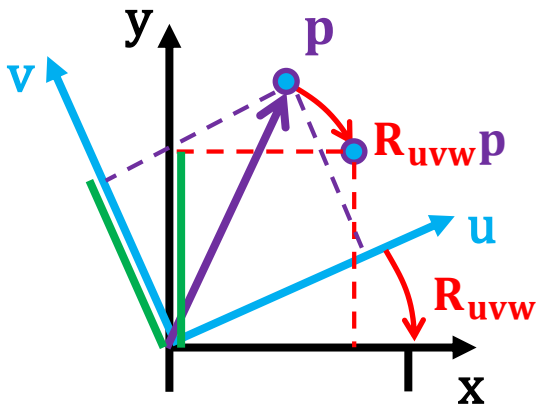
- geg. drei Vektoren $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$, die ein orthonormales System bilden
- Rotationsmatrix $\mathbf{R_{uvw}} = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}$
- $\mathbf{R_{uvw}}$ bildet die Basisvektoren $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ durch Rotation auf die kartesischen Achsen ab:
  - $\mathbf{R_{uvw}u} = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} \mathbf{u} \cdot \mathbf{u} \\ \mathbf{v} \cdot \mathbf{u} \\ \mathbf{w} \cdot \mathbf{u} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \mathbf{x}$
  - $\mathbf{R_{uvw}v} = \mathbf{y}$
  - $\mathbf{R_{uvw}w} = \mathbf{z}$



3D Transformationen: Rotationen

Transformation von Punkten zwischen den Koordinatensystemen

- $\mathbf{R_{uvw}}$ enthält Vektoren $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$, die ein orthonormales System bilden, als Zeilenvektoren
- $\mathbf{R_{uvw}}$ bildet $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ auf die kartesischen Achsen ab
- ist ein Punkt $\mathbf{p}$ definiert in kartesischen Koordinaten bzgl. $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$
  - $\mathbf{R_{uvw}} \cdot \mathbf{p} = \begin{pmatrix} \mathbf{u} \cdot \mathbf{p} \\ \mathbf{v} \cdot \mathbf{p} \\ \mathbf{w} \cdot \mathbf{p} \end{pmatrix}$ berechnet lokale Koordinaten bzgl. $\mathbf{u}, \mathbf{v}, \mathbf{w}$
  - $\mathbf{u} \cdot \mathbf{p}$ ist die Projektion von $\mathbf{p}$ auf $\mathbf{u}$, $\mathbf{v} \cdot \mathbf{p}$ …
    → $\mathbf{R_{uvw}}$ transformiert $\mathbf{p}$ „in“ das KoSys $\mathbf{u}, \mathbf{v}, \mathbf{w}$
    (berechnet lokale Koordinaten bzgl. $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$)
- welcher Rotation für $\mathbf{p}$ entspricht das?



3D Transformationen: Rotationen

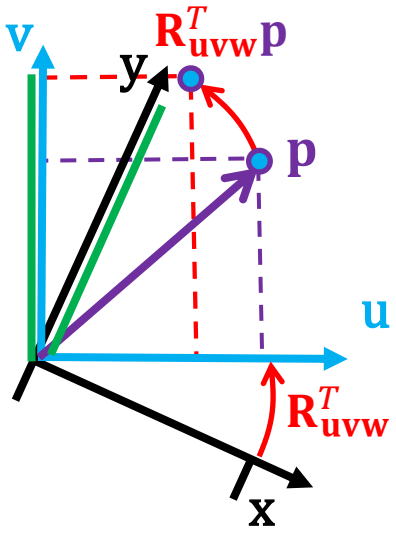
Transformation von Punkten zwischen den Koordinatensystemen

- $\mathbf{R}_{\mathbf{uvw}}^{T}$ bildet die kartesischen Einheitsvektoren auf $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ ab
- sind die Koordinaten vom Punkt $\mathbf{p}$ gegeben bzgl. KoSys $\mathbf{u}, \mathbf{v}, \mathbf{w}$ dann

$$\mathbf{R}_{\mathbf{uvw}}^{T} \cdot \mathbf{p} = \begin{pmatrix} u_x p_x & v_x p_y & w_x p_z \\ u_y p_x & v_y p_y & w_y p_z \\ u_z p_x & v_z p_y & w_z p_z \end{pmatrix} = \mathbf{u} \cdot p_x + \mathbf{v} \cdot p_y + \mathbf{w} \cdot p_z$$

berechnet kartesische Koordinaten von $\mathbf{p}$ im globalen KoSys

- $\mathbf{R}_{\mathbf{uvw}}^{T}$ transformiert $\mathbf{p}$ „aus“ dem KoSys $\mathbf{u}, \mathbf{v}, \mathbf{w}$



Rotation um eine beliebige Achse

Rotation um eine Achse $\mathbf{d}$ um den Winkel ϕ

- bestimme (irgendein) orthonormales KoSys mit $\mathbf{d}$ als eine der Achsen
  - sei $\mathbf{d} = (d_x, d_y, d_z)$ mit $|\mathbf{d}| = 1$
  - wähle z.B. $\mathbf{e} = \frac{1}{\sqrt{d_y^2 + d_z^2}}(0, -d_z, d_y)$ und $\mathbf{f} = \mathbf{d} \times \mathbf{e}$
  - Abbildung von $\mathbf{d}$ auf die $\mathbf{x}$-Achse des kartesischen KoSys (analog $\mathbf{e}$ auf $\mathbf{y}$ und $\mathbf{f}$ auf $\mathbf{z}$): transformiert einen Punkt in das KoSys $\mathbf{d}, \mathbf{e}, \mathbf{f}$

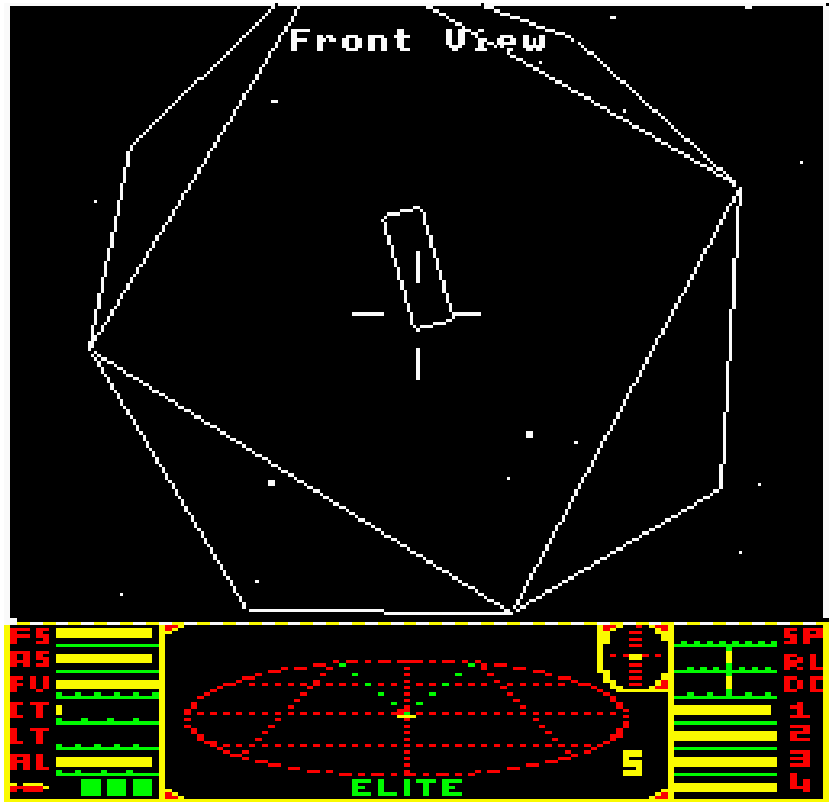
$$\mathbf{M} = \begin{pmatrix} \mathbf{d}^\mathbf{T} \\ \mathbf{e}^\mathbf{T} \\ \mathbf{f}^\mathbf{T} \end{pmatrix} = \begin{pmatrix} d_x & d_y & d_z \\ e_x & e_y & e_z \\ f_x & f_y & f_z \end{pmatrix}$$

- Rotation um die $\mathbf{x}$-Achse (dann im KoSys $\mathbf{d}, \mathbf{e}, \mathbf{f}$): $\mathbf{R_x}(\phi)$
- Transformation zurück in das ursprüngliche KoSys: $\mathbf{M^{-1}} = \mathbf{M^T}$
- vollständige Matrix

$$\mathbf{R_{d,\phi}} = \mathbf{M^{-1}} \mathbf{R_x}(\phi) \mathbf{M}$$

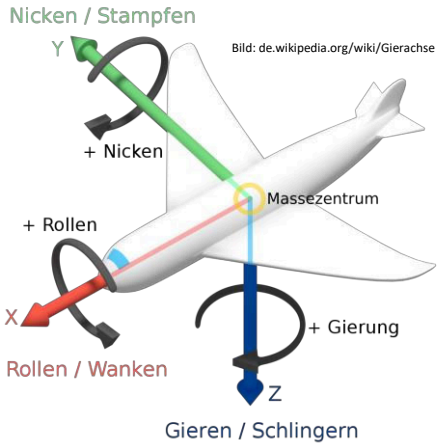
Elite (space trading video game)

- Original von 1984 auf dem BBC Micro (2 MHz 6502, 16-32 KiB RAM) von David Braben und Ian Bell



Euler Rotationen

- jede Rotation kann durch 3 Rotationen um die Hauptachsen (x, y, z) ausgedrückt werden (Leonhard Euler 1707 – 1783, Gerolamo Cardano 1501 – 1576)
  - man kann auch andere Achsen und Reihenfolgen festlegen, z.B. Luftfahrtnorm (DIN 9300) mit Yaw-Pitch-Roll z, y', x''



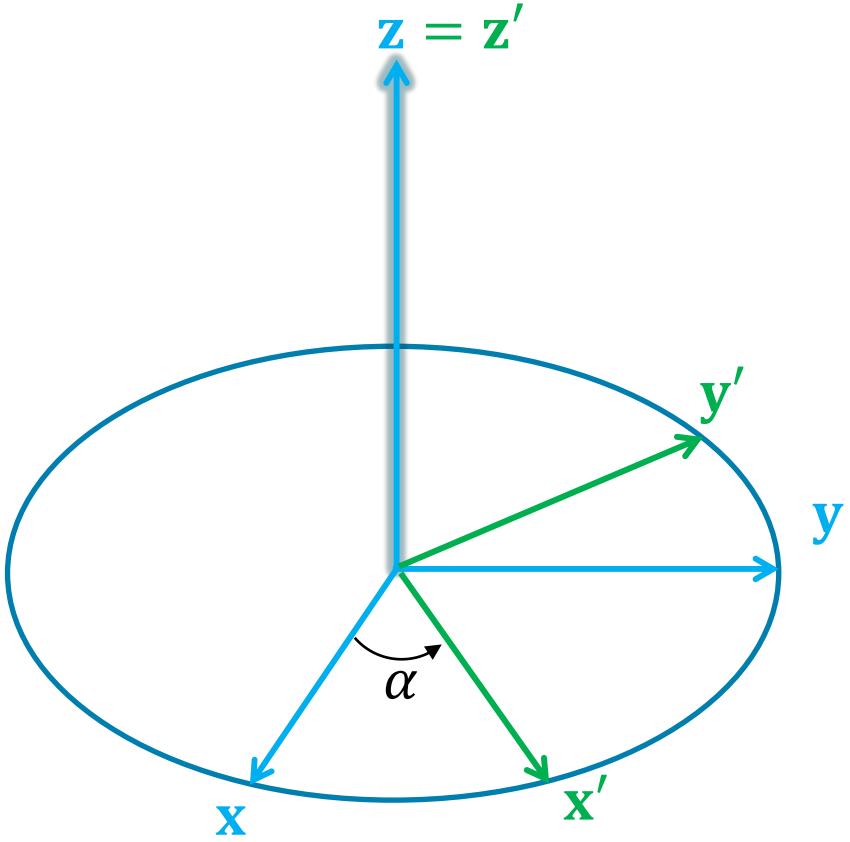
- sind die Rotationen um die x-, y- bzw. z-Achse ψ, θ und ϕ dann ist die Rotationsmatrix:

$$\mathbf{R} = \mathbf{R_z}(\phi)\mathbf{R_y}(\theta)\mathbf{R_x}(\psi) = \begin{pmatrix} \cos\theta\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \cos\theta\sin\phi & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi \\ -\sin\theta & \sin\psi\cos\theta & \cos\psi\cos\theta \end{pmatrix}$$

- die Winkel ψ, θ und ϕ heißen Euler- bzw. Kardanwinkel
  - … und beschreiben die Orientierung eines Objektes
  - … *zusammen mit* der Festlegung der Achsen und der Reihenfolge

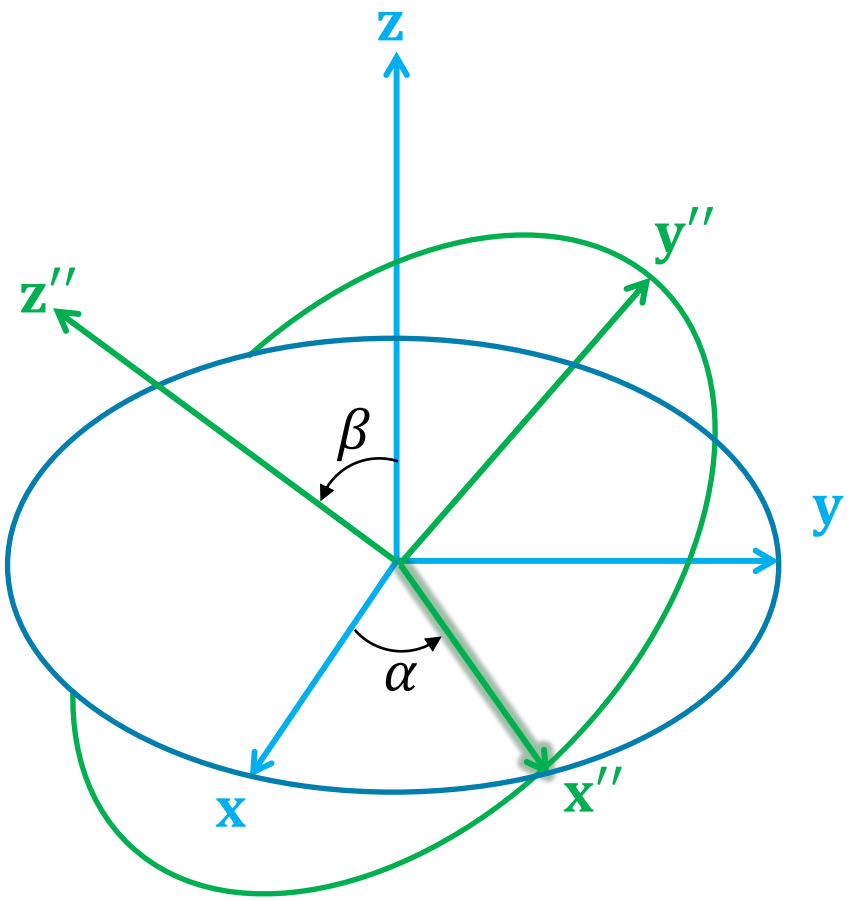
Euler Rotationen

- Rotation kann durch Eulerrotation $\mathbf{R} = \mathbf{R_z}(\gamma)\mathbf{R_x}(\beta)\mathbf{R_z}(\alpha)$ ausgedrückt werden, also Eulerwinkel α, β, γ für die Rotation um **z-x-z**



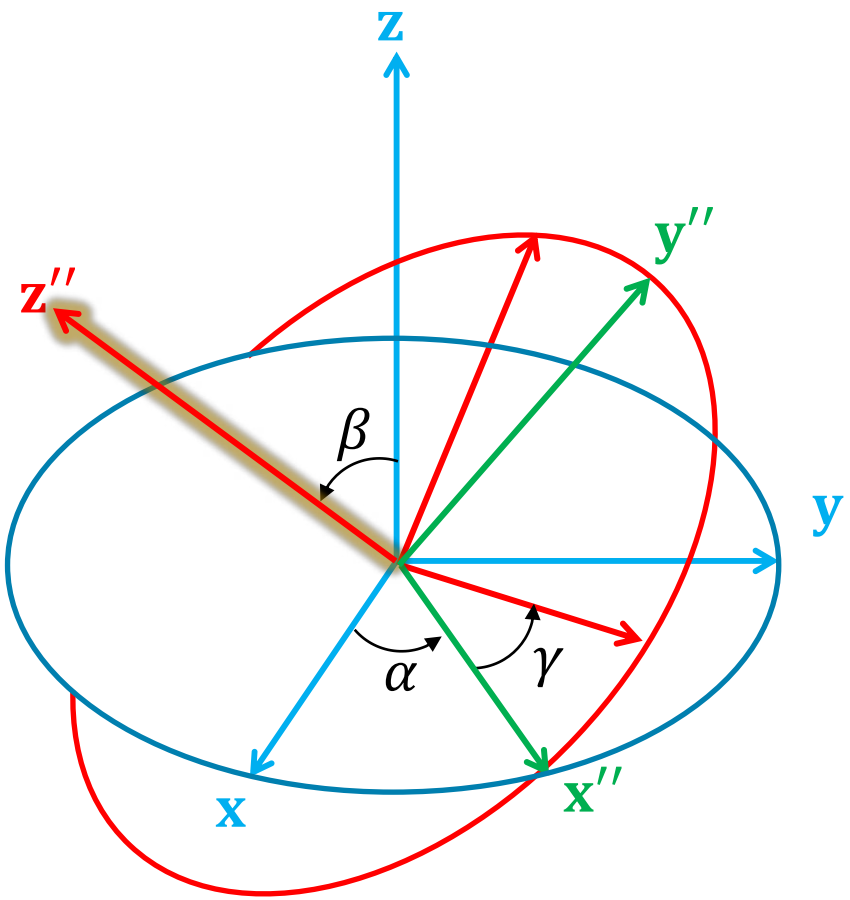
Euler Rotationen

- Eulerwinkel α, β, γ für die Rotation um **z-x-z**



Euler Rotationen

- Eulerwinkel α, β, γ für die Rotation um **z-x-z**



Euler Rotationen

- wenn eine Rotationsmatrix $\mathbf{R}$ und eine Achsenkonvention vorgegeben ist, können die Eulerwinkel abgelesen werden

- $\mathbf{R}_{\psi,\theta,\phi} = \mathbf{R}_\mathbf{z}(\phi)\mathbf{R}_\mathbf{y}(\theta)\mathbf{R}_\mathbf{x}(\psi) =$

$$\begin{pmatrix} \cos\theta\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \cos\theta\sin\phi & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi \\ -\sin\theta & \sin\psi\cos\theta & \cos\psi\cos\theta \end{pmatrix}$$

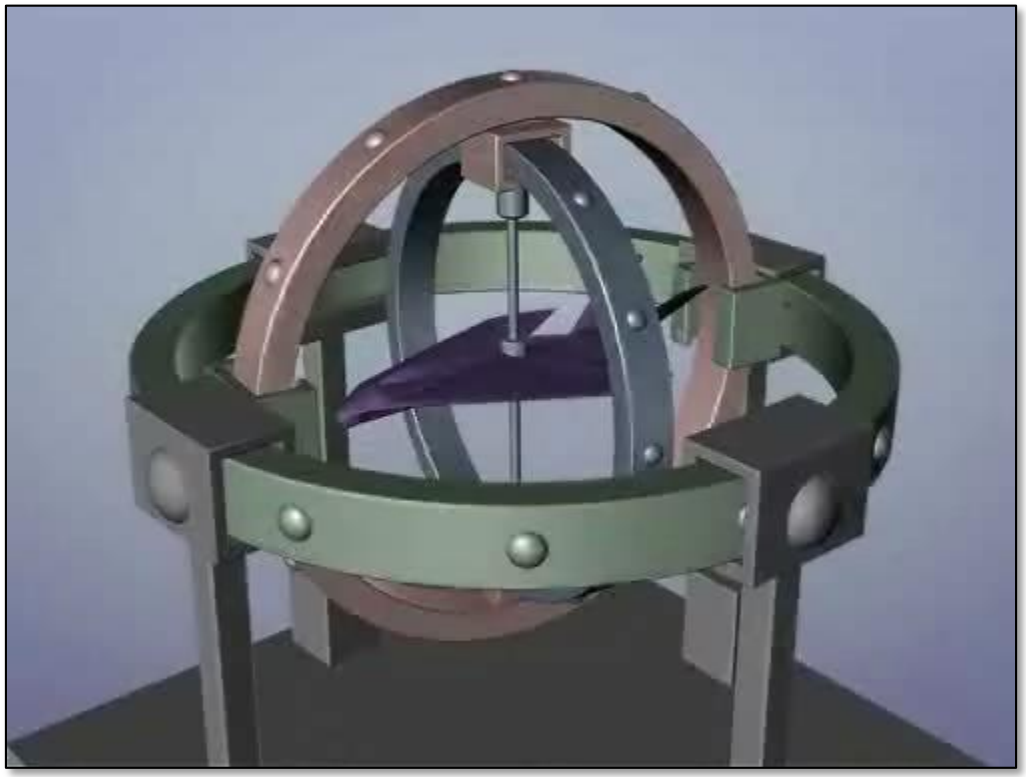
- $\mathbf{R} = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \Rightarrow R_{31} = -\sin\theta,\ \frac{R_{32}}{R_{33}} = \tan\psi,\ \frac{R_{21}}{R_{11}} = \tan\phi$

- Bemerkung:
  - die Lösung ist nicht eindeutig
  - es gibt Spezialfälle, z.B. $\cos\theta = 0$

Euler Rotationen

Kardanische Blockade (engl. Gimbal Lock)

- Problem bei der Eulerrotation: die 1. und 3. Rotationsachse können zusammenfallen, d.h. nur Summe aus 1. und 3. Winkel ist relevant und es fehlt ein Freiheitsgrad
- Anm. die Apparatur im Video nennt sich kardanische Aufhängung



http://www.youtube.com/watch?v=zc8b2Jo7mno

Inverse Transformationen

Matrixinversion

- inverse Matrix → inverse geometrische Transformation
- Matrixinversion im Allgemeinen kein triviales Problem
  - Adjungierte, Cramersche Regel, LU Zerlegung, Gauß-Elimination
  - ineffizient (im Vergleich zum Ansatz unten), u.U. numerische Probleme

- besser: Ausnutzen der Matrixeigenschaften wann immer möglich
  - $\mathbf{S}^{-1}(x,y,z) = \mathbf{S}\left(\frac{1}{x},\frac{1}{y},\frac{1}{z}\right)$
  - $\mathbf{R}^{-1}(\phi) = \mathbf{R}(-\phi)$ oder besser
    Orthogonalität ausnutzen: $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I} \Rightarrow \mathbf{R}^{-1} = \mathbf{R}^T$
  - $\mathbf{T}^{-1}(x,y,z) = \mathbf{T}\,(-x,-y,-z)$ (Translation mit Matrix? gleich mehr!)
  - zusammengesetzte Transformationen: $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$

- natürlich sind nicht alle Transformationen invertierbar
  (z.B. Projektionen, $\mathbf{S}(0,0,0)$)

Affine Abbildungen

- affine Abbildung:
  Kombination aus linearer Abbildung und Translation (hier in 2D)

$$\mathbf{x} \longmapsto \mathbf{A}\mathbf{x} + \mathbf{b} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

- Eigenschaften
  - Linien werden auf Linien abgebildet
  - parallele Linien bleiben parallel
  - teilverhältnistreu
  - nicht winkelerhaltend
  - Beispiele: Rotationen, Translationen, Skalierung, Scherung

Homogene Koordinaten

Projektiver Raum

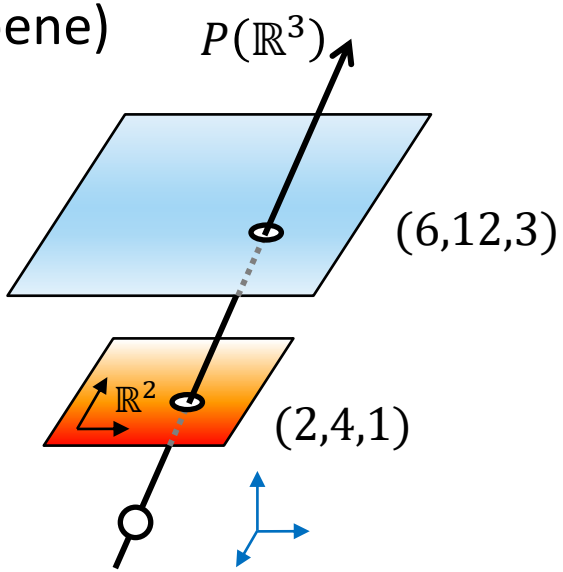
- parallele Geraden im affinen Raum schneiden sich nicht – in einer Projektion allerdings schon – wie formalisiert man das?
  - Geraden besitzen aber eine Richtung
  - Idee: wir ergänzen den euklidischen/affinen Raum (z.B. den $\mathbb{R}^3$ wie wir ihn uns vorstellen) um sog. Fernpunkte oder uneigentliche Punkte, die diese Richtungen beschreiben

- sogenannte „homogene Koordinaten“ werden verwendet um
  - affine Punkte und Richtungen zu beschreiben
  - Translationen durch Matrizen auszudrücken

Homogene Koordinaten

Homogene Koordinaten und projektive Räume (hier $\mathbb{R}^2$ und $P(\mathbb{R}^3)$)

- die Menge aller Geraden durch den Ursprung im $\mathbb{R}^3$ nennt man den reellen projektiven Raum $P(\mathbb{R}^3)$ (hier projektive Ebene)
  - ein Vektor $\mathbf{v} \in \mathbb{R}^3$ definiert eine Gerade
    - $\lambda\mathbf{v},\ \lambda \in \mathbb{R}\backslash\{0\}$ definieren dieselbe Gerade
    - Dimension $\dim\left(P(\mathbb{R}^3)\right) = 2$



- Repräsentation von Punkten:
  Einbettung des $\mathbb{R}^2$ in den $P(\mathbb{R}^3)$
  - $(x, y)_{2D} \rightarrow (x, y, 1)_h \equiv \left\{(x', y', w) \middle| \left(\frac{x'}{w}, \frac{y'}{w}\right) = (x, y)\right\}$
- Repräsentation von Richtungen
  - $(x, y)_{2D} \rightarrow (x, y, 0)_h$
- $P(\mathbb{R}^3)$ enthält also affine Punkte und Richtungen des $\mathbb{R}^2$ „ohne Fallunterscheidungen“

Affine Abbildungen

Homogene Koordinaten

- einfach: füge „1“ als 3. Koordinate (**homogene Koordinate**) hinzu:

$$(x, y)_{2D} \rightarrow (x, y, 1)_h$$

- homogene Koordinaten (etwas allgemeiner)
  - $(3x, 3y, 3)_h$, $\left(-\frac{x}{2}, -\frac{y}{2}, -\frac{1}{2}\right)_h$, $(ax, ay, a)_h$ mit $a \neq 0$ repräsentieren denselben Punkt im $\mathbb{R}^2$
  - **Dehomogenisierung**: $(x, y, w)_h \longrightarrow \left(\frac{x}{w}, \frac{y}{w}\right)_{2D}$

- Ziel: Beschreibung von affinen Punkten und Richtungen, sowie von affinen Abbildungen mit Matrizen
  - Vorteil: zusammengesetzte affine Transformationen

- mathematische Grundlagen: projektive Geometrie

Affine Abbildungen

Affine Abbildung bisher

$$\mathbf{x} \mapsto \mathbf{Ax} + \mathbf{b} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Affine Abbildung mit homogenen Koordinaten

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}_h \mapsto \begin{pmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}_h$$

$$= \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + b_1 \\ a_{21}x_1 + a_{22}x_2 + b_2 \\ 1 \end{pmatrix}_h \mapsto \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + b_1 \\ a_{21}x_1 + a_{22}x_2 + b_2 \end{pmatrix} =$$

$$= \begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \mathbf{Ax} + \mathbf{b}$$

Affine Abbildungen

- Anatomie einer affinen Transformation: linearer Teil und Translation aus dem $\mathbb{R}^2$ werden zu linearer Abbildung in homogenen Koordinaten

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{pmatrix}$$

- zusammengesetzte Transformationen

$$\mathbf{x} \overset{\mathbf{T}}{\rightarrow} \mathbf{Tx} = \mathbf{y} \overset{\mathbf{S}}{\rightarrow} \mathbf{Sy} = \mathbf{z} \quad \equiv \quad \mathbf{z} = \mathbf{S} \cdot \mathbf{T} \cdot \mathbf{x}$$

  - Zusammensetzen ≡ Multiplikation
  - inverse Transformation ≡ inverse Matrix

- **wichtig**: $\mathbf{M}$ und $\lambda\mathbf{M}$ ($\lambda \neq 0$) beschreiben bei homogenen Koordinaten dieselbe Abbildung

2D Transformationen und homogene Koordinaten

Grundlegende 2D Transformationen

- Translation eines (Orts-)Vektors

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}_h = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}_h = \begin{pmatrix} x + \Delta x \\ y + \Delta y \\ 1 \end{pmatrix}_h$$

- Translation verändert Richtungsvektoren nicht

$$\begin{pmatrix} x' \\ y' \\ 0 \end{pmatrix}_h = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}_h = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}_h$$

- Achtung: Transformation von Normalen erfordert spezielle Behandlung (Normale ist ein sog. Bivektor, der senkrecht auf einer Fläche steht)
- im Folgenden lassen wir das „h“ weg – es ist i.d.R. aus dem Kontext klar, wann es sich um homogene Koordinaten handelt

Affine Abbildungen

Beispiel

- Rotation in 2D um den Punkt $\mathbf{c} = (c_x, c_y)$ und den Winkel ϕ
- Schritte: $\mathbf{T}(-\mathbf{c}) \rightarrow \mathbf{R_z}(\phi) \rightarrow \mathbf{T}(\mathbf{c})$

$$\begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{pmatrix}$$

3D Transformationen und homogene Koordinaten

Grundlegende 3D Transformationen

- Skalierung und Scherung (z unverändert)

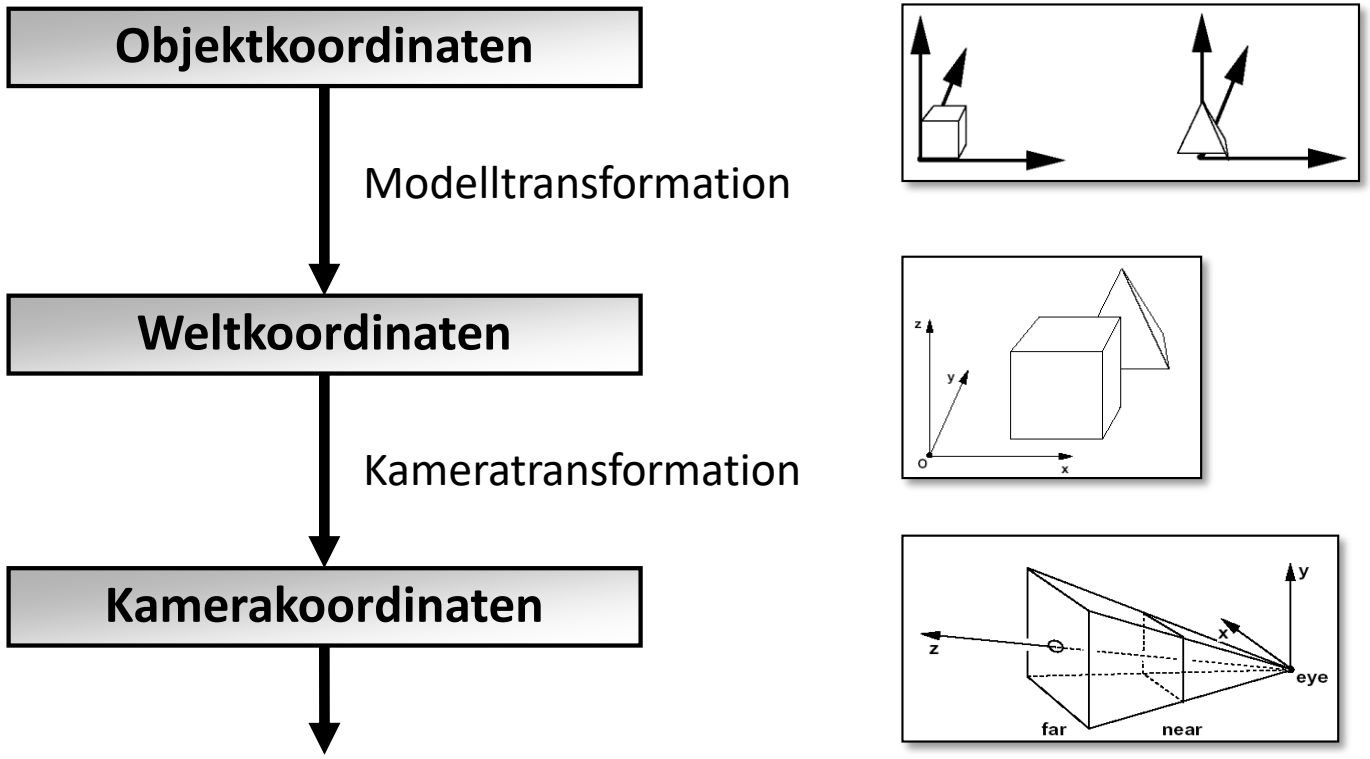
$$scale(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad shear_z(d_x, d_y) = \begin{pmatrix} 1 & 0 & d_x & 0 \\ 0 & 1 & d_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotation um x-, y- und z-Achse

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_y(\phi) = \begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Koordinatensysteme in der CG

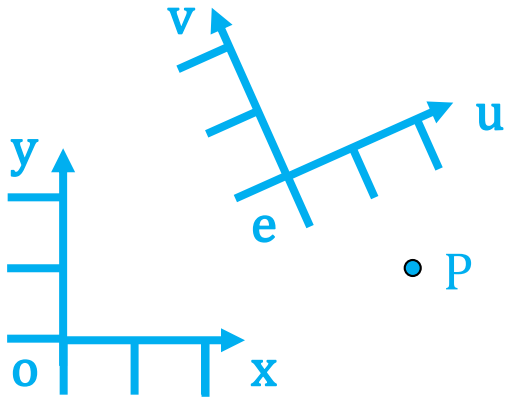
- Objekte in einer Szene werden zur Modellierung (Beschreibung) in ihrem eigenen **Objekt- oder Modell-Koordinatensystem** angegeben
- die Platzierung der Objekte im **Weltkoordinatensystem** erfolgt dann durch Translation, Rotation, Skalierung etc.
- dann erfolgt – beim Raytracing implizit – die Transformation in das **Kamerakoordinatensystem** oder der **Sichtstrahlen in Weltkoordinaten**



Wechsel zwischen Koordinatensystemen

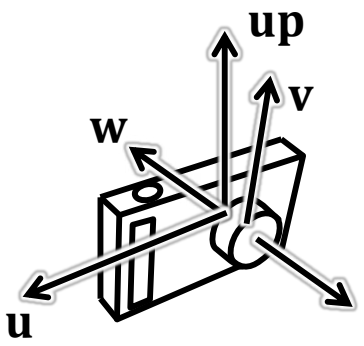
- globales/Welt-KoSys: Ursprung $\mathbf{o}$, Basisvektoren $\mathbf{x}$, $\mathbf{y}$ (nicht explizit gespeichert) ist Bezugssystem für lokale Koordinatensysteme
- lokales/Modell-KoSys: Ursprung $\mathbf{e}$ und Basisvektoren $\mathbf{u}$, $\mathbf{v}$
  - z.B. Position eines Autos in einer virtuellen Szene

- Lage des Punktes P: $(p_x, p_y) = (5,1)$ bzw. $(p_u, p_v) = (1,-2)$
- ausgedrückt durch die Basisvektoren:
  $\mathrm{P} = (p_x, p_y) = \mathbf{e} + p_u\mathbf{u} + p_v\mathbf{v}$
- Wechsel der Koordinatensysteme
  - $\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & e_x \\ 0 & 1 & e_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_x & v_x & 0 \\ u_y & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_u \\ p_v \\ 1 \end{pmatrix}$
  - $\begin{pmatrix} p_u \\ p_v \\ 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -e_x \\ 0 & 1 & -e_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix}$

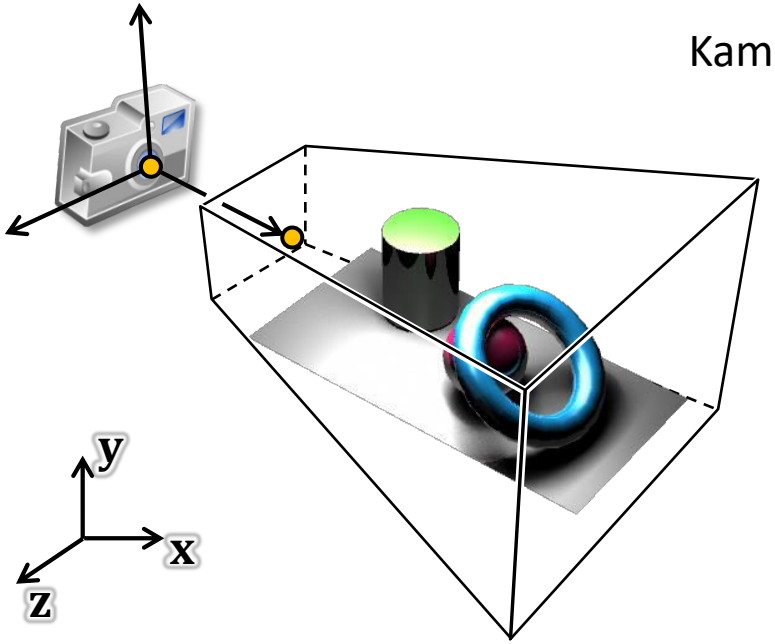


Kameratransformation

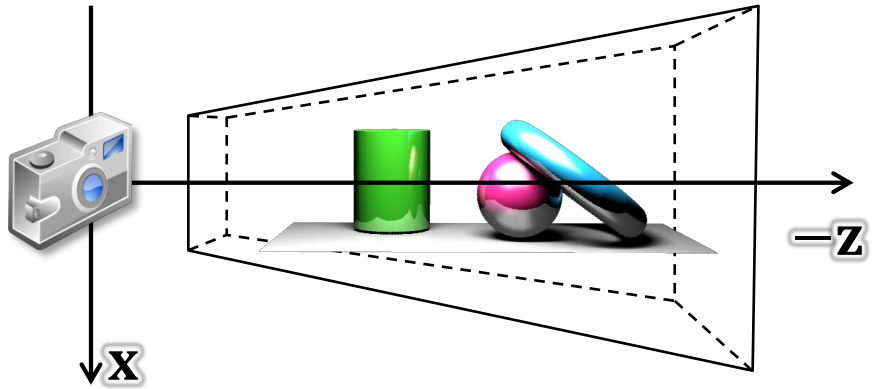
- virtuelle Kamera definiert durch
  - Position $\mathbf{e}$ und (negative) Blickrichtung $\mathbf{w}$
  - „Up-Vektor“ $\mathbf{up}$
  - $\Rightarrow \mathbf{u} = \mathbf{up} \times \mathbf{w}$ und $\mathbf{v} = \mathbf{w} \times \mathbf{u}$
  - $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ bilden Basis des Kamera-Koordinatensystems



- Transformation in dieses Koordinatensystem erleichtert bestimmte nachfolgende Schritte…



Kameratransformation



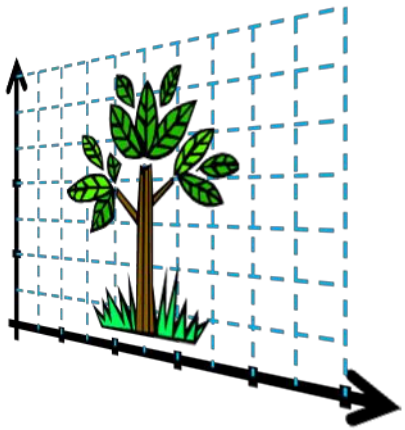
Kamera und Sichtpyramide

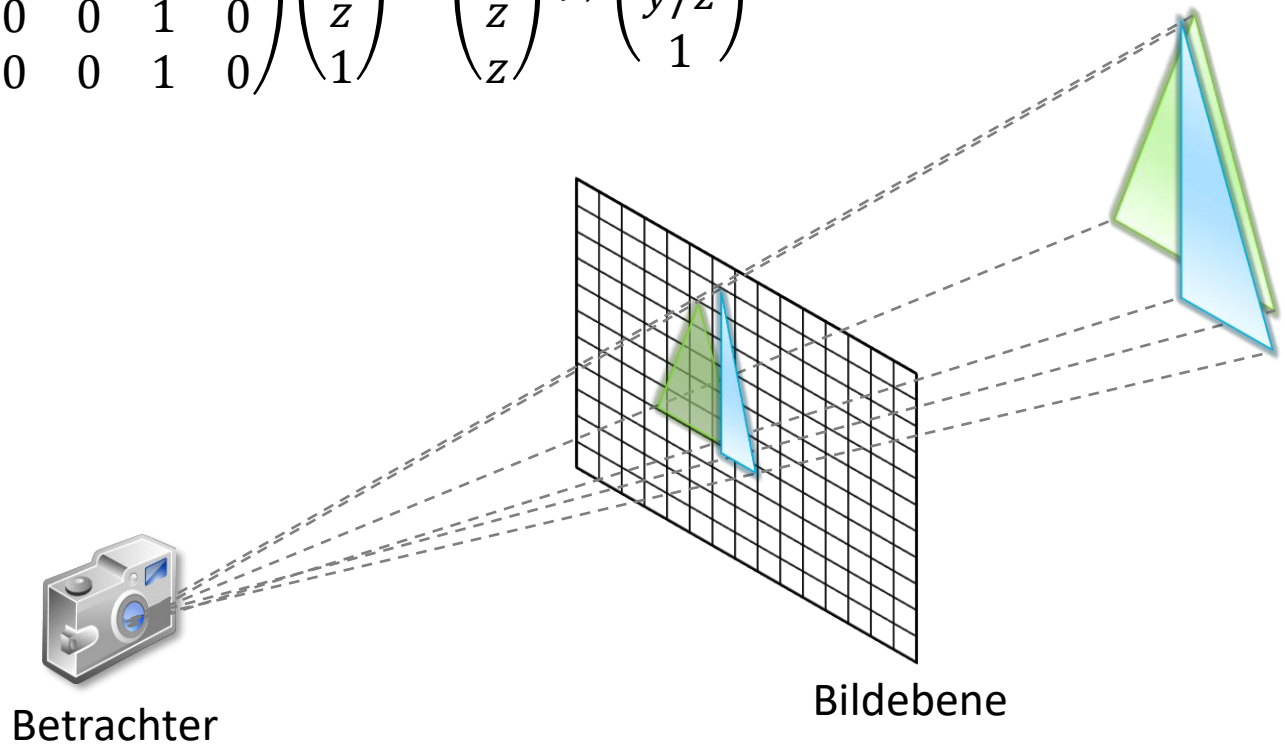
nach der Kameratransformation

Projektive Abbildungen

- … wie beispielsweise die Projektion auf die Bildebene
- projektive Abbildungen besprechen wir später im Kontext von Rasterisierung mit Grafik-Hardware
- Bsp. Projektion auf $z = 1$ Ebene:

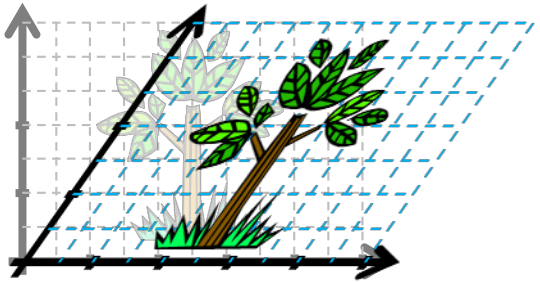
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z \end{pmatrix} \mapsto \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix}$$



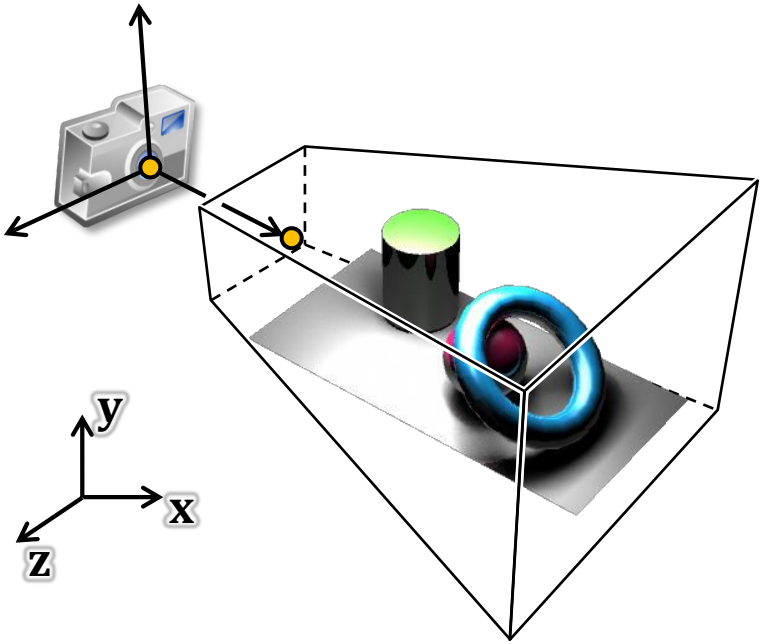


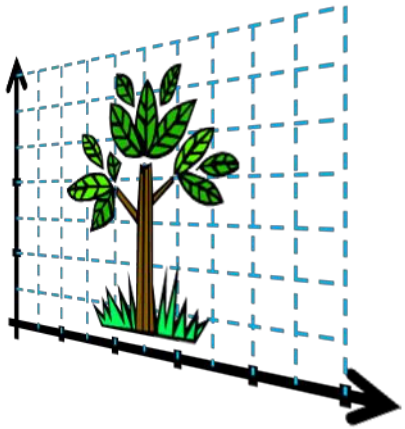
Inhalt

- ...
- Homogene Koordinaten
- Koordinatensysteme in der CG
  und hierarchische Modellierung
- Transformation von Normalenvektoren
- Schnitttests in lokalen Koordinaten



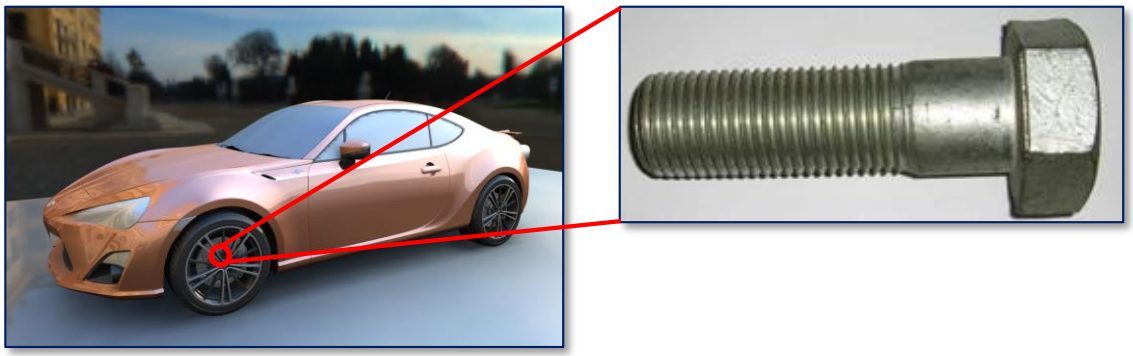
$$\begin{pmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{pmatrix}$$





Hierarchisches Modellieren

- Modelltransformationen sind typischerweise zusammengesetzte Transformationen (Rotation, Skalierung, Translation)
- aber: es macht die Modellierung komplexer Szenen einfacher, wenn man
  - mehrfache Kopien („Instanzen“) von Objekten erstellen kann,
  - Objekte zu Gruppen zusammenfasst und
  - Gruppen anordnet
- Beispiel: Modell eines Autos
  - ein Auto besteht aus Karosserie und 4 Rädern
  - jedes Rad besteht aus Reifen, Felge und 5 Schrauben
  - wollen Sie die Schraube 20 mal modellieren? Das Rad 4 mal? Oder die Geometrie mehrfach speichern?



Beispiel für Instancing: Rendering von Vegetation

- https://www.youtube.com/watch?v=AdabPy6xQ8A

Bild: Unreal Engine 4 Architectural Visualization

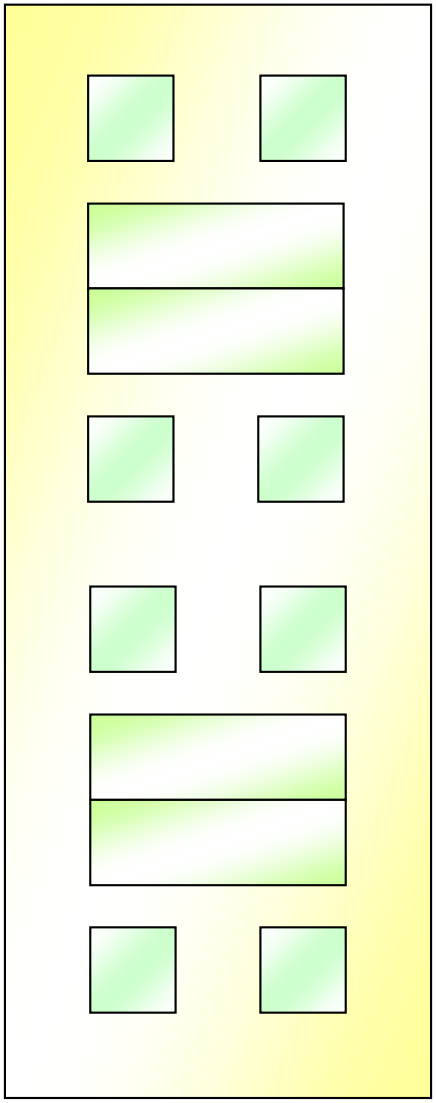
Beispiel für Instancing: Rendering von Vegetation

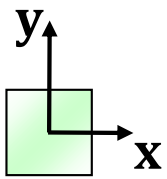
- große Daten bei Filmproduktionen
  - Geometrie oft 30+ Gigabytes (oft animiert), 200 GB – 2 TB Texturen

Jungle scene from Kingsman: Golden Circle. © 2017, 20th Century Fox. All rights reserved.
Sony Pictures Imageworks Arnold, Kulla et al. 2018

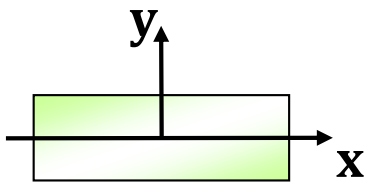
Hierarchisches Modellieren

Beispiel: Platzierung von Stühlen und Tischen in einem Raum

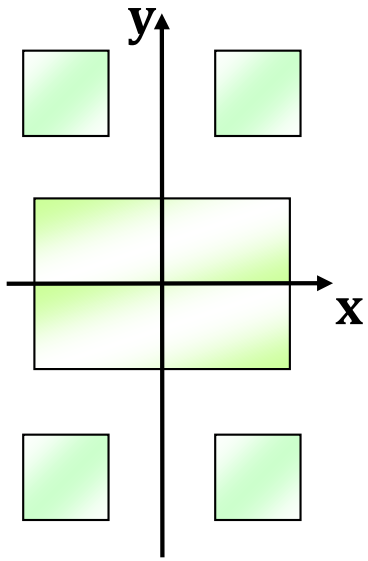




Stuhl



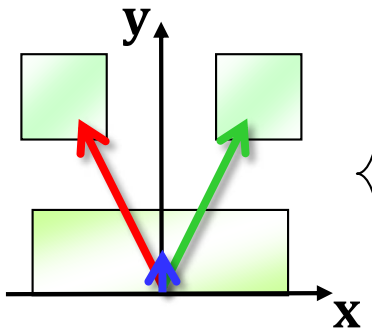
Tisch



Identitätsabb.
Double

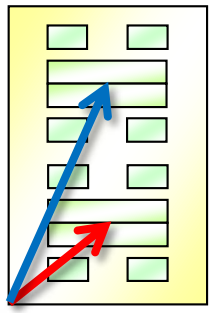
$R_z(180°)$
Double

4er Combo



T_{S1}
T_{S2}
T_T

Double



T_{C1} 4er Combo

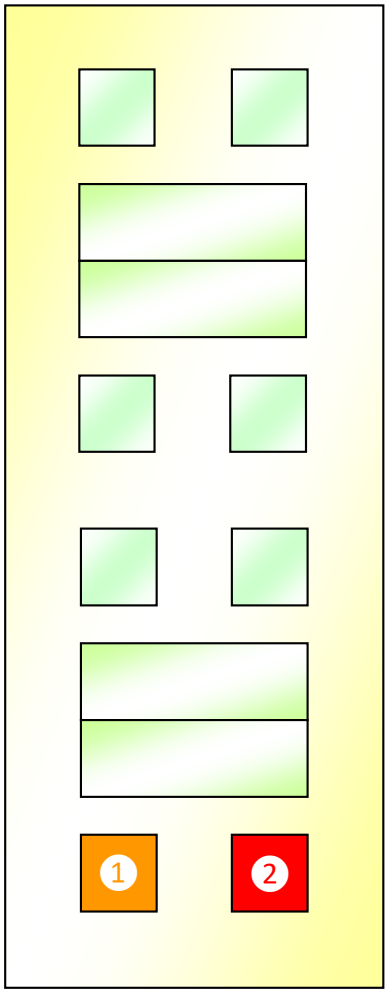
T_{C2} 4er Combo

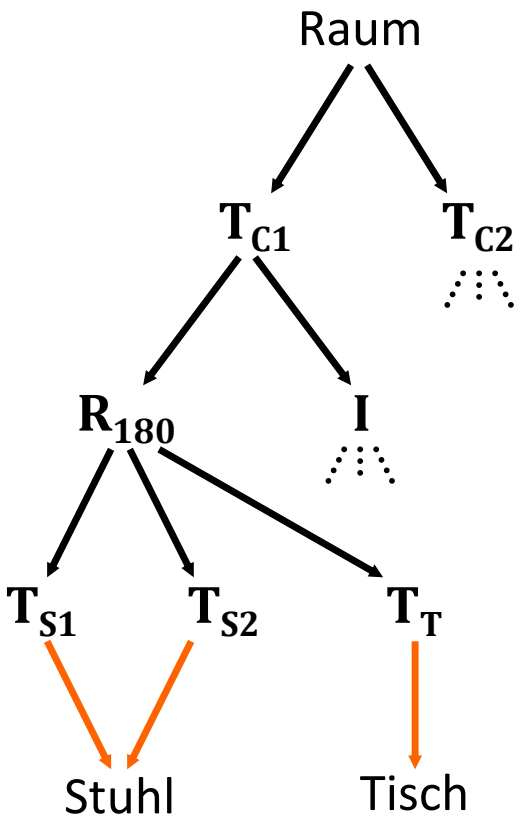
Raum

Szenengraphen: Hierarchisches Modellieren

Beispiel: Platzierung von Stühlen und Tischen in einem Raum

- es entsteht ein **Szenengraph** (gerichteter azyklischer Graph)





Transformationen des Stuhls 1 in Weltkoordinaten:

$(T_{C1}\ R_{180}\ T_{S2})$

Transformationen des Stuhls 2 in Weltkoordinaten:

$(T_{C1}\ R_{180}\ T_{S1})$

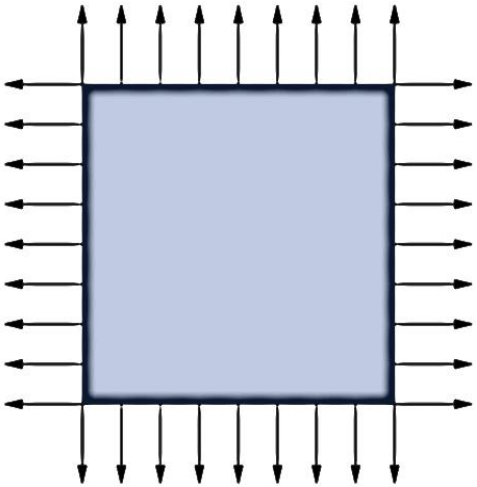
Szenengraphen: Matrix Stacks

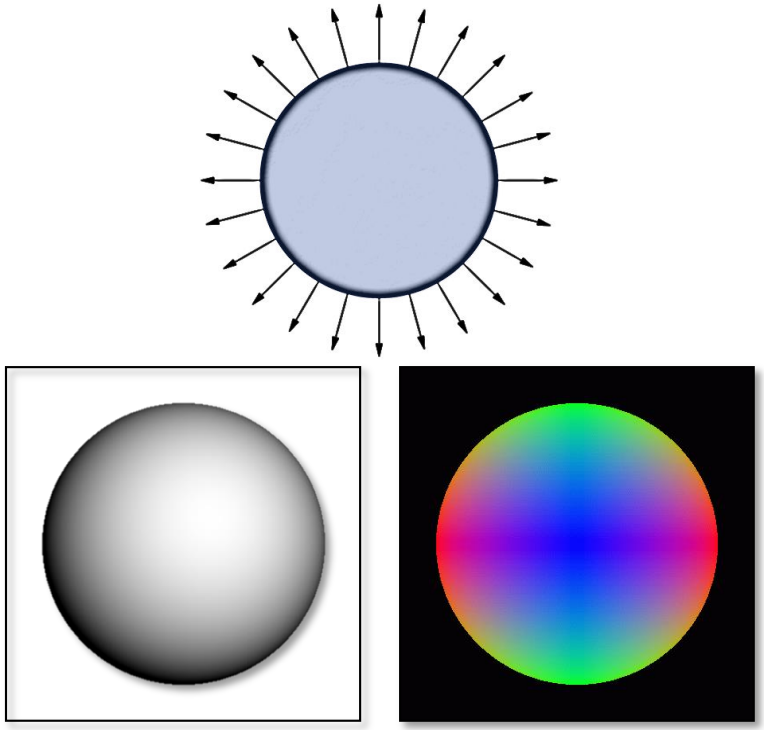
- Szenengraphen erlauben effiziente Operationen in komplexen Szenen
  - u.a. die Wiederverwendung von Matrizen mittels **Matrix Stacks**
  - die jeweils oberste Matrix definiert die aktuelle Transformation
  - spannender sind sog. Culling-Operationen und andere Optimierungen auf Szenengraphen (siehe InCG im Sommersemester)

Load($\mathbf{I}$)	Push	Mult($\mathbf{T_{C1}}$)	Push	Mult($\mathbf{R_{180}}$)	Push	Mult($\mathbf{T_{S1}}$)	Stuhl	Pop	...
$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I\ T_{C1}}$	$\mathbf{I\ T_{C1}}$	$\mathbf{I\ T_{C1}\ R}$	$\mathbf{I\ T_{C1}\ R}$	$\mathbf{I\ T_{C1}\ R\ T_{S1}}$		$\mathbf{I\ T_{C1}\ R}$	
	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I\ T_{C1}}$	$\mathbf{I\ T_{C1}}$	$\mathbf{I\ T_{C1}\ R}$	$\mathbf{I\ T_{C1}\ R}$		$\mathbf{I\ T_{C1}}$	
			$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I\ T_{C1}}$	$\mathbf{I\ T_{C1}}$		$\mathbf{I}$	
					$\mathbf{I}$	$\mathbf{I}$			

Transformation von Normalen

- Oberflächennormale/Normalenvektor:
  Einheitsvektor lokal-senkrecht zur Oberfläche
  - wichtig für Schattierung/Beleuchtung
  - hier: verschiedene Visualisierungen

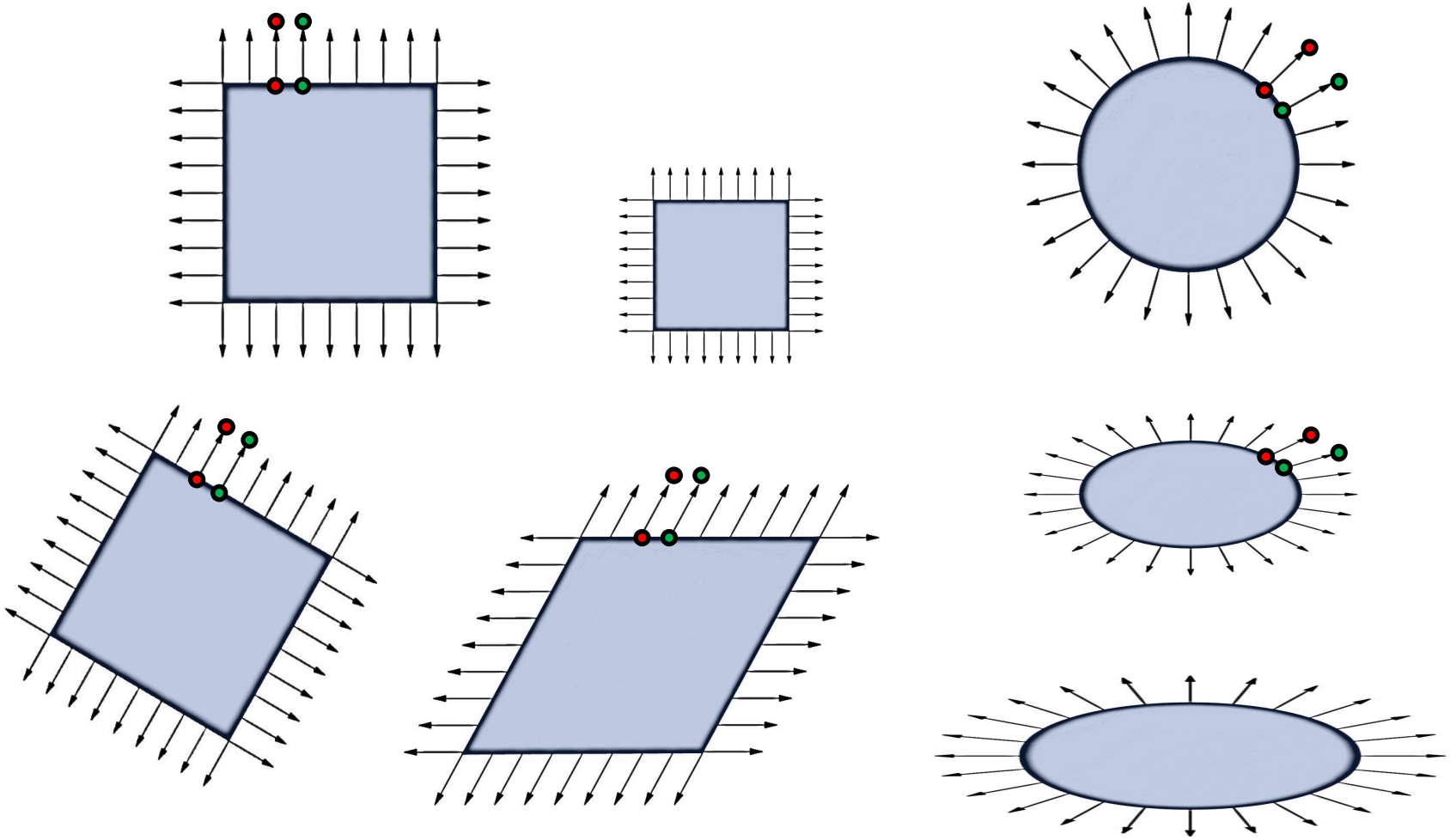




±x, ±y, ±z =
rot/grün/blau

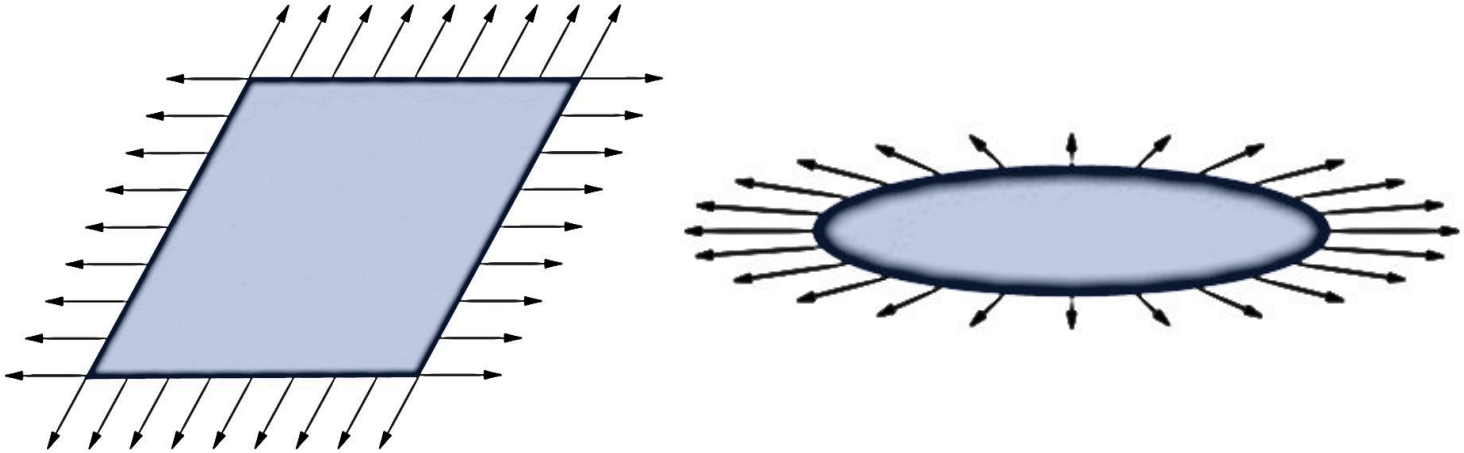
Transformation der Normalen wie Objekte?

- Translation, Rotation, (isotr.) Skalierung, Spiegelung, Scherung, …?

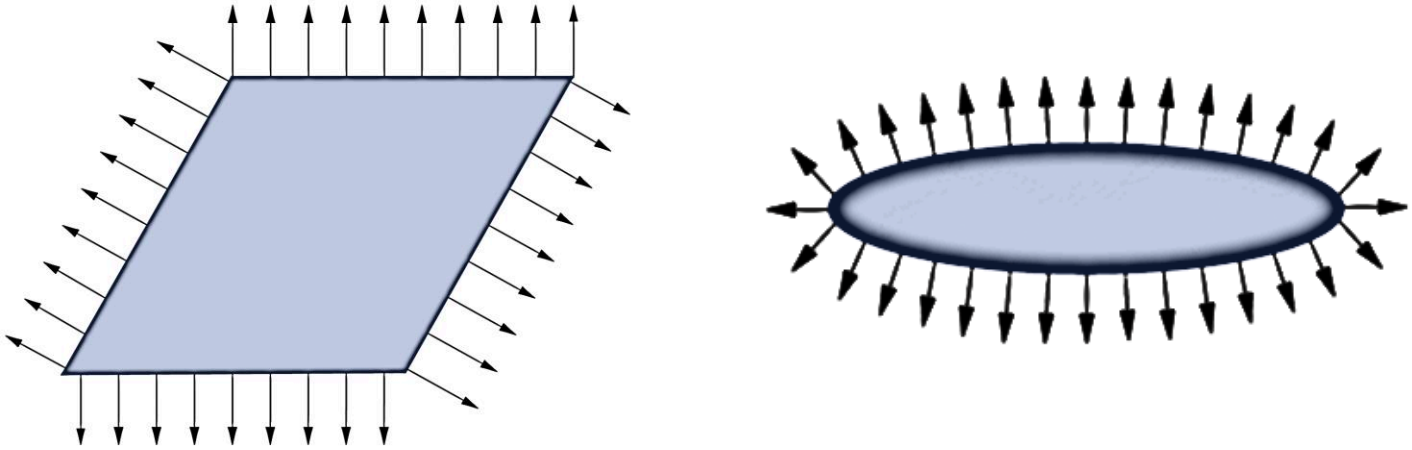


Transformationen für Scherung und Skalierung

- falsche Transformation der Normalenvektoren

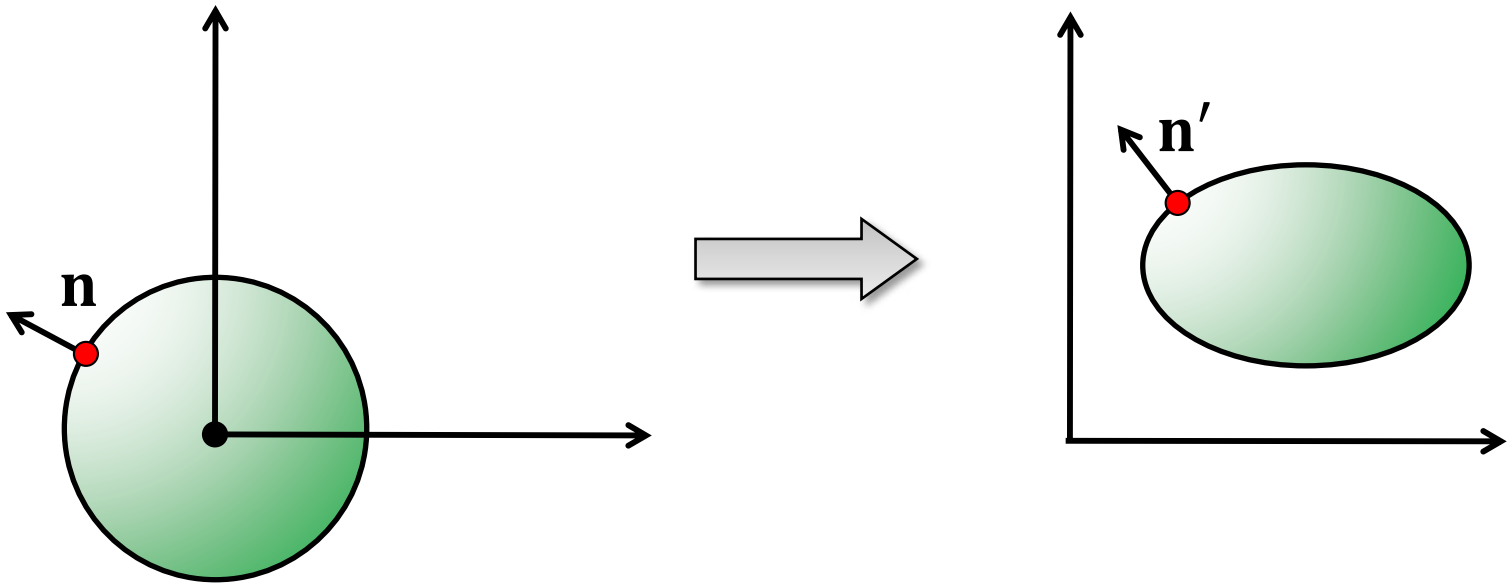


- korrekte Transformation der Normalenvektoren



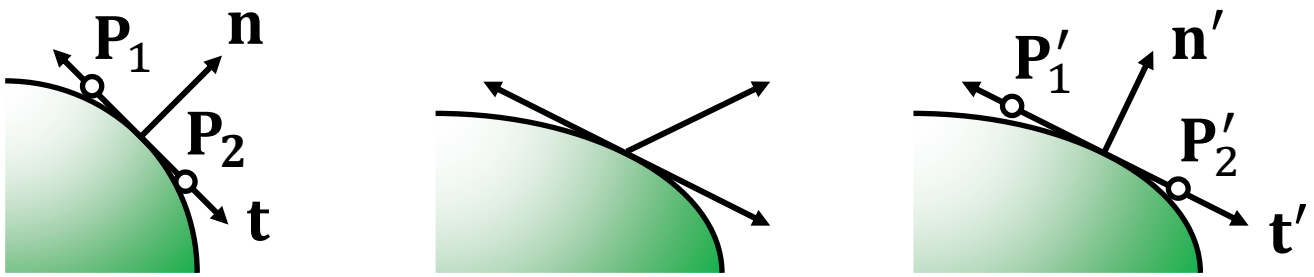
Transformation von Normalen

- Normalen sind sog. Bivektoren: sie stehen senkrecht auf der Tangentialfläche und nicht durch Differenz zweier Ortsvektoren definiert
- lineare und affine Transformationen sind i.A. nicht winkeltreu
  → Normalen können nicht einfach mit-transformiert werden
- wie macht man es also richtig?



Transformation von Normalen

- wir betrachten zwei Punkte $\mathbf{P}_1, \mathbf{P}_2$ auf der Tangentialebene zur Normale
- die Normale muss vor und nach der Transformation senkrecht auf dem Differenzvektor stehen

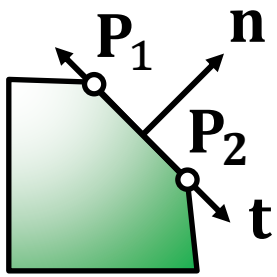


Original | falsch transformiert | korrekt transformiert

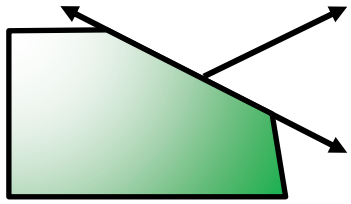
- seien $\mathbf{P}_1$ und $\mathbf{P}_2$ zwei Punkte in der Tangentialebene und $\mathbf{r} = \mathbf{P}_2 - \mathbf{P}_1$
- es gilt: $\mathbf{n}^T \cdot \mathbf{r} = 0$ einsetzen: $\mathbf{r} = (\mathbf{M}^{-1}\mathbf{M})\mathbf{r}$
  - $\mathbf{n}^T(\mathbf{M}^{-1}\mathbf{M})\mathbf{r} = 0$
  - $\underbrace{(\mathbf{n}^T\mathbf{M}^{-1})}_{\mathbf{n}'^T} \cdot \underbrace{(\mathbf{M}\mathbf{r})}_{\mathbf{r}'} = 0$
  - wenn $\mathbf{n}'^T = (\mathbf{n}^T\mathbf{M}^{-1}) \Rightarrow \mathbf{n}' = (\mathbf{n}^T\mathbf{M}^{-1})^T = (\mathbf{M}^{-1})^T\mathbf{n}$

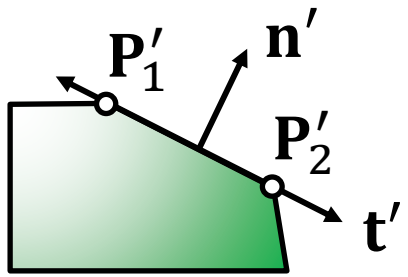
Transformation von Normalen

- wir betrachten zwei Punkte $\mathbf{P}_1$, $\mathbf{P}_2$ auf der Tangentialebene zur Normale
- die Normale muss vor und nach der Transformation senkrecht auf dem Differenzvektor stehen



Original


falsch transformiert



korrekt transformiert

- seien $\mathbf{P}_1$ und $\mathbf{P}_2$ zwei Punkte in der Tangentialebene und $\mathbf{r} = \mathbf{P}_2 - \mathbf{P}_1$
- es gilt: $\mathbf{n}^T \cdot \mathbf{r} = 0$ einsetzen: $\mathbf{r} = (\mathbf{M}^{-1}\mathbf{M})\mathbf{r}$
  - $\mathbf{n}^T(\mathbf{M}^{-1}\mathbf{M})\mathbf{r} = 0$
  - $\underbrace{(\mathbf{n}^T\mathbf{M}^{-1})}_{\mathbf{n}'^T} \cdot \underbrace{(\mathbf{M}\mathbf{r})}_{\mathbf{r}'} = 0$
  - wenn $\mathbf{n}'^T = (\mathbf{n}^T\mathbf{M}^{-1}) \Rightarrow \mathbf{n}' = (\mathbf{n}^T\mathbf{M}^{-1})^T = (\mathbf{M}^{-1})^T\mathbf{n}$

Transformation von Normalen

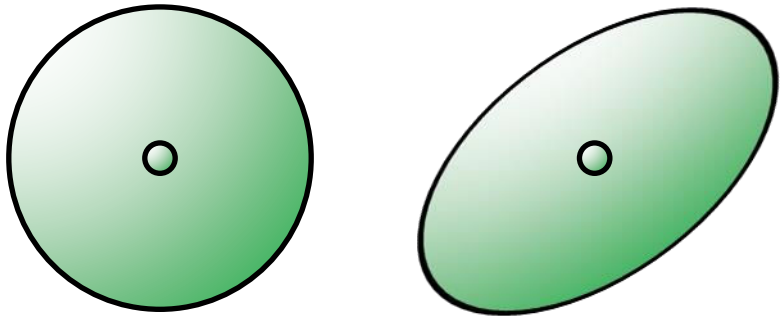
- Transformation des Normalenvektors mit der invers-transponierten Matrix: $\mathbf{n}' = (\mathbf{M}^{-1})^T \mathbf{n}$
  - beachte: es ist möglich das $\|\mathbf{n}'\| \neq 1$, z.B. bei Skalierungen oder Scherungen (d.h. nur Richtung von $\boldsymbol{n}'$ ist korrekt)

- warum „funktioniert“ $\mathbf{n}' = \mathbf{M}\mathbf{n}$ für Ähnlichkeitsabbildungen (= euklidische Transformation plus Skalierung)?
  - es gilt in diesem Fall: $(\mathbf{M}^{-1})^T = \lambda \mathbf{M}$ (erhält zumindest die Richtung)
  - bei einer Matrix mit orthonormaler Basis gilt $(\mathbf{M}^{-1})^T = \mathbf{M}$, also $\lambda = 1$

Transformationen und Schnitttests

Schnittpunktberechnung in Modell- oder Weltkoordinaten?

- Modelltransformationen in das Weltkoordinatensystem:
  - Rotation, isotrope Skalierung, Translation sind einfach
    - Dreieck: Transformation der 3 Eckpunkte
    - Kugel: Veränderung des Kugelmittelpunkts und -radius
  - Scherung, allgemeine Skalierung
    - Dreieck: kein Problem, Transformation der Eckpunkte
    - Kugel: implizite Beschreibung einer gescherten Kugel?

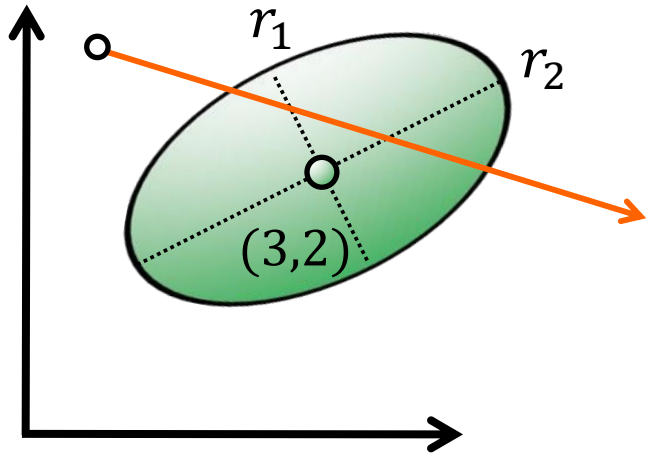


- bessere Alternative: **Schnittpunktberechnung in Modellkoordinaten**

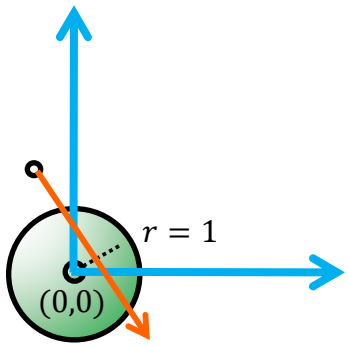
Transformationen und Schnitttests

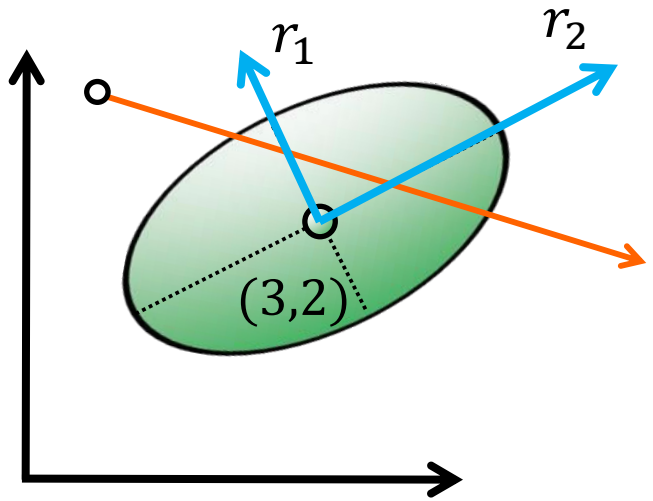
Zwei Möglichkeiten

- Schnittpunktberechnung in Weltkoordinaten:
  jedes Primitiv behandelt seine Transformationen selbst,
  oft (unnötig) kompliziert
  (aber gut bei statischen Dreiecksnetzen ohne Instanziierung)



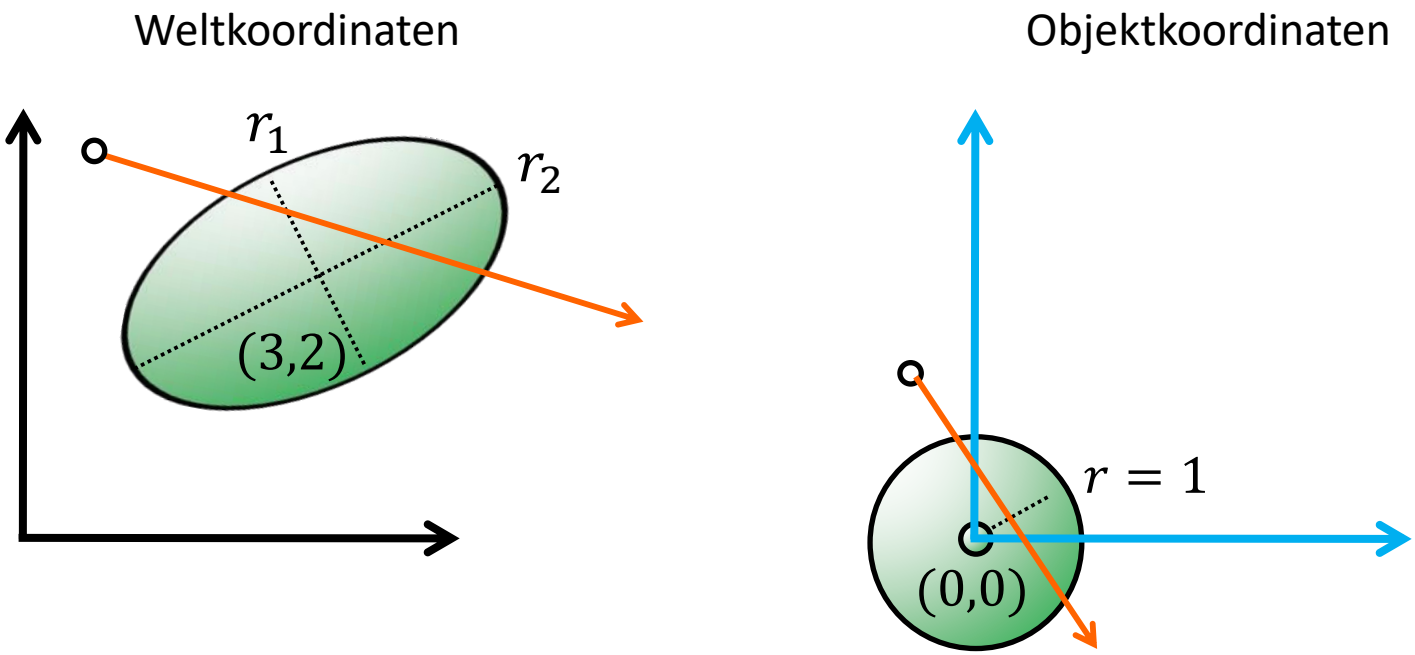
- Schnittpunktberechnung in Modellkoordinaten:
  gut, wenn viele Primitive sich eine Transformation teilen, z.B. ein instanziiertes Dreiecksnetz





Transformation des Strahls

- transformiere den Strahl von Welt- in Objektkoordinaten



- involvierte Transformationen
  - in Weltkoordinaten (world space): $\mathbf{p_{ws}} = \mathbf{M}\mathbf{p_{os}}$
  - in Objektkoordinaten (object space): $\mathbf{p_{os}} = \mathbf{M}^{-1}\mathbf{p_{ws}}$

Transformation des Strahls

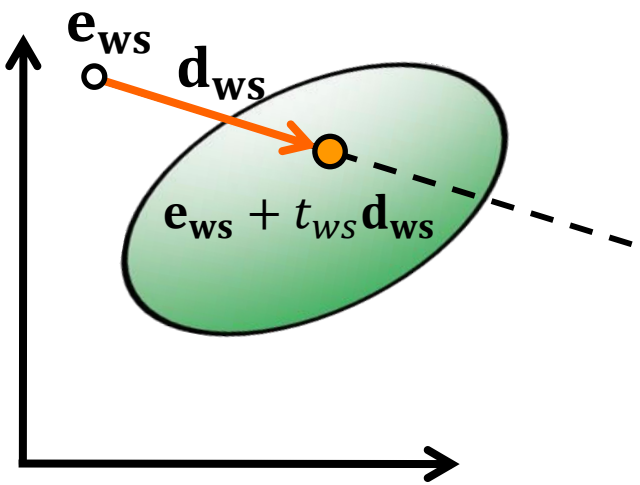
- neuer Strahlursprung

$$\mathbf{e_{os}} = \mathbf{M}^{-1}\mathbf{e_{ws}}$$

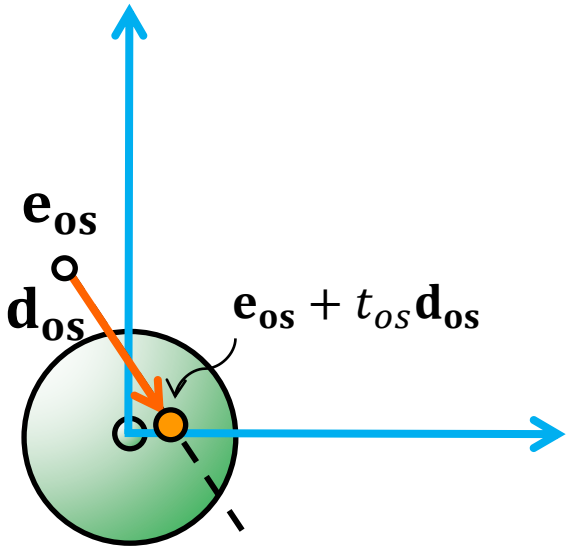
- neue Strahlrichtung (= wie eben Differenz zweier Punkte, daher dürfen wir hier direkt die inverse Matrix verwenden)

$$\mathbf{d_{os}} = \mathbf{M}^{-1}\mathbf{d_{ws}}$$

Weltkoordinaten

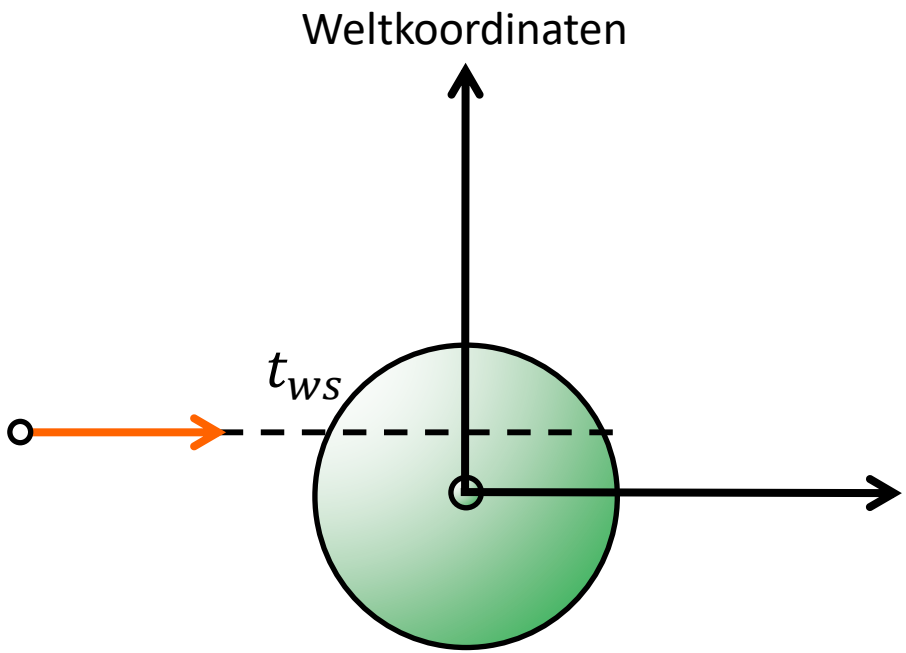


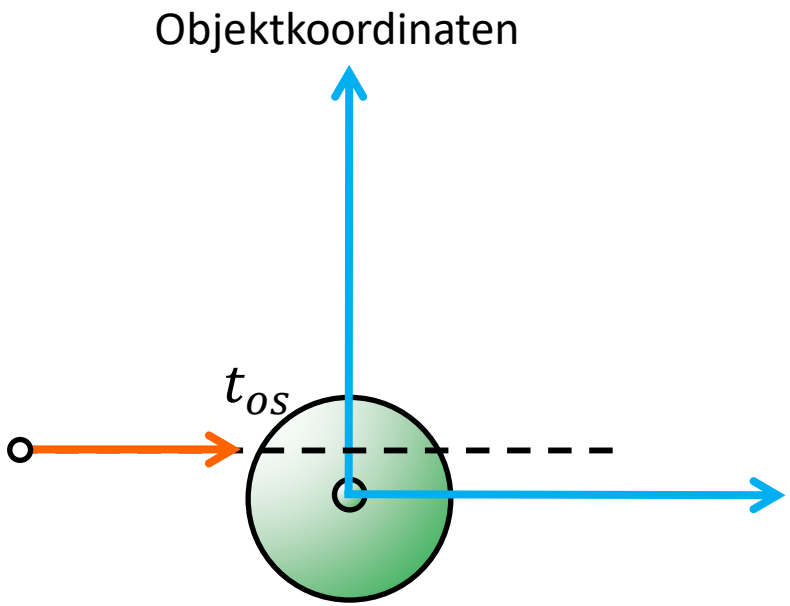
Objektkoordinaten



Transformation des Strahls

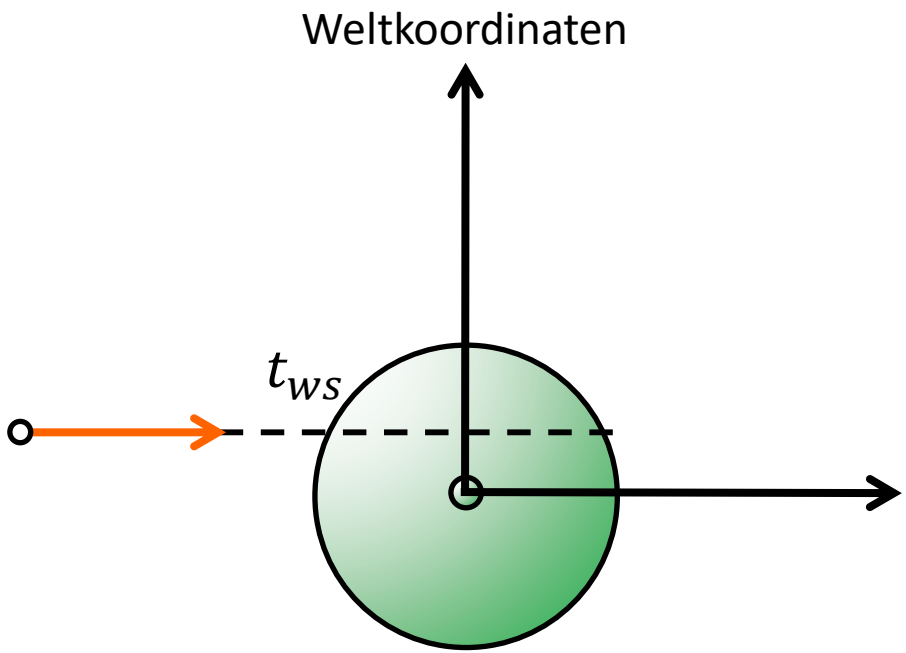
- $\mathbf{d_{os}}$ ist i.A. nicht normalisiert, z.B. wenn $\mathbf{M}$ eine Skalierung enthält
- wieder zwei Optionen
  - normalisiere $\mathbf{d_{os}}$, also verwende $\mathbf{d_{os}}/|\mathbf{d_{os}}|$
  - normalisiere nicht und verwende $\mathbf{d_{os}}$ für die Schnittberechnung
- Fall 1: verwende $\mathbf{d_{os}}/|\mathbf{d_{os}}|$
  - $t_{ws} \neq t_{os}$ (Skalierung nach der Schnittberechnung notwendig)

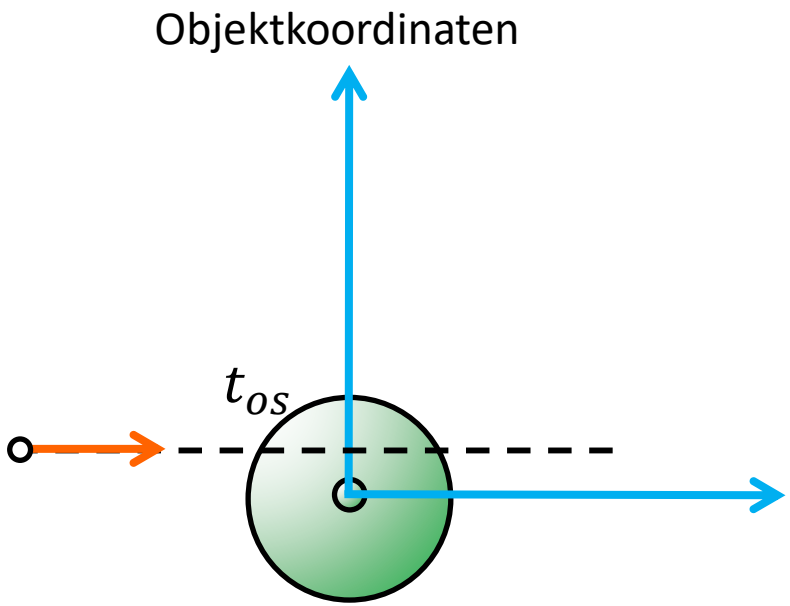




Transformation des Strahls

- $\mathbf{d_{os}}$ ist i.A. nicht normalisiert, z.B. wenn $\mathbf{M}$ eine Skalierung enthält
- wieder zwei Optionen
  - normalisiere $\mathbf{d_{os}}$, also verwende $\mathbf{d_{os}}/|\mathbf{d_{os}}|$
  - normalisiere nicht und verwende $\mathbf{d_{os}}$ für die Schnittberechnung
- Fall 2: verwende $\mathbf{d_{os}}$ (wenn Schnittberechnung es zulässt)
  - $t_{ws} = t_{os}$





Transformationen und Schnitttests

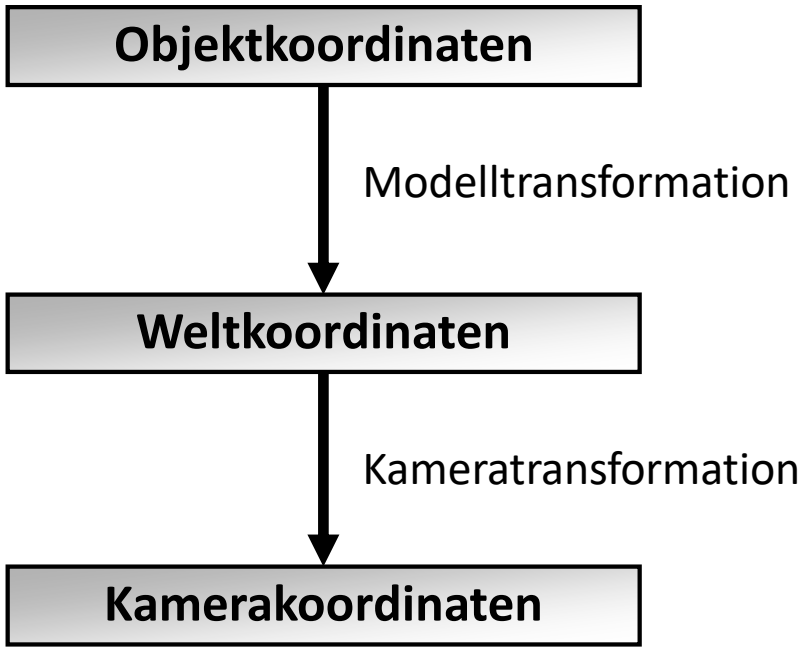
Fazit

- ist (implizite) Beschreibung des mit $\mathbf{M}$ transformierten Objekts schwierig → transformiere den Strahl mit $\mathbf{M}^{-1}$ und führe Schnitttest mit der ursprünglichen Beschreibung durch
- meist günstiger als die Objekte zu transformieren
- einfacher/effizienter, wenn Instanziierung verwendet wird
- **wichtigster Fall in der Praxis: einmalige Strahltransformation für viele Objekte/Objektgruppen, z.B. ein instanziiertes Dreiecksnetz**

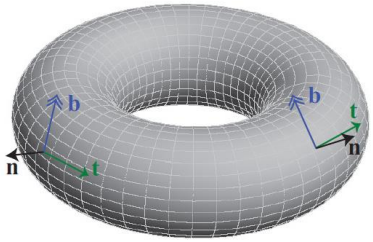
- Kosten der Matrixinversion und Strahltransformation?
  - trotzdem nicht vernachlässigbar → also nur, wenn notwendig
  - Ausnutzen der Modellhierarchie/des Szenengraphen
    - einzelne Transformationen sind einfach umkehrbar
    - Wiederverwenden von Matrizen

Transformationen und Schnitttests

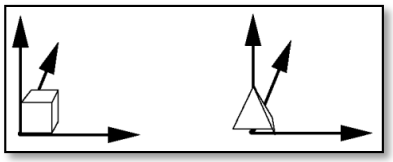
- Strahlerzeugung: typischerweise in Welt- oder Kamerakoordinaten
- Schnittpunktberechnung kann in jedem Koordinatensystem stattfinden



Ausblick: Transformationspipeline

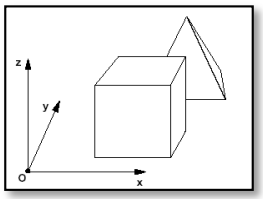


Tangentenraum



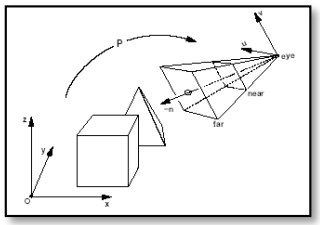
Objektkoordinaten

Modelltransformation

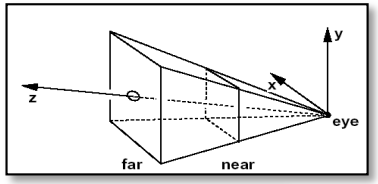


Weltkoordinaten

Kameratransformation

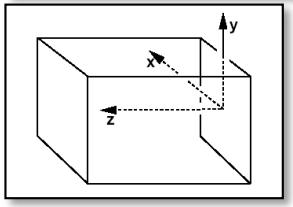


Kamerakoordinaten



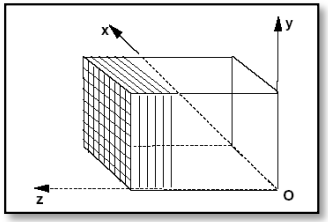
Projektionstransf.

Clip Koordinaten



Normalisierungstransf.

Normalized Device Coords



Viewport Transform

Window