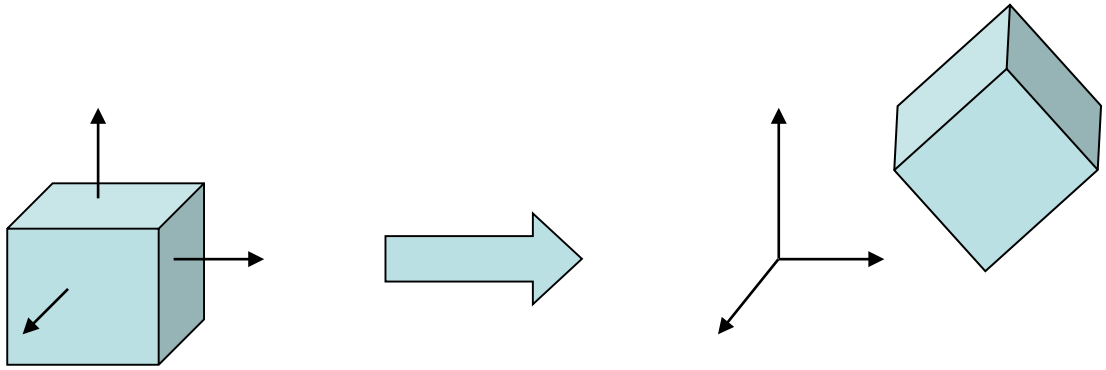
2. Orientierungen

Computer Animation

Wintersemester 11/12

- Aufgabe: Darstellung von Drehungen (Kamera, Objekt)
- Rigid-Body-Transformationen:
  - Translation + Rotation
  - $x_{new} = \boldsymbol{R} x_{old} + t$
    $\boldsymbol{R} \in \mathbb{R}^{3x3}$: 3x3-Matrix, die die Rotation beschreibt
    $\boldsymbol{R}$ ist eine *orthogonale Matrix*, d.h. $(\boldsymbol{R}^t \boldsymbol{R} = Id)$ und $det(\boldsymbol{R}) = 1$
    $t \in \mathbb{R}^3$: Translationsvektor

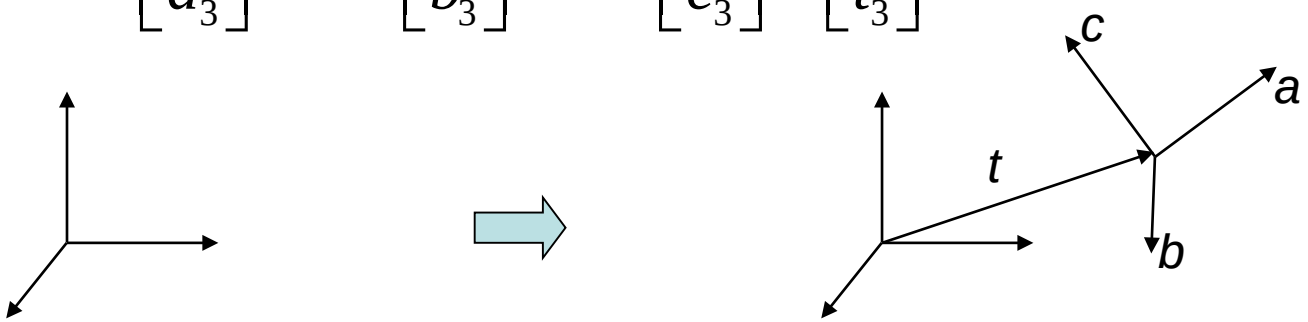


  - mehrere Rigid-Body-Transformationen bilden wieder Rigid-Body:
    $x = \boldsymbol{R}_2 (\boldsymbol{R}_1 x + t_1) + t_2 = \boldsymbol{R}_2 \boldsymbol{R}_1 x + (\boldsymbol{R}_2 t_1 + t_2)$

- wie sieht Rotationsmatrix aus?

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow x_1 \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + x_2 \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + x_3 \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

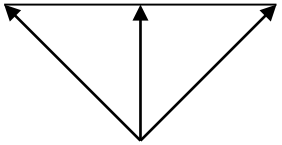


- a,b,c spannen neues Koordinatensystem auf

- wann ist die Matrix eine Rotationsmatrix?
  $\rightarrow$ wenn a,b,c orthonormales Koordinatensystem
- Zahl der Freiheitsgrade:
  - a: beliebiger Einheitsvektor
    = Punkt auf Einheitskugel
    = 2 Freiheitsgrade
  - b: muss senkrecht auf *a* stehen
    = kann in Ebene senkrecht auf *a* rotieren
    = 1 Freiheitsgrad
  - c: muss senkrecht auf *a* und *b* stehen
    = kein Freiheitsgrad
  - gesamt: 3 Freiheitsgrade

- *det(**R**) = 1* oder *det(**R**) = -1* (je nach Orientierung)
  Drehung bzw. Spiegelung

- Interpolation von Rotationsmatrizen $\boldsymbol{R_1}$,$\boldsymbol{R_2}$:
  $(1-\alpha)\,\boldsymbol{R_1} + \alpha\,\boldsymbol{R_2}$
- ist i.A. keine Rotationsmatrix mehr!
- Interpolation von $\boldsymbol{R_1}$ und $\boldsymbol{R_2}$ entspricht Interpolation der Basisvektoren
- interpolierte Einheitsvektoren bleiben keine Einheitsvektoren!
  auch Orthogonalität bleibt nicht erhalten

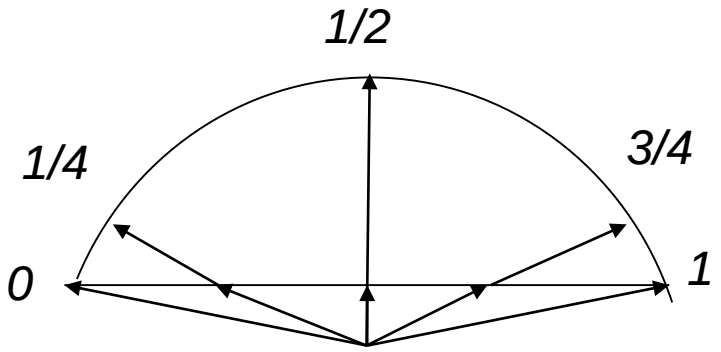


- Beispiel:

$$\frac{1}{2}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}+\frac{1}{2}\begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix}=\frac{1}{2}\begin{bmatrix}1&1&0\\0&1&1\\1&0&1\end{bmatrix}$$

2.1 Matrixdarstellung

- Möglichkeit: Re-Orthonormalisierung
- Interpoliere $\boldsymbol{R_1}$ und $\boldsymbol{R_2} \rightarrow \boldsymbol{R'} = (a', b', c')$
- forme (a', b', c') in orthonormales System (a, b, c) um:
  - normalisiere a' :
    a = normalize(a')
  - setze b auf Vektor senkrecht zu a' und c' und normalisiere:
    b = normalize($a \times c'$)
  - setze c auf Vektor senkrecht zu a und b:
    $c = a \times b$
- Probleme:
  - nicht immer möglich
    (z.B. wenn a' = (0 0 0))
  - nicht-uniforme Drehgeschwindigkeit



2.2 Fixed-Angles-Darstellung

- beschreibe allgemeine Rotation durch Rotation um drei Achsen, z.B.
  $\boldsymbol{R}_z(\gamma)\, \boldsymbol{R}_y(\beta)\, \boldsymbol{R}_x(\alpha)$

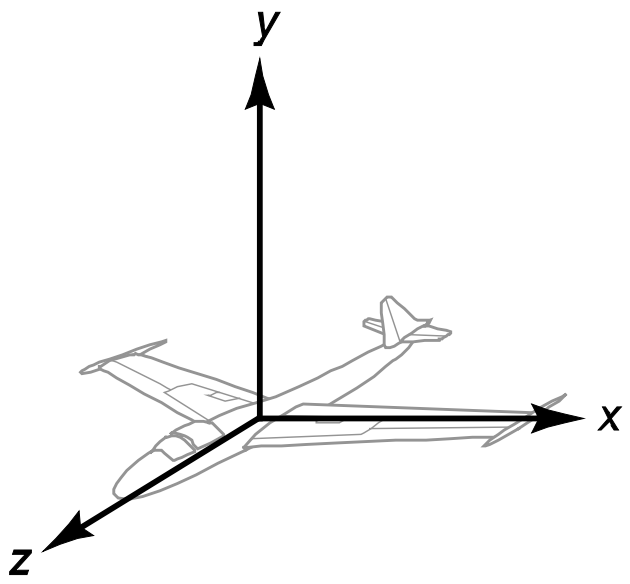
- Achtung:
  Matrizen werden von rechts nach links ausgeführt, d.h. oben erst $\boldsymbol{R}_x$, dann $\boldsymbol{R}_y$, dann $\boldsymbol{R}_z$

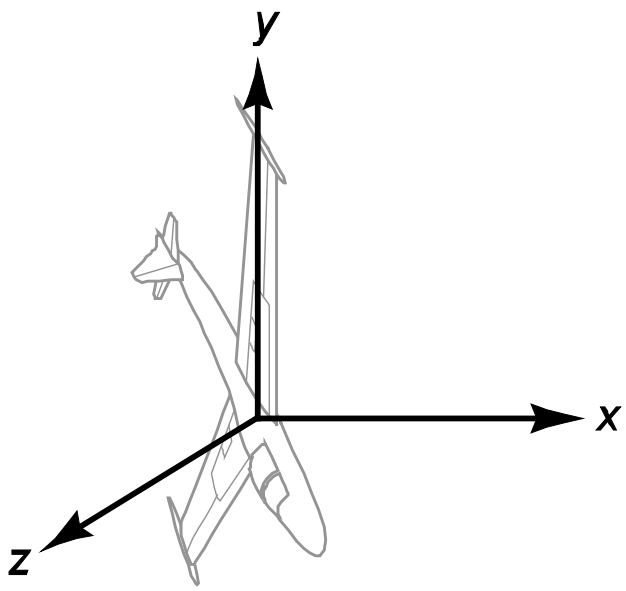
- Rotation beschrieben durch Tripel (α,β,γ)

- Die Rotationsachsen können fast beliebig sein:
  - z.B. xyz (Bsp. oben), zxy, …
  - sogar xyx, zxz , in Technik meist zxz
  - aber nicht: xxy!

2.2 Fixed-Angles-Darstellung

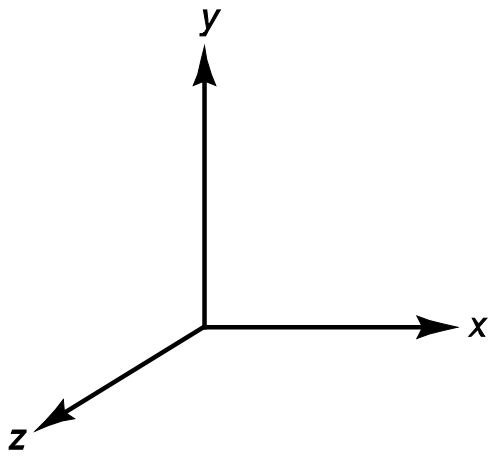
- Bsp.:
  - Achsenreihenfolge xyz
  - Winkel (10°,45°,90°)



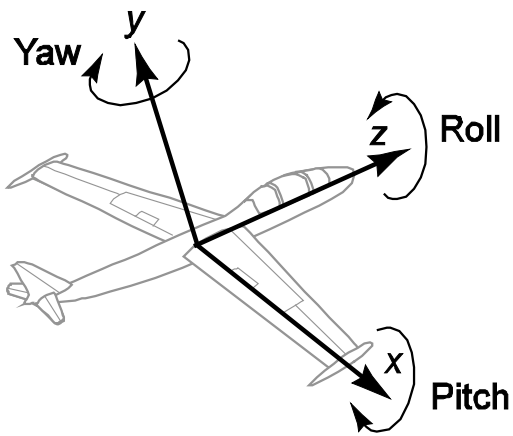


2.3 Euler-Winkel

- bei fixed angles:
  Rotation immer um die globalen Achsen
- Euler-Winkel:
  Rotation um die lokalen Achsen, d.h. man betrachtet ein mitrotierendes Koordinatensystem



Global coordinate system



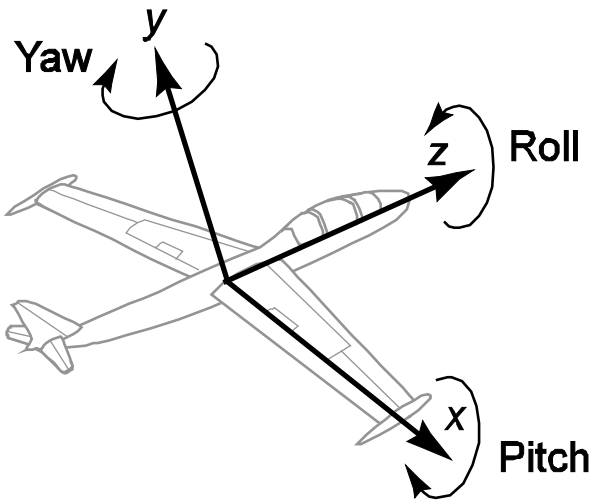
Local coordinate system attached to object

Rotationsachsen Fixed-Angles

Rotationsachsen Euler

2.3 Euler-Winkel

- typisches Beispiel: Flugzeug
  - Yaw α: 2D-Flugrichtung (Nord, SSW, …)
  - Pitch β: Steigung
  - Roll γ: Rotation um Flugrichtung



- Matrixdarstellung?
- zuerst Yaw: $\boldsymbol{R_y(\alpha)}$
- dann Pitch im lokalen System
  - zuerst Yaw rückgängig machen, um *x* drehen, wieder Yaw anwenden:
    $\boldsymbol{R_y(\alpha)\, R_x(\beta)\, R_y(-\alpha)}$
  - kombinieren:
    $\boldsymbol{R_y(\alpha)\, R_x(\beta)\, R_y(-\alpha)\, R_y(\alpha) = R_y(\alpha)\, R_x(\beta)}$
- dann Roll im lokalen System
  - Yaw und Pitch rückgängig, Roll, Yaw und Pitch anwenden:
  - $\boldsymbol{R_y(\alpha)\, R_x(\beta)\, R_z(\gamma)\, R_x(-\beta)\, R_y(-\alpha)}$
  - kombinieren:
    $\boldsymbol{R_y(\alpha)\, R_x(\beta)\, R_z(\gamma)\, R_x(-\beta)\, R_y(-\alpha)\, R_y(\alpha)\, R_x(\beta) = R_y(\alpha)\, R_x(\beta)\, R_z(\gamma)}$
- d.h.:
  für lokale Achsen einfach die Reihenfolge umdrehen!

2.3 Euler-Winkel

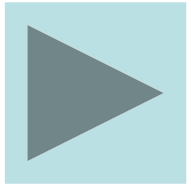
- keine Einigkeit in der Notation
- manche bezeichnen als Euler-Winkel:
  $\boldsymbol{R_y}(\alpha)\ \boldsymbol{R_x}(\beta)\ \boldsymbol{R_z}(\gamma)\ \boldsymbol{R_x}(-\beta)\ \boldsymbol{R}_y(-\alpha)$
- auch (α,β,γ) beschreiben eindeutig eine Rotation
- entspricht mehr der folgenden Darstellung aus Achse und Winkel

2.3 Euler-Winkel

- Interpolation:
  - interpoliere Winkel (geht auch für fixed angles)
  - Probleme: springende Winkel und gimbal lock!

- Bsp.: Flugzeug macht Salto
  - sobald Flugzeug senkrecht nach oben fliegt:
  - Winkel springen um 180°

- Bsp. Gimbal Lock: Analogon Polarkoordinaten
  - wir drehen um x,y,z
  - Konstellation: 0°,90°,0° → lokale x-Achse = globale z-Achse
  - nun probieren wir $(0° \pm \varepsilon, 90° \pm \varepsilon, 0° \pm \varepsilon)$
    - Rotation um z Rotation um y Rotation um y
  - keine Rotation um globale x-Achse mehr möglich!

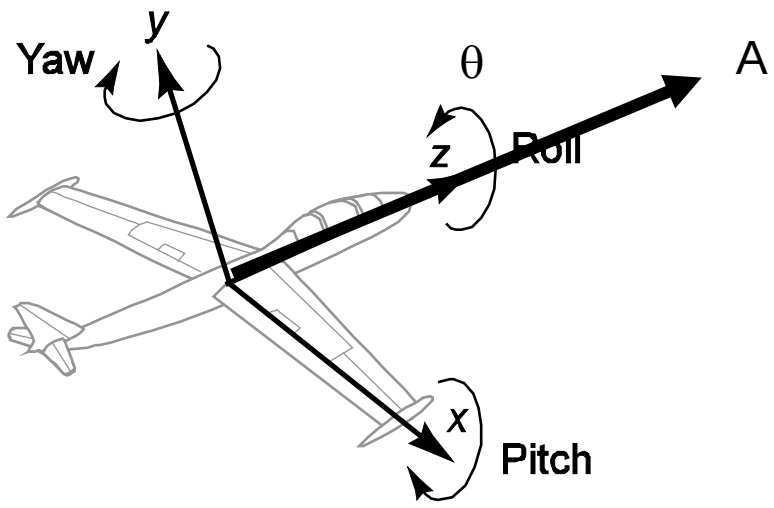


2.4 Achse+Winkel

- Satz von Euler:
  Jede Rotation kann beschrieben werden als eine Rotation um eine einzelne Achse
- Achse beliebig, also nicht nur x, y oder z
- OpenGL:
  - `glRotatef(angle,x,y,z)`

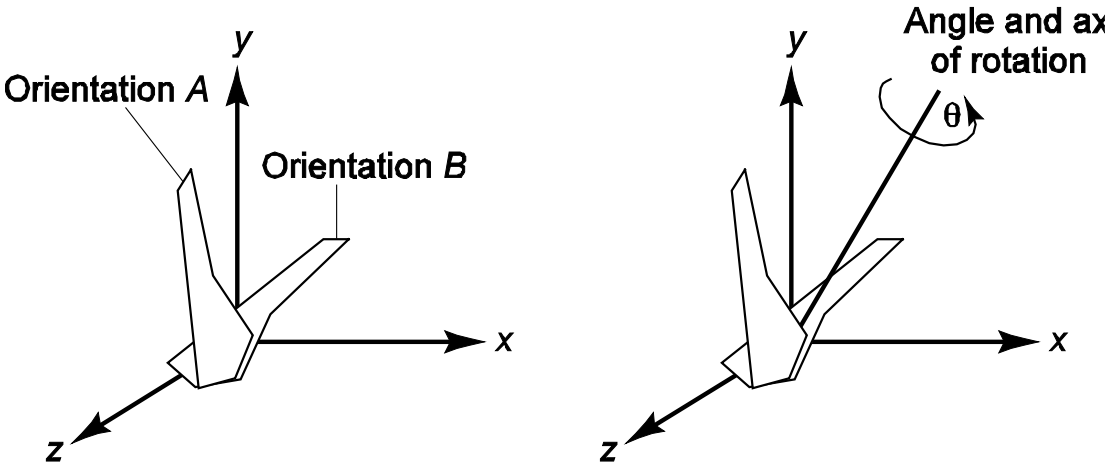
2.4 Achse+Winkel

- nützlich, wenn die Rotation eines Objektes beschrieben ist durch Richtung+Drehwinkel (A,θ)
  z.B.:



2.4 Achse+Winkel

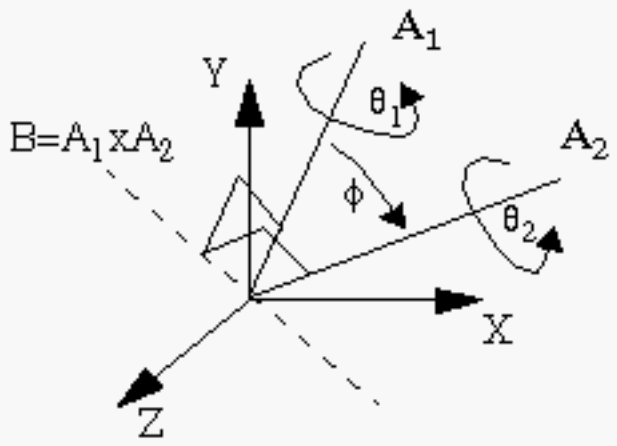
- Interpolation zwischen zwei Lagen (A_1,θ_1) und (A_2,θ_2)



- rotiere A_1 auf A_2 durch Rotation um $B=A_1 \times A_2$
- interpoliere θ_1 und θ_2

2.4 Achse+Winkel

- rotiere A_1 auf A_2 durch Rotation um $B=A_1 \times A_2$
- interpoliere θ_1 und θ_2



$$B = A_1 \times A_2$$

$$\phi = \cos^{-1}\left(\frac{A_1 \bullet A_2}{|A_1||A_2|}\right)$$

$$A_k = R_B(k \cdot \phi) A_1$$

$$\theta_k = (1-k) \cdot \theta_1 + k \cdot \theta_2$$

2.5 Quaternionen

- allgemeines Konzept zur Beschreibung von Rotationen durch Quadrupel $q \in \mathbb{R}^4$:

  $q = (s,x,y,z)$ oder $q = (s,\mathbf{v})$ mit $\mathbf{v} \in \mathbb{R}^3$

- ähnlich wie komplexe Zahlen, aber 3-dim. Imaginärteil

- Polardarstellungen:
  - $z \in \mathbb{C}$, $z = |z|(\cos\phi + i \sin\phi)$
  - $q \in \mathbb{H}$, $q = |q|(\cos\phi, \sin\phi\, \mathbf{v}/|\mathbf{v}|)$

2.5 Quaternionen

- 3D-Vektoren und 3D-Rotationen können als Quaternion ausgedrückt werden:
  - Rotation um Achse *[x,y,z]* mit Winkel θ:
    $q = Rot_{\theta,(x,y,z)} = (\cos(\theta/2), \sin(\theta/2)[x,y,z])$
  - Punkt an der Stelle *[a,b,c]*:
    $q = (0,[a,b,c])$

- Umrechnung $\rightarrow$ axis,angle
  - (s,v) $\rightarrow$ Achse = v, Winkel = 2 arccos s

- Quaternionen erlauben einfache Interpolation (s.u.)

2.5 Quaternionen

- Addition:
  $(s_1,v_1) + (s_2,v_2) = (s_1+s_2, v_1+v_2)$

- Multiplikation:
  $(s_1,v_1)(s_2,v_2) = (s_1s_2 - v_1 \cdot v_2,\ s_2v_1 + s_1v_2 + v_1 \times v_2)$

  Achtung:
  - $q_1q_2 \neq q_2q_1$, aber: $(q_1q_2)q_3 = q_1(q_2q_3)$

2.5 Quaternionen

- Betrag:
  $||(s,[x,y,z])|| = sqrt(s^2+x^2+y^2+z^2)$

- Inverse:
  $(s,\boldsymbol{v})^{-1} = (1/||(s,\boldsymbol{v})||)^2(s,-\boldsymbol{v})$

- Einheitsquaternion:
  $qq^{-1} = (1,[0,0,0])$

- Quaternion normalisieren:
  $q/||q||$

2.5 Quaternionen

- einen Punkt $\boldsymbol{v}=[a,b,c]$ mit Rotation q rotieren:
  $\boldsymbol{v}' = q\,(0,\boldsymbol{v})\,q^{-1}$

- Hintereinanderausführung = Multiplikation
  $v' = q_2\,(q_1 v q_1^{-1})\,q_2^{-1} = (q_2 q_1) v (q_2 q_1)^{-1}$

- Skalierung von q hat keinen Einfluss auf die erzeugte Rotation!
  - insbesondere:
    $-(s,v) = (-s,-v) \equiv (s,v)$ (negative Rot. um neg. Achse)

Zsfg.: Quaternionen

Fields of numbers *(Zahlen-)Körper* :

reelle Zahlen	komplexe Zahlen	Quaternionen
$\mathbb{R}$	$\mathbb{C}$	$\mathbb{H}$
$= \mathbb{R}$	$\equiv \mathbb{R} \times \mathbb{R}$	$\equiv \mathbb{R} \times \mathbb{R}^3$
x	$z = (x, y) = x + iy$	$q = (q_0, \vec{q})$
$+ \quad \cdot$	$+ \quad \cdot$	$+ \quad \cdot$
$0 \quad 1$	$(0,0) \quad (1,0)$	$(0,\vec{0}) \quad (1,\vec{0})$
$\frac{1}{x}$	$\frac{1}{z} = \frac{1}{\lvert z\rvert^2}(x, -y)$	$\frac{1}{q} = \frac{1}{\lvert q\rvert^2}(q_0, -\vec{q})$
(polar coords.)	$z = \lvert z\rvert(\cos(u), \sin(u))$	$\lvert q\rvert(\cos(u), \sin(u)\vec{n}_0)$
(frac. powers)	$z^t = \lvert z\rvert^t(\cos(tu), \sin(tu))$	$\lvert q\rvert^t(\cos(tu), \sin(tu)\vec{n}_0)$

Wh.: 3D Transformationen : Rotationen

- 2D : wähle eine komplexe Zahl vom Betrag 1 : w_0
  $z \mapsto z \cdot w_0$
  definiert eine Rotation in 2D

- 3D : wähle eine (Einheits-)Quaternion q
  $v \mapsto q^{-1} \cdot (0,v) \cdot q$ (conjugation)
  definiert eine Rotation in 3D

- Dies definiert eine 2-zu-1 Abbildung **aller** Einheits-Quaternionen auf die Menge **aller** Rotationen.

  (eine Parameterisierung ohne singuläre Punkte)

2.6 Umrechnungen: (Wechsel zw. versch. Darstellungen)

Verschiedene Darstellungen von Rotationen in 3D

- Orthogonale Matrizen O
  Variante: 3-Bein
- 3 Euler-Winkel: (ϕ, θ, ψ)
  - $Rot_z \rightarrow Rot_x \rightarrow Rot_z$
  - $Rot_x \rightarrow Rot_y \rightarrow Rot_z$
- Drehachse und –winkel der Rotation (n , θ)
- Quaternionen $q = (q_0, \boldsymbol{q})$
- 3D-Vektor m
  - Achse = Vektor,
  - Winkel = Länge des Vektor

2.6 Umrechnungen

- Vektor $\leftrightarrow$ axis & angle $\leftrightarrow$ Quaternion

$$\vec{n},\ \theta \text{ (with } \|\vec{n}\| = 1) \quad \rightarrow \quad \vec{v} = \theta \cdot \vec{n}$$

$$\vec{v} \text{ (with } \|\vec{v}\| \leq \pi) \quad \rightarrow \quad \vec{n} = \frac{\vec{v}}{\|\vec{v}\|},\ \theta = \|\vec{v}\|$$

$$\vec{n},\ \theta \text{ (with } \|\vec{n}\| = 1) \quad \rightarrow \quad \mathbf{q} = \left(\cos(\frac{\theta}{2}), \sin(\frac{\theta}{2}) \cdot \vec{n}\right)$$

$$\mathbf{q} = (q_0, \vec{q}) \text{ (with } \|\mathbf{q}\| = 1) \quad \rightarrow \quad \vec{n} = \vec{q},\ \theta = 2 \cdot \arccos(q_0)$$

2.6 Umrechnungen (Eulerwinkel → Matrix)

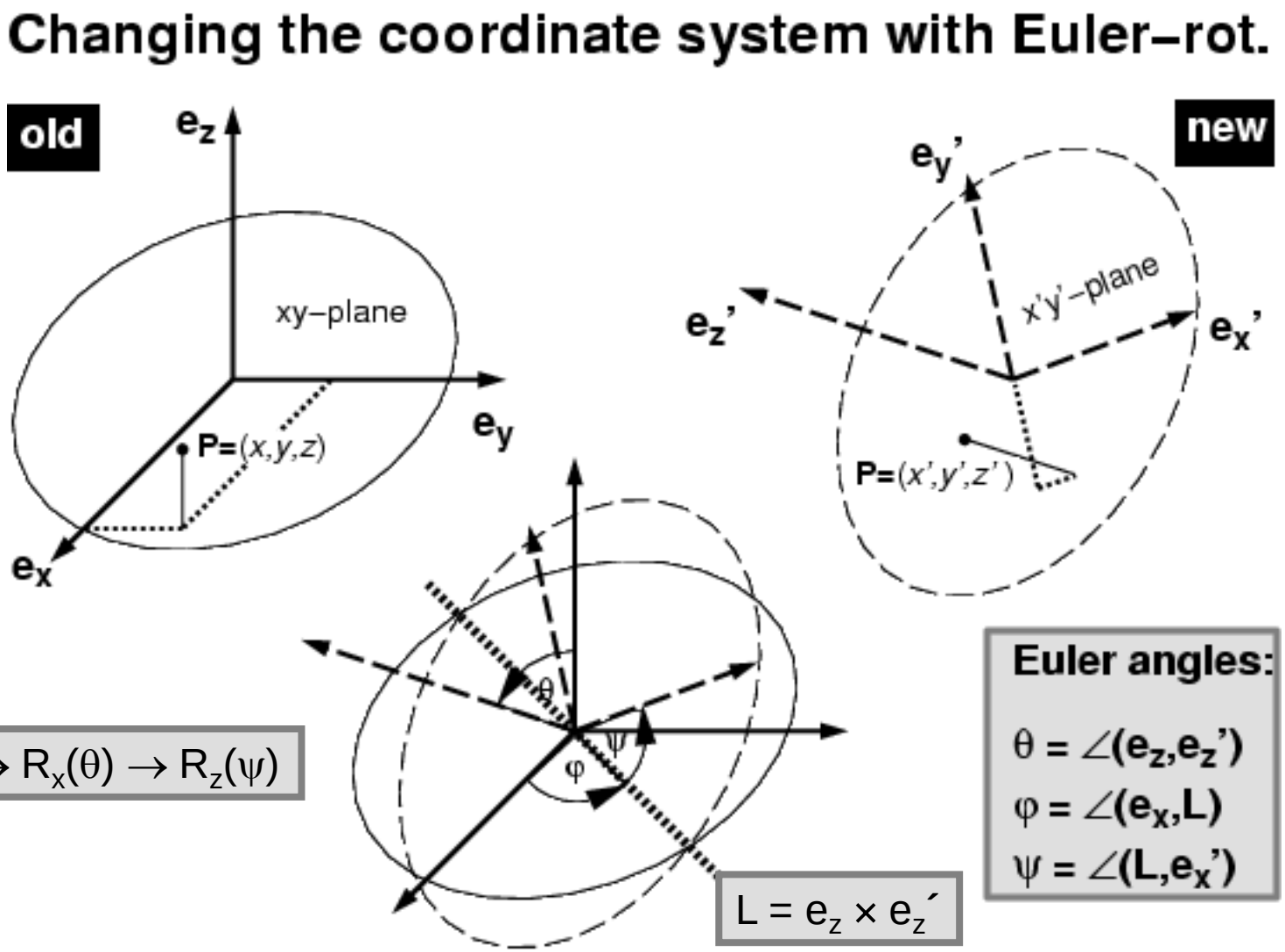
$$O = \begin{pmatrix} o_{11} & o_{12} & o_{13} \\ o_{21} & o_{22} & o_{23} \\ o_{31} & o_{32} & o_{33} \end{pmatrix} = Rot_z(\varphi) Rot_x(\theta) Rot_z(\psi) =$$

$$\begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$O = \begin{bmatrix} o_{11} & o_{12} & o_{13} \\ o_{21} & o_{22} & o_{23} \\ o_{31} & o_{23} & o_{33} \end{bmatrix} = Rot_z(\varphi) \circ Rot_z(\theta) \circ Rot_z(\psi) =$$

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

2.6 Umrechnungen (Matrix → Eulerwinkel)



2.6 Umrechnungen (Matrix ↔ Quaternion)

- Quaterionen → Matrix

  erste Spalte: $\boldsymbol{q}^{-1}\,(0,(1,0,0))\boldsymbol{q} =$
  $(0,(1,0,0)-2*(q_2^2+q_3^2\,,\ -q_1q_2-q_4q_3\,,\ q_4q_2-q_1q_3\,))$
  zweite & dritte Spalte ebenso.

- Matrix → Axis & angle (→ Quaternion)
  - Eigenwert Analyse:
    3 Eigenwerte $1,\ \cos(\phi) + i\cdot\sin(\phi),\ \cos(\phi) - i\cdot\sin(\phi)$
    axis = Eigenvektor zu 1 , angle = ϕ ;

  - $\phi = \arccos(.5*(\mathrm{Trace}(O)-1))$;
    $n = (\ O_{3,1}+O_{1,3},\ O_{3,2}+O_{2,3},1+O_{3,3}-O_{1,1}-O_{2,2}\ );$

2.6 Umrechnungen (Vektor $\rightarrow$ Matrix)

- $v = (a,b,c)$ (Achse = v, Winkel = $\|v\|$)

  $O = \exp(B) = \sum_k 1/k! \cdot B^k$

  wo $B : x \rightarrow v \times x$ (Kreuzprodukt)

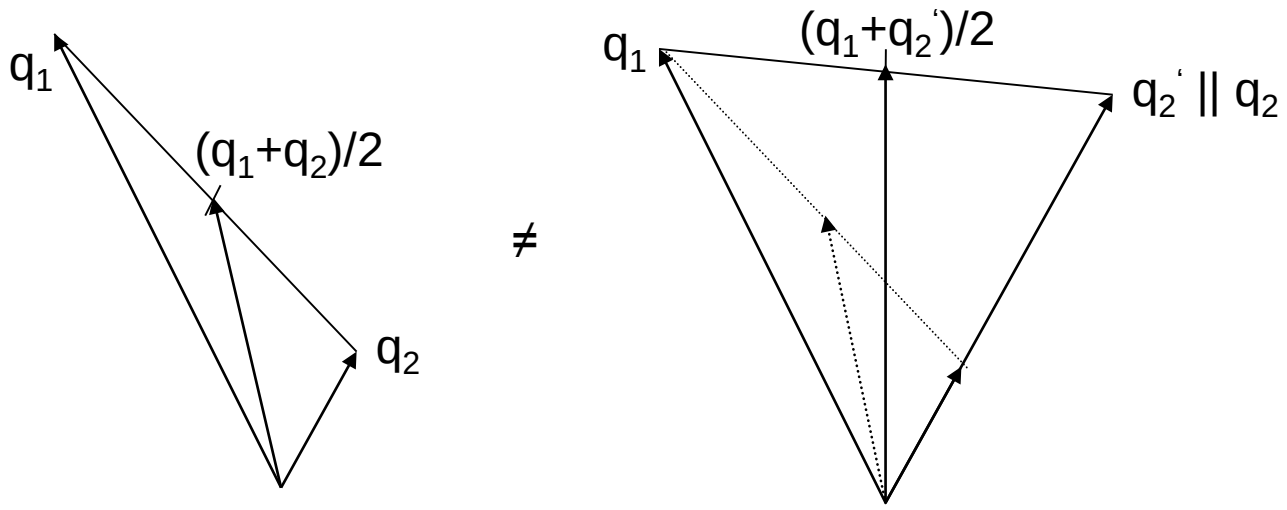
$$B = \begin{pmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{pmatrix}$$

- $\exp(B) = Id + \sin(\lambda)/\lambda \cdot B + (1-\cos(\lambda))/\lambda^2 \cdot B^2$
  wobei $\lambda^2 = a^2+b^2+c^2$

  (Beweis: $B^3 = -\lambda^2 B$, $B^{2n+1} = (-1)^n \lambda^{2n} B$
  $B^{2n+2} = (-1)^n \lambda^{2n} B^2$)

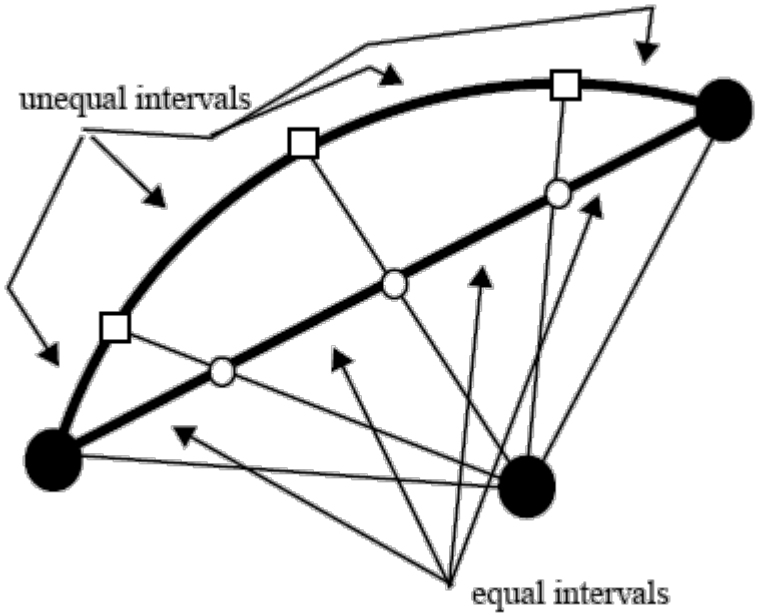
2.7 Interpolation

- Interpolation zwischen zwei Quaternionen
  - direkte Interpolation möglich, aber Ergebnis abhängig von Länge



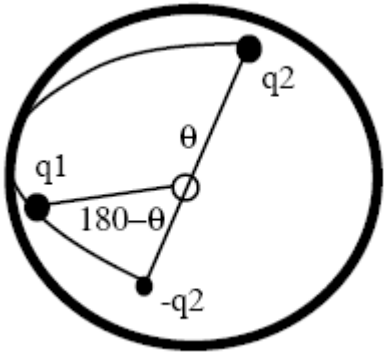
2.7 Interpolation

- → interpoliere Einheitsquaternionen
- aber dann immer noch: uneinheitliche Geschwindigkeit



2.7 Interpolation: Sphärische lineare Interpolation

- außerdem:
  $q_2 \equiv -q_2$, d.h. man kann von q_1 nach q_2 interpolieren oder von q_1 nach $-q_2$ (analog von $-q_1$ nach q_2)
- der kürzere Weg ist der bessere



$$\cos\theta = (\pm q_2) \circ q_1 = \pm (s_1 s_2 + v_2 \circ v_1) > 0$$

2.7 Interpolation

- $\rightarrow$ normalisiere q_1 und q_2
- $q_1 \circ q_2 > 0 \rightarrow$ interpoliere von q_1 nach q_2
  sonst von q_1 nach $-q_2$

- Geschwindigkeitsproblem bleibt
  $\rightarrow$ Interpolation auf Kugel
  (**sphärische lineare Interpolation** = **slerp**)

$$\text{slerp}(q_1, q_2, u) = \frac{\sin((1-u)\theta)}{\sin\theta} q_1 + \frac{\sin(u\theta)}{\sin\theta} q_2$$

dabei ist $\cos\theta = q_1 \circ q_2 = s_1 s_2 + \vec{v}_1 \circ \vec{v}_2$

- $\rightarrow$ ist normalisiert!

2.7 Interpolation

- Herleitung der Formel

1. mit Gram-Schmidt eine ONS in der von q_1 und q_2 aufgespannten Ebene:

   $e_1 = q_1$ und $e_2 = q_2 - (e_1 \circ q_2)e_1 / ||\ldots||$;
2. $q(u) = \cos(u\theta)e_1 + \sin(u\theta)e_2$.

$$\text{slerp}(q_1, q_2, u) = \cos(u\theta) \cdot q_1 + \frac{\sin(u\theta)}{\sin\theta}\left(q_2 - \cos(\theta)q_1\right)$$

$$= \frac{1}{\sin\theta}\left((\cos(u\theta)\sin\theta - \sin(u\theta)\cos\theta)q_1 + \sin(u\theta)q_2\right)$$

$$= \frac{1}{\sin\theta}\left(\sin((1-u)\theta)q_1 + \sin(u\theta)q_2\right)$$

2.7 Interpolation

- Andere Darstellung und Herleitung von *slerp* (analog zu $\mathbb{C}$)

- Zunächst von 1 = (1,(0,0,0)) nach q = (cos(α) , sin(α)·v):

$$q^t = (\cos(t\alpha) , \sin(t\alpha)\cdot v)$$

- Nun beliebig von p nach q

$$\text{slerp}(p,q,t) = p(p^{-1}q)^t = \text{und } (qp^{-1})^t p$$