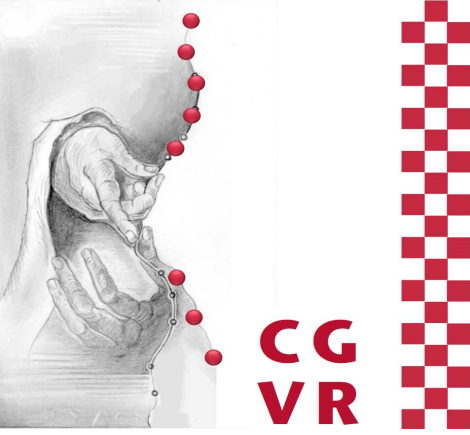




Computer-Graphik I

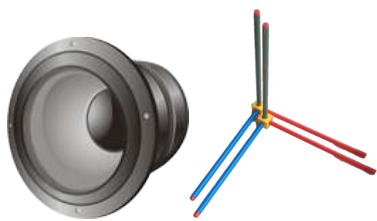
Rotationen

$$\begin{bmatrix} \cos 90^\circ & \sin 90^\circ \\ -\sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ a_1 \end{bmatrix}$$

G. Zachmann

University of Bremen, Germany

cgvr.informatik.uni-bremen.de



Vier Darstellungen von allgemeinen Rotation

Axis+Angle

Euler-Winkel

Quaternionen

Rotationsmatrix

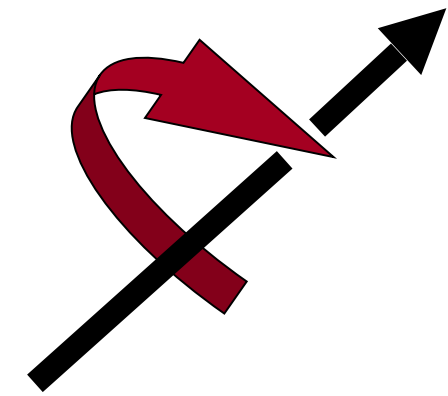
Darstellung: Achse + Winkel

- Intuitive Darstellung: $(\varphi, \mathbf{r})$
 - Meist mit der Nebenbedingung $\|\mathbf{r}\| = 1$
- Anzahl Freiheitsgrade allgemeiner Rotationen: 3 DOFs (*degrees of freedom*)
 - 2 DOFs für die Achse + 1 DOF für den Winkel
- Anmerkung: in der Robotik wird gerne die Variante verwendet, wo $\mathbf{r}$ beliebig lang sein darf, und dabei wird

$$\varphi = \|\mathbf{r}\|$$

als Winkel interpretiert (damit benötigt man wieder nur einen 3D-Vektor)

- Bezeichnungen: Euler-Vektor, oder Rotationsvektor

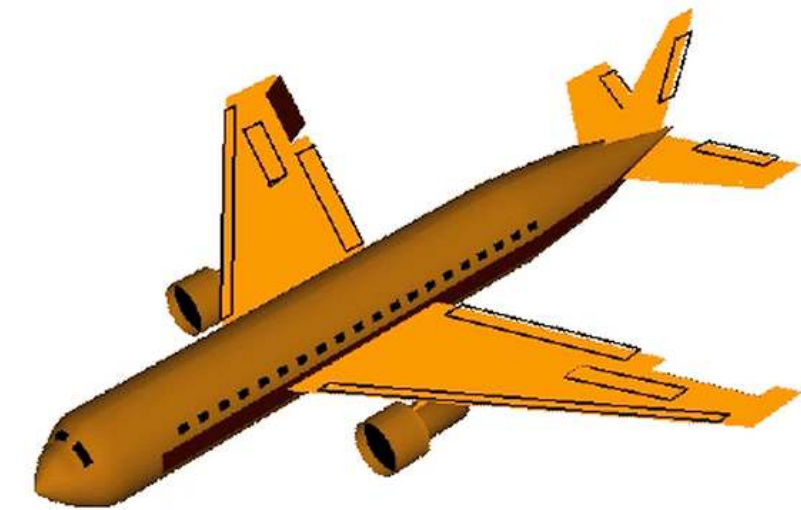


Die Euler-Winkel

- Intuitive(?) Konstruktion einer beliebigen Rotation: Konkatenation aus 3 elementaren Rotationen — erst Rotation um X-, dann um Y-, dann um Z-Achse
- Diese Winkel heißen **Euler-Winkel**
- Häufige Variante im Maschinenbau:

$$R(r, p, y) = R_z(y)R_x(p)R_y(r)$$

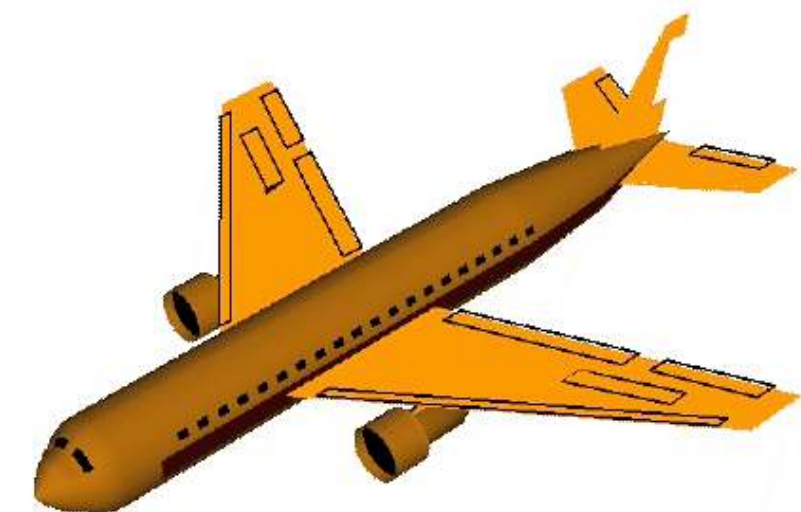
- Bezeichnung:
roll, pitch, yaw (Schiff)
roll, pitch, heading (Flugzeug)



Roll



Pitch



Yaw
(Heading)

Euler's Satz über Rotationen

Jede Rotation kann mit Hilfe 3-er Winkel um (fast) beliebige Achsen beschrieben werden.

Jede beliebige Rotation im Raum lässt sich als Rotation um eine bestimmte Achse mit einem bestimmten Winkel darstellen.

In der Praxis: Euler-Winkeln macht viele Probleme!!

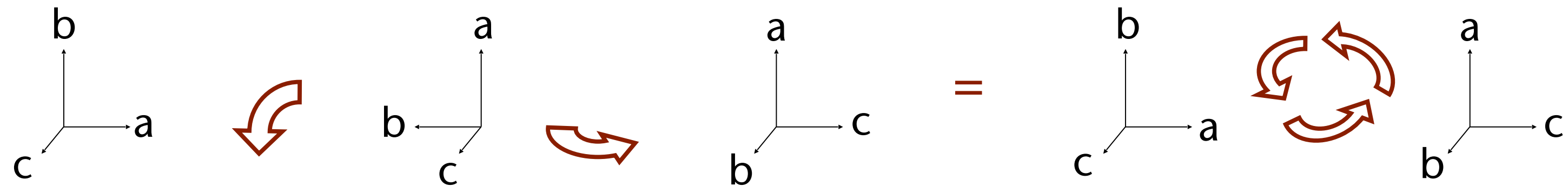
1. Reihenfolge der Rotationen ist nicht festgelegt!
 - Oft "*roll, pitch, yaw*" — aber Zuordnung zu den Achsen nicht definiert!
 - Manchmal auch: z, x, z ! Oder ...
2. Führt manchmal zu **Gimbal Lock**!
3. Werte-Bereiche: für einen der Winkel ist der Bereich nur $[-90, +90]$
 - Sonst: Mehrfachüberdeckung von $SO(3)$
4. Rechnen (z.B. interpolieren oder mitteln) ist i.A. **sinnlos und/oder unmöglich!**

Beispiel

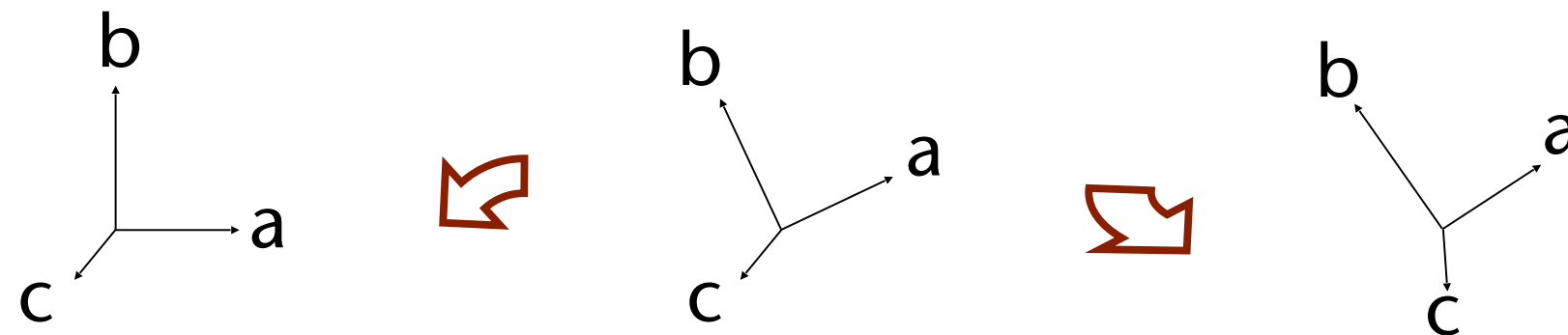
- Interpolation zwischen zwei Rotationen (Orientierungen) mittels Euler-Winkel liefert unerwünschte Resultate

- Beispiel:

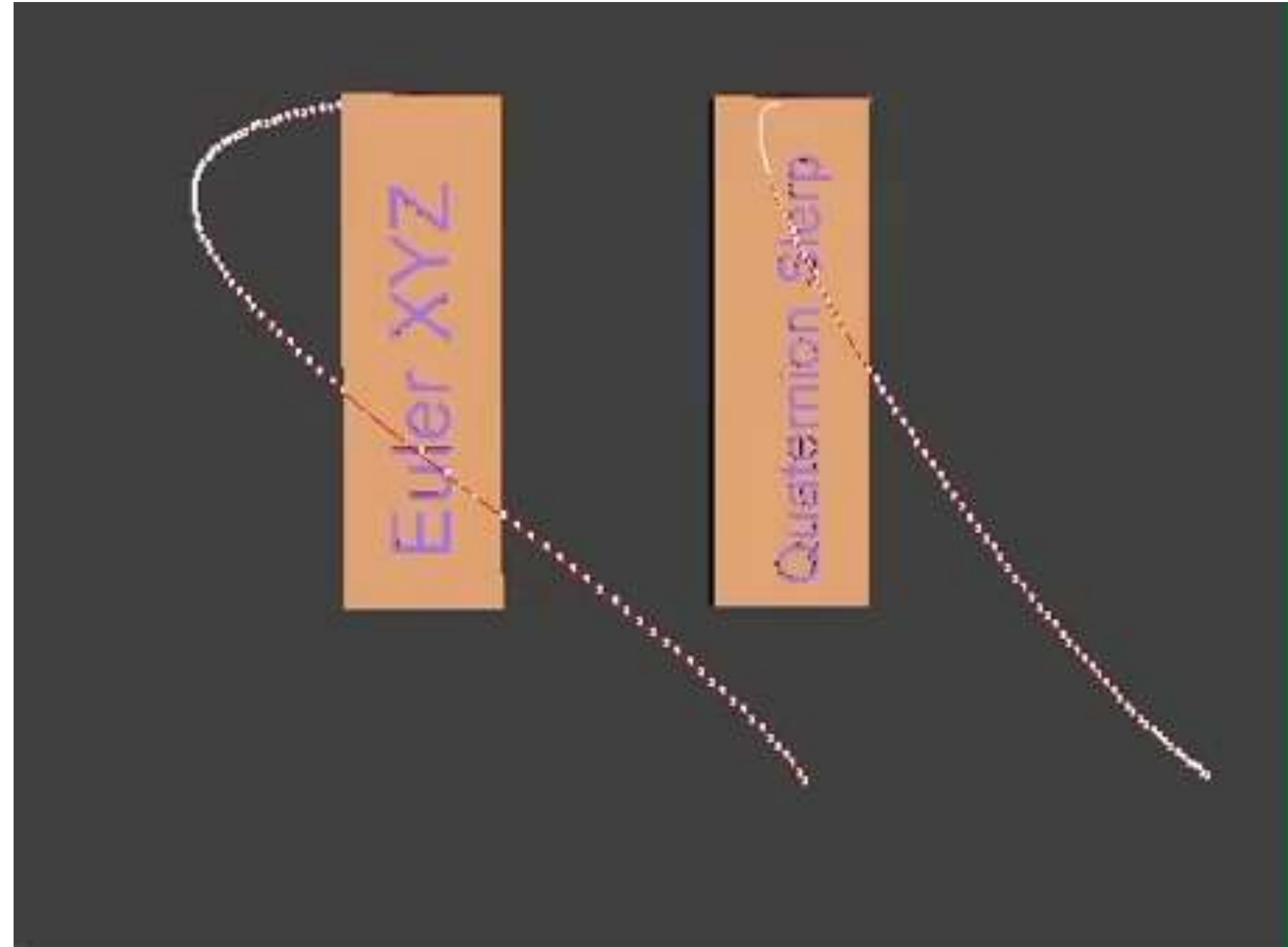
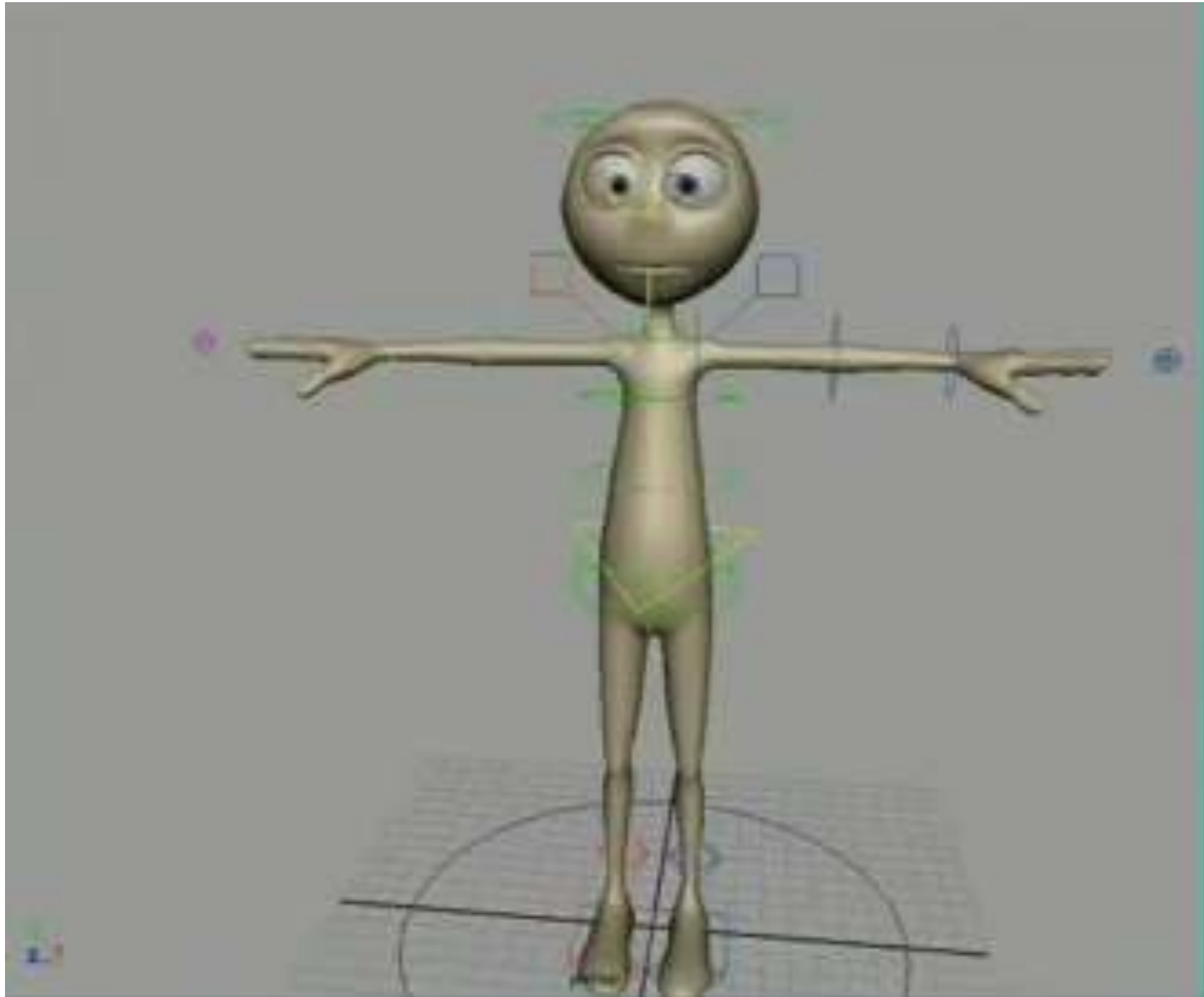
- Rotation von 90° um Z, dann 90° um Y $=$ 120° um $(1, 1, 1)$



- Rotation von 30° um Z, dann 30° um Y $\neq$ 40° um $(1, 1, 1)$!

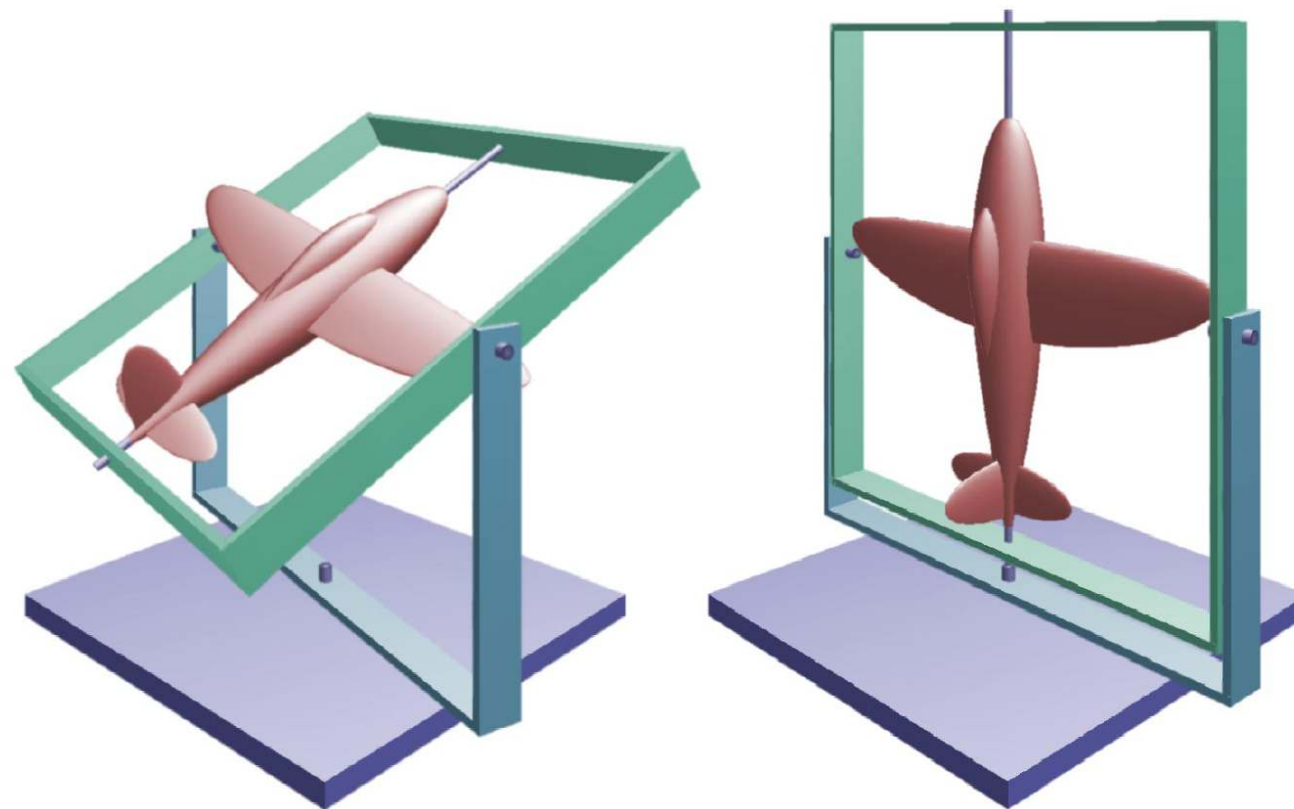


Beispiel für die unerwünschten Effekte bei Interpolation von Euler-Winkeln



Der *Gimbal Lock*

- Immer dann, wenn "innerer" und "zweiter" Winkel = 90 Grad
- Folge: nur noch **2 Freiheitsgrade**! (obwohl es immer noch 3 Parameter sind)
 - Rotation um eine der (lokalen) Flugzeugachsen geht nicht mehr!

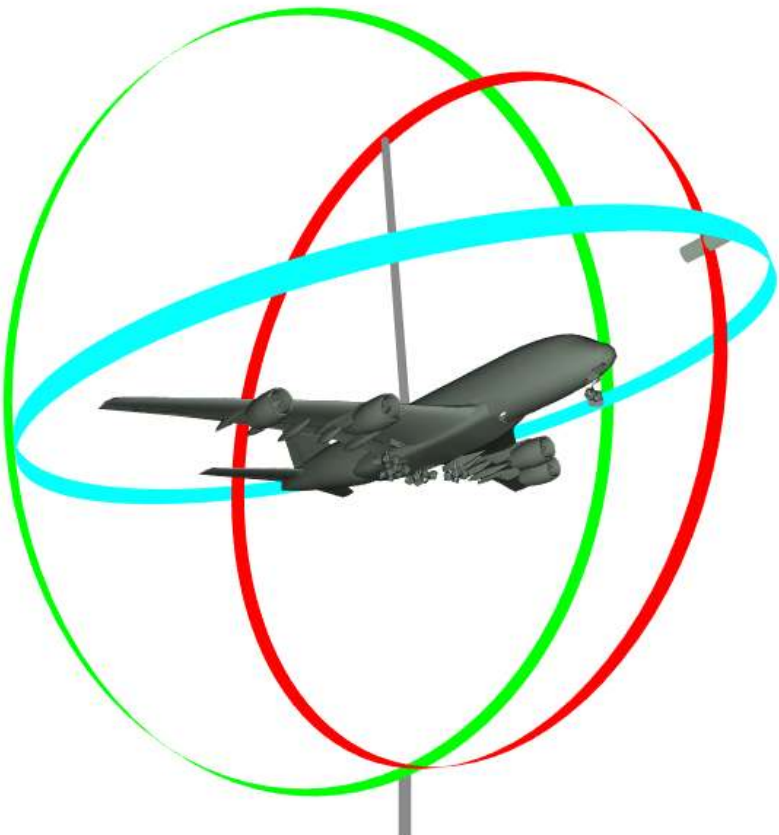


Demo zu Interpolation und Gimbal Lock

EulerAnglesViz.html

file:///Users/zach/Documents/Lehre/CG1/demos/Euler-Winkel/EulerAnglesViz.html

Euler Angles Interpolation Animation and Gimbal Visualization



Euler Angles

Yaw: 136. Set
 Pitch: 19.8 Set
 Roll: 12.6 Set

Animation

	Yaw	Pitch	Roll	
Orientation 1	168.	2.7	2.8	Set from current
Orientation 2	71.6	55.1	32.8	Set from current
Speed/sec	5	Start animation		

Gimbals

Display Gimbals ☒

Directions

- Manipulate the sliders to change roll/pitch/yaw. It is also possible to input your own angles and hit the "set" button to update them
- For an animation which does linear interpolation between two sets of Euler angles, choose an initial yaw/pitch/roll next to "orientation1" in the animation menu to the right, as well as a final orientation next to "orientation 2." You can also automatically fill in the angles corresponding to the current slider positions by clicking "set from current."
- Click the "animate" button to perform linear interpolation between orientation 1 and orientation 2
- Click the "display gimbals" checkbox to toggle displaying of the gimbals. This may make it easier to view the animation
- To view the gimbals from a different point of view, left click and drag your mouse. To zoom, right click and drag. To translate, center click and drag

Source

Modified from [Source](#)

Anekdote: Euler Angles in Spaceflight

- In den Apollo-Raketen wurde ein Kreiselkompass verwendet
 - (Basiert auf der Trägheit einer sich sehr schnell drehenden Masse)
- Kleine Anekdote dazu:

About two hours after the Apollo 11 landing, Command Module Pilot Mike Collins had the following conversation with CapCom Owen Garriott:

104:59:35 Garriott: Columbia, Houston. We noticed you are maneuvering very close to **gimbal lock**. I suggest you move back away. Over.

104:59:43 Collins: Yeah. I am going around it, doing a CMC Auto maneuver to the Pad values of roll 270, pitch 101, yaw 45.

104:59:52 Garriott: Roger, Columbia. (Long Pause)

105:00:30 Collins: (Faint, joking) How about sending me a fourth gimbal for Christmas.

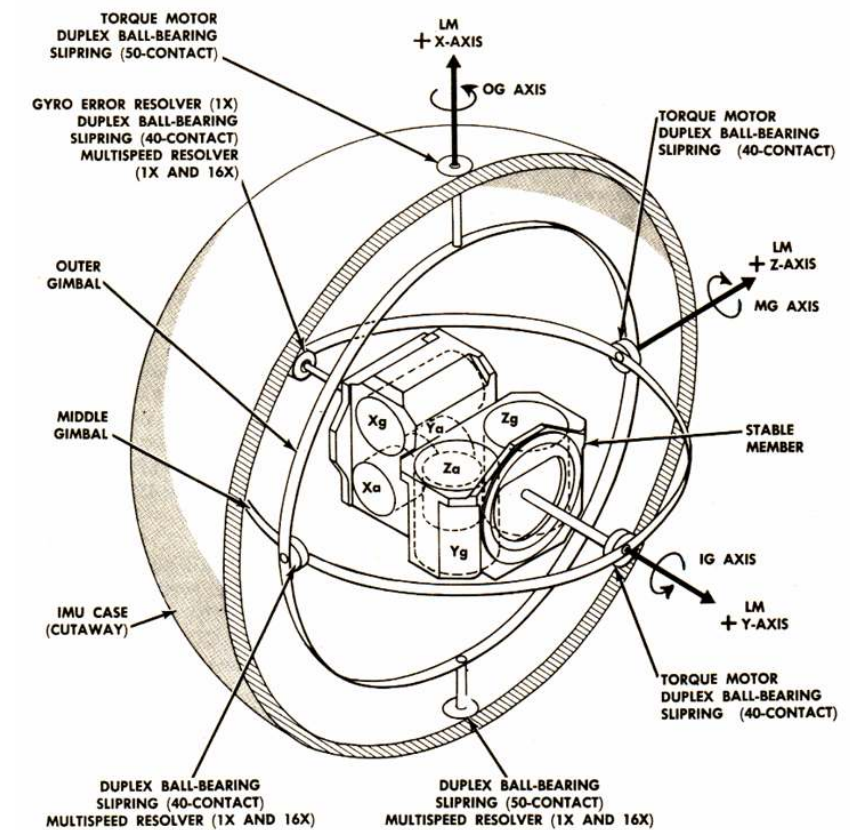


Figure 2.1-24. IMU Gimbal Assembly

- Um Gimbal-Lock zu vermeiden, wurde tatsächlich ein 4-ter Gimbal eingeführt!

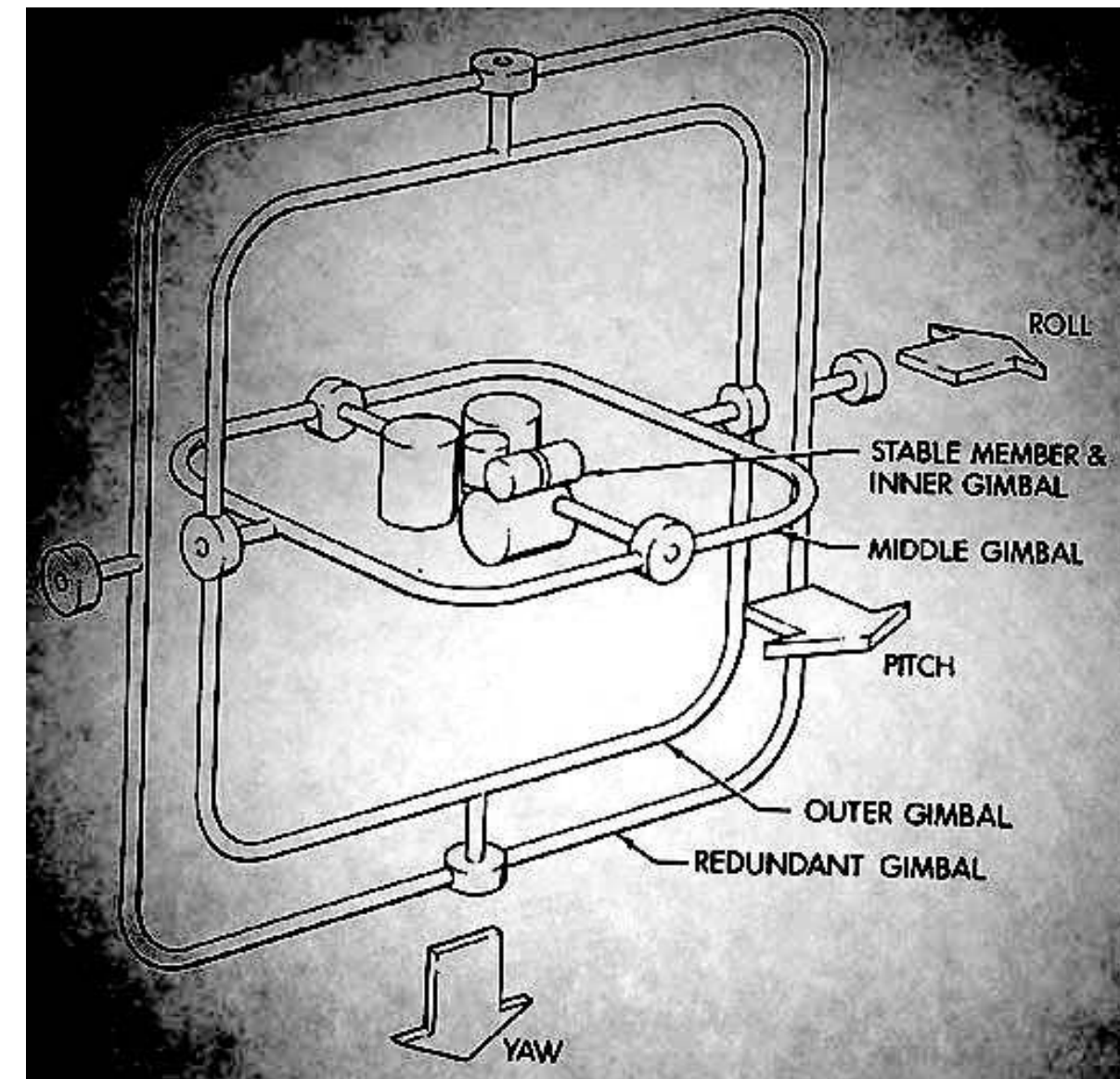
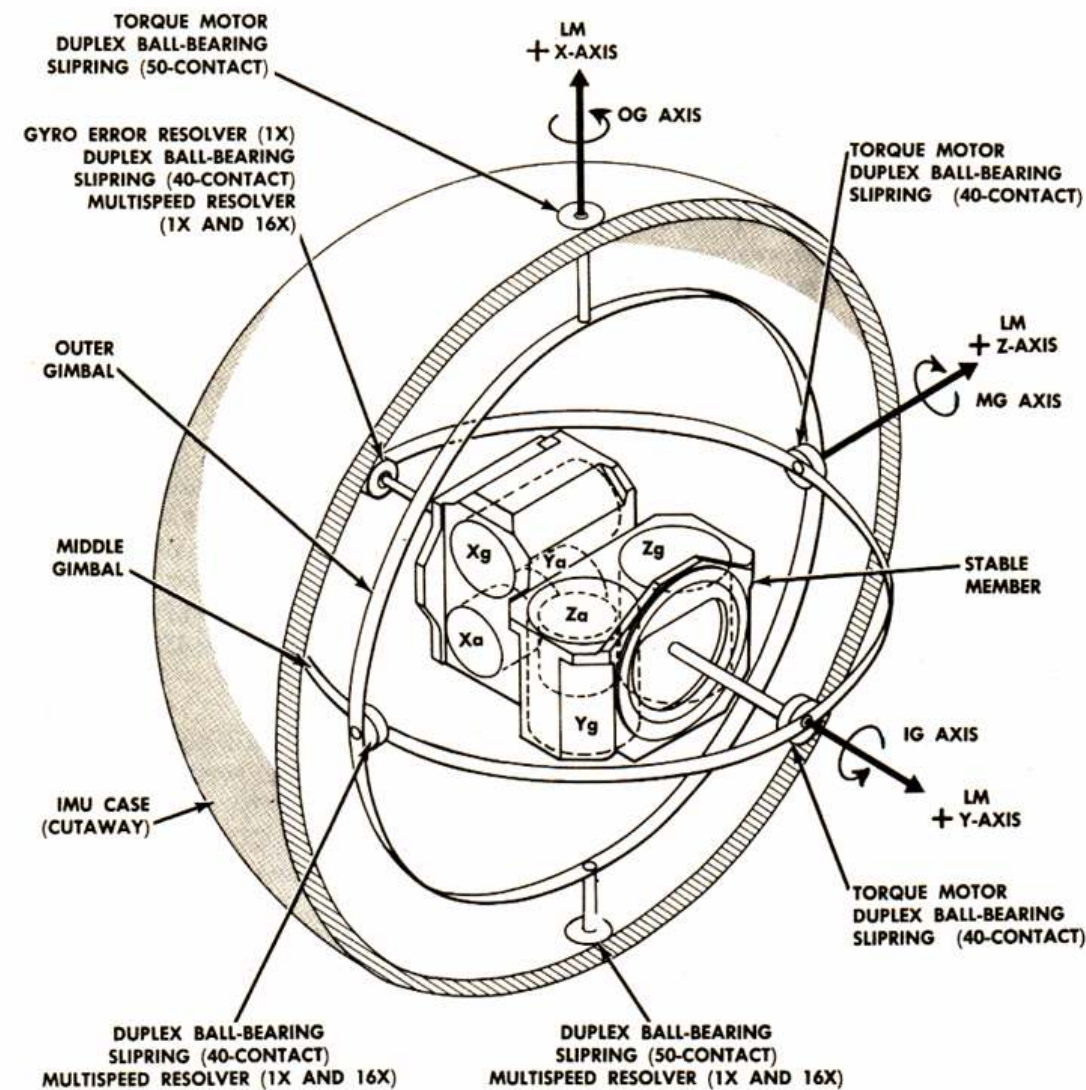


Figure 2.1-24. IMU Gimbal Assembly

Quelle: <http://www.hq.nasa.gov/office/pao/History/alsj/gimbals.html>

Matrix-Darstellung von Rotationen

- Gegeben: Achse $\mathbf{r}$ und Winkel θ
- Gesucht: Rotationsmatrix zu gegebener Rotationsachse $\mathbf{r}$ (oBdA geht $\mathbf{r}$ durch den Ursprung)
- 1. Erzeuge neue Basis $(\mathbf{r}, \mathbf{s}, \mathbf{t})$ (bestimme $\mathbf{s}$ und $\mathbf{t}$, orthogonal zu $\mathbf{r}$; siehe Kapitel "Kurze Wiederholung in Geometrie")
- 2. Transformiere alles, so daß die Basis $(\mathbf{r}, \mathbf{s}, \mathbf{t})$ in die Standardbasis $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ übergeht
- 3. Rotiere mit Winkel θ um $\mathbf{x}$ -Achse
- 4. Transformiere zurück in die ursprüngliche Basis
- Zusammen:

$$M = BR_x(\theta)B^T \quad \text{mit} \quad B = \begin{pmatrix} | & | & | \\ \mathbf{r} & \mathbf{s} & \mathbf{t} \\ | & | & | \end{pmatrix}$$

Konstruktion einer Rotation aus Koordinatenachsen

- Man kann eine Rotationsmatrix direkt aus den 3 Koordinatenachsen eines (neuen) Koordinatensystems konstruieren

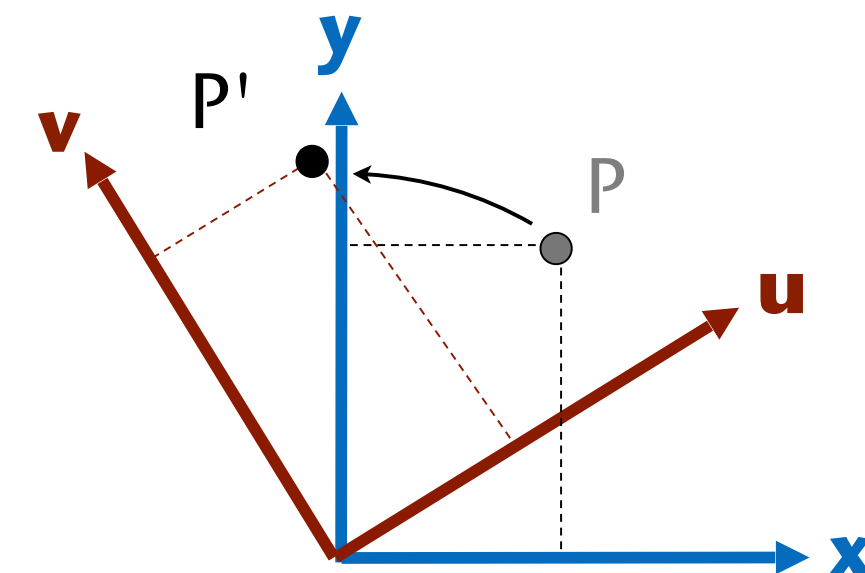
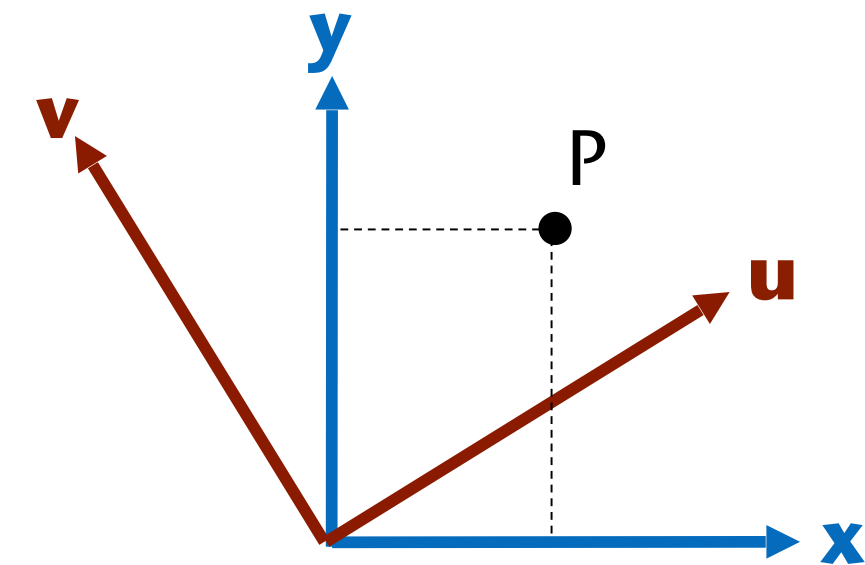
- Gegeben: Einheitsvektoren $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$

- Setze:

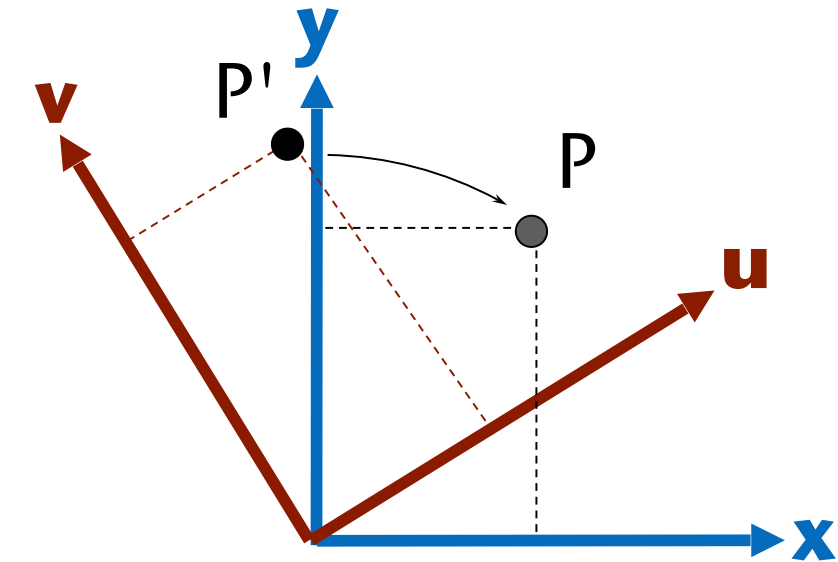
$$R = \begin{pmatrix} | & | & | \\ \mathbf{u} & \mathbf{v} & \mathbf{w} \\ | & | & | \end{pmatrix}$$

- Damit ist $R \cdot R^T = I$ und $\det(R) = 1$
- Also: R ist eine Rotation
- Und zwar von xyz-Koordinaten $\rightarrow$ uvw, denn:

$$R \cdot \mathbf{e}_x = \mathbf{u}, \quad R \cdot \mathbf{e}_y = \mathbf{v}, \quad R \cdot \mathbf{e}_z = \mathbf{w}$$



- Was bedeutet die Matrix R^{-1} ?
- Rotation so dass uvw-Achsen $\rightarrow$ xyz-Achsen übergehen



- Beachte: bei dieser Betrachtungsweise sind die Koordinaten der Punkte (bzw. Ortsvektoren) **IMMER im xyz-Koord.system** gegeben!
 - Denn auch die Vektoren **u, v, w**, haben wir immer in xyz-Koord. definiert!

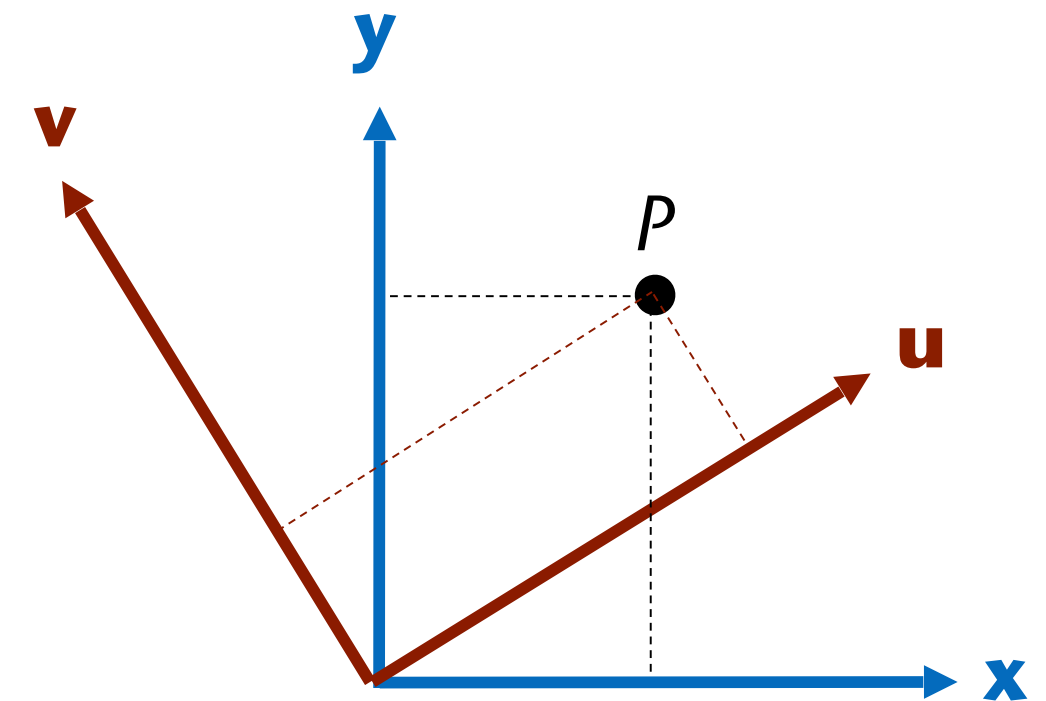
Zusammenhang zwischen Rotation und Basiswechsel

- Betrachte einen Punkt $P = \mathbf{p}_{xyz}$ im xyz-Koord.system und multipliziere mit R^T :

$$R^T \cdot P_{xyz} = R^T \cdot \mathbf{p}_{xyz} = \begin{pmatrix} - & \mathbf{u} & - \\ - & \mathbf{v} & - \\ - & \mathbf{w} & - \end{pmatrix} \cdot \mathbf{p}_{xyz} = \begin{pmatrix} \mathbf{u} \cdot \mathbf{p}_{xyz} \\ \mathbf{v} \cdot \mathbf{p}_{xyz} \\ \mathbf{w} \cdot \mathbf{p}_{xyz} \end{pmatrix} = P_{uvw}$$

➤ P_{uvw} stellt den **selben** Punkt P im Raum wie P_{xyz} dar!

- $\mathbf{p}_{uvw}$ stellt ihn in uvw-Koordinaten dar,
 $\mathbf{p}_{xyz}$ stellt ihn in xyz-Koordinaten dar!



GOOD CODERS...



Orthogonale Matrizen und der Zusammenhang zu Rotationen

- Definition (Wdhg aus Mathe):

eine Matrix R heißt **orthogonal** $\Leftrightarrow RR^T = R^T R = I$

- Folgen:

Die Spalten von R sind zueinander **orthonormal** (nicht nur orthogonal)

$$\det(R) = \pm 1$$

$$R^{-1} = R^T$$

R^T ist orthogonal

$$\|Rv\| = \|v\| \quad (\text{Längenerhaltung})$$

$$(Ru) \cdot (Rv) = u \cdot v \quad (\text{Winkelerhaltung})$$

$$R_1, R_2 \text{ sind orthogonal} \Rightarrow R_1 R_2 \text{ ist orthogonal}$$

Charakterisierung von (reinen) Rotationsmatrizen

- R ist orthogonal $\Leftrightarrow R$ ist Rotationsmatrix und/oder Spiegelung
- R ist eine **ordentliche Rotation** $\Leftrightarrow RR^T = I \wedge \det(R) = +1$
- Die Menge der orthogonalen Matrizen mit Determinante =1 bezeichnet man oft mit **SO(3)** (aka. die *Gruppe der 3D Rotationen*, mit der Multiplikation als Operator)

Zerlegung einer Rotationsmatrix (Konvertierung von Matrix in Achse+Winkel)

FYI

- Gegeben: Rotationsmatrix R

1. Aufgabe: den Rotationswinkel θ bestimmen

- Lösung: $1 + 2 \cos \theta = \text{spur}(R)$

- Beweis:

- Zu R gibt es eine Basiswechselmatrix U , so daß

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

- Es gilt: $\text{spur}(R) = \text{spur}(URU^{-1}) = \text{spur}(R_x(\theta)) = 1 + 2 \cos \theta$

(Spezielle Eigenschaft der Spur)

2. Aufgabe: Rotationsachse $\mathbf{r}$ bestimmen

- Lösung: $\mathbf{r}$ ist der Eigenvektor zum Eigenwert 1 der Matrix R
- Beweis: alle Vektoren auf der Rotationsachse bleiben fest, d.h. $R\mathbf{r} = 1 \cdot \mathbf{r}$
- Berechnung der Eigenvektoren einer 3x3-Matrix:
 - Zur Erinnerung: zu jeder Matrix A gibt es eine adjungierte Matrix $A^\#$
 - Die Elemente $a_{ij}^\#$ der **adjungierten Matrix** sind definiert als

$$a_{ij}^\# = (-1)^{i+j} \det \begin{pmatrix} a_{11} & \cdots & a_{1i} & \cdots \\ \vdots & & \vdots & \\ a_{j1} & \cdots & a_{ji} & \cdots \\ \vdots & & \vdots & \end{pmatrix}$$

- Es gilt: $AA^\# = \det(A)I$

- Falls $\det(A) = 0$, dann ist $AA^\# = 0 \cdot I = 0$
- Also gilt für jeden Spaltenvektor $\mathbf{v}$ aus $A^\#$, daß $A \cdot \mathbf{v} = 0$
- Gesucht: Eigenvektor $\mathbf{r}$ zum Eigenwert 1 von R , also ein $\mathbf{r}$, so daß $(R - I) \cdot \mathbf{r} = 0$
- Das sind gerade die Spalten von $(R - I)^\#$
- Bemerkung: alle Spalten sind Vielfache voneinander
- Bemerkung: das funktioniert auch für größere Matrizen, ist aber nicht mehr effizient

Intermezzo: ein Film, in dem viele Interpolationen von Rotationen / Orientierungen vorkommen

Air on the Dirac Strings

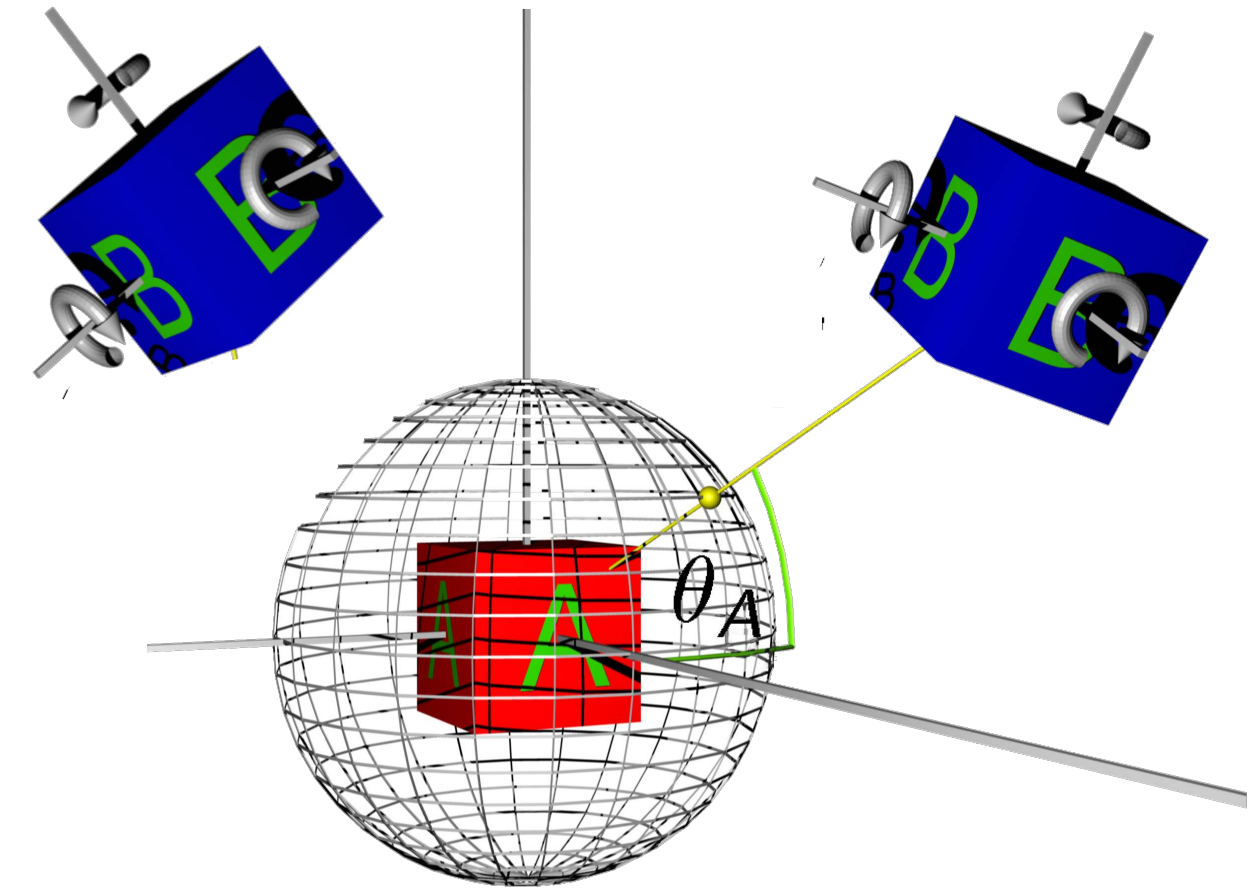
George Francis,
Louis Kauffman,
Dan Sandin, Chris Hartman,
John Hart

<http://www.evl.uic.edu/hypercomplex/>

Interpolation von Orientierungen

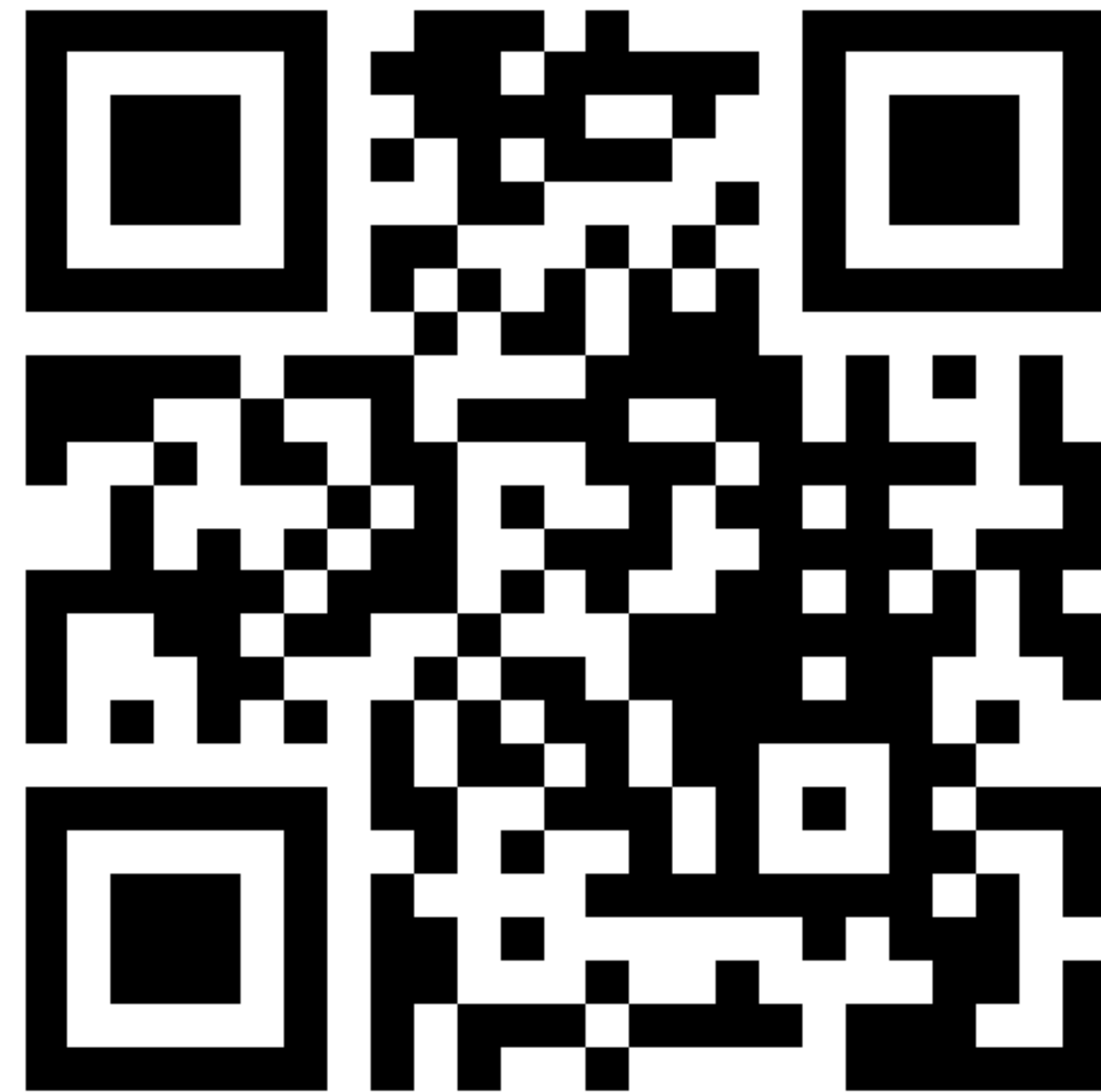
- Definition: **Orientierung**
 = "Lage" (*attitude/pose*) eines Obj.s im Raum
 = Rotation aus dessen Null-Lage

Punkt	Orientierung
Vektor	Rotation
Null-Element = (0,0,0)	Null-Element = Einheitsmatrix



- Häufige Aufgabe:
 - Gegeben: zwei Orientierungen O_1 und O_2 für dasselbe Objekt
 - Gesucht: eine Funktion $O(t)$ zur Interpolation zwischen O_1 und O_2 , die ästhetisch ansprechend und effizient ist
- Frage: welche Repräsentation ist dafür gut geeignet?

Wer hat schon einmal komplexe Zahlen kennengelernt?



<https://www.menti.com/c3vcg3bux8>

Vorbetrachtungen FYI

- Komplexe Zahlen kann man als Punkte in der Ebene betrachten
- Und als Rotation in der Ebene um den Ursprung !

$$e^{i\theta} = \cos \theta + i \sin \theta$$

$$v = r e^{i\varphi} = r \cos \varphi + r i \sin \varphi$$

$$e^{i\theta} v = r e^{i\theta} e^{i\varphi} = r e^{i(\theta+\varphi)} = r \cos(\theta + \varphi) + r i \sin(\theta + \varphi)$$

Visualisierung: siehe Video auf der VL-Homepage unter *Videos*

- Wir können reelle Zahlen leicht invertieren: $x \cdot x^{-1} = 1$
- Auch für komplexe Zahlen können wir ein Inverses finden: $\frac{z \cdot z^*}{|z|^2} = 1$
- Gibt es etwas Analoges auch in "höheren Dimensionen"? → **Nein!**
 - Z.B. eine "dreidimensionale" Verallgemeinerung von $\mathbb{C}$?
- Aber: im 4D klappt es wieder ... (fast)

- Erweiterung der komplexen Zahlen:

$$\mathbb{H} = \{ q \mid q = w + a \cdot \mathbf{i} + b \cdot \mathbf{j} + c \cdot \mathbf{k} , w, a, b, c \in \mathbb{R} \}$$

- Alternative Schreibweise:

$$q = (w, \mathbf{v})$$

- Axiome für die 3 **imaginären Einheiten**:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

$$(\mathbf{ij})\mathbf{k} = \mathbf{i}(\mathbf{jk})$$

- Daraus folgen sofort diese Rechengesetze:

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k}$$

$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i}$$

$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j}$$

Eine Algebra über den Quaternionen

- Addition: $q_1 + q_2 = (w_1 + w_2) + (a_1 + a_2)\mathbf{i} + (b_1 + b_2)\mathbf{j} + (c_1 + c_2)\mathbf{k}$
- Skalierung: $s \cdot q = (sw) + (sa)\mathbf{i} + (sb)\mathbf{j} + (sc)\mathbf{k}$
- Multiplikation:
$$\begin{aligned} q_1 \cdot q_2 &= (w_1 + a_1\mathbf{i} + b_1\mathbf{j} + c_1\mathbf{k}) \cdot (w_2 + a_2\mathbf{i} + b_2\mathbf{j} + c_2\mathbf{k}) \\ &= (w_1w_2 - a_1a_2 - b_1b_2 - c_1c_2) + \\ &\quad (w_1a_2 + w_2a_1 + b_1c_2 - c_1b_2)\mathbf{i} + \\ &\quad (\dots \dots)\mathbf{j} + \\ &\quad (\dots \dots)\mathbf{k} \end{aligned}$$
- Konjugation: $q^* = w - a\mathbf{i} - b\mathbf{j} - c\mathbf{k}$
- Betrag (Norm): $|q|^2 = w^2 + a^2 + b^2 + c^2 = q \cdot q^*$
- Inverse eines Einheitsquaternions: $|q| = 1 \Rightarrow q^{-1} = q^*$

- Behauptung (o. Bew.):

$\mathbb{H}$ mit der Multiplikation ist eine **nicht**-kommutative Gruppe.

- Bemerkung: manchmal ist es zweckmäßig, die Multiplikation zweier Quaternionen auch mit Hilfe einer Matrix-Multiplikation darzustellen

$$q_1 \cdot q_2 = \begin{pmatrix} w_1 & -a_1 & -b_1 & -c_1 \\ a_1 & w_1 & -c_1 & b_1 \\ b_1 & c_1 & w_1 & -a_1 \\ c_1 & -b_1 & a_1 & w_1 \end{pmatrix} \begin{matrix} \uparrow \\ q_2 \end{matrix} = \begin{pmatrix} w_2 & -a_2 & -b_2 & -c_2 \\ a_2 & w_2 & c_2 & -b_2 \\ b_2 & -c_2 & w_2 & a_2 \\ c_2 & b_2 & -a_2 & w_2 \end{pmatrix} \begin{matrix} \uparrow \\ q_1 \end{matrix}$$

Als Spaltenvektor geschrieben! Als Spaltenvektor!

Einbettung des 3D-Vektorraumes in $\mathbb{H}$

- Den Vektorraum $\mathbb{R}^3$ kann man in $\mathbb{H}$ so einbetten:

$$\mathbf{v} \in \mathbb{R}^3 \mapsto q_v = (0, \mathbf{v}) \in \mathbb{H}$$

- Definition:
Quaternionen der Form $(0, \mathbf{v})$ heißen **reine Quaternionen** (*pure quaternions*)

Darstellung von Rotationen mittels Quaternionen

- Gegeben sei Axis & Angle $(\varphi, \mathbf{r})$ mit $\|\mathbf{r}\| = 1$
- Definiere das dazu gehörige Quaternion als

$$q = \left(\cos \frac{\varphi}{2}, \sin \frac{\varphi}{2} \mathbf{r} \right) = \left(\cos \frac{\varphi}{2}, \sin \frac{\varphi}{2} r_x, \sin \frac{\varphi}{2} r_y, \sin \frac{\varphi}{2} r_z \right)$$

- Beobachtung: $|q| = 1$ (rechnet nach!)
- Zurückrechnen: $q = (w, a, b, c)$ sei gegeben, mit $|q| = 1$
Dann ist

$$\varphi = 2 \arccos(w) \quad \mathbf{r} = \frac{1}{\sin \frac{\varphi}{2}} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \frac{1}{\sqrt{1 - w^2}} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

- Satz:
Jedes Einheitsquaternion kann man in der Form $(\cos \frac{\varphi}{2}, \sin \frac{\varphi}{2} \mathbf{r})$ darstellen.
- Beweis: siehe vorige Folie
- Theorem: **Rotation mittels eines Quaternions**
Sei $\mathbf{v} \in \mathbb{H}$ ein pures Quaternion und $q \in \mathbb{H}$ ein Einheitsquaternion. Dann beschreibt die Abbildung ("sandwich product")
$$R : \mathbf{v} \mapsto q \cdot \mathbf{v} \cdot q^* = \mathbf{v}'$$
eine (rechtshändige) Rotation von $\mathbf{v}$, wobei Winkel und Achse durch q bestimmt sind.

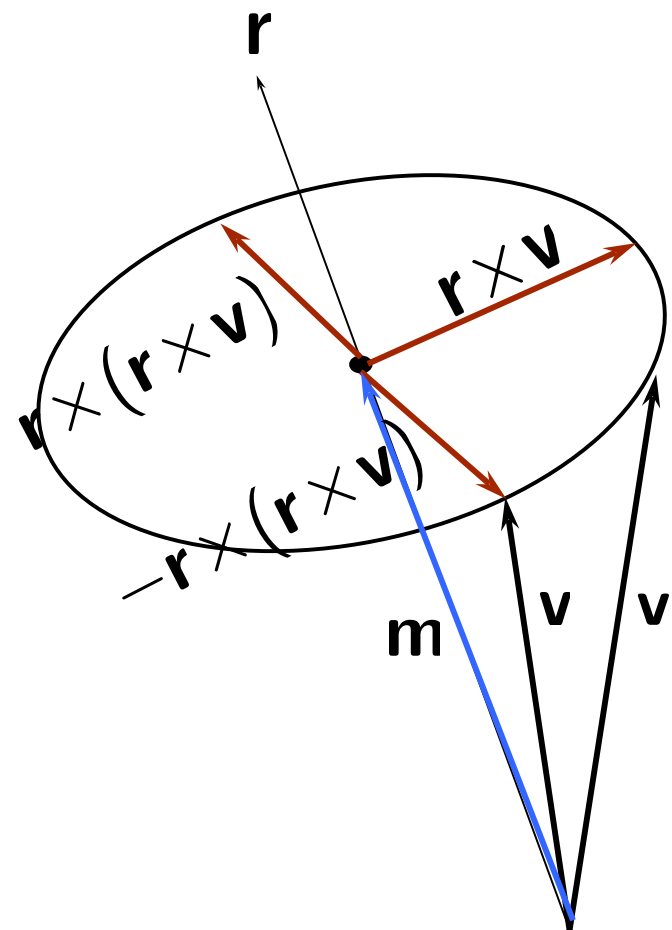
$$q\mathbf{v}q^* = (c, s\mathbf{r}) \cdot (0, \mathbf{v}) \cdot (c, -s\mathbf{r}) \quad \text{mit} \quad c = \cos \frac{\varphi}{2}, s = \sin \frac{\varphi}{2}$$

$$= \dots (*)$$

$$= (0, \underbrace{\mathbf{v} + \sin \varphi \cdot \mathbf{r} \times \mathbf{v} + (1 - \cos \varphi) \cdot \mathbf{r} \times (\mathbf{r} \times \mathbf{v})})$$

||

$$\underbrace{\mathbf{v} + \mathbf{r} \times (\mathbf{r} \times \mathbf{v})}_{\mathbf{m}} + \sin \varphi \cdot \mathbf{r} \times \mathbf{v} + \cos \varphi \cdot (-\mathbf{r} \times (\mathbf{r} \times \mathbf{v})) = \mathbf{v}'$$



*) Zwischendurch benötigt man diese trigonometrischen Identitäten:

$$\sin \varphi = 2 \sin \frac{\varphi}{2} \cos \frac{\varphi}{2} \quad 1 - \cos \varphi = 2 \sin^2 \frac{\varphi}{2}$$

- Bemerkung: die so definierte Rotationsabbildung ist mit der Quaternionen-Multiplikation **verträglich**, d.h., dass

$$R_{q_1}(R_{q_2}(\mathbf{v})) = R_{q_1 \cdot q_2}(\mathbf{v})$$

- Caveat: die beiden Einheits-Quaternionen $q = (w, x, y, z)$ und $-q = (-w, -x, -y, -z)$ stellen die **selbe** Rotation dar!
 - M.a.W.: die 4-dim. Einheitskugel in $\mathbb{H}$ deckt die Gruppe aller Rotationen, $SO(3)$, **doppelt** ab!

Lineare Interpolation von Quaternionen

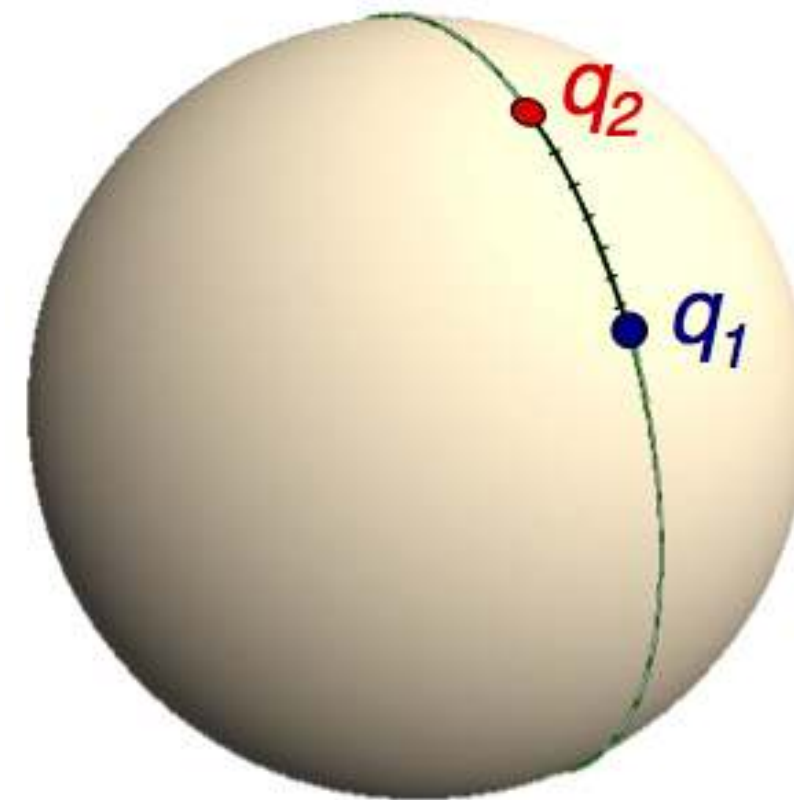
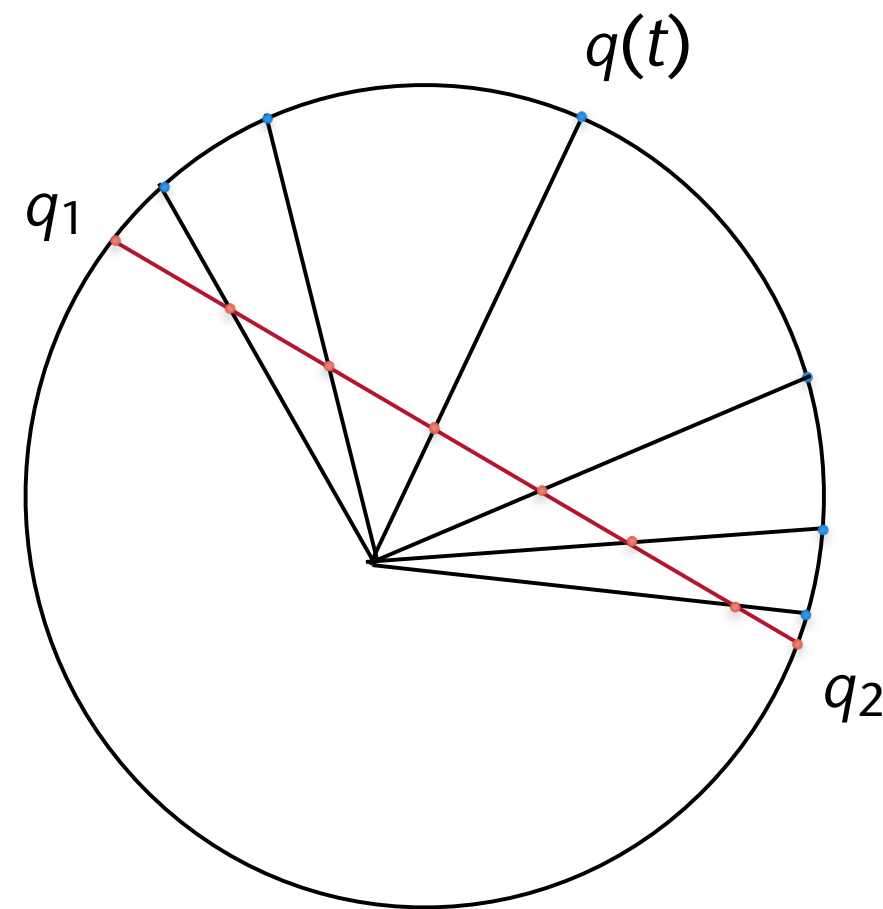
- Gegeben: zwei Orientierungen q_1 , q_2
- Aufgabe: Orientierungen dazwischen interpolieren
- Einfachste Lösung: linear interpolieren

$$q(t) = (1 - t)q_1 + tq_2 =: \text{lerp}(t; q_1, q_2)$$

- Wichtig: $q(t)$ hinterher immer **normieren**!
- Vorteil: **Kein Gimbal Lock!**

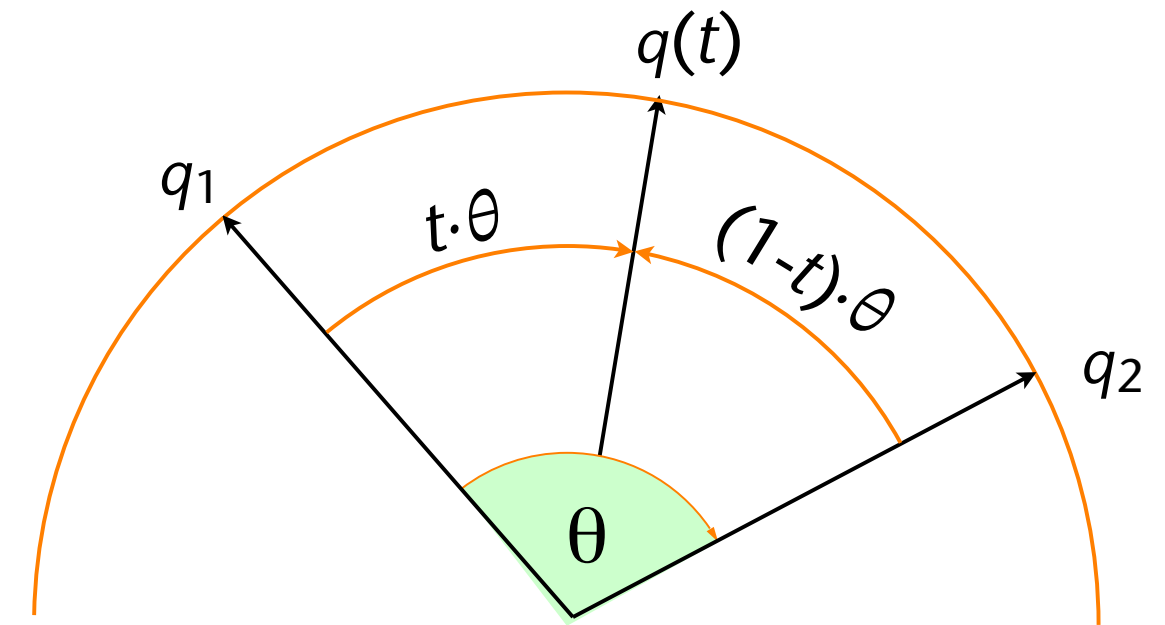
Sphärische lineare Interpolation

- Nachteil (noch): keine konstante Winkelgeschwindigkeit
- Problem: Geschwindigkeit an den "Enden" der Interpolation ist langsamer als in der "Mitte"



- Besser ist die **sphärische lineare Interpolation**
“*slerp*”

- Ansatz: interpoliere nicht die *Distanz* zwischen q_1 und q_2 , sondern den *Winkel* dazwischen



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

mit $\cos \theta = q_1 \odot q_2$ Skalarprodukt der Vektoren q_1 und q_2

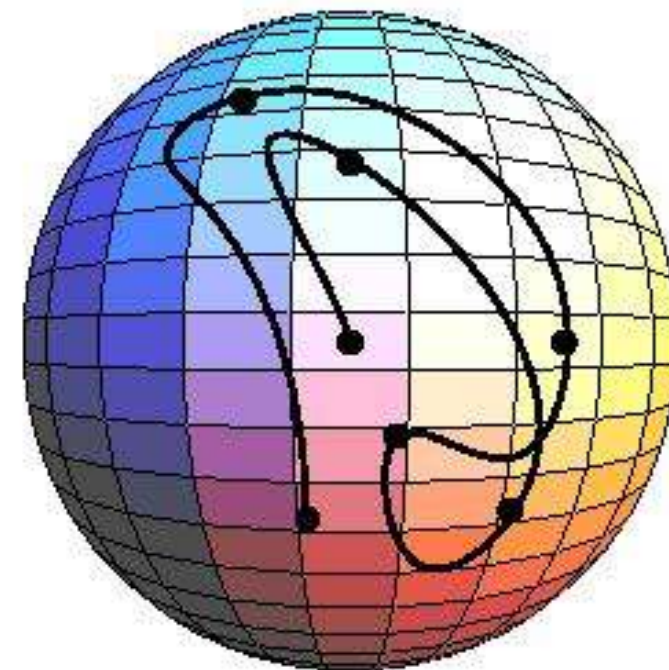
Nebenbemerkung

- In der Praxis interpoliert man die Quaternionen mit höherem Grad, mit irgend einem Standard-Interpolationsverfahren, das parametrische Kurven liefert
- Z.B. Bezier-Kurven (oder B-Splines)
- Mit De Casteljau geht das sehr einfach:
 - Statt

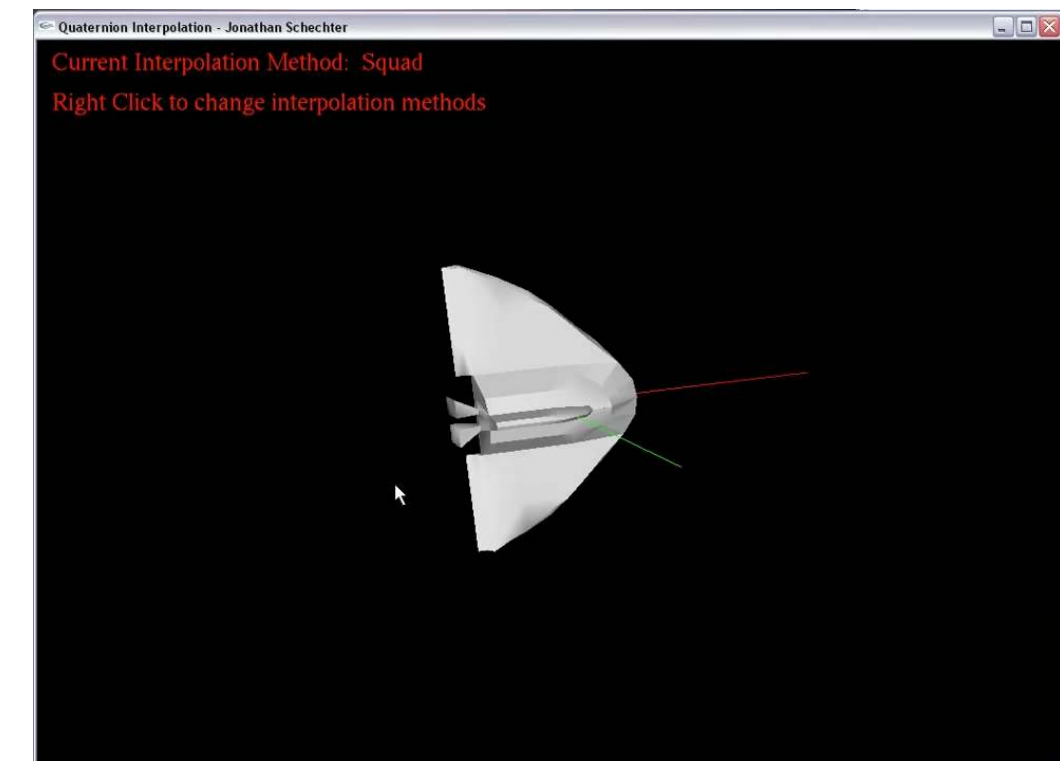
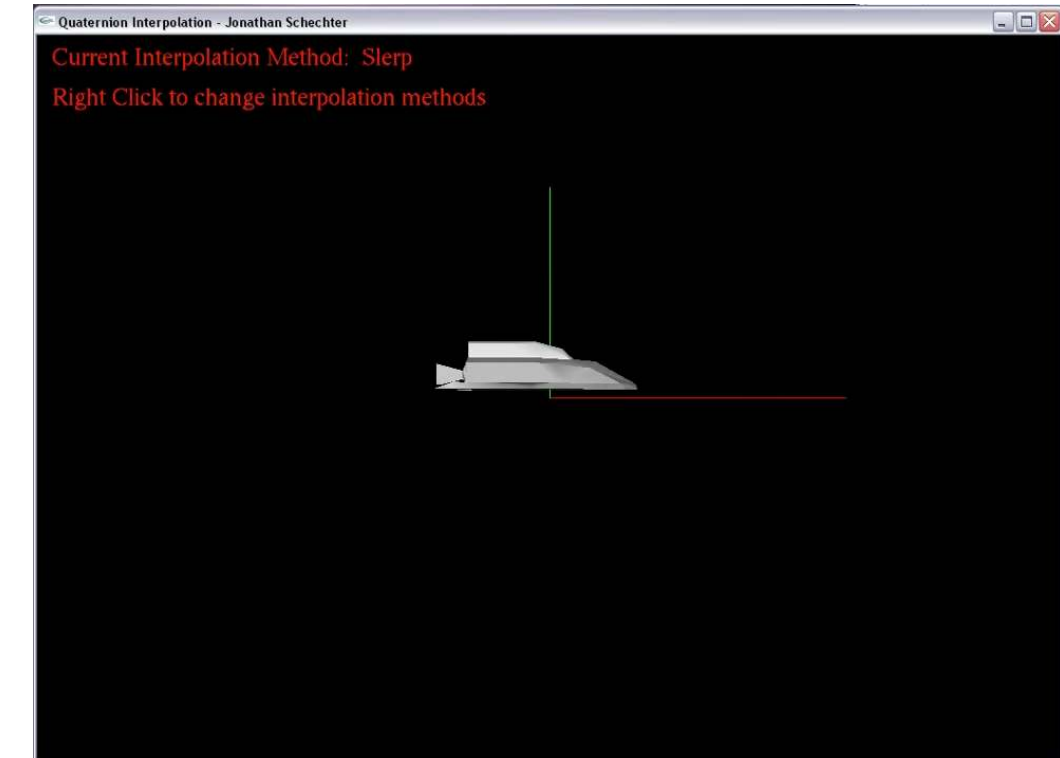
$$Q_i = (1 - t)P_i + tP_{i+1}$$

berechne fortgesetzt

$$Q_i = \text{slerp}(t; P_i, P_{i+1})$$



Lineare Interpolation mit slerp



Kubische Interpolation mit slerp

Vergleich der verschiedenen linearen Interpolationsarten

Interpolation
of Euler angles



Naïve interpolation
of matrices



Slerp of
quaternions

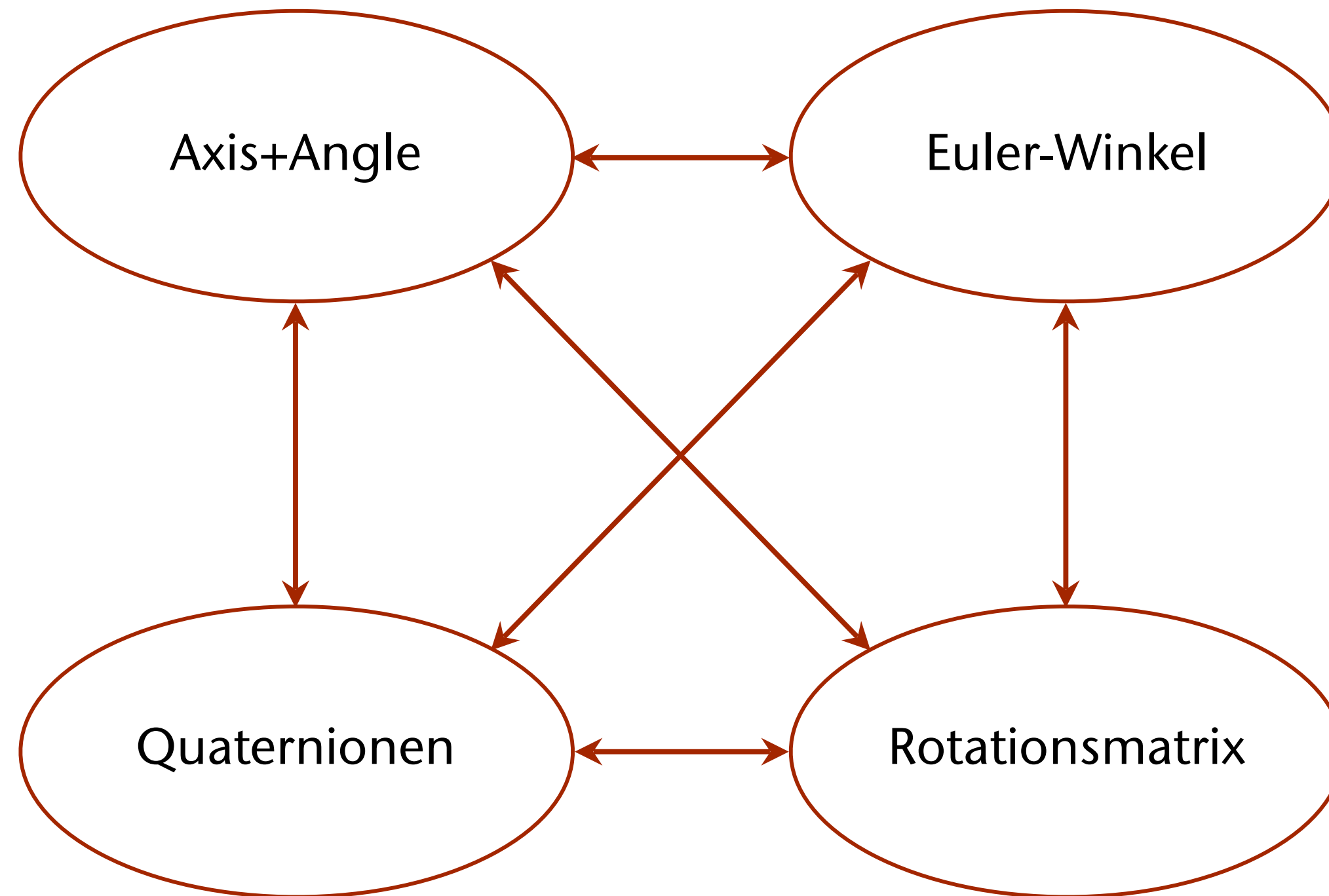


Lerp of quaternions
(with normalization)



Gianluca Vatinno, Trinity College Dublin

Alle Darstellungen von Rotation lassen sich ineinander umrechnen



Mehr Infos: siehe die Tutorials auf der Homepage der Vorlesung!

Vergleich der verschiedenen Arten zur Darstellung von Rotationen

Matrix	Euler-Winkel
Quaternionen	Achse+Winkel

Vor- und Nachteile der verschiedenen Repräsentationen von Rotationen



<https://www.menti.com/>

Vergleich der verschiedenen Arten zur Darstellung von Rotationen

Matrix - o Ubiquitär in OpenGL / Game Engines - 16 Floats (evtl. 12) + Vektor rotieren = 15 FLOPs + Einheitliche Repräsentation für alle affinen Transformationen - Rundungsfehler nach mehrfacher Konkatination lassen sich nicht leicht korrigieren - Konkat. von Rotationen = 45 FLOPs	Euler-Winkel + Intuitiv (zunächst) + 3 Floats - Gimbal lock - keine Algebra (Rechenop.) darauf → Konkatination fkt. nicht direkt
Quaternionen + Intuitiv + 4 Floats + Rundungsfehler nach mehrfacher Konkatination lassen sich leicht korrigieren (einfach normieren) + Konkat. von Rotationen = 28 FLOPs - Vektor rotieren = 30 FLOPs	Achse+Winkel + Sehr intuitiv + 3 Floats - keine Algebra (Rechenop.) darauf - Vektor rotieren = 40 FLOPs

Anwendung: virtueller Trackball

- Interaktionsaufgabe: ein Objekt um eine beliebige Achse rotieren
- Mit echtem Trackball ist es trivial
- Wie gibt man beliebige Rotationen mit der 2D-Maus ein?



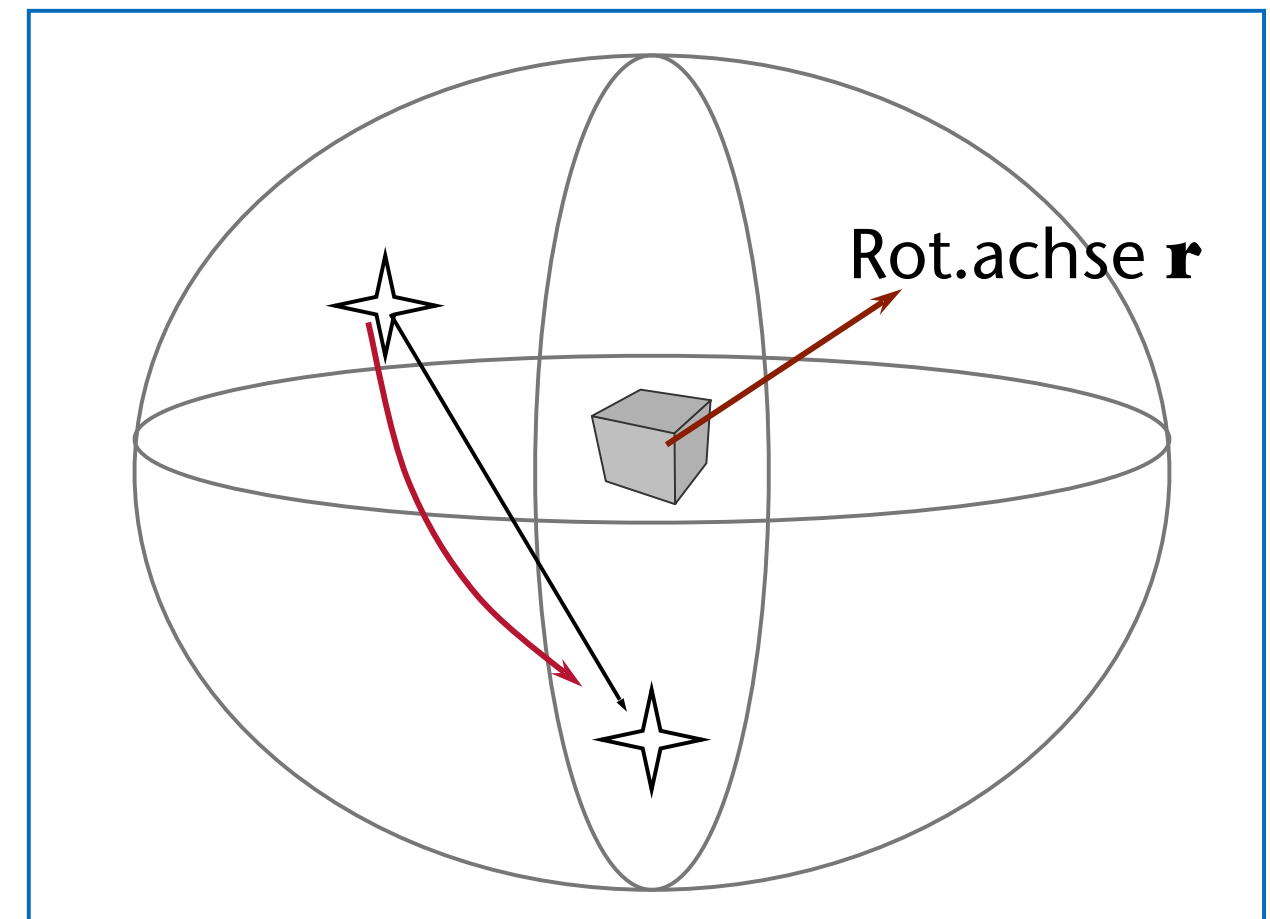
Die Interaktionsmetapher

- Idee:
 - Lege gedachte (virtuelle) Kugel um das Objekt
 - Kugel kann um ihr Zentrum rotieren
 - Maus pickt Punkt auf Oberfläche, den man zieht
- Geg.: Startpunkt = (x_1, y_1) , Endpunkt = (x_2, y_2) (in 2D!)
- Ges.: Rotationsachse $\mathbf{r}$ (in 3D), Rotationswinkel
- Berechnung:

1. Bestimme 3D Punkte

$$\mathbf{p}_i = (x_i, y_i, z_i) \quad z_i = \sqrt{1 - (x_i^2 + y_i^2)}$$

2. Rotationsachse $\mathbf{r} = \mathbf{p}_1 \times \mathbf{p}_2$

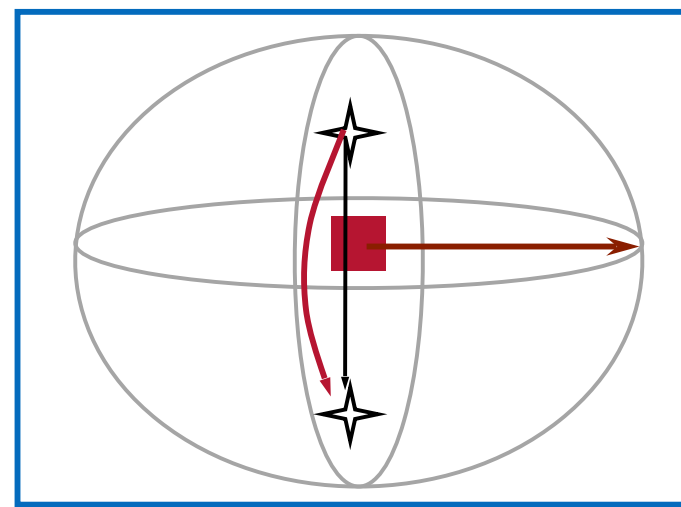


Gedachter Weg
auf der Kugel
= Segment des
Großkreises

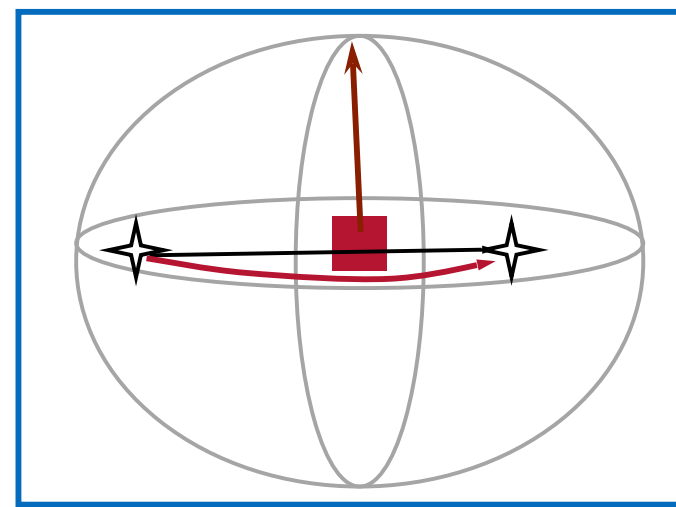
Weg der
Maus im
Fenster

Bemerkungen

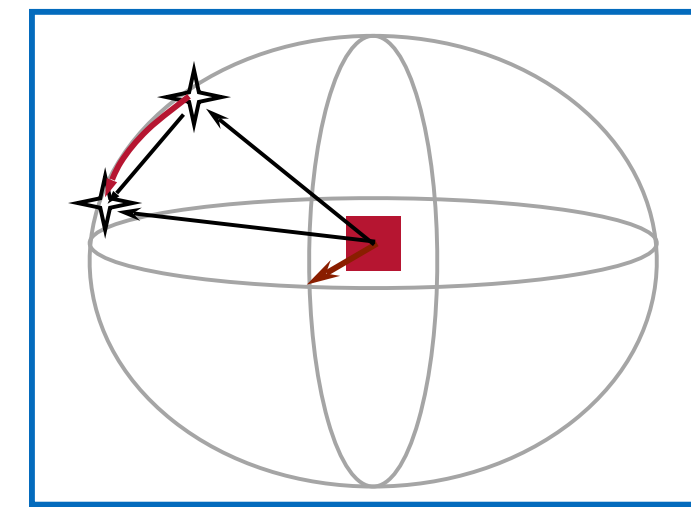
- Variante 1: $\mathbf{p}_1$ = erster Mausklick, $\mathbf{p}_2$ = aktuelle Mausposition: unintuitiv
- Variante 2: $\mathbf{p}_1$ = Mausposition vom vorigen Frame, $\mathbf{p}_2$ = aktuelle Mausposition: intuitiv, aber Rotation exakt um Z-Achse unmöglich



X



Y



Z

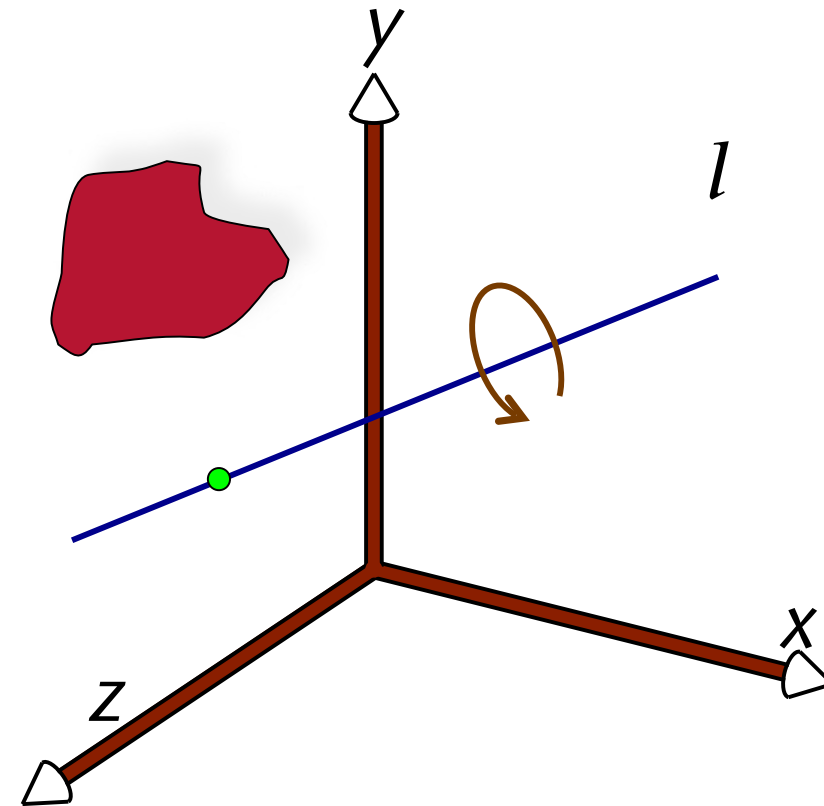
- Verbesserungen:
 - "Spinning trackball" vermeidet häufiges Nachfassen
 - "Snapping" für exaktes Rotieren um eine Koord.achse
 - Was macht man, wenn $\mathbf{p}_2$ die Ellipse verlässt? → andere 3D-Fläche verwenden, die an der Silhouette stetig anschließt

Bemerkungen

- Die Rotationsachse $\mathbf{r}$ ist zunächst im Kamera-Koordinatensystem definiert!
 - Sie muss aber nach Weltkoordinaten oder Obj.koordinaten zurückgerechnet werden (je nach dem, ob diese Rotation als letztes oder als erstes auf das Obj angewendet werden soll)
- Achtung: bei Variante 2 (inkrementeller Trackball) werden sehr viele Rotationen mit sehr kleinen Winkeln akkumuliert (pro Frame eine kleine Rotation) → achte auf numerische Robustheit und Drift

Rotation um eine *beliebige* Achse im Raum

- Man möchte mit θ um die Gerade l rotieren, Gerade geht durch den *beliebigen* Punkt p



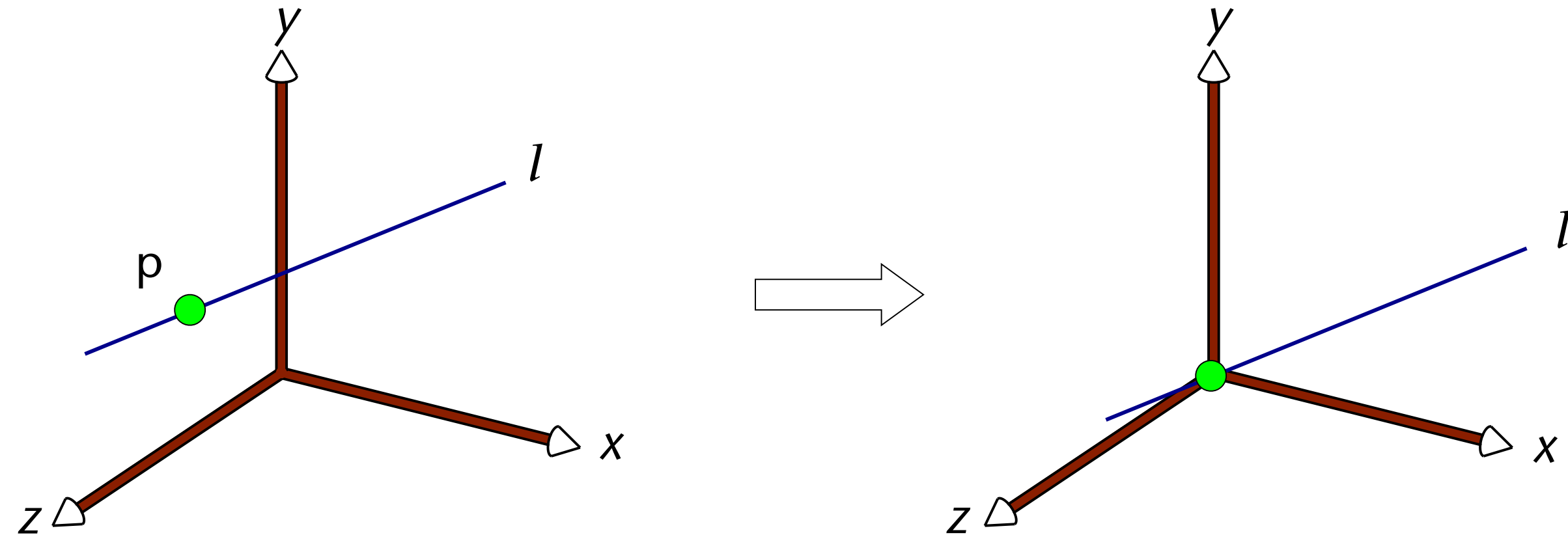
- Gesucht: eine Matrix M , die diese Transformation enthält
- Wir wissen, wie man um eine Koordinatenachse rotiert
- Somit müssen wir die Szene in eine Situation transformieren, mit der wir umgehen können

Grundidee

1. Verschiebe einen Punkt der Gerade in den Ursprung
2. Rotiere um eine Achse, so daß l in einer Koordinatenebene liegt
3. Rotiere um eine weitere Achse, so daß l auf einer Koordinatenachse liegt
4. Rotiere um diese Achse mit Winkel θ
5. Invertierte Rotation um die Koordinatenachse aus Schritt 3
6. Invertierte Rotation um die Koordinatenachse aus Schritt 2
7. Invertiere Verschiebung aus Schritt 1, so daß l wieder in Ausgangsposition

Schritt 1

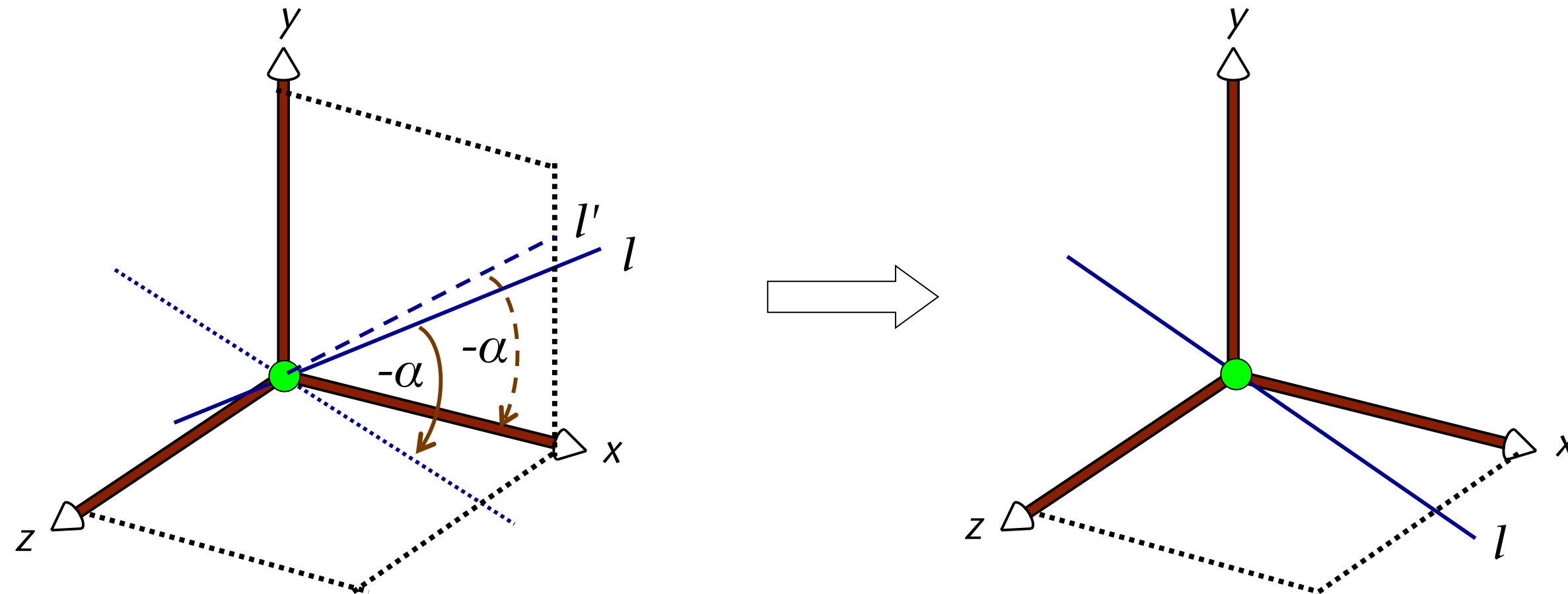
- Verschiebe Gerade, so daß ein Punkt von l im Ursprung liegt:



$$T_1 = \begin{pmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Schritt 2

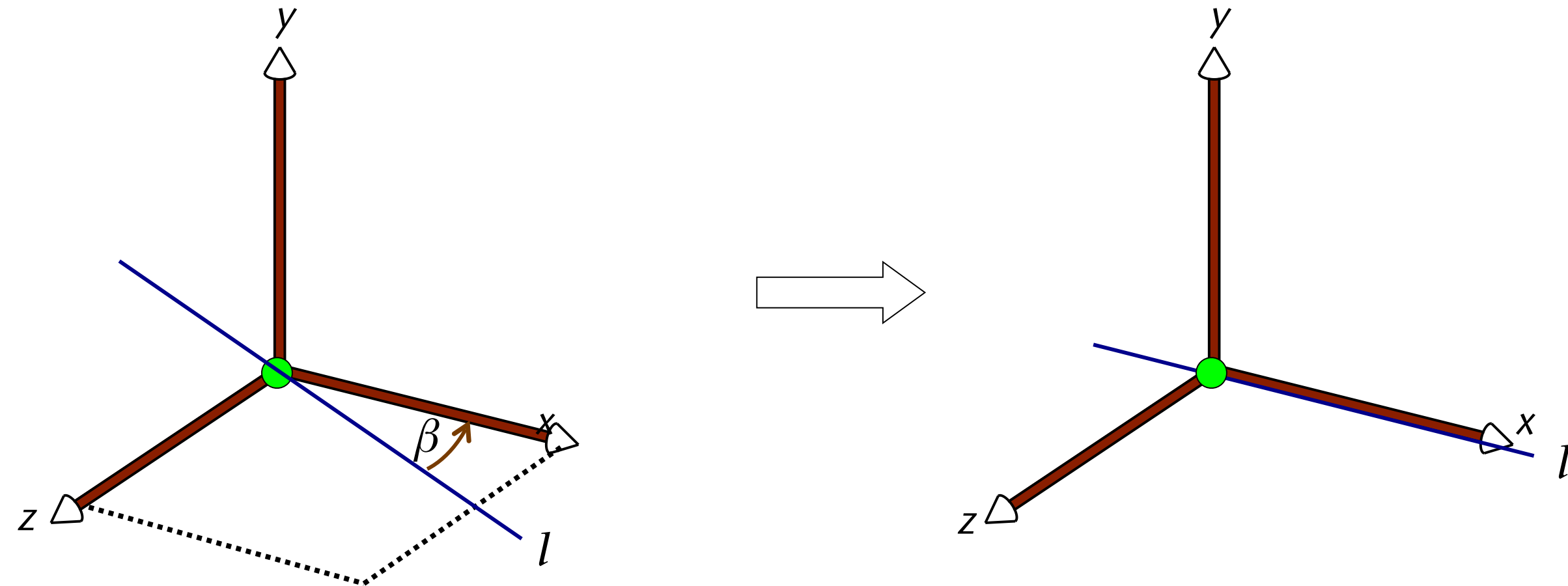
- Rotiere mit $-\alpha$ um die z-Achse, so daß l in der xz-Ebene liegt



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

Schritt 3

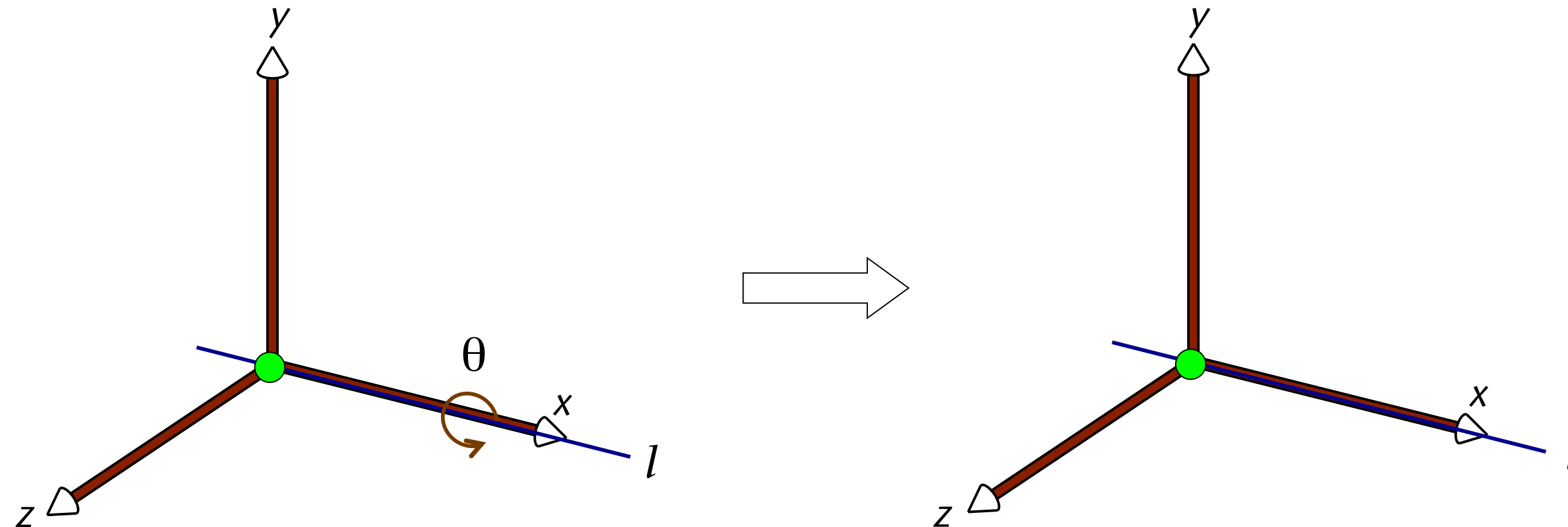
- Rotiere mit β um die y -Achse damit Gerade auf der x -Achse liegt



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

Schritt 4

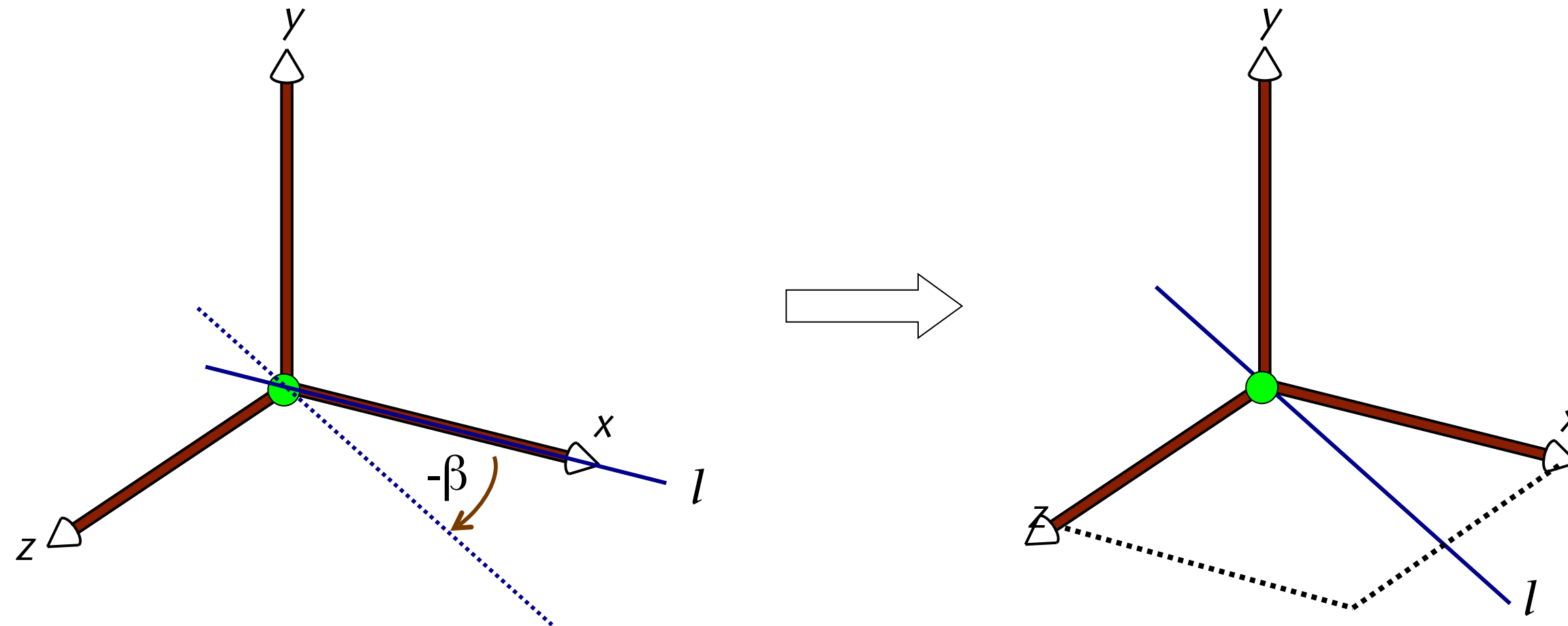
- Durchführen der gewünschten Rotation (rotiere mit θ um x -Achse)



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

Schritt 5

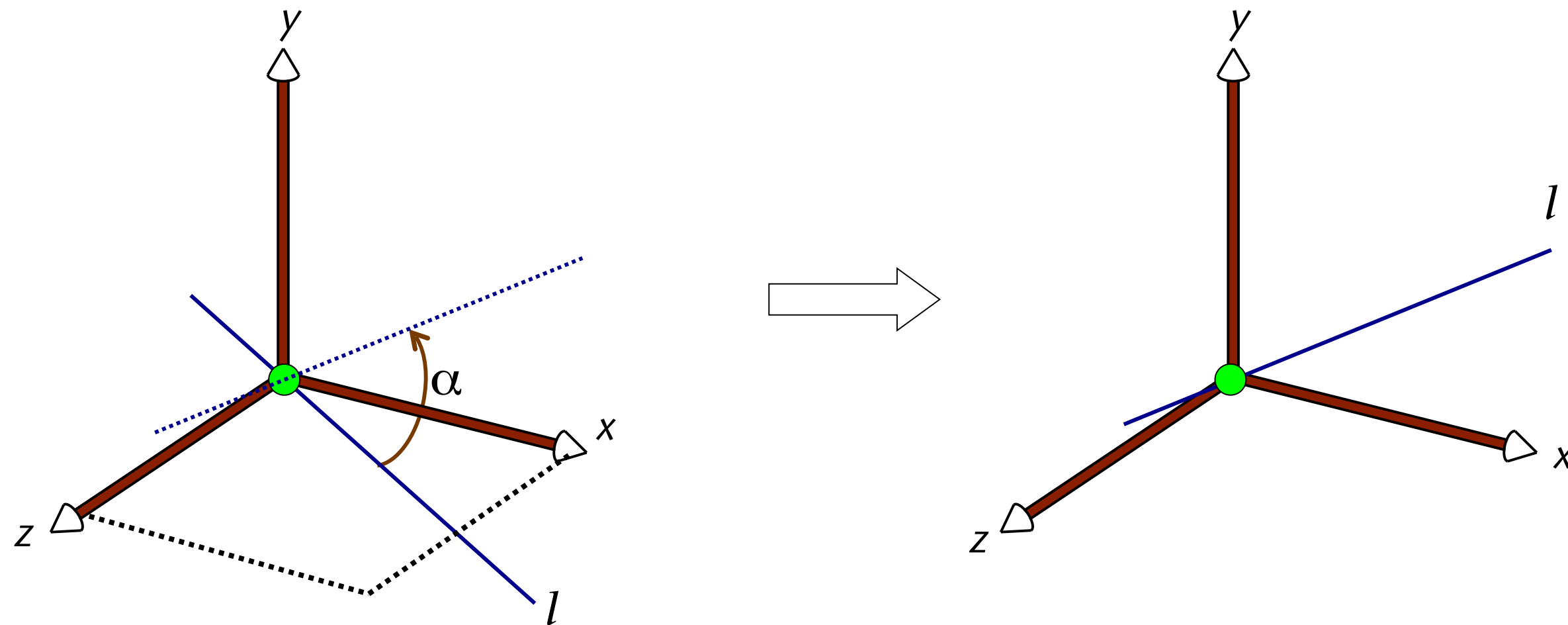
- Invertiere Rotation von l aus Schritt 3: rotiere mit $-\beta$ um die y -Achse



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

Schritt 6

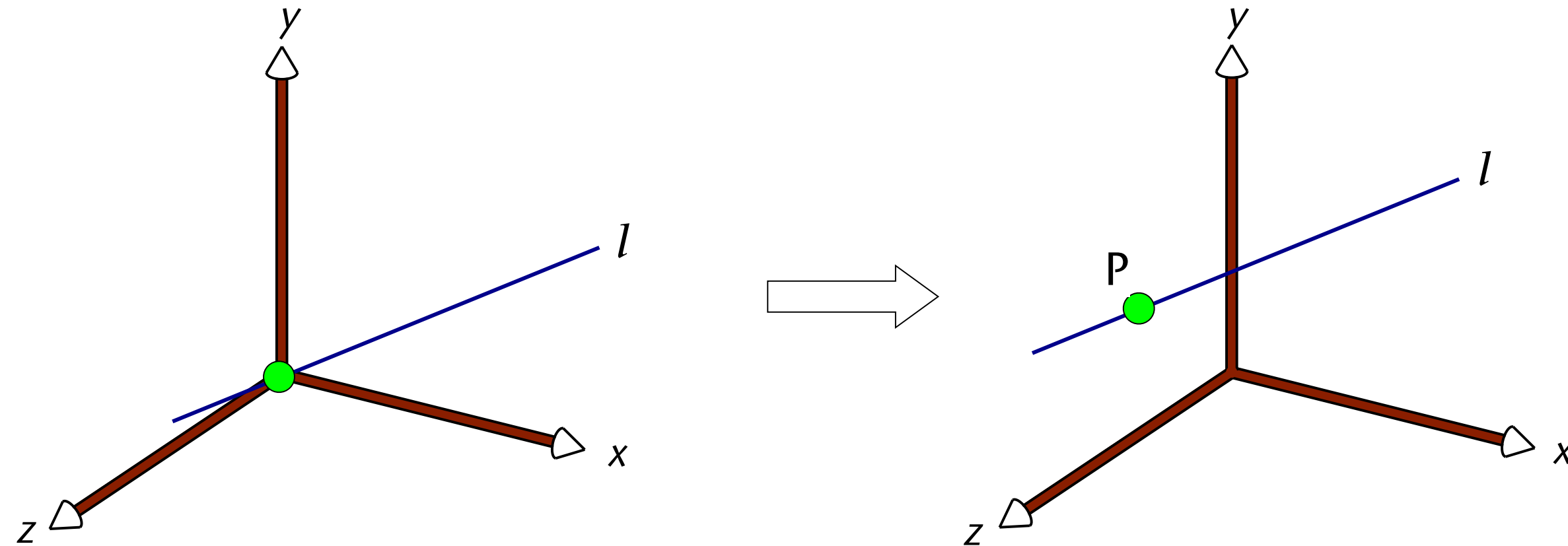
- Invertiere Rotation aus Schritt 2: rotiere mit α um z-Achse



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

Schritt 7

- Invertiere die Translation aus Schritt 1



$$T_2 = \begin{pmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Zusammenfassung

- Die vollständige Transformation zum Rotieren um eine beliebige Achse ist:

$$R_{arb} = T_2(p_x, p_y, p_z) R_z(\alpha) R_y(-\beta) R_x(\theta) \cdot \\ R_y(\beta) R_z(-\alpha) T_1(-p_x, -p_y, -p_z)$$

- (Es gibt auch andere Varianten)
- Hat man diese Matrix, so wendet man diese auf jeden Vertex des Objektes an, was den Effekt der Rotation dieses Objektes um die vorgegebene Achse hat
- Das überläßt man natürlich dem Vertex-Shader