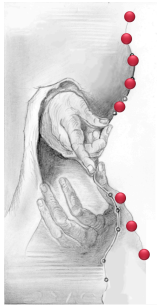
Bremen



CG VR

Computer-Graphik I
Transformationen

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

G. Zachmann

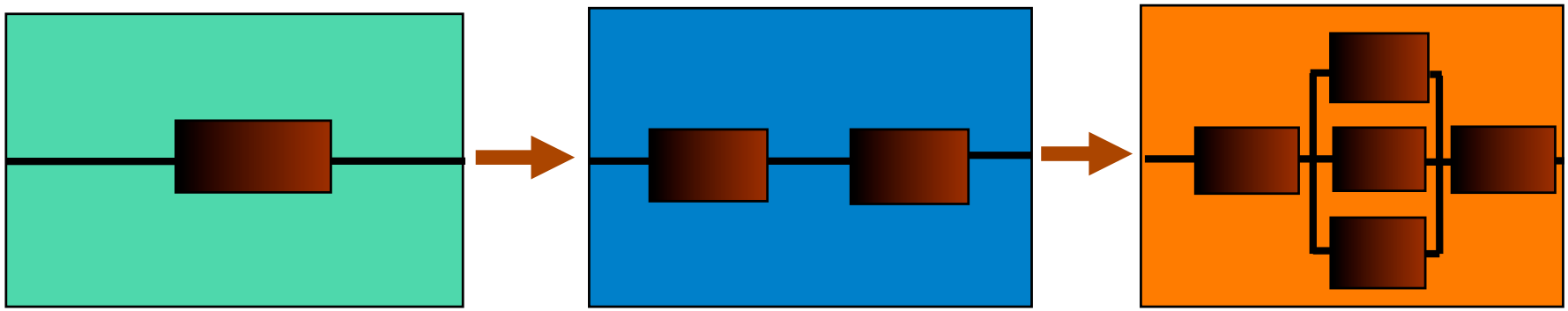
University of Bremen, Germany

cgvr.informatik.uni-bremen.de

Motivation

- Transformationen werden benötigt, um ...
  - Objekte, Beleuchtung und Kamera zu positionieren und animieren;
  - alle Berechnungen im selben Koordinatensystem durchzuführen;
  - Objekte zu projizieren.
- Teilaspekt: Viewing-Transformation = welche Transformationen muß man verwenden, um die 3D-Welt auf den 2D-Bildschirm zu projizieren
- OpenGL verwendet 4x4-Matrizen zur Spezifikation von Transormationen (warum?)

Die Graphik-Pipeline (stark vereinfacht)



Anwendung

Geometrie-Stufe

Alle Berechnungen, die 1x pro Polygon oder pro Vertex (Ecke) durchgeführt werden
Z.B.:
- Modell- und Viewing-Transformation
- Projektion
- Beleuchtung
- Clipping

Arbeitet im 3D

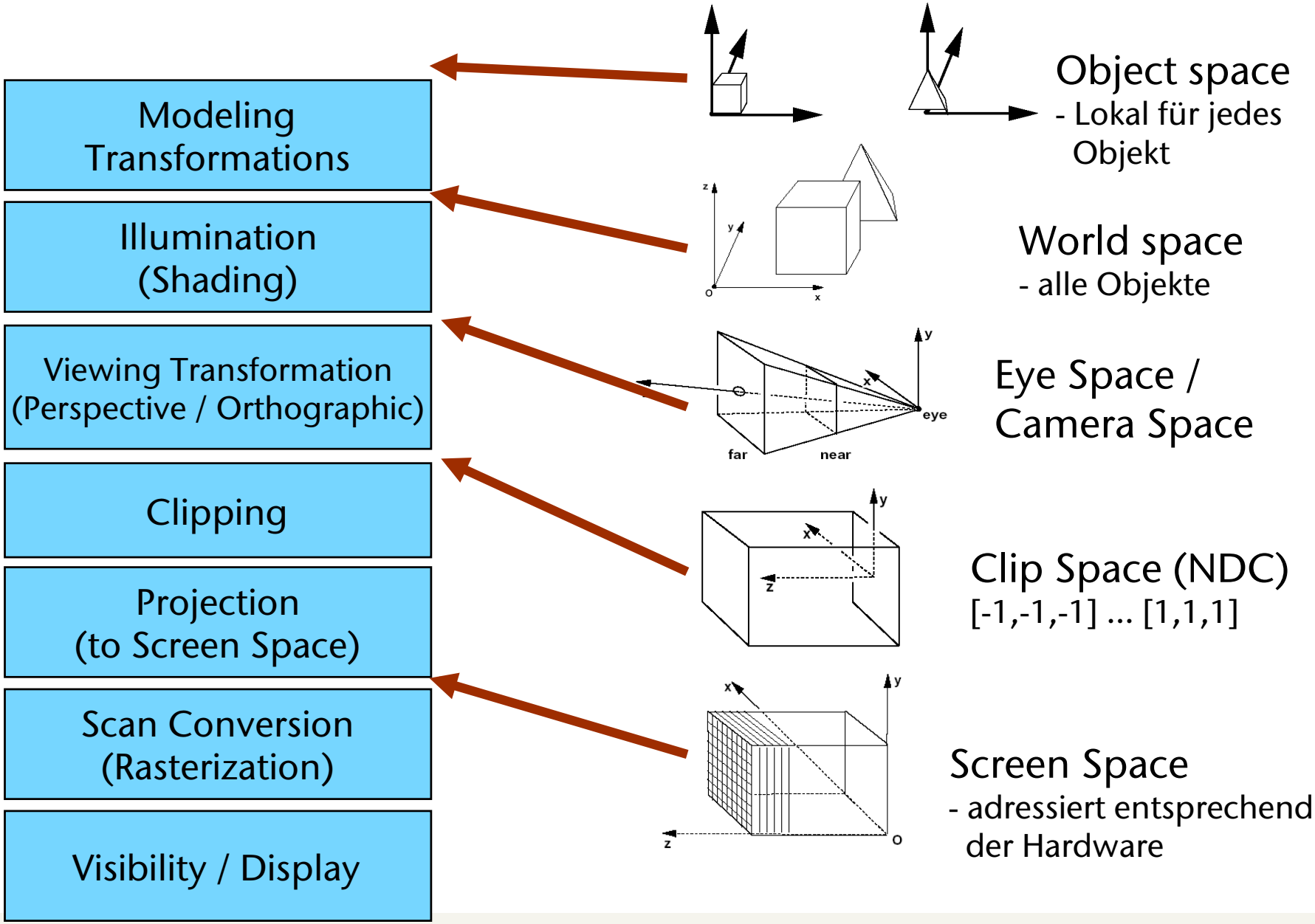
Im folgenden diese Tasks

Raster-Stufe

Kennen wir (teilweise) schon
Z.B.:
- Scan Conversion

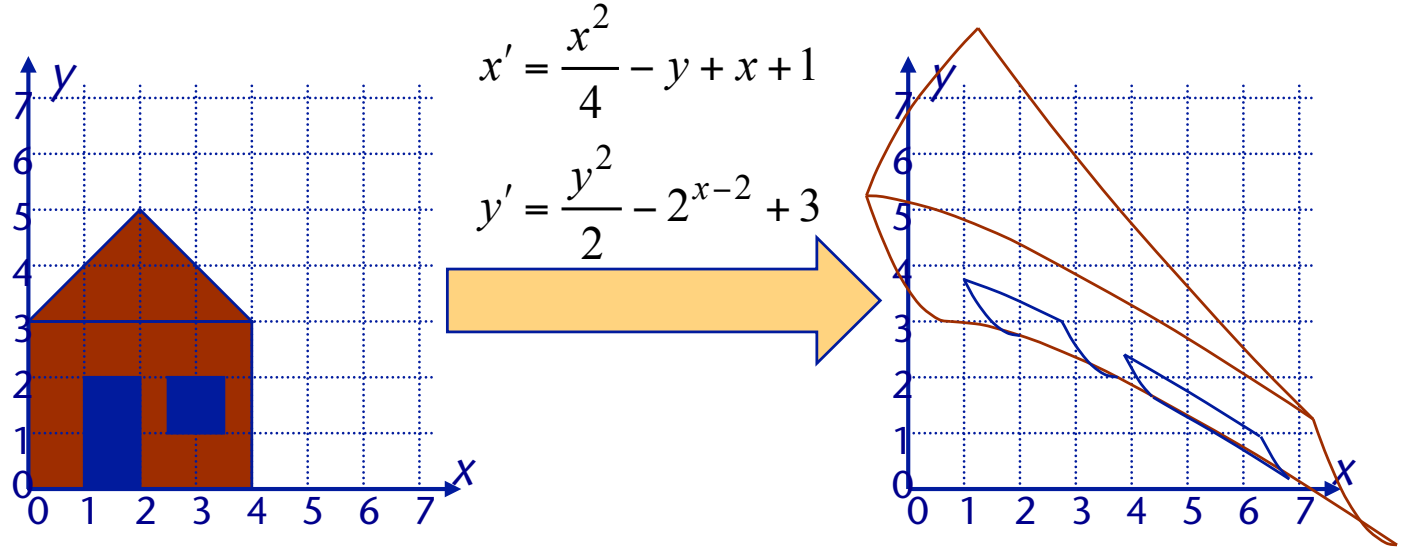
Arbeitet im 2D

Koordinatensysteme in der Pipeline



Allgemeine Transformationen

- Allgemeine Transformation ist evtl. nicht linear → Geraden werden i.A. nicht wieder auf Geraden abgebildet
- Beispiel:



- Folge: jeder Punkt (auf einer Linie, in einem Polygon, ...) muss transformiert werden → nicht effizient, nicht interessant für Echtzeit-Graphik

Grundtransformationen im 3D

- Rotation
- Skalierung
- Scherung (kommt in der Praxis fast nie vor)
- Translation

- Verkettung (*concatenation*)
- Starrkörpertransformation (*rigid body transformation*)

Elementare Rotation

- Rotation um x-, y-, z-Achse um Winkel ϕ

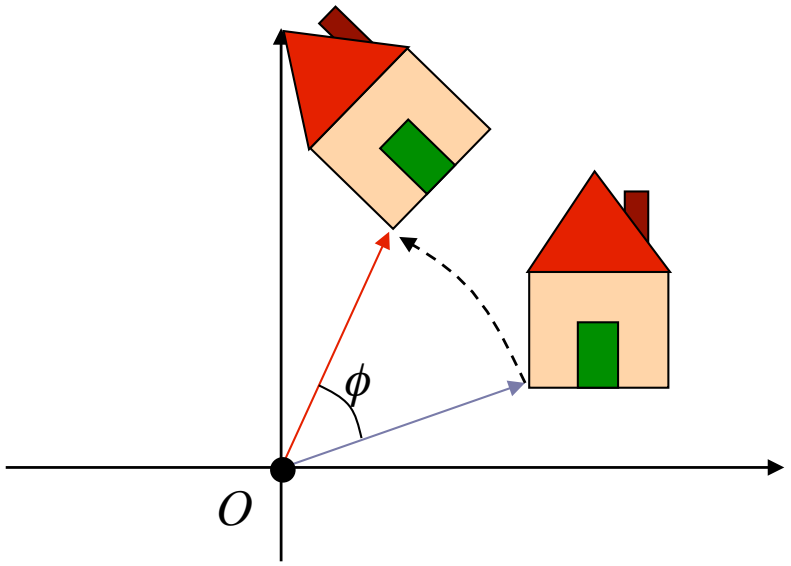
$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & ?\cos\phi & ?\sin\phi \\ 0 & ?\sin\phi & ?\cos\phi \end{pmatrix}$$

X-Koord. bleibt unverändert

Vorzeichentest: ϕ=90 →
y geht nach z, z geht nach -y.

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

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



Orthogonale Matrizen

- Definition (Wdhg aus Mathe): eine Matrix R heißt orthogonal $\Leftrightarrow$

$$RR^T = R^T R = I$$

- Folgen:

$\det(R) = \pm 1$

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

R^{T} ist orthogonal

$\|Rv\| = \|v\|$ (Längenerhaltung)

$(Ru)\cdot(Rv) = u\cdot v$ (Winkelerhaltung)

R_1, R_2 sind orthogonal $\Rightarrow$ $R_1 R_2$ ist orthogonal

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

Charakterisierung von (reinen) Rotationsmatrizen

- R ist orthogonal $\Leftrightarrow$ R ist Rotationsmatrix und/oder Spiegelung
- R ist eine ordentliche Rotation $\Leftrightarrow$

$$RR^{\mathsf{T}} = I \ \wedge \ \det(R) = +1$$

Skalierung

- Kann zum Vergrößern oder Verkleinern verwendet werden:

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

- s_x, s_y, s_z beschreiben Längenänderung in x-, y-, z-Richtung
  → nicht-uniforme (anisotrope) Skalierung
- Uniforme (isotrope) Skalierung: $s_x = s_y = s_z$
- Inverse:

$$S^{-1}(s_x, s_y, s_z) = S\left(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z}\right)$$

- Ein alternative Skalierungs-Matrix:

$$S(s,s,s) = \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cong \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{s} \end{pmatrix}$$

- Aber besser die "normale" Skalierungsmatrix verwenden

Scherung (*Shearing*)

- Verschiebt z.B. die *x*-Koordinate abhängig von der Entfernung zur Ebene *z*=0 (d.h., der *xy*-Ebene), also abhängig von der z-Koord.
- Zum Beispiel: $H_{XZ}(s)$ schert den *x*-Wert gemäß dem *z*-Wert

$$H_{xz}(s)\cdot\mathbf{p} = \begin{pmatrix} 1 & 0 & s \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} p_x + sp_z \\ p_y \\ p_z \end{pmatrix}$$

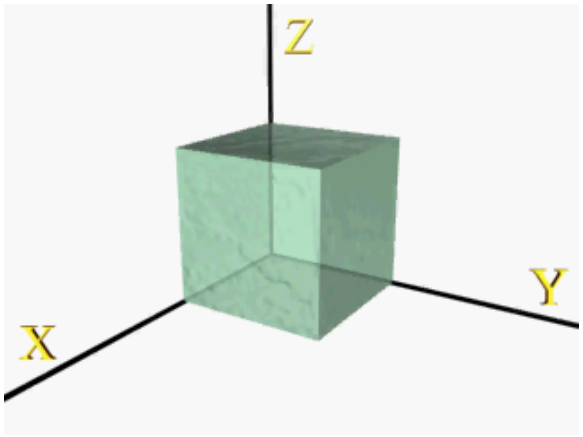
- Inverse:

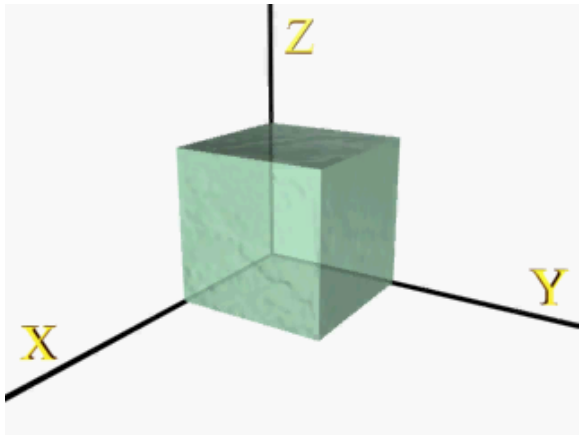
$$H_{xz}^{-1}(s) = H_{xz}(-s)$$

- Achtung: Determinante = 1 → Volumen bleibt erhalten
  - Aber Winkel werden hier nicht erhalten!

Visualisierung

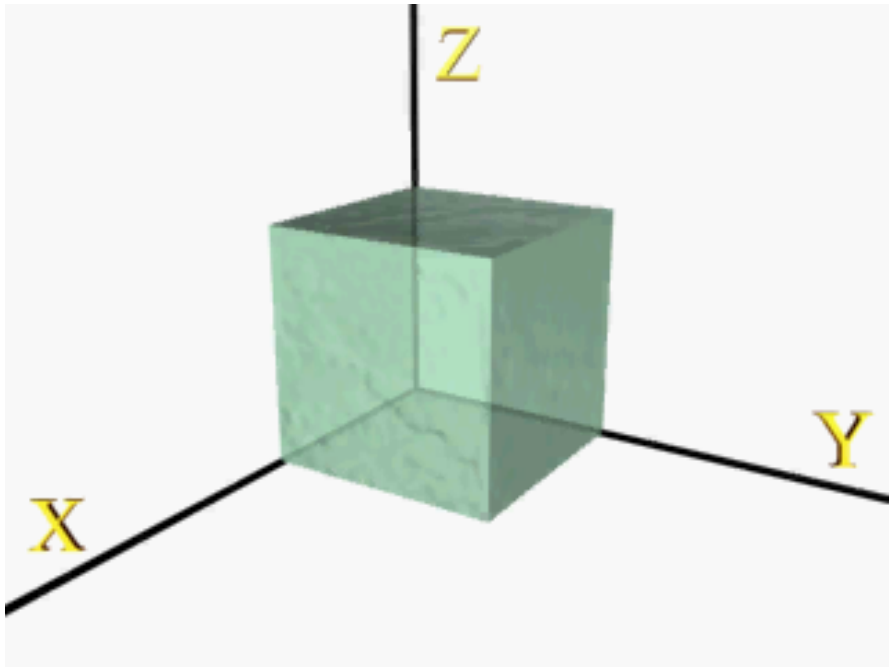
H_{xy} = X mit Y





H_{xz} = X mit Z

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



H_{xz} dann H_{zy}

Spiegelung

- Spiegelung entlang der x-Achse, m.a.W., Spiegelung bzgl. der yz-Ebene:

$$M_x = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

  - Analog die anderen beiden Spiegelungen

- Achtung:

$$\det(M_x) < 0 \quad !$$

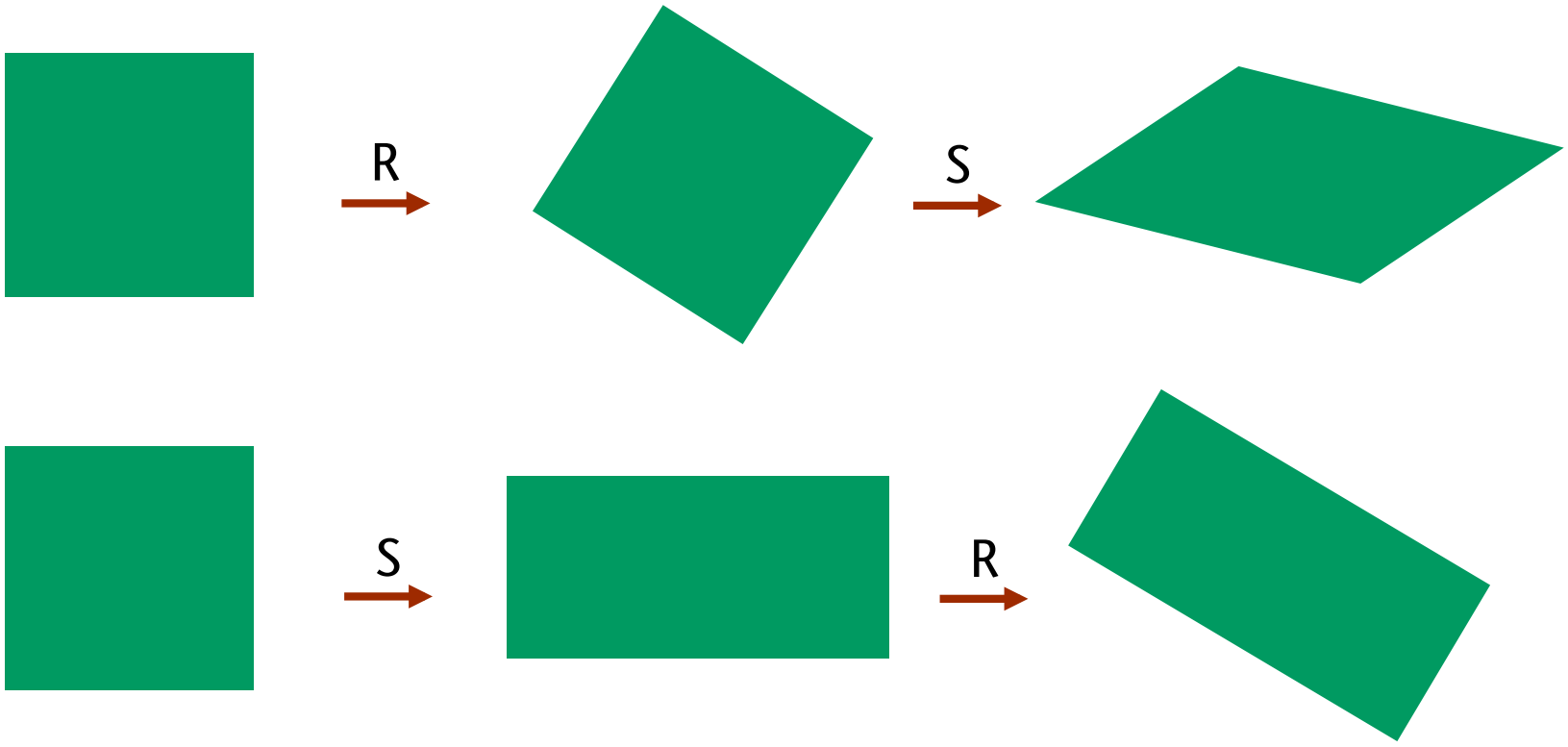
  - Bei allen anderen Transformationen T bisher war

$$\det(T) > 0$$

- Spiegelungen sind in der CG eigtl. immer ausgeschlossen
  - U.a., weil der Umlaufsinn der Polygone umgedreht wird

Verknüpfung / *Concatenation*

- Nützlich zur Steigerung der Effizienz
- Achtung: Multiplikation von Matrizen ist nicht kommutativ → Reihenfolge der Transformation spielt eine Rolle!
- Beispiel:

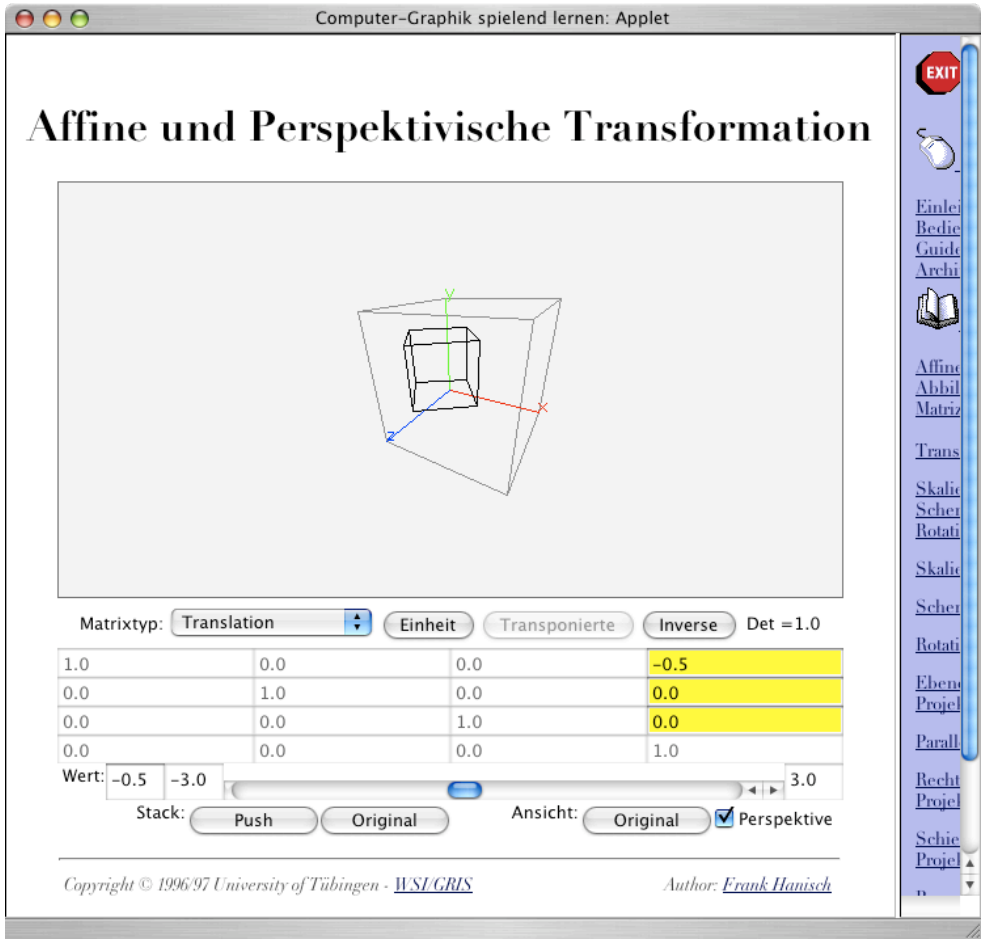


- Reihenfolge in einer Matrixkette:

$$p' = M_n \cdot \ldots \cdot M_2 \cdot M_1 \cdot p$$

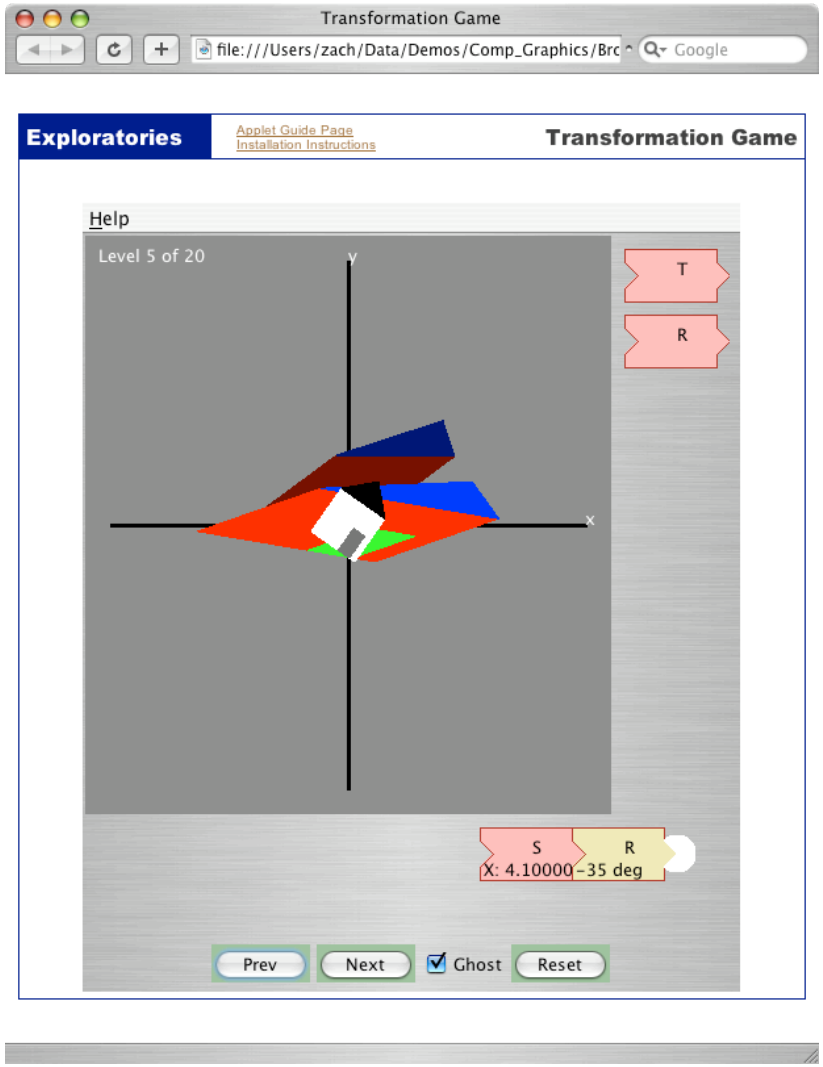
Reihenfolge der Ausführung

Demo zur Konkatenation von Transformationen



Quelle: http://www.gris.uni-tuebingen.de/gris/GDV/java/doc/html/etc/AppletIndex.html

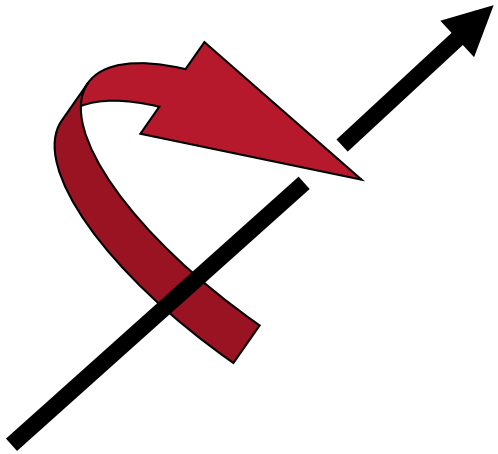
Und noch eine ...



http://www.cs.brown.edu/exploratories → *Transformation Game*

Wieviele Freiheitsgrade haben allg. Rotationen?

- Antwort: 3 DOFs (*degrees of freedom*)
  - 2 DOFs für die Achse + 1 DOF für den Winkel; oder
  - 3 Euler-Winkel

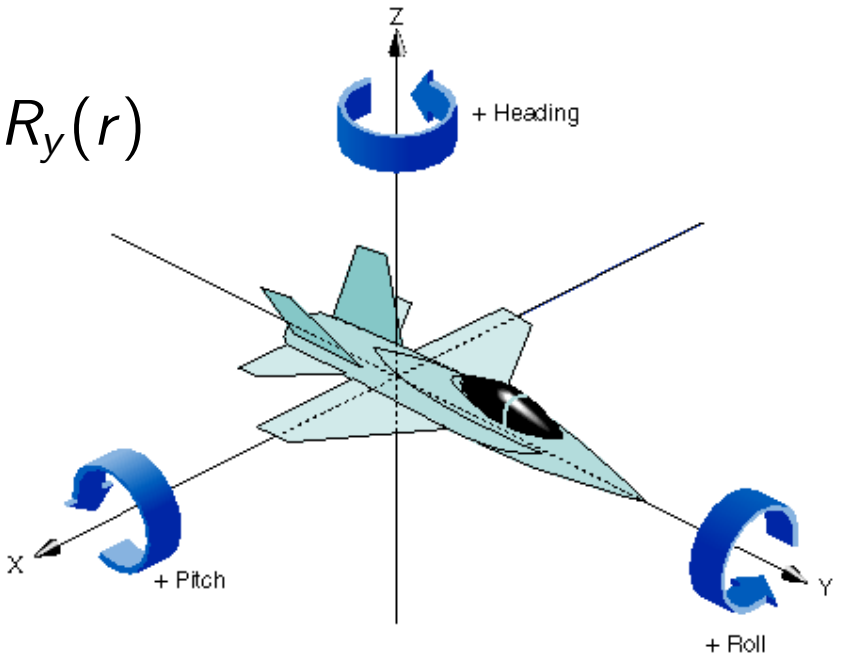


Die Euler-Winkel

- Euler's Rotations-Satz:
  Jede Rotation kann mit Hilfe 3-er Winkel um (fast) beliebige Achsen zusammengesetzt werden.
- Diese Winkel heißen Euler-Winkel und bezeichnen meistens Rotationen um eine der drei Welt-Koordinaten-Achsen
- Häufige Variante im Maschinenbau:

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

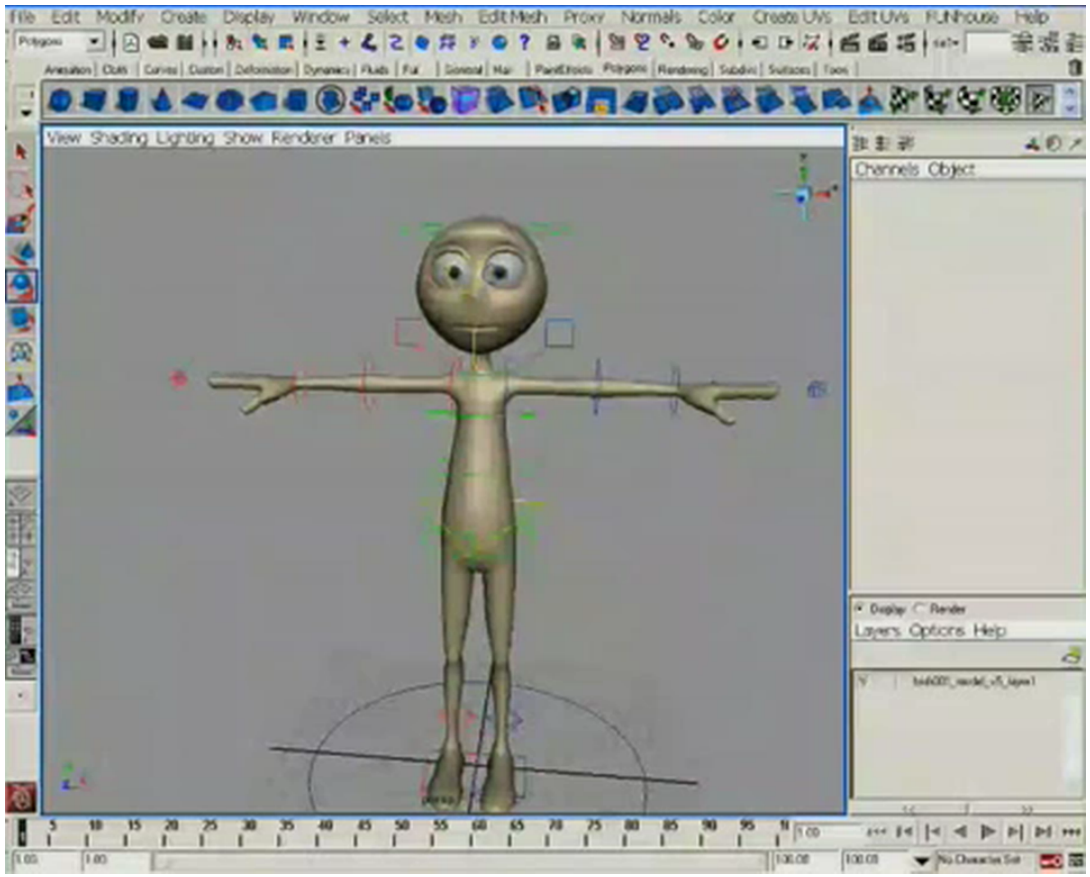
- Bezeichnung oft:
  roll, pitch, yaw (Schiff)
  roll, pitch, heading (Flugzeug)



Nachteile

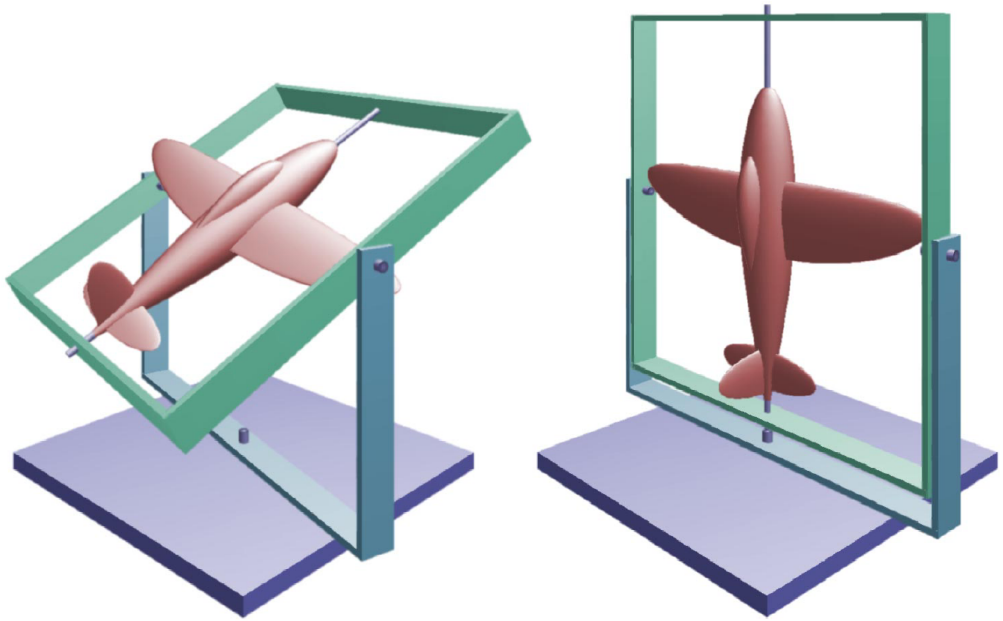
- Erscheint zunächst sehr natürlich (3 DOFs → 3 Winkel)
- Achtung: die Spezifikation einer Rotation mittels Euler-Winkeln macht viele Probleme!!!!!!!!!
  - Reihenfolge der Rotationen ist nicht festgelegt!
    - Oft *"roll, pitch, yaw"* — aber Zuordnung zu den Achsen nicht definiert!
    - Manchmal auch: z, x, z ! Oder ...
  - Gimbal Lock
  - Werte-Bereiche: für einen der Winkel ist der Bereich nur [-90, +90]
    - Das kann auch gar nicht anders sein
  - Rechnen (z.B. interpolieren oder mitteln) ist unmöglich

- Beispiel für die unerwünschten Effekte bei Interpolation von Euler-Winkeln:



Der *Gimbal Lock*

- Tritt immer dann ein, wenn 2 Achsen (fast) gleich sind
- Folge: immer noch 3 Parameter, aber nur 2 Freiheitsgrade!
  - Die Abbildung von Orientierung → Euler-Winkel ist nicht mehr eindeutig
  - Rotation um eine der (lokalen) Flugzeugachsen geht nicht mehr!



Euler Angles in the Real World

- In den Apollo-Raketen wurde ein Kreiselkompass (Trägheitsmessgerät) verwendet
- Kleine Anekdote dazu:

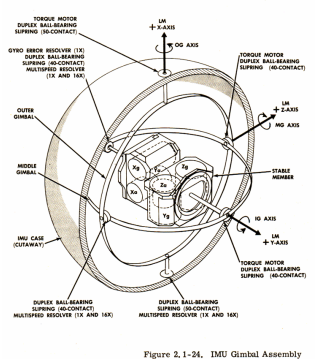


Figure 2.1-24. IMU Gimbal Assembly

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.

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

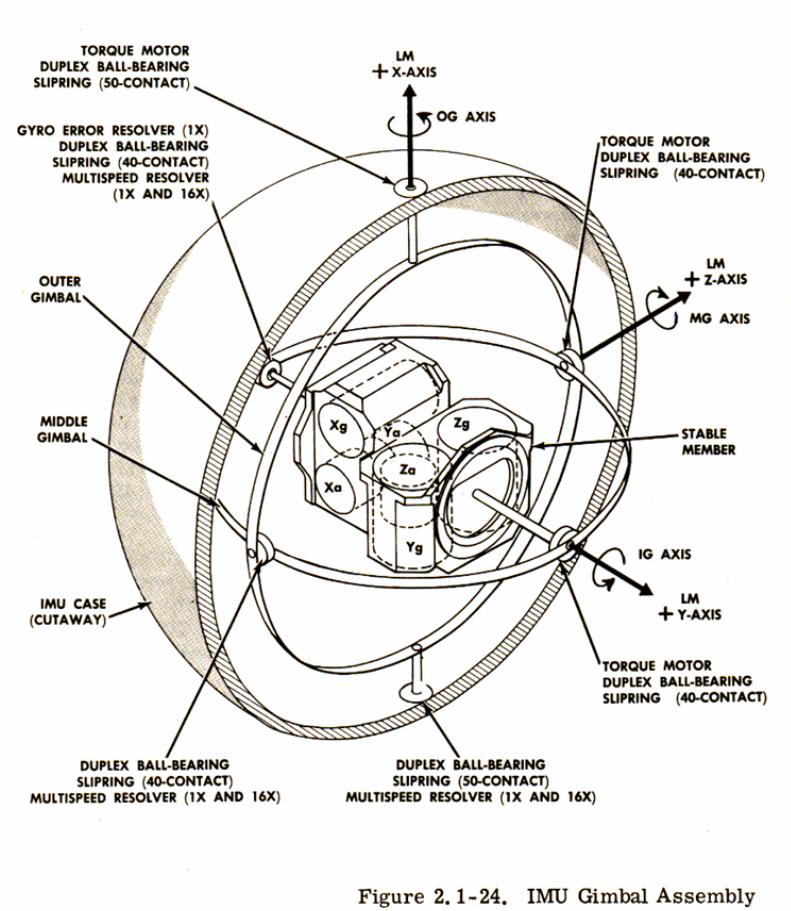
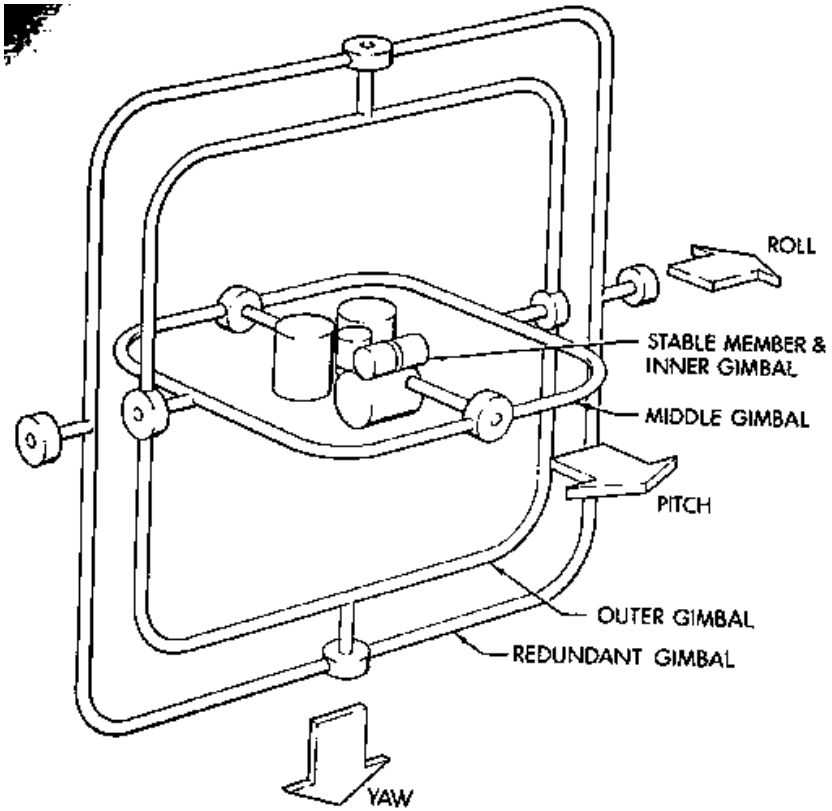
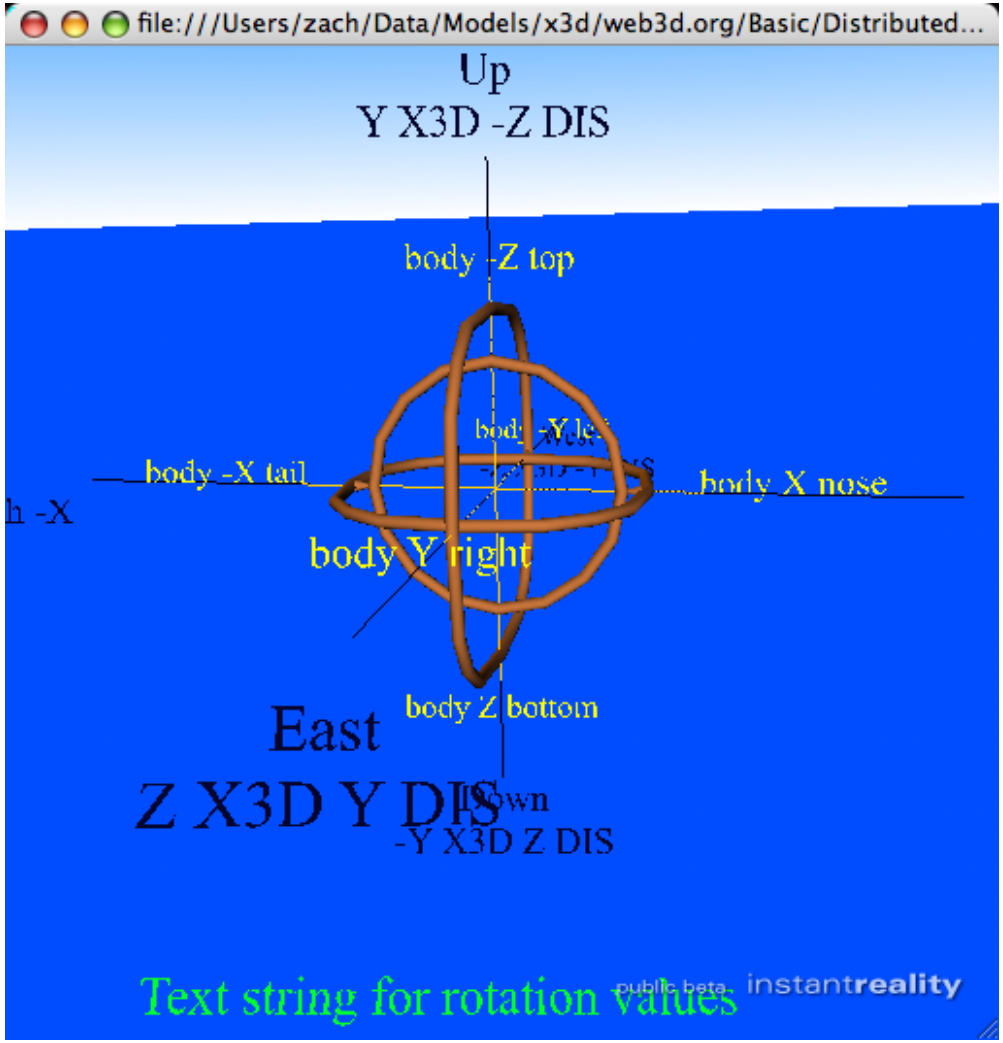


Figure 2.1-24. IMU Gimbal Assembly



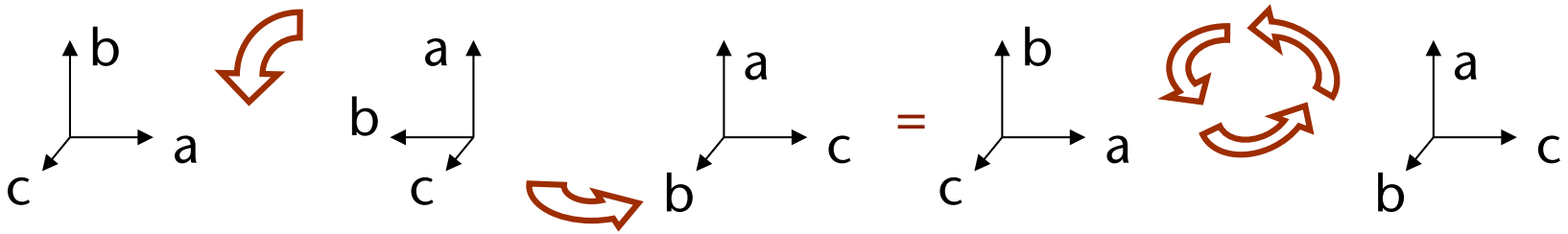
Quelle: http://www.hq.nasa.gov/office/pao/History/alsj/gimbals.html

Interaktive Demo (VRML)

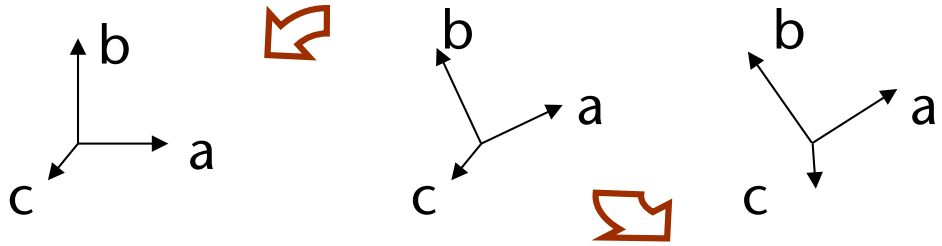


Gimbals.wrl

- Weiteres Problem: Interpolation zwischen zwei Rotationen (Orientierungen) mittels Euler-Winkel klappt nicht
- Beispiel:
  - Rotation von 90° um Z, dann 90° um Y = 120° um (1, 1, 1)



  - Aber: Rotation von 30° um Z, dann 30° um Y ≠ 40° um (1, 1, 1) !



Intermezzo

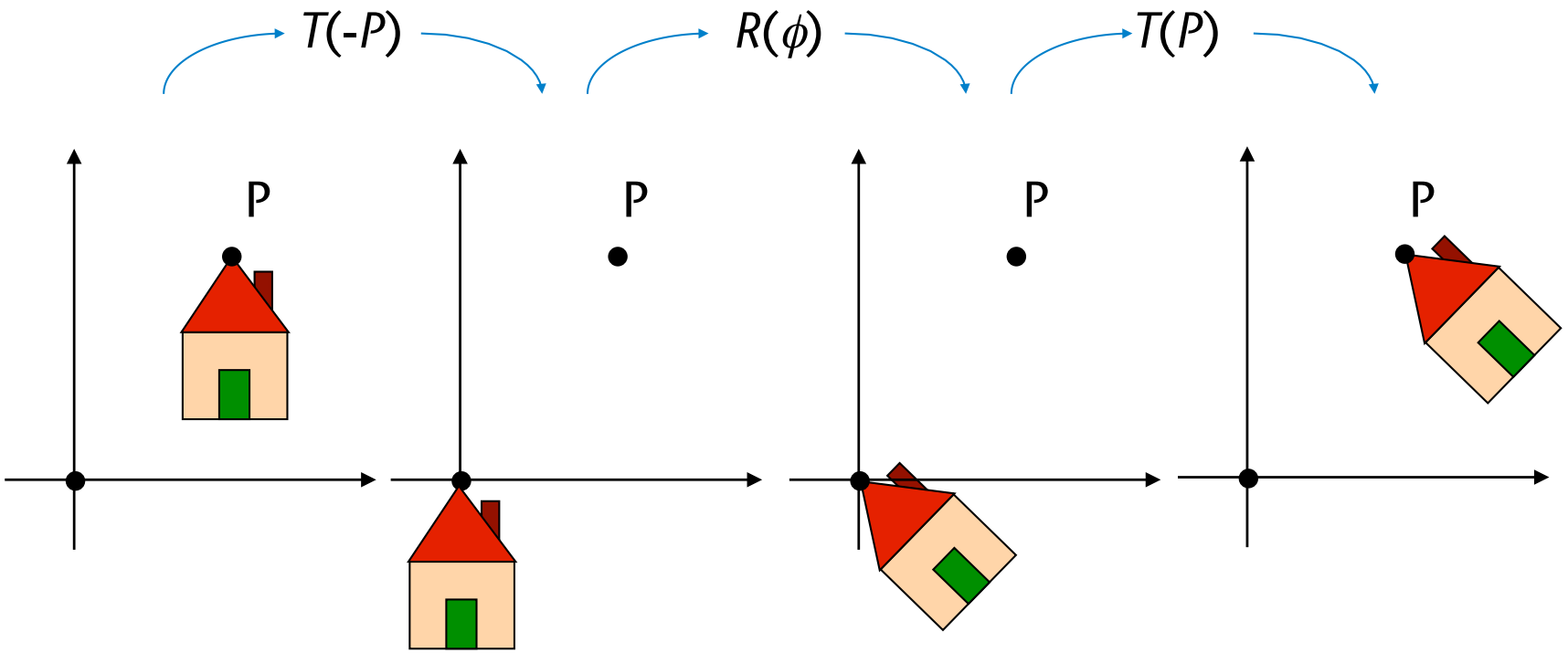
- Ein Film, in dem viele Interpolationen von Rotationen / Orientierungen vorkommen

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

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

Rotation um einen beliebigen Punkt in 2D

- Gesucht: Rotation mit Winkel ϕ um P



$$M = T_P R_\theta T_{-P}$$

Rotation um eine beliebige Achse in 3D

- Gesucht: Rotationsmatrix zu gegebener Achse **r** (oBdA geht **r** durch den Ursprung)

1. Erzeuge neue Basis (**r**, **s**, **t**) (bestimme **s** und **t**, orthogonal zu **r**; siehe Kapitel "Kurze Wiederholung in Geometrie")
2. Transformiere alles, so daß die Basis (**r**, **s**, **t**) in die Standardbasis (**x**, **y**, **z**) übergeht
3. Rotiere mit Winkel ϕ um **x**-Achse
4. Transformiere zurück in die ursprüngliche Basis

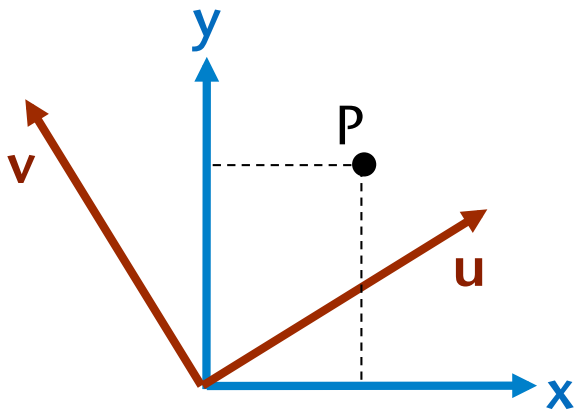
- Zusammen:

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

Zusammenhang zwischen Rotation und Basiswechsel

- Man kann eine Rotationsmatrix direkt aus den 3 Koordinatenachsen eines (neuen) Koordinatensystems konstruieren
- Gegeben: Einheitsvektoren **u, v, w**
- Setze:

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

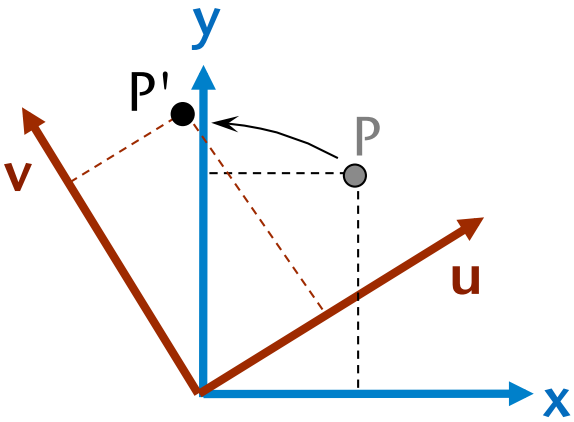


- Damit ist

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

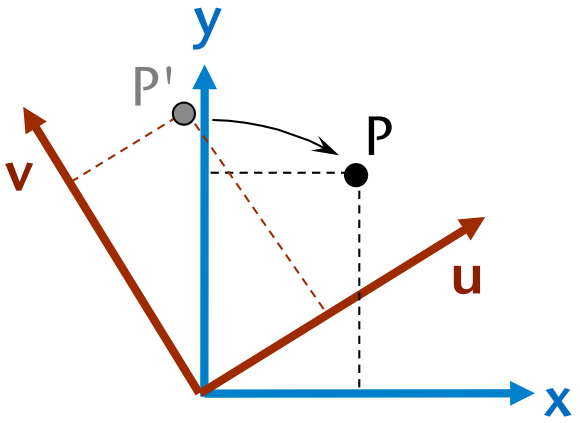
$$R \cdot R^T = I$$

$$\det(R) = 1$$



- Also: R ist Rotation
  - Und zwar von xyz-Koordinaten → uvw-Koordinaten

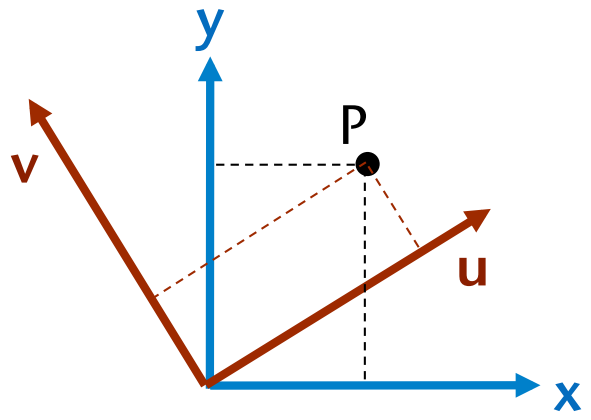
- Was bedeutet die Matrix R^{-1} ?
  - Rotation von uvw → xyz (klar)



- Aber außerdem:

$$\mathbf{p}' = R^{-1} \cdot P = R^{\mathsf{T}} \cdot \mathbf{p} = \begin{pmatrix} - & \mathbf{u} & - \\ - & \mathbf{v} & - \\ - & \mathbf{w} & - \end{pmatrix} \mathbf{p} = \begin{pmatrix} \mathbf{u} \cdot \mathbf{p} \\ \mathbf{v} \cdot \mathbf{p} \\ \mathbf{w} \cdot \mathbf{p} \end{pmatrix}$$

  - $\mathbf{p}'$ stellt den selben Punkt P im Raum dar wie $\mathbf{p}$!
    $\mathbf{p}'$ stellt ihn in uvw-Koordinaten dar,
    $\mathbf{p}$ stellt ihn in xyz-Koordinaten!



Zerlegung einer Rotationsmatrix

- 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:

$$\text{spur}(R) = \text{spur}\left(URU^{-1}\right) = \text{spur}\left(R_x(\theta)\right) = 1 + 2\cos\theta$$

Spezielle Eigenschaft der Spur

2. Aufgabe: Rotationsachse **r** bestimmen

- Lösung: **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

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

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

- Falls $\det(A) = 0$, dann ist

$$AA^{\#} = 0 \cdot I = 0$$

- Also gilt für jeden Spaltenvektor **v** aus $A^{\#}$, daß

$$A \cdot \mathbf{v} = 0$$

- Gesucht ist der Eigenvektor **r** zum Eigenwert 1 von R, also ein **r**, so daß

$$(R - I) \cdot \mathbf{r} = 0$$

- Das sind gerade die Spalten von

$$(R - I)^{\#}$$

  - Alle Spalten sind Vielfache einer der Spalten

- Bemerkung: das funktioniert auch für größere Matrizen, ist aber nicht mehr effizient

Darstellung der Rotation als Achse & Winkel

- Damit haben wir folgenden wichtigen Satz bewiesen ...

- Satz (Euler):
  Jede beliebige Rotation im Raum lässt sich als Rotation um eine bestimmte Achse mit einem bestimmten Winkel darstellen.

- Anmerkung: in der Robotik wird gerne die Variante verwendet, wo

$$\text{Winkel} = \| \text{Rotationsachse} \|$$

  (damit benötigt man also wieder nur einen 3D-Vektor)

Complex Numbers as Rotations

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

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

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

Vorbemerkung

- Wir können reelle Zahlen einfach 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"?
  - Z.B. eine "dreidimensionale" Verallgemeinerung von $\mathbb{C}$? → Nein!
  - Aber: im 4D klappt es wieder ... (fast)

Quaternionen

[Hamilton, 1843]

- Erweiterung der komplexen Zahlen (geht leider nicht kommutativ):

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

- 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:

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k} \qquad \mathbf{jk} = -\mathbf{kj} = \mathbf{i} \qquad \mathbf{jk} = -\mathbf{kj} = \mathbf{i}$$

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:

$$\begin{aligned} q_1 \cdot q_2 &= (w_1 + a_1\mathbf{i} + b_1\mathbf{j} + c_1\mathbf{k}) \cdot (w_2 + a_2\mathbf{i} + b_2\mathbf{j} + c_2\mathbf{k}) \\ &= (w_1 w_2 - a_1 a_2 - b_1 b_2 - c_1 c_2) + \\ &\quad (w_1 a_2 + w_2 a_1 + b_1 c_2 - c_1 b_2)\,\mathbf{i} + \\ &\quad (\quad \ldots \quad \ldots \quad)\,\mathbf{j} + \\ &\quad (\quad \ldots \quad \ldots \quad)\,\mathbf{k} \end{aligned}$$

- 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.

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

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

- Beobachtung: $|q| = 1$

- Zurückrechnen:

  $q = (w, a, b, c)$ ist gegeben, mit $|q| = 1$

  Dann ist

$$\varphi = 2\arccos(w)$$

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

- Satz:

  Jedes Einheitsquaternion kann man in der Form

  $$\left(\cos \frac{\varphi}{2}, \sin \frac{\varphi}{2} \mathbf{r} \right)$$

  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

  $$\mathbf{v} \mapsto q \cdot \mathbf{v} \cdot q^* = \mathbf{v}'$$

  eine (rechtshändige) Rotation von **v**, wobei Winkel und Achse durch q bestimmt sind.

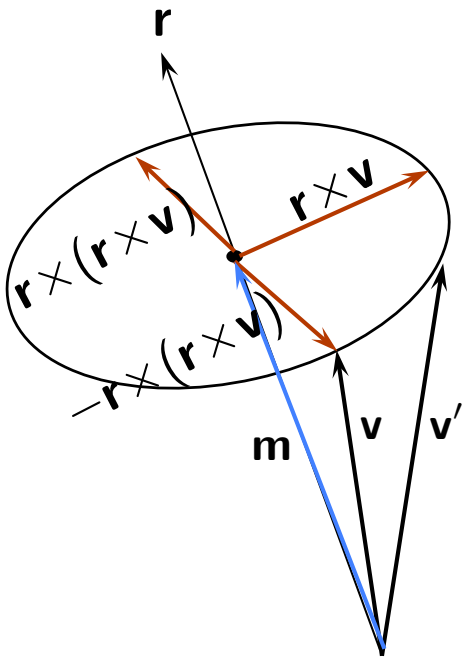
- Beweisskizze:

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

$$= \ldots \qquad (*)$$

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

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



*) Zwischendurch benötigt man diese trigonometrischen Identitäten:

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

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

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

Lineare Interpolation von Quaternionen

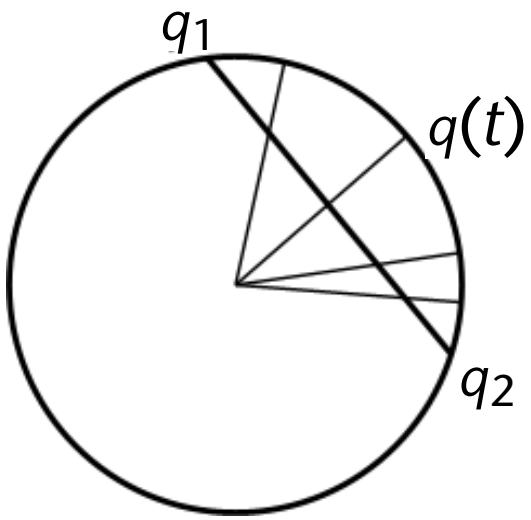
- Gegeben: zwei Orientierungen q_1 , q_2
  (Orienterung = Rotation aus der Null-Lage)
- Aufgabe: dazwischen interpolieren
- Einfachste Lösung:

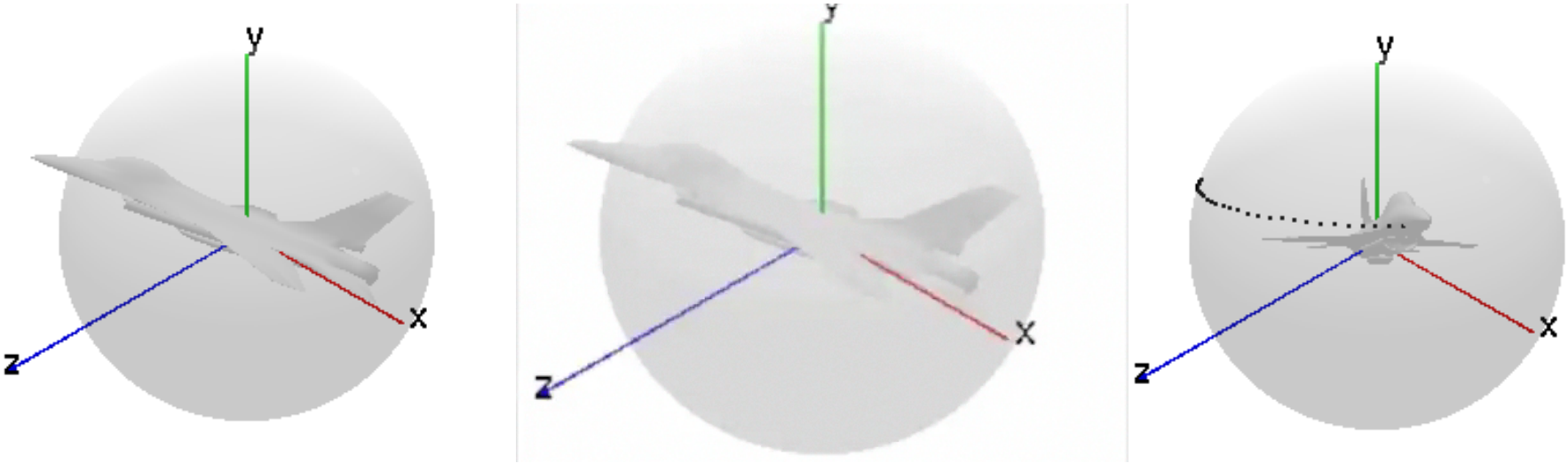
$$q(t) = \text{lerp}(t; q_1, q_2) = (1 - t)q_1 + tq_2$$

  - Wichtig: $q(t)$ hinterher immer normieren!

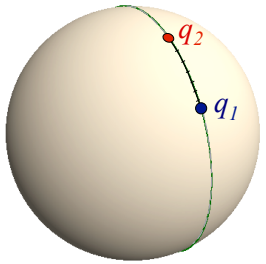
- Vorteil: Kein Gimbal Lock!

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





Sphärische lineare Interpolation

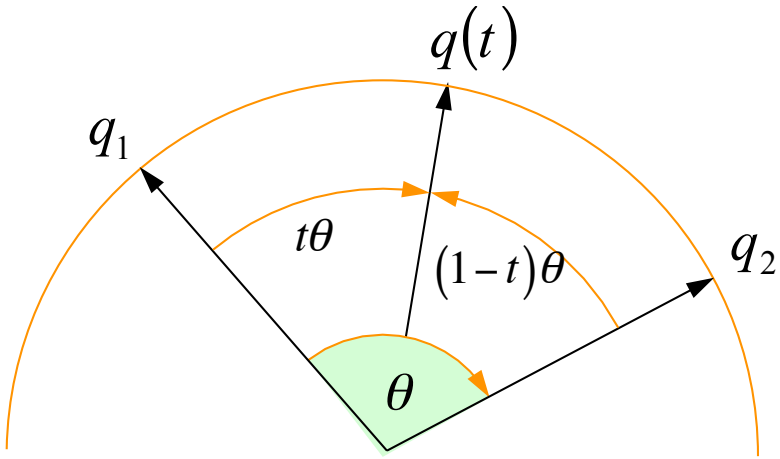


- Besser ist die sphärische lineare Interpolation "*slerp*":

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

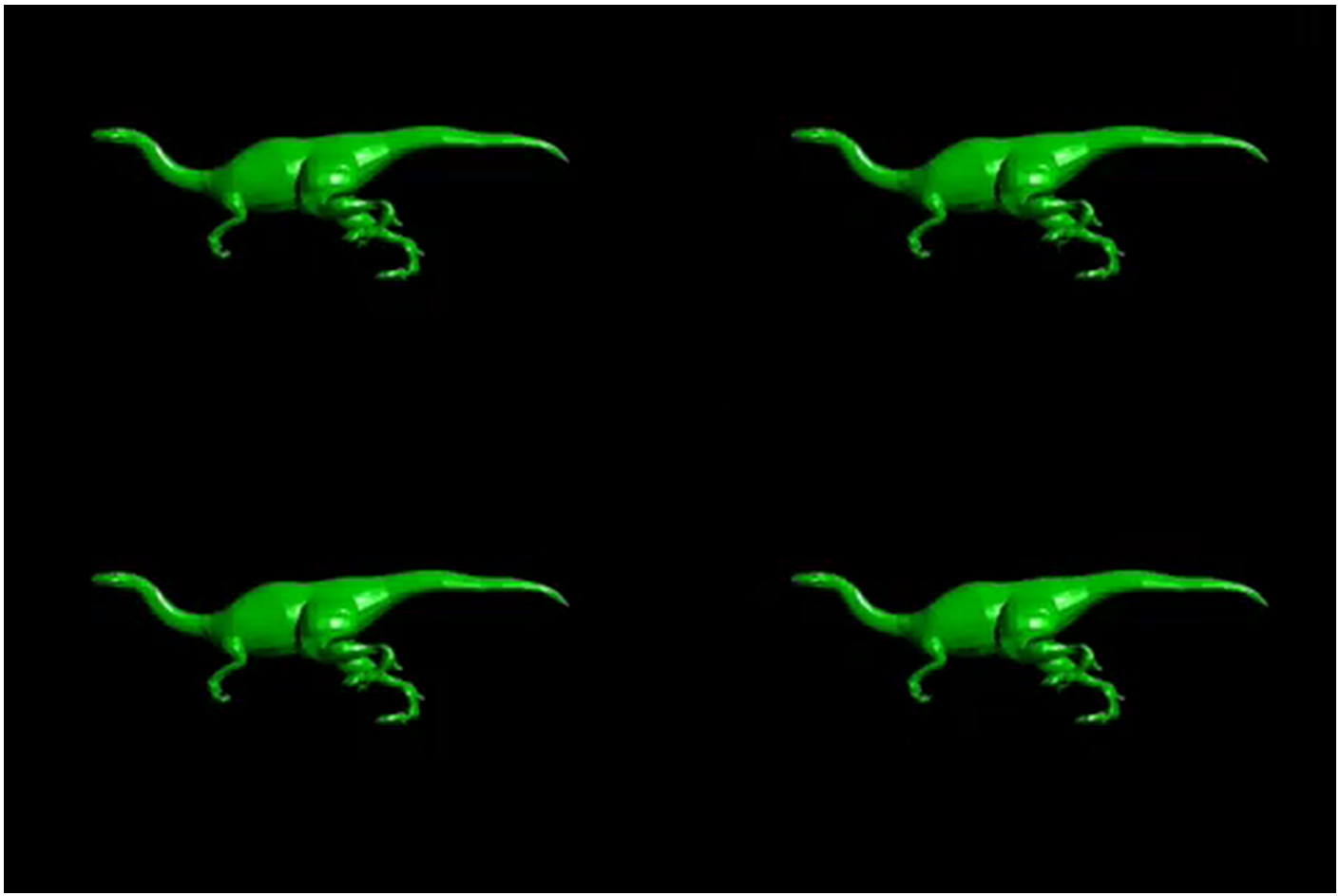
mit $\cos\theta = q_1 \odot q_2$ Skalarprodukt der *Vektoren* q_1 und q_2

- Hier gilt:



Vergleich der verschiedenen Interpolationsarten

Interpolation of Euler angles



Naïve interpolation of matrices

Slerp of quaternions

Lerp of quaternions (with normalization)

Gianluca Vatinno, Trinity College Dublin

Umwandlung Quaternion → Rot.matrix

- Zunächst das Analogon im 2D:
  wenn a,b, mit $a^2 + b^2 = 1$, gegeben sind, dann ist

$$M = \begin{pmatrix} a^2 - b^2 & -2ab \\ 2ab & a^2 - b^2 \end{pmatrix}$$

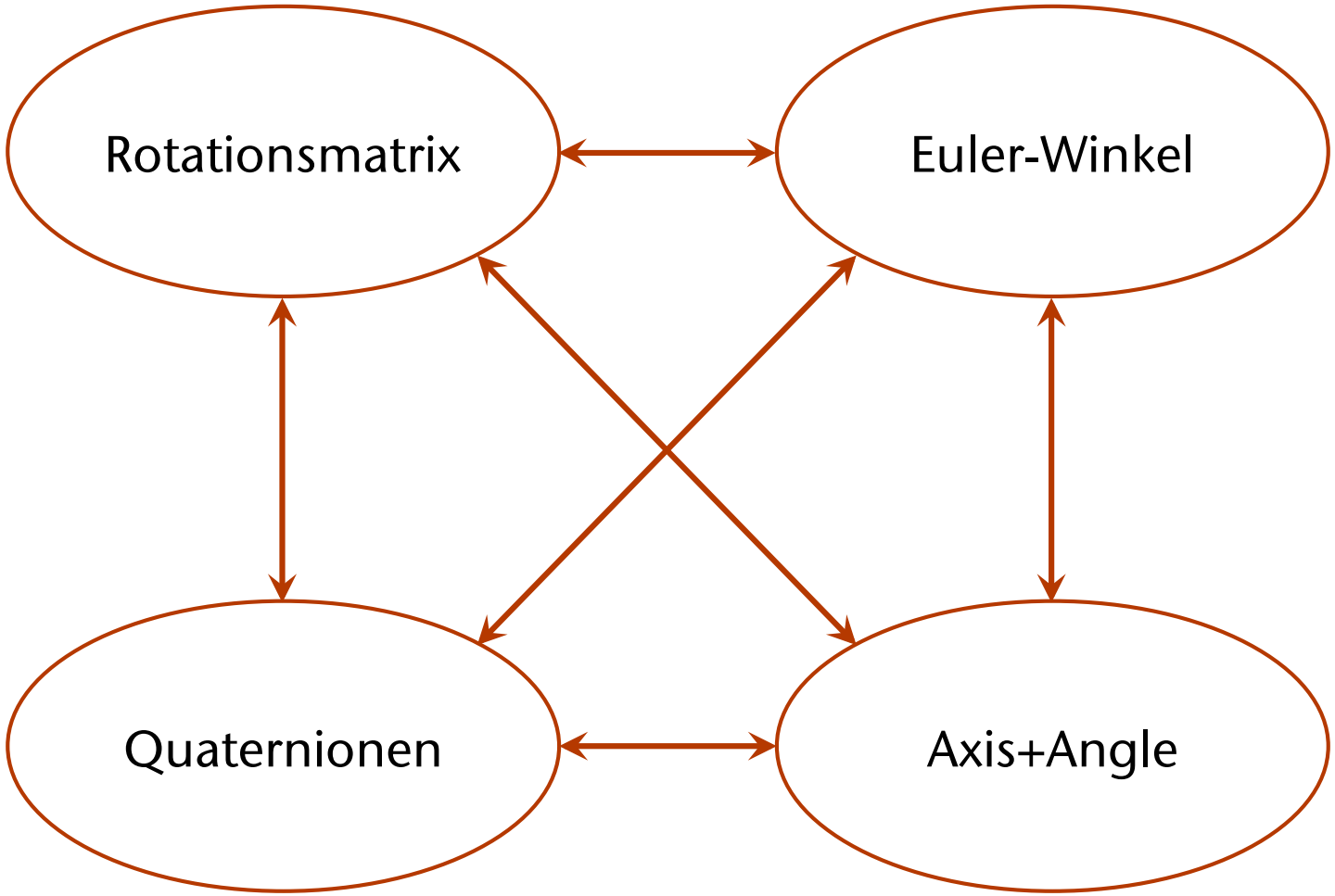
  eine Rotationsmatrix.

- So ähnlich kann man eine Rotationsmatrix im 3D aus einem Quaternion $q = w + ai + bj + ck$, mit $|q| = 1$, bilden:

$$R(q) = \begin{pmatrix} w^2 + a^2 - b^2 - c^2 & 2ab - 2wc & 2ac - 2wb \\ -2ab + 2wc & w^2 - a^2 + b^2 - c^2 & 2bc - 2wa \\ -2ac + 2wb & -2bc + 2wa & w^2 - b^2 - c^2 + d^2 \end{pmatrix}$$

- Überprüfung:
  - Spalte i × Spalte $j = 0 \Leftrightarrow i \neq j$
  - Spalte i × Spalte $i = (w^2 + a^2 + b^2 + c^2)^2$

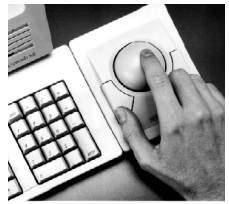
Alle Darstellungen von Rotation lassen sich ineinander umrechnen



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

Der virtuelle Trackball

- Wie gibt man Orientierungen mit der Maus ein?
- Idee:
  - Lege Kugel um das Objekt / die Szene
  - Kugel kann um ihr Zentrum rotieren
  - Maus pickt Punkt auf Oberfläche, den man zieht



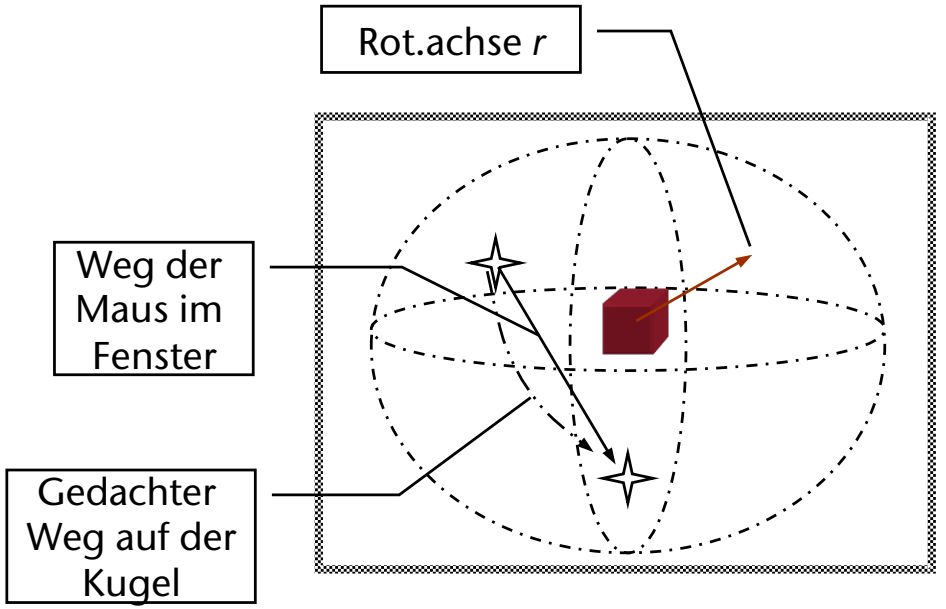
- Geg.: 2D Punkte Startpunkt = (x_1,y_1), Endpunkt = (x_2,y_2)
- Ges.: Rotationsachse r
- Berechnung:
  1. Bestimme 3D Punkte

$$\mathbf{p}_i = (x_i, y_i, z_i)$$

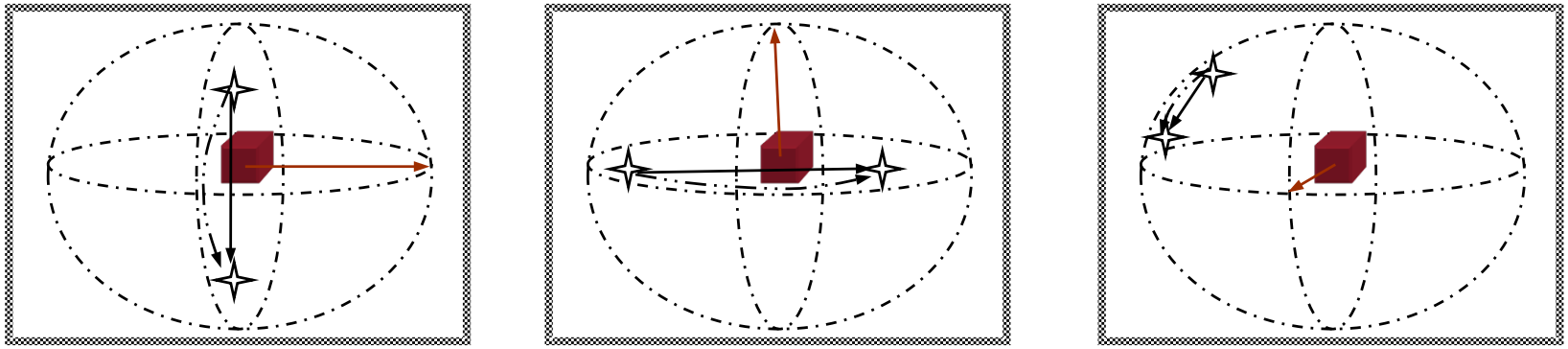
$$z_i = 1 - \sqrt{x_i^2 + y_i^2}$$

  2. Rotationsache

$$\mathbf{r} = \mathbf{p}_1 \times \mathbf{p}_2$$



- Man kann um alle Achsen (bis auf eine) direkt rotieren:



- Verbesserungen:
  - "Spinning trackball" vermeidet häufiges Nachfassen
  - "Locking" für exaktes Rotieren um eine Koord.achse
  - Was macht man, wenn (x,y) die Ellipse verlassen?
    - Nichts(?) → z wird negativ → dann noch x,y am Kreis nach innen spiegeln → p liegt auf der Rückseite der Kugel

Lineare und affine Abbildung

- Folge aus Linearität → Geraden werden auf Geraden abgebildet
- Lineare Transformationen (z.B. Rotation, Skalierung und Scherung) können durch eine 3x3 Matrix dargestellt werden:

$$x' = A \cdot (\alpha x + \beta y) = \alpha A \cdot x + \beta A \cdot y$$

- Merke die Konvention:

"Matrix mal Vektor"

- Problem: affine Transformationen (z.B. Translation), können **nicht** alle als 3x3 Matrix dargestellt werden:

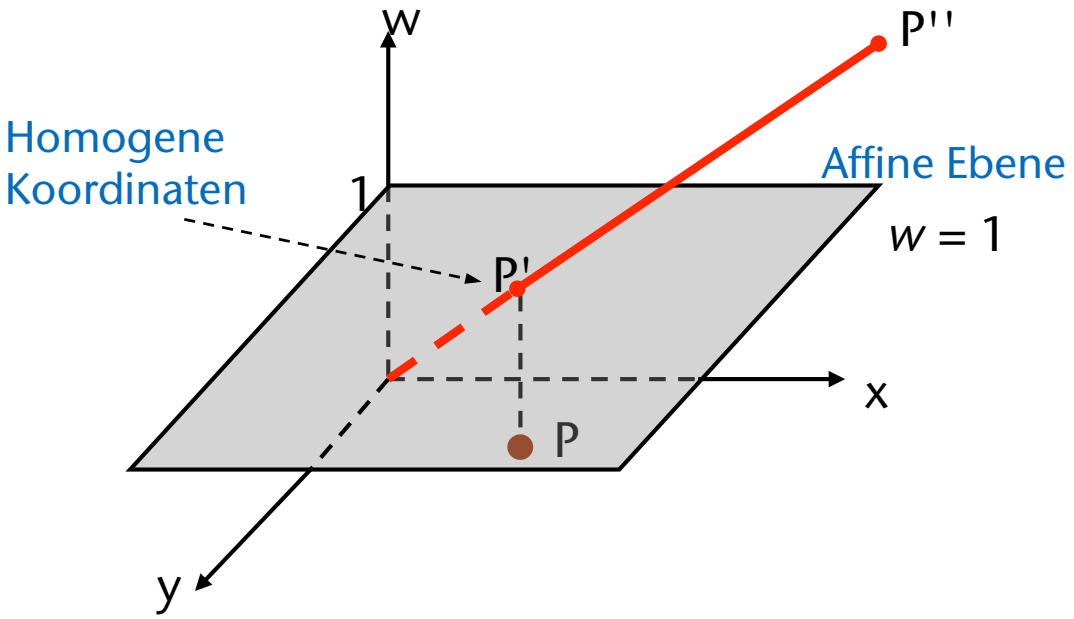
$$X' = A \cdot x + t$$

Lösung: Homogene Koordinaten im 3D

- Homogene Darstellung ist nützlich für Transformationen von Punkten *und* Vektoren
- "Trick": erweitere 3D Punkte und Vektoren zu 4D Punkten und Vektoren
- Homogener Punkt $P = (p_x, p_y, p_z, p_w) \quad p_w = 1$
- Homogener Vektor $\mathbf{p} = (p_x, p_y, p_z, p_w) \quad p_w = 0$

Veranschaulichung im 2D

- Erweitere Punkt P = (*x*,*y*) zu P' = (*x*,*y*,1)
- Assoziiere Linie *w*·(*x*,*y*,1) = (*wx*, *wy*, *w*) mit P'



- M.a.W.: ein 3D-Vektor (*x*, *y*, *w*) beschreibt ...
  - ... den 2D-Punkt (*x*/*w*, *y*/*w*) für *w* ≠ 0
  - ... den 2D-Vektor (*x*, *y*) für *w* = 0

Homogenisierung im 3D

- Der homogene Punkt

$$P = (x, y, z, w) \quad w \neq 0$$

beschreibt den Punkt an der Stelle

$$P = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$$

Addition von Punkten und Vektoren in homogenen Koord.

- Punkt + Vektor = Punkt

$$\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} + \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{pmatrix}$$

- Vektor + Vektor = Vektor

$$\begin{pmatrix} u_x \\ u_y \\ u_z \\ 0 \end{pmatrix} + \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} u_x + v_x \\ u_y + v_y \\ u_z + v_z \\ 0 \end{pmatrix}$$

- Punkt – Punkt = Vektor

$$\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} - \begin{pmatrix} q_x \\ q_y \\ q_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x - q_x \\ p_y - q_y \\ p_z - q_z \\ 0 \end{pmatrix}$$

Homogene Matrizen für Transformationen in 3D

- Matrix

$$M = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$$

- Homogene Form:

$$M_{4x4} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Lineare Abb. (Matrix-Vektor-Multiplikation)

- 3x3-Form:

$$M \cdot \mathbf{v} = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

- Homogene Form:

$$M_{4x4} \cdot \mathbf{v}' = \begin{pmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix}$$

Affine Abbildungen im 3D

- 3x3-Form:

$$M \cdot \mathbf{p} + t = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

- Homogene Form:

$$M_{4x4} \cdot \mathbf{p}_4 = \begin{pmatrix} m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

➢ In homogenen Koordinaten lassen sich sogar affine Abbildungen als einfache Matrix-Vektor-Multiplikation darstellen!

Translation

- Translation eines Punktes:

$$T_t \cdot P = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix}$$

- Translation eines Vektors:

$$T_t \cdot \mathbf{v} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 0 \end{pmatrix}$$

- Inverse:

$$(T_t)^{-1} = T_{-t}$$

Die elem. linearen Transformationen in homogenen Koordinaten

- Rotation:

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

- Skalierung:

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

- Scherung:

$$H_{xz}(s)\cdot\mathbf{p} = \begin{pmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + sp_z \\ p_y \\ p_z \\ 1 \end{pmatrix}$$