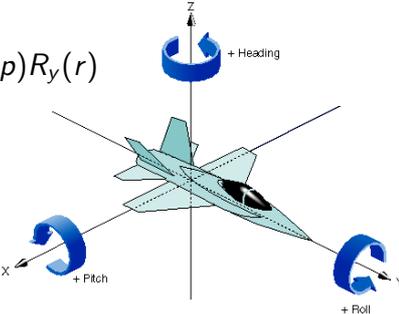


Euler-Winkel

- Euler's Rotations-Satz:
Jede Rotation kann man 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: 1. x, 2. y, 3. z

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

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



G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 29

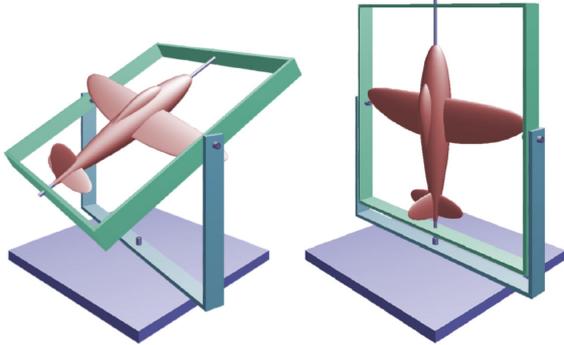
Nachteile

- 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) ist unmöglich

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 30

Der Gimbal Lock

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



G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 31

Euler Angles in the Real World

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

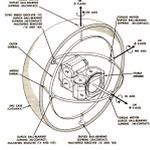


Figure 1.10. IMU (Inertial Measurement Unit)

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.

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 32

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

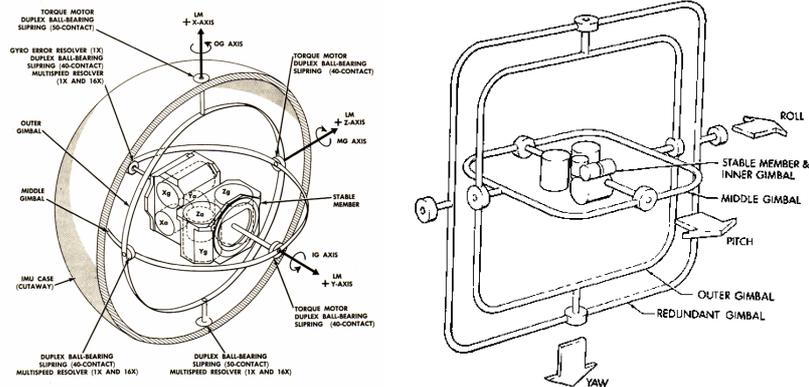
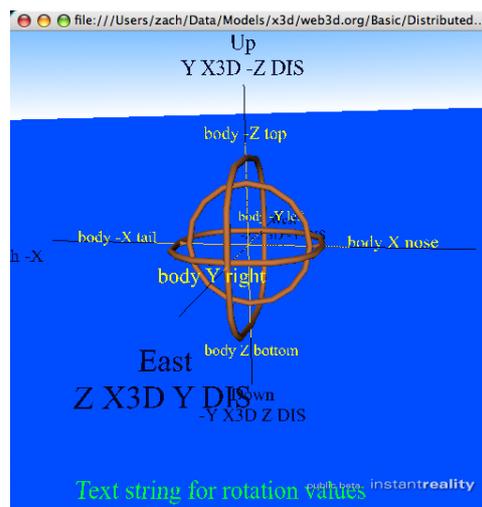


Figure 2.1-24. IMU Gimbal Assembly

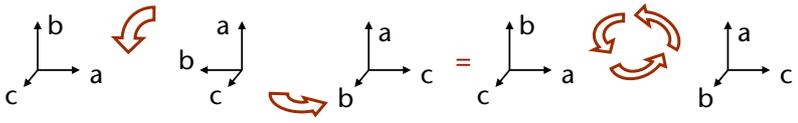
Quelle: <http://www.hq.nasa.gov/office/pao/History/alsi/gimbals.html>

Interaktive Demo (VRML)

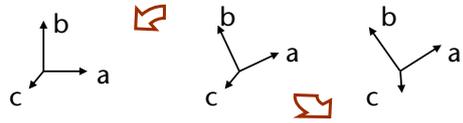


[Gimbals.wrl](#)

- 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 \neq 40° um (1, 1, 1) !



G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 35

Intermezzo

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



G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 36

Rotation um einen beliebigen Punkt in 2D

- Rotation mit Winkel ϕ um P

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

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 37

Rotation um eine beliebige Achse in 3D

- Man möchte mit θ um die Gerade l rotieren

- Gesucht: eine Matrix R , 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

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 38

■ Grundidee:

1. Verschiebe einen Punkt der Geraden 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 θ
5. Invertierte Rotation um die Koordinatenachse aus Schritt 3
6. Invertierte Rotation um die Koordinatenachse aus Schritt 2
7. Verschieben, so daß l wieder in Ausgangsposition (Schritt 1)

Transformationen 39

■ Schritt 1

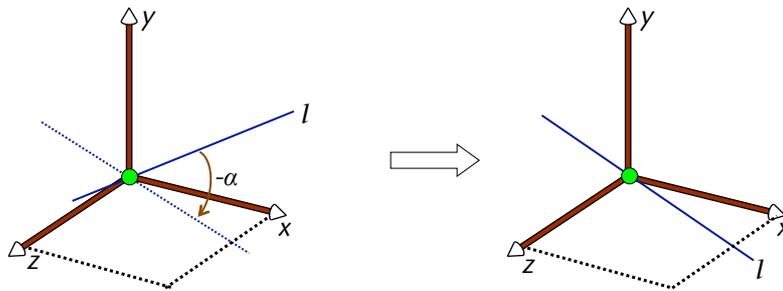
■ Verschiebe Gerade, so daß ein Punkt im Ursprung liegt:

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

Transformationen 40

Schritt 2

- Rotieren, so daß l in einer Koordinatenebene liegt.
- Hier rotiere mit $-\alpha$ um die z -Achse, so daß l in der xz -Ebene liegt

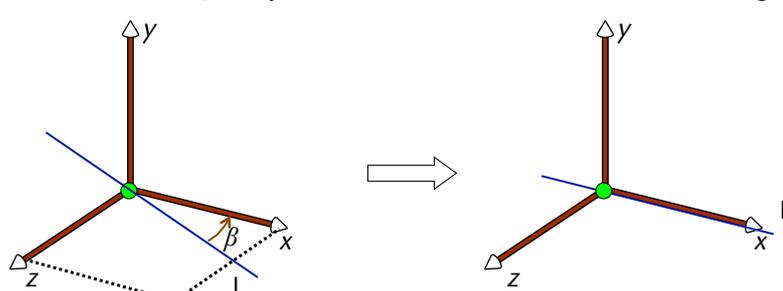


$$R_z(-\alpha) = \begin{pmatrix} \cos(-\alpha) & -\sin(-\alpha) & 0 & 0 \\ \sin(-\alpha) & \cos(-\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 41

Schritt 3

- Rotieren, so daß l auf einer Koordinatenachse liegt
- Hier: rotiere mit β um y -Achse damit Gerade auf der x -Achse liegt

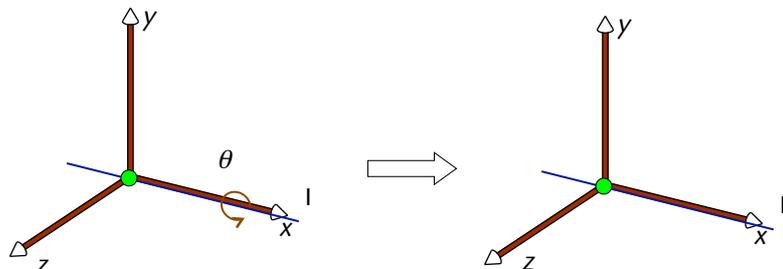


$$R_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 42

Schritt 4

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

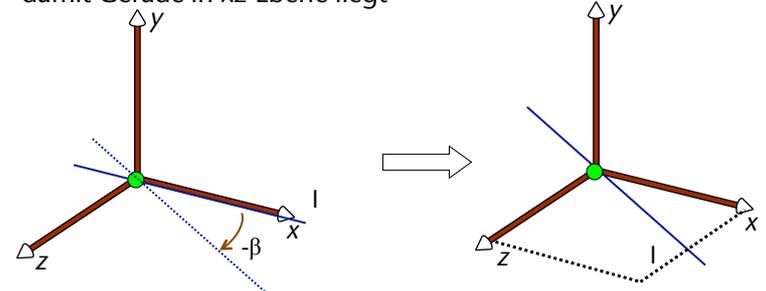


$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 43

Schritt 5

- Durchführen der invertierten Rotation um I zurück in die Koordinatenebene zu verschieben: rotiere mit $-\beta$ um die y-Achse damit Gerade in xz-Ebene liegt



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

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 44

Schritt 6

- Durchführen der invertierten Rotation, um l aus der Koordinatenebene zu verschieben: rotiere mit α um z-Achse

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

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 45

Schritt 7

- Invertiere die Ausgangstranslation, um den Punkt aus dem Ursprung an seinen Ausgangspunkt zu verschieben

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

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 46

Zusammenfassung

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

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

- Hat man diese Matrix, so wendet man diese auf jeden Punkt des Objektes an, was den Effekt der Rotation dieses Objektes um die beliebige Achse hat
 - Das überläßt man natürlich OpenGL

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 47

Alternative

- Rotationsmatrix zu geg. Achse \mathbf{r} auf einen Schlag aufstellen:

- Erzeuge neue Basis $(\mathbf{r}, \mathbf{s}, \mathbf{t})$ (bestimme \mathbf{s} und \mathbf{t} , orthogonal zu \mathbf{r} ; siehe "Math Recap")
- Transformiere alles, so daß die Basis $(\mathbf{r}, \mathbf{s}, \mathbf{t})$ in die Standardbasis $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ übergeht
- Rotiere mit Winkel ϕ um \mathbf{x} -Achse
- Transformiere zurück in die ursprüngliche Basis

- Zusammen:

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

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 48

Charakterisierung von (reinen) Rotationsmatrizen

- Erinnerung:
 - R ist orthogonal $\Leftrightarrow RR^T = I$
 - R orthogonal $\Rightarrow \det(R) = \pm 1$
- Achtung: dabei können noch Spiegelungen enthalten sein!
- R ist eine **ordentliche Rotation** \Leftrightarrow

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

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 53

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:

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

↑
Spezielle Eigenschaft der Spur

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 55

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

$$a_{ij}^\# = (-1)^{i+j} \det \begin{pmatrix} a_{11} & \cdots & a_{1j} & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ a_{j1} & \cdots & a_{jj} & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$
 - Es gilt: $AA^\# = \det(A)I$

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 56

- 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 ist der 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

$$(R - I)^\#$$
 - Alle Spalten sind Vielfache einer der Spalten
- Bemerkung: das funktioniert auch für größere Matrizen, ist aber nicht mehr effizient

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 57

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

$$\text{Winkel} = \|\text{Vektor}\|$$
 (damit benötigt man also wieder nur einen 3D-Vektor)

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 58

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

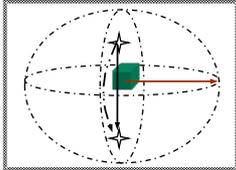
$$\mathbf{p}_i = (x_i, y_i, z_i)$$

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

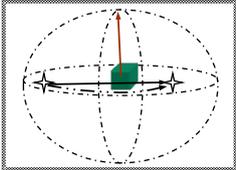
$$\mathbf{r} = \mathbf{p}_1 \times \mathbf{p}_2$$

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 84

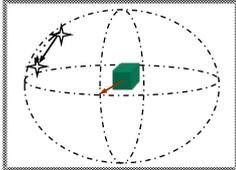
- Man kann um alle Achsen (bis auf eine) direkt rotieren:



X



Y



Z

- 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

G. Zachmann Computer-Graphik 1 – WS 11/12
Transformationen 85

Anatomie einer Matrix

- Erst Rotation, dann Translation:

$$P' = (TR)P = MP = R_{3 \times 3} \cdot P + T$$

$$M = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \left(\begin{array}{ccc|c} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ \hline 0 & 0 & 0 & 1 \end{array} \right) = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix}$$

G. Zachmann Computer-Graphik 1 – WS 11/12
Transformationen 87

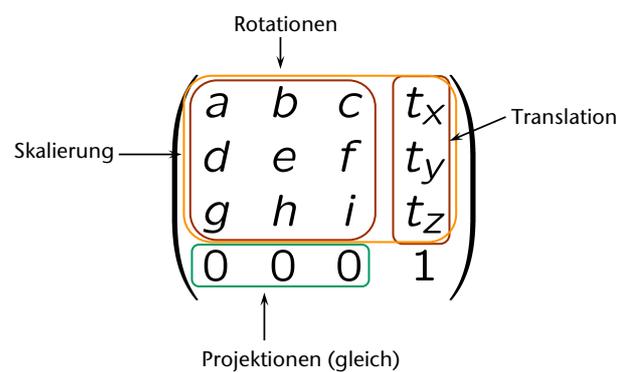
- Erst Translation, dann Rotation:

$$P' = (RT)P = MP \cong R(P + T) = RP + RT$$

$$M = \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} R_{3 \times 3} & R_{3 \times 3} T_{3 \times 1} \\ 0_{1 \times 3} & 1 \end{pmatrix}$$

- Allgemeiner Aufbau:



Starre Transformationen (*Rigid-Body Transform*)

- Hintereinanderausführung von Translationen und Rotationen
- Erhält Längenverhältnisse und Winkel eines Objektes
 - Objekte werden nicht deformiert / verzerrt
- Allgemeine Form:

$$M = T_t R = \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Inverse Rigid-Body Transformation:

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

$$M = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \quad M^{-1} = \begin{pmatrix} R^T & -R^T t \\ 0 & 1 \end{pmatrix}$$

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 90

Transformationen in OpenGL

- Einfache Befehle zur Objekttransformation:

```
glRotate(fd)( TYPE angle, x, y, z );
```

rotiert um **angle Grad(!)** um die angegebene Achse;

```
glTranslate(fd)( TYPE x,y,z );
```

transliert um den angegebenen Betrag;

```
glScale(fd)( TYPE x,y,z );
```

skaliert um die angegebenen Faktoren.

- Ein **glRotate** / **glTranslate** (u.ä.) wirkt sich nur auf die **nachfolgende** Geometrie aus!

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 91

Matrizen in OpenGL

- Es gibt eine „globale“ Matrix „**MODELVIEW**“, die anfangs mit der Einheitsmatrix besetzt ist
- Jeder Aufruf von **glRotate**, **glScale** etc. resultiert in der Multiplikation der entsprechenden Matrix mit der „globalen“ Matrix von **rechts**, z.B.

$$\text{glScalef}(sx, sy, sz) \iff M_{\text{MODELVIEW}} \cdot \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{glTranslatef}(tx, ty, tz) \iff M_{\text{MODELVIEW}} \cdot \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 92

- Beachte die Reihenfolge in einer Matrixkette:

Reihenfolge der OpenGL-Befehle
 $\xrightarrow{\hspace{2cm}}$

$$p' = M_n \cdot \dots \cdot M_2 \cdot M_1 \cdot p$$
 $\xleftarrow{\hspace{2cm}}$
 Reihenfolge der Ausführung

- Die Anordnung entspringt aus dem Programmablauf
- Konzeptionell kann man es sich wie folgt vorstellen:

```
glScalef(1.5, 1, 1);
glTranslatef(.2, 0, 0);
glRotatef(30, 0, 0, 1);
render geometry
```

↑ „Die Geometrie wandert rückwärts durch das Programm und sammelt die Transformationen ein“

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 93

Direkte Matrizenspezifizierung

- Man kann auch direkt Matrizen als Trafo's angeben:


```
glMultMatrix{fd}( TYPE * m );
```

 multipliziert die Matrix auf die aktuelle **MODELVIEW**-Matrix


```
glLoadMatrix{fd}( TYPE * m );
```

 ersetzt die aktuelle **MODELVIEW**-Matrix durch die angegebene.


```
glLoadIdentity();
```

 Spezialfall: Einheitsmatrix
- Matrixabfrage:


```
glGetFloatv( GL_MODELVIEW_MATRIX, float * m );
```

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 94

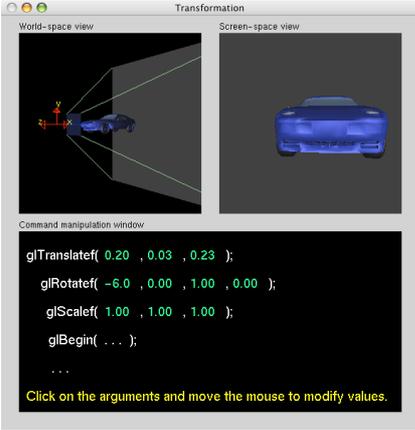
- Achtung: Matrizen werden **spaltenweise** abgelegt, nicht, wie in C üblich, zeilenweise!
 - Das nennt sich "*column-major order*" (der Standard, z.B. in C, ist *row-major order*)

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \iff$$

```
GLfloat matrix[] =
{
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    tx, ty, tz, 1
};
```

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 95

Demo ...

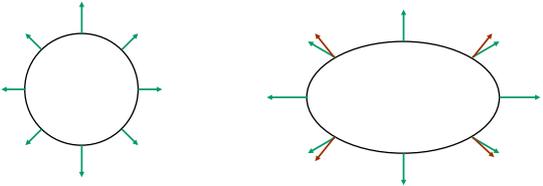


<http://www.xmission.com/~nate/tutors.html>

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 96

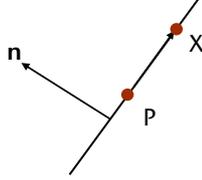
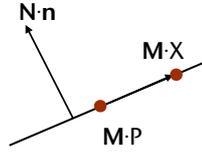
Transformation von Normalen

- Behauptung:
Wenn das Objekt um M transformiert wird, dann müssen die Normalen der Oberfläche um $N = (M^{-1})^T$ transformiert werden
- Bei starren Transformationen:
 - Translation beeinflusst die Normalen der Oberfläche nicht
 - Im Fall der Rotation ist $M^{-1} = M^T$ und somit $N = M$
- Bei nicht-uniformer Skalierung und Scherung ist $N = (M^{-1})^T \neq M$!
 - Beispiel:



G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 97

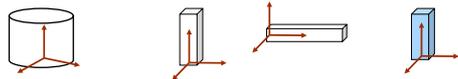
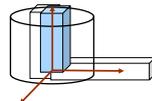
- Beweis:**
wir wissen $(X - P)^T \mathbf{n} = 0$
wir hätten gerne $(M \cdot X - M \cdot P)^T \cdot (N \cdot \mathbf{n}) = (X - P)^T \cdot M^T \cdot N \cdot \mathbf{n} = 0$
setze also $N = (M^T)^{-1}$
damit ist $(X - P)^T \cdot M^T \cdot (M^T)^{-1} \cdot \mathbf{n} = (X - P)^T \cdot I \cdot \mathbf{n} = 0$

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 98

Relative Transformationen

- Eine Konkatenierung von Transformationen kann man auch als eine Folge von (voneinander abhängigen) Koordinatensystemen ansehen
- Beispiel: Roboter**
 - Besteht aus diesen Einzelteilen
- Jedes Teil wurde in seinem eigenen Koordinatensystem spezifiziert (als Array von Punkten) → heißt **Objektkoordinatensystem**
- Rendert man alle Teile ohne jede Transformation, entsteht folgendes:

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 99

- Würde man jedes Teil, ausgehend vom Ursprung des Weltkoordinatensystems, an seinen Platz transformieren, sähe das ungefähr so aus:

```

glLoadIdentity();
// set up camera

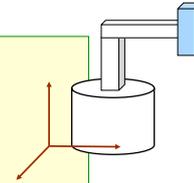
glTranslatef( tr_base_x, tr_base_y , ... );
render base ...

glLoadIdentity();
// set up camera
glTranslatef( tr_base_x, tr_base_y + 10, ... );
render upper arm ...

glLoadIdentity();
// set up camera
glTranslatef( tr_base_x, tr_base_y + 10 + 5, ... );
render lower arm ...

. . .

```



Ann.: Höhe der Basis ist 10

Ann.: Höhe des Oberarms ist 5

- Natürlich macht man es ungefähr so:

```

glLoadIdentity();
// set up camera

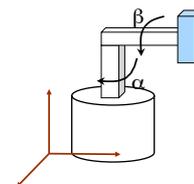
glTranslatef( tr_base_x, tr_base_y , ... );
render base ...

glTranslatef( 0, HEIGHT_BASE, 0 );
glRotatef( alpha, 0, 1, 0 );
render upper arm ...

glTranslatef( 0, HEIGHT_UPPER_ARM, 0 );
glRotatef( beta, 1, 0, 0 );
render lower arm ...

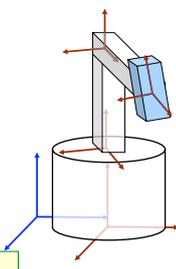
glTranslatef( X_SIZE_LOWER_ARM, 0, 0 );
render hand ...

```



Solche Parameter würde man natürlich in einer Klasse 'Roboter' als Instanzvariablen speichern

■ Alternative Betrachtungsweise ist, daß bei jeder Transformation ein neues **lokales Koordinatensystem** entsteht, das **bezüglich** seines **Vater-Koordinatensystems** um genau diese Transf. transformiert ist



```

glLoadIdentity();
// set up camera
glTranslatef( tr_base_x, tr_base_y , ... );
render base ...

glTranslatef( 0, HEIGHT_BASE, 0 );
glRotatef( alpha, 0, 1, 0 );
render upper arm ...

glTranslatef( 0, HEIGHT_UPPER_ARM, 0 );
glRotatef( beta, 1, 0, 0 );
render lower arm ...

glTranslatef( X_SIZE_LOWER_ARM, 0, 0 );
render hand ...
  
```

In dieser Reihenfolge entstehen die lokalen Koordinatensysteme aus dem Weltkoordinatensystem

In dieser Reihenfolge werden die Transformationen auf die Geometrie (d.h., die Punkte) angewendet

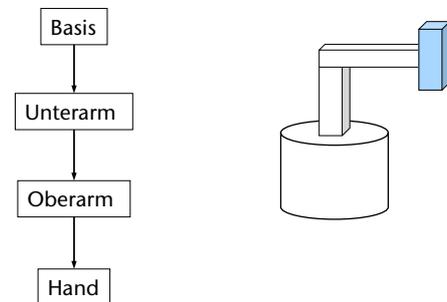
G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 102

Objekthierarchien

■ Dadurch ergibt sich eine Abhängigkeit der Objekte

- Sie betrifft vor allem deren Transformationen
- Betrifft später auch andere Attribute (z.B. Farbe)

■ Der so definierte Baum heißt **Szenengraph**



■ Bemerkung: wir werden in diesem Semester Szenengraphen noch nicht explizit darstellen

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 103

Ein etwas komplizierteres Beispiel:

The diagram shows a robot with a cylindrical base labeled 'Basis'. Two arms, 'Linker Arm' and 'Rechter Arm', extend from the base. Each arm consists of an 'Oberarm' (upper arm) and an 'Unterarm' (lower arm), ending in a 'Hand'. To the right, a hierarchical tree structure represents this structure: 'Basis' is the root, branching into two 'Unterarm' nodes. Each 'Unterarm' node branches into an 'Oberarm' node, which in turn branches into a 'Hand' node. The left side of the tree is labeled '(linker Arm)' and the right side is labeled '(rechter Arm)'.

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 104

Aufgabe: folgende Konfiguration darstellen

Natürliche Vorgehensweise ist Depth-First-Traversal durch den Szenengraph:

The diagram illustrates a 3D coordinate system with red axes and a robot model with green axes. A white arrow points from the coordinate system to the robot model. Below, a hierarchical tree structure is shown with a red line indicating a depth-first traversal path: Basis → Unterarm (left) → Oberarm → Hand → Unterarm (right) → Oberarm → Hand. To the right, a list of drawing commands is shown in a yellow box: 'Do transformation(s)', 'Draw base', 'Do transformation(s)', 'Draw left arm', 'Do transformation(s)', and 'Draw right arm'. Arrows point from the text 'Welche sind das ??' to the second, fourth, and sixth lines of the list.

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 105

Erster (falscher) Versuch

```

    Basis
    /  \
  /    \
Unteraarm  Oberarm
 /  \    /  \
Hand Hand Hand Hand
(linker Arm) (rechter Arm)
  
```

Translate(5,0,0)
 Draw base
 Rotate(75, 0, 1, 0)
 Draw left arm
 Rotate(-75, 0, 1, 0)
 Draw right arm

Was ist hier falsch?!

Antwort: der rechte Arm soll **relativ zur Basis** um -75 Grad gedreht sein, in diesem Programm aber wird er **relativ zum linken Arm** gedreht!

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 106

Lösung

Initiale MODELVIEW Matrix M
 Translate(5,0,0) → M = M·T
 Draw base
 Rotate(75, 0, 1, 0)
 Draw left arm
 Rotate(-75, 0, 1, 0)
 Draw right arm

Speichere die MODELVIEW-Matrix an dieser Stelle in einem Zwischenspeicher

Restauriere diese gemerkte MODELVIEW-Matrix an dieser Stelle aus dem Zwischenspeicher

⇒ Lösung: ein Matrix-Stack

Initiale MODELVIEW Matrix M
 Translate(5,0,0) → M = M·T
 Draw base
 Rotate(75, 0, 1, 0)
 Draw left arm
 Rotate(-75, 0, 1, 0)
 Draw right arm

An dieser Stelle die aktuelle MODELVIEW-Matrix auf den Stack pushen

An dieser Stelle die oberste Matrix vom Stack pop-en und in die MODELVIEW-Matrix schreiben

G. Zachmann Computer-Graphik 1 – WS 11/12 Transformationen 107

Der Matrix-Stack in OpenGL

- In OpenGL gibt es einen **MODELVIEW-Matrix-Stack**
- Die **oberste Matrix** auf diesem Stack ist die **aktuelle** MODELVIEW-Matrix, die für die Geometrie-Transformation verwendet wird
- Alle Transformations-Kommandos (`glLoadMatrix`, `glMultMatrix`, `glTranslate`, ...) operieren auf dieser obersten Matrix!
- Operationen:

```
glPushMatrix();
```

dupliziert die oberste Matrix auf dem Stack und legt diese oben auf dem Stack ab;

```
glPopMatrix();
```

wirft die oberste Matrix vom Stack weg.

Beispiel

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glMultMatrix( M1 );
glTranslate( T );
glPushMatrix();
glRotate( R );
glPushMatrix();
glMultMatrix( M2 );
glPopMatrix();
glScale( S );
glPopMatrix();
```

Aktuelle
MODELVIEW-
Matrix:

I
M1
M1·T
M1·T
M1·T·R
M1·T·R
M1·T·R·M2
M1·T·R
M1·T·R·S
M1·T·R

Zustand des
Matrix-Stacks:

I		
M1		
M1·T		
M1·T	M1·T	
M1·T	M1·T·R	
M1·T	M1·T·R	M1·T·R
M1·T	M1·T·R	M1·T·R·M2
M1·T	M1·T·R	
M1·T	M1·T·R·S	
M1·T		

Beispiel im Code

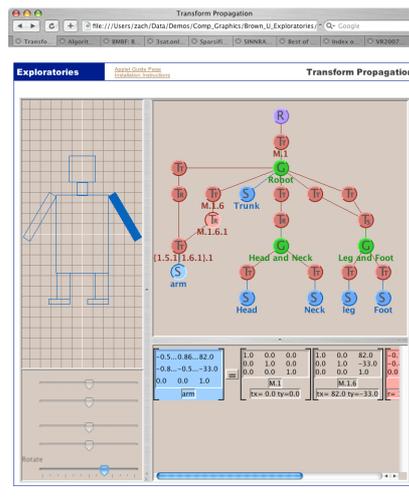
```

void GLWidget::recursiveSphere(const GLfloat * * * )
{
    int dx = m_x() - m_xSphere();
    int dy = m_y() - m_ySphere();

    bool ctrl_Key = m_keyboard() & Qt::Key_Control; // only needed for Mac OS X, but doesn't hurt on other OSes
    if ( (m_keyboard() & Qt::Key_Alt) || ctrl_Key )
    {
        // set location of sphere to 0 * dx;
        // set location of sphere to 0 * dy;
        m_xSphere += dx * dx;
        m_ySphere += dy * dy;
    }
    else if (m_keyboard() & Qt::Key_Return)
    {
        m_xSphere = 0;
        m_ySphere = 0;
    }
    m_xSphere = m_xSphere();
    m_ySphere = m_ySphere();
    update();
}

/** Render a "sphere fiske"
 * @param scaling scaling to be applied with each recursion
 * @param m_recursions number of recursions for the fiske
 * @param lat_long number of latitudes and longitudes
 * @param radius radius of the sphere
 * @param depth This produces spheres that are inside the larger ones (two levels up in the recursion hierarchy)
 */
void GLWidget::renderSphereFiske(const GLfloat * * * , unsigned int m_recursions) const
{
    glColor3f(0.5, 0.5, 0.5);
    if ( m_recursions == 0 )
        return;
    glPushMatrix();
    glTranslatef( m_xSphere(), m_ySphere(), 0.0 );
    glScalef( scaling, scaling, scaling );
    renderSphereFiske( scaling, m_recursions-1 );
    glPopMatrix();
    glTranslatef( m_xSphere(), m_ySphere(), 0.0 );
    glScalef( scaling, scaling, scaling );
}
    
```

Demo zum Szenengraph



<http://www.cs.brown.edu/exploratories> → Transformation Propagation

