4 Kapitel 4: Transformationen und Szenegraph

4.1 Grundlagen und homogene Koordinaten

Definition 4.1.1 (absoluten Positionen) – Punkt im Raum

- Punkte z.B. Mittelpunkt der Ellipse, Ecken des Würfels

Definition 4.1.2 (relative Größen) – Verschiebung von Punkten

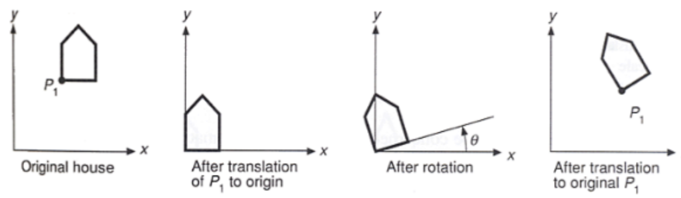
- Vektoren z.B. Radien etc.

Wirkweise:

- Translation: Punkte verändern, aber Vektoren nicht
- Skalierung: Vektoren verändern, aber Punkte nicht
- → **Vektoren sind Positionsunabhängig, absolute Punkte jedoch nicht**

Rotationen um einen beliebigen Punkt

(1) Translation von P1 in den Ursprung.
(2) Rotation um den Ursprung
(3) Translation von P1 in die ursprüngliche Position



Bemerkung: Die Matrizenmultiplikation ist nicht kommutativ,
d.h. die Reihenfolge der Matrizen muss der Reihenfolge der Operationen entsprechen.

Komposition von Transformationen – Matrizenmultiplikation

- Die Hintereinanderausführung von Rotation, Translation und Skalierung führt auf die Abbildungsgleichung P' = S · (T + R · P).
- Müssen mehrere solcher Transformationen hintereinander ausgeführt werden, so stört die Addition in der der obigen Gleichung: P' =Mn ·...·M3 ·M2 ·M1 ·P)

- Das kostenaufwendigste bei der Matrizenmultiplikation ist die Anwendung von Transformationsmatrizen auf alle Punkte. Diese Transformationsmatrizen zusammenzufassen und dann auf alle Punkte anzuwenden ist weniger kostenintensiv. Da Nutzten wir die Assoziativität aus. Sonst müssen wir pro Punkt immer dieselbe Matrix neu ausrechnen.

Beispielrechnung: 5 Matrizen (4x4) auf 100.000 Punkte.

a.) Ausnutzen der Assoziativität

- Aufwand, um x Matrizen zu einer zu machen
= Anzahl der Operationen pro Element * Anzahl Spalten * Anzahl Zeilen * (Anzahl Matrizen - 1)
= 4 *4 * 4 * (5-1) = 256 (wir nutzten die Assoziativität aus)
- Matrix-Vektor-Multiplikation = (Anzahl der Spalten * Anzahl der Zeilen) * Punkte
= (4 * 4) * 100.000 = 1.6 Mio. (4x4) Matrix
- Aufwand = Aufwand Matrix * (Matrix-Vektor-Multiplikation) = 256 + 1.6 Mio.

b.) Ohne Assoziativität

- Aufwand = Matrizen * Matrix-Vektor-Multiplikation
= 5 * (4 * 4 * 100.000) = 8 Mio.

4.2 Homogene Koordinaten

Definition 4.2 (Homogene Koordinaten)

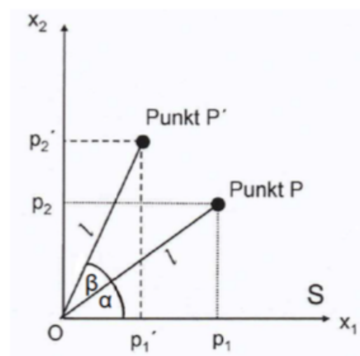
- Das Quadrupel $[x,y,z,\lambda]^T$ stellt die homogenen Koordinaten des Punktes $[x/\lambda,y/\lambda,z/\lambda]^T \in R_3$ dar.
- Bei homogenen Koordinaten ist ein **Lambda λ** (Flag) was einen Punkt oder einen Vektor definiert.
  - **λ = 0 → Vektor**
  - **λ = 1 → Punkt**
- Da die Translationsangaben in der rechten Spalte sind werden diese mit dem **λ** multipliziert. Falls **λ** = 0 ist wird wie gewollt nichts geändert. Bei einem Punkt mit **λ** = 1 wird etwas geändert

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ \lambda \end{bmatrix} = \begin{bmatrix} x + t_x \cdot \lambda \\ y + t_y \cdot \lambda \\ z + t_z \cdot \lambda \\ \lambda \end{bmatrix}$$

$\lambda = 0$, falls Vektor
$\lambda = 1$, falls Punkt

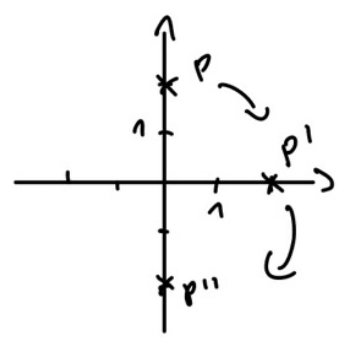
2D-Roation um den Ursprung

$$x = l \cdot \cos\alpha$$
$$y = l \cdot \sin\alpha$$
$$x' = l \cdot \cos\alpha = l \cdot (\cos\alpha \cdot \cos\beta - \sin\alpha \cdot \sin\beta)$$
$$= x \cdot \cos\beta - y \cdot \sin\beta$$
$$y' = l \cdot \sin\alpha = l \cdot (\sin\alpha \cdot \sin\beta - \cos\alpha \cdot \cos\beta)$$
$$= y \cdot \cos\alpha - y \cdot \sin\beta$$
$$\Rightarrow \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$



bei der die Hintereinander-Ausführung zweier Rotationen ist diese nicht kommutativ

- Man nehme zwei Rotationen die nicht um denselben Punkt gehen Rotiere z.B. einen Punkt bei 2,0 um 90° um den Ursprung und danach um 90° um dem Punkt (0,2) => Punkt liegt bei 0,2. Wende die Rot umgekehrt an => Punkt liegt irgendwo im 2. Quadranten mit neg x und pos y Koordinate



Homogene Koordinaten - Rotationen - zunächst 2D

- Rotation eines Punktes $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$ um den Winkel φ um den Ursprung

$$\begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\varphi - y\sin\varphi \\ x\sin\varphi + y\cos\varphi \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

eigentliche Rotation

homogene Koordinaten

- Rotation von $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$ um den Winkel φ um den Punkt $\begin{bmatrix} P_x & P_y \end{bmatrix}^T$

$$\left(\begin{bmatrix} 1 & 0 & P_x \\ 0 & 1 & P_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -P_x \\ 0 & 1 & -P_y \\ 0 & 0 & 1 \end{bmatrix} \right) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

$(M_3 \cdot M_2 \cdot M_1) \cdot P$

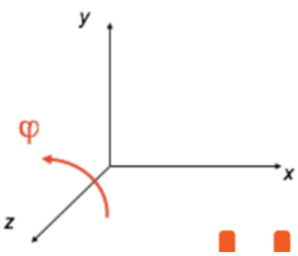
- **Bemerkung:** Auswewrtungsreihenfolge von Rechts nach Links!

Homogene Koordinaten - 3D Rotation um die z-Achse

- Rotation eines Punktes $\begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$ um den Winkel φ um die z-Achse

2D-Rotation

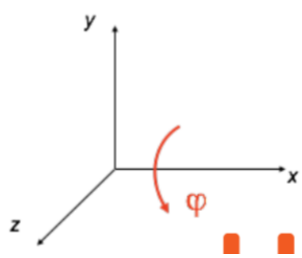
$$\underbrace{\begin{bmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{=:R_z(\varphi)} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\varphi - y\sin\varphi \\ x\sin\varphi + y\cos\varphi \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z \\ 1 \end{bmatrix}$$



Homogene Koordinaten - 3D Rotation um die x-Achse

- Rotation eines Punktes $\begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$ um den Winkel φ um die x-Achse

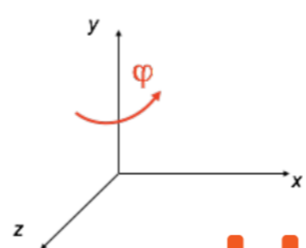
$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{=:R_x(\varphi)} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y\cos\varphi - z\sin\varphi \\ y\sin\varphi + z\cos\varphi \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y' \\ z' \\ 1 \end{bmatrix}$$



Homogene Koordinaten - 3D Rotation um die y-Achse

- Rotation eines Punktes $\begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$ um den Winkel φ um die y-Achse

$$\underbrace{\begin{bmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{=:R_y(\varphi)} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} z\sin\varphi + x\cos\varphi \\ y \\ z\cos\varphi - x\sin\varphi \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y \\ z' \\ 1 \end{bmatrix}$$

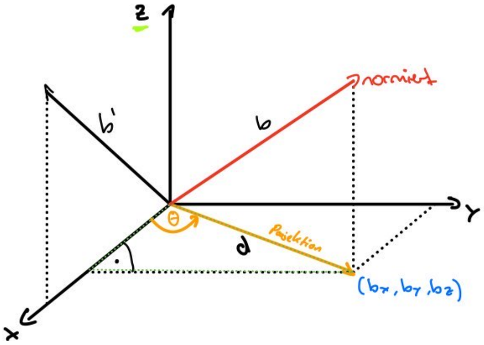


- **Bemerkung:** Drehung um den Winkel φ um die y-Achse entspricht dem 2D-Fall, wobei die y-Koordinate konstant bleibt.

4.2.1 3D Rotation um eine beliebige Achse durch den Ursprung

Schritt 1: Dreh den Vektor b in die zx-Ebene (etwas sortierter)

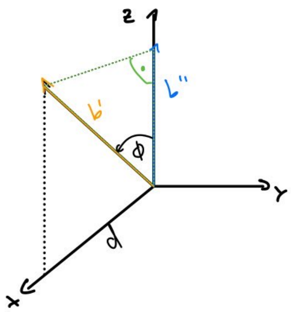
- Annahme: Achse geht durch den Ursprung **und** b normiert!
- Die beliebige Achse b um einen Winkel -tetta um die z-Achse rotieren. Ergebnis b‘



$$\cos(-\theta) = \cos(\theta) = \frac{b_x}{d}$$
$$\sin(-\theta) = \sin(\theta) = \frac{b_y}{d}$$
$$\boldsymbol{R_z(-\theta)}$$

Schritt 2: Dreh den resultierenden Vektor b\` auf die z-Achse (die neue Richtung fällt in die z-Achse)

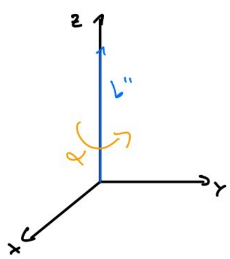
- Die nächste Rotation geht auf die z-Achse um die y-Achse.



$$\cos(-\phi) = \cos(\phi) = \frac{b_z}{\mathbf{1}} = \boldsymbol{b_z}$$
$$\sin(-\phi) = \sin(\phi) = -\frac{d}{1} = -\boldsymbol{d}$$
$$\boldsymbol{R_y(-\phi)}$$

Schritt 3: Dreh den resultierenden Vektor b\`\` um die z Achse um den Winkel alpha

- Nun folgt die eigentliche Rotation um alpha **(Drehung um die z-Achse)**



$$\cos(\alpha)$$
$$\sin(\alpha)$$
$$\boldsymbol{R_z(\alpha)}$$

Schritt 4: Alles am Ende wieder Rückwärts rechnen

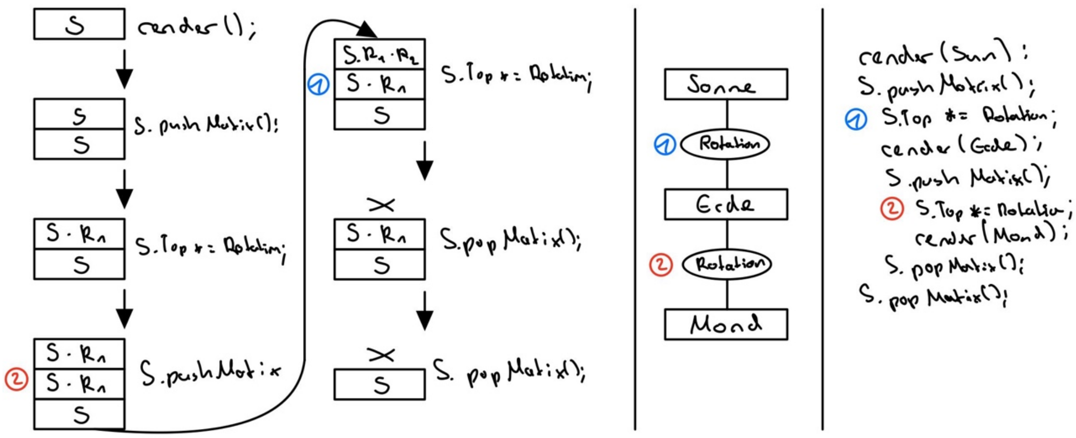
$$R_b(\alpha) = \underbrace{R_z(\theta) * R_y(\phi)}_{\text{wieder zurück rotieren}} * R_z(\alpha) * \underbrace{R_y(-\phi)}_{\text{af z}} * \underbrace{R_z(-\theta)}_{\text{in xy-Ebene}}$$

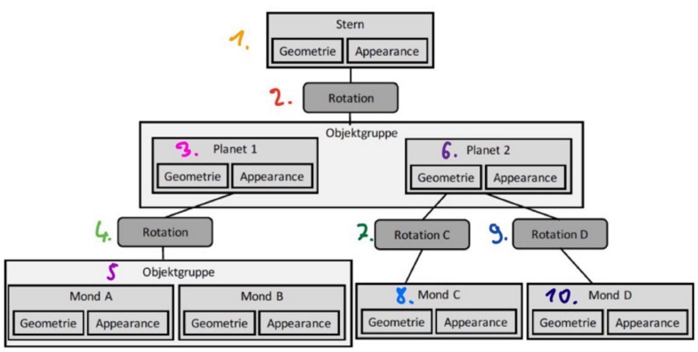
4.3 Grundidee Szenegraph

- Bewegung in oberer Hierarchie wirken sich auf Objekte unterer Hierarchien aus z.B. die Erde rotiert um die Sonne und zusätzlich rotiert es seine eigene Achse. **(DSF erst in die Tiefe dann in die Breite)**

4.3.1 Szenegraph - Abarbeitung mit Matrix-Stack

- Bewegung in oberer Hierarchie wirken sich auf Objekte unterer Hierarchien.
- Die Idee eines Stacks ist, das man die aktuelle Matrix noch mal kopiert (pushen) und danach die Transformationsmatrix des aktuell zu zeichnenden Objektes da dran zu multiplizieren.
- Diese Matrix kann in Form eines Baumes abgearbeitet werden. Wichtig ist nicht zu vergessen das wenn keine Kinder mehr vorhanden sind, die aktuelle Matrix zu Popen (vergessen).






```
1.) render(Stern, S.top());
    S.pushMatrix();
        2.)S.Top() *= Rot1;
        3.)render(Planet1, S.top());
           S.pushMatrix();
            4.)S.Top() *= Rot2 ;
            5.)render(Mond A, S.top());
               render(Mond B, S.top());
           S.popMatrix();
        6.)render(Planet2, S.top());
           S.pushMatrix();
            7.)S.Top() *= RotC ;
            8.)render(Mond C, S.top());
           S.popMatrix();
           S.pushMatrix();
            9.)S.Top() *= RotD ;
           10.)render(Mond D, S.top());
           S.popMatrix();
    S.popMatrix();
```

- Graph iterieren in die Tiefe => **Depth First Search**

Weitere Erweiterungen:

- Die Trennung von Geometrie und **Erscheinung**, jedes Objekt(-Gruppe) kann eine andere Farbe etc. haben
- Es werden **Geometire-Referenzenbenutzt**, also es wird immer dieselbe Instanz mit verschiedenen Transformationen und Aussehen gezeichnet. → Ein Objekt wird nur 1x auf die Grafikkarte initialisiert.

Erweiterungsmöglichkeit A: Factory

- Diese Factory erzeugt und verwaltet die Objekte gibt das Objekt in der passenden Auflösung und Detailliertheit zurück.

Erweiterungsmöglichkeit B: Objektselektierung

- Automatische Selektion der passenden Auflösung für ein geometrisches Objekt durch CgScenegraphEntity selbst