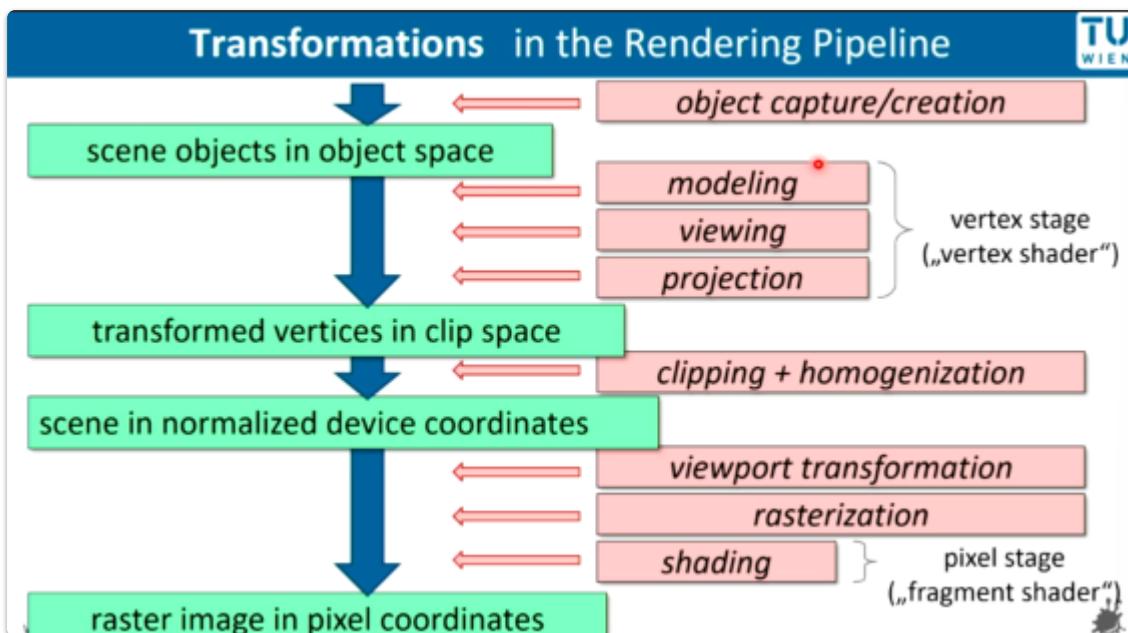


# 3. Transformationen

Behandelt:

- Verschieben
  - Vergrößern
  - Verkleinern
  - Drehen
  - Spiegeln
  - usw...
- ...innerhalb oder zwischen Koordinatensystemen.

In der Rendering Pipeline wäre das hier:



## Einfache 2D-Transformationen

**Translation (Verschiebung)**

Das Verschieben eines Punktes  $(x, y)$  um den Vektor  $(t_x, t_y)$  liefert den transformierten Punkt:

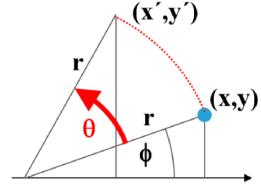
$$(x', y') = (x + t_x, y + t_y)$$

**Rotation (Drehung)**

Durch das Drehen eines Objektes mit dem Winkel  $\theta$  um den Koordinatenursprung kommt der Punkt  $(x, y)$  wegen  $x = r \cdot \cos\theta$  und  $y = r \cdot \sin\theta \Rightarrow x' = r \cdot \cos(\phi + \theta) = r \cdot \cos\phi \cdot \cos\theta - r \cdot \sin\phi \cdot \sin\theta = x \cdot \cos\theta - y \cdot \sin\theta$  (und  $y'$  analog) auf

$$(x', y') = (x \cdot \cos\theta - y \cdot \sin\theta, x \cdot \sin\theta + y \cdot \cos\theta)$$

zu liegen.

**Skalierung (Vergrößerung oder Verkleinerung)**

Beim Skalieren eines Objektes um den Faktor  $s$  um den Ursprung  $(0, 0)$  wird ein Punkt  $(x, y)$  auf

$$(x', y') = (s \cdot x, s \cdot y)$$

abgebildet. Wenn in x- und y-Richtung unterschiedliche Skalierungsfaktoren  $s_x$  und  $s_y$  verwendet werden, dann erhält man

$$(x', y') = (s_x \cdot x, s_y \cdot y).$$

**Reflexion (Spiegelung)**

Die Spiegelung an einer Koordinatenachse ist ein Sonderfall der Skalierung mit  $s_x = -1$  oder  $s_y = -1$ .

Alle anderen Transformationen können durch Hintereinanderausführen der beschriebenen einfachen Transformationen erreicht werden. Diese Abbildungen (mit Ausnahme der Translation) lassen sich auch durch Transformationsmatrizen darstellen. Dabei werden die Punkte als Vektoren dargestellt, um mit ihnen die Matrixoperationen durchführen zu können:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad (x', y') = (x + t_x, y + t_y) \quad \dots?$$

(a) Skalierung

(b) Rotation (gegen Uhrzeigersinn)

(c) Spiegelung um x-Achse

(d) Verschiebung

# Homogene Koordinaten

## Grundprinzip

- Ziel: Auch Translation soll in Matrixform darstellbar sein → **homogene Koordinaten**
- Erweiterung eines Punkts  $(x, y)$  zu  $(x, y, h)$ , meist mit  $\mathbf{h} = 1$
- Rückrechnung in 2D:
  - $x = \frac{x'}{h}$
  - $y = \frac{y'}{h}$

## Grundlegende Transformationen

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

(a) 2D-Skalierung

(b) 2D-Rotation

(c) 2D-Translation

## Vorteil einheitlicher Matrixform

- Alle Transformationen können als Matrizen dargestellt und kombiniert werden
- Durch **Assoziativität** der Matrizen: Vorab-Multiplikation möglich → eine Gesamtmatrix M  
→ effizientere Berechnung

Warum ist es vorteilhaft, alle Transformationen in einheitlicher Matrixschreibweise zu formulieren? Meist werden größere Teile (Objekte, Bilder) als Ganzes transformiert, d.h. auf jeden Punkt dieser Gebilde wird die gleiche Folge von Transformationen angewendet. Dies entspricht einer sequenziellen Multiplikation eines Punktes  $P$  mit Matrizen  $M_1, M_2, M_3, \dots$ :  $P' = M_1 \cdot P, P'' = M_2 \cdot P', P''' = M_3 \cdot P'', \dots$  Nun kann man sich die Assoziativität der Matrixmultiplikation [ also  $(M_1 \cdot M_2) \cdot M_3 = M_1 \cdot (M_2 \cdot M_3)$  ] zunutze machen und den Rechenaufwand damit massiv reduzieren:

$$\begin{aligned} \text{Statt } P^{(n)} &= M_n \cdot (M_{n-1} \cdot \dots \cdot (M_3 \cdot (M_2 \cdot (M_1 \cdot P)))) \dots \\ \text{schreibt man } P^{(n)} &= (M_n \cdot M_{n-1} \cdot \dots \cdot M_3 \cdot M_2 \cdot M_1) \cdot P \end{aligned}$$

Nun kann man  $M = (M_n \cdot M_{n-1} \cdot \dots \cdot M_3 \cdot M_2 \cdot M_1)$  vorher ausrechnen und diese eine Gesamtmatrix dann auf alle Punkte anwenden.

## Kurznotation für Transformationen

- $T(tx, ty)$  = Translation um Vektor  $(tx, ty)$
- $R(\theta)$  = Rotation um Winkel  $\theta$
- $S(s_x, s_y)$  = Skalierung in x- und y-Richtung

## Inverse Transformationen

- $T^{-1}(tx, ty) = T(-tx, -ty)$
- $R^{-1}(\theta) = R(-\theta)$
- $S^{-1}(sx, sy) = S(1/sx, 1/sy)$

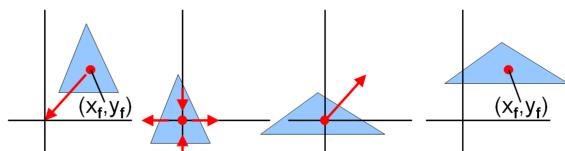
## Komplexere Transformationen

### Skalierung um einen anderen Punkt als den Koordinatenursprung:

1. Schritt = Verschieben des Skalierungszentrums in den Koordinatenursprung:  $T(-x_f, -y_f)$
2. Schritt = Skalieren des Objektes im Koordinatenursprung:  $S(s_x, s_y)$
3. Schritt = Zurückverschieben des Objektes an die ursprüngliche Stelle:  $T^{-1}(-x_f, -y_f) = T(x_f, y_f)$

Die allgemeine Matrix für die Skalierung mit  $(x_f, y_f)$  als Zentrum erhält man nun so:

$$S(x_f, y_f, s_x, s_y) = T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)$$



Als weiteres Beispiel betrachten wir die

### Spiegelung an einer beliebigen Achse $y = mx + b$ :

1. Schritt = Verschieben, so dass die Achse durch den Koordinatenursprung geht:  $T(0, -b)$
2. Schritt = Drehen, so dass die Achse z.B. mit der x-Achse zusammenfällt:  $R(-\theta)$  [ $m = \tan\theta$ ]
3. Schritt = Spiegeln an der x-Achse:  $S(1, -1)$
4. Schritt = Zurückdrehen, so dass Achse ursprünglichen Winkel hat:  $R^{-1}(-\theta) = R(\theta)$
5. Schritt = Zurückverschieben, so dass Achse an der ursprünglichen Stelle liegt:  $T^{-1}(0, -b) = T(0, b)$

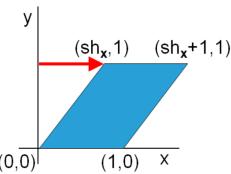
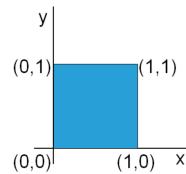
Die allgemeine Matrix für die Spiegelung an der Achse  $y = mx + b$  erhält man nun so:

$$X(m, b) = T(0, b) \cdot R(\theta) \cdot S(1, -1) \cdot R(-\theta) \cdot T(0, -b)$$

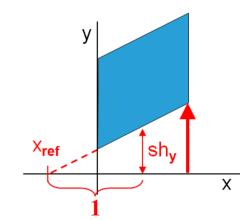
## Scherung

Eine weitere wichtige Transformation ist die Scherung, sie hat im einfachsten Fall in x-Richtung mit fixierter x-Achse die Form:

$$\begin{bmatrix} 1 & \text{sh}_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 0 \\ \text{sh}_y & 1 & -\text{sh}_y \cdot x_{\text{ref}} \\ 0 & 0 & 1 \end{bmatrix}$$



Etwas allgemeiner kann die Scherung auch an einer Parallel zu einer Achse erfolgen, das sieht etwa in y-Richtung dann so aus:

Natürlich kann man jetzt auch wieder ganz leicht die allgemeine Matrix für eine Scherung ableiten, die nicht parallel zu einer der Koordinatenachsen erfolgt: zuerst Drehung in achsenparallele Lage, dann Scherung, und dann wieder zurückdrehen.

## Viewing-Transformation

- Transformation von Weltkoordinaten in **Kamerakoordinaten** (auch Viewport-Koordinaten genannt)
- Kombination aus:
  - Ursprung verschieben
  - Rotation zur Achsenausrichtung
  - Skalierung der Achsen
- Kein Rückverschieben/-drehen nötig

## Affine Transformationen

- Alle behandelten Transformationen = affine Transformationen
- Definition: Koordinaten werden über lineare Abbildung + Translation transformiert
- Eigenschaften:
  - Erhalten Kollinearität (3 Punkte auf Linie bleiben auf Linie)
  - Erhalten Verhältnis von Streckenlängen auf Geraden
  - Parallele Linien bleiben parallel
  - Endliche Punkte bleiben endlich
- Zusammensetzbar aus: Skalierung, Rotation, Translation, Scherung, Spiegelung
- Transformationen mit nur Rotation, Translation, Spiegelung = längen- und winkelerhaltend

# 3D Transformationen

## 3D Transformationen

Alle 2D-Konzepte lassen sich leicht auf 3D erweitern. Man benötigt wieder eine homogene Komponente, so dass 4x4-Matrizen auf 4-dimensionalen Vektoren operieren. Später werden wir sehen, dass man auch Projektionen auf diese Art formulieren kann.

Hier sind einmal die wichtigsten 3D-Transformationen:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) 3D-Skalierung

(b) 3D-Translation

(c) Spiegelung um yz-/xz-/xy-Ebene

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

(d) 3D-Rotation um x-Achse

(e) 3D-Rotation um y-Achse

(f) 3D-Rotation um z-Achse

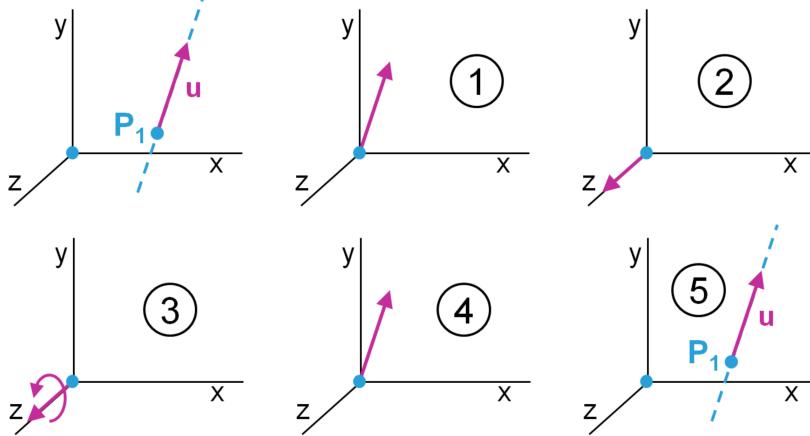
Die Namen für diese einfachen Transformationsmatrizen sehen nun so aus:

- $\mathbf{T}(t_x, t_y, t_z)$  = Translation um den Vektor  $(t_x, t_y, t_z)$
- $\mathbf{R}_x(\theta)$  = Rotation um den Winkel  $\theta$  um die x-Achse; y- und z-Achse analog
- $\mathbf{S}(s_x, s_y, s_z)$  = Skalierung um die Faktoren  $s_x, s_y$  und  $s_z$ .

Als Beispiel für eine komplexere Transformation wollen wir eine

Drehung um den Winkel  $\theta$  um eine beliebige Achse im Raum

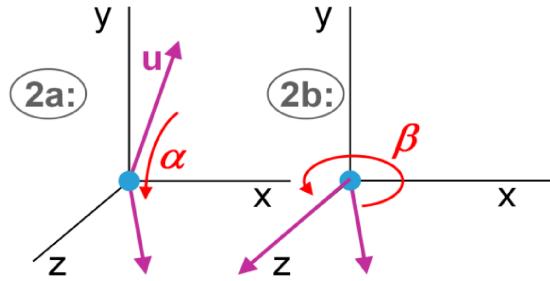
herleiten. Die Achse sei durch einen Punkt  $P_1(x_1, y_1, z_1)$  und einen Richtungsvektor  $u$  gegeben.



1. Schritt = Punkt  $P_1$  in den Koordinatenursprung verschieben:  $T(-x_1, -y_1, -z_1)$
2. Schritt = Vektor  $u$  in die z-Achse drehen
  - (a) Vektor  $u$  um die x-Achse in die xz-Ebene drehen:  $R_x(\alpha)$   
Sei  $u = (a, b, c)$ , dann ist  $u' = (0, b, c)$  die Projektion von  $u$  auf die yz-Ebene. Der Drehungswinkel  $\alpha$  um die x-Achse ergibt sich aus  $\cos\alpha = c/d$  mit  $d = \sqrt{(b^2 + c^2)}$
  - (b) Vektor  $u$  um die y-Achse in die z-Achse drehen:  $R_y(\beta)$   
Der Drehungswinkel  $\beta$  um die y-Achse ergibt sich aus  $\cos\beta = d$  (bzw.  $\sin\beta = -a$ )
3. Schritt = Drehung um  $\theta$  um die z-Achse ausführen:  $R_z(\theta)$
4. Schritt = Vektor  $u$  in die ursprüngliche Richtung zurückdrehen: zuerst  $R_y(-\beta)$ , dann  $R_x(-\alpha)$
5. Schritt = Punkt  $P_1$  an die ursprüngliche Position zurückverschieben:  $T(x_1, y_1, z_1)$

Die resultierende Matrix berechnet sich also so:

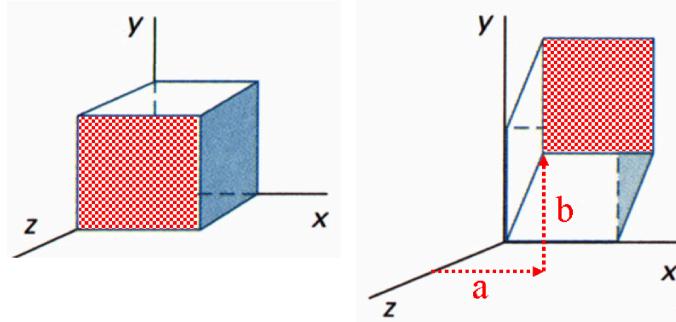
$$\begin{aligned} R(\theta) &= T^{-1}(-x_1, -y_1, -z_1) \cdot R_x - 1(\alpha) \cdot R_y - 1(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T(-x_1, -y_1, -z_1) = \\ &= T(x_1, y_1, z_1) \cdot R_x(-\alpha) \cdot R_y(-\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T(-x_1, -y_1, -z_1) \end{aligned}$$



Die **Scherung in 3D** ist ebenfalls einfach darstellbar:

Eine Scherung parallel zur xy-Ebene um den Wert  $a$  in x-Richtung und den Wert  $b$  in y-Richtung erreicht man mittels

$$\begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Eine Scherung mit einer anderen Fixebene als einer der Koordinatenhauptebenen kann man sich leicht ableiten.

