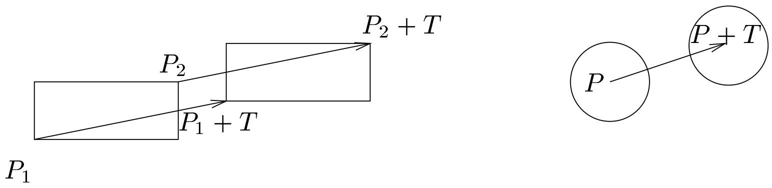
Kapitel 6

2D-Transformationen

Transformationen eines Objekts werden durch Operationen auf den Definitionspunkten realisiert.
Durch 2 Punkte definierte Rechtecke müssen zunächst zu Polygonen gemacht werden.

6.1 Translation

Geradlinige Verschiebung um einen Translationsvektor $T = (t_x, t_y)$, d.h. $P' := P + T$



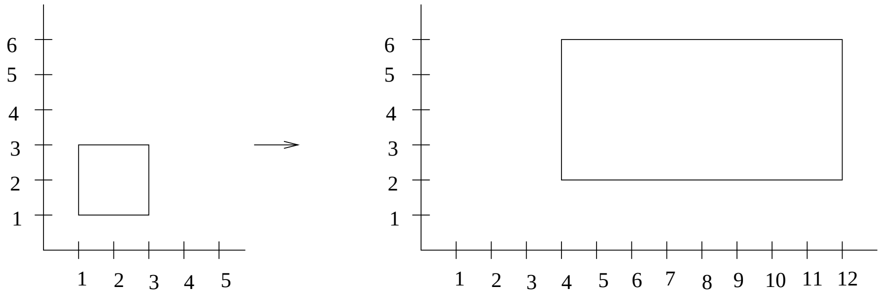
6.2 Skalierung (Scaling, Zooming)

Vergrößerung bzw. Verkleinerung bzgl. eines Fixpunktes. Es liege der Fixpunkt im Ursprung $(0, 0)$.

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

$s_x = s_y \Rightarrow$ uniforme Skalierung; $s_x \neq s_y \Rightarrow$ Verzerrung

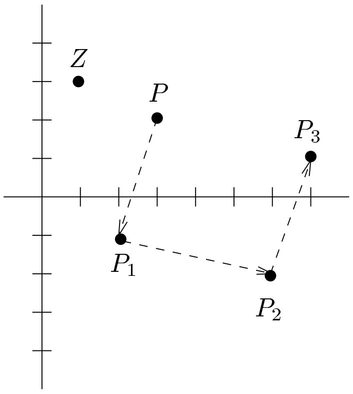
Beispiel: $s_x = 4, s_y = 2$



Bei Wahl eines beliebigen Fixpunktes (Z_x, Z_y) folgt für den Punkt P:

1. Translation um $(-Z_x, -Z_y)$ liefert P_1.
2. Skalierung mit (s_x, s_y) liefert P_2.
3. Translation um (Z_x, Z_y) liefert $P_3 = P'$.

Beispiel: $Z = (1, 3)$, $s_x = 3$, $s_y = 2$



$$\Rightarrow (x', y') := ((x - Z_x) \cdot s_x + Z_x, (y - Z_y) \cdot s_y + Z_y)$$

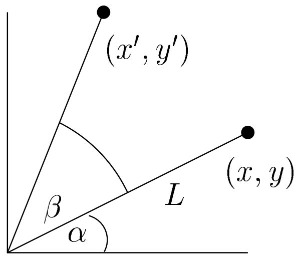
Vorteilhafter bei mehreren Objekten:

$$\begin{aligned} (x', y') &= (x \cdot s_x + d_x, y \cdot s_y + d_y) \\ & \quad \text{mit } d_x = Z_x \cdot (1 - s_x), d_y = Z_y \cdot (1 - s_y) \end{aligned}$$

6.3 Rotation

Drehung des Objekts bzgl. eines Fixpunktes um einen Winkel β.
Es liege der Fixpunkt im Ursprung.
Es gilt der Winkelsummensatz

$$\begin{aligned}\sin(\alpha+\beta) &= \cos(\beta)\cdot\sin(\alpha)+\sin(\beta)\cdot\cos(\alpha)\\ \cos(\alpha+\beta) &= \cos(\beta)\cdot\cos(\alpha)-\sin(\beta)\cdot\sin(\alpha)\end{aligned}$$



$$\begin{aligned}L &= \sqrt{x^2+y^2}\\ \sin(\alpha) &= y/L\\ \cos(\alpha) &= x/L\end{aligned}$$

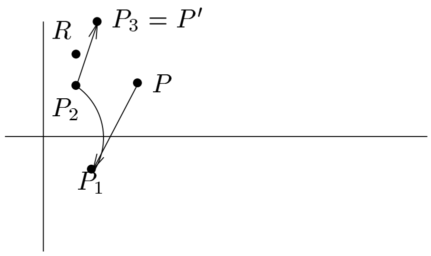
$$\begin{aligned}\sin(\alpha+\beta)=y'/L &= \cos(\beta)\cdot\sin(\alpha)+\sin(\beta)\cdot\cos(\alpha)\\ \Rightarrow y' &= L\cdot\cos(\beta)\cdot\frac{y}{L}+L\cdot\sin(\beta)\cdot\frac{x}{L}\\ \Rightarrow y' &= y\cdot\cos(\beta)+x\cdot\sin(\beta)\end{aligned}$$

$$\begin{aligned}\cos(\alpha+\beta)=x'/L &= \cos(\beta)\cdot\cos(\alpha)-\sin(\beta)\cdot\sin(\alpha)\\ \Rightarrow x' &= L\cdot\cos(\beta)\cdot\frac{x}{L}-\sin(\beta)\cdot\frac{y}{L}\cdot L\\ \Rightarrow x' &= x\cdot\cos(\beta)-y\cdot\sin(\beta)\end{aligned}$$

Positive Werte für den Drehwinkel β bewirken eine Rotation gegen den Uhrzeigersinn.
Bei Wahl eines beliebigen Rotationszentrums (R_x, R_y) folgt für den Punkt P

1. Translation um $(-R_x, -R_y)$, liefert P_1
2. Rotation bzgl. Ursprung um Winkel β, liefert P_2
3. Translation um (R_x, R_y), liefert $P_3 = P'$

Beispiel für $\beta = 90°$:



```
/*************************************************************************************/
/*                                                                                   */
/*                      Skalieren und Rotieren eines Polygons                        */
/*                      Obacht: Da die Polygon-Koordinaten ganzzahlig sind,          */
/*                      verstaerken sich die Rundungsfehler nach jeder Transformation   */
/*                                                                                   */
/*************************************************************************************/

private static final double DEG_TO_RAD = Math.PI/180.0; // Umrechnung Grad in Bogenmass


private void skalier_polygon(          // skaliert ein Polygon
                        Point[] points, // gespeichert im Array points
                        int n,          // mit n Eckpunkten
                        double sx,      // in x-Richtung um den Faktor sx
                        double sy,      // in y-Richtung um den Faktor sy
                        Point  z)       // bezueglich des Punktes z
{
        int i;
        for (i=0; i< n; i++) {

                points[i].x = (int)((points[i].x - z.x ) * sx + z.x + 0.5);
                points[i].y = (int)((points[i].y - z.y ) * sy + z.y + 0.5);

        }
}


private void rotate_polygon(           // rotiert ein Polygon
                        Point[] points, // gespeichert im Array points
                        int   n,        // mit n Eckpunkten
                        int   g,        // um einen Winkel von g Grad
                        Point z)        // bezueglich des Punktes z
{
        int i;
double beta = g * DEG_TO_RAD;   // Winkel im Bogenmass
        Point P = new Point();

        for (i=0; i < n; i++) {

                P.x = (int)((points[i].x - z.x ) * Math.cos( beta ) -
                      (points[i].y - z.y ) * Math.sin( beta ) + z.x + 0.5);

                P.y = (int)((points[i].y - z.y ) * Math.cos( beta ) +
                      (points[i].x - z.x ) * Math.sin( beta ) + z.y + 0.5);

                points[i].x = P.x;
                points[i].y = P.y;

        }
}
```

6.4 Homogene Koordinaten

Häufig werden mehrere Transformationen hintereinander auf ein Objekt angewendet. Es entstehen Rundungsfehler, wenn nach jeder Einzel-Transformation die ganzzahligen Koordinaten bestimmt werden. Deshalb sollten mehrere Transformationen zu einer zusammengesetzt werden.

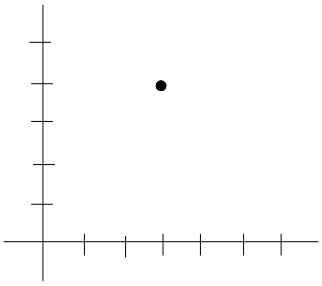
Def.: Ein Punkt $P = (x, y)$ hat die *homogenen* Koordinaten $[x_h, y_h, w]$ mit $w \neq 0$ und

$$\begin{aligned} x_h &= x \cdot w \\ y_h &= y \cdot w \end{aligned}$$

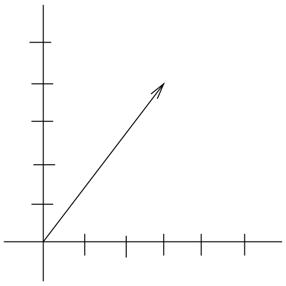
Beispiel: Das homogene Koordinatentripel $[6, 8, 2]$ gehört zum Punkt $P = (3, 4)$, $[3, 4, 1]$ auch.

Ein Richtungsvektor $R = (x, y)$ hat die homogenen Koordinaten $[x, y, 0]$.

Beispiel: Das homogene Koordinatentripel $[3, 4, 0]$ gehört zum Richtungsvektor $(3, 4)$.



Punkt (3, 4)



Richtungsvektor (3, 4)

Die Transformationen Translation, Skalierung und Rotation werden nun als 3×3-Matrizen realisiert. Zusammengesetzte Transformationen ergeben sich durch Matrix-Multiplikation.

Translation

$$[x', y', 1] := [x, y, 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = [x + t_x, y + t_y, 1]$$

Skalierung

$$[x', y', 1] := [x, y, 1] \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x \cdot s_x, y \cdot s_y, 1]$$

Rotation

$$[x', y', 1] := [x, y, 1] \cdot \begin{bmatrix} \cos(\beta) & \sin(\beta) & 0 \\ -\sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x \cdot \cos(\beta) - y \cdot \sin(\beta), x \cdot \sin(\beta) + y \cdot \cos(\beta), 1]$$

Für die Auswertung einer zusammengesetzten Transformation

- gilt das Assoziativgesetz, d.h. $A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$,
- gilt im allgemeinen nicht das Kommutativgesetz, d.h. $A \cdot B \neq B \cdot A$.

Beispiel: Translation um $(-3, -1)$ mit anschließender Rotation um $90°$ ist verschieden von Rotation um $90°$ mit anschließender Translation um $(-3, -1)$.

Beispiel: Rotation bzgl. $(3, 1)$ um $60°$

Matrix für Translation um $(-3, -1)$ lautet

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & -1 & 1 \end{bmatrix}$$

Matrix für Rotation um $60°$ lautet

$$B = \begin{bmatrix} 0.50 & 0.86 & 0.00 \\ -0.86 & 0.50 & 0.00 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$

Matrix für Translation um $(3, 1)$ lautet

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$

Matrix für gesamte Transformation lautet

$$D = A \cdot B \cdot C = \begin{bmatrix} 0.50 & 0.86 & 0 \\ -0.86 & 0.50 & 0 \\ 2.36 & -2.08 & 1 \end{bmatrix}$$

6.5 Allgemeine Transformationen

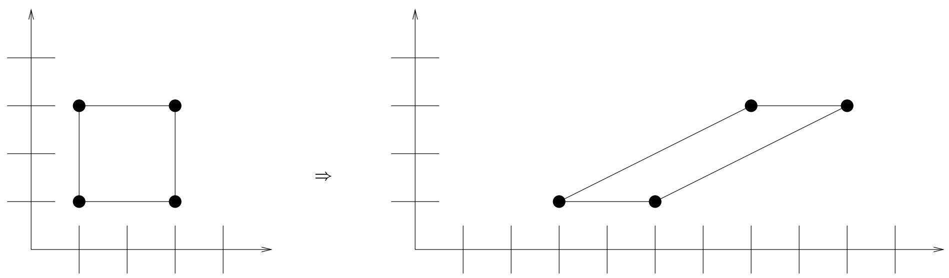
Weitere Transformationen, die sich durch Matrizen darstellen lassen, sind

- Spiegelung an einer beliebigen Geraden,
- Spiegelung an einem Punkt,
- Scherung.

Bei der Scherung in x bleiben die y-Werte konstant, und die x-Werte werden proportional zu den y-Werten horizontal verschoben, d.h. $x' = x + Sch_x \cdot y$.
Die Transformationsmatrix lautet:

$$\begin{bmatrix} 1 & 0 & 0 \\ Sch_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ bzw. } \begin{bmatrix} 1 & Sch_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Beispiel: Scherung mit $Sch_x = 2$



6.6 Raster-Transformationen

Es werden nicht die Definitionspunkte, sondern die Pixel im Frame-Buffer modifiziert:

- Verschieben eines Fensterinhalts (Move),
- Kopieren eines Fensterinhalts (Copy),
- Vergrößern eines Fensterinhalts (Zoom).

Statt Pixelwerte zu setzen, können auch AND-, OR-, XOR-Verknüpfungen verwendet werden. Es gilt $(x \text{ XOR } y) \text{ XOR } y = x$. XOR ist daher geeignet, ein Objekt über den Schirm zu bewegen, ohne den Hintergrund zu zerstören.

```
zeichne Objekt mit XOR
repeat
   loesche Objekt mit XOR
   modifiziere Objekt_Koordinaten
   zeichne Objekt mit XOR
until Objekt am Ziel
```

Allerdings ändert das bewegte Objekt je nach Hintergrundfarbe sein Aussehen.