Computer Graphics

Chapter 7
Two-Dimensional Geometric Transformations

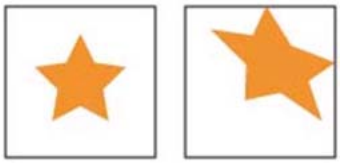
Chapter 7
Two-Dimensional Geometric Transformations

Part I.

- Basic 2D Geometric Transformations
- Matrix Representations and Homogeneous Coordinates
- 2D Composite Transformations

Transformation

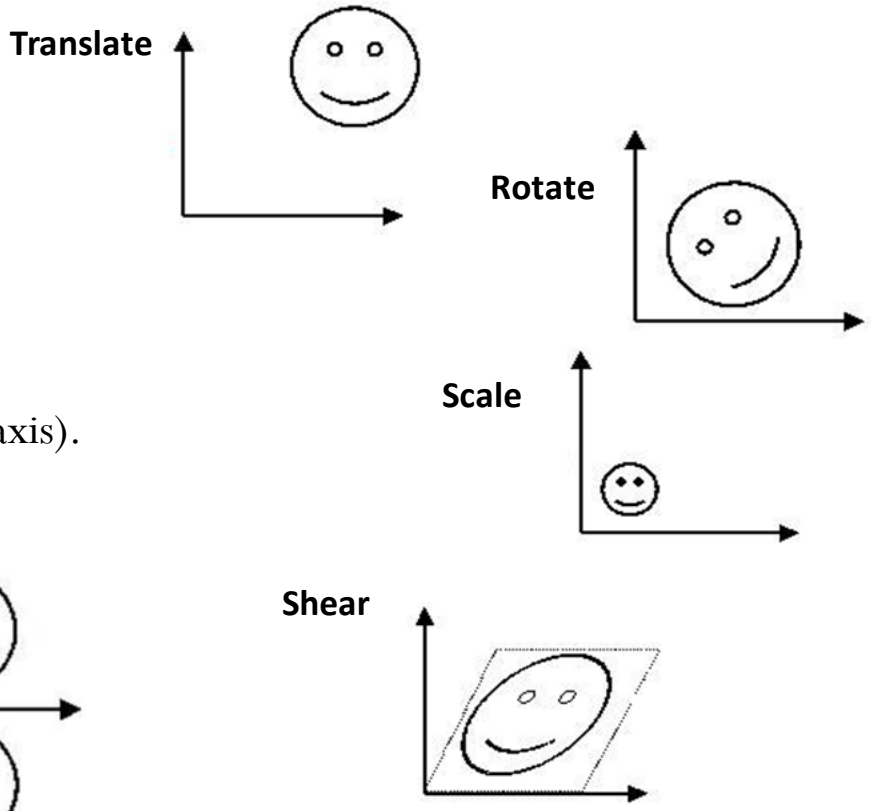
- A transformation is an operation that transforms or changes a shape.
- **Linear** and **affine** (仿射) transformations are central to computer graphics
  - **Linear**: a mapping that preserves linear combinations in a vector space, including rotation, scaling (缩放), shearing(剪切） …
  - **Affine**: any transformation that preserves collinearity (共綫性, i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation).
    - Composed of linear transformations and a translation (or "shift")

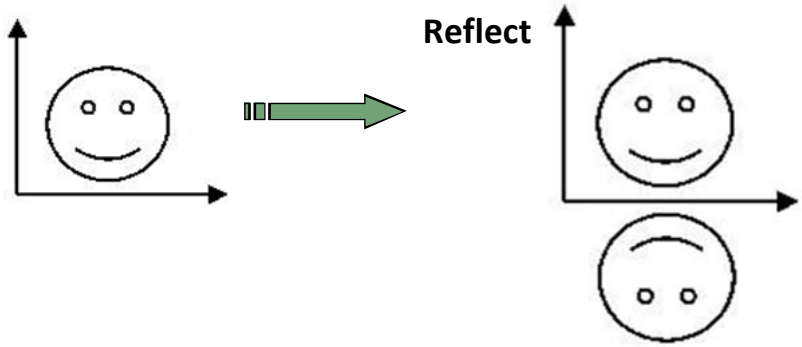


- Important features:
  - Straight lines to straight lines
  - Preserve parallelism
  - Implemented by matrix multiplication

Transformation

- Several basic ways you can change a shape:
  - Translation (moving it)
  - Rotation (turning it round)
  - Scaling (making it bigger or smaller).
  - Shear (changing the main shape).
  - Reflection (mirroring the shape about axis).



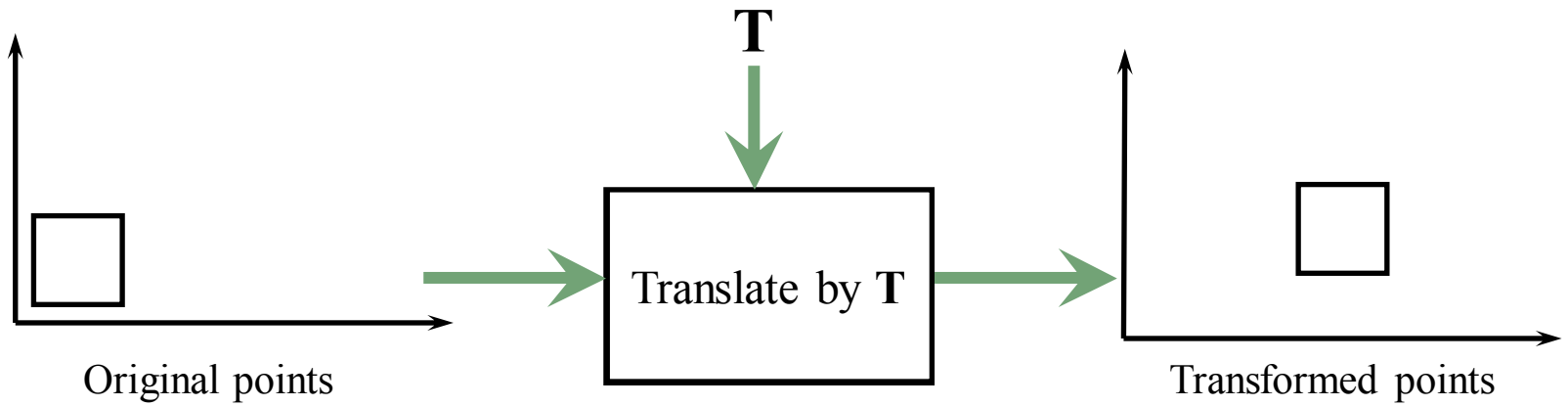


2D Translation

- A **translation** **moves** an object into a different position in a scene **without** changing its shape.

- This is achieved by adding an **offset/translation vector** $\mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$ to the original coordinates $\mathbf{P} = \begin{bmatrix} P_x \\ P_y \end{bmatrix}$

- In vector notation: $\begin{bmatrix} P'_x \\ P'_y \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

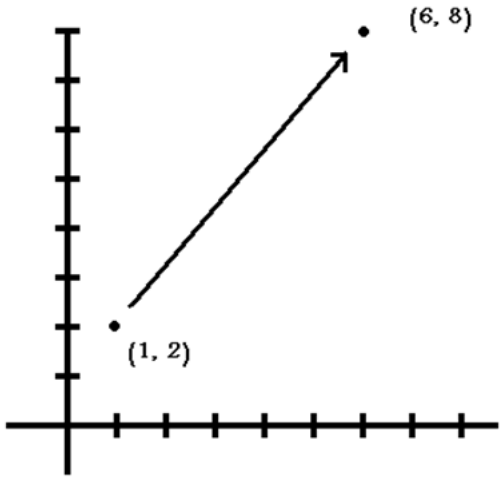


2D Translation

- E.g.: translate the point (1,2) using translation vector (5,6).

$$\begin{bmatrix}1\\2\end{bmatrix}+\begin{bmatrix}5\\6\end{bmatrix}=\begin{bmatrix}1+5\\2+6\end{bmatrix}=\begin{bmatrix}6\\8\end{bmatrix}$$

The new point position is (6, 8).



- To translate line → translate its endpoints.
- To translate polygon → translate its vertices.

2D Translation

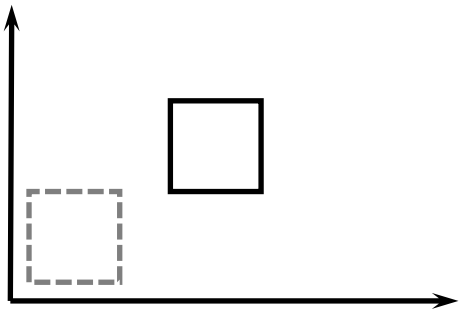
- The routine of the transformation operation

```
void translatePolygon (wcPt2D * verts,  GLint nVerts,  GLfloat tx,  GLfloat ty)
{
        GLint k;

        for (k = 0; k < nVerts; k++) {
            verts [k].x = verts [k].x + tx;
            verts [k].y = verts [k].y + ty;
        }

        glBegin (GL_POLYGON);
         for (k = 0; k < nVerts; k++)
            glVertex2f (verts [k].x, verts [k].y);
        glEnd ();
}
```

```
class wcPt2D {
   public:
      GLfloat x, y;
};
```



2D Rotation

- To rotate points through the specified rotation angle (θ) and rotation axis (Z - axis).
- Rotation axis is perpendicular to XY plane, and they intersect at point (x_r, y_r) (pivot point)
- θ is positive - a counterclockwise rotation.

  θ is negative - a clockwise rotation.

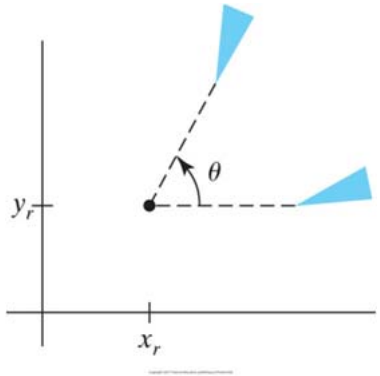


Figure 7-3 Rotation of an object through Angle θ about the pivot point (x_r, y_r).

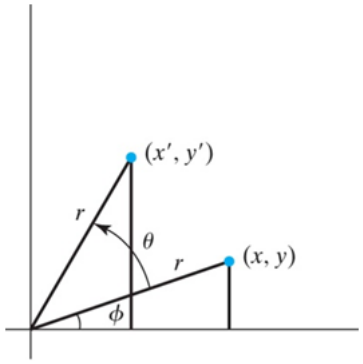


Figure 7-4 Rotation of a point from position (x, y) to position (x', y') through an angle θ relative to the coord. origin.

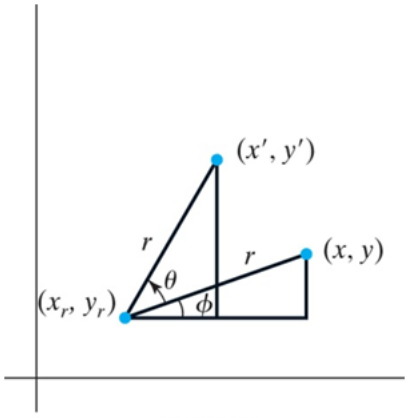
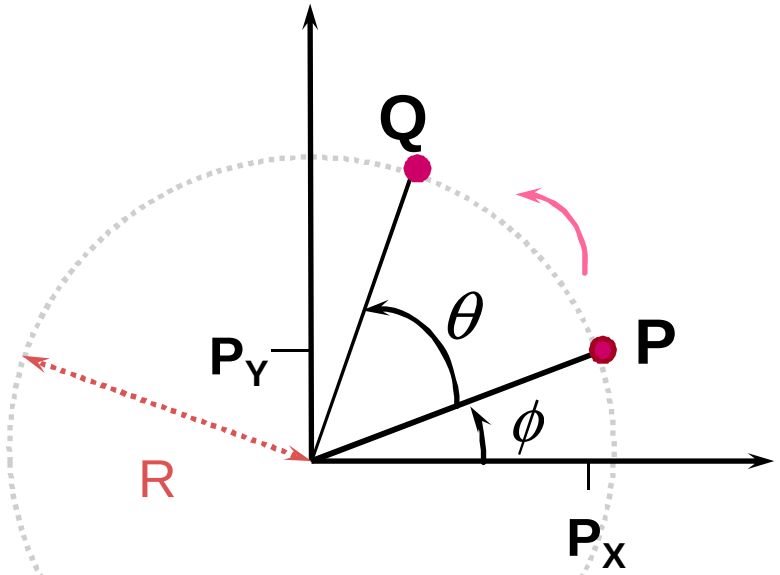


Figure 7-5 Rotating a point from position (x, y) to position (x', y') through an angle θ about rotation point (x_r, y_r).

2D Rotation - derivation

Rotation of point **P** relative to the coordinate origin:



$$Q_X = R\cos(\theta+\phi) \quad [1]$$
$$Q_y = R\sin(\theta+\phi) \quad [2]$$

$$P_X = R\cos(\phi) \quad [3]$$
$$P_y = R\sin(\phi) \quad [4]$$

$$\cos(\theta+\phi)=\cos(\theta)\cos(\phi)-\sin(\theta)\sin(\phi)$$
$$\sin(\theta+\phi)=\sin(\theta)\cos(\phi)+\cos(\theta)\sin(\phi)$$

[1] ⟹ $Q_x = R\cos(\theta)\cos(\phi)-R\sin(\theta)\sin(\phi)$

Substituting [3] and [4] into [1]:

$$Q_x = P_x\cos(\theta)-P_y\sin(\theta)$$

Similarly for [2]:

$$Q_y = P_y\cos(\theta)+P_x\sin(\theta)$$

(Trigonometric identities （三角恒等式）)

2D Rotation - derivation

- With the column-vector representations for coordinate positions, the rotation equations in the matrix form:

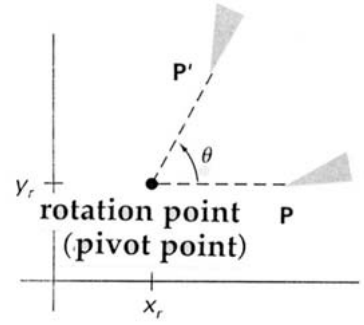
$$\mathbf{Q} = \mathbf{R} \cdot \mathbf{P}$$

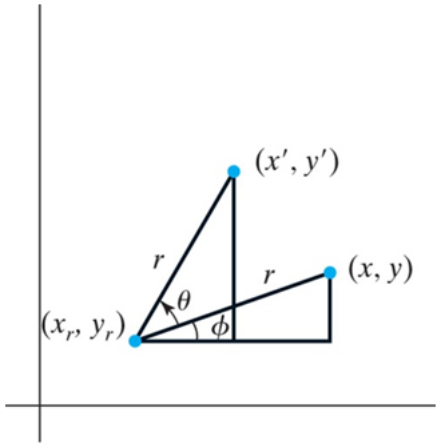
where the rotation matrix is:

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Therefore, general equations for rotation of a point (x, y) about any specified pivot (x_r, y_r) (e.g. Fig. 7-9) :

$$\begin{cases} x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta \\ y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \end{cases}$$





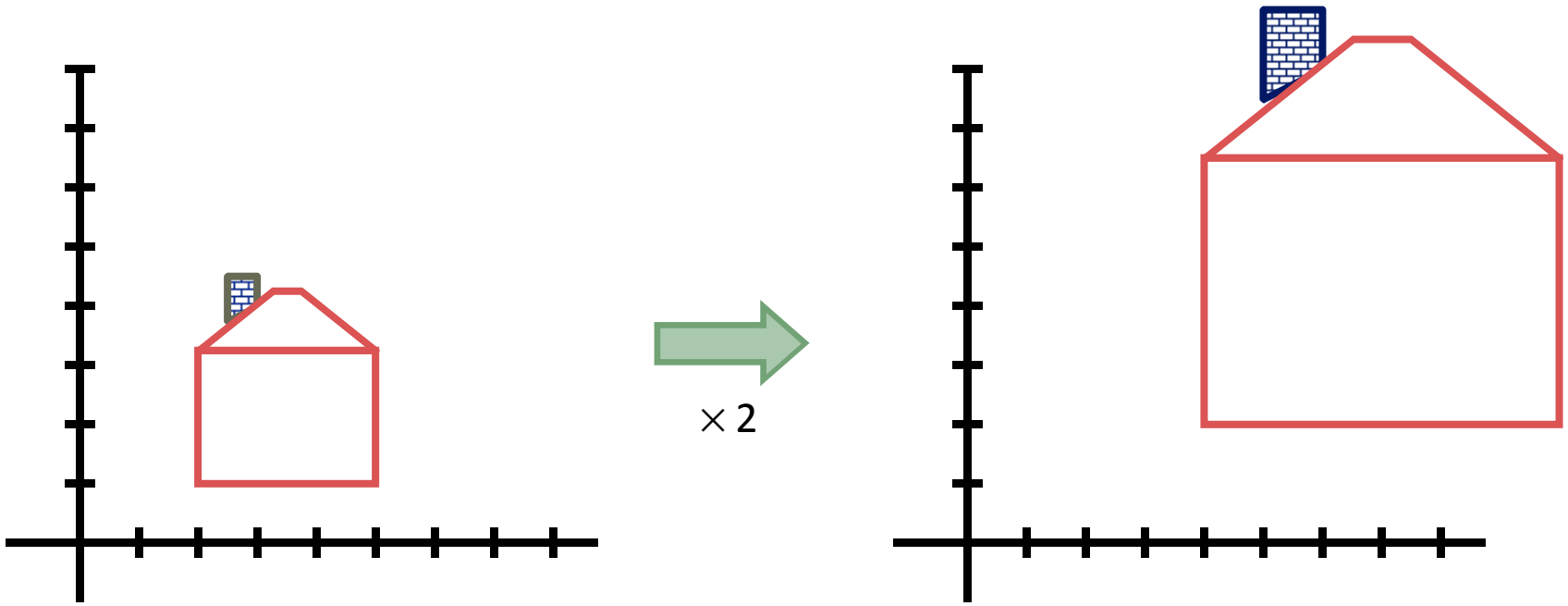
2D Scaling

- To change the size of object.

- Scaling is done by multiplying each coordinate with a scalar (S_x, S_y).
  - Uniform scaling: this scalar is the same for all coordinates.
  - None-uniform scaling: different scalars per coordinate.

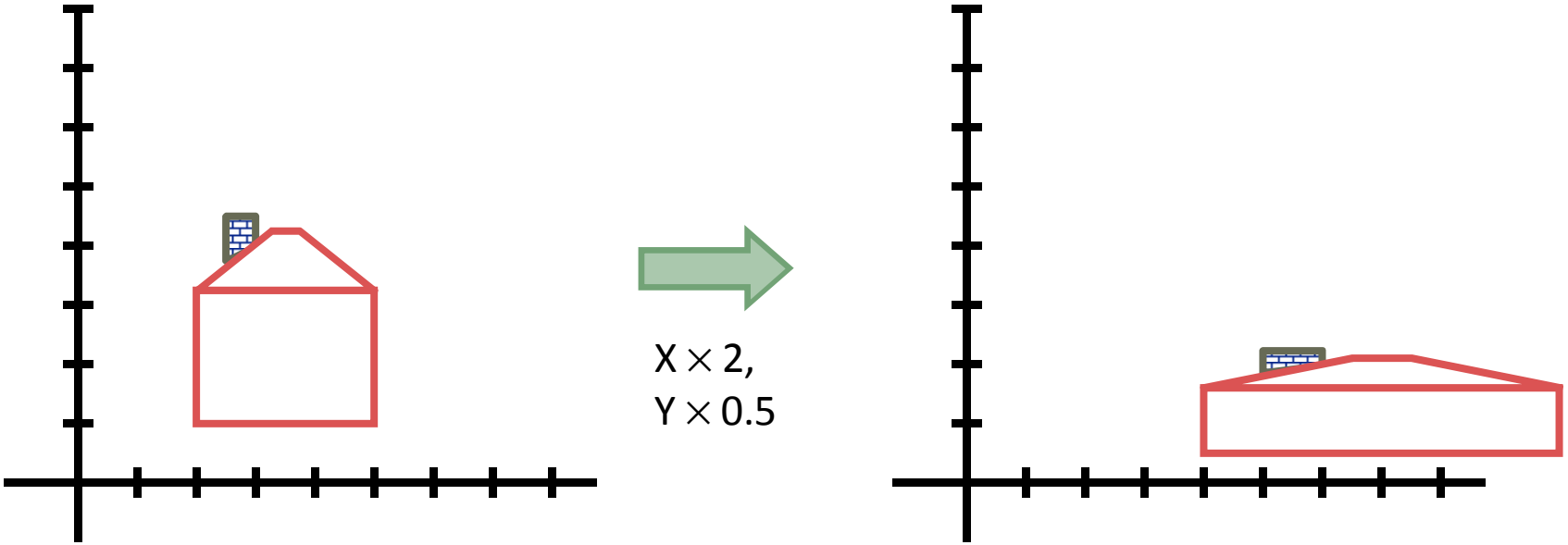
2D Scaling

- Uniform scaling



2D Scaling

- None uniform scaling



2D Scaling – relative to the origin

- Scaling relative to the origin point (0, 0)

$$x' = x \cdot S_x , \quad y' = y \cdot S_y \qquad (7\text{-}10)$$

S_x scales objects in the x direction, while

S_y scales in the y direction.

- In matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \qquad (7\text{-}11)$$

$$\mathbf{P'} = \mathbf{S} \cdot \mathbf{P} \qquad (7\text{-}12)$$

where **S** is the 2 by 2 scaling matrix.

2D Scaling – relative to fixed point

- Scaling relative to the fixed point (x_f, y_f)

$$x' = x_f + (x - x_f)\, s_x, \quad y' = y_f + (y - y_f)\, s_y \tag{7-13}$$

Scaling the distance from the point to the fixed point.

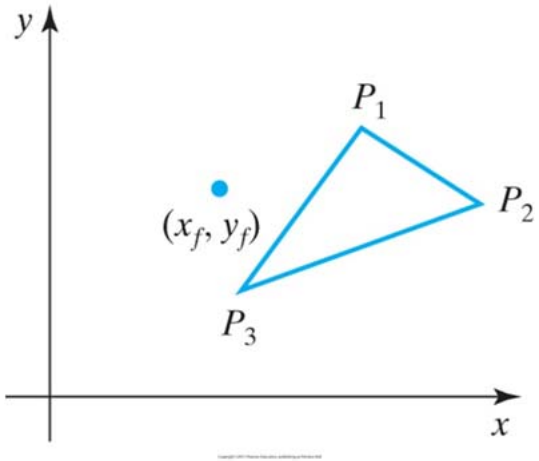


Figure 7-8 Scaling relative to a chosen fixed point (x_f, y_f). The distance from each polygon vertex to the fixed point is scaled by Equations 7-13.

- We can rewrite these scaling transformations to:

$$\begin{cases} x' = x \cdot s_x + x_f(1 - s_x) \\ y' = y \cdot s_y + y_f(1 - s_y) \end{cases}$$

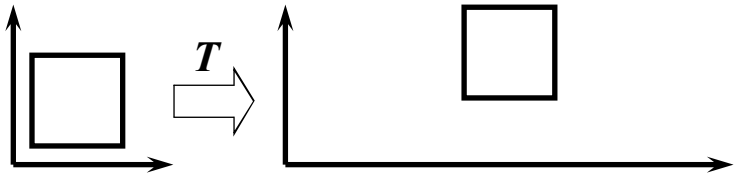
where the additive terms $x_f(1 - s_x)$ and $y_f(1 - s_y)$ are constant.

- It can be represented in the general matrix form.

General Matrix Representation

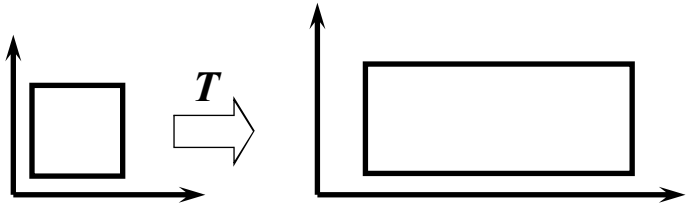
1: Translation

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



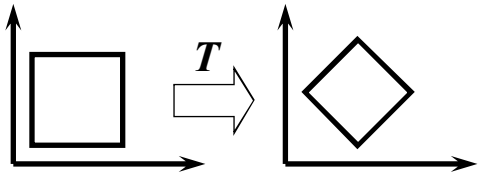
2: Scaling

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



3: Rotation

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



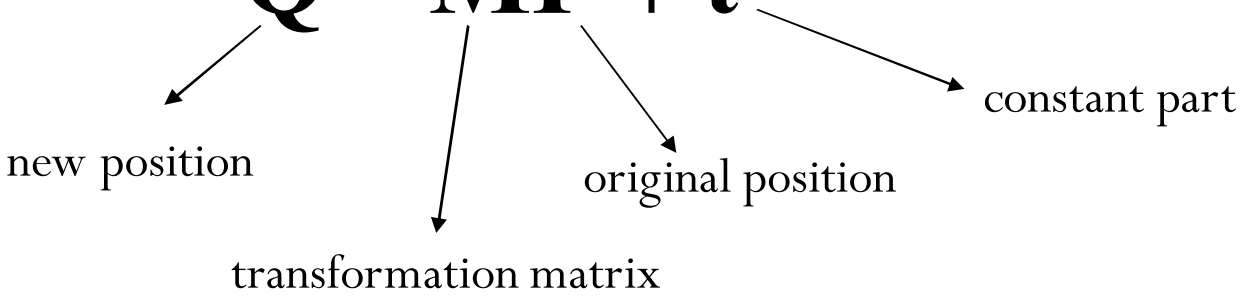
General Matrix Representation

- All these 2D transformations (translation, scaling, rotation) can be generically described in terms of a generic matrix equation as follows:

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

i.e.

$$\mathbf{Q} = \mathbf{MP} + \mathbf{t}$$



Advantage & Disadvantage

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\mathbf{Q} = \mathbf{MP} + \mathbf{t}$$

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- The transformation is composed of a linear combination followed by a translation

- Unfortunately, the translation portion is not a matrix multiplication and must be added as an extra term, or vector – this is inconvenient

Homogeneous Coordinates（齐次坐标）

- The "trick" we use is to add an additional component 1 to both P and Q, and also a third row and column to M, consisting of zeros and a 1

  i.e.

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\mathbf{Q} = \mathbf{MP} + \mathbf{t}$$

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

- Then **all transformation equations can be expressed as matrix multiplications.**
- **Homogeneous coordinates**

Homogeneous Coordinates

- **Homogeneous coordinates**: expand 2D coordinate-position representation (x, y) to a three-element representation (x_h, y_h, h).
  - ***h*** is a homogeneous parameter, which is a **nonzero** value such that

$$x = \frac{x_h}{h}, \quad y = \frac{y_h}{h}$$

  - For geometric transformation, we can choose **any nonzero** value for ***h***, for simplicity, to set ***h=1***.

Homogeneous Coordinates

- For **translation** by (t_x, t_y)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (7\text{-}17)$$

in the abbreviated form

$$P' = T(t_x, t_y) \cdot P \qquad (7\text{-}18)$$

Homogeneous Coordinates

- For **rotation** transformation by θ about the coordinate origin

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (7\text{-}19)$$

in the abbreviated form

$$P' = R(\theta) \cdot P \qquad (7\text{-}20)$$

The rotation transformation operator $R(\theta)$ is the 3 by 3 matrix in Eq.7-19 with rotation parameter θ.

How about θ is replaced with $-\theta$?

Homogeneous Coordinates

- For **scaling** transformation by s_x, s_y relative to the coordinate origin

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (7\text{-}21)$$

in the abbreviated form

$$P' = S(s_x , s_y) \cdot P \qquad (7\text{-}22)$$

Homogeneous Coordinates

- In homogeneous form

Translation by (t_x, t_y)

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation by θ

$$\mathbf{R}^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

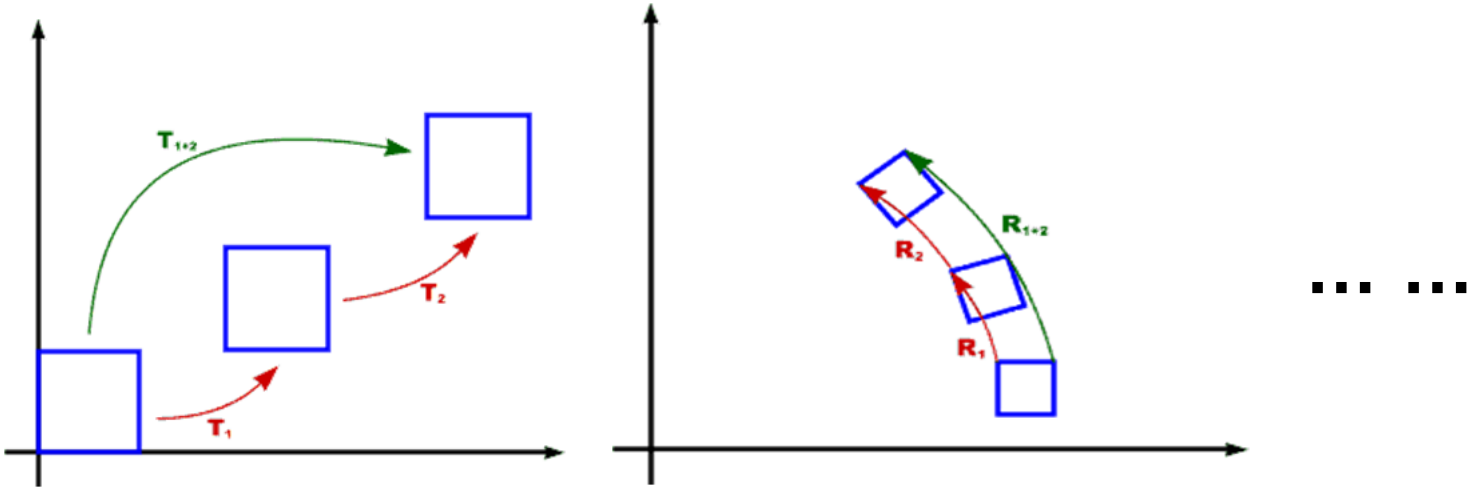
Scale by S_x, S_y

$$\mathbf{S}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

How about the inverse transformation?

2D Composite Transformations

- We can use more than one transformation operation (same type or different) on the same object.



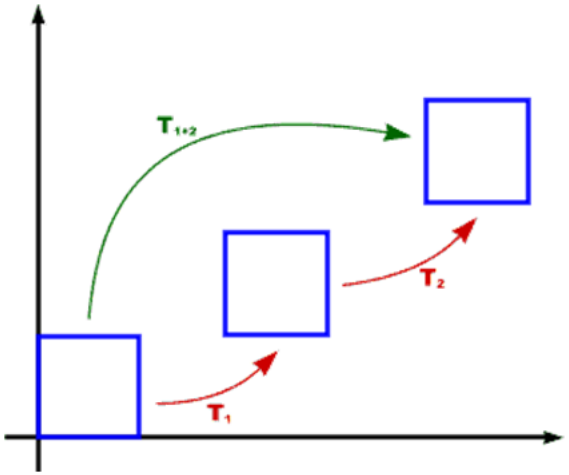
2D Composite Translation

- Two successive translations (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate point $\boldsymbol{P}$

$$P' = T(t_{x2}, t_{y2}) \cdot (T(t_{x1}, t_{y1}) \cdot P) = (T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1})) \cdot P$$

where P and P' are represented as homogeneous coordinate column vectors

$$\begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x1}+t_{x2} \\ 0 & 1 & t_{y1}+t_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$



or $\quad T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1}) = T(t_{x1} + t_{x2}, t_{y1} + t_{y2})$

Two successive translations are **additive** of the translation factor.

2D Composite Rotation

- Two successive rotations with θ_1 and θ_2 round the **origin** applied to point P

$$P' = R(\theta 2) \cdot \left(R(\theta 1) \cdot P \right) = \left(R(\theta 2) \cdot R(\theta 1) \right) \cdot P$$

where P and P' are represented as homogeneous coordinate column vectors

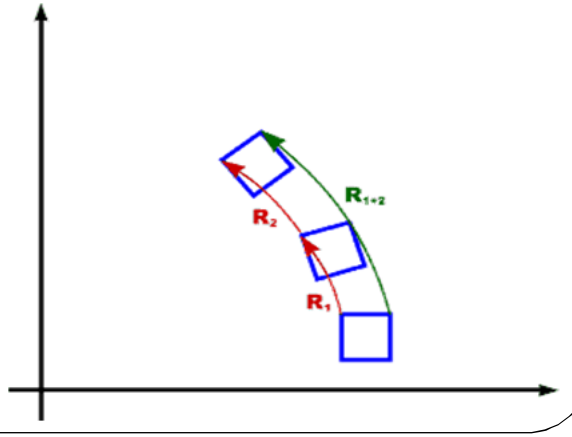
- By multiplying the two rotation matrices, we can verify that two successive rotations are **additive**:

$$R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$$

So that the final rotated coordinates can be calculated with the composite rotation matrix as

$$P' = R(\theta_1 + \theta_2) \cdot P$$

$$where\ R(\theta 1 + \theta 2) = \begin{bmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) & 0 \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



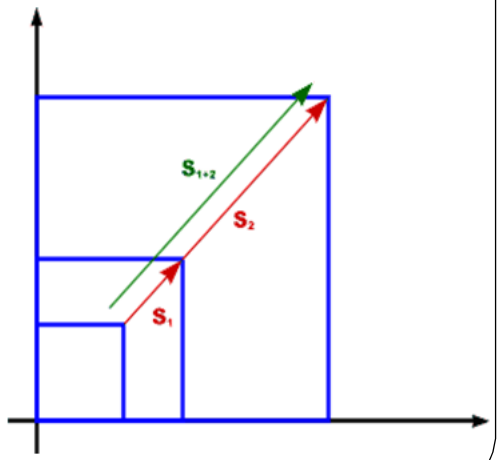
2D Composite Scaling

- Two successive scaling operations

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1}.s_{x2} & 0 & 0 \\ 0 & s_{y1}.s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S(S_{X2}, S_{Y2}) \cdot S(S_{X1}, S_{Y1}) = S(S_{X1} \cdot S_{X2}, S_{Y1} \cdot S_{Y2})$$

- Two successive scaling are **multiplicative**.



General 2D Pivot-Point Rotation

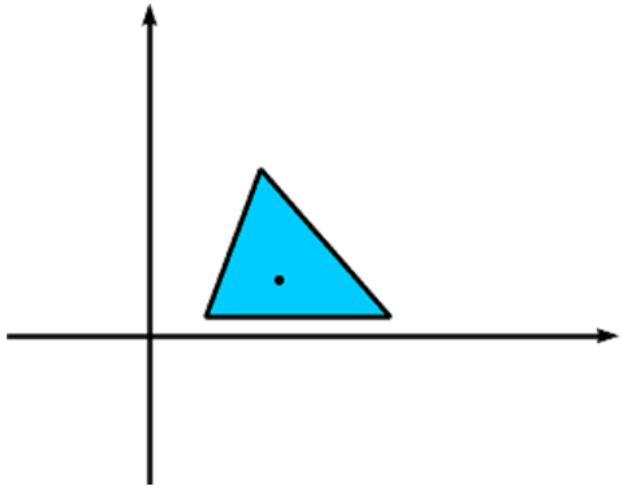
- A transformation sequence for rotating an object about a specified pivot point

(a) (d)

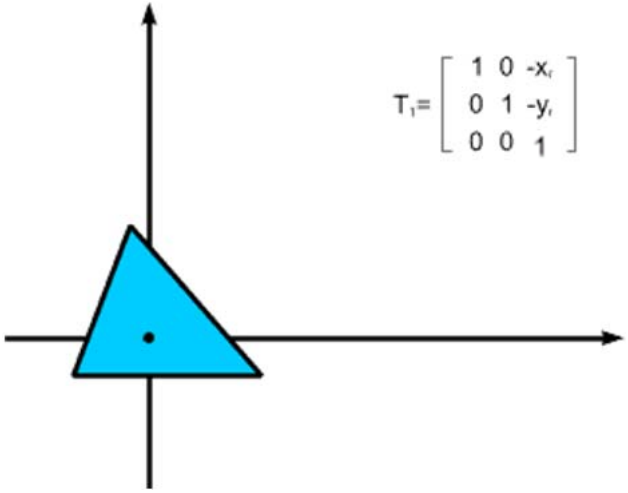
Steps:

- Translate the object so that the pivot-point position is moved to the coordinate origin;
- Rotate the object about the coordinate origin;
- Translate the object so that the pivot point is returned to its original position.

General 2D Pivot-Point Rotation

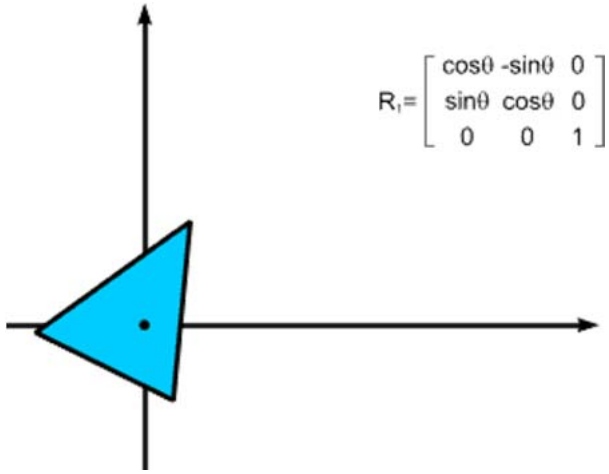


- Original Position of Object and Pivot Point (x_r, y_r).



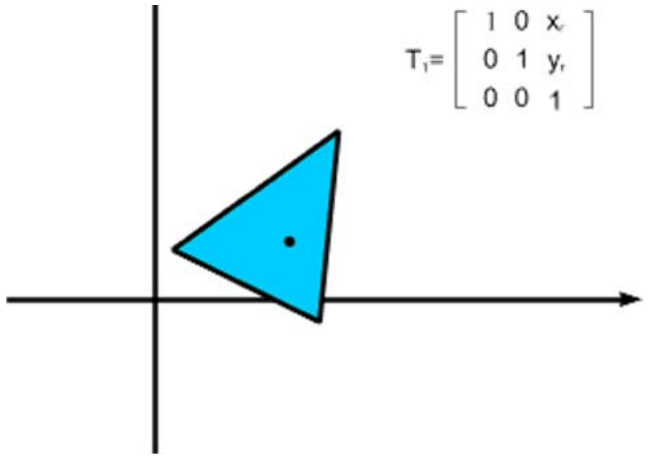
- Translation of Object so that Pivot Point is at Origin.

General 2D Pivot-Point Rotation



$$R_1 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Rotation about Origin.



$$T_1 = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix}$$

- Translation of Object so that Pivot Point is Returned to original Position.

General 2D Pivot-Point Rotation

- Rotation with respect to a pivot point (x_r, y_r)

$$\begin{pmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{pmatrix}$$

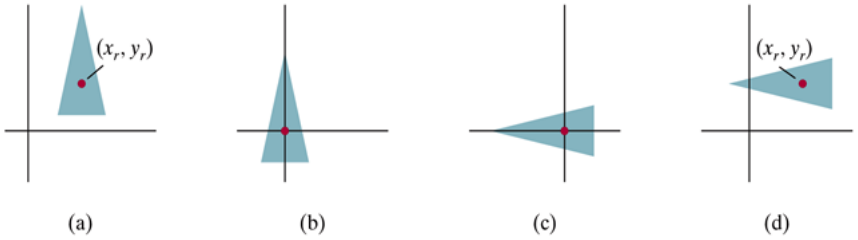
$$= \begin{pmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{pmatrix}$$

or in form

$$T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

where

$$T(-x_r, -y_r) = T^{-1}(x_r, y_r)$$



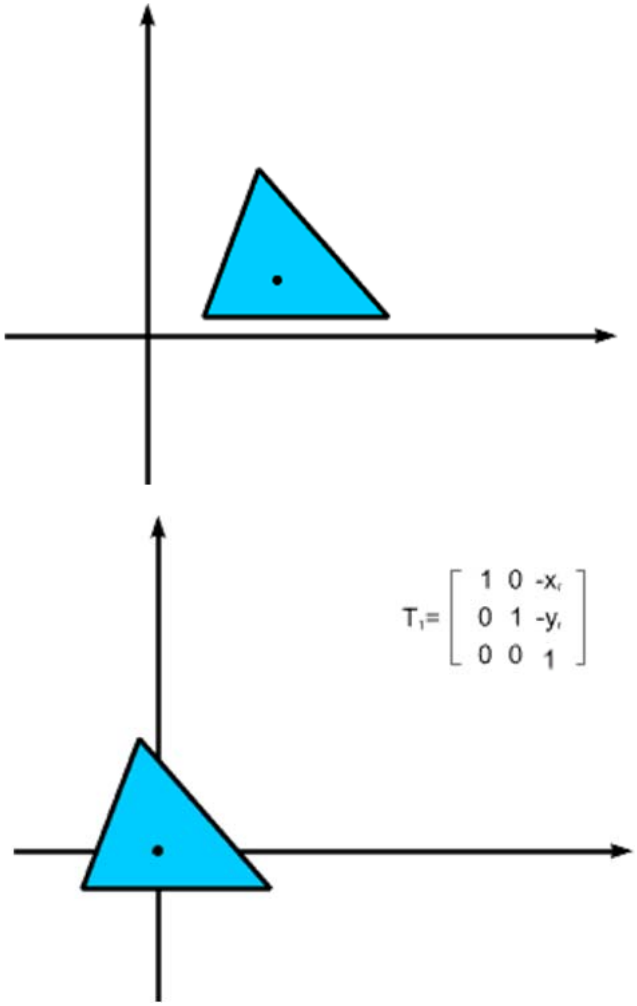
General 2D Fixed-Point Scaling

- A transformation sequence for scaling an object with respect to a specified fixed position using the scaling matrix

Steps:

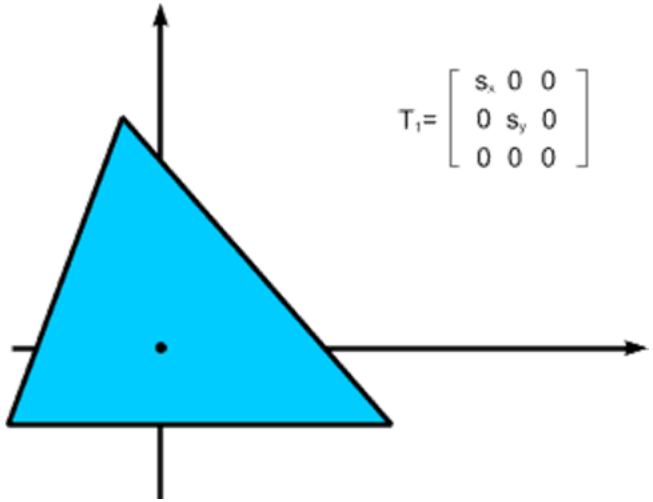
- Translate object so that the fixed point coincides with the coordinate origin;
- Scale the object with respect to the coordinate origin;
- Use the inverse translation of step 1 to return the object to its original position.

General 2D Fixed-Point Scaling



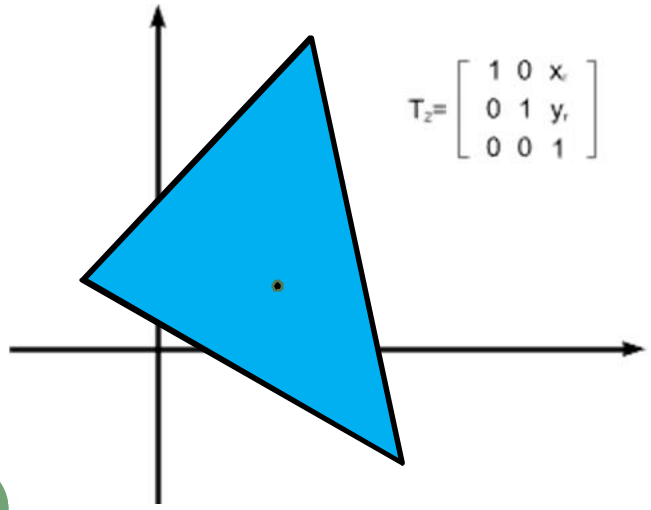
- Original Position of Object and Fixed point.
- Translation of Object so that Fixed Point is at Origin.

General 2D Fixed-Point Scaling



$$T_1 = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Scale Object with Respect to Origin.



$$T_2 = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix}$$

- Translation of Object so that Fixed Point is Returned to original Position.

General 2D Fixed-Point Scaling

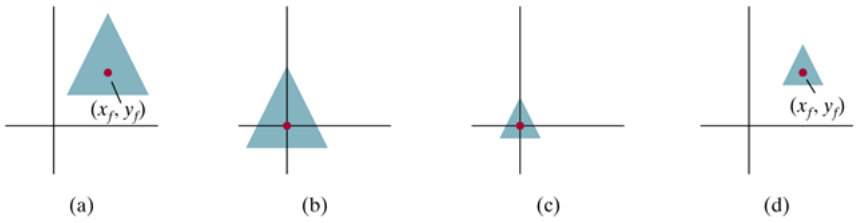
- Concatenating the matrices for these operations produces the required scaling matrix

$$S(x_x, y_y, s_x, s_y)$$

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

$$= \begin{pmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_x & 0 & (1-s_x) \cdot x_f \\ 0 & s_y & (1-s_y) \cdot y_f \\ 0 & 0 & 1 \end{pmatrix}$$



Matrix Composition

- Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

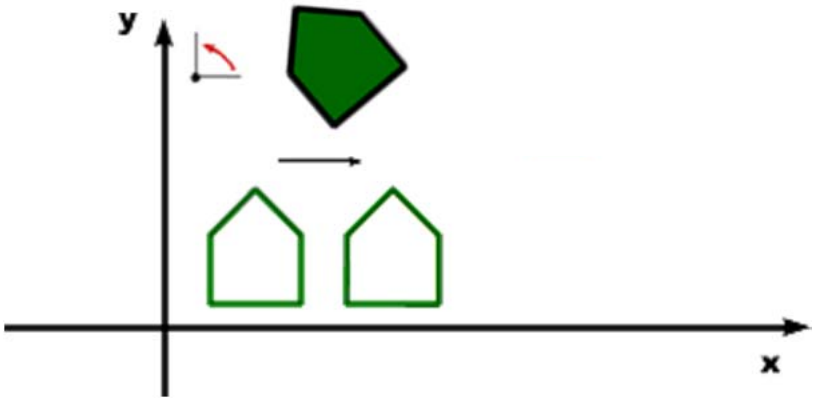
$$\mathbf{p}' = \underbrace{\quad}_{T(t_x,t_y)} \quad \underbrace{\quad}_{R(Q)} \quad \underbrace{\quad}_{S(s_x,s_y)} \quad \mathbf{p}$$

$$\mathbf{p}' = (T * (R * (S*\mathbf{p}) \ ) \ )$$
$$\mathbf{p}' = (T*R*S) * \mathbf{p}$$

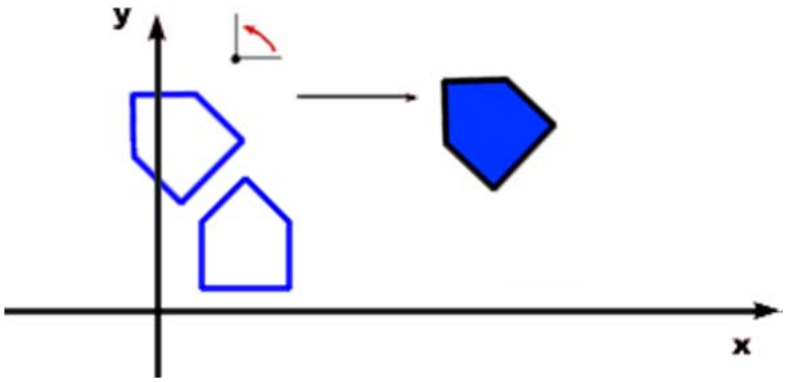
- **Order of transformations**
  - Matrix multiplication is not commutative

$$\mathbf{p}' = T * R * S * \mathbf{p}$$

Order of Transformations



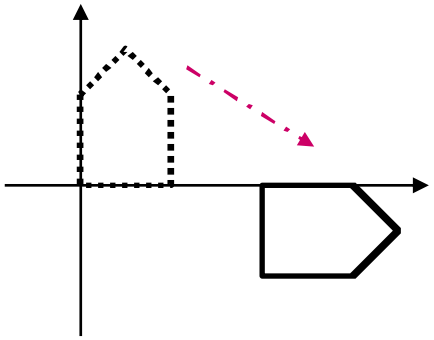
- $P' = R \cdot T \cdot P$



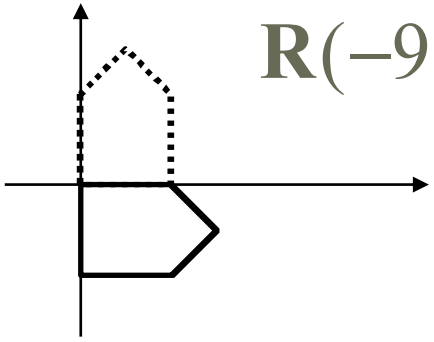
- $P' = T \cdot R \cdot P$

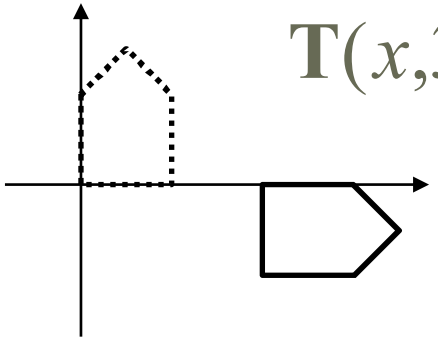
More Example

- Suppose we want,



- We have to compose two transformations



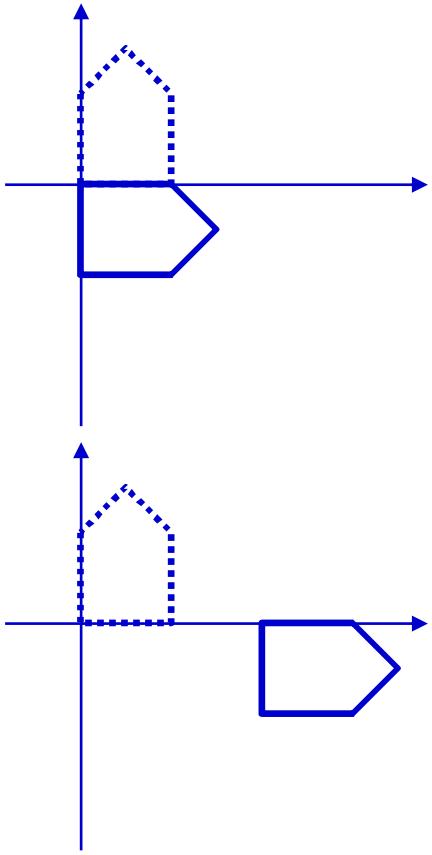


More Example

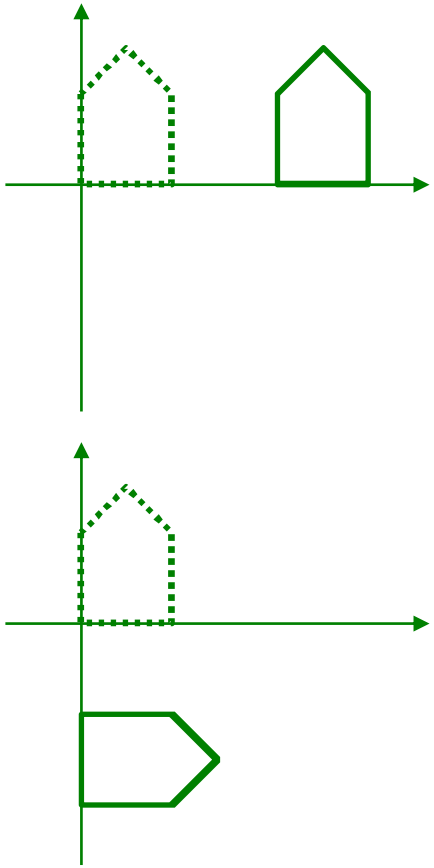
- Matrix multiplication is not commutative

$$\mathbf{T}(x,3)\cdot\mathbf{R}(-90°) \neq \mathbf{R}(-90°)\mathbf{T}(x,3)$$

Rotation followed by translation



Translation followed by rotation



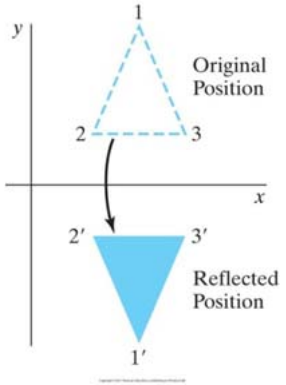
Chapter 7
Two-Dimensional Geometric Transformations

Part II.

- Other 2D Transformations
- Raster Methods for Geometric Transformations
- OpenGL Raster Transformations

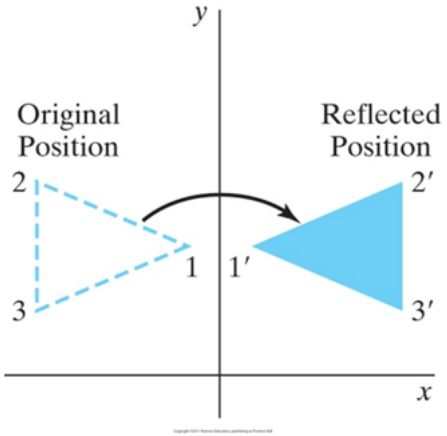
Reflection

- Reflection: produce a mirror image of an object.
- Reflection about x-axis:
  - x' = x
  - y' = -y



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Reflection about y-axis:
  - x' = -x
  - y' = y

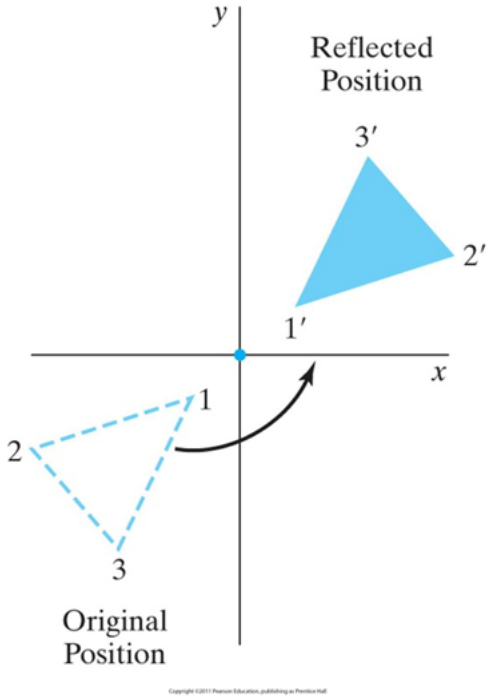


$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

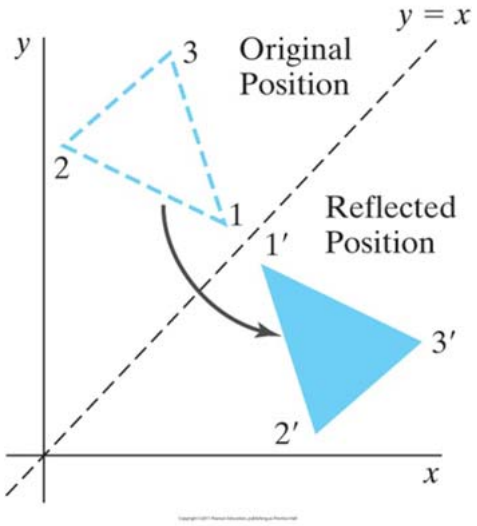
Reflection

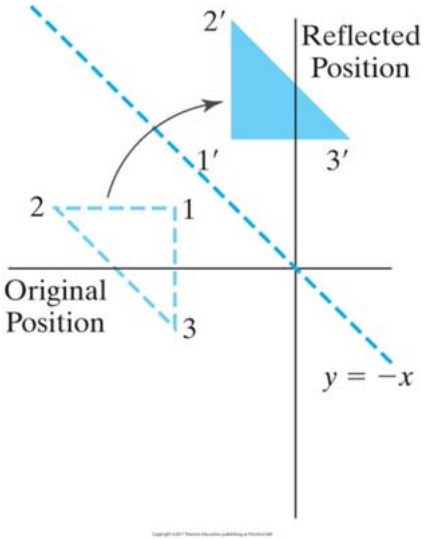
- Reflection about origin point (0, 0)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



- Reflection about the line y = x and y = -x.





$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

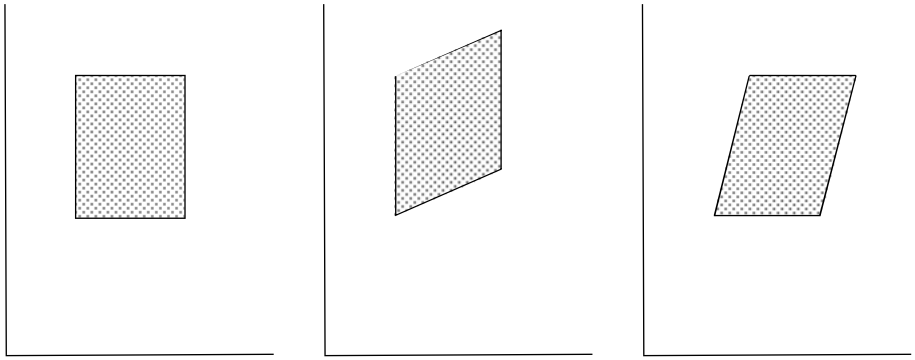
- Shear

  Distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other.

- It is done by changing the value of one coordinate and keeping the other coordinate without changing.

- Two types of shear:
  - To shift in x.
  - To shift in y.

  To stretch rectangles into parallelograms.



X-direction Shear: refer to x-axis

- X- direction shear:

$$x' = x + sh_x \cdot y$$
$$y' = y$$

In matrix representation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \textbf{(7-57)}$$

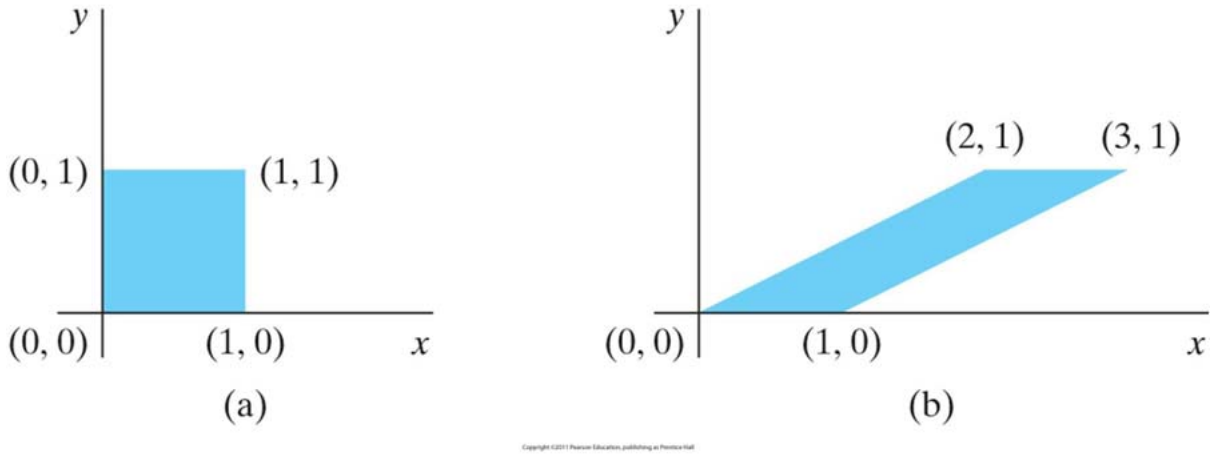
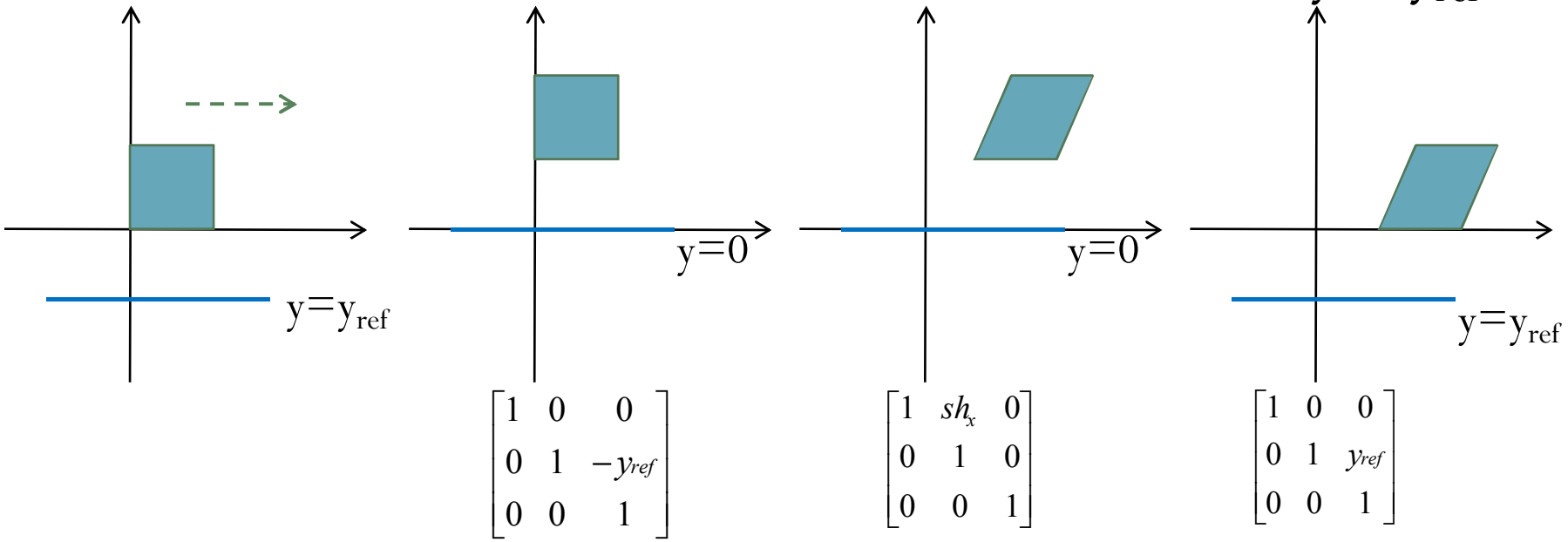


FIGURE 7-23 A unit square (a) is converted to a parallelogram (b) using the x-direction shear matrix 7-57 with $sh_x = 2$.

X-direction Shear: refer to $y=y_{ref}$

- x-direction shears relative to other reference lines $y = y_{ref}$



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -y_{ref} \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -y_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$x' = x + sh_x(y - y_{ref})$$
$$y' = y$$

Y-direction Shear: refer to y-axis

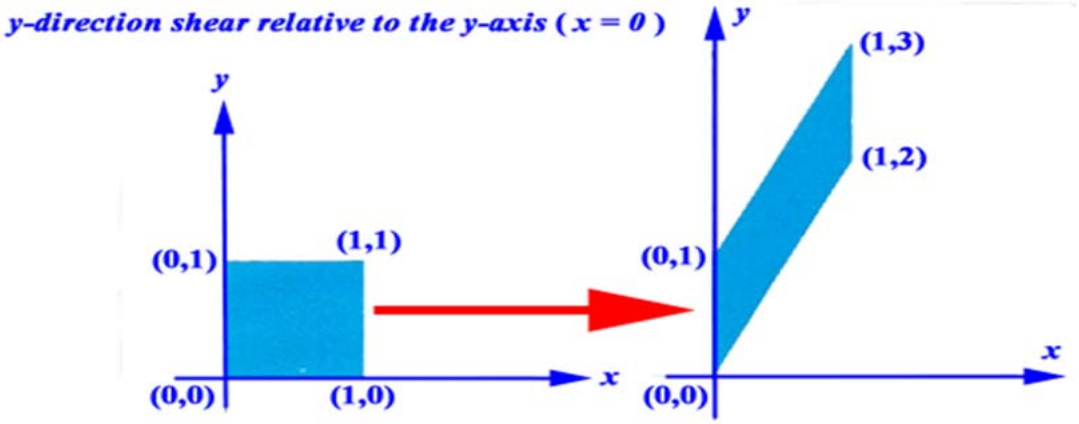
- Y- direction shear:

  $\mathbf{x' = x}$

  $\mathbf{y' = y + sh_y \cdot x}$

  In matrix representation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Given $sh_y = 2$, relative to the y-axis ($x = 0$), and a square with coordinates (0,0), (0,1), (1,1), (1,0).

Y-direction Shear: refer to $x=x_{ref}$

- y-direction shear relative to the line $x = x_{ref}$

$$\mathbf{x' = x}$$

$$\mathbf{y' = sh_y(x - x_{ref}) + y}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

(7-61)

Example: *y*-direction shear relative to the line $x_{ref} = -1$

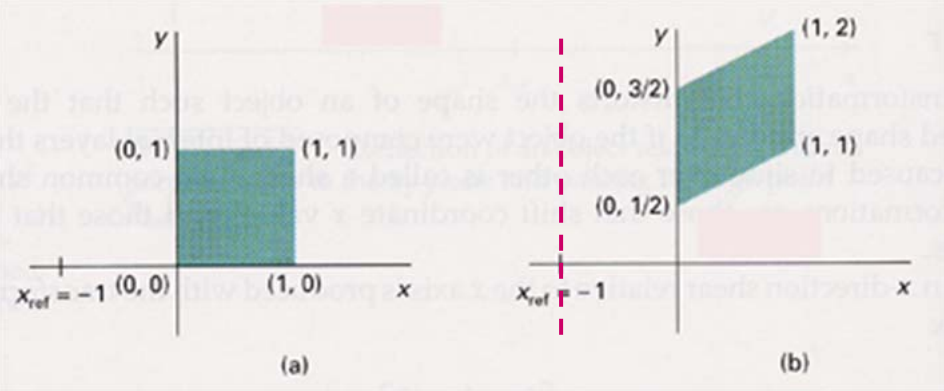


FIGURE 7-25 A unit square (a) is turned into a shifted parallelogram (b) with parameter values $sh_y = 0.5$ and $x_{ref} = -1$ in the y-direction shearing transformation 7-61.

Raster Methods for Geometric Transformations

- An alternative method for performing 2D transformations
  - To use raster operations to directly manipulating pixel values in the frame buffer
    1. Block Transfer (bitblt / pixblt)
       - Block transfer (bitblt (bit-block transfer), pixblt): moving a block of pixel values from one position to another.

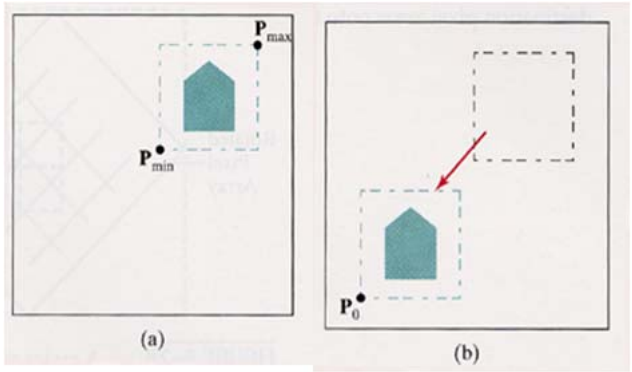


FIGURE 7-26 Translating an object from screen position (a) to the destination position shown in (b) by moving a rectangular block of pixel values. Coordinate positions P_{min} and P_{max} specify the limits of the rectangular block to be moved, and P_0 is the destination reference position.

Raster Methods for Geometric Transformations

Rotations

- Rotations in 90 or 180 degrees are easily accomplished by rearranging the elements of a pixel array
  - 90°-counterclockwise rotation by reversing the pixel values in each row of the array;
  - 180°-counterclockwise rotation by reversing the order of the elements in each row, then reversing the order of the rows.

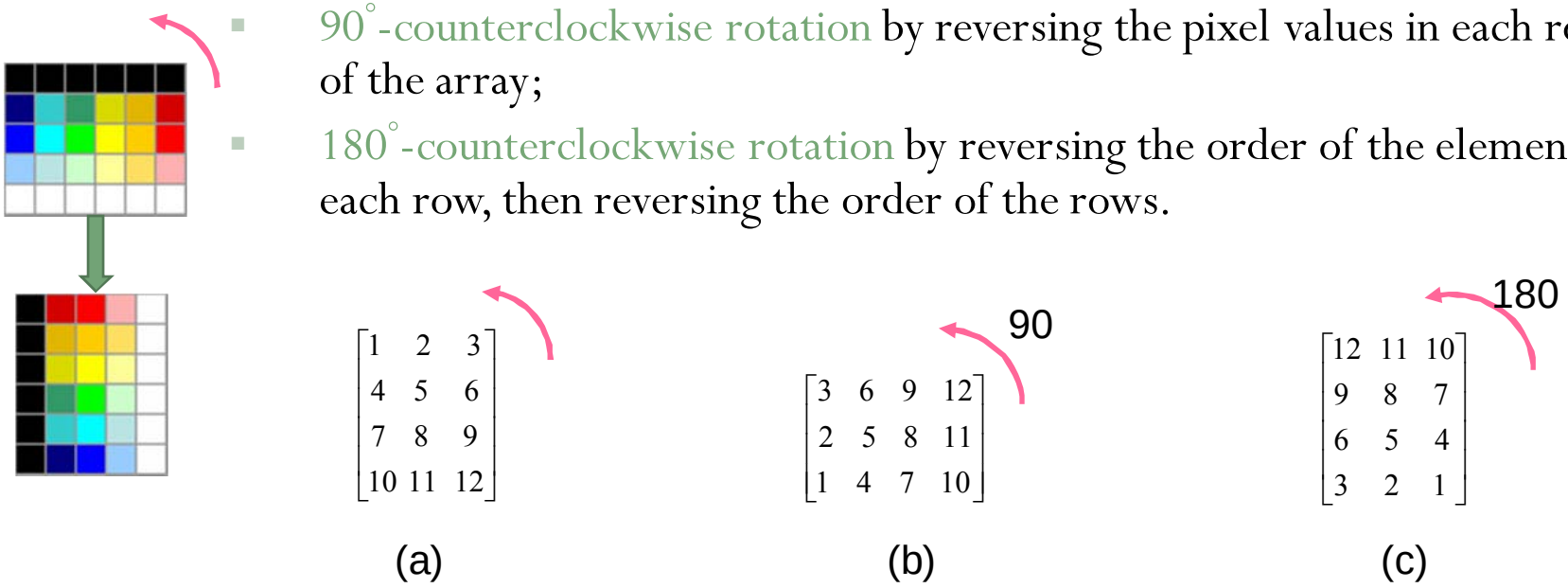


FIGURE 7-27 Rotating an array of pixel values. The original array is shown in (a), the positions of the array elements after a 90° counterclockwise rotation are shown in (b), and the positions of the array elements after a 180° rotation are shown in (c).

OpenGL Raster Transformations

1) Translation/Copy

A translation of a rectangular array of pixel-color values from one buffer to another can be accomplished in OpenGL as a copy operation

glCopyPixels (xmin, ymin, width, height, GL_COLOR);

-- **GL_COLOR**: buffer type to be copied
(others: **GL_DEPTH, GL_STENCIL**)

-- Destination: Current Raster Position.

The translation can be
- carried out on any OpenGL buffers for refreshing;
- used between different buffers.
source buffer is chosen with **glReadBuffer**;
destination buffer is selected with **glDrawBuffer**.

OpenGL Raster Transformations

2) Rotation a block of pixel-color values in 90 degrees

- Read a block of pixel-color values from a buffer and store in an array by **glReadPixels**

  glReadPixels (xmin, ymin, width, height, GL_RGB, GL_UNSIGNED_BYTE, colorArray);

  -- The color of block read into colorArray[ ] with data format **GL_RGB**, & data type **GL_UNSIGNED_BYTE**.

- Rearrange the rows and columns of the array;
- Placing the rotated array back in the buffer.

glDrawPixels (width, height, GL_RGB, GL_UNSIGNED_BYTE, colorArray);

-- The block is written by colorArray[ ] to Current Raster Position.

OpenGL Raster Transformations

3) Scaling a block of pixel-color values

- Firstly, specify the scaling factors by **glPixelZoom**

  glPixelZoom (sx, sy);

  -- Zoom referred to Current Raster Position;

  -- **sx, sy**: nonzero floating-point values;

  -- > 1.0 positive values: increase the size of an element in the source array;

  -- < 1.0 positive values: decrease element size;

  -- negative value: produce a reflection and scaling the array elements.

- Then, write back (either **glCopyPixels** or **glDrawPixels**).

- End of Part I&II -