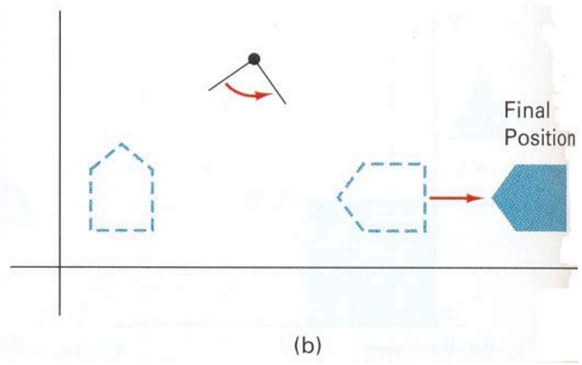
Universidade de Aveiro
Departamento de Electrónica,
Telecomunicações e Informática

2D Transformations



Computação Visual

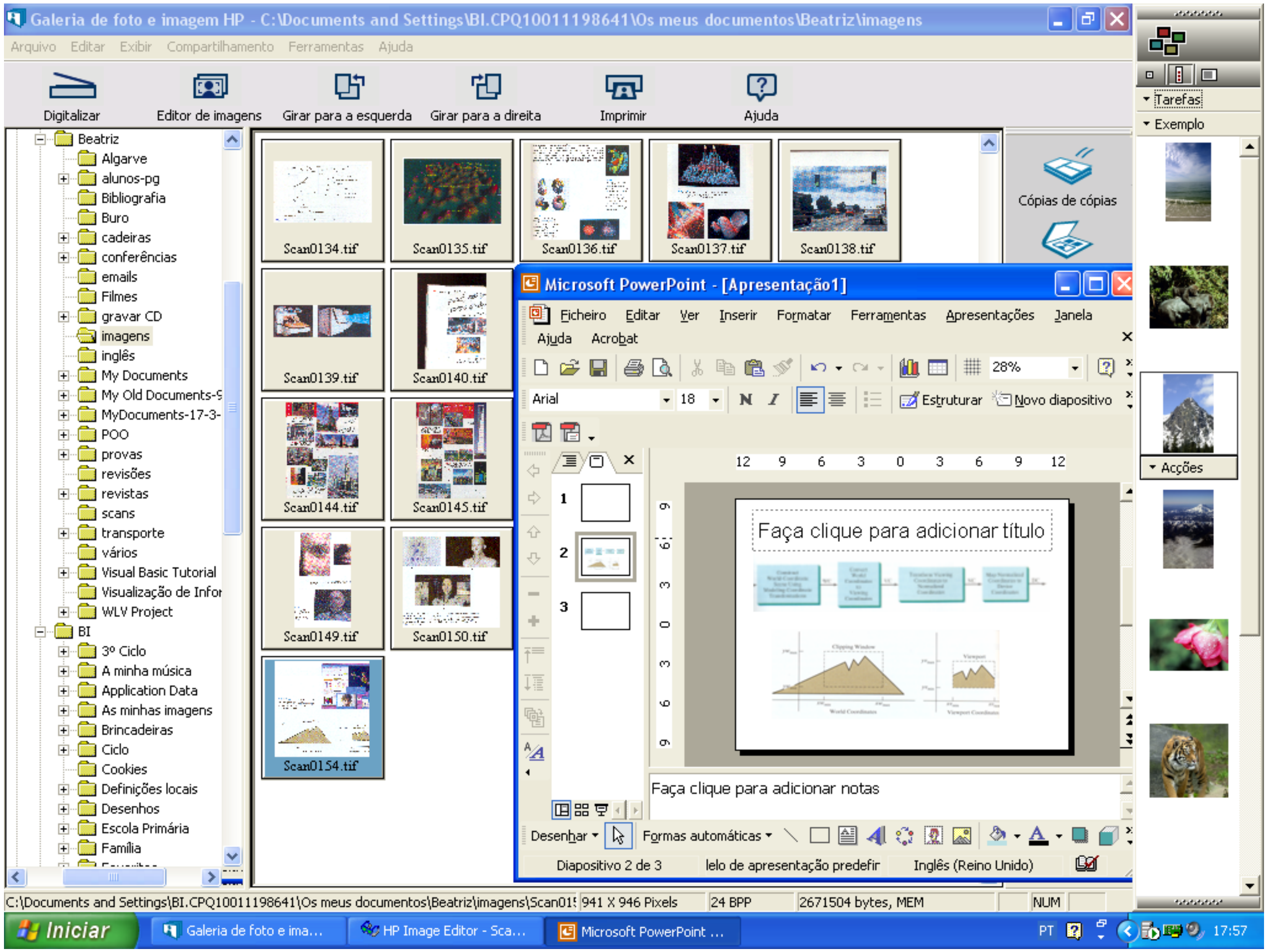
Beatriz Sousa Santos, Joaquim Madeira

Overview

- Recap – 2D visualization pipeline
- 2D Transformations
- Translation / Rotation / Scaling
- Homogeneous Coordinates
- Concatenating Transformations
- Other Transformations: Simmetry / Shearing
- Application Examples

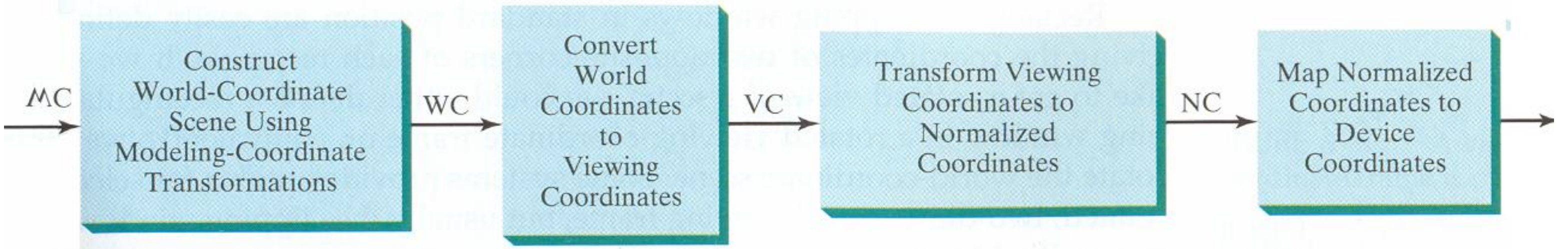
2D VISUALIZATION PIPELINE

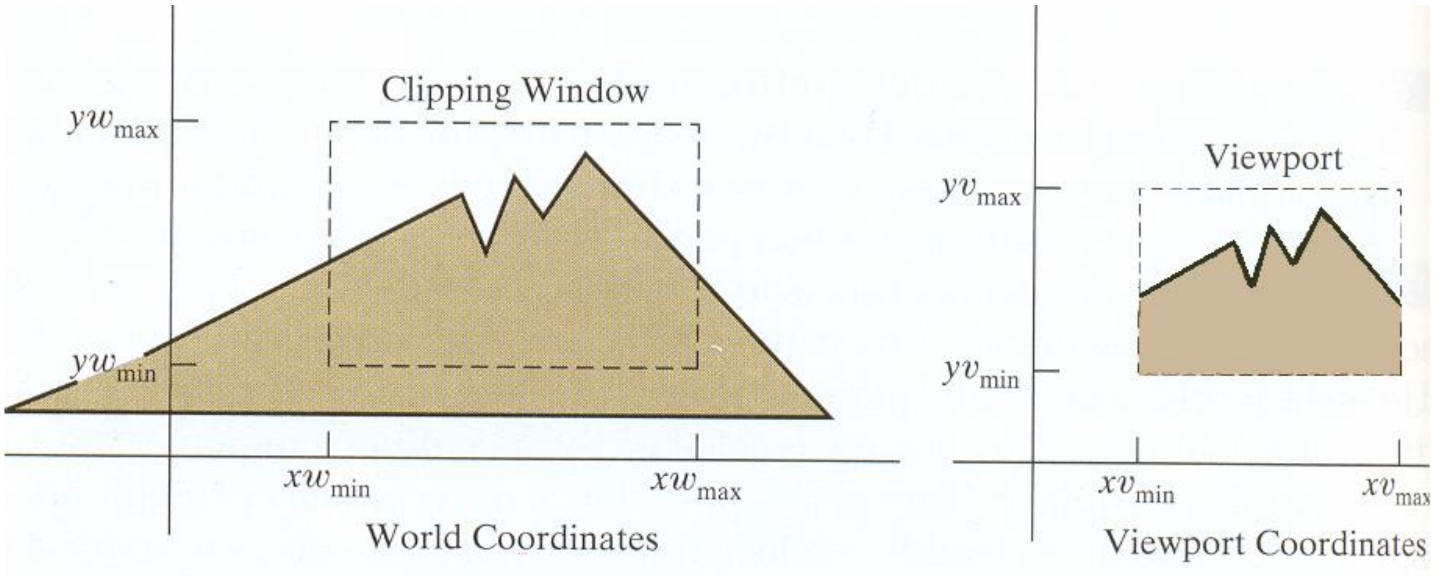
2D Visualization



Use 2D transformations to show different scenes (or parts of scenes) in various display areas

2D Visualization





2D TRANSFORMATIONS

2D Transformations

- Position, orientation and scaling for objects in XOY
- Basic transformations
  - Translation / Displacement
  - Rotation relative to the coordinates' origin
  - Scaling relative to the coordinates' origin
- Representation using matrices
  - Homogeneous coordinates
- Complex transformations
  - Decompose into a sequence of basic transformations

Basic 2D transformations

p = (x, y) → *original, given point* $\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}$

p' = (x', y') → *transformed point* $\mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$

Column vector representation

- The basic transformations are:

  - Translation / Displacement

  - Scaling

  - Rotation

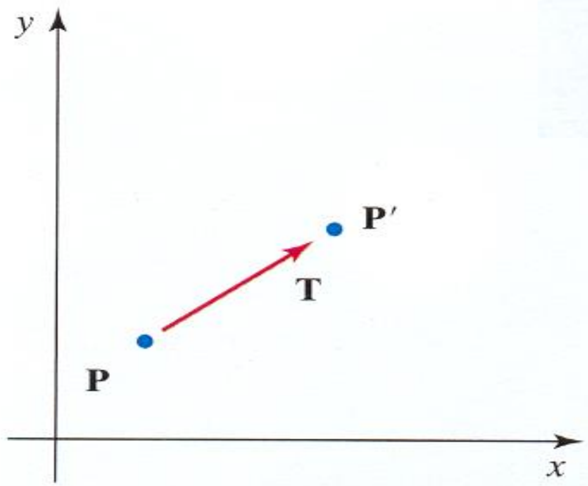
Some older books and graphics APIs represent each point as a row vector and not as a column vector: **P** = [x y]

2D TRANSLATION

Translation

- To translate a point we need the displacement values in $\boldsymbol{x}$ and $\boldsymbol{y}$

$$x' = x + t_x, \qquad y' = y + t_y$$



$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

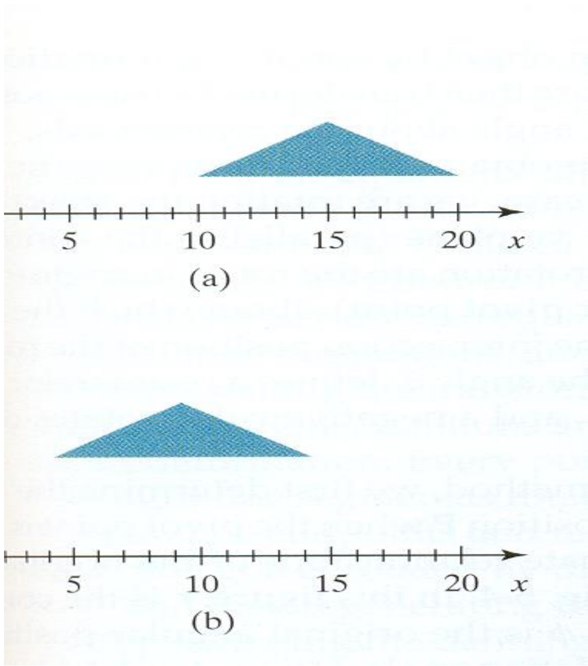
$$\mathbf{P}' = \mathbf{P} + \mathbf{T}$$

Translation

- Each object is displaced without any deformation:
  it is a rigid-body transformation

- To displace a straight-line segment, apply the transformation to the two end-points and draw the resulting line segment.

- To displace a polygon, apply the transformation to the polygon's vertices.



2D ROTATION

Rotation

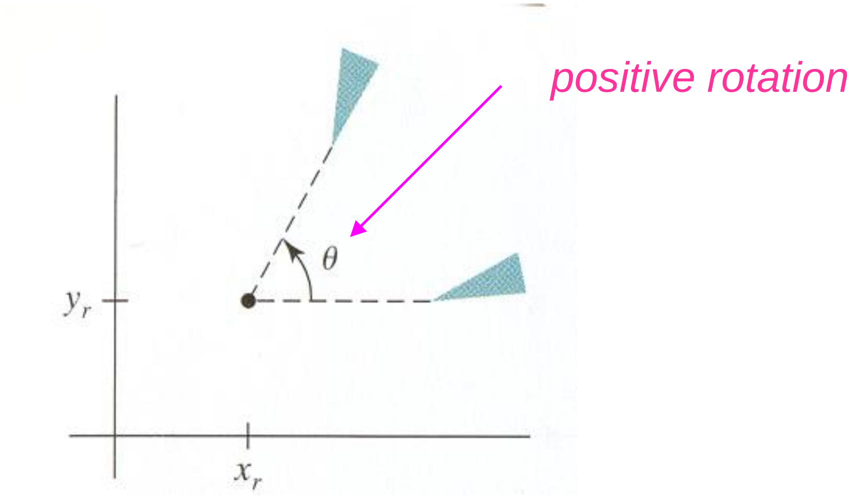
- To apply a rotation we need:

  - a point: the center of rotation

    (x_r, y_r)

    (intersection point between a perpendicular rotation axis and XOY)

  - a rotation angle θ (positive, if counter-clockwise - CCW)



Rotation around the origin of the coordinates' system

- It is easier to determine the transformation representing a rotation around (0,0):

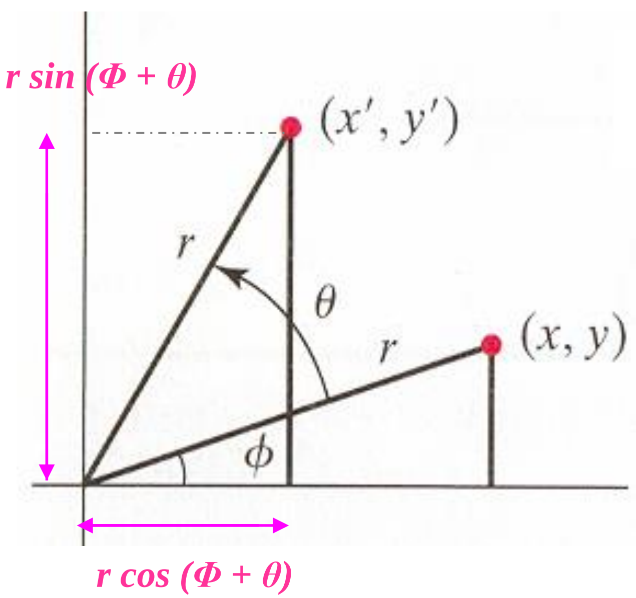
$$x' = r\cos(\Phi + \theta) = r\cos\Phi\cos\theta - r\sin\Phi\sin\theta$$

$$y' = r\sin(\Phi + \theta) = r\cos\Phi\sin\theta + r\sin\Phi\cos\theta$$

Original point coordinates in polar coordinates:

$$x = r\cos\Phi$$

$$y = r\sin\Phi$$



Replacing in the above equations, we get the desired result:

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

Rotation around the origin of the coordinates' system

$$x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi + \theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta$$

$$x = r\cos\phi, \qquad y = r\sin\phi$$

If a point is represented by a row vector, the multiplication order is changed and the correponding rotation matrix is the transpose: P' = P . R^T

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$$ com $$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Matrix for rotation around the origin, with angle $\boldsymbol{\theta}$ em torno da origem

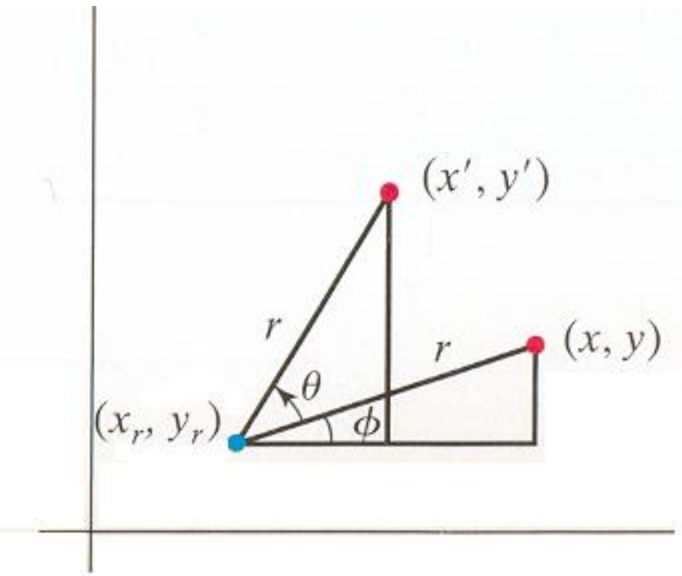
Rotation around an arbitrary point

- Using the figure, the rotation equations are obtained as:

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

An alternative method is to consider this transformation as being made up of a sequence elementary transformations – wait for those slides.



- Rotations are also rigid-body transformations
- To rotate a straight-line segment, transform ist end-points and draw the line segment
- To rotate a polygon, transform its vertices

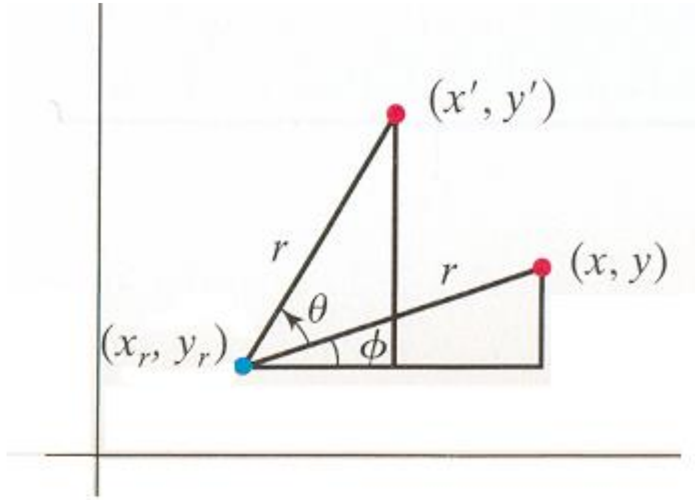
Doing the maths for the x' coordinate

$x' = r \cos(\theta + \Phi) + x_r$ $\qquad$ $x = r \cos \Phi + x_r$

$y' = r\ sen(\theta + \Phi) + y_r$ $\qquad$ $y = r\ sen\ \Phi + y_r$

$x' = r \cos\theta \cos\Phi - r\ sen\ \theta\ sen\ \Phi + x_r$

$y' = r \cos\theta\ sen\ \Phi + r\ sen\ \theta \cos\Phi + y_r$



$x' = (x - x_r) \cos\theta - (y - y_r)\ sen\ \theta + x_r$

$x' = x_r + (x - x_r) \cos\theta - (y - y_r) \sin\theta$

2D SCALING

Scaling relative to the coordinates' origin

- The scaling transformation is applied to change the size of an object: s_x and s_y are the scaling factors.

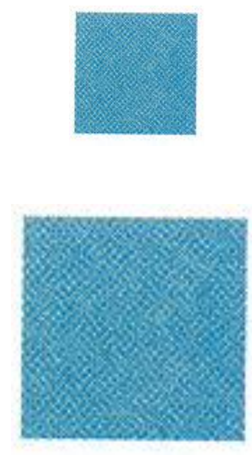
$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

transformation matrix

$$P' = S \cdot P$$

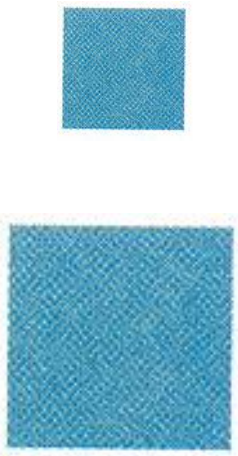


Obtaining a larger square through a scaling transformation, s_x=2, s_y=2

Scaling

- Scaling factors are positive: $s > 0$

Obtaining a larger square through a scaling transformation, s_x=2, s_y=2

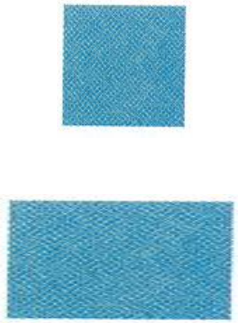


$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

$s_x = s_y \longrightarrow$ uniform scaling

$s_x \neq s_y \longrightarrow$ non-uniform scaling



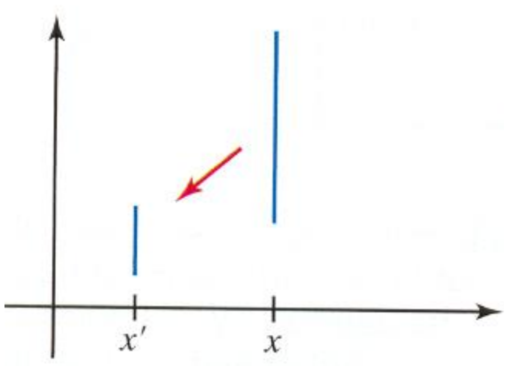
Transforming a square into a rectangle: the scaling has s_x=2, s_y=1

Scaling

- The scaled objects are repositioned if not originally centered on the coordinates' origin:

s < 1 → it will be closer to the origin

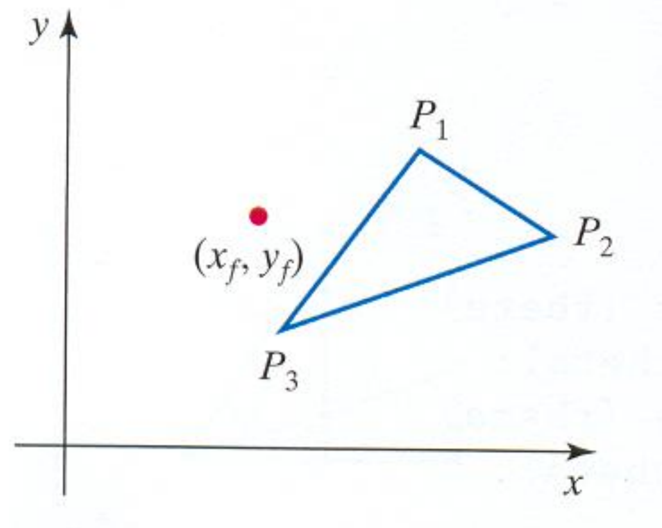
s > 1 → it will be farther from the origin



A straight-line segment becomes shorter and closer to the origin through the scaling $\boldsymbol{s_x} = \mathbf{s}_y = 0{,}5$

Scaling relative to a fixed point

- We can control the positon of the object by choosing a fixed point (x_f, y_f) that remains unchanged:



$$x' - x_f = (x - x_f) \cdot s_x$$

$$y' - y_f = (y - y_f) \cdot s_y$$

$$x' = x \cdot s_x + x_f (1 - s_x)$$

$$y' = y \cdot s_y + y_f (1 - s_y)$$

constant for every point

This scaling transformation, as well as the arbitrary rotation, can be applied with just one matrix multiplication for every point – wait for those slides.

HOMOGENEOUS COORDINATES

Homogeneous coordinates

- Most graphical applications apply sequences of transformations

- For instance:

  - the visualization transformation corresponds to sequences of translations and rotations to display a given scene

  - an animation might require that an object be displaced and rotated between consecutive frames

- To carry out sequences of transformations in an efficient way, each transformation is represented as a matrix using homogeneous coordinates

- The three basic transformations can be represented generally as::

$$\mathbf{P' = M_1 \cdot P + M_2}$$

M_1 is a 2x2 matrix

M_2 is a column vector, representing the displacement vector

- A more efficient representation uses just one matrix which

  - can represent all the transformations in a sequence

  - is applied just once to every point

- Such a representation uses homogeneous coordinates

- A single 3x3 matrix represents all multiplicative and additive terms

- All transformations are represented by a 3x3 matrix

- The third matrix column represents the displacement (additive) factors

- Every point is now represented by three coordinates:

$$(x, y) \rightarrow (x_h, y_h, h), \quad h \neq 0$$

$$x = x_h / h \qquad y = y_h / h$$

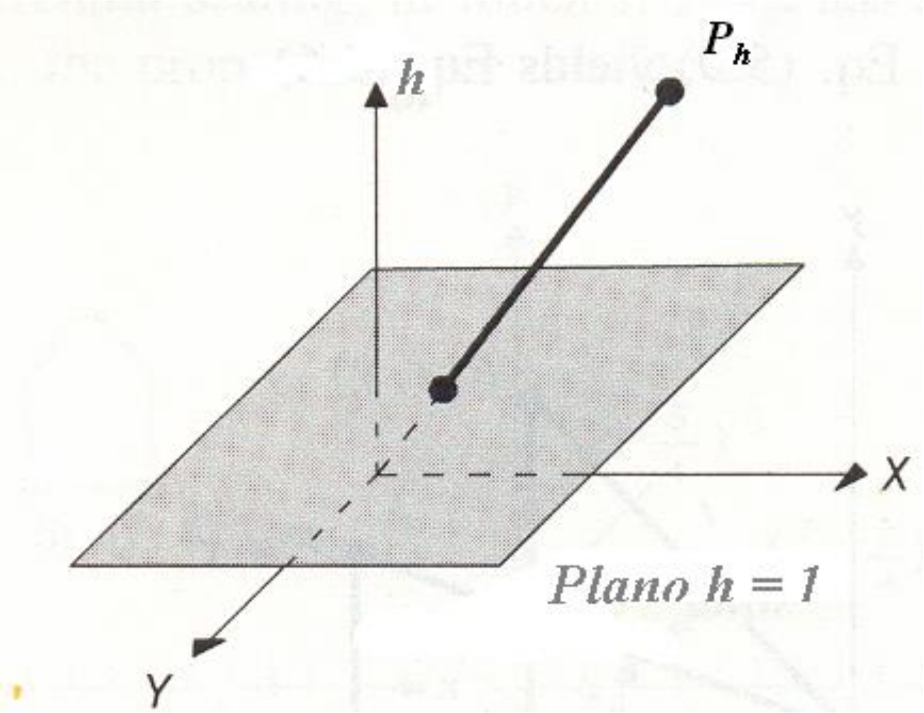
$$(x.h, y.h, h)$$

- An easy choice is:

$h = 1$

- Which implies:

$(x, y) \rightarrow (x, y, 1)$



- There is an indefinite number of points P_h in the 3D homogeneous space that correspond to a single Euclidean point (x,y)

REPRESENTING TRANSFORAMTIONS USING HOMOGENEOUS COORDINATES

2D transformations using homogeneous coordinates

- When using homogeneous coordinates, all transformations are carried out by matrix multiplication

- 2D translation:

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

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$$

- 2D rotation:

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

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$$

- 2D scaling:

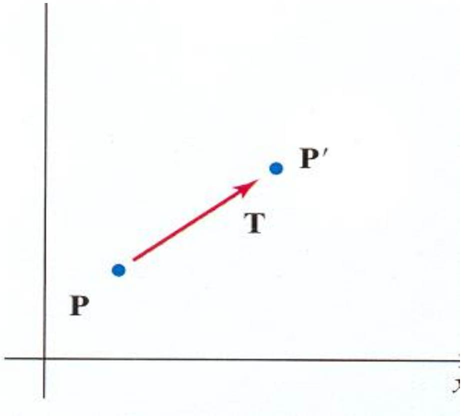
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P}$$

Inverse transformations

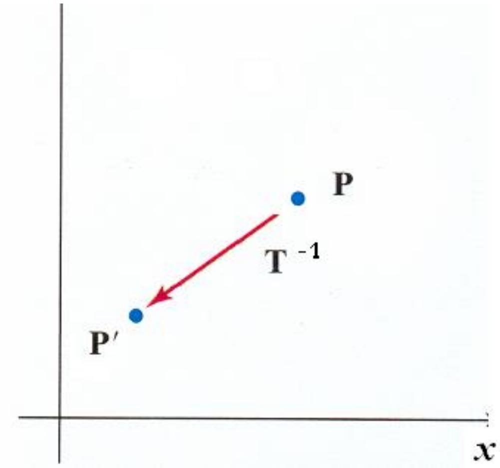
- The inverse of a given translation:

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



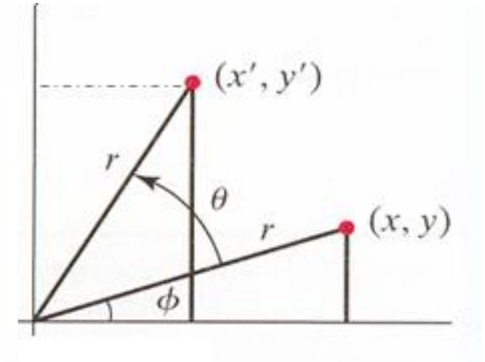
is also a translation with symmetrical parameters in x and y:

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



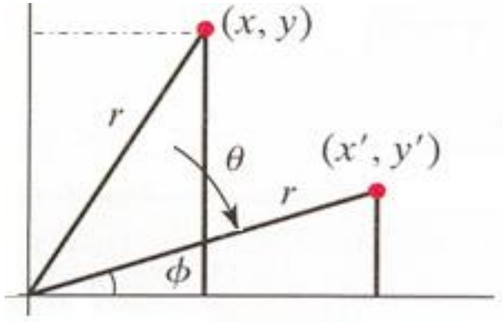
- The inverse rotation is obtained by using the symmetrical rotation angle:

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



- The inverse rotation matrix is the transpose of the original rotation matrix:

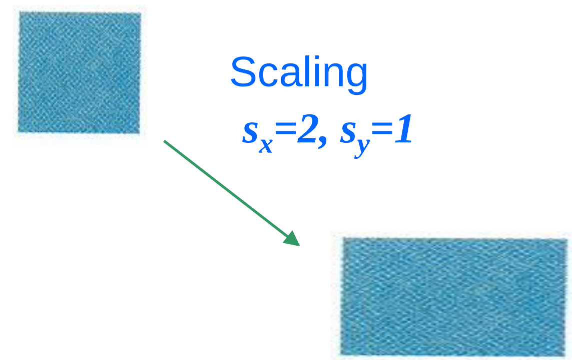
$$(\mathbf{R}^{-1} = \mathbf{R}^{T})$$

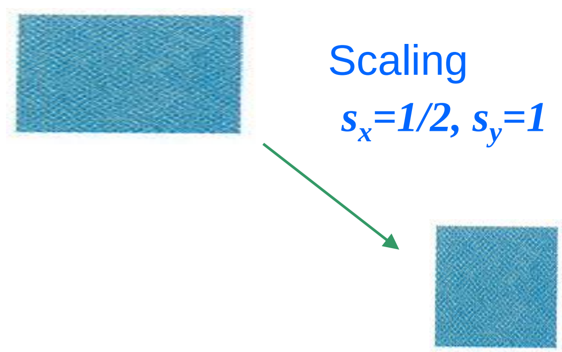


- Only the sinus terms are affected by changing the sign of the rotation angle

- For the inverse scaling matrix, replace the each scaling factor s by 1/s:

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





- The product of any matrix M, representing a given transformation, by the matrix representing its inverse transformation results in the identity matrix:

$$
\mathbf{M} \cdot \mathbf{M}^{-1} = \mathbf{I}
$$

CONCATENATION OF TRANSFORMATIONS

Concatenation of transformations

- With the matricial representation, we can compute the matrix representing a sequence of transformations by multiplying the matrices representing the individual transformations, in the appropriate order.

- The product of transformation matrices represents the concatenation or composition of transformations.

- The concatenation of two transformations is represented as:

$$P' = \underbrace{M_2}_{\text{second transformation to be applied}} . \underbrace{M_1}_{\text{first transformation to be applied}} . P$$
$$= M . P$$

- The coordinates of the transformed point P' are computed with a single matrix multiplication

Concatenation of two translations

$$\begin{aligned}\mathbf{P}' &= \mathbf{T}(t_{2x}, t_{2y}) \cdot \{\mathbf{T}(t_{1x}, t_{1y}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y})\} \cdot \mathbf{P}\end{aligned}$$

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

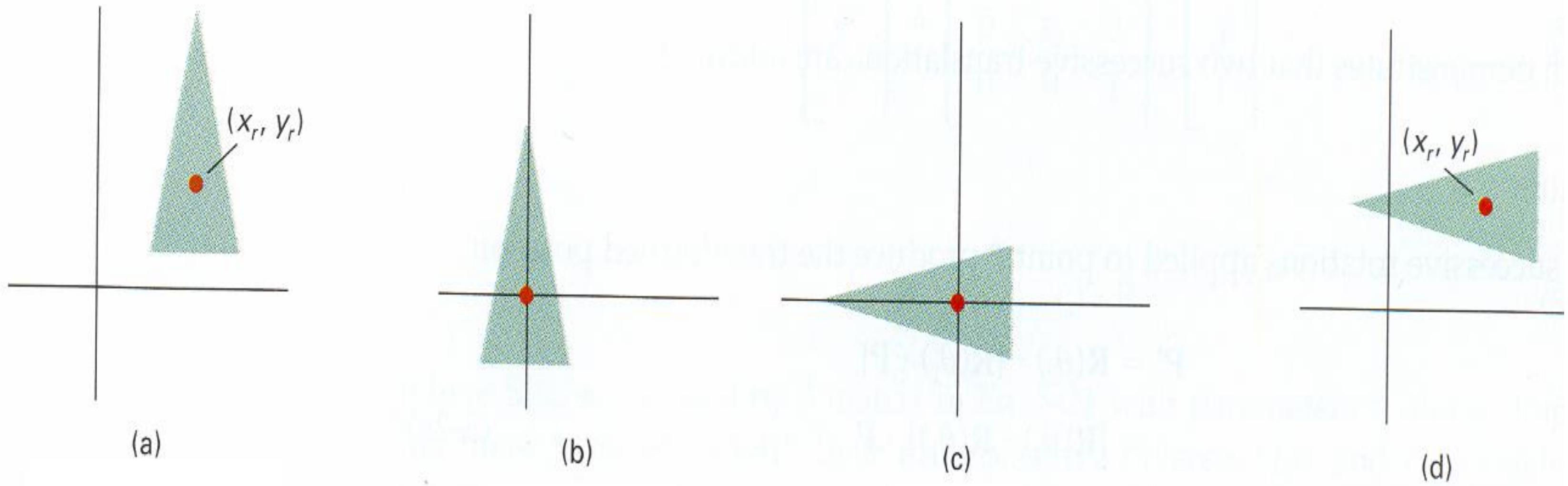
$$\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

Concatenation of two scalings

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S}(s_{2x}, s_{2y}) \cdot \mathbf{S}(s_{1x}, s_{1y}) = \mathbf{S}(s_{1x} \cdot s_{2x}, \quad s_{1y} \cdot s_{2y})$$

Rotation around an arbitrary point (x_r, y_r)



Original position of the triangle and point (x_r, y_r)

1- A translation moves point (x_r, y_r) to the origin

2- Rotation around the origin

3- Inverse translation moves the rotation center back to (x_r, y_r)

Rotation around an arbitrary point (x_r, y_r)

- Apply:

1 - a translation so that the arbitrary point moves to the origin

2 - a rotation around the origin

3 - the inverse translation to move the rotation center back to its original position

3- Inverse translation

2- θ degrees rotation

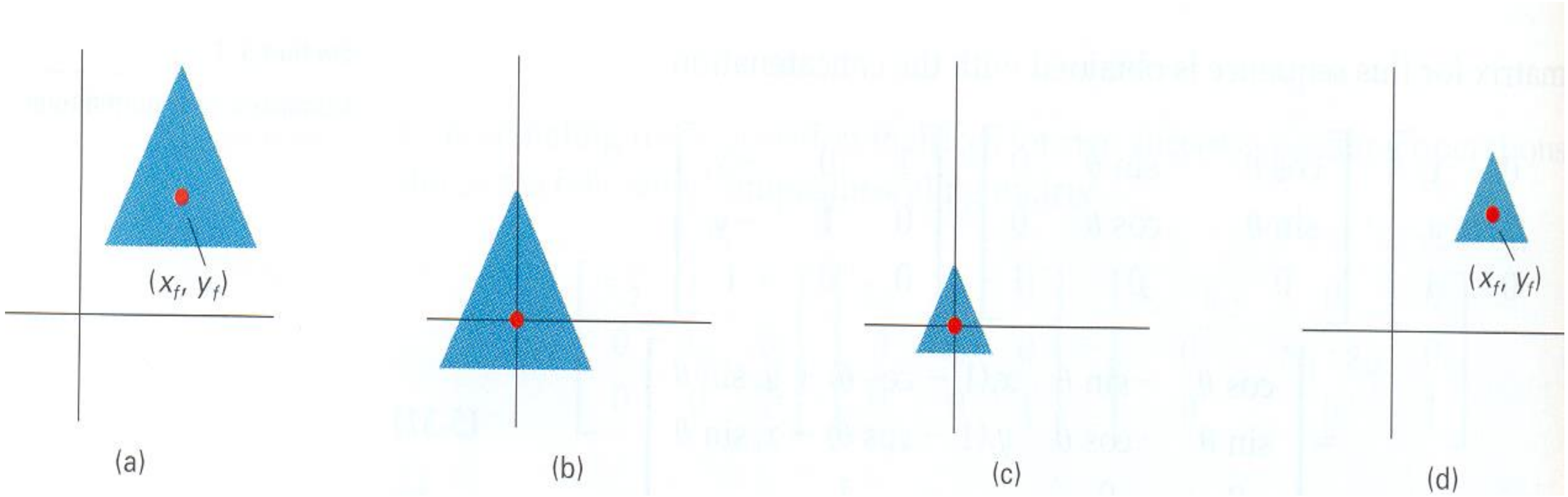
1- Moving to the origin

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

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

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

Scaling relative to a fixed point (x_f, y_f)



Original position of the triangle and point (x_f, y_f)

1- Move point (x_f, y_f) to the origin

2- Scaling

3- Inverse translation back to (x_f, y_r)

Concatenation of transformations - Properties

- Matrix multiplication is associative

- Given any three matrices $\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{M_3}$, their product can be computed multiplying first $\mathbf{M_3}$ by $\mathbf{M_2}$, or multiplying first $\mathbf{M_2}$ by $\mathbf{M_1}$

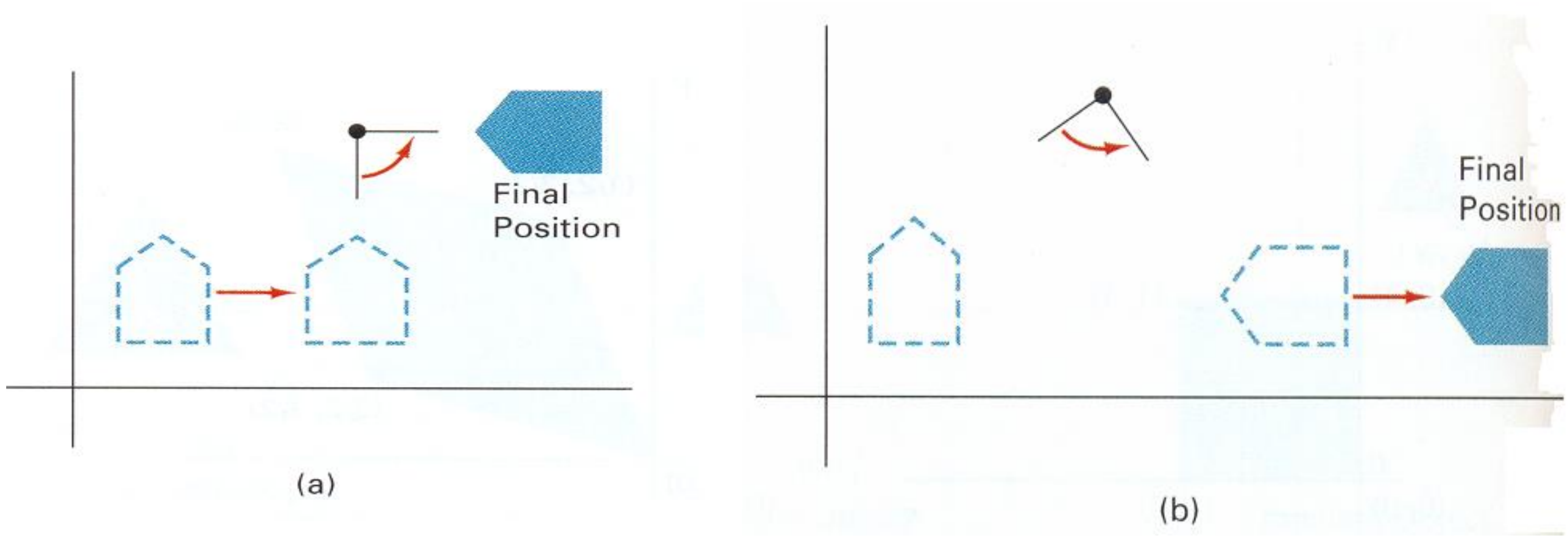
$$\mathbf{M_3 \,.\, M_2 \,.\, M_1 = (M_3 \,.\, M_2) \,.\, M_1 = M_3 \,.\, (M_2 \,.\, M_1)}$$

- In general, matrix multiplication is not commutative:

$$\mathbf{M2 \,.\, M1 \neq M1 \,.\, M2}$$

- For instance, to apply a rotation and a translation to an object, care is needed to carry out the multiplication in the appropriate order

• For instance, to apply a rotation and a translation to an object, care is needed to carry out the multiplication in the appropriate order



Changing the order of a transformation sequence might affect the final result. In *a)* the translation is applied first, followed by the 90º CCW rotation. In *b)* the rotation is applied first, followed by the translation.

- In some particular cases, matrix multiplication is commutative.
- E.g., two successive rotations, or two successive scalings, or two successive translations.

Concatenation of transformations – Efficiency

- A 2D transformation representing a concatenation of transformations (translations, rotations, scalings) can be represented as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} rs_{xx} & rs_{xy} & trs_x \\ rs_{yx} & rs_{yy} & trs_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

multiplicative terms corresponding to rotations and scalings

terms corresponding to displacement distances, or rotation center or fixed-point for scaling

- Example: to apply a scaling followed by a rotation, both relative to an object's center ***(x_c, y_c)*** , followed by a translation, we gete depois sofrer uma translação, temos:

$$\mathbf{T}(t_x, t_y) \cdot \mathbf{R}(x_c, y_c, \theta) \cdot \mathbf{S}(x_c, y_c, s_x, s_y)$$

$$= \begin{bmatrix} s_x \cos\theta & -s_y \sin\theta & x_c(1 - s_x \cos\theta) + y_c s_y \sin\theta + t_x \\ s_x \sin\theta & s_y \cos\theta & y_c(1 - s_y \cos\theta) - x_c s_x \sin\theta + t_y \\ 0 & 0 & 1 \end{bmatrix}$$

- The transformed coordinates are given by:

$$x' = x \cdot rs_{xx} + y \cdot rs_{xy} + trs_x, \qquad y' = x \cdot rs_{yx} + y \cdot rs_{yy} + trs_y$$

- For any sequence of transformations, represented by one global transformation matrix, we need only:

  - 4 multiplications

  - 4 additions

- If each transformation was independently apllied, the number of multiplications and additions would be larger

- Use only the single, global transformation matrix resulting from the concatenation of the individual transformations

- The elements of that matrix have only to be computed once !

SIMMETRY & SHEARING

Additional transformations

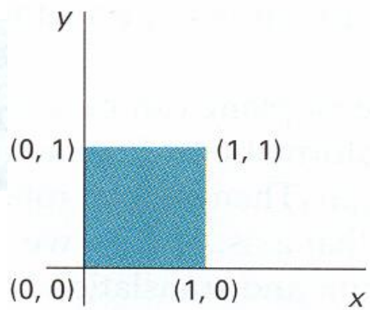
- In addition to the basic transformations (displacement, rotation and scaling) some graphics APIs offer other useful transformations, such as:

  - symmetry
  - *shearing*

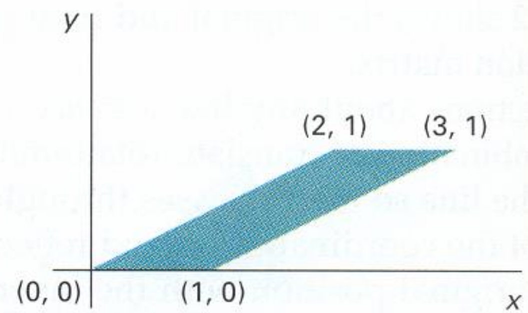
- *Shearing* in the XX' direction:

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

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



unit square



polygon resulting from the XX'-shearing

Symmetry

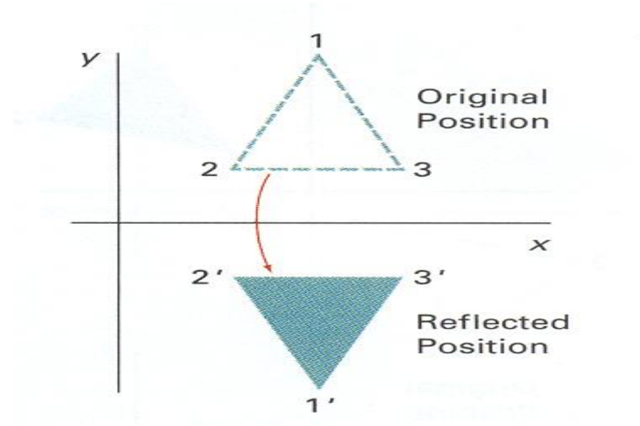
- The symmetry is a transformation producing a "mirror image" of the transformed object

- To carry out a symmetry relative to the XX' axis ***(y=0)*** multiply the Y-ccordinates by -1

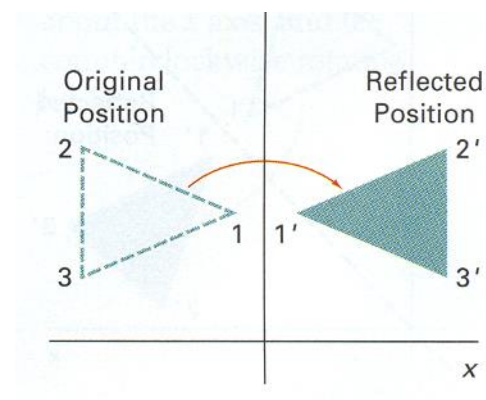
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- To carry out a symmetry relative to the YY' axis ***(x=0)*** multiply the X-ccordinates by -1

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Symmetry relative to the XX' axis



Symmetry relative to the YY' axis

Questions

- How to apply a bi-directional *shearing* ?
- And symmetries regarding axes that are parallel to the coordinate axes ?
- Or regarding simmetry axes containing (0,0) ?
  - Even- or odd-quadrant bissectors
  - With any slope
- Or even around any symmetry axis ?

MAIN IDEAS FROM TODAY

2D Transformations

- Position, orientation and scaling for objects in XOY
- Basic transformations
  - Translation / Displacement
  - Rotation relative to the coordinates' origin
  - Scaling relative to the coordinates' origin
- Representation using matrices
  - Homogeneous coordinates
- Complex transformations
  - Decompose into a sequence of basic transformations

TASKS

Rotating a rectangle

- Rectangle is defined by vertices

  $A=(0,0)$, $B=(2,0)$, $C=(2,4)$ and $D=(0,4)$
- Rotate the rectangle around its center
- Rotation angle is –45 degrees

- Decompose into basic transformations
- Multiply them to get the global transformation matrix
- Compute the coordinates of the transformed vertices

Additional problems (see PDF)

1- Given the square, defined by the vertices (2, 2), (3, 2), (3, 3) and (2, 3), it is to be rotated around its center by an angle of 90 degrees.

2- Given the triangle, defined by the vertices (2, 0), (4, 2) and (-1, 5), determine the triangle resulting from applying a symmetry transformation relative to the $y = x$ straight-line.

REFERENCES

References

- D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 3rd Ed., Addison-Wesley, 2004

- E. Angel and D. Shreiner, *Introduction to Computer Graphics,* 6th Ed., Pearson Education, 2012

- J. Foley et al., *Introduction to Computer Graphics*, Addison-Wesley, 1993