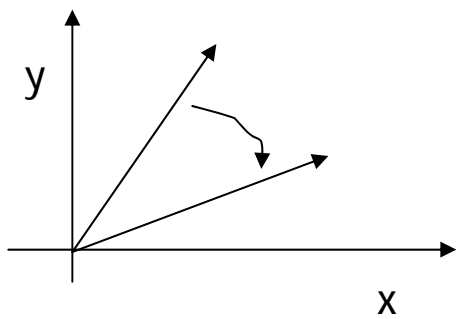
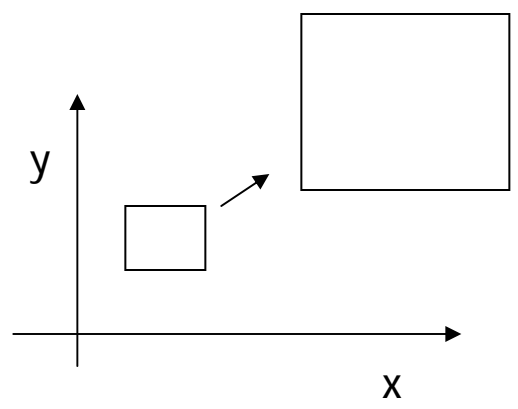
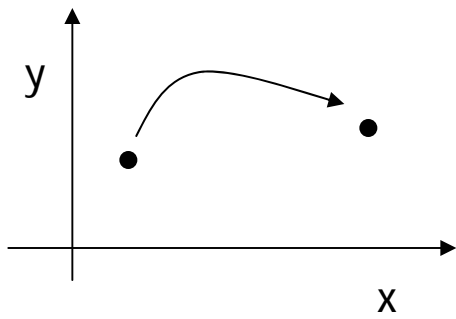


2D Transformations





2D Transformation

- Given a 2D object, transformation is to change the object's
 - Position (translation)
 - Size (scaling)
 - Orientation (rotation)
 - Shapes (shear)
- Apply a sequence of matrix multiplication to the object vertices



Point representation

- We can use a column vector (a 2x1 matrix) to represent a 2D point $\begin{vmatrix} x \\ y \end{vmatrix}$
- A general form of *linear* transformation can be written as:

$$x' = ax + by + c$$

$$\text{OR} \quad \begin{vmatrix} X' \\ Y' \\ 1 \end{vmatrix} = \begin{vmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

$$y' = dx + ey + f$$

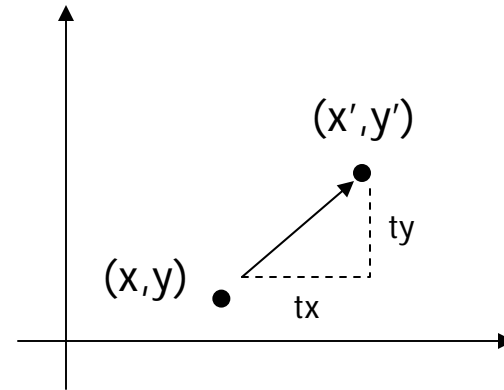
Translation

- Re-position a point along a straight line
- Given a point (x,y) , and the translation distance (tx,ty)

The new point: (x', y')

$$x' = x + tx$$

$$y' = y + ty$$

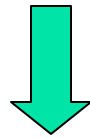


OR $P' = P + T$ where $P' = \begin{vmatrix} x' \\ y' \end{vmatrix}$ $p = \begin{vmatrix} x \\ y \end{vmatrix}$ $T = \begin{vmatrix} tx \\ ty \end{vmatrix}$



3x3 2D Translation Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix}$$



Use 3 x 1 vector

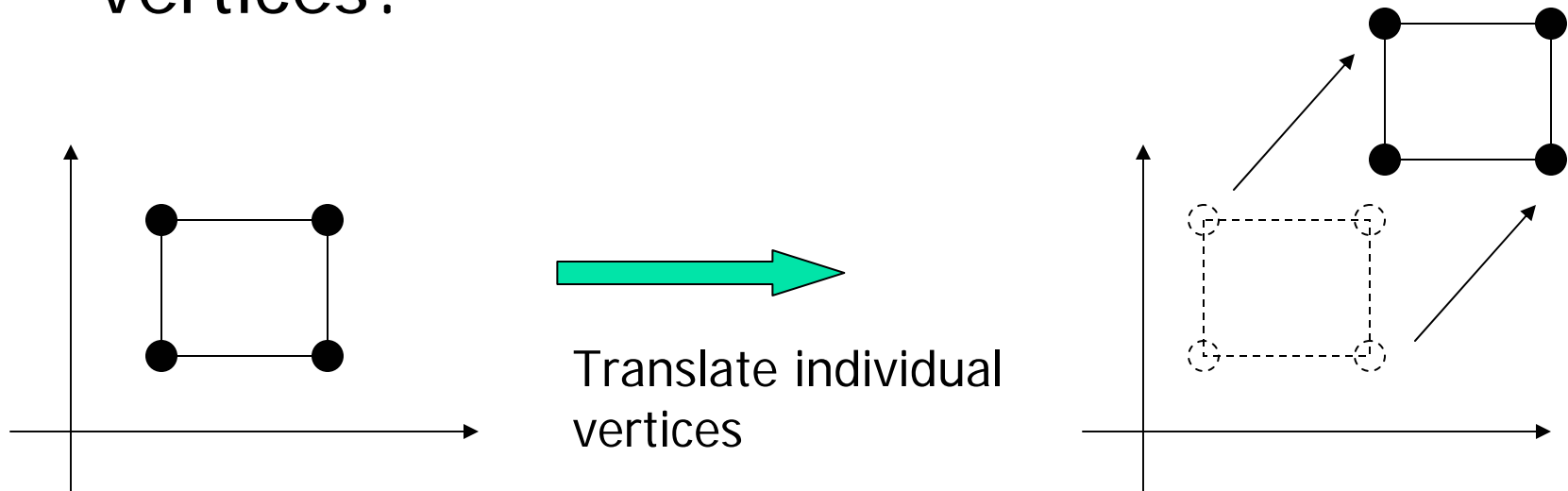
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Note that now it becomes a matrix-vector multiplication



Translation

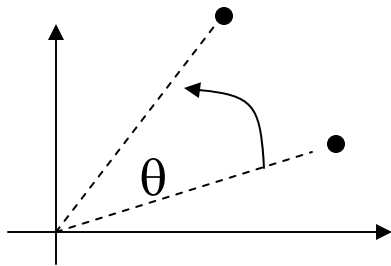
- How to translate an object with multiple vertices?



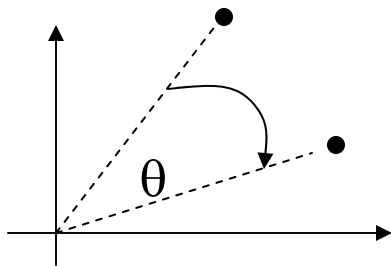


2D Rotation

- Default rotation center: Origin (0,0)



$\theta > 0$: Rotate counter clockwise



$\theta < 0$: Rotate clockwise

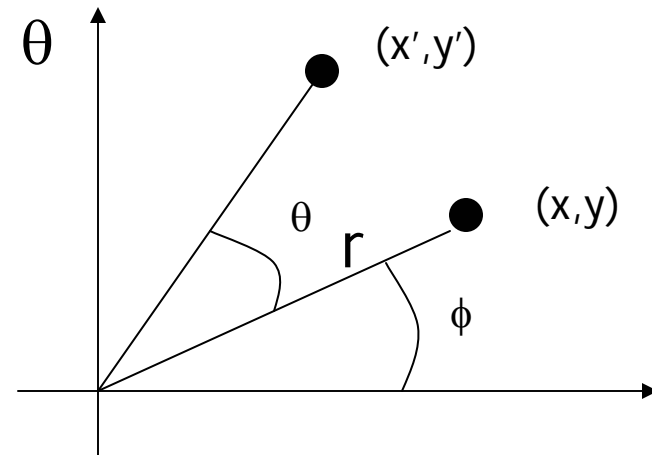


Rotation

(x, y) \rightarrow Rotate *about the origin* by θ

$\longrightarrow (x', y')$

How to compute (x', y') ?



$$x = r \cos (\phi) \quad y = r \sin (\phi)$$

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

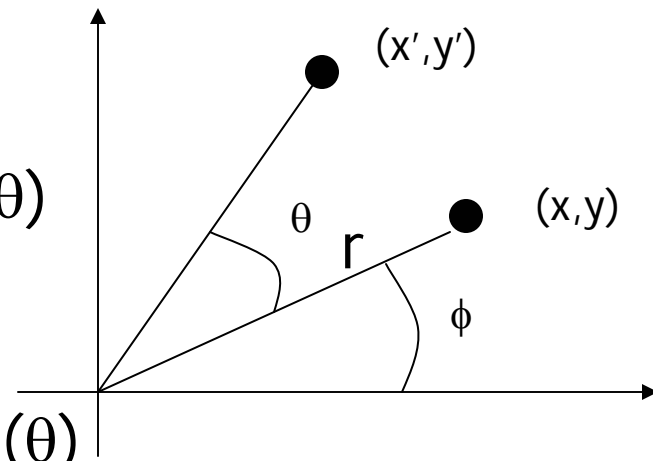
Rotation

$$x = r \cos(\phi) \quad y = r \sin(\phi)$$

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

$$\begin{aligned} x' &= r \cos(\phi + \theta) \\ &= r \cos(\phi) \cos(\theta) - r \sin(\phi) \sin(\theta) \\ &= x \cos(\theta) - y \sin(\theta) \end{aligned}$$

$$\begin{aligned} y' &= r \sin(\phi + \theta) \\ &= r \sin(\phi) \cos(\theta) + r \cos(\phi) \sin(\theta) \\ &= y \cos(\theta) + x \sin(\theta) \end{aligned}$$



Rotation

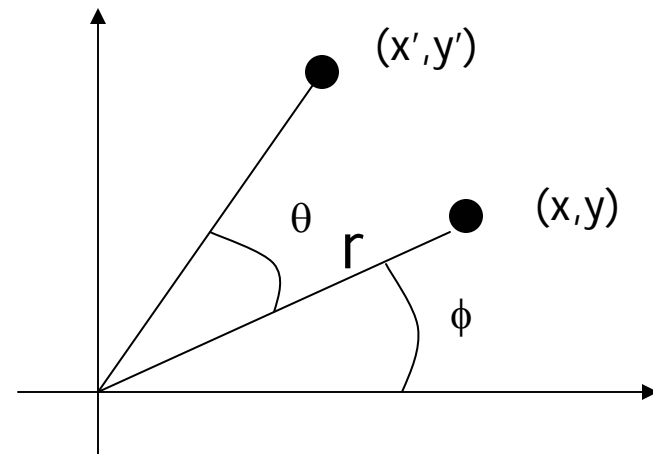
$$x' = x \cos(\theta) - y \sin(\theta)$$

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

Matrix form?

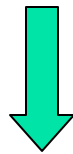
$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$

3 x 3?

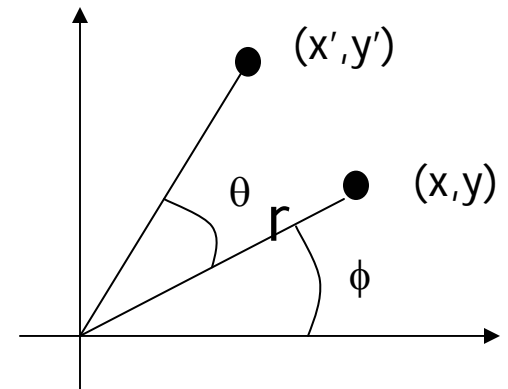


3x3 2D Rotation Matrix

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$



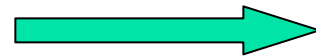
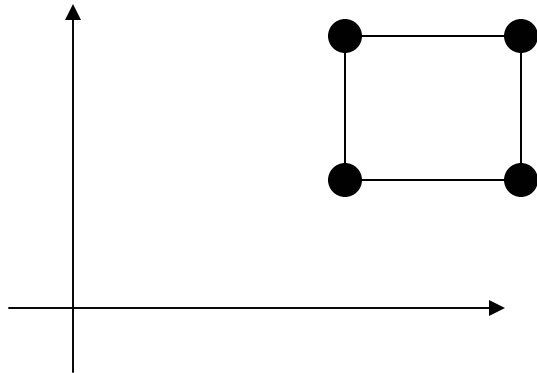
$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$



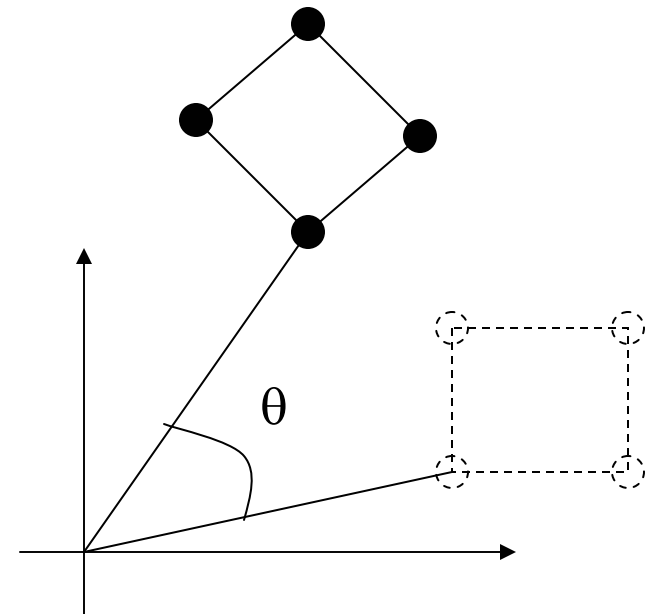


Rotation

- How to rotate an object with multiple vertices?



Rotate individual
Vertices





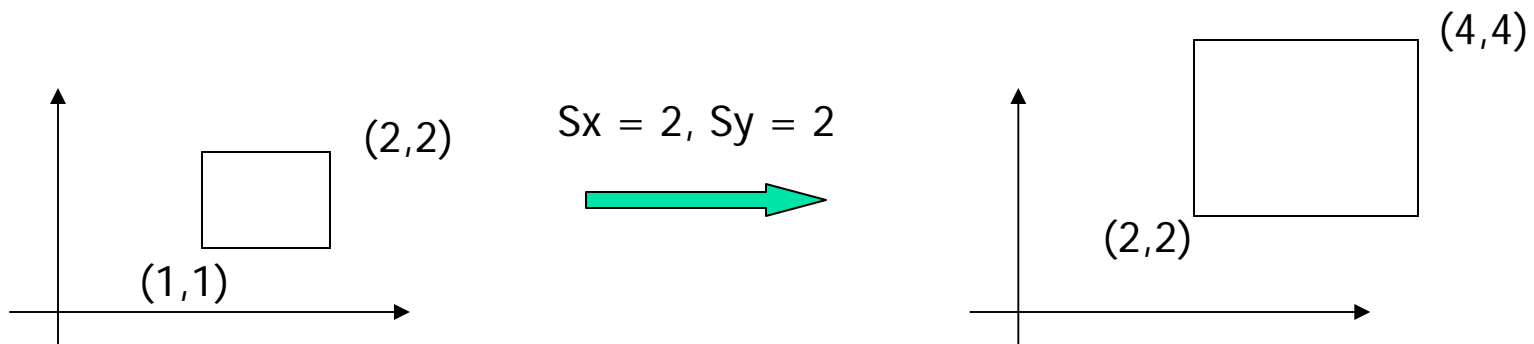
2D Scaling

Scale: Alter the size of an object by a scaling factor (S_x, S_y) , i.e.

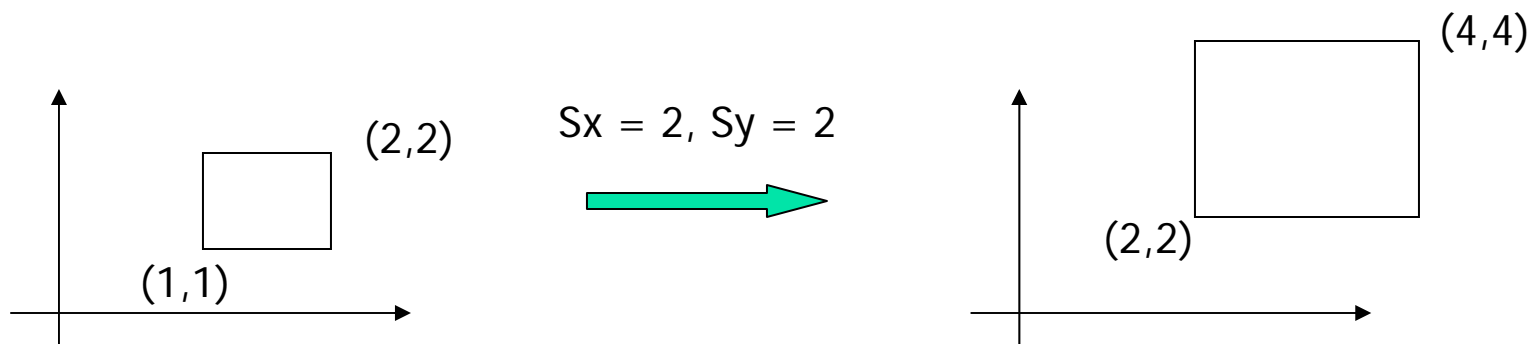
$$\begin{aligned}x' &= x \cdot S_x \\y' &= y \cdot S_y\end{aligned}$$



$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} S_x & 0 \\ 0 & S_y \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$



2D Scaling

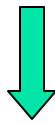


- Not only the object size is changed, it also moved!!
- Usually this is an undesirable effect
- We will discuss later (soon) how to fix it



3x3 2D Scaling Matrix

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} Sx & 0 \\ 0 & Sy \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix}$$



$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$



Put it all together

- Translation:
$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} tx \\ ty \end{vmatrix}$$

- Rotation:
$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix}$$

- Scaling:
$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} Sx & 0 \\ 0 & Sy \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix}$$



Or, 3x3 Matrix representations

- Translation:
$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

- Rotation:
$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

- Scaling:
$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

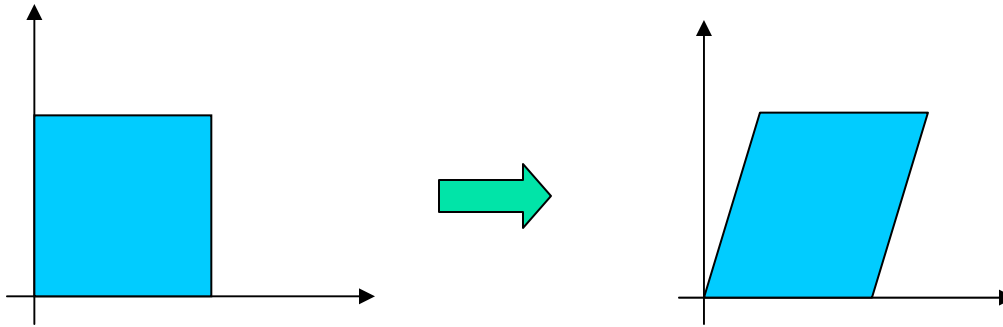
Why use 3x3 matrices?



Why use 3x3 matrices?

- So that we can perform all transformations using matrix/vector multiplications
- This allows us to *pre-multiply* all the matrices together
- The point (x,y) needs to be represented as $(x,y,1)$ -> this is called **Homogeneous coordinates!**

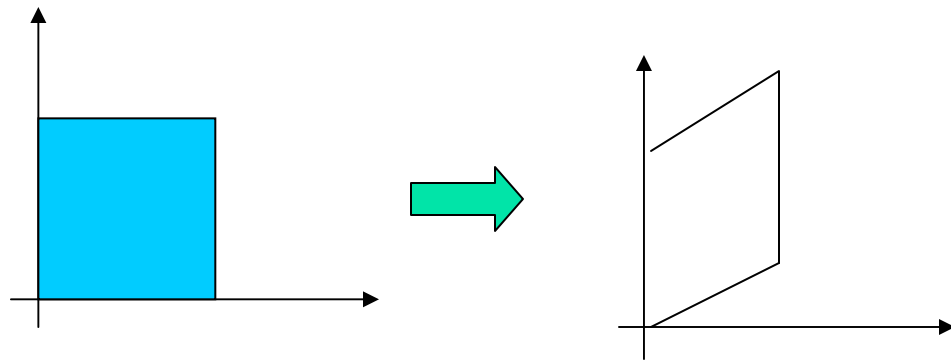
Shearing



- Y coordinates are unaffected, but x coordinates are translated linearly with y
- That is:
 - $y' = y$
 - $x' = x + y * h$

$$\begin{vmatrix} x \\ y \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Shearing in y



$$\begin{vmatrix} x \\ y \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ g & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

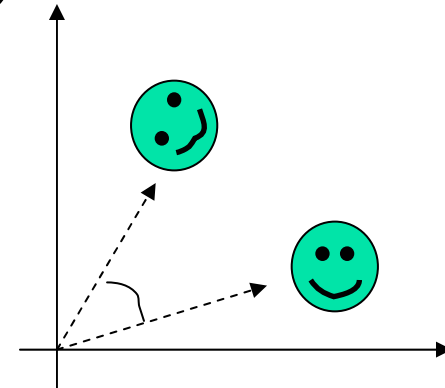
Interesting Facts:

- A 2D rotation is three shears
- Shearing will not change the area of the object
- Any 2D shearing can be done by a rotation, followed by a scaling, and followed by a rotation

Rotation Revisit

- The standard rotation matrix is used to rotate about the origin (0,0)

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

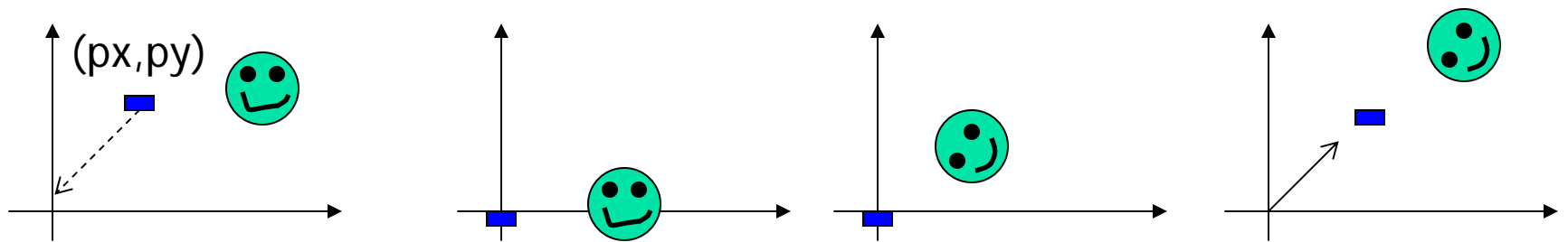


- What if I want to rotate about an arbitrary center?



Arbitrary Rotation Center

- To rotate about an arbitrary point $P (p_x, p_y)$ by θ :
 - Translate the object so that P will coincide with the origin: $T(-p_x, -p_y)$
 - Rotate the object: $R(\theta)$
 - Translate the object back: $T(p_x, p_y)$





Arbitrary Rotation Center

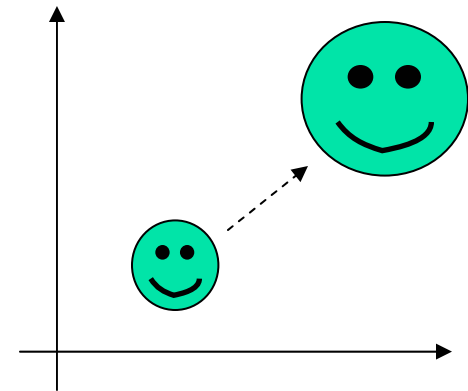
- Translate the object so that P will coincide with the origin: $T(-p_x, -p_y)$
 - Rotate the object: $R(\theta)$
 - Translate the object back: $T(p_x, p_y)$
- Put in matrix form: $T(p_x, p_y) R(\theta) T(-p_x, -p_y) * P$

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Scaling Revisit

- The standard scaling matrix will only anchor at (0,0)

$$\begin{matrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{matrix}$$

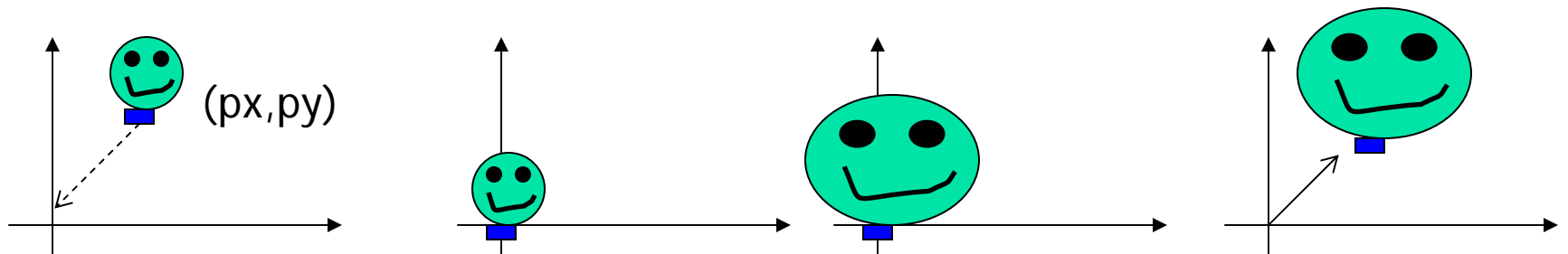


- What if I want to scale about an arbitrary pivot point?



Arbitrary Scaling Pivot

- To scale about an arbitrary pivot point P (p_x, p_y) :
 - Translate the object so that P will coincide with the origin: $T(-p_x, -p_y)$
 - Rotate the object: $S(s_x, s_y)$
 - Translate the object back: $T(p_x, p_y)$





Affine Transformation

- Translation, Scaling, Rotation, Shearing are all affine transformation
- Affine transformation – transformed point P' (x',y') is a **linear combination** of the original point P (x,y), i.e.

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

- Any 2D affine transformation can be decomposed into a rotation, followed by a scaling, followed by a shearing, and followed by a translation.

Affine matrix = translation x shearing x scaling x rotation



Composing Transformation

- **Composing Transformation** – the process of applying several transformation in succession to form one overall transformation
- If we apply transform a point P using M1 matrix first, and then transform using M2, and then M3, then we have:

$$(M3 \times (M2 \times (M1 \times P))) = M3 \times M2 \times M1 \times P$$

(pre-multiply)

M



Composing Transformation

- Matrix multiplication is associative

$$M_3 \times M_2 \times M_1 = (M_3 \times M_2) \times M_1 = M_3 \times (M_2 \times M_1)$$

- Transformation products may not be commutative

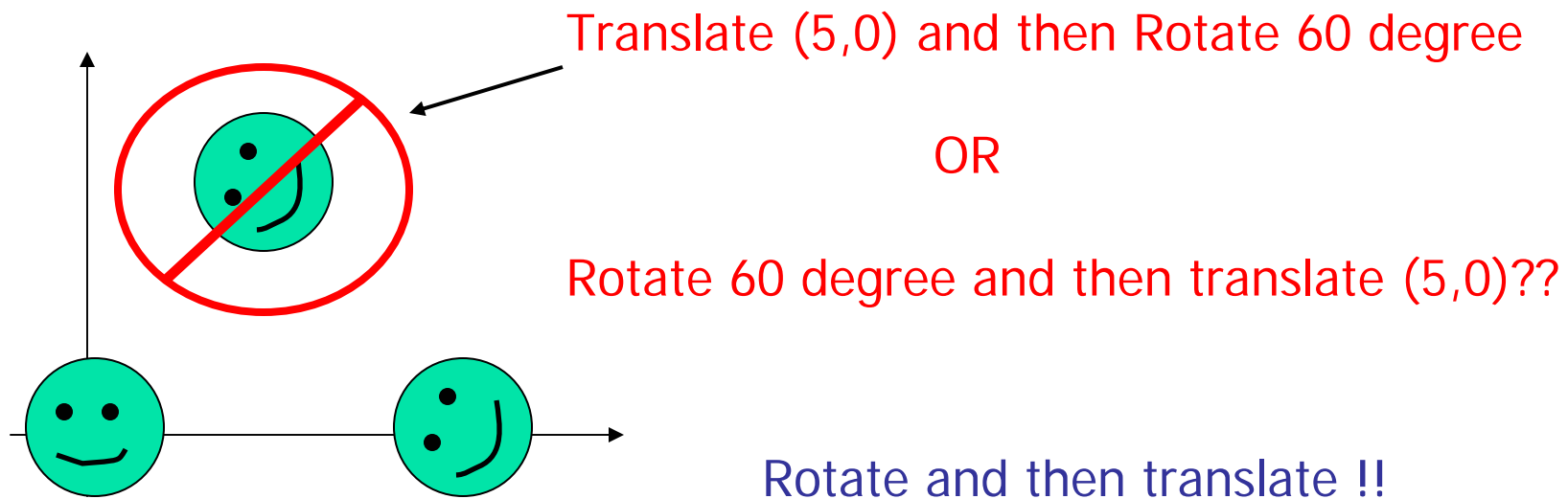
$$A \times B \neq B \times A$$

- Some cases where $A \times B = B \times A$

A	B
translation	translation
scaling	scaling
rotation	rotation
uniform scaling	rotation
($s_x = s_y$)	

Transformation order matters!

- Example: rotation and translation are not commutative





How OpenGL does it?



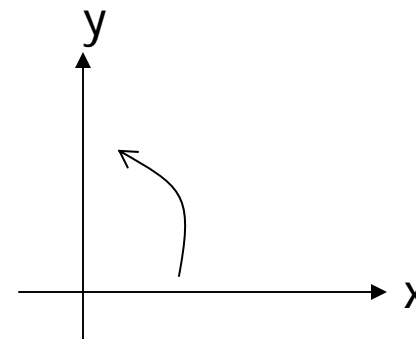
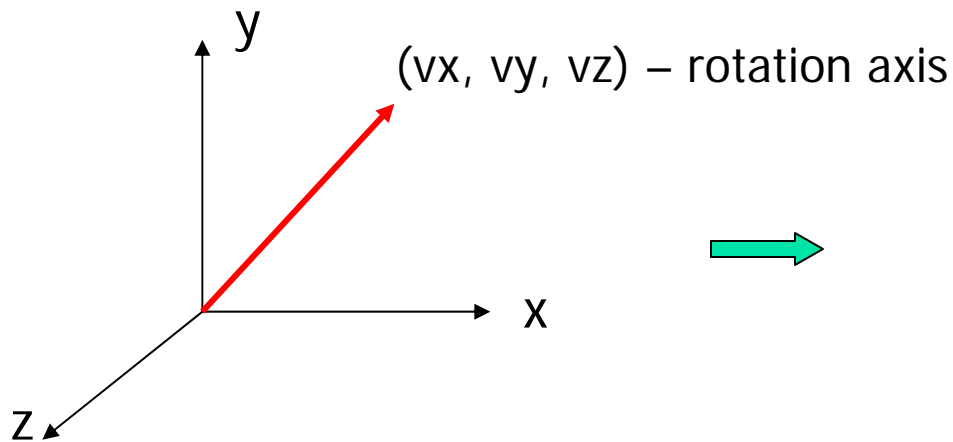
- OpenGL's transformation functions are meant to be used in 3D
- No problem for 2D though – just ignore the z dimension
- Translation:
 - `glTranslatef(d)(tx, ty, tz)` ->
`glTranslatef(d)(tx,ty,0)` for 2D

How OpenGL does it?



- Rotation:

- `glRotatef(d)(angle, vx, vy, vz) ->`
`glRotatef(d)(angle, 0,0,1)` for 2D



You can imagine z is pointing out of the slide



OpenGL Transformation Composition

- A global modeling transformation matrix

(`GL_MODELVIEW`, called it `M` here)

`glMatrixMode(GL_MODELVIEW)`

- The user is responsible to reset it if necessary

`glLoadIdentity()`

-> $M = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$



OpenGL Transformation Composition

- Matrices for performing user-specified transformations are multiplied to the model view global matrix
- For example,

$$\text{glTranslated}(1,1,0); \quad M = M \times \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

- All the vertices P defined within `glBegin()` will first go through the transformation (modeling transformation)

$$P' = M \times P$$



Transformation Pipeline

