CS5182 Computer Graphics Geometric Transformation

2024/25 Semester A

City University of Hong Kong (DG)

Outline

- Foundation Mathematics
- 2D Transformations
- 3D Transformations

transformations是什么？2D transformation是输入2D输出3D嘛，还是说点云之类数据类型的转换？

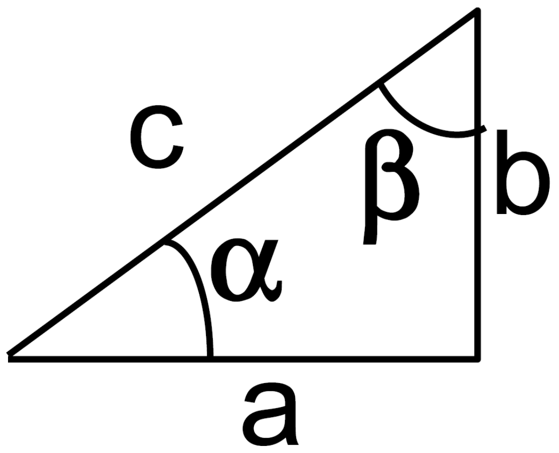
Foundation Mathematics

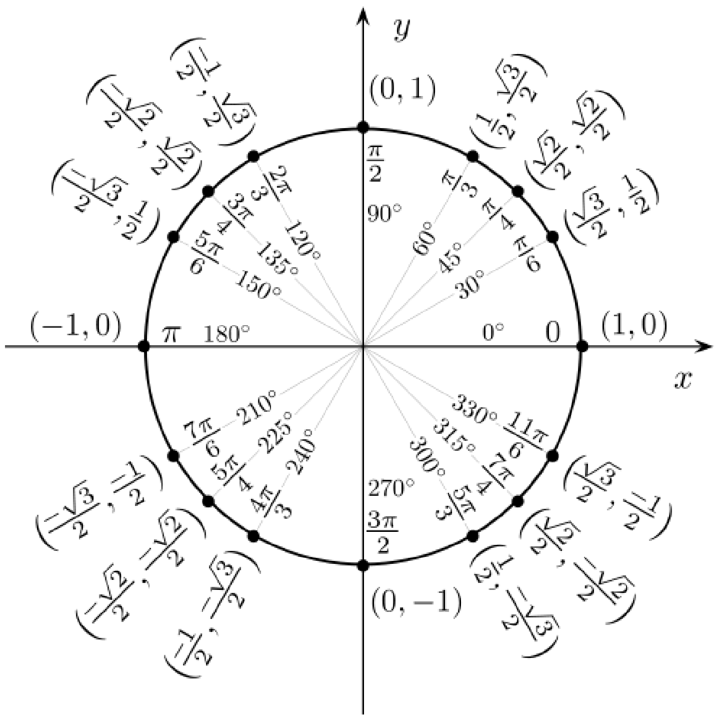
- Trigonometry
  - Studying relationships involving lengths and angles of triangles.

$a = c \times \cos\alpha \quad a = c \times \sin\beta$

$b = c \times \cos\beta \quad b = c \times \sin\alpha$

$b = a \times \tan\alpha \quad a = b \times \tan\beta$





Foundation Mathematics

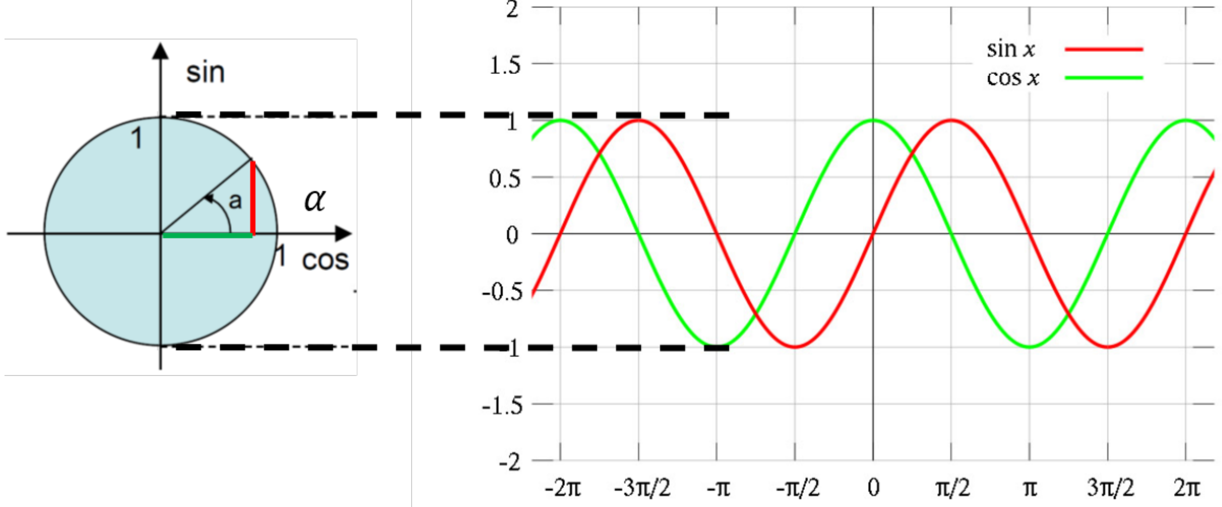
- Properties in Trigonometry

$$\sin^2\alpha + \cos^2\alpha = 1 \qquad \cos(-\alpha) = \cos\alpha$$

$$\sin(-\alpha) = -\sin\alpha \qquad \cos\alpha = \sin\left(\frac{\pi}{2} - \alpha\right)$$

$$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$$

$$\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$$



Matrix

- A rectangular array of quantities (numerical values, expressions, or functions)

$$\underset{(m \times n)}{\mathbf{A}} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- In most cases of computer graphics, we consider only the elements as numerical values.

- Square matrix if $m = n$.

Basic Matrix Operations

- Addition/Subtraction
  - Just add/subtract the corresponding elements
  - Two matrices must have an equal number of rows and columns to be added.

$$\underset{(m\times n)}{\mathbf{A}} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad \underset{(m\times n)}{\mathbf{B}} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ a_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

$$\underset{(m\times n)}{\mathbf{A} \pm \mathbf{B}} = \begin{bmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & \cdots & a_{1n} \pm b_{1n} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & \cdots & a_{2n} \pm b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} \pm b_{m1} & a_{m2} \pm b_{m2} & \dots & a_{mn} \pm b_{mn} \end{bmatrix}$$

Basic Matrix Operations

- Scalar multiplication
  - Multiply each element a_{ij} by the scalar λ

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad \lambda\mathbf{A} = \begin{bmatrix} \lambda a_{11} & \lambda a_{12} & \cdots & \lambda a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \cdots & \lambda a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda a_{m1} & \lambda a_{m2} & \dots & \lambda a_{mn} \end{bmatrix}$$

- Transpose
  - Interchange rows and columns

$$\mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}, \quad \begin{bmatrix} a & b & c \end{bmatrix}^T = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Basic Matrix Operations

- Matrix Multiplication

$$\underset{(m \times n)}{\mathbf{A}} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad \underset{(n \times p)}{\mathbf{B}} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{np} \end{bmatrix}$$

$$\underset{(m \times p)}{\mathbf{C} = \mathbf{AB}} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mp} \end{bmatrix} \qquad c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$$

Multiplication of two matrices is defined only if the number of columns of the left matrix is the same as the number of rows of the right matrix.

Basic Matrix Operations

- Matrix Multiplication
  - Examples

$$\begin{bmatrix} 1 & 0 & 2 \\ -1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 1 \\ 2 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} (1\times 3+0\times 2+2\times 1) & (1\times 1+0\times 1+2\times 0) \\ (-1\times 3+3\times 2+1\times 1) & (-1\times 1+3\times 1+1\times 0) \end{bmatrix}$$

$(2 \times 3)(3 \times 2)$

$$= \begin{bmatrix} 5 & 1 \\ 4 & 2 \end{bmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix} \quad \text{is not defined}$$

$(4 \times 3)(2 \times 4)$ product is not defined

Basic Matrix Operations

- Inverse matrix
  - If the inverse matrix of a square matrix exists, the matrix is said to be nonsingular or invertible.
  - The inverse of matrix $\mathbf{A}$ is denoted as $\mathbf{A}^{-1}$, and

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

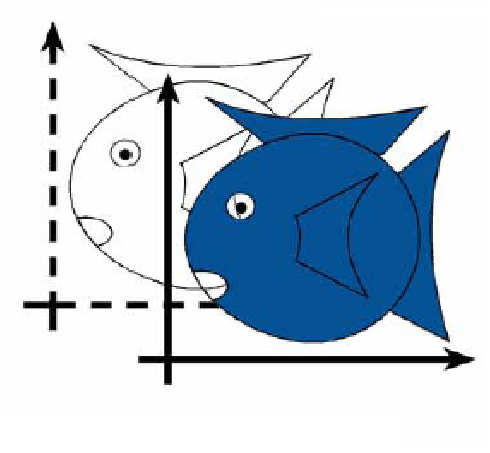
- Orthogonal matrix
  - An orthogonal matrix is a square matrix $\boldsymbol{Q}$ whose transpose is its inverse

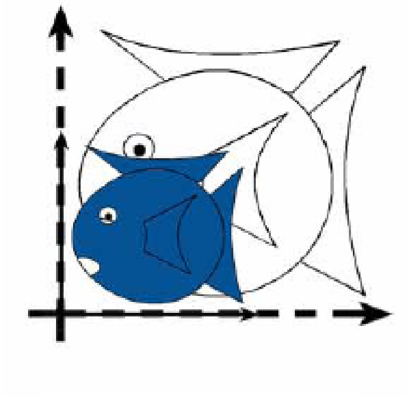
$$\mathbf{Q}^{T}\mathbf{Q} = \mathbf{Q}\mathbf{Q}^{T} = I$$

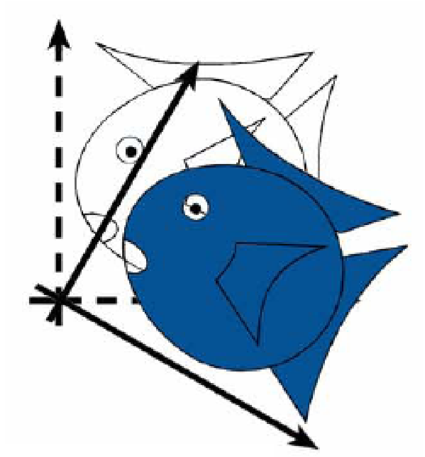
$$\mathbf{Q}^{T} = \mathbf{Q}^{-1}$$

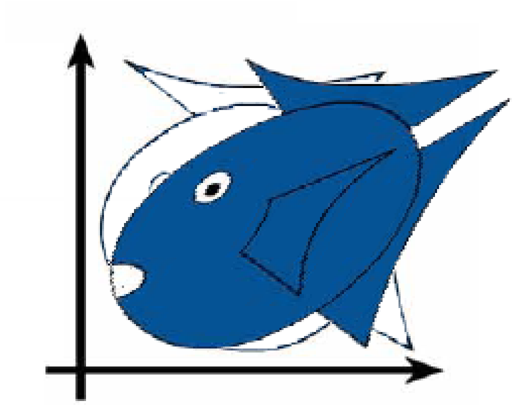
2D Transformation

- Translation
- Scaling
- Rotation
- Shearing


Translation


Uniform Scaling

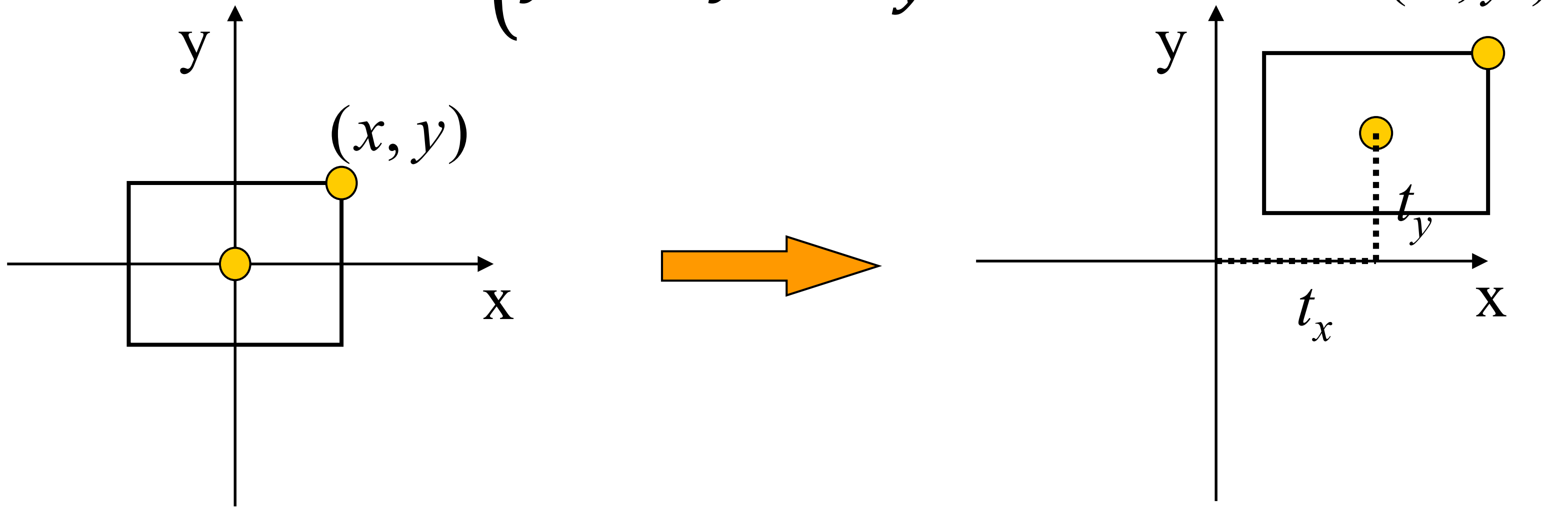

Rotation


Shearing

2D Translation

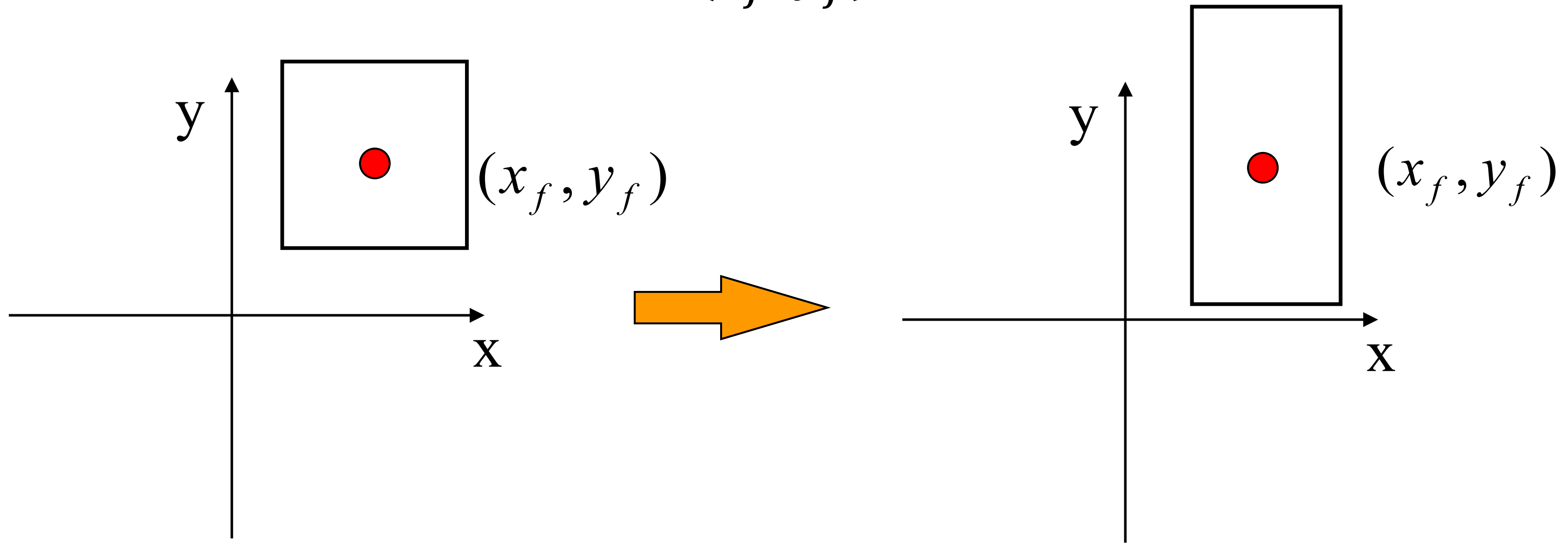
- Move the object along the 2D plane
  - Original vertex: (x, y)
  - Translation distances: (t_x, t_y)
  - New vertex: (x', y')

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$



2D Scaling

- Change the size of the object
  - Scaling factors: (s_x, s_y). Uniform scaling if $s_x = s_y$; non-uniform, otherwise.
  - Reference/fixed point: (x_f, y_f)

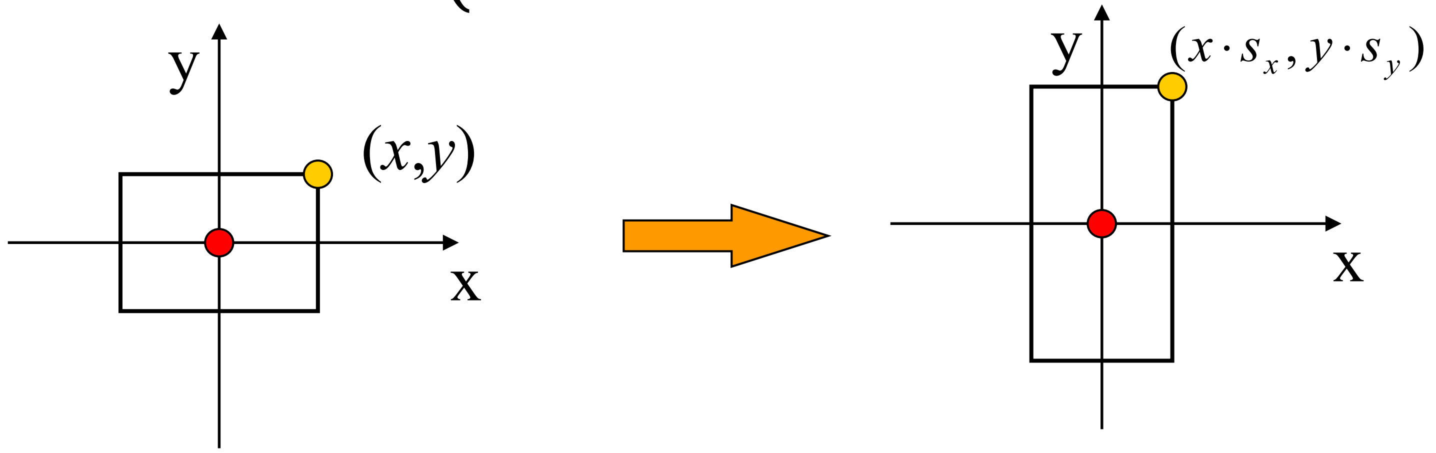


The reference/fixed point remains unchanged after the scaling.

2D Scaling

- Simple case: the reference/fixed point is the **origin**
  - Original vertex: (x, y)
  - Scaling factors: (s_x, s_y)
  - New vertex: (x', y')

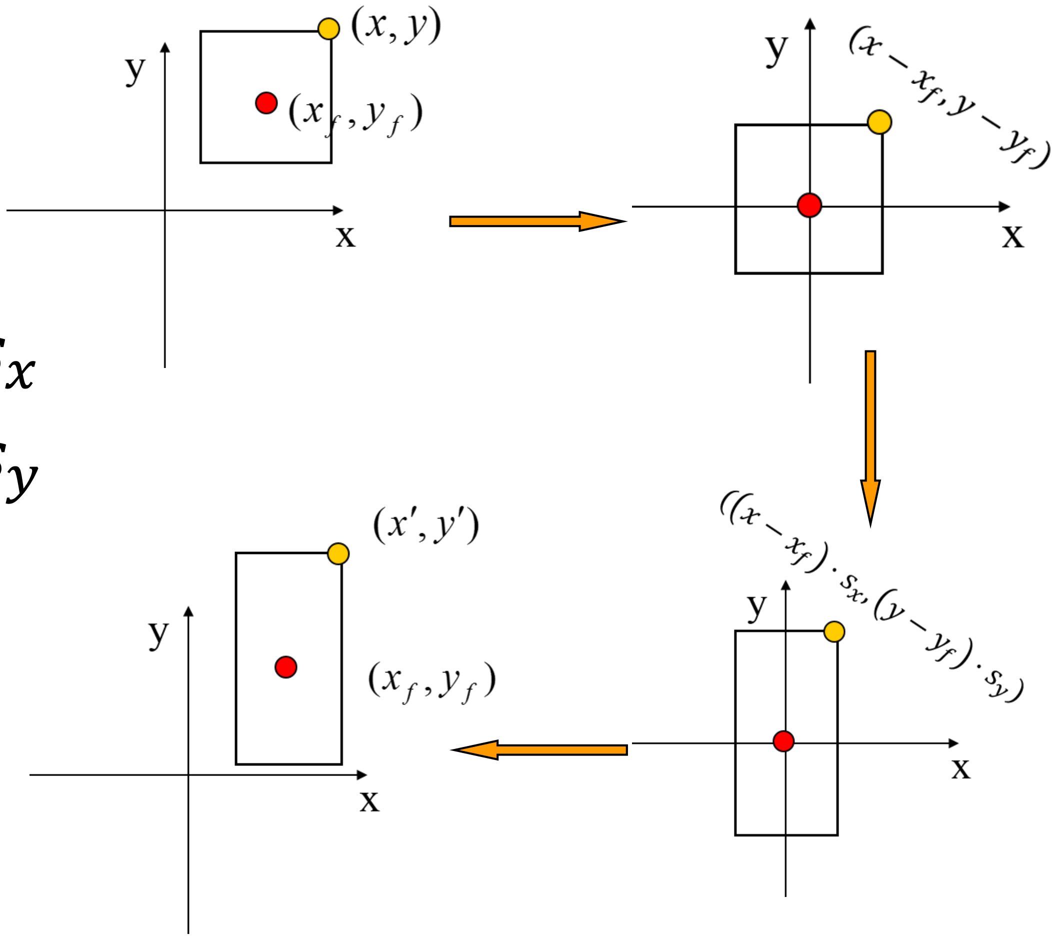
$$\begin{cases} x' = x \cdot s_x \\ y' = y \cdot s_y \end{cases}$$



2D Scaling

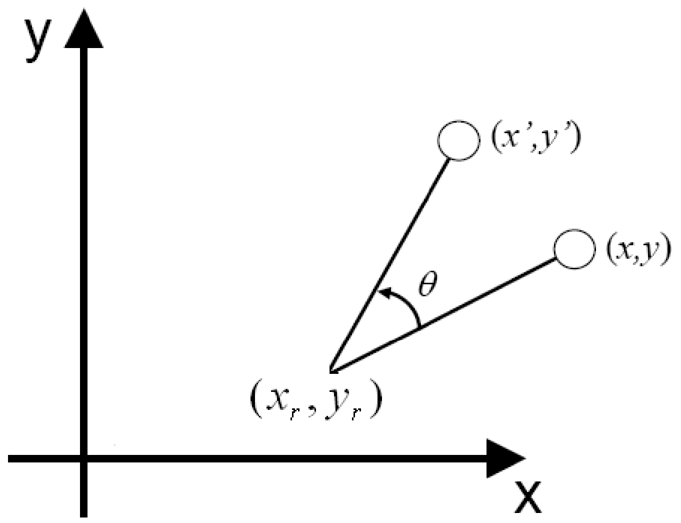
- General case: **arbitrary** reference/fixed point(x_f, y_f)
  - Original vertex: (x, y)
  - Scaling factors: (s_x, s_y)
  - New vertex: (x', y')

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



2D Rotation

- Rotate the object about a point on the 2D plane
  - Rotation (pivot) point: (x_r, y_r)
  - Rotation angle (**counter clockwise**): θ
  - Original vertex: (x, y)
  - New vertex: (x', y')



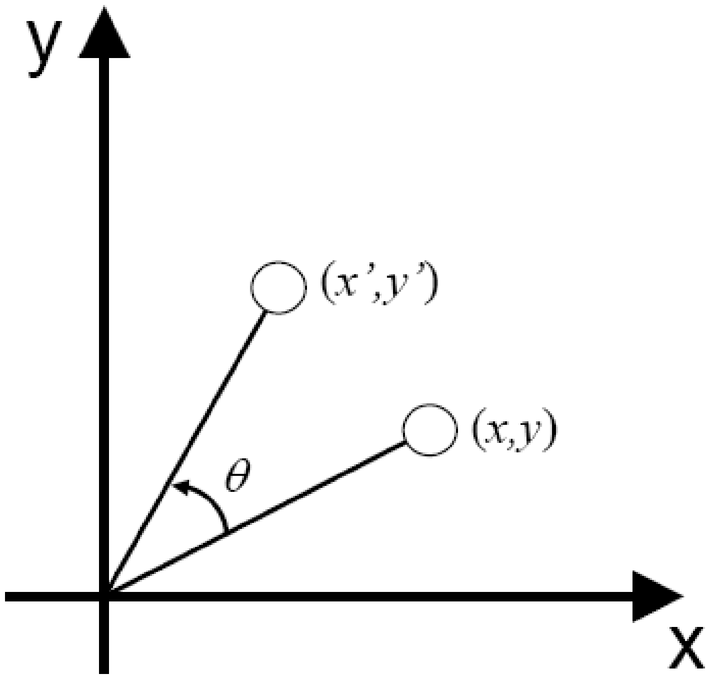
- What if the rotation angle is measured clockwise?
  - Rotation by an angle θ in CW is equivalent to the rotation by an angle $2\pi - \theta$ in CCW.

2D Rotation

- Simple case: the pivot point is the **origin**
  - Rotation (pivot) point: $(0,0)$
  - Rotation angle (counter clockwise): θ
  - Original vertex: (x, y)
  - New vertex: (x', y')

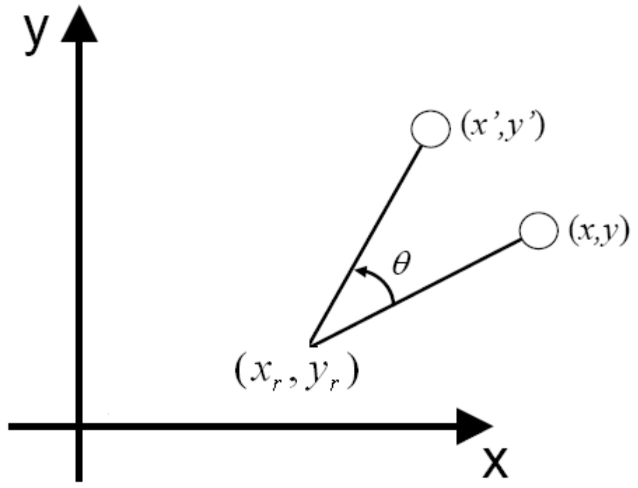
$$x' = x\cos(\theta) - y\sin(\theta)$$

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



2D Rotation

- General case: arbitrary pivot position
  - Rotation (pivot) point: (x_r, y_r)
  - Rotation angle (counter clockwise): θ
  - Original vertex: (x, y)
  - New vertex: (x', y')



Step 1) Translate the object such that the pivot point coincides with the origin;

Step 2) rotate the object around the origin (the pivot point);

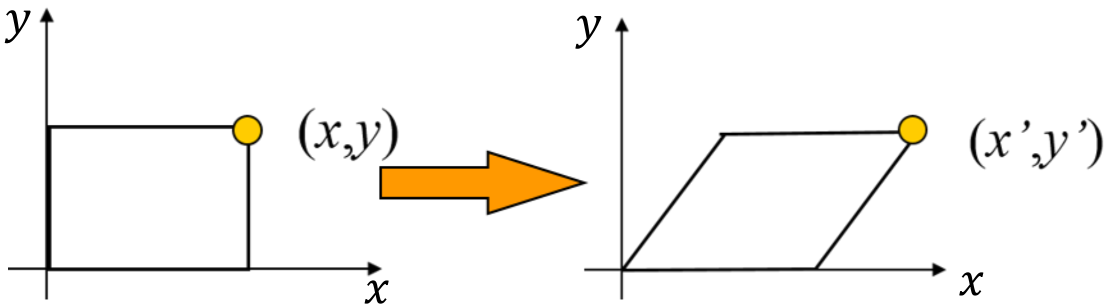
Step 3) undo the translation.

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

2D Shearing

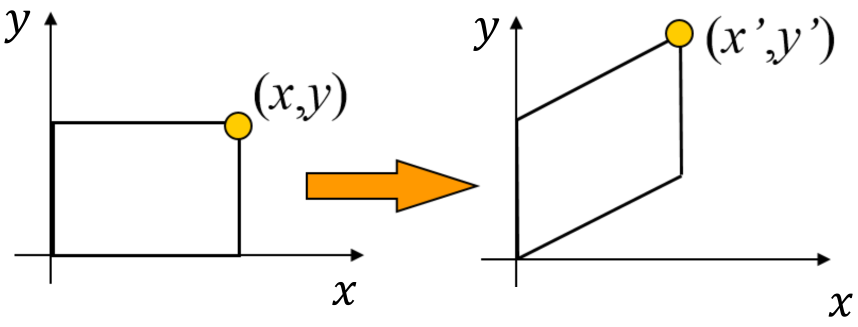
- Leave all points on one axis fixed, while the other points are shifted parallel to the axis by *a distance proportional to their perpendicular distance from that axis*.
  - A horizontal shearing (or x-shearing) is given by

$$\begin{cases} x' = x + \lambda_x \times y \\ y' = y \end{cases}$$



  - A vertical shearing (or y-shearing) is given by

$$\begin{cases} x' = x \\ y' = y + \lambda_y \times x \end{cases}$$



Homogeneous Coordinates

- A 2D coordinate (x, y) can be written as (x_h, y_h, h) such that

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

where h is a scaling factor. (x_h, y_h, h) is called a homogeneous coordinate. Typically, h is set to 1. The homogeneous coordinate of a point (x, y) is $(x, y, 1)$.

- With homogeneous coordinates, we can build a unified approach to combine the transformations so that the final coordinate positions are obtained directly from the initial coordinate **by the product of the transformation matrices.**

2D Translation Matrix

- Using homogeneous coordinates, we can represent the translation using the following matrix multiplication

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \end{aligned} \quad \Longrightarrow \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{T}(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},$$

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

2D Scaling Matrix

- Scaling relative to the origin

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

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \qquad \mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},$$

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

2D Scaling Matrix

- Scaling relative to an arbitrary fixed point (x_f, y_f)

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{T}(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},$$

$$\mathbf{P}' = \mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \underbrace{\mathbf{T}(-x_f, -y_f) \cdot \mathbf{P}}_{\text{Translate the object such that the fixed point is moved to the origin}}$$

$$\underbrace{}_{\text{Scale the object about the origin}}$$

$$\underbrace{}_{\text{Translate the object such that the fixed point is returned to the original position}}$$

2D Rotation Matrix

- Rotation around the origin

$$\begin{aligned} x' &= x\cos(\theta) - y\sin(\theta) \\ y' &= x\sin(\theta) + y\cos(\theta) \end{aligned} \quad \Longrightarrow \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},$$

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

2D Rotation Matrix

- Rotation around an arbitrary point (x_r, y_r)

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

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{T}(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},$$

$$\mathbf{P}' = \underbrace{\mathbf{T}(x_r, y_r)}_{\text{Translate the object such that the pivot point is returned to the original position}} \cdot \underbrace{\mathbf{R}(\theta)}_{\text{Rotate the object about the coordinate origin}} \cdot \underbrace{\mathbf{T}(-x_r, -y_r) \cdot \mathbf{P}}_{\text{Translate the object such that the pivot point is moved to the origin}}$$

2D Shearing Matrix

- Horizontal shearing (or x-shearing)

$$\begin{cases} x' = x + \lambda_x \times y \\ y' = y \end{cases} \Rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \lambda_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Vertical shearing (or y-shearing)

$$\begin{cases} x' = x \\ y' = y + \lambda_y \times x \end{cases} \Rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \lambda_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Inverse Transformation

- A transformation **M** moves a point **P** to **P**′. Can we *undo* the transformation, i.e., recover **P** from **P**′?

$$\mathbf{P}' = \mathbf{M}\mathbf{P}$$

$$\mathbf{P} = \mathbf{M}^{-1}\mathbf{P}'$$

We need to compute the inverse matrix $\mathbf{M}^{-1}$

Inverse Transformation

- Inverse of the translation matrix
  - Negate the translation distance

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

Verify that $\mathbf{T}^{-1} \cdot \mathbf{T}(t_x, t_y) = \mathbf{I}$

Inverse Transformation

- Inverse of the rotation matrix
  - Negate the rotation angle, which is equivalent to taking the transpose of the rotation matrix

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

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

Verify that $\mathbf{R}^{-1} \cdot \mathbf{R}(\theta) = \mathbf{I}$

Inverse Transformation

- Inverse of the scaling matrix
  - Replace the scaling factors with their reciprocals

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

Verify that $\mathbf{S}^{-1} \cdot \mathbf{S}(t_x, t_y) = \mathbf{I}$

Inverse Transformation

- Inverse of the shearing matrix
  - Negate the shearing factor

$$\mathbf{S}_h(\lambda_x) = \begin{bmatrix} 1 & \lambda_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{S}_h^{-1} = \begin{bmatrix} 1 & -\lambda_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Verify that $\mathbf{S}_h^{-1} \cdot \mathbf{S}_h(\lambda_x) = \mathbf{I}$

Order of 2D Transformations

- The order of transformations **does** matter!
  - Matrix multiplication is **not** commutative! In general,

$$\mathbf{AB} \neq \mathbf{BA}$$

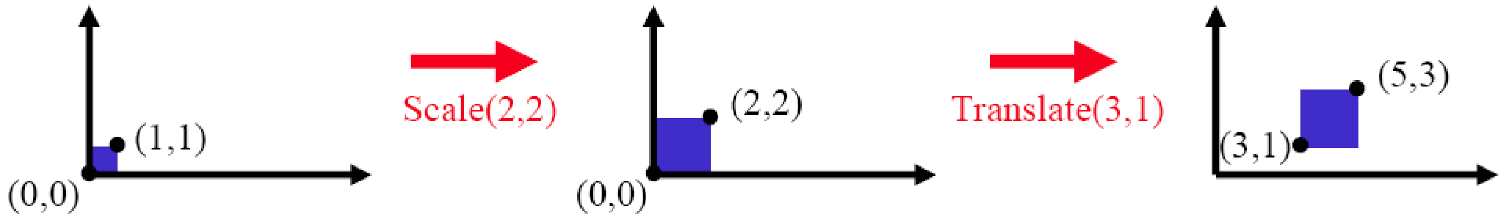
  - Reversing the order in which a sequence of transformations is performed **may** affect the transformed position of an object!

Order of Transformations

- Scale then translate

$$p' = T(Sp) = TSp$$

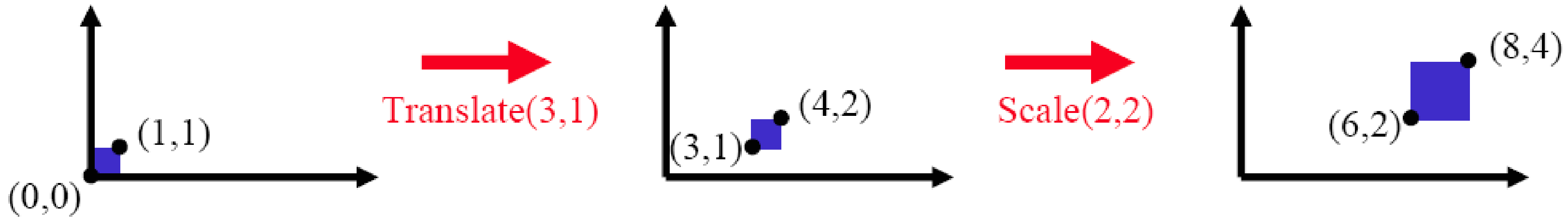
$$TS = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$



- Translate then scale

$$p'' = S(Tp) = STp$$

$$ST = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$



3D Transformation

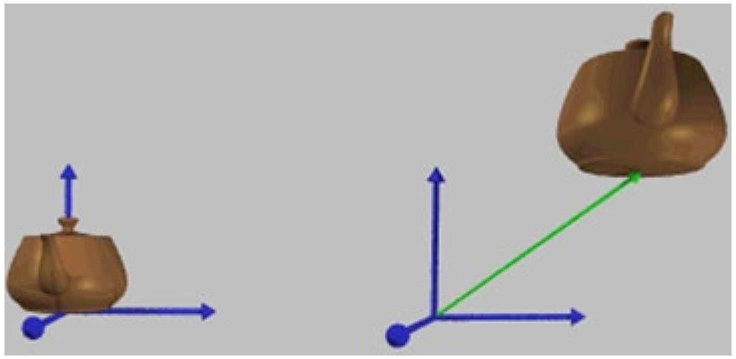
- Translation
- Scaling
- Rotation

- The coordinate of a point in 3D space is denoted as (x, y, z), and its homogeneous coordinate is represented in $(x, y, z, 1)$.

3D Translation

- We can translate points in space to new positions by adding offsets to their coordinates, as shown in the following vector equation.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$



- With homogeneous coordinate, we can rewrite the equation into matrix form.

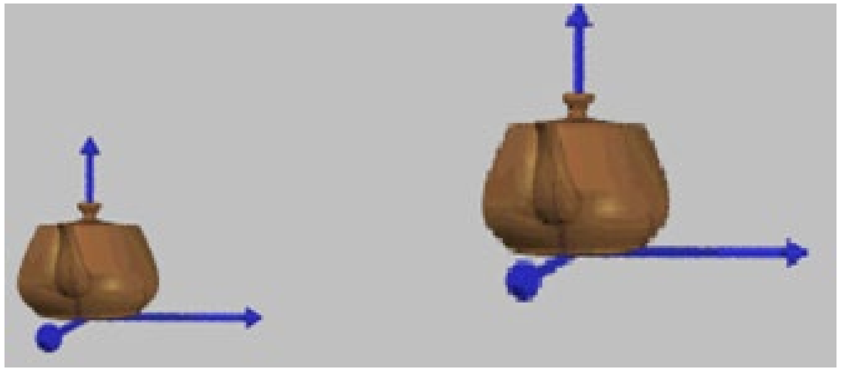
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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

3D Scaling

- Scaling relative to the **origin**

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



- With homogeneous coordinates, we obtain

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

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

3D Scaling

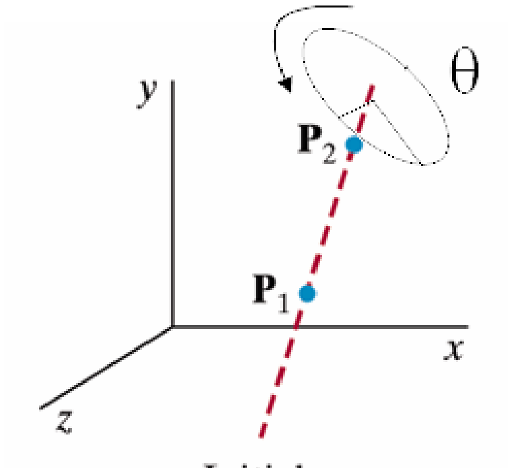
- What if the reference/fixed point is not the origin?
  - To scale about an arbitrary fixed point, (x_f, y_f, z_f), we apply the classic three-step trick:

$$\mathbf{P}' = \underbrace{\mathbf{T}\left(x_f, y_f, z_f\right)}_{\text{Translate the object such that the fixed point is returned to the original position}} \cdot \underbrace{\mathbf{S}\left(s_x, s_y, s_z\right)}_{\text{Scale the object about the origin}} \cdot \underbrace{\mathbf{T}\left(-x_f, -y_f, -z_f\right) \cdot \mathbf{P}}_{\text{Translate the object such that the fixed point is moved to the origin}}$$

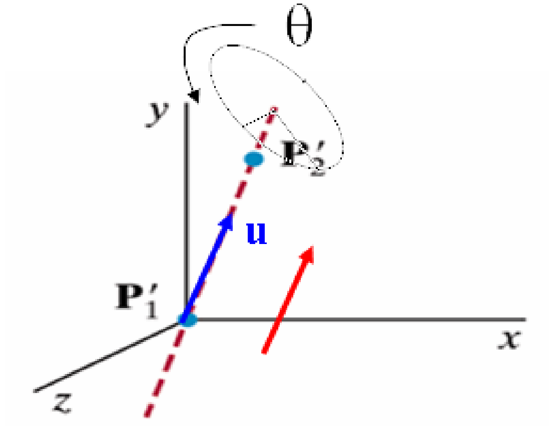
- Objects can be scaled to different sizes. If the scaling is uniform, the shape is preserved.

3D Rotation

- 3D rotation is defined by a rotation angle and a rotation axis.
- A general 3D rotation axis is defined by two points P_1 and P_2.
- We can translate the axis such that P1 is at the origin. Then we perform the 3D rotation. Finally, we translate the axis back to the original location



Initial Position

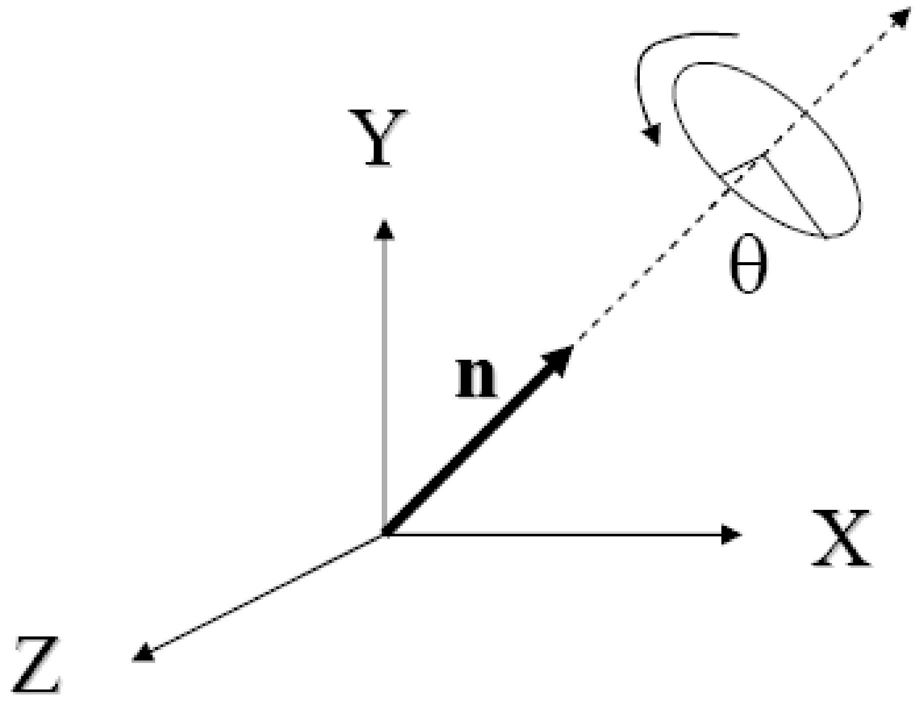


Translate P_1 to the Origin

3D Rotation

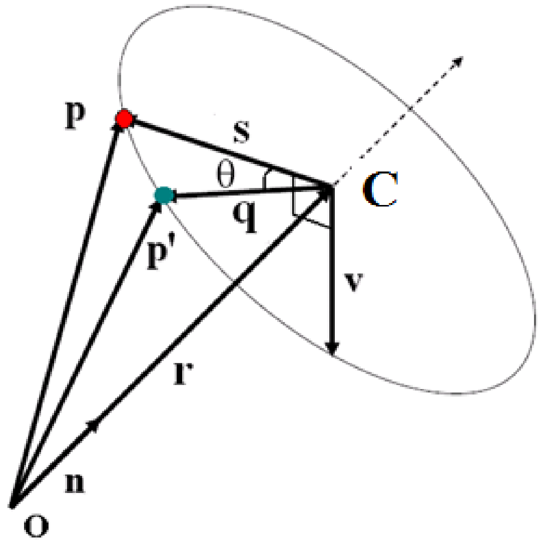
- We consider the special case that the rotation axis passes through the origin.

- A unit vector **n** indicates the axis of rotation and a scalar θ indicates the angle of rotation



3D Rotation

- 3D rotation in terms of angle-axis parameters
  - Rotate the point **p** in a counterclockwise direction about the ray from the origin through the point (*x*, *y*, *z*). The angle of rotation is **θ**.



$$\mathbf{n} = \frac{(x, y, z)}{\sqrt{x^2 + y^2 + z^2}}$$

$$\mathbf{r} = (\mathbf{p}\cdot\mathbf{n})\mathbf{n}$$

$$\mathbf{s} = \mathbf{p} - \mathbf{r}$$

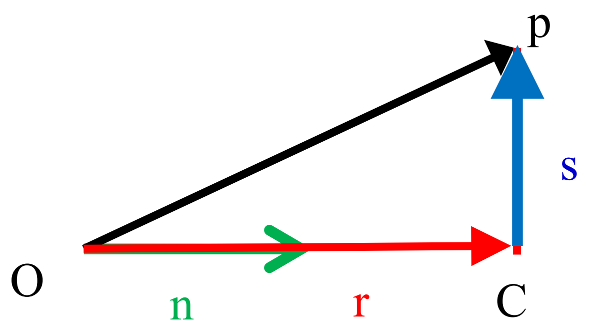
$$\mathbf{v} = \mathbf{n}\times\mathbf{s} = \mathbf{n}\times\mathbf{p} \quad (\because \mathbf{n}\times\mathbf{r} = 0)$$

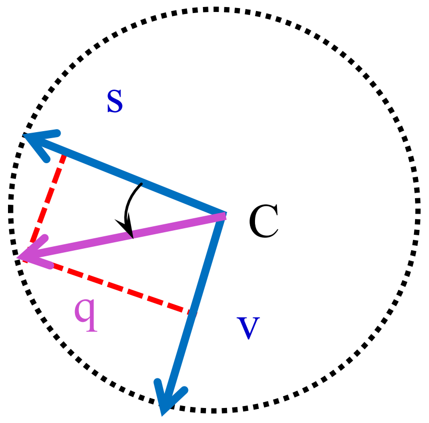
$$\mathbf{q} = \cos\theta\mathbf{s} + \sin\theta\mathbf{v}$$

$$\mathbf{p}' = \mathbf{r} + \mathbf{q}$$

$$= (\mathbf{n}\cdot\mathbf{p})\mathbf{n} + \cos\theta(\mathbf{p} - (\mathbf{n}\cdot\mathbf{p})\mathbf{n}) + \sin\theta(\mathbf{n}\times\mathbf{p})$$

$$= \mathbf{p}\cos\theta + (\mathbf{n}\cdot\mathbf{p})\mathbf{n}(1-\cos\theta) + (\mathbf{n}\times\mathbf{p})\sin\theta$$





We need to put it into a matrix representation!

3D Rotation

- Forming the 3D rotation matrix in terms of angle-axis parameters

$$\text{Let } \mathbf{M} = \frac{1}{\sqrt{x^2+y^2+z^2}} \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

$$\mathbf{R}(\theta, x, y, z) = \mathbf{I} + \sin\theta\mathbf{M} + (1-\cos\theta)\mathbf{M}^2$$

- Verify the rotation matrix

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\mathbf{Rp} = \left(\mathbf{I} + \sin\theta\mathbf{M} + (1-\cos\theta)\mathbf{M}^2\right)\mathbf{p}$$

$$= \mathbf{p} + \sin\theta\mathbf{Mp} + (1-\cos\theta)\mathbf{M}^2\mathbf{p} \qquad \text{distributative law}$$

$$= \mathbf{p} + \sin\theta\mathbf{n}\times\mathbf{p} + (1-\cos\theta)\mathbf{n}\times(\mathbf{n}\times\mathbf{p}) \qquad \text{cross product matrix}$$

$$= \mathbf{p} + \sin\theta\mathbf{n}\times\mathbf{p} + (1-\cos\theta)(\mathbf{n}(\mathbf{n}\cdot\mathbf{p}) - \mathbf{p}(\mathbf{n}\cdot\mathbf{n})) \quad \text{cross product identity}$$

$$= \mathbf{p} + \sin\theta\mathbf{n}\times\mathbf{p} + (1-\cos\theta)(\mathbf{n}(\mathbf{n}\cdot\mathbf{p}) - \mathbf{p}) \quad \text{unit vector dot product}$$

$$= \mathbf{p}\cos\theta + (\mathbf{n}\cdot\mathbf{p})\mathbf{n}(1-\cos\theta) + (\mathbf{n}\times\mathbf{p})\sin\theta \quad \text{algebra}$$

$$= \mathbf{p}'$$

Notice that this is the same formula we derived earlier.

3D Rotation

- 3D rotation matrix in homogeneous coordinates

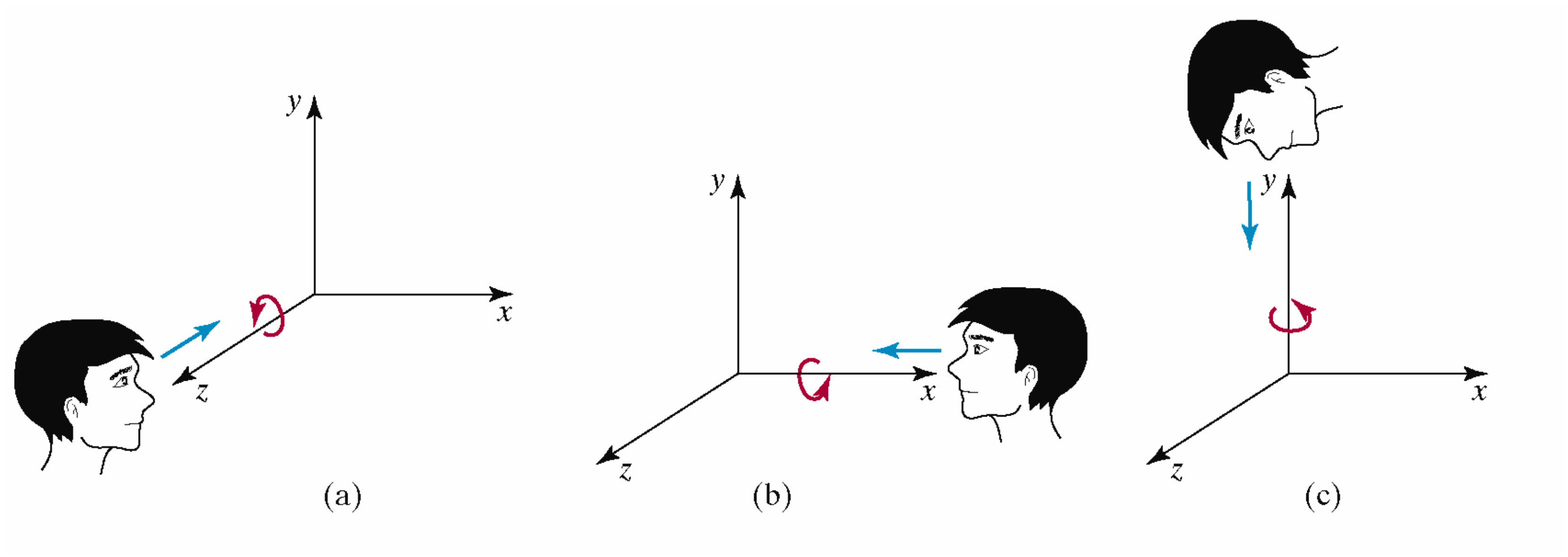
$$\mathbf{R}(\theta, x, y, z) = \begin{bmatrix} \hat{x}^2 + c(1 - \hat{x}^2) & (1-c)\hat{x}\hat{y} - s\hat{z} & (1-c)\hat{z}\hat{x} + s\hat{y} & 0 \\ (1-c)\hat{x}\hat{y} + s\hat{z} & \hat{y}^2 + c(1 - \hat{y}^2) & (1-c)\hat{y}\hat{z} - s\hat{x} & 0 \\ (1-c)\hat{z}\hat{x} - s\hat{y} & (1-c)\hat{y}\hat{z} + s\hat{x} & \hat{z}^2 + c(1 - \hat{z}^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{where } (\hat{x}, \hat{y}, \hat{z}) = \frac{(x, y, z)}{\sqrt{x^2 + y^2 + z^2}}, c = \cos\theta, s = \sin\theta$$

You are not required to memorize this matrix!!!

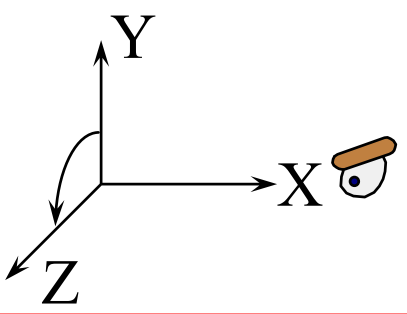
3D Rotation

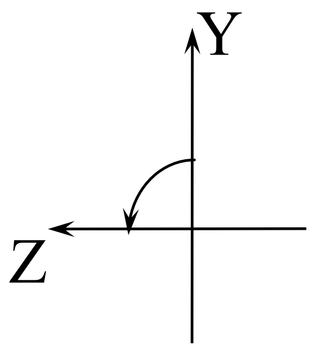
- Special cases: rotation about a coordinate axis
  - Rotations about a coordinate axis are counterclockwise, when **looking along the positive half of the axis toward the origin**



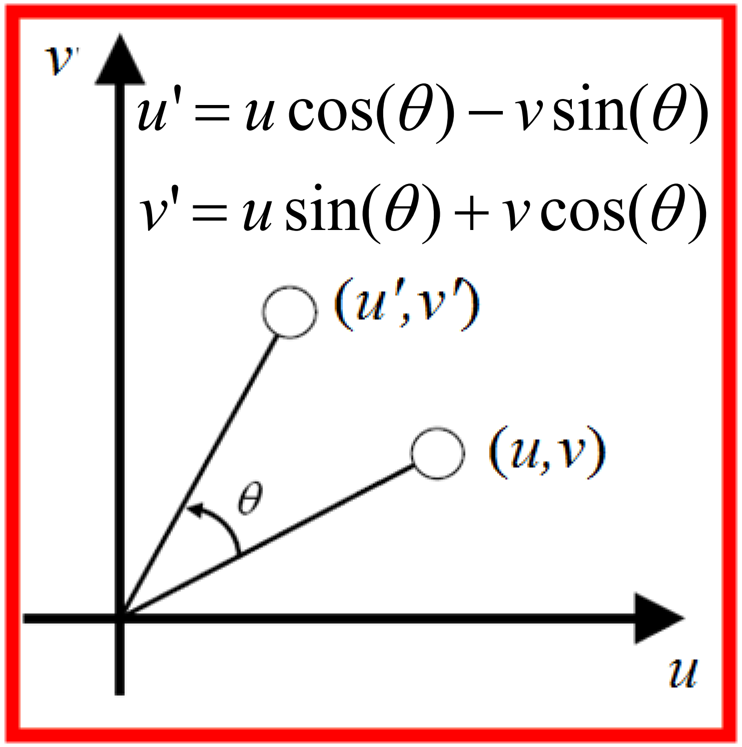
3D Rotation

- Rotation around x- axis
  - x-coordinate remains unchanged





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

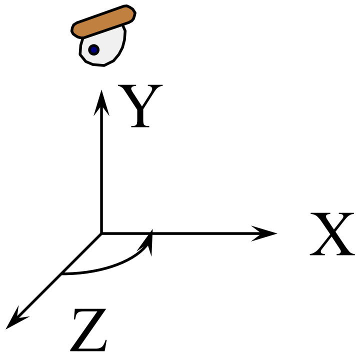


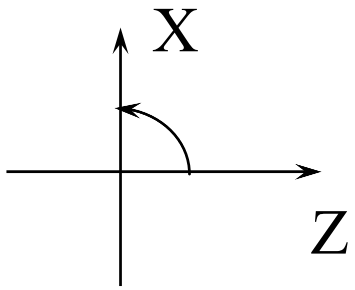
$$u' = u\cos(\theta) - v\sin(\theta)$$
$$v' = u\sin(\theta) + v\cos(\theta)$$

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

3D Rotation

- Rotation around y-axis
  - y-coordinate remains unchanged





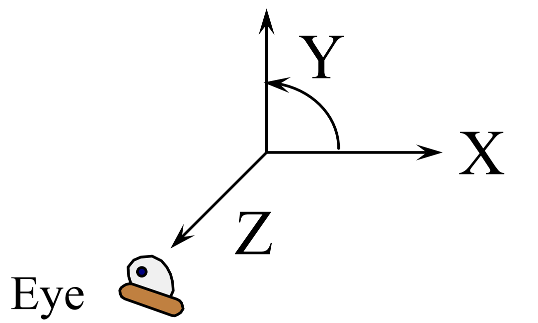
$$z' = z\cos\theta - x\sin\theta$$

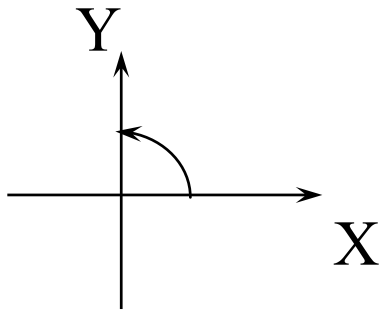
$$x' = z\sin\theta + x\cos\theta$$

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

3D Rotation

- Rotation around z-axis
  - z-coordinate remains unchanged





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

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

3D Rotation

- An arbitrary rotation about the **origin** can be decomposed into three successive rotations about the three axes.

- 3D rotations are not commutative, meaning that changing the order of the above equation will affect the result. In general,

$$\underset{\substack{\uparrow\\ \text{3rd}}}{\mathbf{R}_x(\alpha)}\underset{\substack{\uparrow\\ \text{2nd}}}{\mathbf{R}_y(\beta)}\underset{\substack{\uparrow\\ \text{1st}}}{\mathbf{R}_z(\gamma)}\mathbf{P} \neq \underset{\substack{\uparrow\\ \text{3rd}}}{\mathbf{R}_x(\alpha)}\underset{\substack{\uparrow\\ \text{2nd}}}{\mathbf{R}_z(\gamma)}\underset{\substack{\uparrow\\ \text{1st}}}{\mathbf{R}_y(\beta)}\mathbf{P}$$

Concatenation of Matrices

- If we need to apply a sequence of transformations on an input point, we may represent it as

$$\mathbf{P}' = \mathbf{M}_n \mathbf{M}_{n-1} \cdots \mathbf{M}_1 \mathbf{P}$$

where $\mathbf{P}$ is the input point and $\mathbf{P}'$ is the output point after applying transformation matrices $\mathbf{M}_i$ to $\mathbf{P}$.

- Instead of applying the n matrices to the input point one by one, we may first combine the three matrices into one before we apply it to the input point, i.e., $\mathbf{M} = \mathbf{M}_n \mathbf{M}_{n-1} \cdots \mathbf{M}_1$ and then $\mathbf{P}' = \mathbf{M}\mathbf{P}$.
- The advantage of this is that the computational cost will be significantly reduced when handling a large number of points.

Concatenation of Matrices

- A common task in computer graphics: given 1000 points, apply 100 transformations to every point.

Apply 100 transformations to 1000 points:

```
for (i = 0; i<1000; i++)
 for (j = 0; j < 100; j++)
  point[i]=apply(transformation[j],point[i]);
```

First multiply 100 matrices and then apply the product matrix to each of 1000 points:

```
t = identityMatrix;
for (j=0; j<100; j++)
 t=matrixMatrixMultiply(matrix[j],t);
for (i = 0; i<1000; i++)
 point[i]=matrixVectorMultiply(t,point[i]);
```