



5CM507 Graphics

Lecture A02b Transformation

Dr Youbing Zhao

October 2, 2025

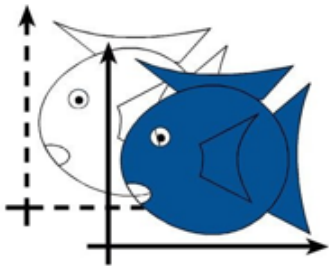
Contents

- ▶ 2D and 3D Scaling and Translation
- ▶ Homogeneous Coordinates
- ▶ 2D and 3D Rotation

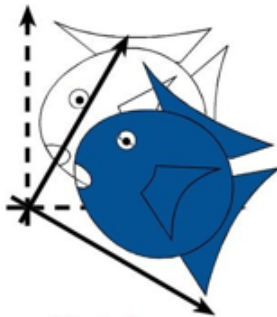


Common Transformations

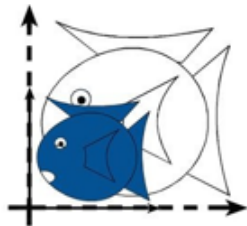
- We need to find transformations that follow certain rules



Translation



Rotation



Isotropic
(Uniform)
Scaling

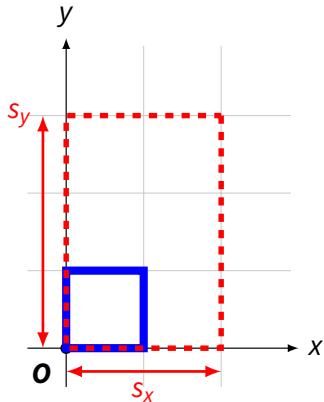
The background is composed of two large, overlapping geometric shapes. A teal-colored shape occupies the top-left corner, while a light beige shape occupies the bottom-left corner. The rest of the background is white. The word "Transforms" is centered in the white area.

Transforms

2D and 3D Scaling

- ▶ If we want to scale a rigid object centred at the origin by a scale factor of $S = S(s_x, s_y)$
- ▶ that is to transform every point p on the object to p' with the following equations
 - ▶ $x' = s_x x \quad y' = s_y y$
- ▶ Matrix form: $\mathbf{p}' = \mathbf{S}\mathbf{p}$

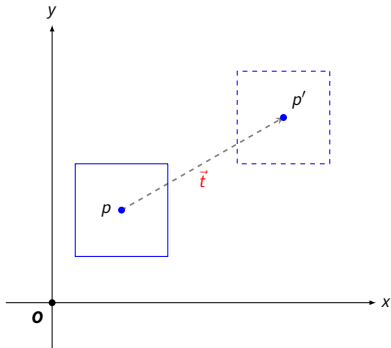
$$\mathbf{S} = \mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$



2D and 3D Translation

- Move (translate, displace) a point to a new location
 - $x' = x + t_x$ $y' = y + t_y$
- Vector form : displacement determined by a vector \vec{T}

- $p' = p + \vec{t}$
- $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}, \mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$



Homogenous Coordinates

Homogenous Coordinates

The problem



Translation cannot be represented as a matrix like

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \text{ but in } \begin{cases} x' = x + t_x \\ y' = y + t_y \\ z' = z + t_z \end{cases} \quad t_x, t_y, t_z \text{ do not depend on } x, y, z$$

Idea: add an extra "scaling" dimension:

$$\mathbf{P} = [x \quad y \quad z \quad 1]^T$$

$$\mathbf{P}' = [x' \quad y' \quad z' \quad w']^T \iff \left[\frac{x}{w} \quad \frac{y}{w} \quad \frac{z}{w} \quad 1 \right]^T$$

You will see $w \neq 1$ in perspective projection.

This is why you see this in the vertex shader: `gl_Position = vec4(pos, 1.0);`

3D Translation and Scaling Matrix



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

`glm::mat4 glm::translate(glm::vec3(tx, ty, tz))`

$$\begin{cases} x' = s_x x \\ y' = s_y y \\ z' = s_z z \end{cases} \quad \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

`glm::mat4 glm::scale(glm::vec3(sx, sy, sz))`

Now multiple transforms can be concatenated using matrix multiplication.

2D and 3D rotations

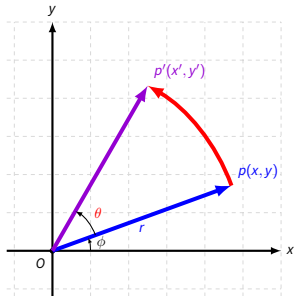
2D Rotation on XY plane to 3D rotation about Z

- ▶ Rotate a point $\mathbf{p}(x, y)$ about the origin for angle θ in counterclockwise direction to $\mathbf{p}'(x', y')$

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

- ▶ Add $z' = z \Rightarrow$ 3D rotation about Z axis

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



3D Rotation about global X and Y axes



Keep x unchanged => rotate about X axis

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

Keep y unchanged => rotate about Y axis

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

Homogeneous coordinate is trivial:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & 0 \\ r_{10} & r_{11} & r_{12} & 0 \\ r_{20} & r_{21} & r_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

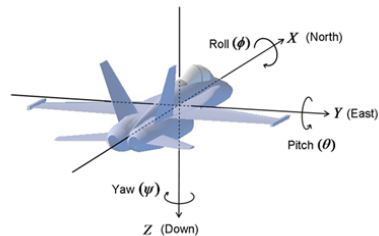
Idea: concatenating the rotations about global X, Y, Z, we can achieve arbitrary 3D rotation.

$\mathbf{R}(\mathbf{p}) = R(\theta_x)R(\theta_y)R(\theta_z)\mathbf{p}$ is a rotation of point \mathbf{p} in ZYX order.

General 3D Rotation about 3 major axes - Euler angles



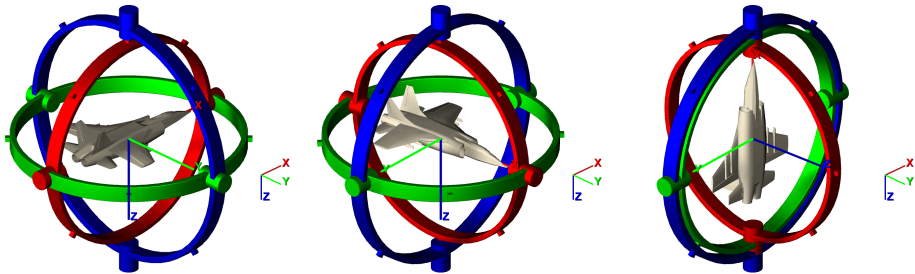
- ▶ Euler angles: 3 angles about 3 major axes
 - ▶ Local/body frame: intrinsic, e.g. Pitch, Yaw, Roll
 - ▶ Global/external frame: extrinsic
- ▶ $\mathbf{R} = R_a(\alpha)R_b(\beta)R_c(\gamma)$
- ▶ **Note:** for intrinsic rotation, matrix order is reversed: R_a 1st but multiplied last
 - ▶ We will revisit the topic in the next week.
- ▶ 12 orders of rotations, such as XYZ, XZX, [Wiki](#)



Euler Angles - Gimbal lock example

- ▶ A 90° second rotation \Rightarrow a third axis coincides with the first
- ▶ Lose a degree of freedom, a real gimbal gyroscope gets locked
- ▶ In CG: change multiple rotation angles to leave the state, but results in unintuitive motion

[Video of a real gimbal] [A Unity Gimbal lock demo]



$z = -x$ leading to a 2D rotation; for a real gimbal, Yaw is lost

More 3D Rotations - Rotate about arbitrary axis



► Rodrigues Formula

$$\mathbf{R} = \begin{bmatrix} \cos \theta + u_x^2(1 - \cos \theta) & u_x u_y(1 - \cos \theta) - u_z \sin \theta & u_x u_z(1 - \cos \theta) + u_y \sin \theta \\ u_y u_x(1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2(1 - \cos \theta) & u_y u_z(1 - \cos \theta) - u_x \sin \theta \\ u_z u_x(1 - \cos \theta) - u_y \sin \theta & u_z u_y(1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2(1 - \cos \theta) \end{bmatrix}$$

► Quaternions : good for smooth rotation interpolation

$$\mathbf{R} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_0 q_1 + q_2 q_3) \\ 2(q_0 q_2 + q_1 q_3) & 2(q_2 q_3 - q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

No need to worry, you only need to call glm APIs to generate the rotation matrix.

```
glm::mat4 glm::rotate(glm::radians(angle_in_degree), glm::vec3(axis_x, axis_y,  
axis_z));
```

The background consists of two large, overlapping geometric shapes. A teal-colored shape is in the upper-left corner, and a light gray shape is in the lower-left corner. They meet at a diagonal line that runs from the top-left towards the bottom-right. The rest of the background is white.

Composite Transforms

Composite Transforms



- ▶ Composite transform matrix can be calculated by multiplying simple transformation matrices.
 - ▶ $\mathbf{M} = \mathbf{M}_n \dots \mathbf{M}_i \dots \mathbf{M}_2 \mathbf{M}_1$
- ▶ Order of Matrix Multiplications (transforms)
 - ▶ OpenGL & GLSL (column-major matrices): right to left
 - ▶ $p' = \mathbf{T}_3 \mathbf{T}_2 \mathbf{T}_1 p = \mathbf{T}_3 (\mathbf{T}_2 (\mathbf{T}_1 p))$
 - ▶ Direct3D & HLSL (row-major matrices): left to right
 - ▶ $p'^T = p^T \mathbf{T}_1^T \mathbf{T}_2^T \mathbf{T}_3^T = ((p^T \mathbf{T}_1^T) \mathbf{T}_2^T) \mathbf{T}_3^T$

We will use the OpenGL convention throughout this module.

Composition of Translation and Rotation



A common transform composite is the Translation and Rotation couple.

If an object rotates followed by a translation in the world frame

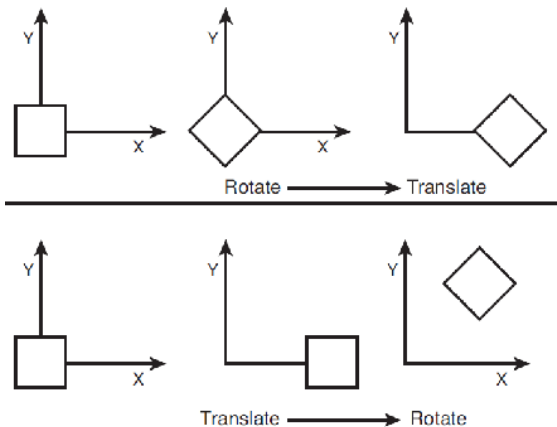
$$\mathbf{M} = T(-t_x, -t_y, -t_z)\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & 0 \\ r_{10} & r_{11} & r_{12} & 0 \\ r_{20} & r_{21} & r_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A combined translation and rotation can be easily written into one matrix.

Chained in hierarchical transforms (the next week) $\mathbf{M}_n = \mathbf{T}_1\mathbf{R}_1\ldots\mathbf{T}_i\mathbf{R}_i\ldots\mathbf{T}_{n-1}\mathbf{R}_{n-1}\mathbf{T}_n\mathbf{R}_n$

The Matrix Order Matters

- ▶ Be careful with transform/matrix orders.
- ▶ $\mathbf{TR} \neq \mathbf{RT}$, both can happen, depending on the reference frame.
- ▶ Matrix multiplication is NOT commutative!
- ▶ We will revisit the topic the next week.



Summary



- ▶ Representing transforms as matrices
 - ▶ Homogeneous coordinates
- ▶ Divide and conquer
 - ▶ Decompose complex transformations into simple ones
 - ▶ 3D rotations (HARD math, easy APIs)
 - ▶ Composite translation and rotation
- ▶ Matrix order matters

Questions?