5CM507 Graphics

Lecture A02b Transformation

Dr Youbing Zhao

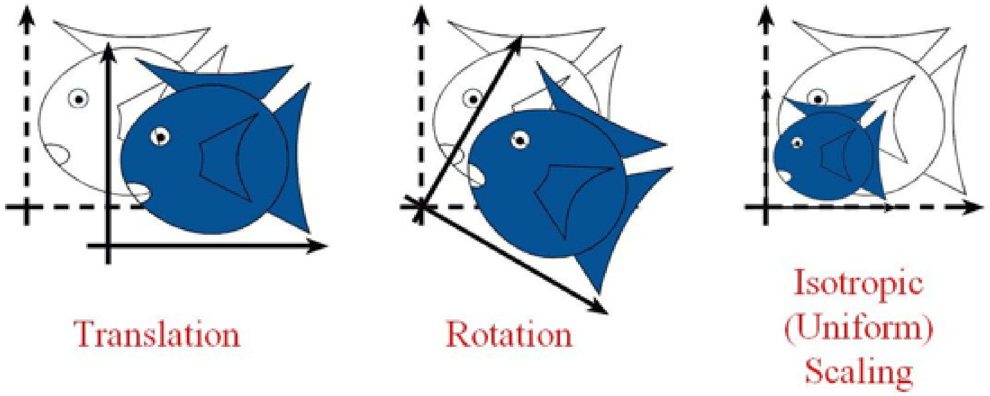
September 30, 2025

Contents

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

Common Transformations

- We need to find transformations that follow certain rules

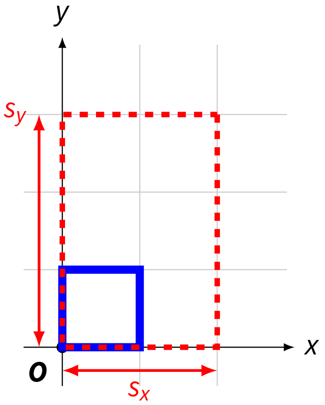


Transforms

2D and 3D Scaling

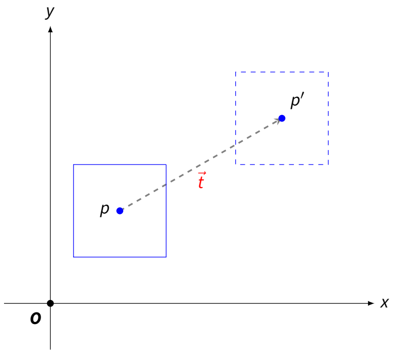
- If we want to scale a rigid object centred at the origin by a scale factor of $S = S(s_x, s_y)$
- that is to transform every point p on the object to p' with the following equations
  - $x' = s_x x \quad y' = s_y y$
- Matrix form: $\mathbf{p}' = \mathbf{S}\mathbf{p}$

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



2D and 3D Translation

- Move (translate, displace) a point to a new location
  - $x' = x + t_x \quad y' = y + t_y$
- 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

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

Idea: add an extra "scaling" dimension:

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

$$\mathbf{P}' = \begin{bmatrix} x' & y' & z' & w' \end{bmatrix}^T \iff [\tfrac{x}{w} \quad \tfrac{y}{w} \quad \tfrac{z}{w} \quad 1]^T$$

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

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

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

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

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

Now multiple transforms can be concatenated using matrix multiplication.

2D and 3D rotations

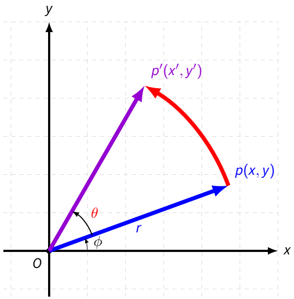
2D Rotation on *XY* plane to 3D rotation about *Z*

- 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 $z' = z$ => 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:

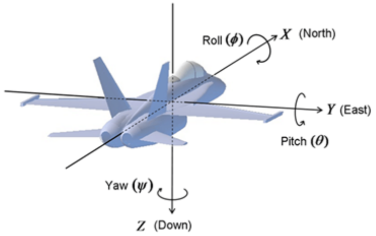
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{00} & 0 \\ r_{10} & r_{11} & r_{12} & 0 \\ r_{20} & r_{21} & r_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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 **p** in *ZYX* order.

General 3D Rotation about 3 major axes - Euler angles

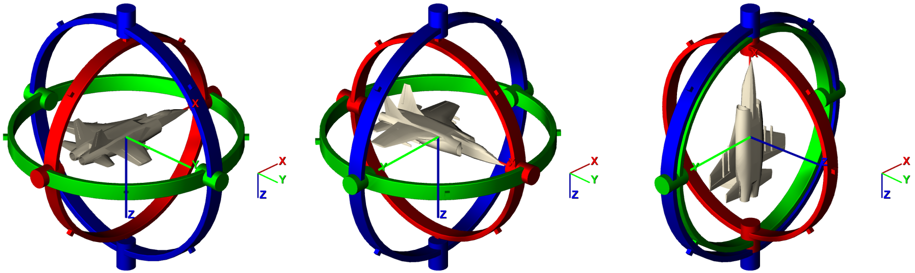
- Euler angles: 3 angles about 3 major axes
  - Local/body frame: intrinsic, e.g. Pitch, Yaw, Roll
  - Global/external frame: extrinsic
- $\mathbf{R} = R_a(\alpha)R_b(\beta)R_c(\gamma)$
- **Note**: for intrinsic rotation matrix order is reversed: R_a 1st but multiplied last
  - Why: imagine transforming the final body/local frame back to the original world frame
- 12 orders of rotations, such as XYZ, XZX, Wiki



Euler Angles - Gimbal lock example

- A 90° second rotation => a third axis coincides with the first
- Lose a degree of freedom, a real gimbal gyroscope get locked
- For graphics: change multiple rotation angles to leave the state, but result 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

$$\mathbf{R} = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2\,(q_1q_2+q_0q_3) & 2\,(q_1q_3-q_0q_2) \\ 2\,(q_1q_2-q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2\,(q_0q_1+q_2q_3) \\ 2\,(q_0q_2+q_1q_3) & 2\,(q_2q_3-q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix}$$

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));
```

Summary

- Representing affine transformations as matrices
  - Homogeneous coordinates
- Divide and conquer
  - Decompose complex transformations into simple ones
  - 3D rotations
  - Hierarchical modelling (the next week)
- 3D rotations are HARD, but APIs are easy to use
- Matrix order is critical

Questions?