

Modeling Transformation

Viewing Transformation

2D Transformation

Linear Transformation

$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$	$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$
Shear Matrix $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$ <p>Hints: Horizontal shift is 0 at $y=0$ Horizontal shift is a at $y=1$ Vertical shift is always 0</p>	Rotation Matrix $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Linear Transformation Conclusion

Linear Transforms = Matrices

(of the same dimension)

$$x' = ax + by$$

$$y' = cx + dy$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = M \cdot x$$

Homogeneous Coordinates 仿次坐标

Translation??

$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ $x' = x + t_x$ $y' = y + t_y$	Solution: Homogenous Coordinates Add a third coordinate (w -coordinate) <ul style="list-style-type: none"> • 2D point = $(x, y, 1)^T$ • 2D vector = $(x, y, 0)^T$ <p>Matrix representation of translation:</p> $\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$ <p>What if you translate a vector?</p>
---	--

Why Homogeneous Coordinates

- Translation cannot be represented in matrix form
- But we don't want translation to be a special case
- Is there a unified way to represent all transformations? (and what's the cost?)

2D Transformations

Affine map = linear map + translation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Using homogenous coordinates:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Always $x, y \neq 0$ for translation

一个矩阵

two matrices

one matrix

Scale

$$S(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation

$$R(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Translation

$$T(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Homogenous Coordinates

Valid operation if w -coordinate of result is 1 or 0

- vector + vector = vector
- point + vector = vector
- point + vector = point
- point + point = ??

In homogeneous coordinates,

$\begin{pmatrix} x \\ y \\ w \end{pmatrix}$ is in the 2D point $\begin{pmatrix} x/w \\ y/w \end{pmatrix}$ w $\neq 0$

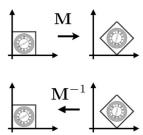
如果 $w=0$

Inverse Transformation

Inverse Transform

M^{-1}

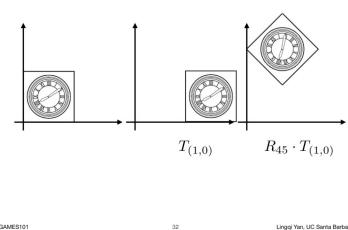
M^{-1} is the inverse of transform M in both a matrix and geometric sense



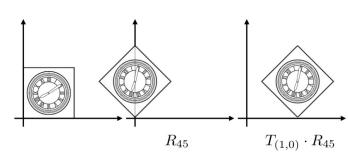
Composing Transform

ORDER MATTERS!

Translate Then Rotate?



Rotate Then Translate



Transform Ordering Matters!

Matrix multiplication is not commutative

$$R_{45} \cdot T_{(1,0)} \neq T_{(1,0)} \cdot R_{45}$$

Note that matrices are applied right to left:

$$T_{(1,0)} \cdot R_{45} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

TRANSLATION ROTATE

Composing Transforms

Sequence of affine transforms A_1, A_2, A_3, \dots

- Compose by matrix multiplication
- Very important for performance!

$$A_n(\dots A_2(A_1(x))) = A_n \cdots A_2 \cdot A_1 \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

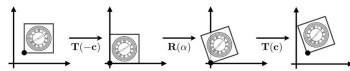
Pre-multiply n matrices to obtain a single matrix representing combined transform

Decomposing Transformation

Decomposing Complex Transforms

How to rotate around a given point c ?

1. Translate center to origin
2. Rotate
3. Translate back



Matrix representation?

$$T(c) \cdot R(\alpha) \cdot T(-c)$$

3D Transformation

3D Transformations

Use homogeneous coordinates again:

- 3D point = $(x, y, z, 1)^T$
- 3D vector = $(x, y, z, 0)^T$

In general, (x, y, z, w) ($w \neq 0$) is the 3D point:

$$(x/w, y/w, z/w)$$

Homogeneous Coordinate in 3D

3D Transformations

Use 4×4 matrices for affine transformations

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

What's the order?

Linear Transform first or Translation first?



Scale and Translation

Scale

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

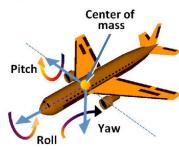
Compose any 3D rotation from R_x, R_y, R_z ?

$$R_{xyz}(\alpha, \beta, \gamma) = R_x(\alpha) R_y(\beta) R_z(\gamma)$$



- So-called Euler angles

- Often used in flight simulators: roll, pitch, yaw



Rotation around x-, y-, or z-axis

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y(\alpha) = \begin{pmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\mathbb{Z} \times \mathbb{Z} \Rightarrow \mathbb{Y}$
not counter-clockwise $\Rightarrow \mathbb{X} \times \mathbb{Z}$
so flip \mathbb{X} and $\mathbb{Z} (\pi)$

Anything strange about R_y ?

Rodrigues' Rotation Formula

Rodrigues

Rotation by angle α around axis n

$$R(n, \alpha) = \cos(\alpha) I + (1 - \cos(\alpha)) nn^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_N$$

How to prove this magic formula?

Check out the supplementary material on the course website!

Viewing Transformation

View / Camera transformation

- What is view transformation?
- Think about how to take a photo
 - Find a good place and arrange people (**model** transformation)
 - Find a good "angle" to put the camera (**view** transformation)
 - Cheese! (**projection** transformation) 将面部换成一张面庞

MVP
→
渲染过程

Viewing (观測) transformation

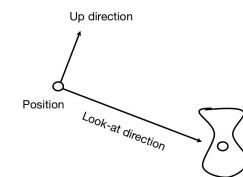
- View (视图) / Camera transformation
- Projection (投影) transformation
- Orthographic (正交) projection
- Perspective (透视) projection

View / Camera Transformation

- How to perform view transformation?

- Define the camera first

- Position \vec{e}
- Look-at / gaze direction \hat{g}
- Up direction \hat{t} (assuming perp. to look-at)



Key observation

- If the camera and all objects move together, the "photo" will be the same



How about that we always transform the camera to

- The origin, up at Y, look at -Z
- And transform the objects along with the camera

M_{view} in math?

- Let's write $M_{view} = R_{view} T_{view}$

- Translate e to origin

$$T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotate g to $-Z$, t to Y , $(g \times t)$ to X

- Consider its inverse rotation: X to $(g \times t)$, Y to t , Z to $-g$

$$R_{view}^{-1} = \begin{bmatrix} x_{\hat{g} \times \hat{t}} & x_t & x_{-g} & 0 \\ y_{\hat{g} \times \hat{t}} & y_t & y_{-g} & 0 \\ z_{\hat{g} \times \hat{t}} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{WHY?} \quad R_{view} = \begin{bmatrix} x_{\hat{g} \times \hat{t}} & y_{\hat{g} \times \hat{t}} & z_{\hat{g} \times \hat{t}} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

△ 旋转矩阵的逆就是它的转置 < 正交矩阵的性质 >



将摄像机及其环境变换至原点位置

视图变换

Summary

- Transform objects together with the camera
- Until camera's at the origin, up at Y, look at -Z

Also known as ModelView Transformation

- But why do we need this?

- For projection transformation!

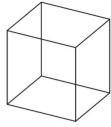
↙ ↘

Projection Transformation

- Projection in Computer Graphics

- 3D to 2D
- Orthographic projection
- Perspective projection

Orthographic projection

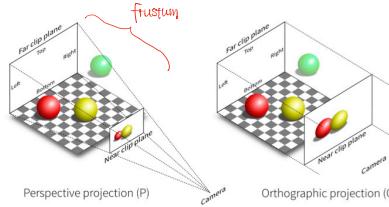


Perspective projection



Fig. 7.1 from Fundamentals of Computer Graphics, 4th Edition

- Perspective projection vs. orthographic projection



Projection

Perspective

Orthographic Projection

A simple way of understanding
- Camera located at origin, looking at Z-axis (x, y, z more familiar)
- Drop coordinates
Transform to create the resulting rectangle 0-1-1-0

Argument
- We want to map a coordinate system to another one
- We want to map a coordinate system to another one
- We want to map a coordinate system to another one

Transformation matrix
- Translate center to origin (0, 0, 0) and make length scale to 1
 $M_{ortho} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Before we move on
• Recall property of homogeneous coordinates
- $(x, y, z, 1)$ has the same meaning as $(x, y, z, 0)$. (i.e., $x, y, z, 1$ is better)
- It's easier to work with $(x, y, z, 1)$
- $(x, y, z, 1)$ and $(x, y, z, 0)$ both represent (x, y, z)
- Simple, but useful

How to do perspective projection
- First "map" the frustum into a cuboid ($0 \leq x \leq 1, 0 \leq y \leq 1, 0 \leq z \leq 1$)
- Do orthographic projection (M_{ortho} , already known)

In order to find a transformation
- Find the relationship between transformed points (x', y') and original points (x, y, z)

So the "inverse" (perspective to orthographic) projection does this
 $M_{proj \rightarrow ortho}^{-1} = \begin{bmatrix} \frac{x}{z} & \frac{y}{z} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

In homogeneous coordinates:
 $\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x/z \\ y/z \\ 1 \\ 1 \end{pmatrix}$

Observation: the third row is responsible for z
- Any point on the near plane will change
- Any point on the far plane will not change

2D 和 3D 空间为同维条件推导
the 2D hom mat is $\begin{pmatrix} 0 & 0 & A & B \end{pmatrix}$

Premise ① NEAR $\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \Rightarrow z = 1$

Therefore $\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = Ax + B = n^2$

Premise ② CENTER OF FAR $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \Rightarrow z = 1$

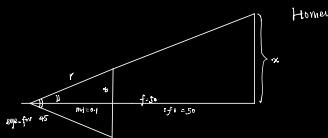
Therefore $\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = Af + B = f^2$

$$\begin{cases} An + B = n^2 \\ Af + B = f^2 \end{cases} \Rightarrow \begin{cases} A = n^2 - f^2 \\ B = -nf \end{cases}$$

n, f 为坐标 (有负)

$$M_{proj \rightarrow ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & nf - n^2 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Homework 1 Note



Aspect ratio $\frac{n}{h} = 1$

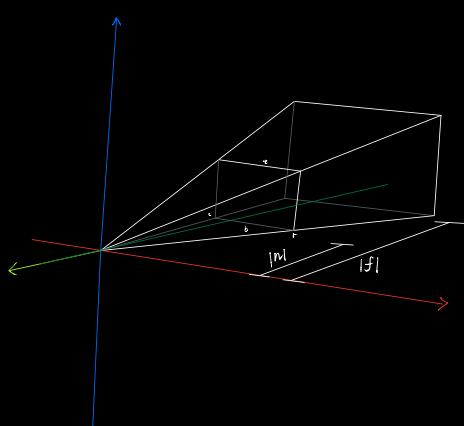
Transformation matrix!

- Translate source to origin first, then scale length width height to 1

$$M_{ortho} = \begin{pmatrix} 1/n & 0 & 0 & 0 \\ 0 & 1/h & 0 & 0 \\ 0 & 0 & 1/f & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\tan(\frac{\text{eye_fov}}{2}) = \frac{n}{\text{rel}} * \frac{\text{width}}{\text{height}}$$

$$\tan(\frac{\text{eye_fov}}{2}) =$$



What's near plane's l, b, t then?

- Explicitly specified good
- Sometimes people prefer:
- eye of view (fov)
- aspect ratio (width/height)

How to convert from $fovY$ and aspect to l, t, b, f ?

$$\tan(\frac{fovY}{2}) = \frac{t - b}{2 * \text{rel}}$$

$$\text{aspect} = \frac{w}{h}$$

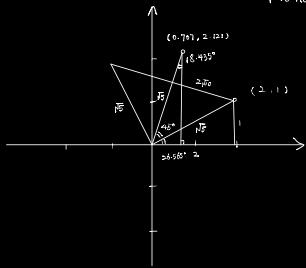
$$l = -\text{rel} * \frac{w}{2}$$

$$t = \text{rel} * \frac{w}{2}$$

$$b = -\text{rel} * \frac{h}{2}$$

$$f = \text{rel} * \frac{h}{2}$$

Homework 0 Note



由入射点为原点条件推得

$$\text{Premise } \oplus \text{ NEAR} \quad \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} = An + Bz = n^*$$

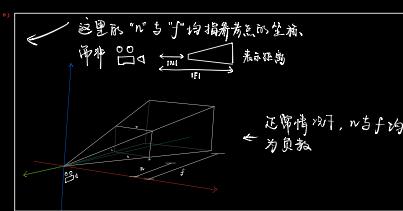
$$\text{Therefore, } (0 + A + B) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = An + Bz = n^*$$

$$\text{Premise } \ominus \text{ center of FRM} \quad \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = Af + Bz = f^*$$

$$\text{Therefore, } (0 + A + B) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = Af + Bz = f^*$$

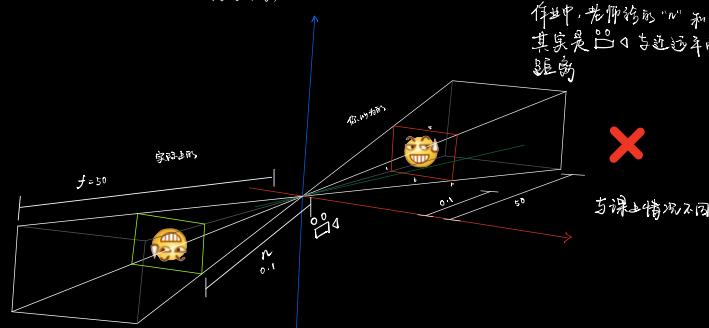
$$\begin{cases} An + Bz = n^* \\ Af + Bz = f^* \end{cases} \Rightarrow \begin{cases} A = n + f \\ B = -nf \end{cases}$$

$$M_{\text{perspective}} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & nf & -nf \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



```
while (key != 27) {
    r.clear(rst::Buffers::Color | rst::Buffers::Depth);
    r.set_model(get_model_matrix(axis, angle));
    r.set_view(get_view_matrix(eye_pos));
    r.set_projection(get_projection_matrix(45, 1, [0.1, 50]));
```

作业中, 老师说的“n”和“f”
其实是 Δ 与 Δ 与近远平面之
距离



因此, 直接套用公式的话会出错颠倒

解决方法

①

```
Eigen::Matrix4f perspective = Eigen::Matrix4f::Identity();
perspective << zNear, 0, 0, 0,
            0, zNear, 0, 0,
            0, 0, zNear + zFar, -(zNear * zFar),
            0, 0, 1, 0;
```

改写 perspective to OpenGL 程序的值
以达到手动填符号的效果

②

```
while (key != 27) {
    r.clear(rst::Buffers::Color | rst::Buffers::Depth);
    r.set_model(get_model_matrix(axis, angle));
    r.set_view(get_view_matrix(eye_pos));
    r.set_projection(get_projection_matrix(45, 1, [-0.1, -50]));
```

修改输入以使程序与课程情况一致

```
// calculate the sizes of the frustum
eye_fov      = (eye_fov / 180) * MY_PI;
float yTop   = abs(zNear) * tan(eye_fov / 2);
float yBottom = -yTop;
float xRight = yTop * aspect_ratio;
float xLeft  = -xRight;
```

注意, 若修改了输入, 此处要加绝对值