Assignment1

Task

实现三角形显示，并实现任意角度任意轴旋转

需要补全的函数

- Eigen::Matrix4f get_model_matrix(float rotation_angle)
- Eigen::Matrix4f get_projection_matrix(float eye_fov, float aspect_ratio,float zNear, float zFar)
- Eigen::Matrix4f get_rotation_matrix(Vector3f axis, float angle)

Method

MVP操作

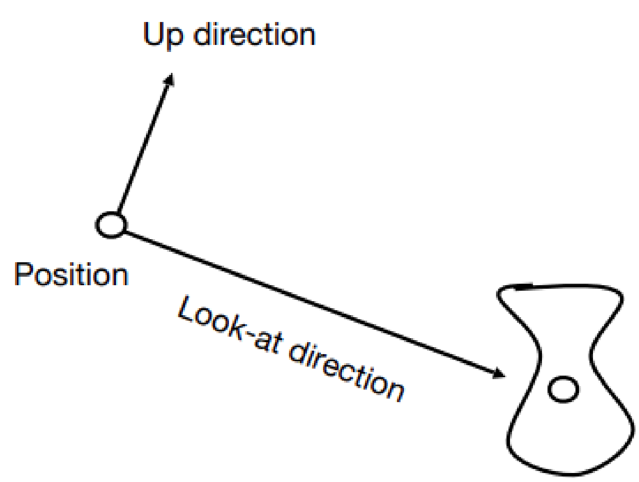
- Find a good place and arrange people (**model** transformation)
- Find a good “angle” to put the camera (**view** transformation)
- Cheese! (**projection** transformation)

Model transformation和View transformation经常被一起叫作模型视图变换（ModelView Translation）

Model transformation

View transformation

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



将相机摆放到固定位置，GAMES101中默认为原点位置，-z轴为gaze direction，y轴为up direction

其transformation思路为

- 先将e平移到原点
- 然后对其进行旋转，使得-z轴为gaze direction，y轴为up direction

- 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 x t) To X
  - Consider its inverse rotation: X to (g x t), Y to t, Z to -g

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

Projection transformation

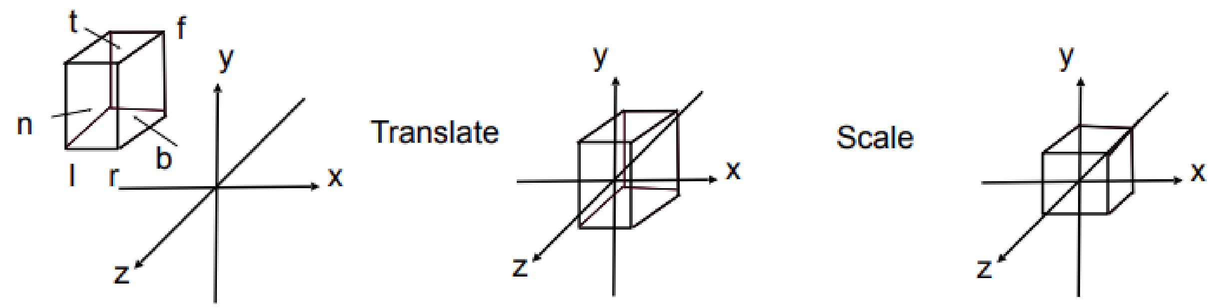
Projection transformation分为

- Perspective Projection：将透视投影转化为正交投影
- Orthographic Projection：将正交投影转换到正则立方体

Orthographic Projection

将正交投影转换到正则立方体

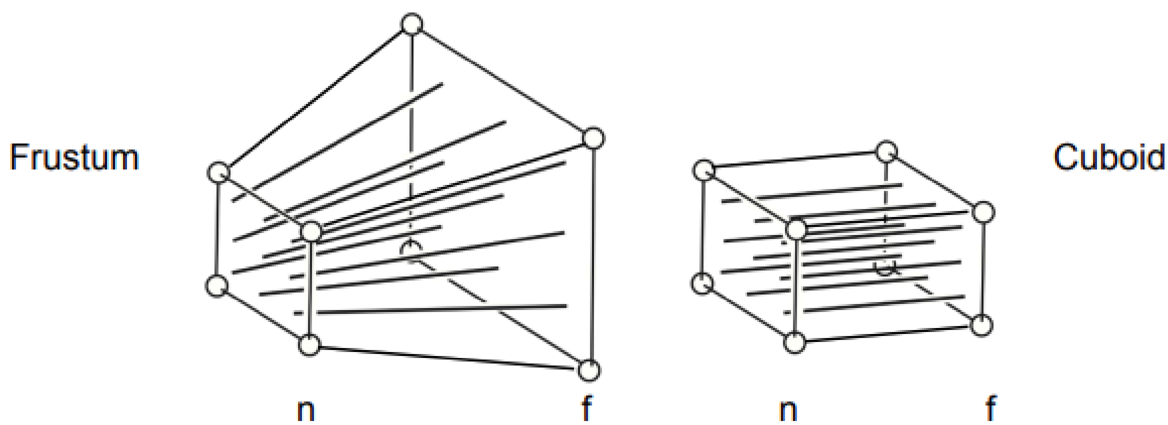
- In general
  - We want to map a cuboid [l, r] x [b, t] x [**f, n**] to the “canonical (正则、规范、标准)” cube $[-1, 1]^3$



最后变化矩阵为

$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection



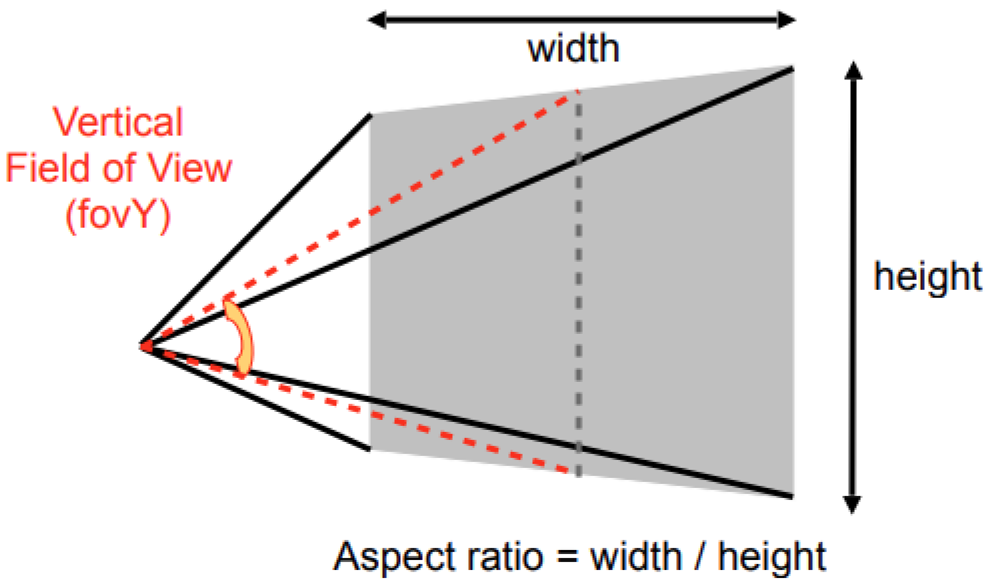
最后的Perspective Projection为

$$M_{persp \to ortho} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

视锥

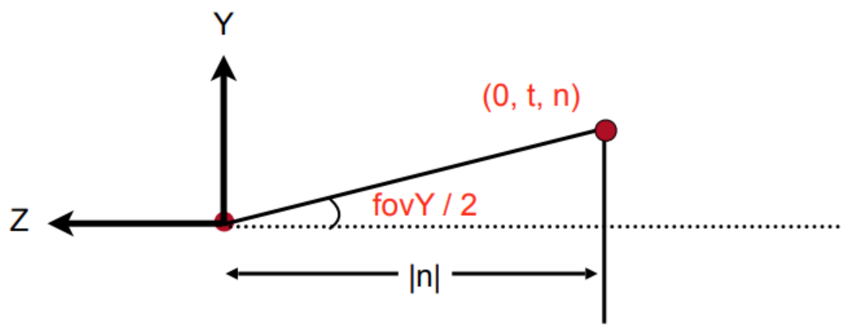
定义视锥：

- 长宽比 Aspect
- 垂直的角度 FovY



因此可以得到

- Trivial



$$\tan\frac{fovY}{2} = \frac{t}{|n|}$$

$$aspect = \frac{r}{t}$$

Viewport transformation

将规划化立方体投射到屏幕的变换

接下来，看看如何将(-1,1)规范化坐标范围映射到nx*ny个像素组成的屏幕坐标，可以通过缩放再平移的方式来实现：

1. 缩放：x、y轴从2、2放大到nx、ny，因此缩放比例为nx/2, ny/2;
2. 平移: 缩放后的原点位置不变，这个时候需要通过平移将左下角的点移动到屏幕坐标系的坐标原点，x、y轴向正向移动的距离分别为nx/2-0.5、ny/2-0.5，之所以有这个0.5是因为屏幕坐标原点为左下角像素（一个像素相当于一个很小的矩形）的中心点。

$$Mvp = \begin{bmatrix} nx/2 & 0 & 0 & (nx-1)/2 \\ 0 & ny/2 & 0 & (ny-1)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Code

get_projection_matrix

```
Eigen::Matrix4f get_projection_matrix(float eye_fov, float aspect_ratio,
                                      float zNear, float zFar)
{
    // Students will implement this function

    Eigen::Matrix4f projection = Eigen::Matrix4f::Identity();

    // TODO: Implement this function
    // Create the projection matrix for the given parameters.
    // Then return it.

    float fovY = eye_fov / 180 * MY_PI;

    //立方体六个面
    float n, f, r, l, t, b;
        n = zNear, f = zFar;
    t = tan(fovY / 2) * (-n);
    b = -t;
    r = aspect_ratio * t;
    l = -r;

    //透视转换为正交
    Eigen::Matrix4f Mp_to_o;
    Mp_to_o << n, 0, 0, 0,
        0, n, 0, 0,
        0, 0, n + f, -n * f,
        0, 0, 1, 0;
    //正交归一化归一化
    Eigen::Matrix4f Mo, Mt, Ms;
    Mt << 1, 0, 0, -(r + l) / 2,
        0, 1, 0, -(t + b) / 2,
        0, 0, 1, -(n + f) / 2,
        0, 0, 0, 1;
    Ms << 2 / (r - l), 0, 0, 0,
        0, 2 / (t - b), 0, 0,
        0, 0, 2 / (n - f), 0,
        0, 0, 0, 1;
    Mo = Ms * Mt;
    projection = Mo * Mp_to_o;
    return projection;
}
```

get_model_matrix

```
Eigen::Matrix4f get_model_matrix(float rotation_angle)
{
    Eigen::Matrix4f model = Eigen::Matrix4f::Identity();
```

```
    // TODO: Implement this function
    // Create the model matrix for rotating the triangle around the Z axis.
    // Then return it.

    float angle = rotation_angle / 180 * MY_PI;

    model << cos(angle), -sin(angle), 0, 0,
        sin(angle), cos(angle), 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1;


    return model;
}
```