COSE436

Lecture 7: Viewing

Sep 26, 2024

Won-Ki Jeong
(wkjeong@korea.ac.kr)

KOREA UNIVERSITY

Overview

- Mathematics of projection
- Coordinate transformation pipeline
- Derive GL projection matrices

Computer Viewing

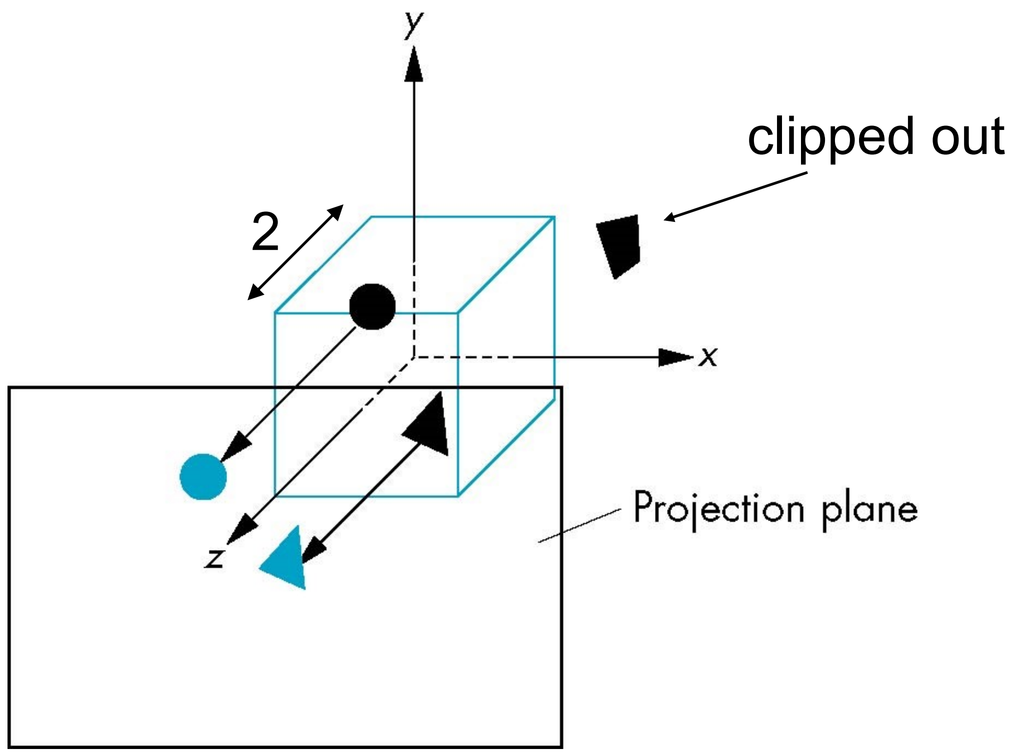
- There are three aspects of the viewing process implemented in the graphics pipeline
  - Placing objects, positioning the camera
    - Setting the ***model-view matrix***
  - Selecting a lens
    - Setting the ***projection matrix***
  - Clipping
    - Setting the ***view volume***

The GL Camera

- In GL, initially the object and camera frames are the same
- GL specifies a default view volume that is a cube with sides of length 2 centered at the origin
- If the modelview matrix is an identity, then the camera is located at origin and points to the negative z direction
- If the default projection matrix is an identity, then it is orthogonal projection

Default Camera & Projection

- Orthogonal



Moving the Camera Frame

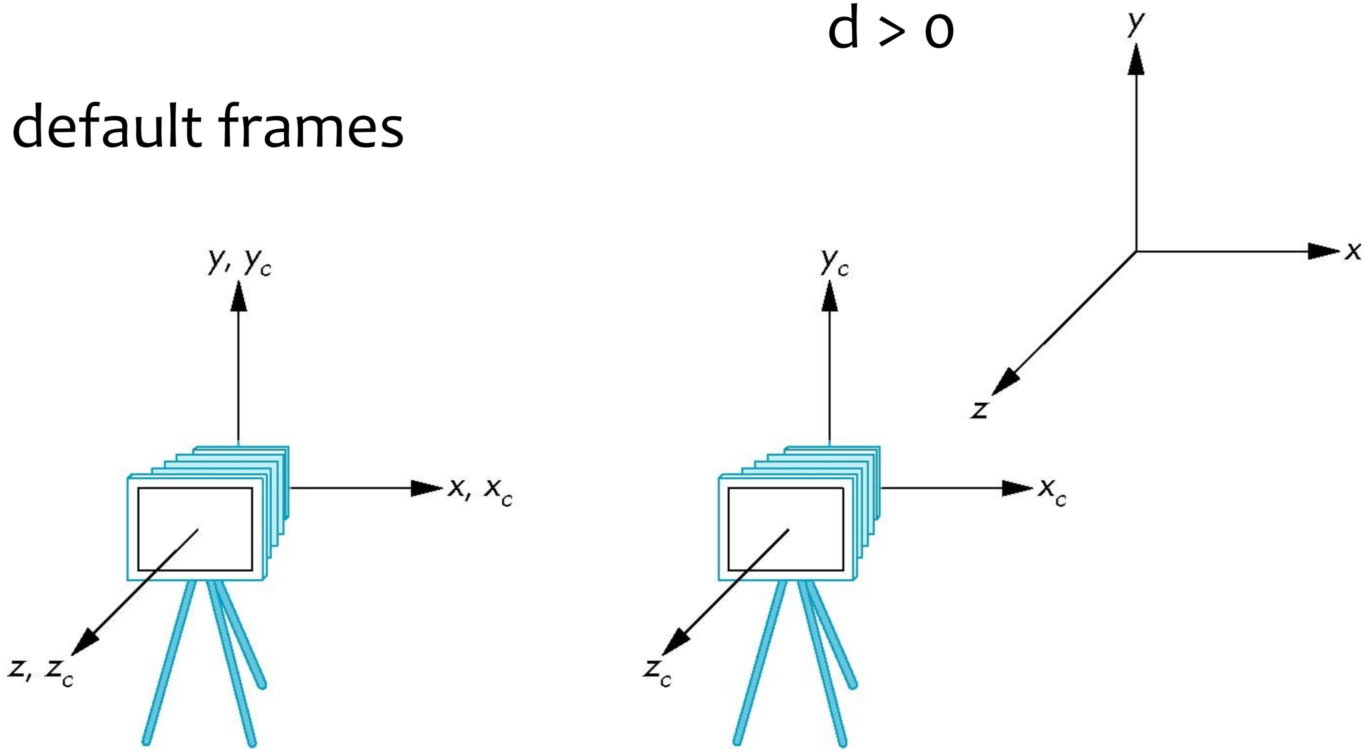
- If we want to visualize object with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame
- Both of these views are equivalent and are determined by the model-view matrix
  - Want a translation (`Translate(0.0,0.0,-d);`)
  - `d > 0`

Moving the Camera

frames after translation by –d

d > 0

default frames



(a)

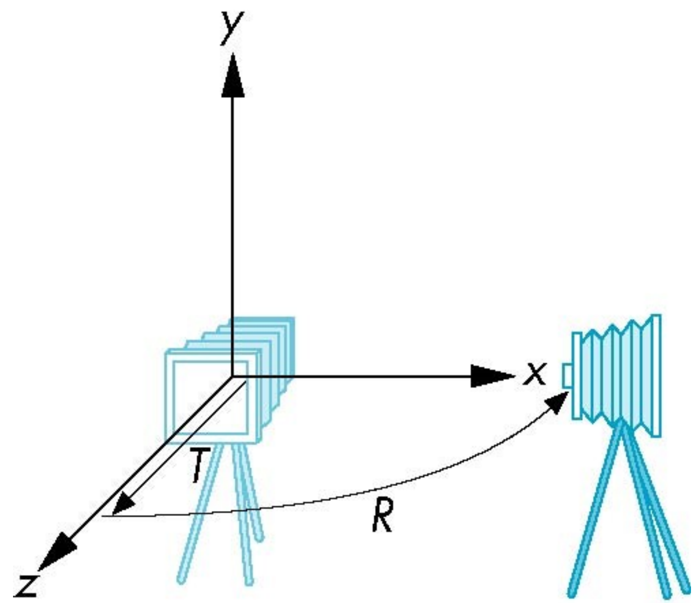
(b)

Example

- Point (10, 20 -30)
- Place camera at (0,0,10)
  - Translate camera 10 along z or,
  - Translate point -10 along z
  - In modelview matrix, we move points
- Point coordinate in camera frame (relative to the camera origin)
  - (10,20,-40)

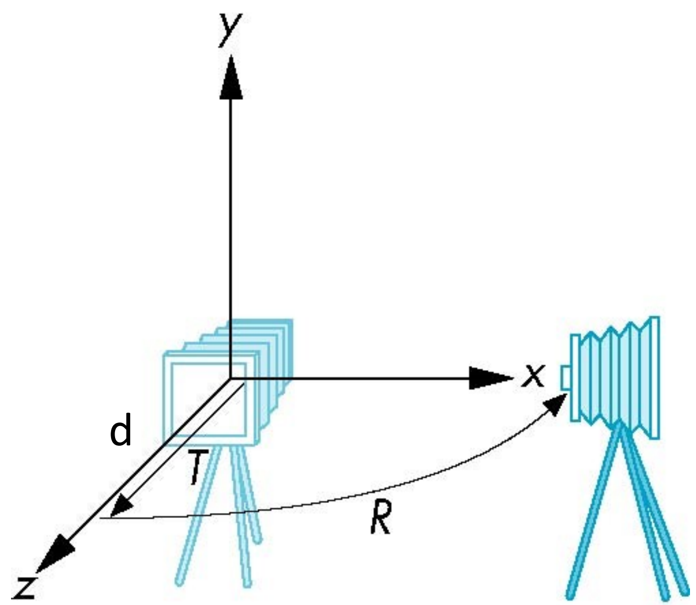
Moving the Camera

- We can move the camera to any desired position by a sequence of rotations and translations
- Example: side view
  - Rotate the camera
  - Move it away from origin
  - Model-view matrix C = TR



Moving the Camera

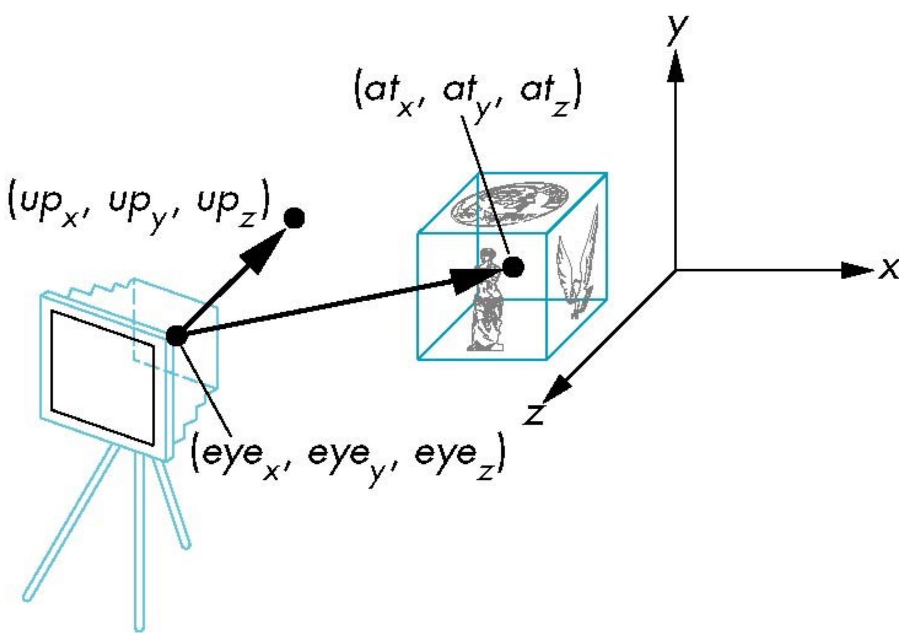
- Camera is moving
  - RotY(90)->TrX(d) = TrX(d)*RotY(90), or
  - TrZ(d)->RotY(90) = RotY(90)*TrZ(d)
- Object is moving
  - Inverse of camera movement
  - TrZ(-d)*RotY(-90)



Note A*B means B->A order!

lookAt Function

- mat.h provides a function to form the modelview matrix through a simple interface
- lookAt() does the same job
  - Can concatenate with modeling transformations



From 3D to 2D in GL

- The default projection in the eye (camera) frame is orthogonal
- For points within the default (canonical) view volume

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$

- How to apply Parallel or Perspective?

Orthographic Projection

- Homogeneous coordinate form

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$
$$w_p = 1$$

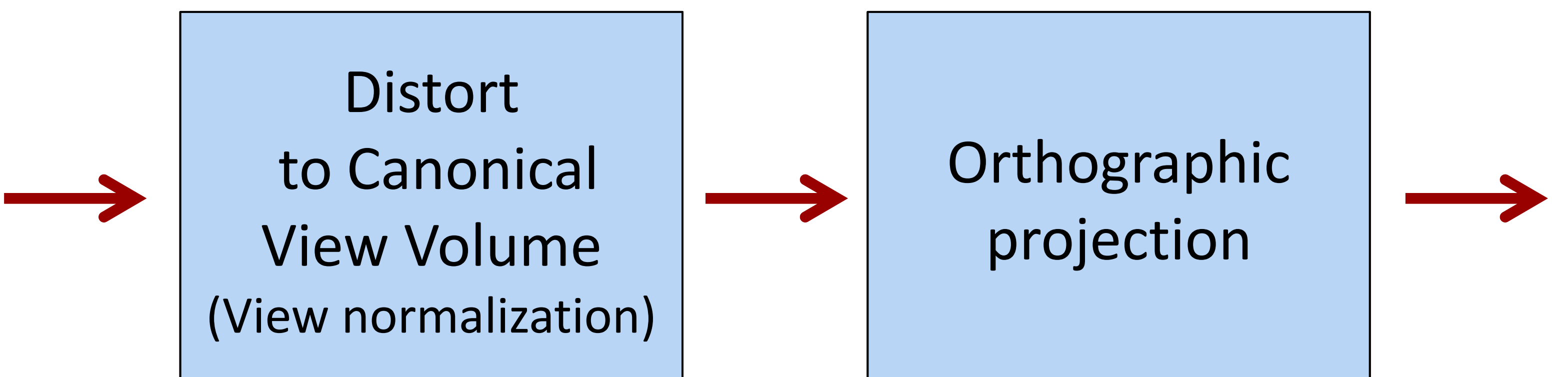
$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In practice, we can let **M** = **I** and set the *z* term to zero later

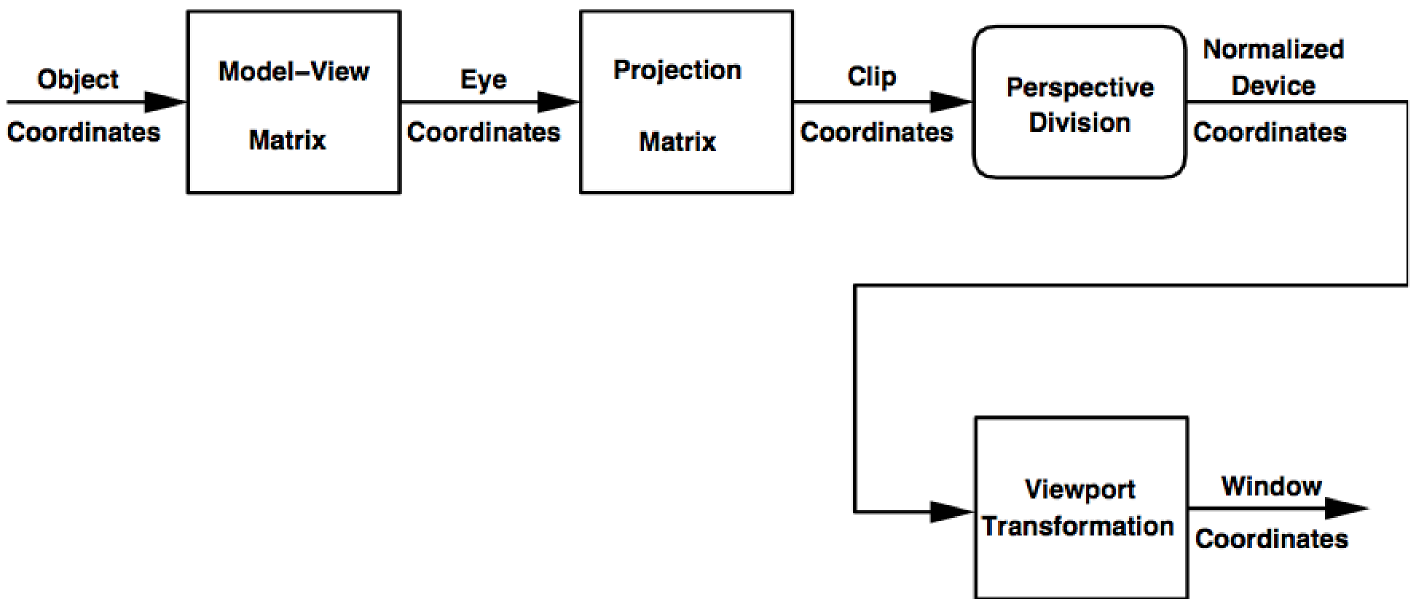
Projection Normalization

- GL uses *view normalization*
  - All other views are converted to the default view by transformations that determine the projection matrix
  - Use **canonical view volume** (x,y,z = ±1)
  - Allows use of the <u>same pipeline</u> for all views



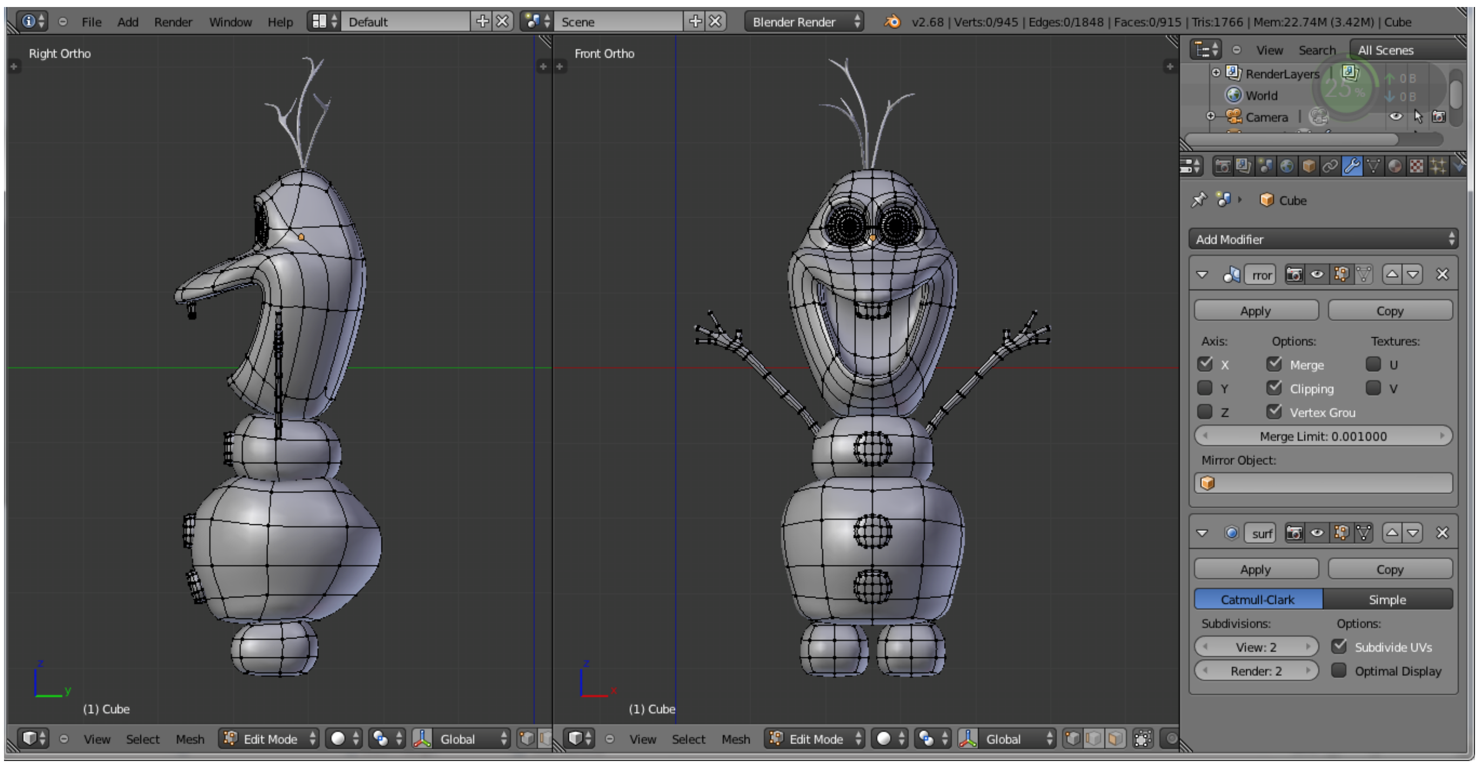
Coordinate Transformations

- Object coordinate
- Eye coordinate
- Clip coordinate
- Normalize device coordinate
- Windows coordinate



Object Coordinate

- Local frame per each object
- 3D coordinate



Eye Coordinate

- How objects are located relative to my eye
  - Coordinate in camera (eye) frame
- Modelview matrix
  - Modeling transformation (M_T), and
  - Viewing transformation (M_V)

$$\mathbf{p}_{eye} = \mathrm{M_V M_T} \mathbf{p}_{obj}$$

Modeling Transformation

- Per-object affine transformation
- Translate, rotate, scale, shear, ...
- Homogeneous coordinate

$$M_T = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformation

- Camera position and orientation
- Matrix for lookAt(E,C,U) E:eye, C:at, U:up

$$f = \frac{(C - E)}{\|(C - E)\|}, \quad s = \frac{f \times U}{\|f \times U\|}, \quad u = \frac{s \times f}{\|s \times f\|}$$

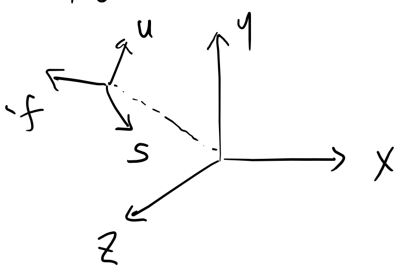
?

Viewing Transformation

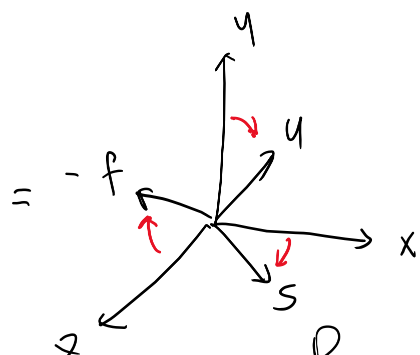
- Camera position and orientation
- Matrix for lookAt(E,C,U) E:eye, C:at, U:up

$$f = \frac{(C-E)}{\|(C-E)\|},\quad s = \frac{f \times U}{\|f \times U\|},\quad u = \frac{s \times f}{\|s \times f\|}$$

* Move Camera

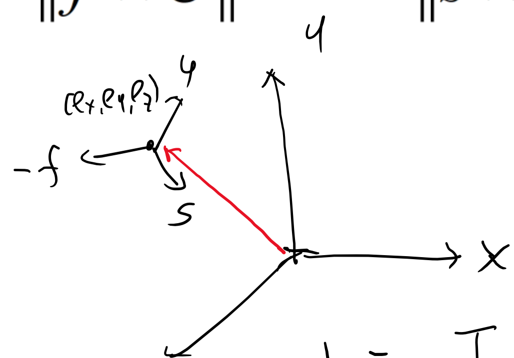


$\therefore T \cdot R$



Rotation R

$$\begin{pmatrix} | & | & | & 0 \\ s & u & -f & 0 \\ | & | & | & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Translation T

$$\begin{pmatrix} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

* Move point(object)

$$(T \cdot R)^{-1}$$

$$= R^{-1} T^{-1}$$

$$= R^{T} \cdot \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Viewing Transformation

- Camera position and orientation
- Matrix for lookAt(E,C,U) E:eye, C:at, U:up

$$f = \frac{(C-E)}{\|(C-E)\|}, \quad s = \frac{f \times U}{\|f \times U\|}, \quad u = \frac{s \times f}{\|s \times f\|}$$

$$\mathrm{M_V} = \begin{bmatrix} s_x & s_y & s_z & -s_x e_x - s_y e_y - s_z e_z \\ u_x & u_y & u_z & -u_x e_x - u_y e_y - u_z e_z \\ -f_x & -f_y & -f_z & f_x e_x + f_y e_y + f_z e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

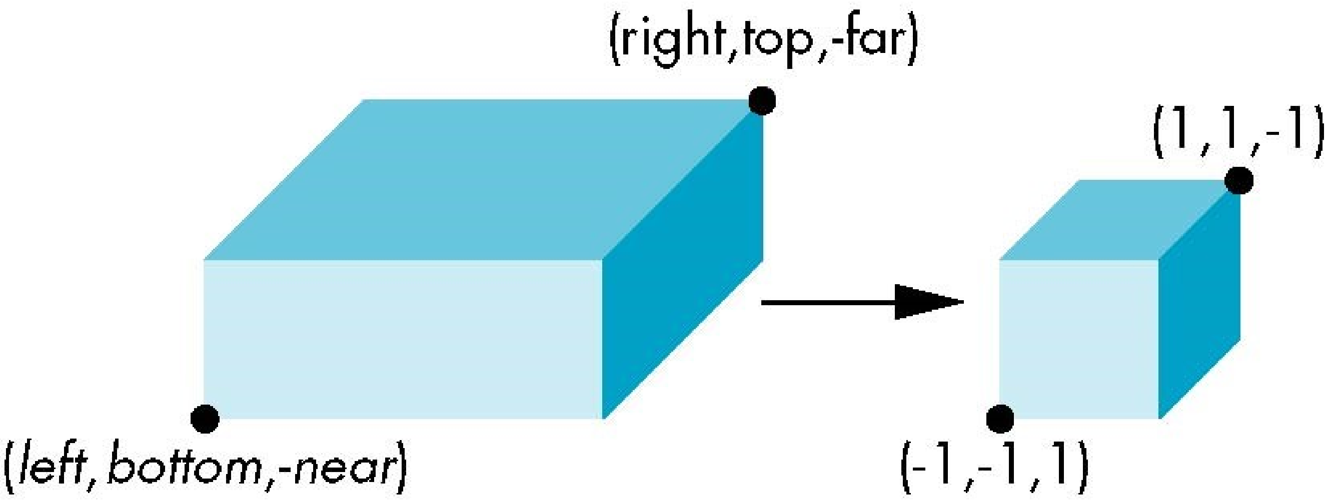
Clip Coordinate

- Projection matrix
  - Orthogonal projection
  - Oblique projection
  - Perspective projection
- Clipping happens in this coordinate
- Homogeneous coordinate

$$\begin{aligned} \mathbf{p}_{clip} &= \mathrm{M_P}\mathbf{p}_{eye} \\ &= \mathrm{M_P M_V M_T}\mathbf{p}_{obj} \end{aligned}$$

Orthogonal Projection

- Mapping view volume to normalized cube
- Ortho() in mat.h



Orthogonal Projection

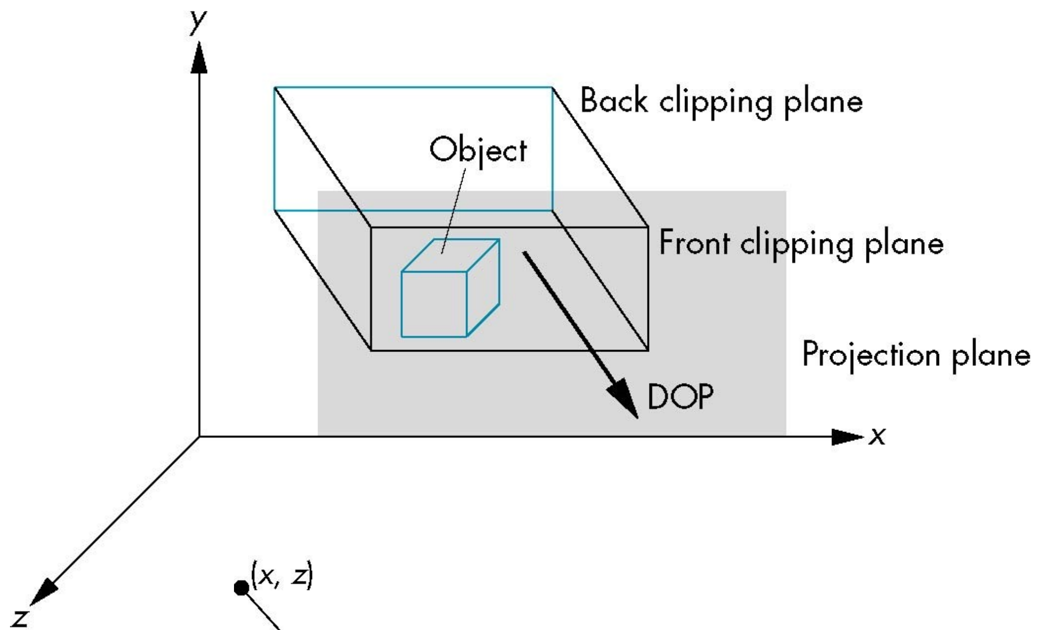
- Move center to origin
  - T(-(left+right)/2, -(bottom+top)/2,(near+far)/2))
- Scale to have sides of length 2
  - S(2/(right-left),2/(top-bottom),2/(near-far))

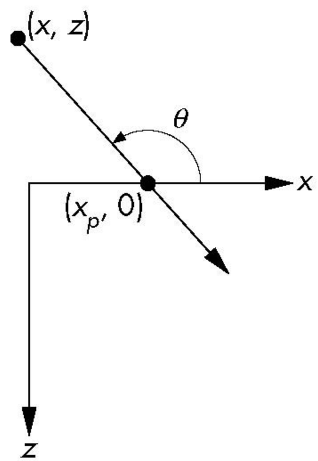
$$M_P = ST = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Oblique Projection

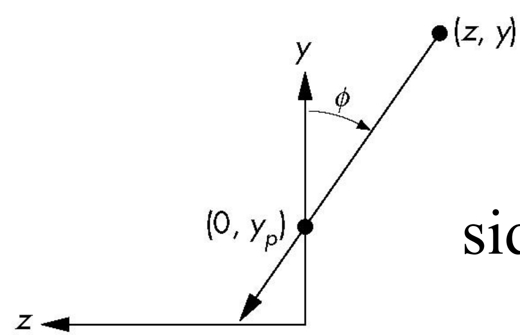
- General parallel projection
  - Projector does not need to be orthogonal to projection plane
- If we look at the example of the cube, it appears that the cube has been sheared
- Oblique Projection = Shear + Orthogonal Projection

General Shear





top view



side view

Oblique Projection

- Shear matrix

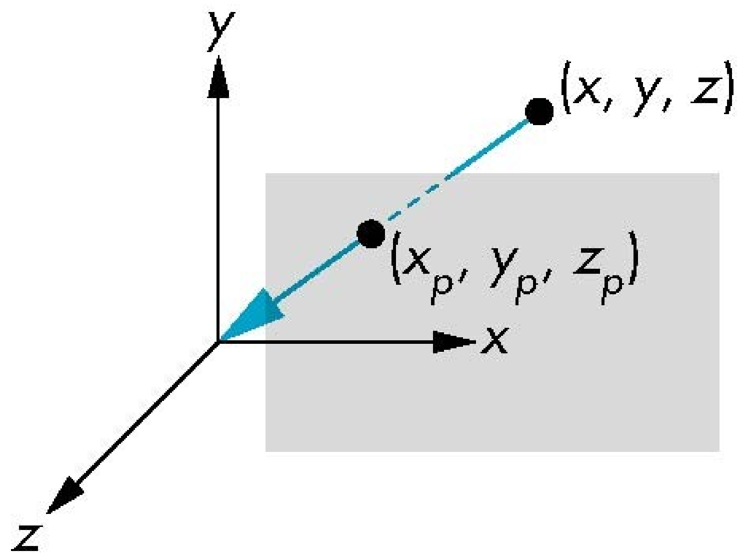
$$M_s = \begin{bmatrix} 1 & 0 & \cot\theta & 0 \\ 0 & 1 & \cot\varphi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- General oblique projection matrix

$$M_p = STM_s = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cot\theta & 0 \\ 0 & 1 & \cot\varphi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

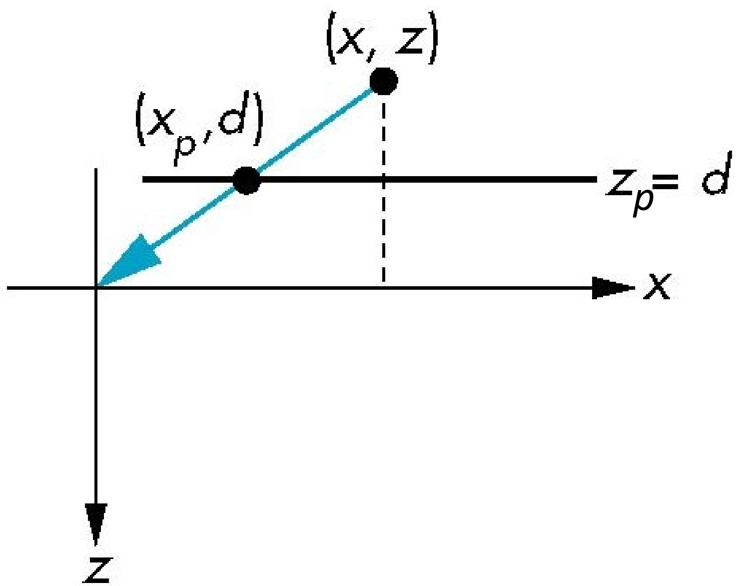
Simple Perspective

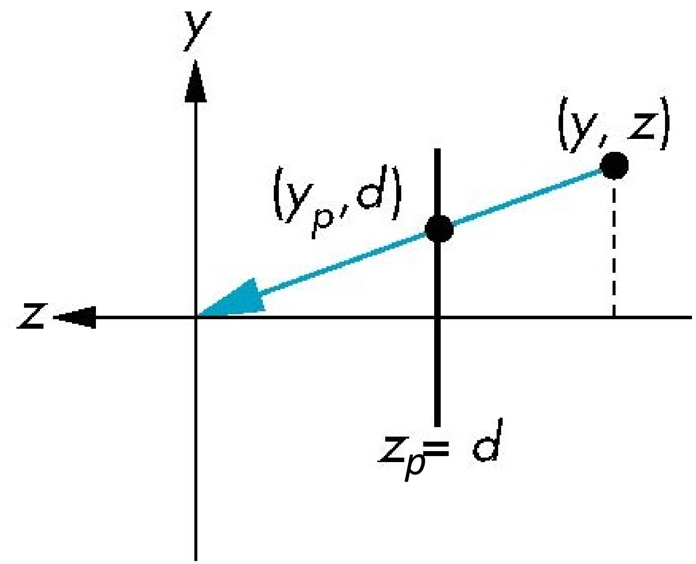
- Center of projection at the origin
- Projection plane $z = d$, $d < 0$



Perspective Equations

- Top and side views





$$x_p = \frac{x}{z/d}, \quad y_p = \frac{y}{z/d}, \quad z_p = d$$

Homogeneous Coordinate Form

Consider $\mathbf{q} = \mathbf{M}\mathbf{p}$ where

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Perspective Division

- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates

- This *perspective division* yields the desired perspective equations

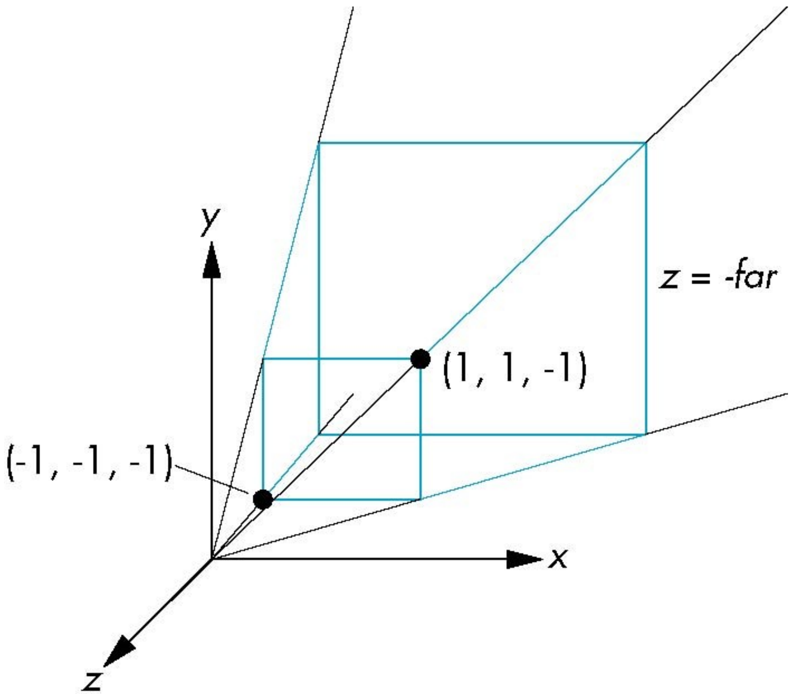
$$x_p = \frac{x}{z/d}, \quad y_p = \frac{y}{z/d}, \quad z_p = d$$

Simple Perspective Projection

- Symmetric, 90 degree field of view frustum

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

near plane is z=-1
far plane is not defined



Generalization

- After perspective division, the point (*x*, *y*, *z*, 1) goes to

$$x' = -\frac{x}{z}$$

$$y' = -\frac{y}{z}$$

$$z' = -(\alpha + \frac{\beta}{z})$$

$$N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspective normalization matrix

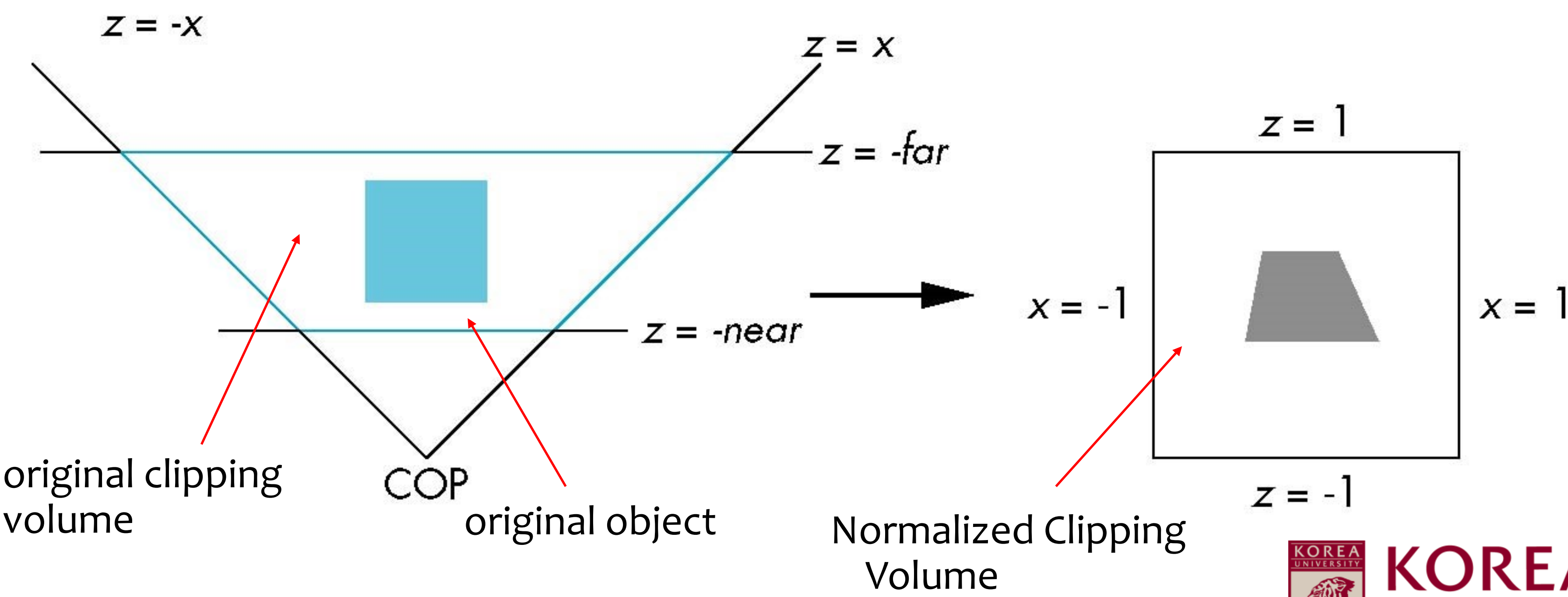
which projects orthogonally to the desired point regardless of α and β

Picking α and β

- Map view volume to unit cube
  - the near plane is mapped to z = -1
  - the far plane is mapped to z =1
  - and the sides are mapped to x = ± 1, y = ± 1
    if x = ±z and y = ±z

$$\alpha = \frac{\text{near} + \text{far}}{\text{near-far}}$$

$$\beta = \frac{2\text{near} * \text{far}}{\text{near} - \text{far}}$$

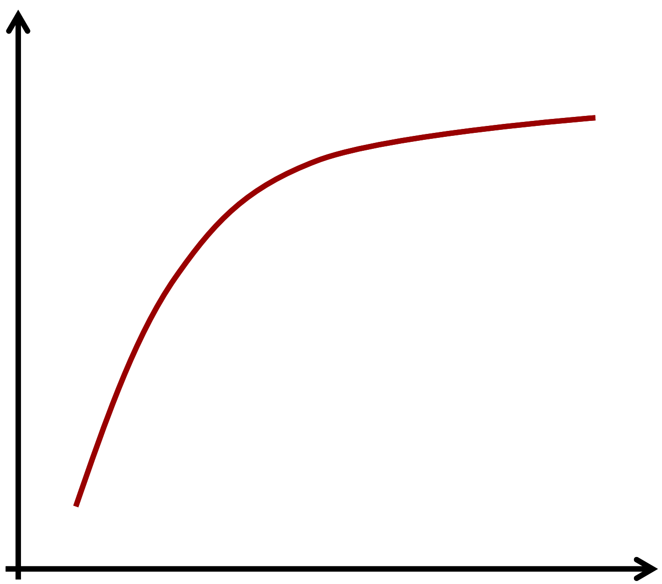


Normalization and Hidden-Surface Removal

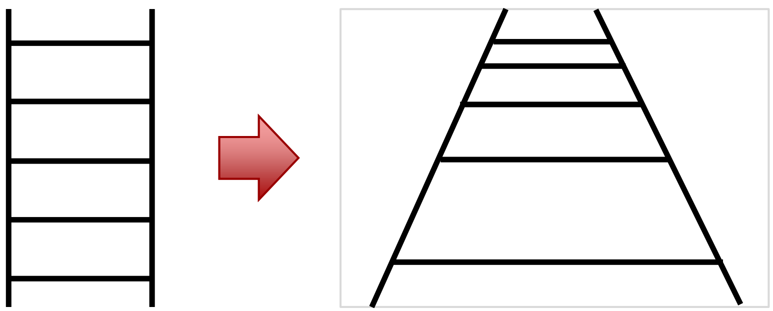
- Our selection of the form of the perspective matrices was chosen so that if $z_1 > z_2$ in the original clipping volume then the for the transformed points $z_1' > z_2'$
- Hidden surface removal works in the normalized clipping volume
- However, the formula $z' = -(\alpha + \frac{\beta}{z})$ implies that the distances are distorted by the normalization which can cause numerical problems

Depth Precision

- Nonlinear z-scaling



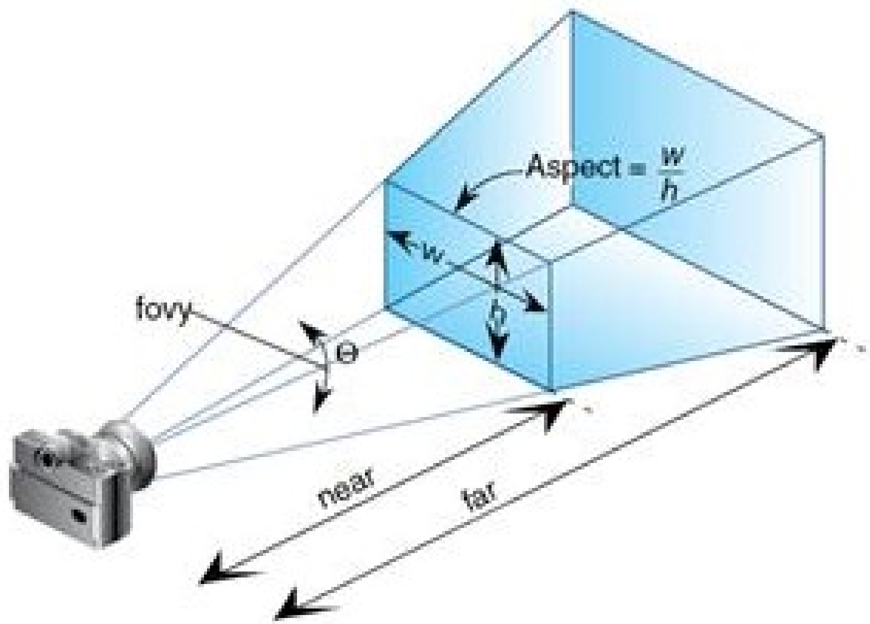
$$z' = -(\alpha + \frac{\beta}{z})$$



linear binning for z-buffer = precision problem

Perspective Projection

- Perspective() in mat.h



Perspective Matrix for Arbitrary View Frustum

- The normalization in view frustum requires an initial shear to form a right viewing pyramid, followed by a scaling to get the normalized perspective volume.

$$\mathrm{M_P} = \mathrm{NSM_S}$$

our previously defined perspective matrix

shear and scale

Normalized Device Coordinate

- Perspective division
  - (x/w, y/w, z/w)
- 3D coordinate

$$
\begin{aligned}
\mathbf{p}_{ndc} &= \mathbf{p}_{clip} / w \\
&= \mathrm{M_P}\mathbf{p}_{eye} / w \\
&= \mathrm{NSM_S}\mathbf{p}_{eye} / w \\
&= \mathrm{NSM_S M_V M_T}\mathbf{p}_{obj} / w
\end{aligned}
$$

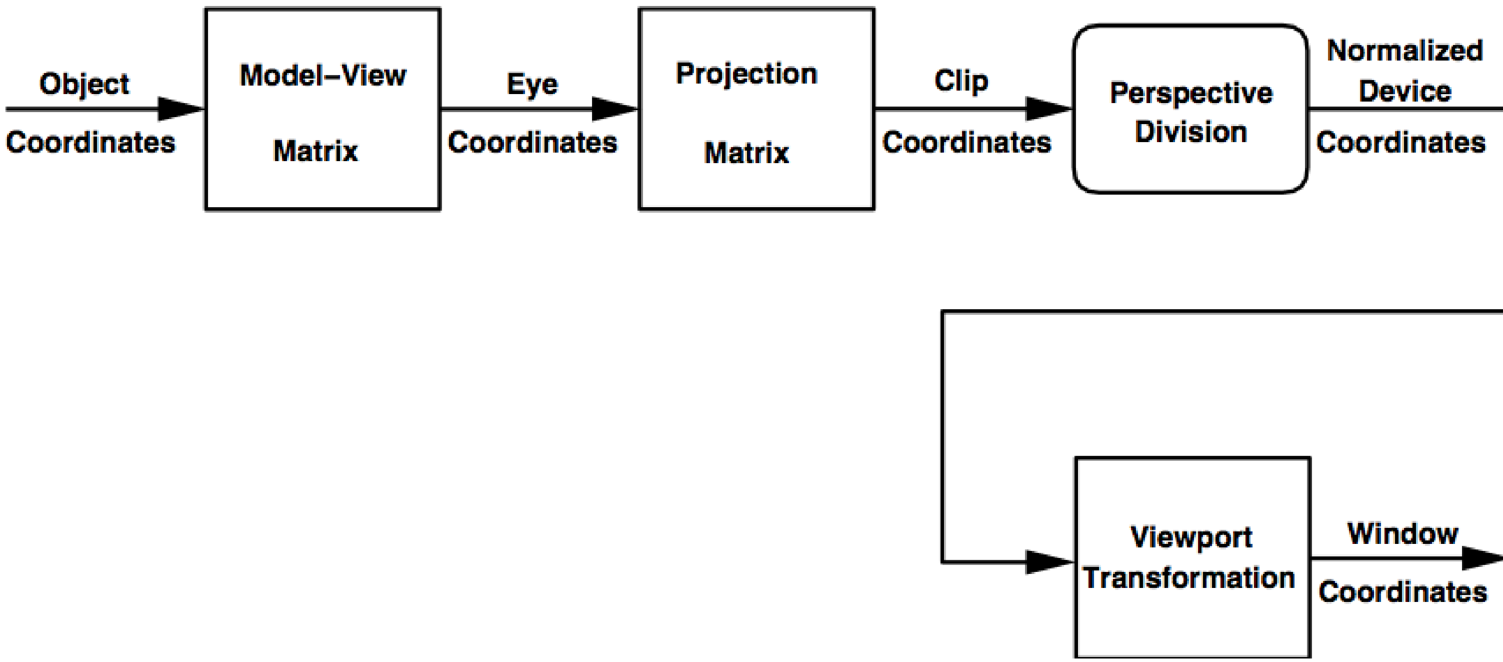
Windows Coordinate

- Viewport transformation
  - Canonical view volume to screen
- 2D coordinate
  - [-1,1] mapped to min/max pixel coordinates
- Depth value
  - [-1,1] quantized to [0,1] (16/32 bit)

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} \frac{w}{2} x_{ndc} + (x + \frac{w}{2}) \\ \frac{h}{2} y_{ndc} + (y + \frac{w}{2}) \\ \frac{f-n}{2} z_{ndc} + \frac{f+n}{2} \end{pmatrix}$$

w, h: width/height of the window
x, y : window offset (in the screen)
n: 0, *f*: 1

Coordinate Transformations



Questions?