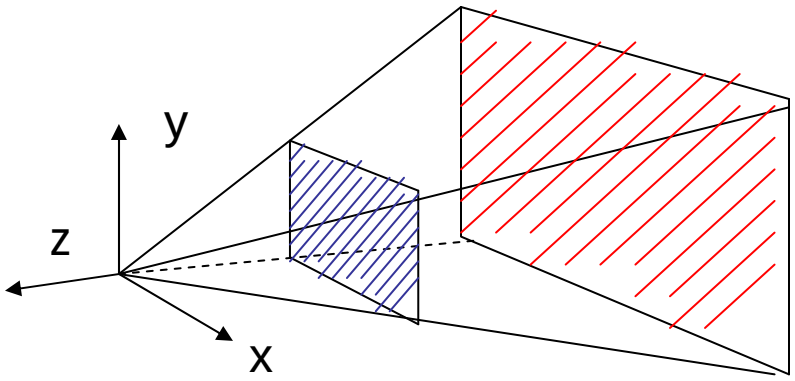
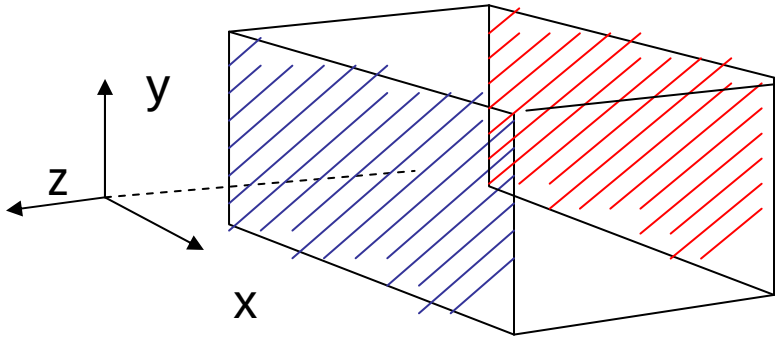
Projection Transformation

- Projection – map the object from 3D space to 2D screen



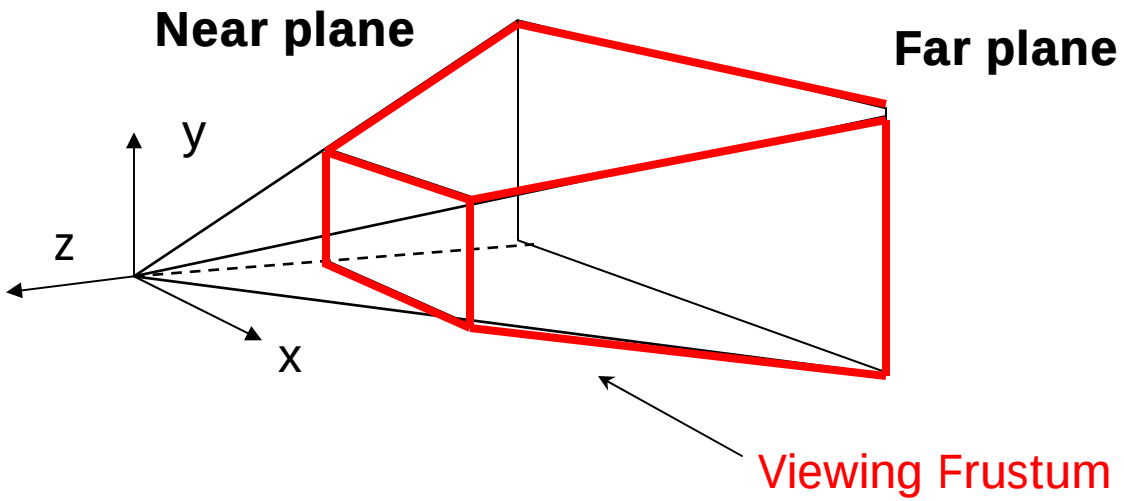
Perspective: **gluPerspective()**



Parallel: **glOrtho()**

Viewing Frustum

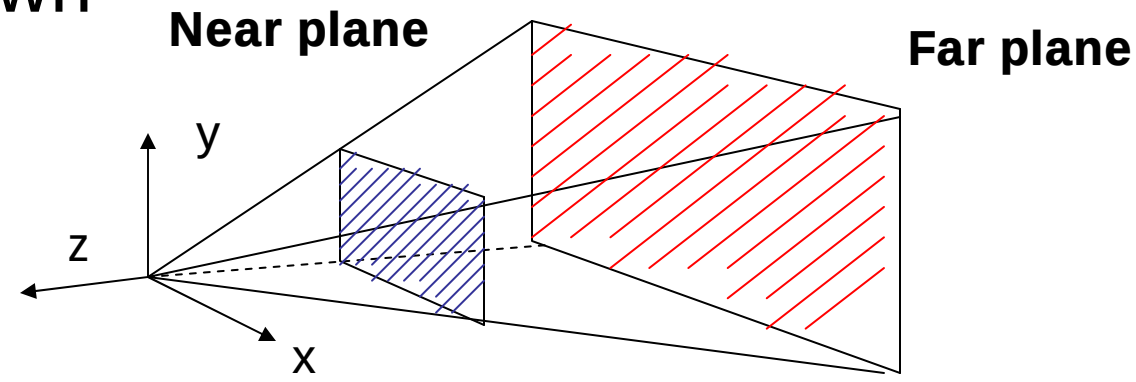
- 3D counterpart of 2D world clip window



- Objects outside the frustum are clipped

Near and Far Clipping Planes

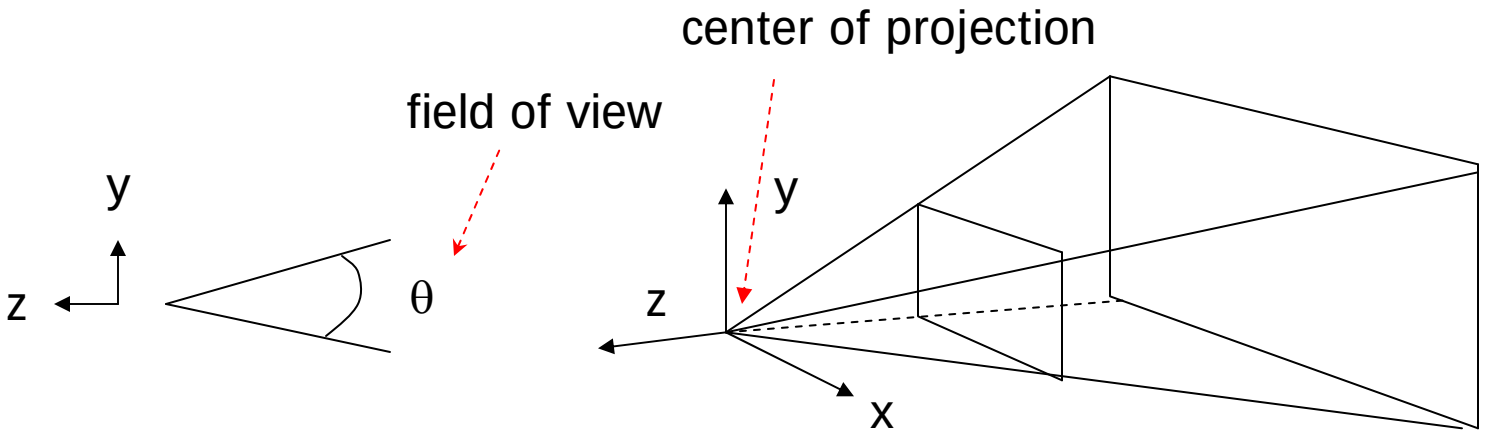
- Only objects between near and far planes are drawn



- Near plane + far plane + field of view = Viewing Frustum

Field of View

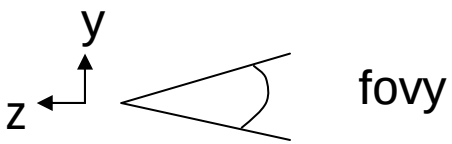
- Determine how much of the world is taken into the picture

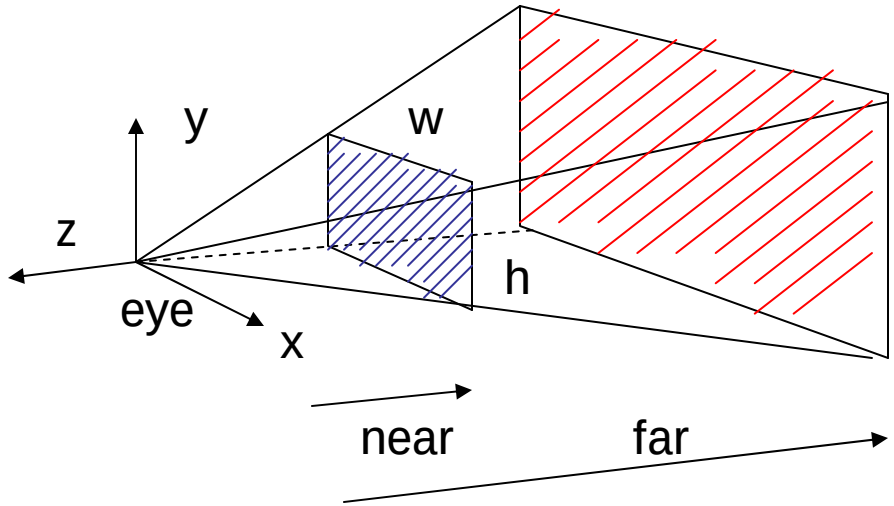


- The larger is the field view, the smaller is the object projection size

gluPerspective(fovy, aspect, near, far)

- Aspect ratio is used to calculate the window width

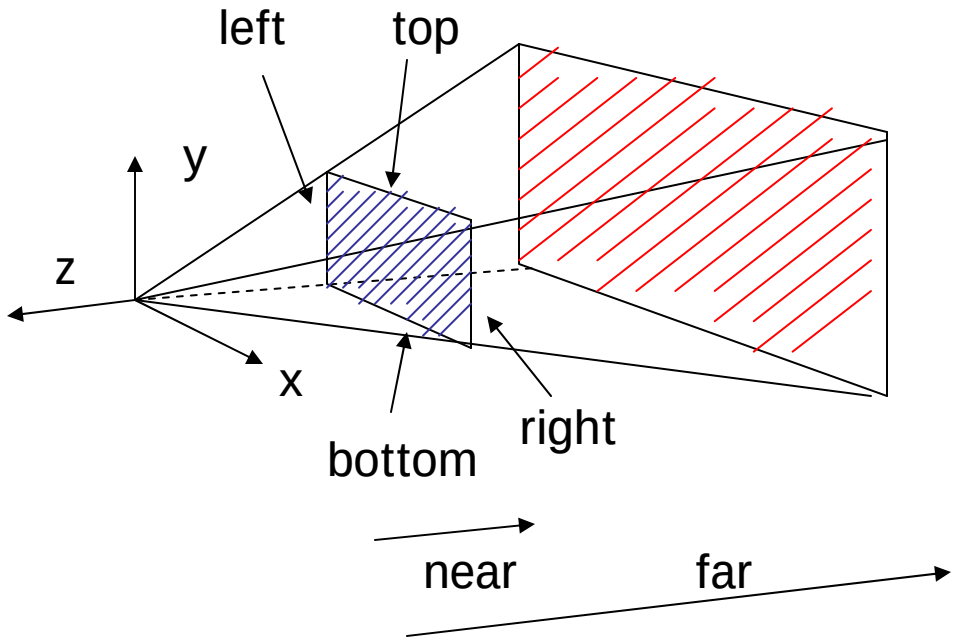




Aspect = w / h

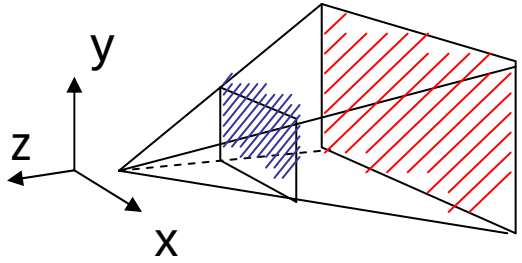
glFrustum(left, right, bottom, top, near, far)

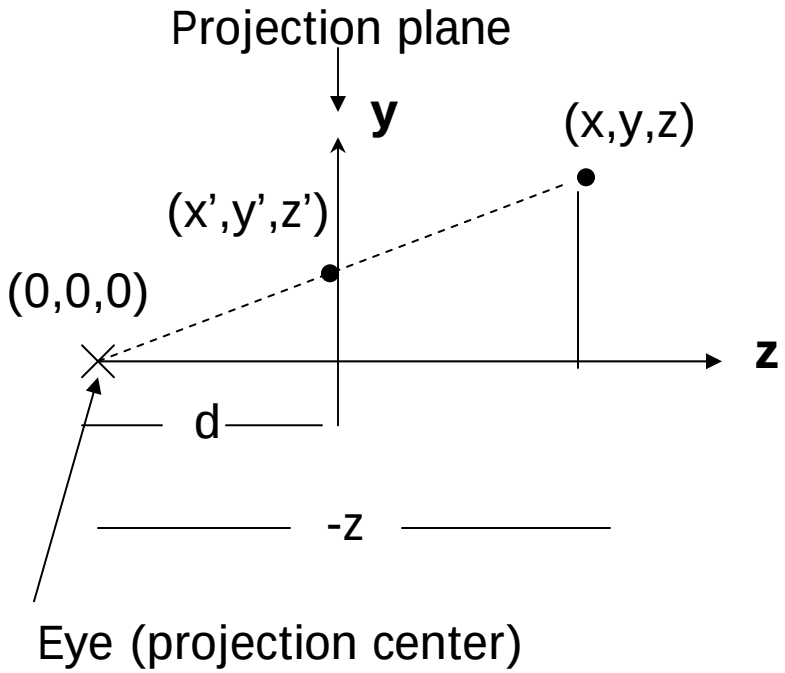
- Or You can use this function in place of gluPerspective()



Perspective Projection

- Side view:





Based on similar triangle:

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

$$Y' = y \times \frac{d}{-z}$$

Perspective Projection (2)

- Same for x. So we have:

  $x' = x \times d / -z$

  $y' = y \times d / -z$

  $z' = -d$

- Put in a matrix form:

$$\begin{vmatrix} x' \\ y' \\ z' \\ w \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & (1/-d) & 0 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

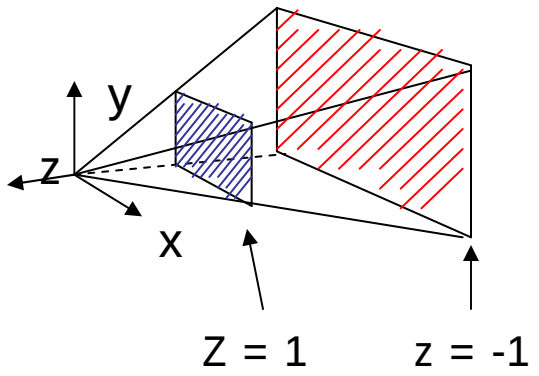
- OpenGL assume d = 1, i.e. the image plane is at z = -1

Perspective Projection (3)

- We are not done yet. We want to somewhat keep the z information so that we can perform depth comparison
- Use pseudo depth – OpenGL maps the near plane to 1, and far plane to -1
- Need to modify the projection matrix: solve a and b

$$\begin{vmatrix} x' \\ y' \\ z' \\ w \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & (1/-d) & 0 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

How to solve a and b?



Perspective Projection (4)

- Solve a and b

$$\begin{vmatrix} x' \\ y' \\ z' \\ w \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & (1/-d) & 0 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

- $(0,0,1)^T = M \times (0,0,-near)^T$
  $(0,0,-1)^T = M \times (0,0,-far)^T$

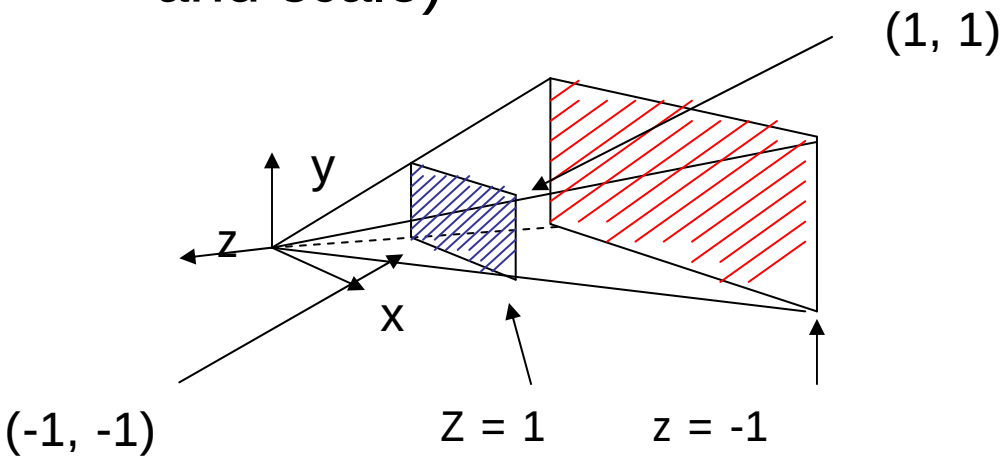
M

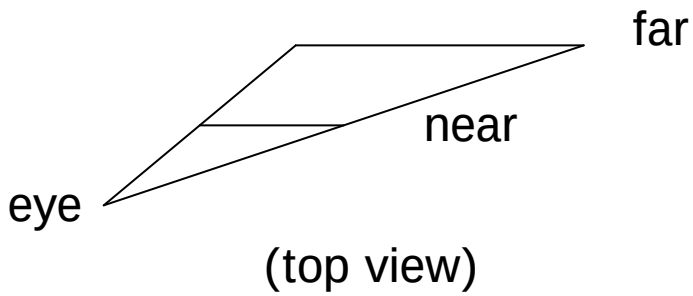
- $a = -(far+near)/(far-near)$
  $b = (-2 \times far \times near) / (far-near)$

Verify this!

Perspective Projection (5)

- Not done yet. OpenGL also normalizes the x and y ranges of the viewing frustum to [-1, 1] (translate and scale)



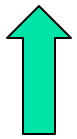


- And takes care the case that eye is not at the center of the view volume (shear)

Perspective Projection (6)

- Final Projection Matrix:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{vmatrix} 2N/(xmax-xmin) & 0 & (xmax+xmin)/(xmax-xmin) & 0 \\ 0 & 2N/(ymax-ymin) & (ymax+ymin)/(ymax-ymin) & 0 \\ 0 & 0 & -(F+N)/(F-N) & -2F*N/(F-N) \\ 0 & 0 & -1 & 0 \end{vmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



glFrustum(xmin, xmax, ymin, ymax, N, F) N = near plane, F = far plane

Perspective Projection (7)

- After perspective projection, the viewing frustum is also projected into a canonical view volume (like in parallel projection)