Viewing

Chapter 10
Projection

Camera transforms

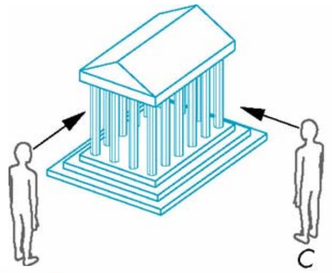
- Until now we have considered all of our geometry in a 3D space
- Ultimately everything ended up in eye coordinates with coordinates
- We said that the camera is placed at the origin of the eye frame, and that it is looking down the eye's negative z-axis.
- This somehow produces a 2D image.
- We had a magic matrix which created gl_Position
- Now we will study this step

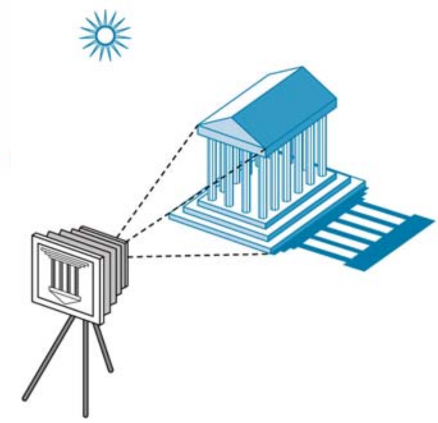
Graphics Models

- In 3D world …
  image formation process

  a set of vertices   locations
- Objects and Viewers
- Light

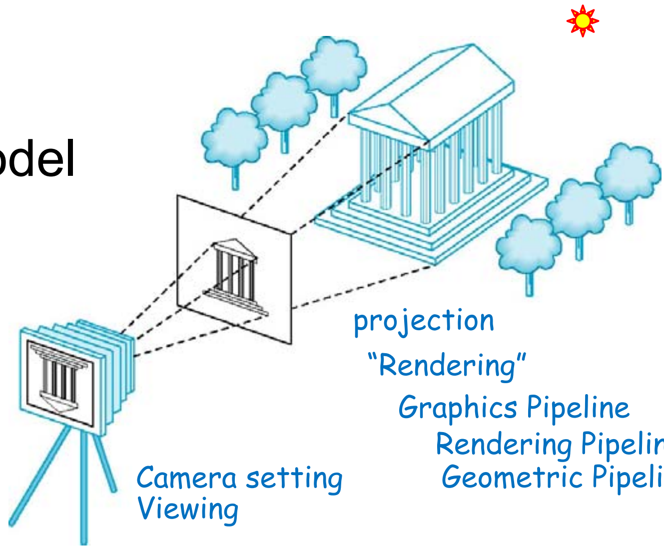
- Imaging Systems
  - The human visual system
  - The pinhole camera
  - Microscopes
  - Telescopes …

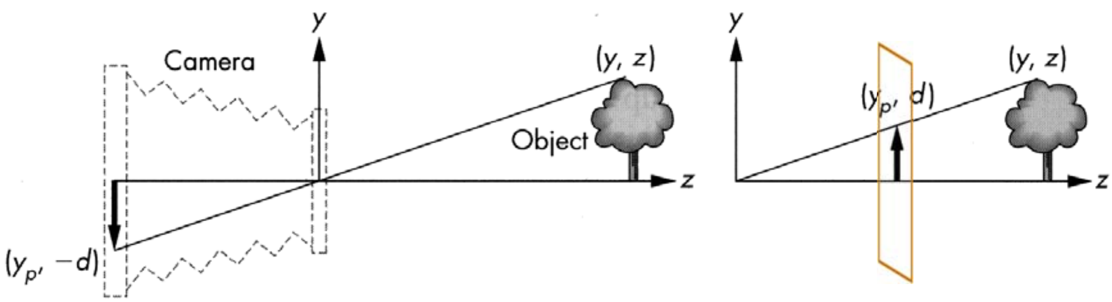




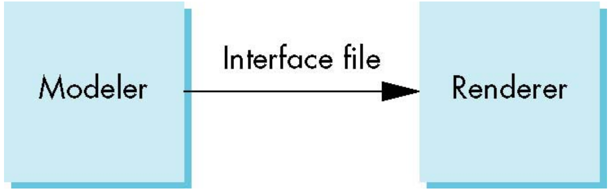
Virtual Camera

- Synthetic camera model
  - Pinhole camera model
  - Move the image plane
    to the front

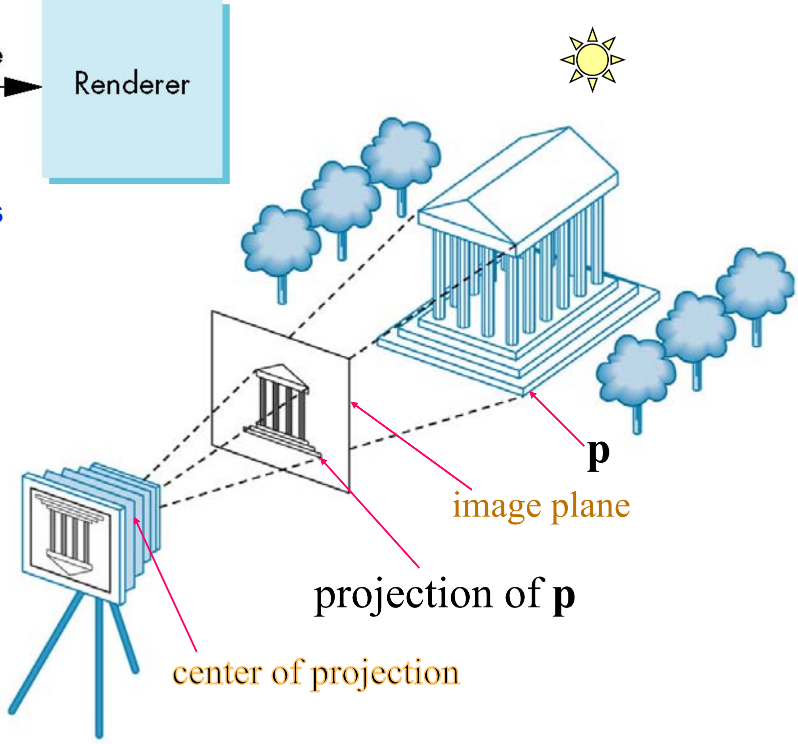




Modeling – Rendering Paradigm

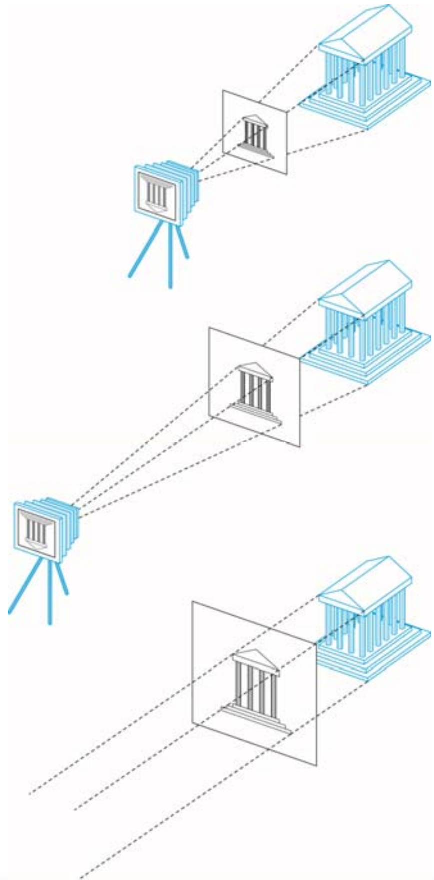


a set of vertices
and faces info
for the object



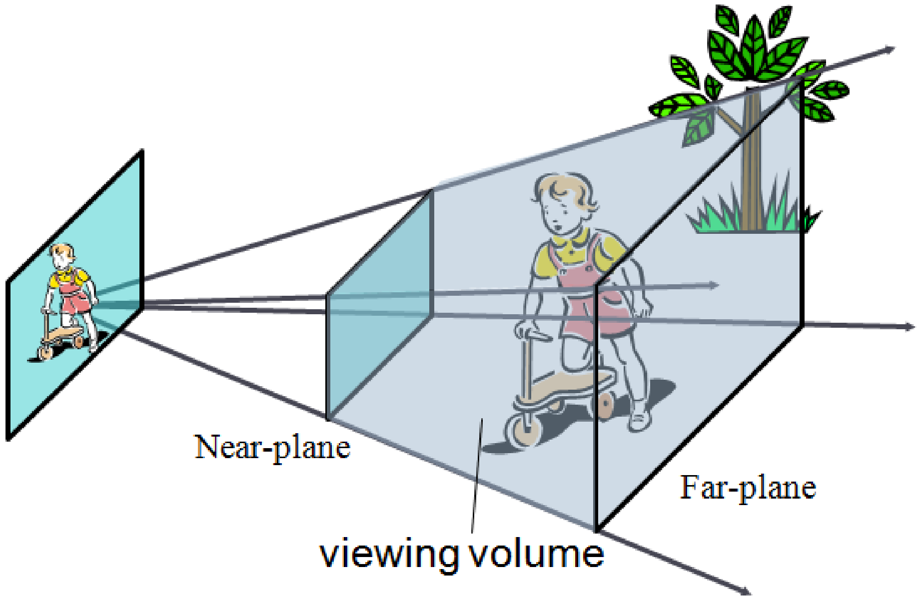
Viewing

- Perspective view
- Orthographic view
  - All the projectors become parallel and the center of projection is replaced by a direction of projection.

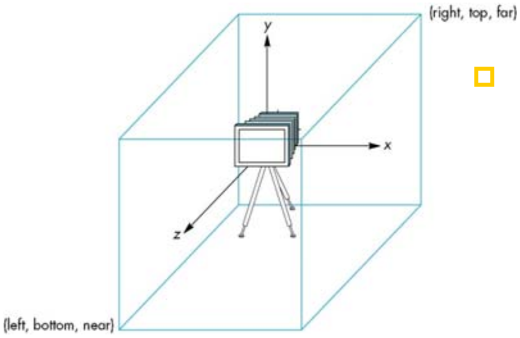


Viewing volume

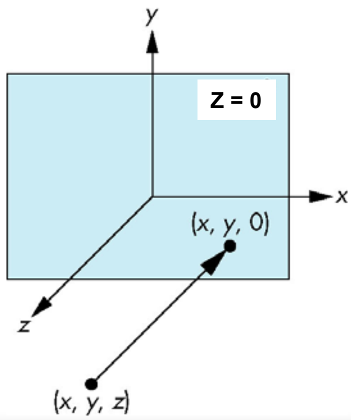
- **Frustum:** the part of a solid, as a cone or pyramid, between two parallel cutting planes

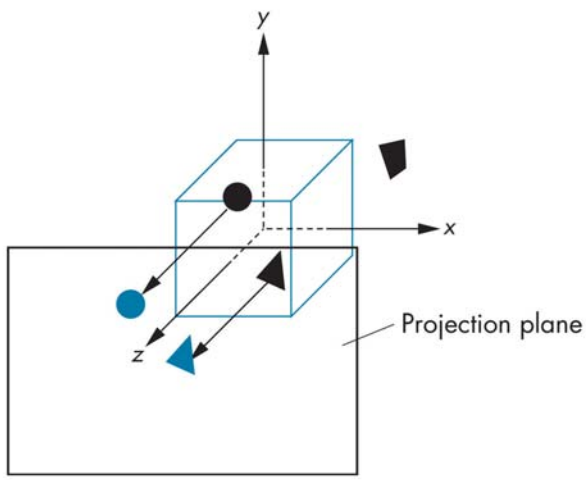


(old)OpenGL default camera



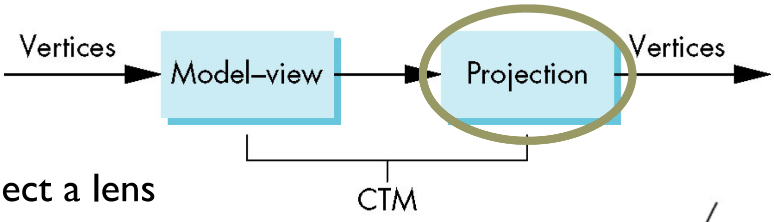
- -1, 1, -1, 1, -1, 1
  glOrtho (left, right, bottom, top, near, far)



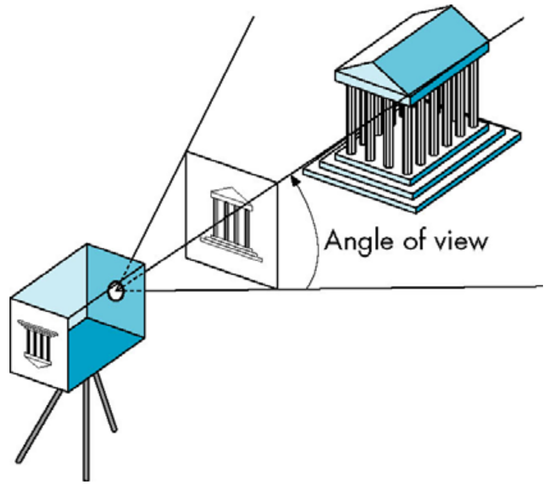


Projection

- The camera is placed at the desired location.



- Let us select a lens
  - A wide-angle lens
    - Provides dramatic perspective views
  - A telephoto lens
    - Appears flat
- Set the projection matrix



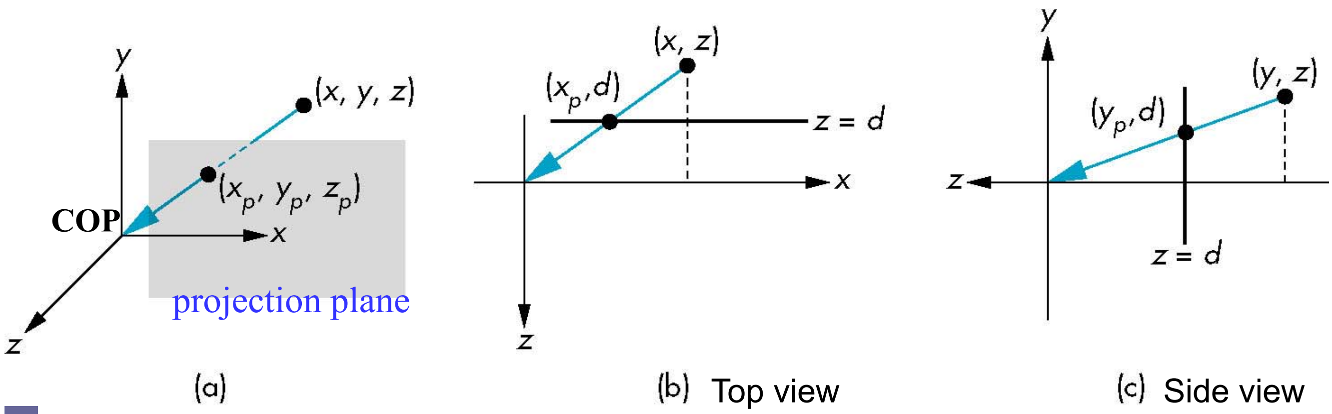
Mathematics of simple projection

- Perspective Projection
  - Let d be the focal length.
  - $z_p = d$

$$x_p = \frac{d}{z}x$$

$$y_p = \frac{d}{z}y$$

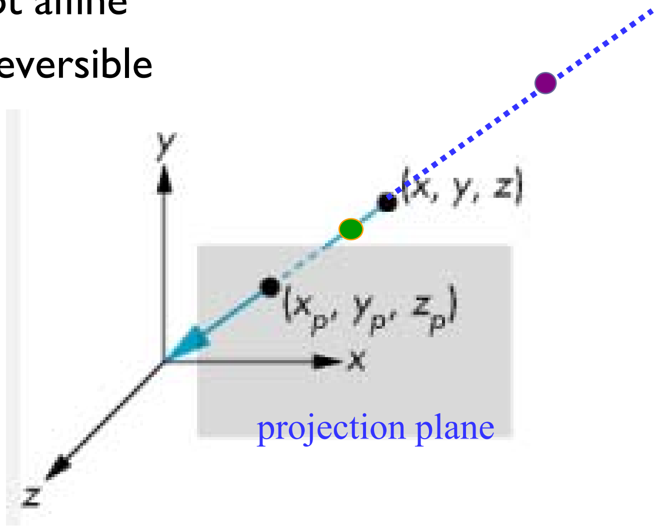
non-uniform foreshortening



Mathematics of simple projection

- Perspective Projection
  - Preserves lines
  - Not affine
  - Irreversible

Images of objects farther from the COP are reduced in sized compared to the images of objects closer to the COP



Mathematics of simple projection

- Perspective Transformation
  - Let's see in 4D

$$\begin{tikzcd}
(x,y,z) \arrow[r, "P"] \arrow[d] & (x_p, y_p, z_p) \\
(x,y,z,1) \arrow[r, "M"] & (x', y', z', h) \arrow[u]
\end{tikzcd}$$

(xd/z, xd/y, d)

(wx, wy, wz, w)

$$x_p = \frac{d}{z} x$$

$$y_p = \frac{d}{z} y$$

$$z_p = d$$

M transforms
the point (x, y, z, 1)
to the point (x, y, z, z/d)

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

Perspective Projection

- Homogenous coordinates
  - 4D point: p = $[wx\ \ wy\ \ wz\ \ w]^T$, $w \neq 0$
  - We can recover 3D point (x, y, z)

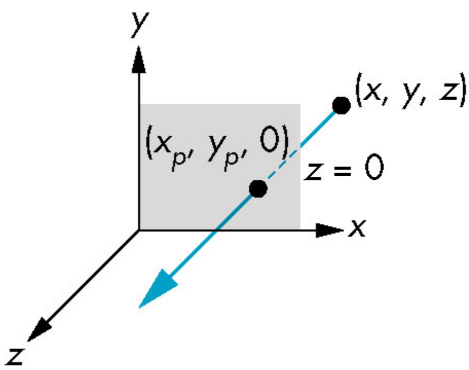
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{d}{z}x \\ \frac{d}{z}y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

- Perform the perspective division at the end of the projection pipeline

Orthogonal Projections

- Orthographic projections are a special case of parallel projections.

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



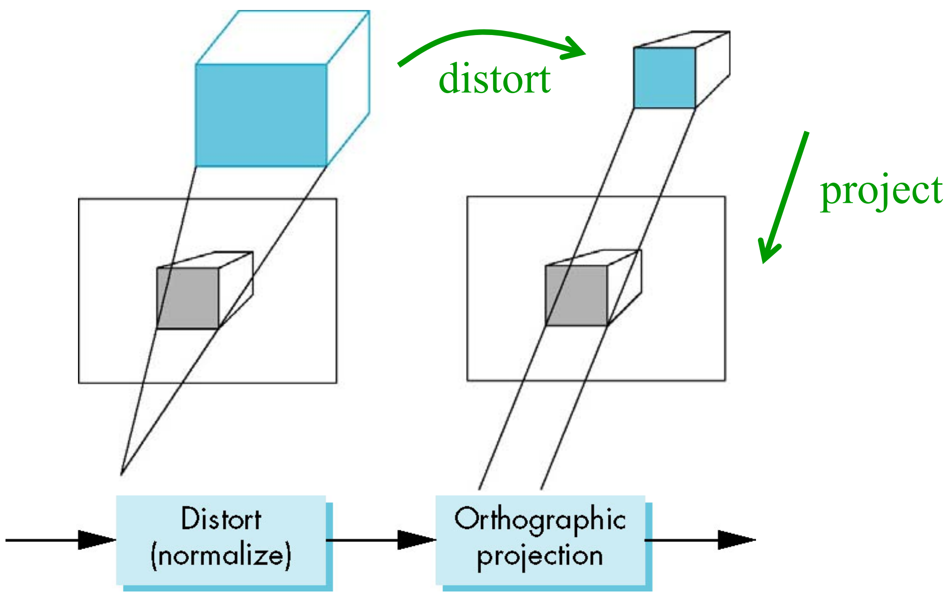
$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Parallel Projection Matrix

- Two issues:
  - How to treat two different projections as a single, uniform fashion?
  - Scene clipping is difficult for the frustum, while there is a very efficient way to do it for the rectangular view volume.
  - How to have a parallel oblique view?
- Answers:
  - We can set up a projection matrix from scratch
  - We can modify on of the standard views

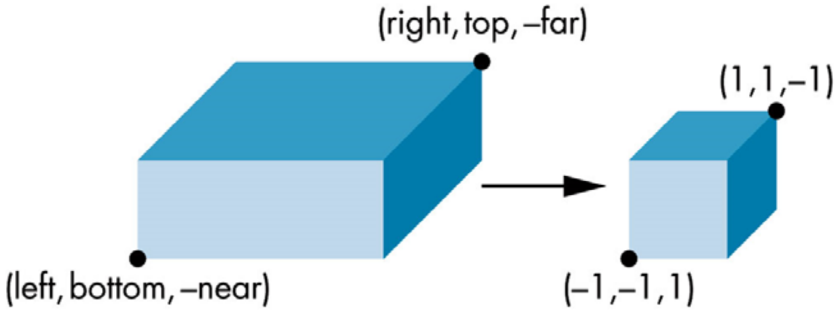
Projection Normalization

- Convert all projections into orthogonal projections by first distorting the objects such that the orthogonal projection of the distorted objects is the same as the desired projection of the original objects.



Orthogonal Projection Matrices

- Map a view volume to the canonical view volume
  - Clipping volume is set with:
    - x = ±1
    - y = ±1
    - z = ±1



  - Any arbitrary view volume will be **normalized** to the canonical view volume
  - No distortion, only Translation and Scaling

Orthogonal Projection Matrices

- Center of the view volume

$$x_c = \frac{x_{\max} + x_{\min}}{2}, y_c = \frac{y_{\max} + y_{\min}}{2}, z_c = \frac{-z_{\max} - z_{\min}}{2}$$

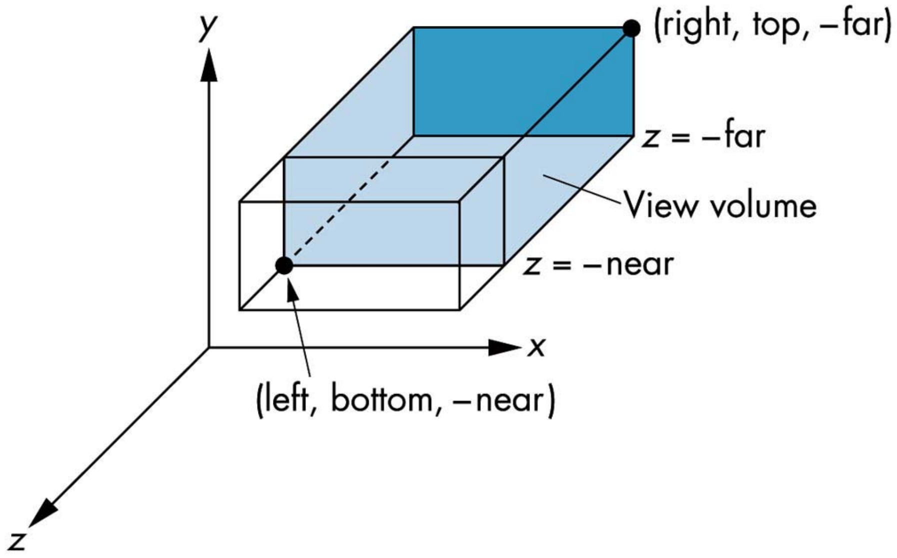
- Size of the view volume

$$\Delta x = x_{\max} - x_{\min}, \Delta y = y_{\max} - y_{\min}, \Delta z = z_{\max} - z_{\min}$$

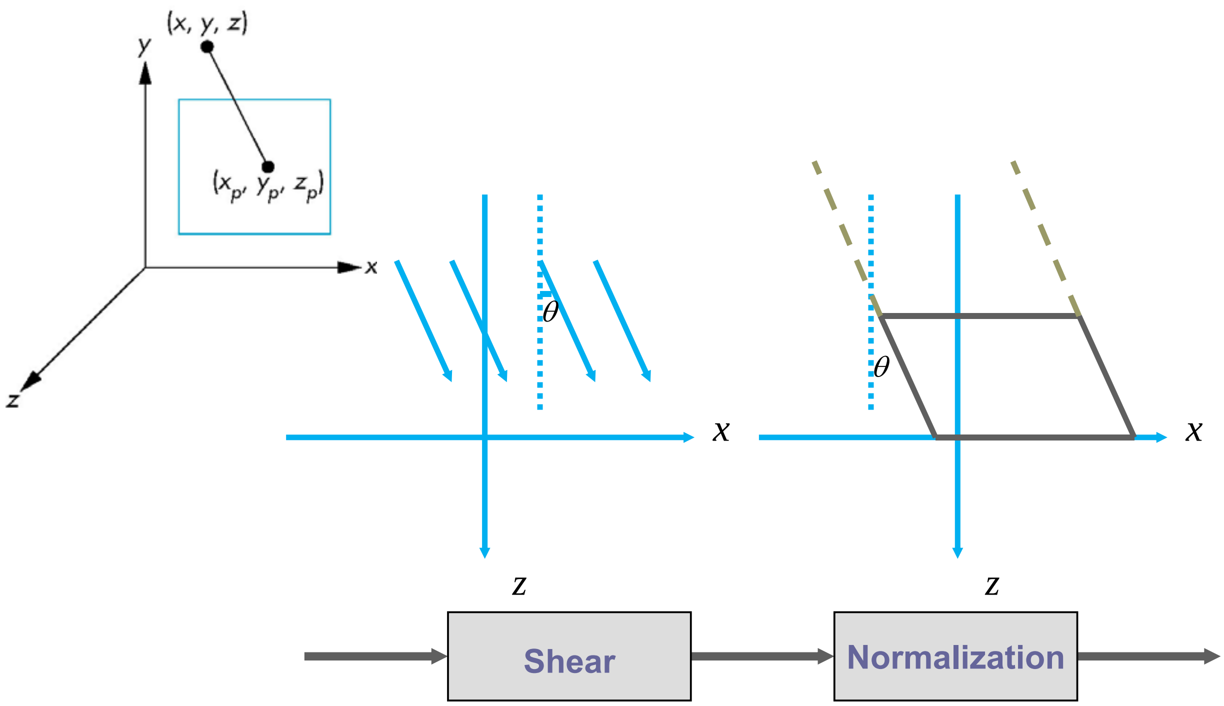
$$P = ST = \begin{bmatrix} \frac{2}{x_{\max} - x_{\min}} & 0 & 0 & -\frac{x_{\max} + x_{\min}}{x_{\max} - x_{\min}} \\ 0 & \frac{2}{y_{\max} - y_{\min}} & 0 & -\frac{y_{\max} + y_{\min}}{y_{\max} - y_{\min}} \\ 0 & 0 & \frac{2}{z_{\max} - z_{\min}} & \frac{z_{\max} + z_{\min}}{z_{\max} - z_{\min}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projections in OpenGL

- Orthographic
  - `glOrtho(xmin,xmax,ymin,ymax,near,far)`

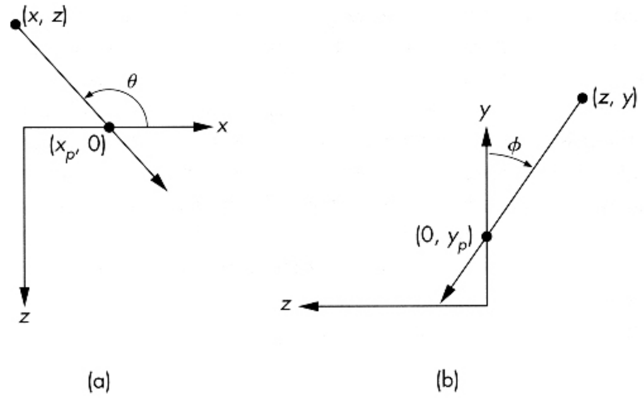


Oblique Projections



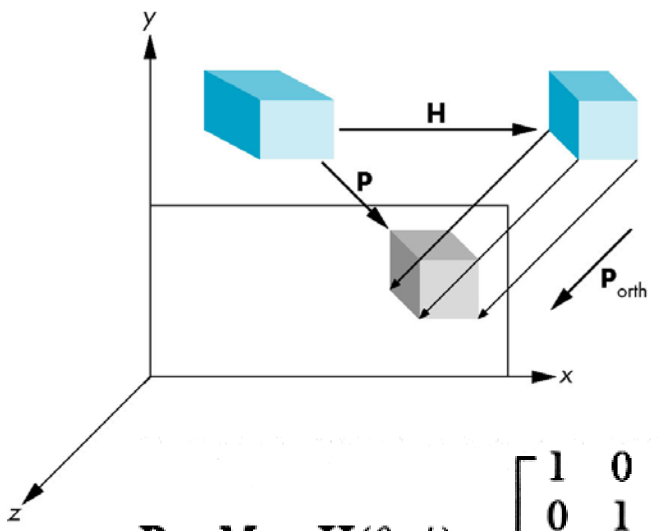
Oblique Projections

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{H}(\theta,\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Oblique Projections



$$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{H}(\theta,\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

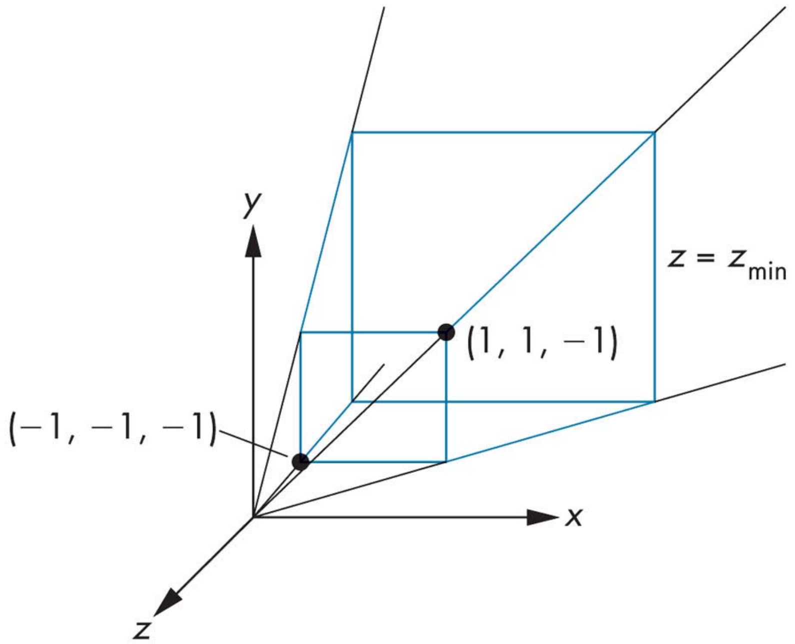
Oblique Projections

$$\mathbf{ST} = \begin{bmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & -\frac{x_{max}+x_{min}}{x_{max}-x_{min}} \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & -\frac{y_{max}+y_{min}}{y_{max}-y_{min}} \\ 0 & 0 & \frac{2}{z_{max}-z_{min}} & -\frac{z_{max}+z_{min}}{z_{max}-z_{min}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Adding the volume normalization ST,

$$\mathbf{P} = \mathbf{M}_{orth}\mathbf{STH}.$$

Perspective-Projection Matrices



Perspective-Projection Matrices

- Perspective-normalization transformation
  - Converts a perspective projection to an orthogonal projection

- Simple perspective-projection matrix
- Projection plane at z = –1 (or d=1)

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspective-Projection Matrices

- Let's consider

$$N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & -2 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

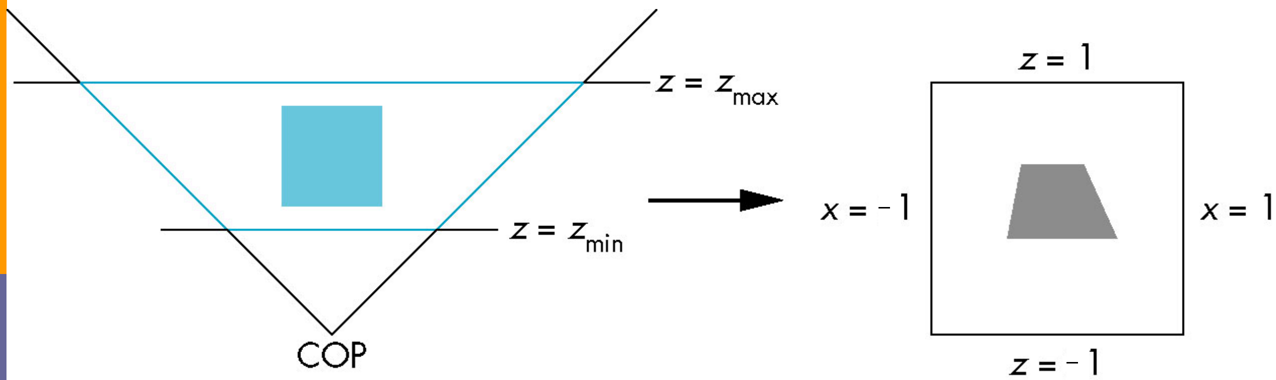
N will transform $P = \begin{bmatrix} x & y & z \end{bmatrix}$ into

$$x' = -\frac{x}{z} \quad y' = -\frac{y}{z} \quad z' = 2(1 + \frac{1}{z})$$

Futhermore, the frustrum becomes a box

$$x = \pm 1, \; y = \pm 1, \; z = 0, \quad 2(1 + \frac{1}{z_{\max}})$$

Perspective-Projection Matrices



Perspective-Projection Matrices

- In general,

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

- Then

front plane:

$$z'' = -\left(\alpha + \frac{\beta}{z_{\min}}\right)$$

far plane:

$$z'' = -\left(\alpha + \frac{\beta}{z_{\max}}\right)$$

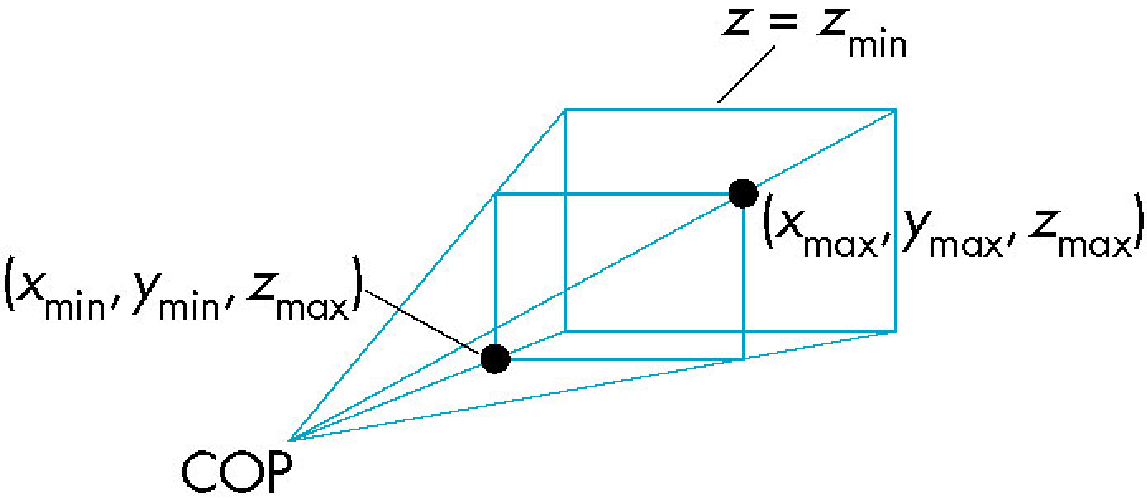
- If we select

$$\alpha = \frac{z_{\max} + z_{\min}}{z_{\max} - z_{\min}},$$

$$\beta = \frac{2 z_{\max} z_{\min}}{z_{\max} - z_{\min}},$$

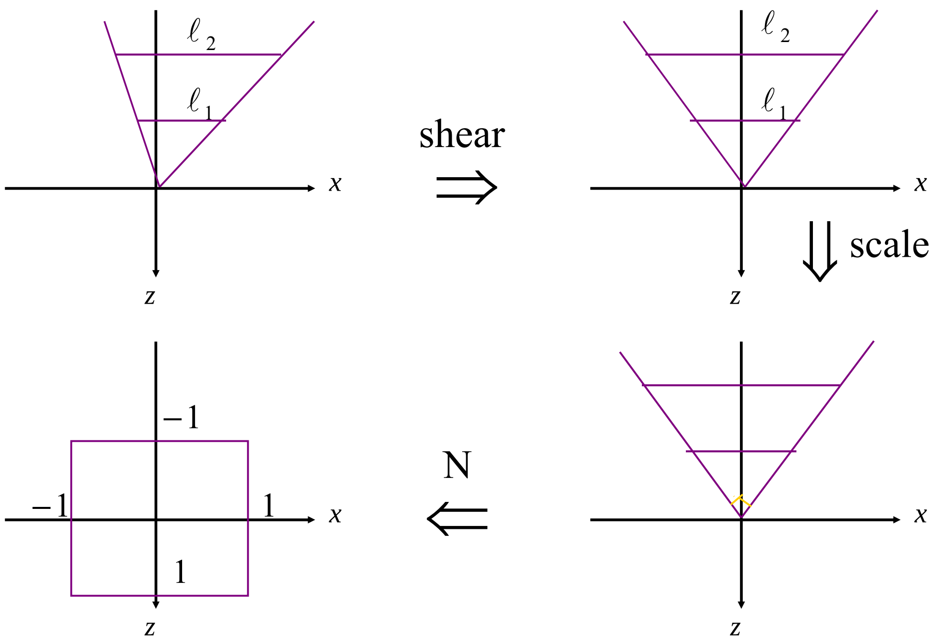
we obtain the canonical view volume.

OpenGL Perspective




```
glFrusturm(xmin,xmax,ymin,ymax,near,far)
```

OpenGL Perspective



OpenGL Perspective

- Shear

$$\mathbf{H}(\cot\theta, \cot\phi) = \mathbf{H}\left(\frac{x_{min}+x_{max}}{2z_{min}}, \frac{y_{max}+y_{min}}{2z_{min}}\right)$$

- Then,

$$x = \pm\frac{x_{max}-x_{min}}{2z_{min}},$$
$$y = \pm\frac{y_{max}-y_{min}}{2z_{min}},$$
$$z = z_{max},$$
$$z = z_{min}.$$

$$\alpha = \frac{z_{max}+z_{min}}{z_{max}-z_{min}},$$
$$\beta = \frac{2z_{max}z_{min}}{z_{max}-z_{min}},$$

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

- Scale

$$x = \pm z,$$
$$y = \pm z,$$

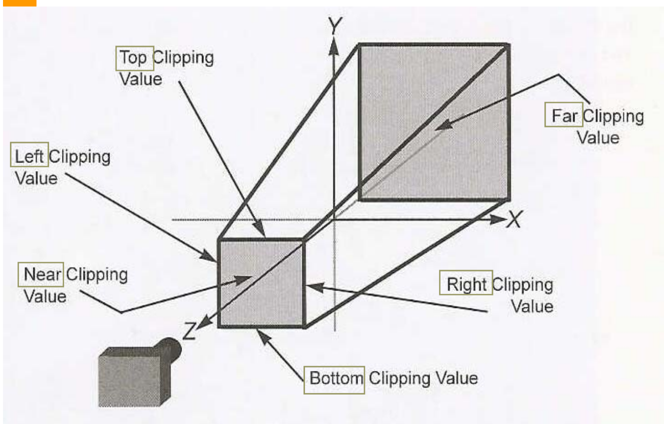
OpenGL Perspective

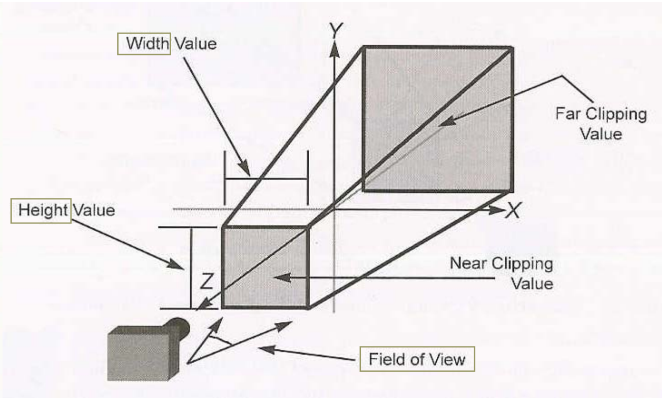
$$\mathbf{P} = \mathbf{NSH} = \begin{bmatrix} \frac{2z_{min}}{x_{max}-x_{min}} & 0 & \frac{x_{max}+x_{min}}{x_{max}-x_{min}} & 0 \\ 0 & \frac{2z_{min}}{y_{max}-y_{min}} & \frac{y_{max}+y_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & -\frac{z_{max}+z_{min}}{z_{max}-z_{min}} & -\frac{2z_{max}z_{min}}{z_{max}-z_{min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

$$\mathbf{P} = \mathbf{NSH} = \begin{bmatrix} \frac{2z_{min}}{x_{max}-x_{min}} & 0 & \frac{x_{max}+x_{min}}{x_{max}-x_{min}} & 0 \\ 0 & \frac{2z_{min}}{y_{max}-y_{min}} & \frac{y_{max}+y_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Projections in OpenGL

- Perspective
  - `glFrusturm(xmin,xmax,ymin,ymax,near,far)`
    left right bottom top
  - `gluPerspective(fovy, aspect, near, far)`





Projection tutorial by Nate Robins