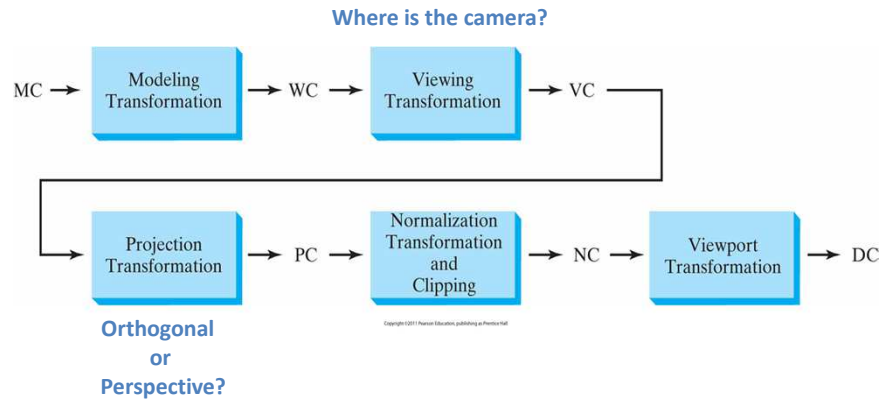


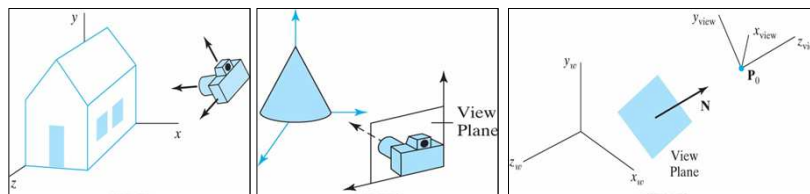
3D Viewing Pipeline



1

Viewing Coordinates

- Choose where to place a camera
- Set camera at position $P_0 = (x_0, y_0, z_0)$ which is called view point, eye position or camera position
- Specify a **view up vector** V , which defines the y_{view} direction
- Specify a **vector** N to be in direction from a reference point to P_0
- View plane (projection plane) is normally perpendicular to z_{view}



2

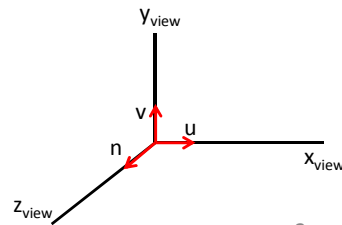
Transformation from World to Viewing

1. Translate the viewing-coordinate origin to world coordinate origin
2. Rotate to align x_{view} , y_{view} , and z_{view} axes with world x_w , y_w , and z_w

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R.T = \begin{bmatrix} u_x & u_y & u_z & -u.P_0 \\ v_x & v_y & v_z & -v.P_0 \\ n_x & n_y & n_z & -n.P_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$\begin{aligned} -u.P_0 &= -x_0u_x - y_0u_y - z_0u_z \\ -v.P_0 &= -x_0v_x - y_0v_y - z_0v_z \\ -n.P_0 &= -x_0n_x - y_0n_y - z_0n_z \end{aligned}$$



3

Projections

- ❑ Projection
 - Transformation from n -D coordinate system to m -D coordinate system, where $m < n$
- ❑ Our concern
 - $n = 3$ and $m = 2$
 - ⇒ projection from 3D to 2D
- ❑ Terminology
 - Projectors: Straight projection rays
 - Center of projection: Where the projectors emanated from
 - Projection plane: Where the projection forms

4

Projections

- ❑ Projection from 3D to 2D defined by
 - Projectors emanate from COP, pass through each point of the object, and intersect the projection plane

- ❑ Planar geometric projections
 - Projection is onto a plane
 - Referred to as "projections" here

5

Projections

- ❑ Two Basic Classes
 - Perspective
 - Parallel (Orthographic)

- ❑ Perspective
 - Distance between projection plane and COP is finite
 - Visual effect similar to human visual system
 - ❖ Perspective foreshortening:
 - distance from COP longer, size smaller
 - Exact shape, measurement, parallelism not reserved

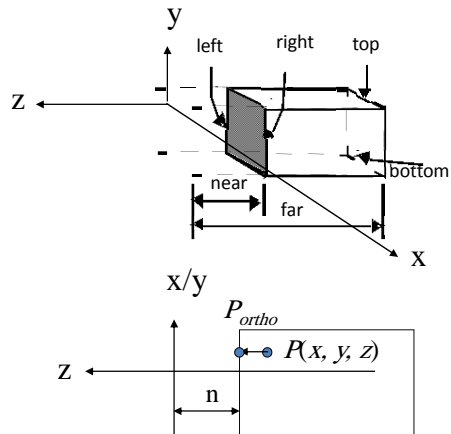
6

Orthographic Projection Math

- Projection parallel to z axis and perpendicular to projection plane

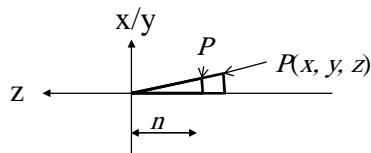
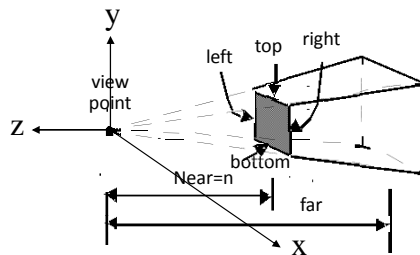
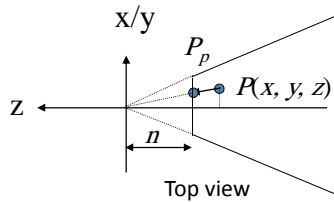
$$\begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & n \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P_{ortho}(x, y, n)$$



9

Perspective Projection Math



$$x_p/n = x/z \Rightarrow x_p = x/(z/n)$$

The projection onto the near clipping plane:

$$(x_p, y_p, z_p) = \left(\frac{x}{z/n}, \frac{y}{z/n}, n \right)$$

10

In homogeneous coordinates

In matrix form, perspective projection matrix:

$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix}$$

Transforming in homogeneous coordinates yields the [general homogeneous point](#):

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = M_{per} P;$$

where $\Rightarrow w = z/n$

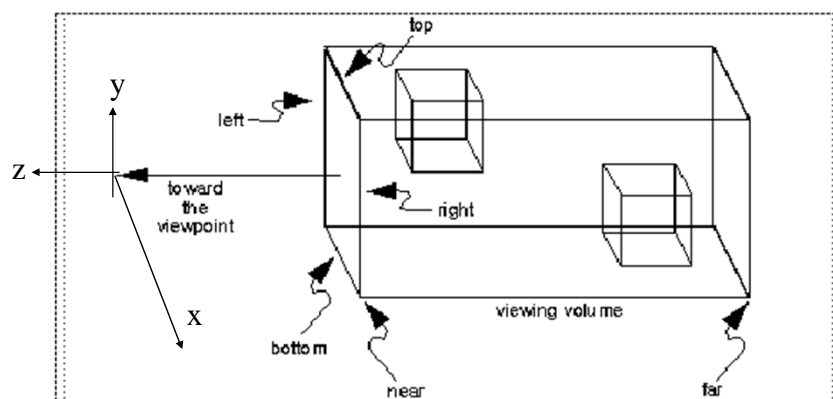
The 3D coordinates can be calculated from the homogeneous coordinates ([perspective division](#), or divide by w):

$$(x_p, y_p, z_p) = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w} \right) = \left(\frac{x}{z/n}, \frac{y}{z/n}, n \right)$$

11

OpenGL Orthographic Projection

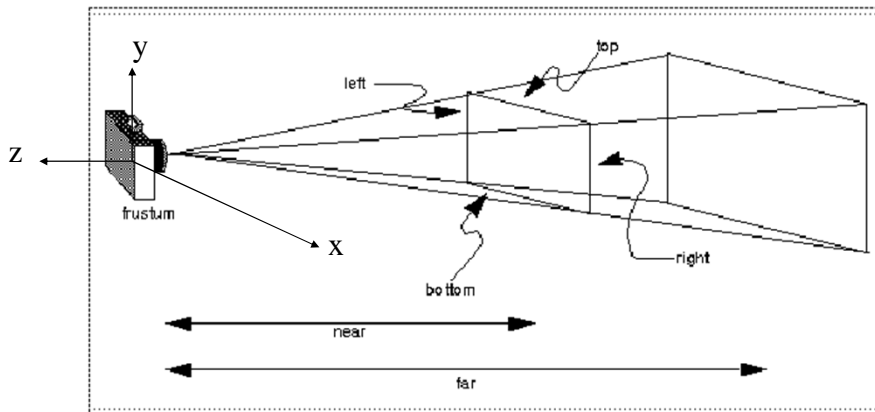
`void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);`



12

OpenGL Perspective Projection

❑ `void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);`

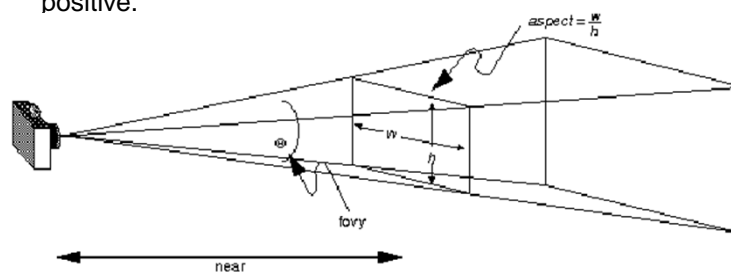


13

Another Perspective Projection

❑ `gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far)`

- fovy -- angle of the field of view in x-z plane (0-180 degree)
- aspect = w / h ;
- znear, zfar -- the distances between the viewpoint and the clipping planes along the negative z-axis. They should be always positive.



14

Perspective Viewing with OpenGL

❑ Positioning and aiming camera

- `glMatrixMode(GL_MODELVIEW);`
`glLoadIdentity();`
`gluLookAt(eye.x, eye.y, eye.z, // eye position`
`look.x, look.y, look.z, // look at point`
`up.x, up.y, up.z) // up vector`
- Up vector is often set to (0, 1, 0)

15

Transformation Matrix for LookAt

❑ Camera coordinate system

- Axes: **u**, **v**, **n**
 $\mathbf{n} = \text{eye} - \text{look}$
 $\mathbf{u} = \mathbf{up} \times \mathbf{n}$
 $\mathbf{v} = \mathbf{n} \times \mathbf{u}$
- Origin: *eye* (looking in the direction $-\mathbf{n}$)
- If $\mathbf{up} = (0, 1, 0)$ then
 $\mathbf{u} = (n_z, 0, -n_x)$ i.e., horizontal
 $\mathbf{v} = (-n_x n_y, n_x^2 + n_z^2, -n_z n_y)$
i.e., more or less upward

16

Transformation Matrix for LookAt

Transformation matrix

$$\mathbf{V} = \begin{pmatrix} u_x & u_y & u_z & -eye \cdot \mathbf{u} \\ v_x & v_y & v_z & -eye \cdot \mathbf{v} \\ n_x & n_y & n_z & -eye \cdot \mathbf{n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Verify that \mathbf{V} transformed

- eye to (0, 0, 0, 1)
- \mathbf{u} to $\mathbf{x} = (1, 0, 0, 0)$
- \mathbf{v} to $\mathbf{y} = (0, 1, 0, 0)$
- \mathbf{n} to $\mathbf{z} = (0, 0, 1, 0)$

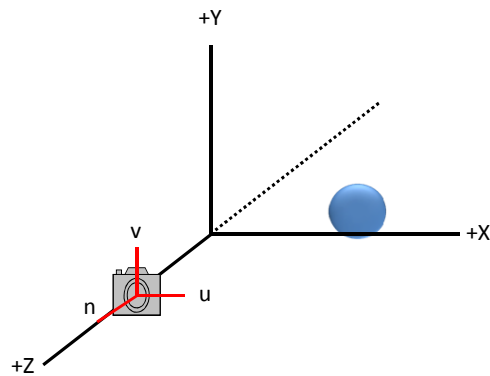
17

Example: 3D Viewing

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
gluLookAt (0, 0, 5, 0, 0, 0, 1, 0);

glTranslatef (3, 0, -2);
glutSolidSphere (1, 10, 10);
```

$$\mathbf{V} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



The sphere in viewing coordinates

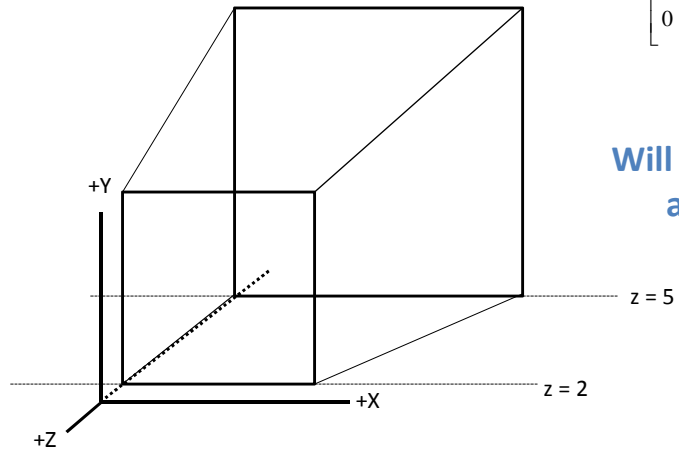
$$\begin{bmatrix} 3 \\ 0 \\ -7 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ -2 \\ 1 \end{bmatrix}$$

18

Example: 3D Viewing

```
glMatrixMode (GL_PROJECTION);  
glLoadIdentity ();  
glFrustum (0, 5, 0, 5, 2, 5);
```

$$M_{pers} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}$$



**Will the sphere
appear?**