

# OpenGL Projection Tutorial

Spring 2008
Utah School of Computing
1

## View Frustum

Parameterized by: `[glFrustum]`

- left, right, top, bottom (generally symmetric)
- near, far

Or, when symmetric, by: `[gluPerspective]`

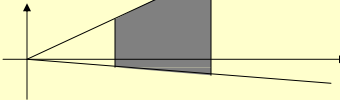
- **Field of view (FOV)**, **aspect ratio**
- near, far
- Aspect ratio is the x/y ratio of the final displayed image. Common values:
  - 4/3 for TV & old movies; 1.66 for cartoons & European movies; 16/9 for American movies & HDTV; 2.35 for epic movies

$$\text{aspect ratio} = \frac{\text{right} - \text{left}}{\text{top} - \text{bottom}} = \frac{\text{right}}{\text{top}}$$

$$\tan(\text{FOV} / 2) = \frac{\text{top}}{\text{near}}$$

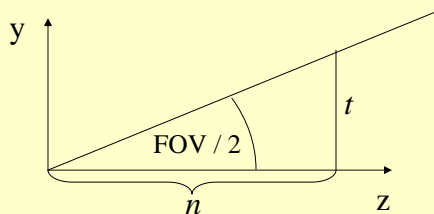
## OpenGL

- `gluPerspective(...)`
  - Field of view in the  $y$  direction,  $FOV$ , (vertical field-of-view)
  - Aspect ratio,  $a$ , should match window aspect ratio
  - Near and far clipping planes,  $n$  and  $f$
  - Defines a symmetric view volume
- `glFrustum(...)`
  - Give the near and far clip plane, and places where the other clip planes cross the near plane
  - Defines the general case
  - Used for stereo viewing, mostly



## gluPerspective to glFrustum

- As noted previously, `glu` functions don't add basic functionality, they are just more convenient
  - So how does `gluPerspective` convert to `glFrustum`?
  - Symmetric, so only need  $t$  and  $l$



## Demo Projection Tutor

## Canonical to Window

- Canonical Viewing Volume (what is it?)
- To Window

$$\mathbf{M}_{window} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{sys} = \mathbf{M}_{window} \mathbf{M}_{persp} \mathbf{M}_{view}$$

## Complete Perspective Projection

- After applying the perspective matrix, we map the orthographic view volume to the canonical view volume:

$$\mathbf{M}_{persp} = \mathbf{M}_O \mathbf{M}_P = \begin{bmatrix} \frac{2}{(r-l)} & 0 & 0 & \frac{(r+l)}{(r-l)} \\ 0 & \frac{2}{(t-b)} & 0 & \frac{(t+b)}{(t-b)} \\ 0 & 0 & \frac{2}{(n-f)} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & (n+f) & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{M}_{sys} = \mathbf{M}_{window} \mathbf{M}_{persp} \mathbf{M}_{view}$$

## Complete Perspective Projection

- After applying the perspective matrix, we map the orthographic view volume to the canonical view volume:

$$\mathbf{M}_{persp} = \mathbf{M}_O \mathbf{M}_P = \begin{bmatrix} \frac{2}{(r-l)} & 0 & 0 & \frac{(r+l)}{(r-l)} \\ 0 & \frac{2}{(t-b)} & 0 & \frac{(t+b)}{(t-b)} \\ 0 & 0 & \frac{2}{(n-f)} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & (n+f) & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{M}_{sys} = \mathbf{M}_{window} \mathbf{M}_{persp} \mathbf{M}_{view}$$

`glViewport()` `gluFrustum()` `gluLookAt()`

## GL Matrix Example

```
// Clear screen
glClear(GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT);

// Set up projection
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fov, aspect, nearclip, farclip);

// Set up camera view
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eye.x, eye.y, eye.z, target.x, target.y, target.z, 0, 1, 0);

// Draw all objects
for(each object) {
    glPushMatrix();
    glTranslatef(pos[i].x, pos[i].y, pos[i].z);
    glRotatef(axis[i].x, axis[i].y, axis[i].z, angle[i]);
    Model[i]->Draw();
    glPopMatrix();
}

// Finish
glFlush();
glSwapBuffers();
```