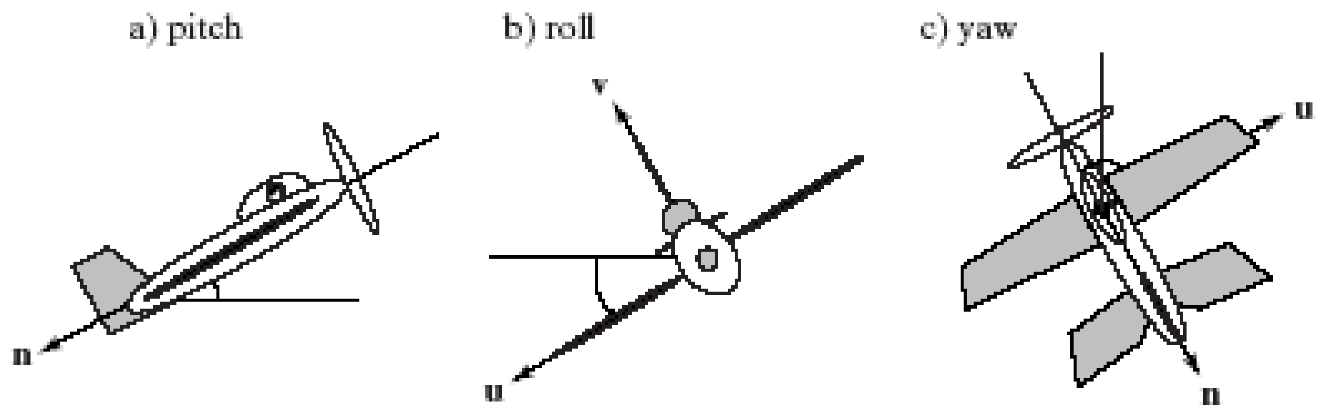
Other Camera Controls

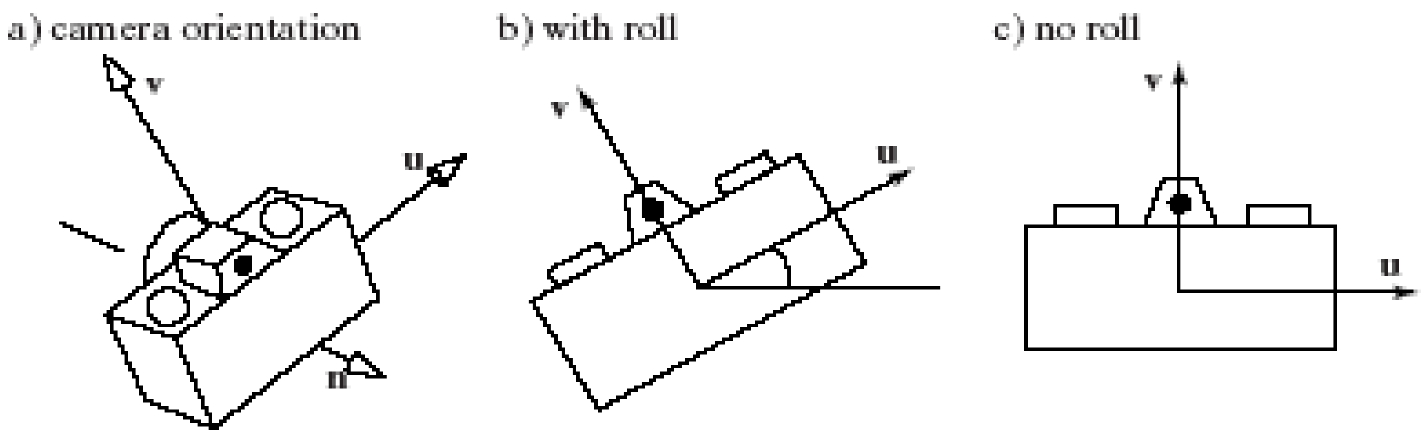
- The LookAt function is only for positioning camera
- Other ways to specify camera position/movement
  - Yaw, pitch, roll
  - Elevation, azimuth, twist
  - Direction angles

Flexible Camera Control

- Sometimes, we want camera to move
- Like controlling an airplane's orientation
- Adopt aviation terms:
  - **Pitch:** nose up-down
  - **Roll:** roll body of plane
  - **Yaw:** move nose side to side

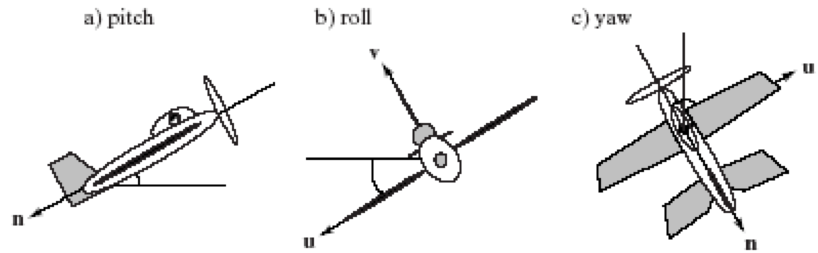


Yaw, Pitch and Roll Applied to Camera



Flexible Camera Control

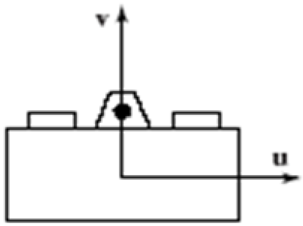
- Create a **camera** class, store **eye** and axes (**u, v, n**)




```
class Camera
   private:
        Point3 eye;
        Vector3 u, v, n;…. etc
```

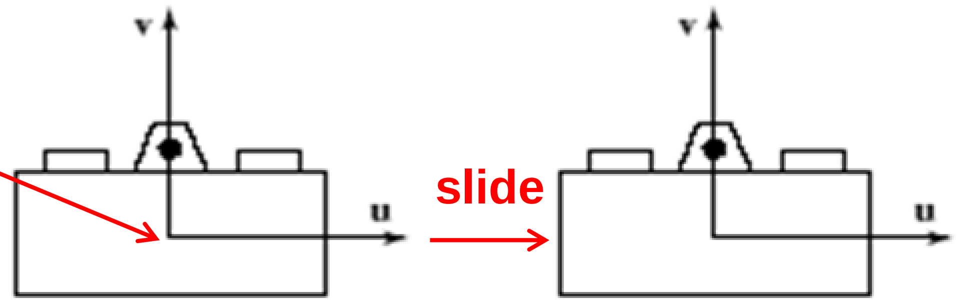
- Camera methods (functions) to specify pitch, roll, yaw. E.g

```
         u  v  n
cam.slide(1, 0, 2); // slide camera right 1 and backward 2
cam.roll(30);   // roll camera 30 degrees
cam.yaw(40);   // yaw camera 40 degrees
cam.pitch(20);  // pitch camera 20 degrees
```



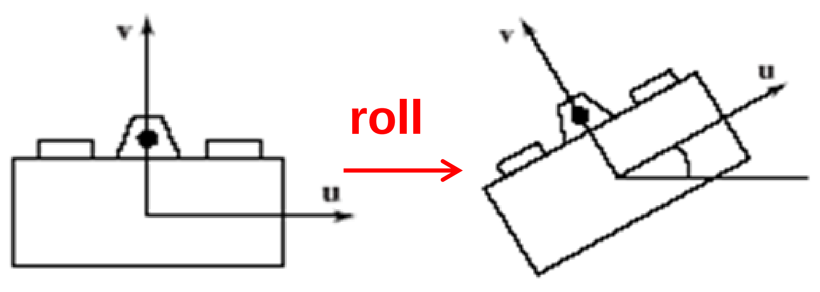
Recall: Final LookAt Matrix

- Slide along u, v or n
- Changes eye position
- Changes these components



$$\left|\begin{array}{ccc|c} ux & uy & uz & -\mathbf{e}\,.\,\mathbf{u} \\ vx & vy & vz & -\mathbf{e}\,.\,\mathbf{v} \\ nx & ny & nz & -\mathbf{e}\,.\,\mathbf{n} \\ 0 & 0 & 0 & 1 \end{array}\right|$$

- Pitch, yaw, roll rotates u, v or n
- Changes u, v or n
- E.g roll changes **u,v** -> **u’,v’**



Implementing Flexible Camera Control

- Camera class: maintains current (u,v,n) and eye position

```
class Camera
private:
        Point3 eye;
        Vector3 u, v, n;…. etc
```

- User inputs desired roll, pitch, yaw angle or slide
  1. **Roll, pitch, yaw:** calculate modified vector (u', v', n')
  2. **Slide:** Calculate new eye position
  3. Update lookAt matrix, Load it into CTM

Example: Camera Slide

- Recall: the axes are unit vectors
- User changes eye by delU, delV or delN
- eye = eye + changes (delU, delV, delN)
- Note: function below combines all slides into one

  E.g moving camera by ***D*** along its u axis = **eye + *D*u**

```
void camera::slide(float delU, float delV, float delN)
{
   eye.x += delU*u.x + delV*v.x + delN*n.x;
   eye.y += delU*u.y + delV*v.y + delN*n.y;
   eye.z += delU*u.z + delV*v.z + delN*n.z;
   setModelViewMatrix( );
}
```

OpenGL Matrices: Column Major

- Slide changes **eVec**,
- roll, pitch, yaw, change **u, v, n**

- Want to update lookAt matrix, store matrices

$$
\begin{vmatrix}
ux & uy & uz & -\mathbf{e} \cdot \mathbf{u} \\
vx & vy & vz & -\mathbf{e} \cdot \mathbf{v} \\
nx & ny & nz & -\mathbf{e} \cdot \mathbf{n} \\
0 & 0 & 0 & 1
\end{vmatrix}
$$

```
0, 4, 8,  12
1, 5, 9,  13
2, 6, 10, 14
3, 7, 11, 15
```

Update matrix elements after slide, pitch, etc

Note: OpenGL matrices are stored in column major order (see above)

Load Matrix into CTM

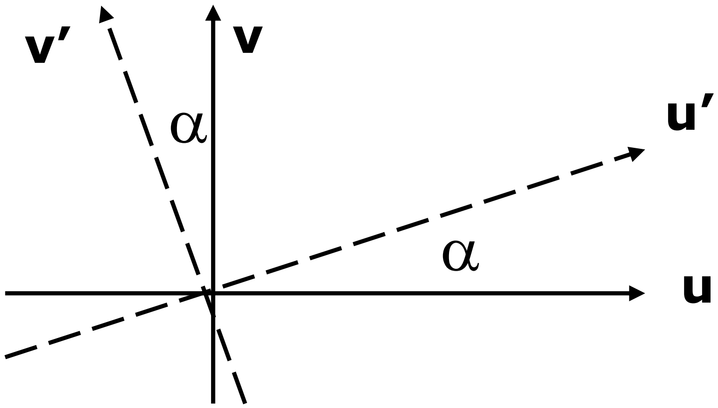
0,	4,	8,	12
1,	5,	9,	13
2,	6,	10,	14
3,	7,	11,	15

$$\begin{array}{cc|cc|} ux & uy & uz & -\mathbf{e}\,.\,\mathbf{u} \\ vx & vy & vz & -\mathbf{e}\,.\,\mathbf{v} \\ nx & ny & nz & -\mathbf{e}\,.\,\mathbf{n} \\ 0 & 0 & 0 & 1 \end{array}$$

```
void Camera::setModelViewMatrix(void)
{  // load modelview matrix with camera values
   mat4 m;
   Vector3 eVec(eye.x, eye.y, eye.z);// eye as vector
   m[0] = u.x; m[4] = u.y; m[8] = u.z;  m[12] = -dot(eVec,u);
   m[1] = v.x; m[5] = v.y; m[9] = v.z;  m[13] = -dot(eVec,v);
   m[2] = n.x; m[6] = n.y; m[10] = n.z; m[14] = -dot(eVec,n);
   m[3] = 0;   m[7] = 0;   m[11] = 0;   m[15] = 1.0;
   CTM = m; // Finally, load matrix m into CTM Matrix
}
```

- **Call setModelViewMatrix after slide, roll, pitch or yaw**

Example: Camera Roll



$$\mathbf{u}' = \cos(\alpha)\mathbf{u} + \sin(\alpha)\mathbf{v}$$

$$\mathbf{v}' = -\sin(\alpha)\mathbf{u} + \cos(\alpha)\mathbf{v}$$

```
void Camera::roll(float angle)
{  // roll the camera through angle degrees
   float cs = cos(3.142/180 * angle);  // cos argument is in radians
   float sn = sin(3.142/180 * angle);
   Vector3 t = u; // remember old u
   u.set(cs*t.x - sn*v.x, cs*t.y - sn.v.y, cs*t.z - sn.v.z);
   v.set(sn*t.x + cs*v.x, sn*t.y + cs.v.y, sn*t.z + cs.v.z)
   setModelViewMatrix( );
}
```

Computer Graphics (CS 543) Lecture 6 (Part 1): Introduction to Projection

Prof Emmanuel Agu

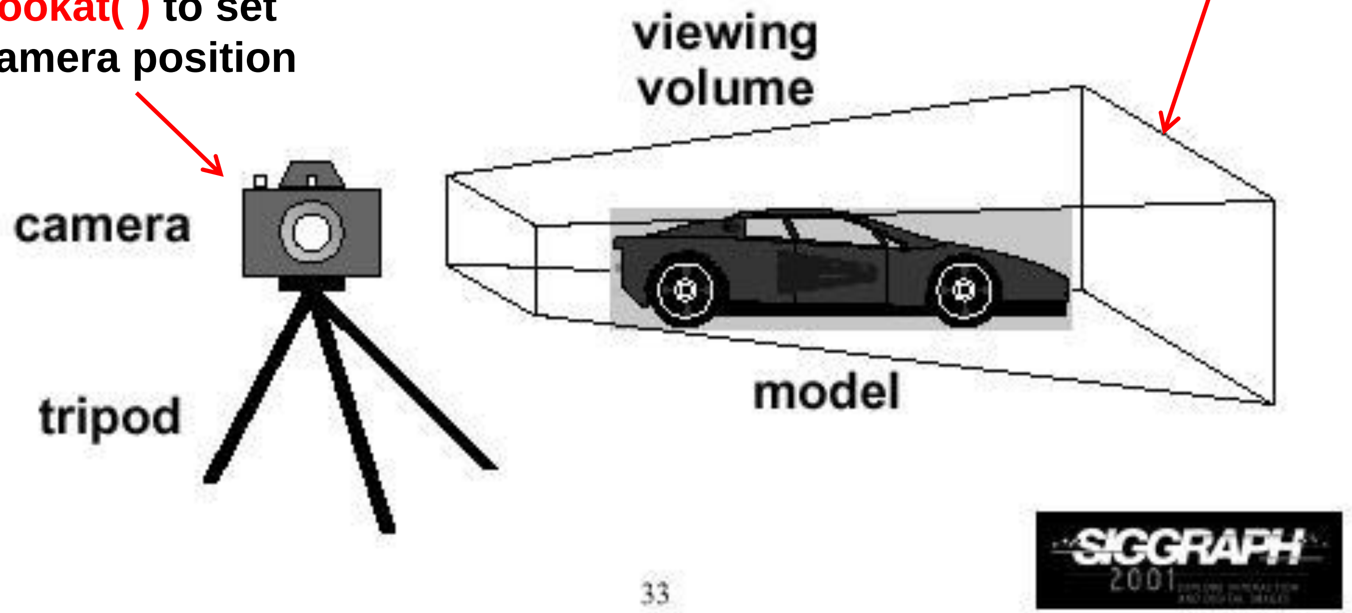
Computer Science Dept.

Worcester Polytechnic Institute (WPI)

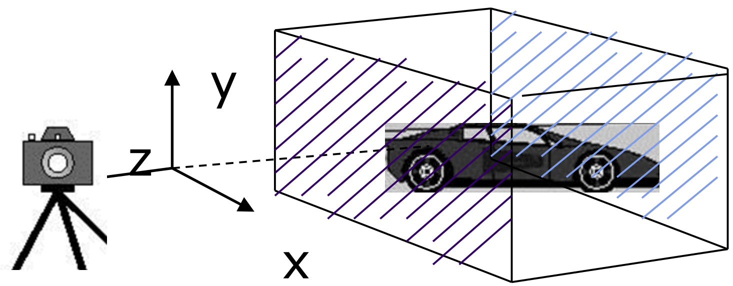
Recall: 3D Viewing and View Volume

Now:
Set view volume

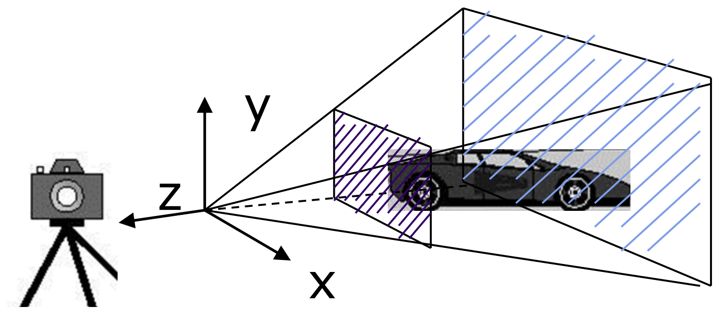
Previously:
Lookat( ) to set camera position



Recall: Different View Volume Shapes



Orthogonal view volume
(no foreshortening)



Perspective view volume
(exhibits foreshortening)

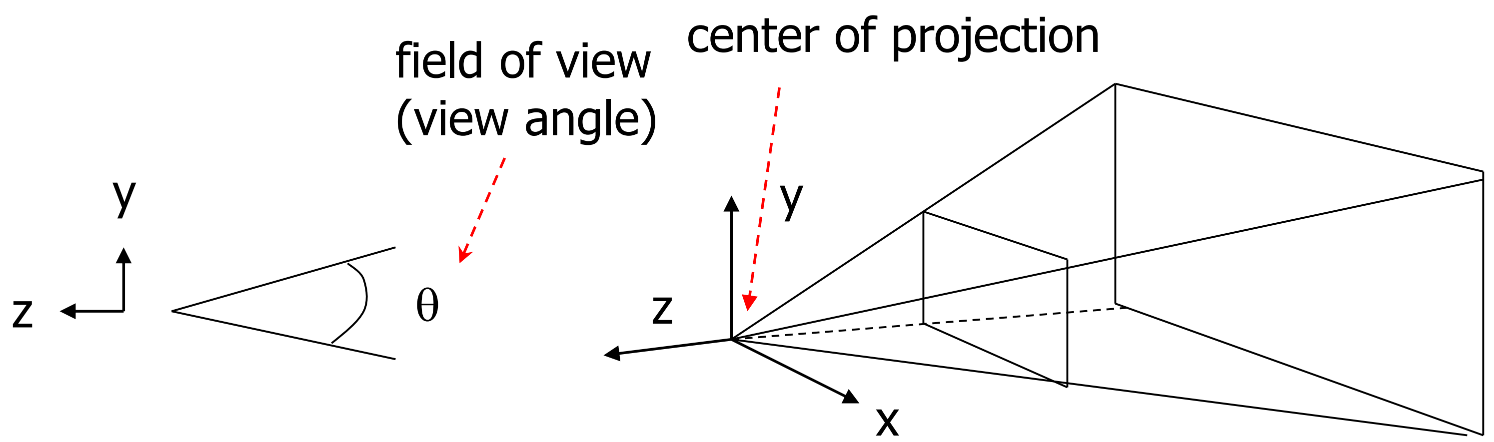
- Different view volume => different look
- **Foreshortening?** Near objects bigger

View Volume Parameters

- Need to set
  - **Projection type:** perspective, orthographic, etc.
  - **View volume parameters:** Field of view and aspect ratio
  - Near and far clipping planes

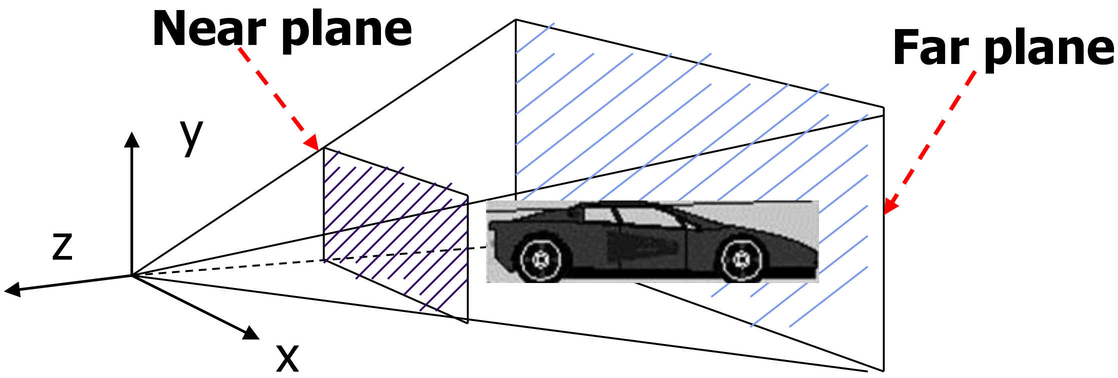
Field of View

- View volume parameter
- Determines how much of world in picture (vertically)
- Larger field of view = smaller objects drawn



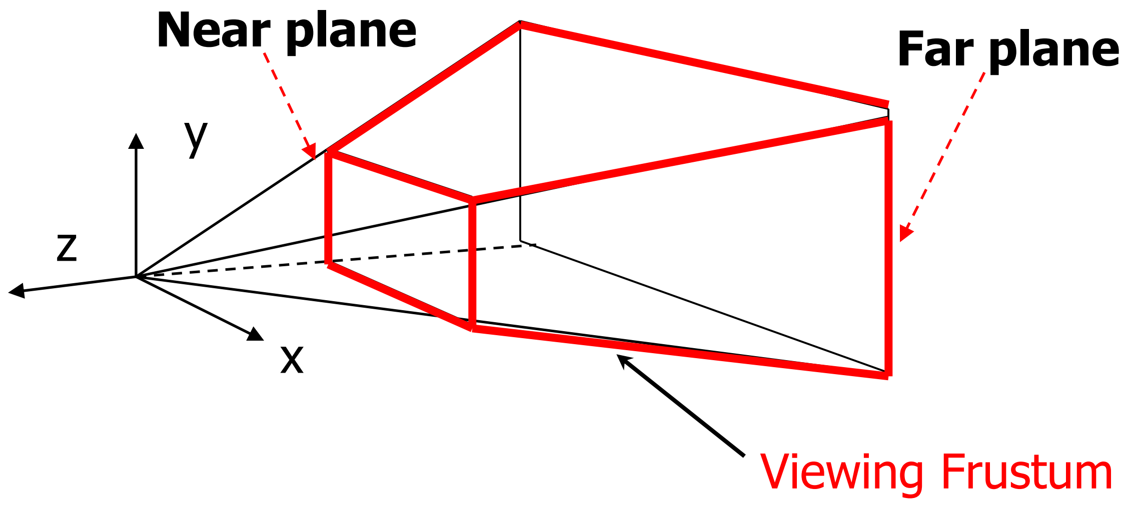
Near and Far Clipping Planes

- Only objects between near and far planes drawn



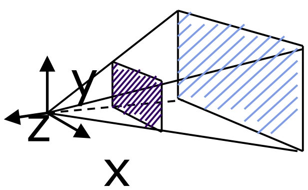
Viewing Frustrum

- Near plane + far plane + field of view = **Viewing Frustum**
- Objects outside the frustum are clipped

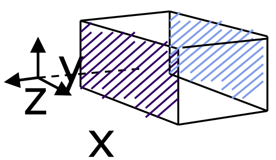


Setting up View Volume/Projection Type

- Previous OpenGL projection commands **deprecated**!!
  - Perspective view volume/projection:
    - **gluPerspective**(fovy, aspect, near, far) **or**
    - **glFrustum**(left, right, bottom, top, near, far)



  - Orthographic:
    - **glOrtho**(left, right, bottom, top, near, far)

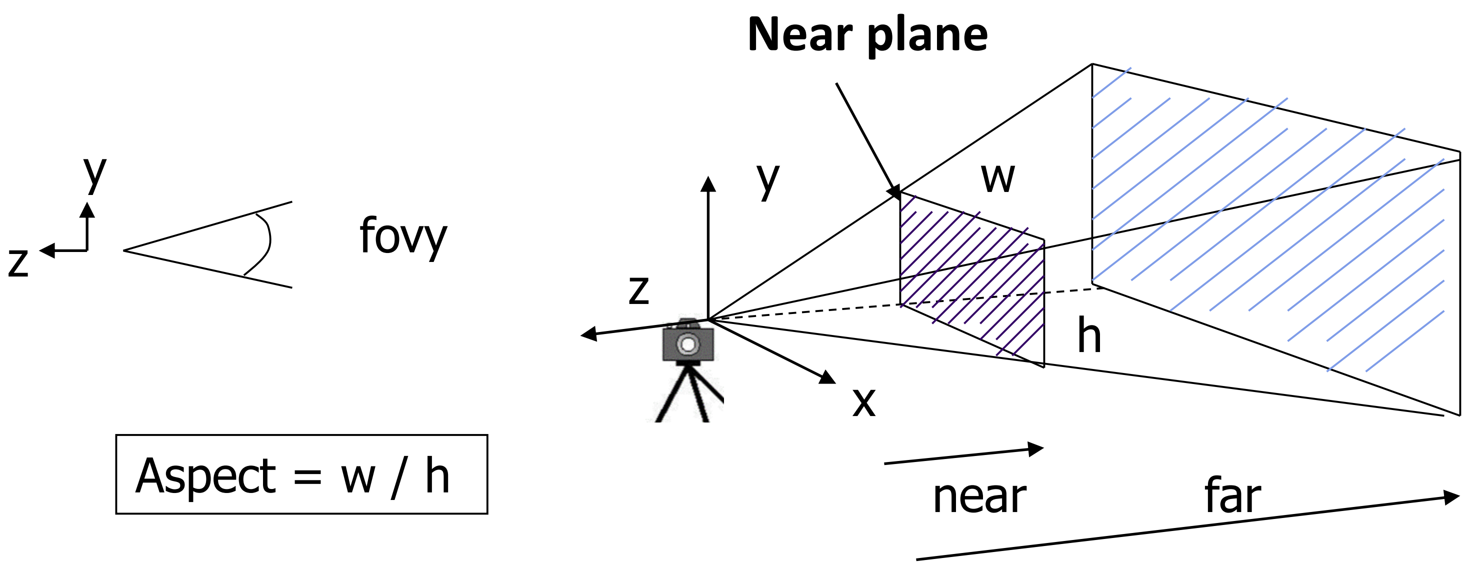


- Useful functions, so we implement similar in `mat.h`:
    - **Perspective**(fovy, aspect, near, far) **or**
    - **Frustum**(left, right, bottom, top, near, far)
    - **Ortho**(left, right, bottom, top, near, far)

What are these arguments? Next!

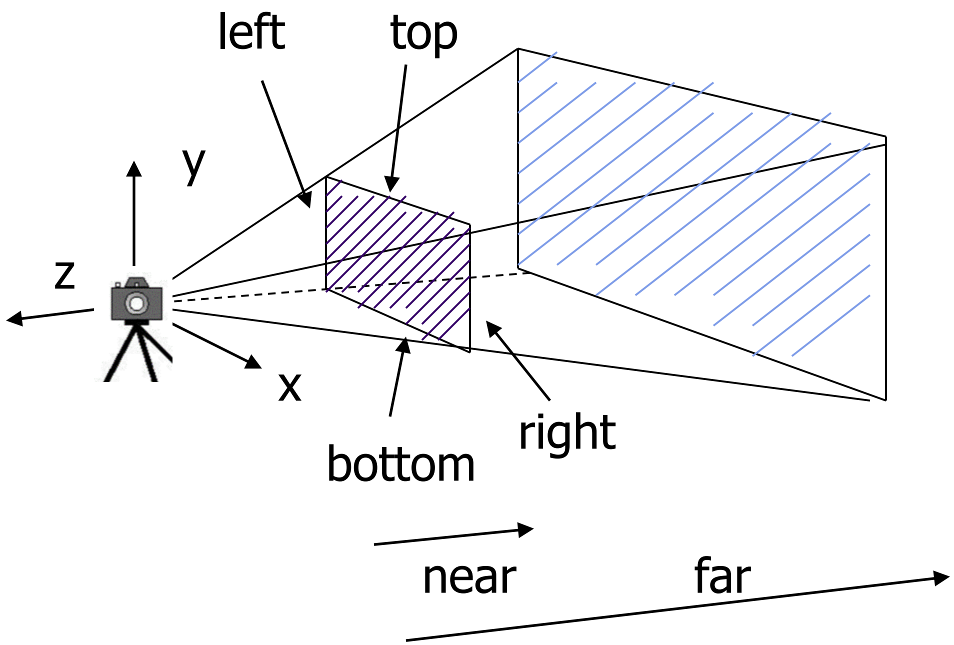
Perspective(fovy, aspect, near, far)

- Aspect ratio used to calculate window width



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

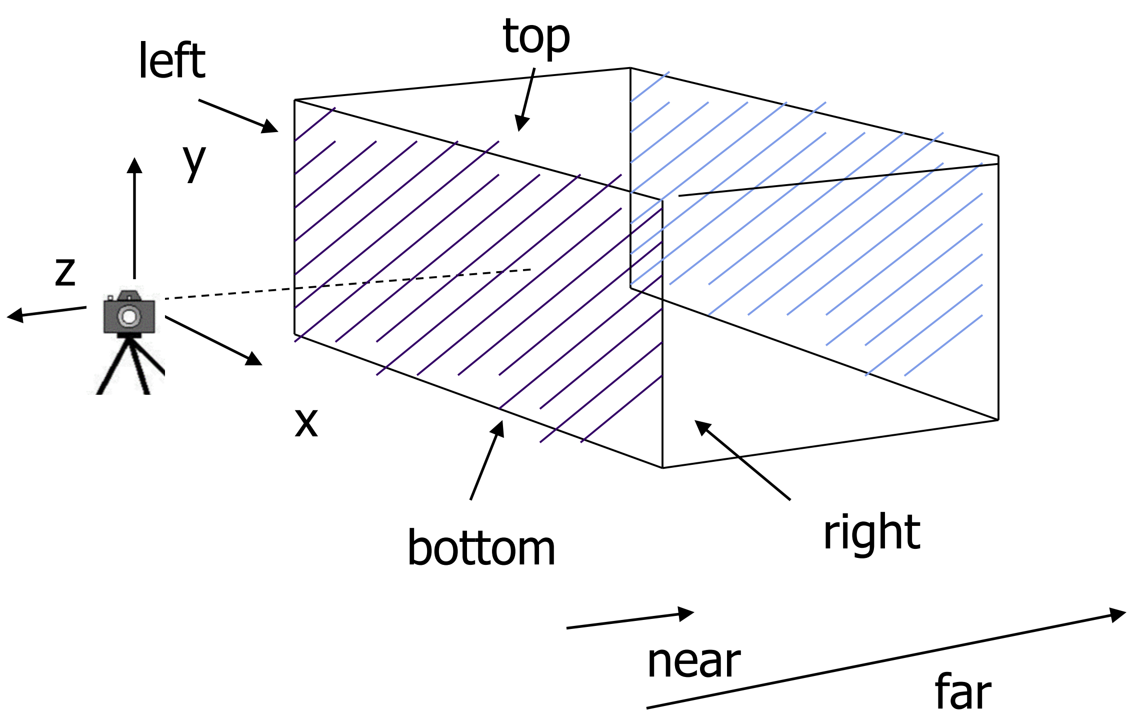
- Can use **Frustrum( )** in place of **Perspective**()
- Same view volume **shape**, different **arguments**



near and **far** measured from camera

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

- For orthographic projection



near and **far** measured <u>from</u> camera

Demo

- Nate Robbins demo on projection

Example Usage:
Setting View Volume/Projection Type

```
void display()
{       // clear screen
        glClear(GL_COLOR_BUFFER_BIT);
        ……….. .
        // Set up camera position
        LookAt(0,0,1,0,0,0,0,1,0);
                 eye    at    up
        ……….. .
        // set up perspective transformation
        Perspective(fovy, aspect, near, far);

         ……….. .
        // draw something
        display_all();     // your display routine
}
```

Implementation

- Set modelview and projection matrices in application program
- Pass matrices to shader

Build 4x4 projection matrix

```
void display( ){
    .....
    model_view = LookAt(eye, at, up);
    projection = Ortho(left, right, bottom,top, near, far);

    // pass model_view and projection matrices to shader
    glUniformMatrix4fv(matrix_loc, 1, GL_TRUE, model_view);
    glUniformMatrix4fv(projection_loc, 1, GL_TRUE, projection);
  .....
}
```

Implementation

- And the corresponding shader

```
in vec4 vPosition;
in vec4 vColor;
Out vec4 color;
uniform mat4 model_view;
Uniform mat4 projection;

void main( )
{
   gl_Position = projection*model_view*vPosition;
   color = vColor;
}
```

References

- Interactive Computer Graphics (6th edition), Angel and Shreiner
- Computer Graphics using OpenGL (3rd edition), Hill and Kelley