

Computer Graphics

CS 543 – Lecture 6 (Part 2)

Projection (Part I)

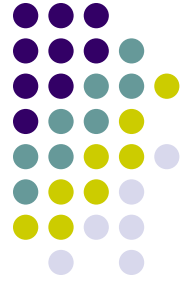
Prof Emmanuel Agu

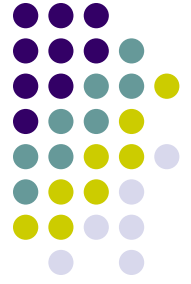
*Computer Science Dept.
Worcester Polytechnic Institute (WPI)*



Objectives

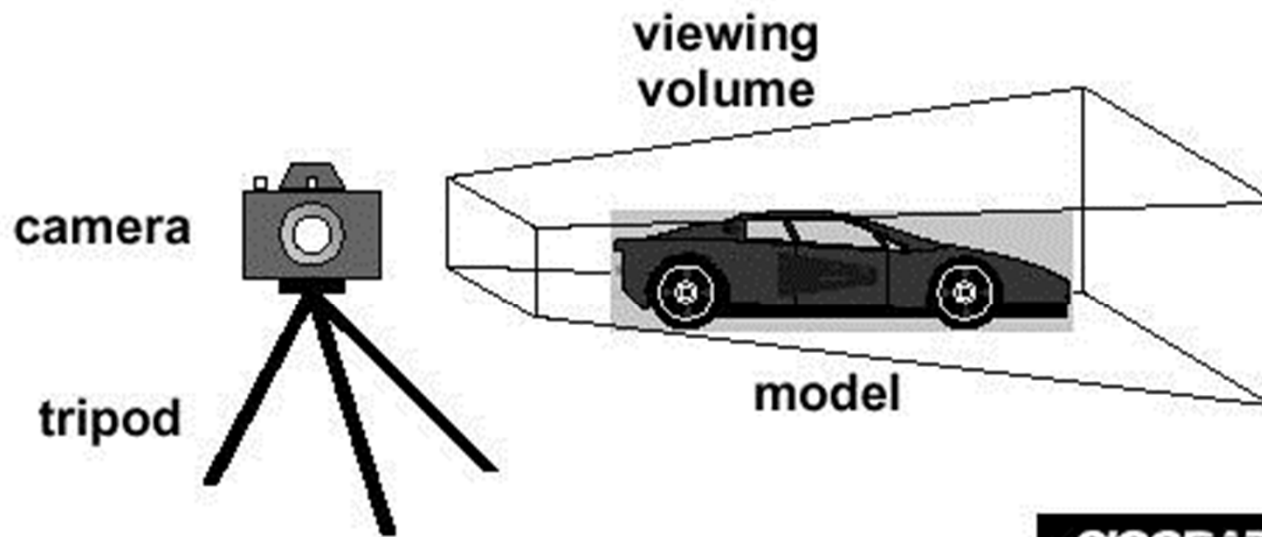
- Understand what is projection?
- Types of projection
 - Orthographic
 - Perspective Projection
- Derive projection matrices
 - Orthographic projection
 - Perspective projection
- Implementation





3D Viewing and View Volume

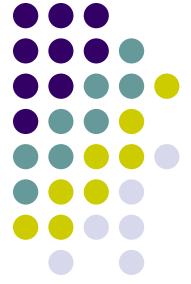
- Recall: 3D viewing set up





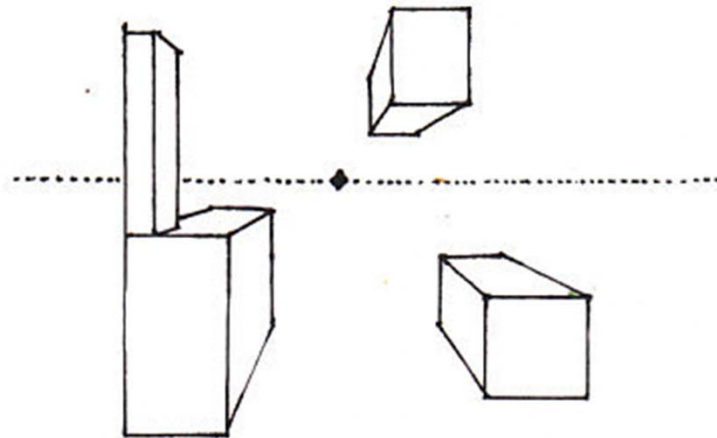
Projection Transformation

- View volume can have different shapes
- Different types of projection:
 - parallel, perspective, etc
- Control view volume parameters
 - Projection type: perspective, orthographic, etc.
 - Field of view and aspect ratio
 - Near and far clipping planes



Perspective Projection

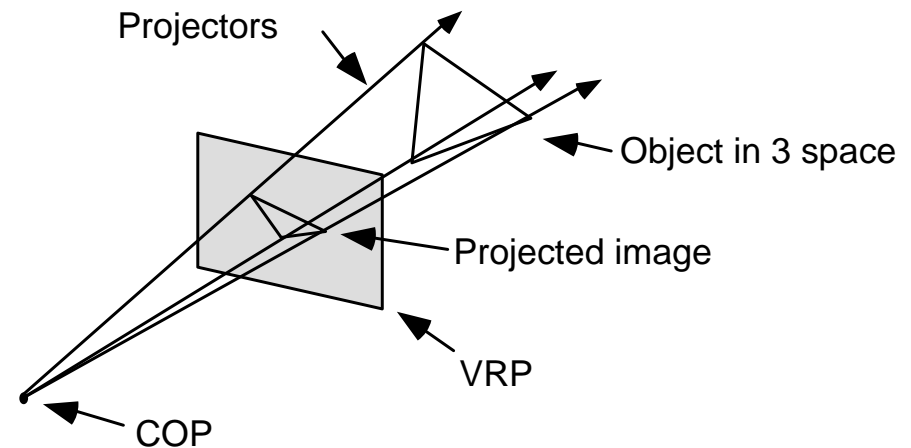
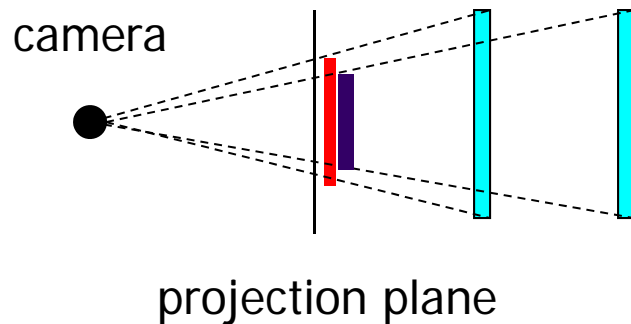
- Similar to real world
- **object foreshortening:** Objects appear larger if closer to camera





Perspective Projection

- Need:
 - Projection center
 - Projection plane
- Projection?
 - Draw line from object to projection center
 - Calculate where each cuts projection plane





Orthographic Projection

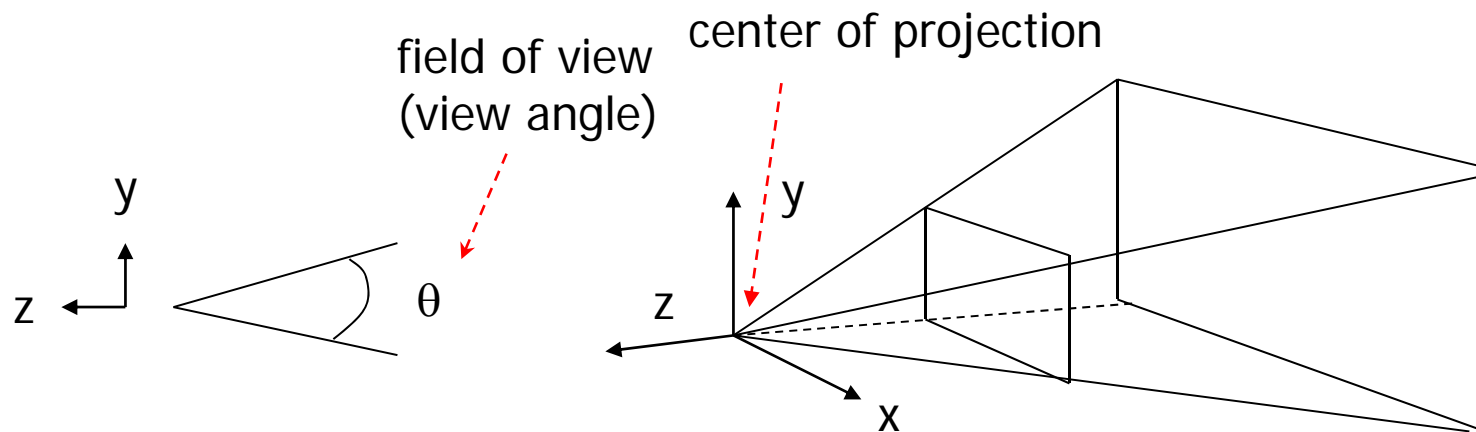
- No foreshortening effect – object distance from camera does not matter
- The projection center is at infinite
- Projection calculation – just drop z coordinates





Field of View

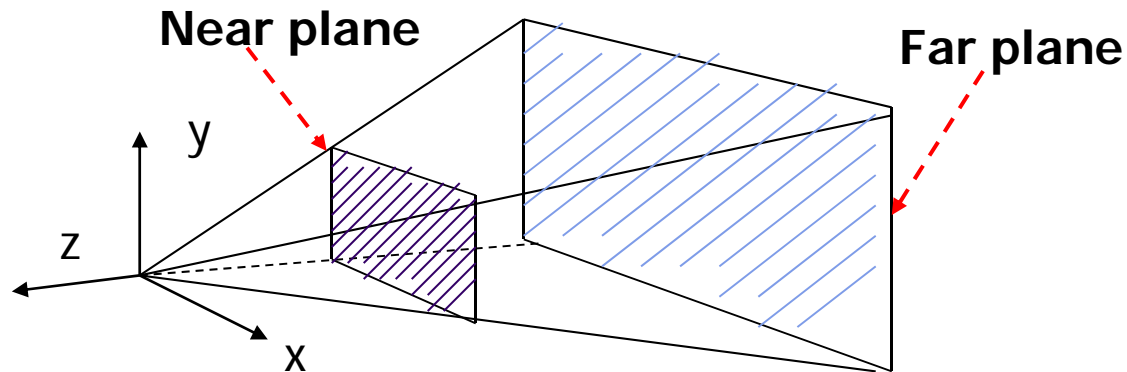
- View volume parameter
- Determines how much of world is taken into picture
- Larger field of view = smaller object projection size





Near and Far Clipping Planes

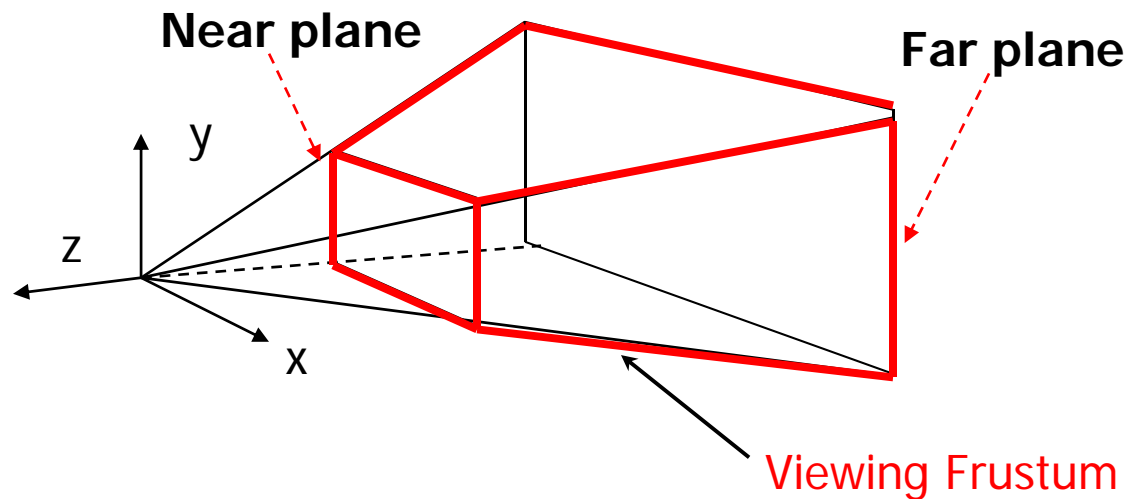
- Only objects between near and far planes are drawn



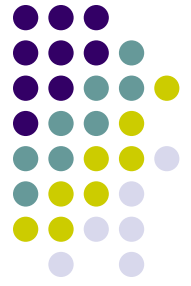


Viewing Frustum

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



Applying Projection Transformation

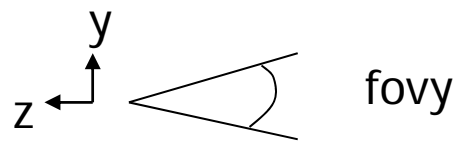


- Previous OpenGL projection commands **deprecated!!**
 - Perspective projection:
 - **gluPerspective**(fovy, aspect, near, far) or
 - **glFrustum**(left, right, bottom, top, near, far)
 - Orthographic:
 - **glOrtho**(left, right, bottom, top, near, far)
- Useful transforms so we implement similar in **mat.h**:
 - **Perspective**(fovy, aspect, near, far) or
 - **Frustum**(left, right, bottom, top, near, far)
 - **Ortho**(left, right, bottom, top, near, far)

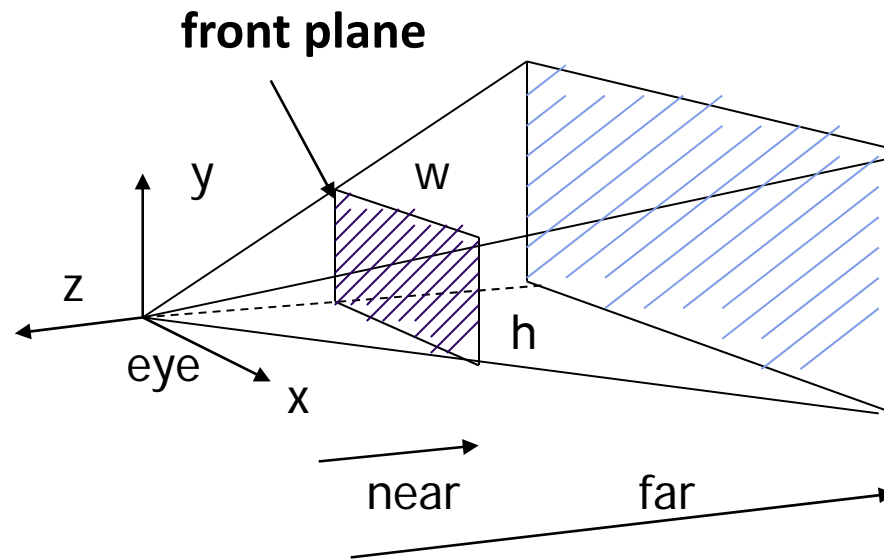


Perspective(fovy, aspect, near, far)

- Aspect ratio is used to calculate the window width



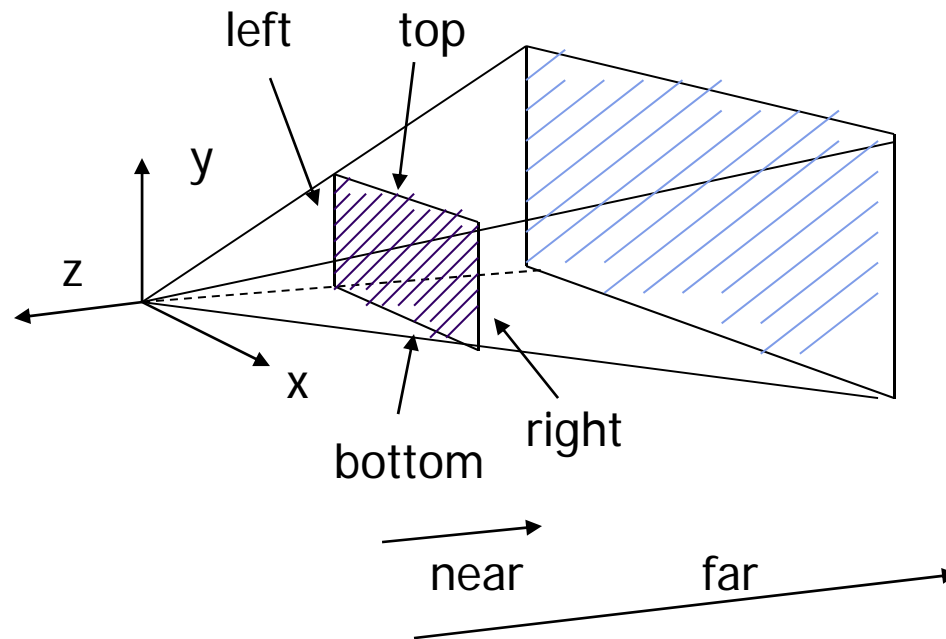
$$\text{Aspect} = w / h$$





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

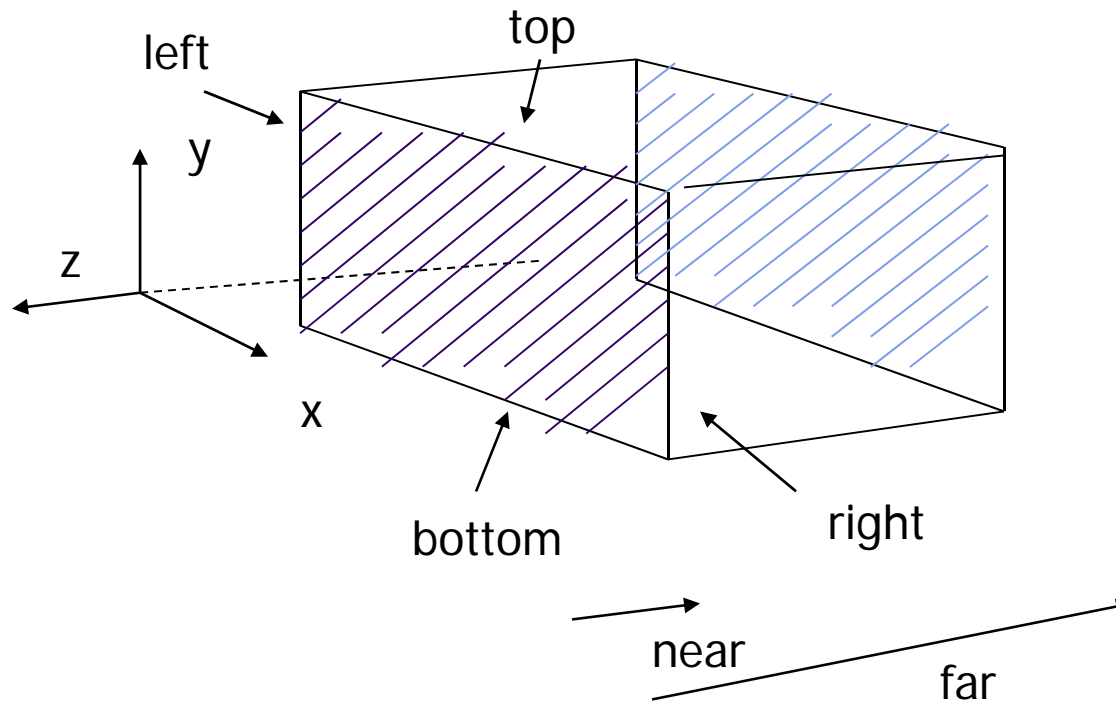
- Can use this function in place of **Perspective()**
- Same functionality, different **arguments**





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

- For orthographic projection



near and far measured from camera

Example Usage: Setting Projection Transformation



```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    .....
    // Set up camera position
    mat4 model_view = LookAt(0,0,1,0,0,0,0,1,0);

    .....
    // set up perspective transformation
    mat4 projection = Perspective(fovy, aspect,
                                   near, far);

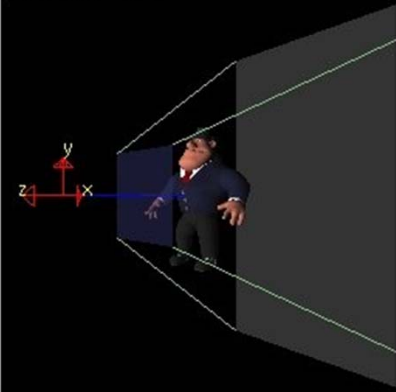
    .....
    // draw something
    display_all();    // your display routine
}
```




Demo

- Nate Robbins demo on projection

World-space view



Screen-space view



Command manipulation window

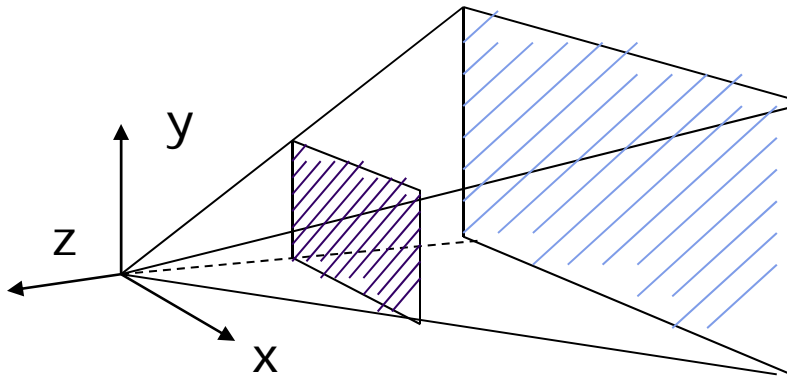
```
fovy aspect zNear zFar
gluPerspective( 60.0 , 1.00 , 1.0 , 10.0 );
gluLookAt( 0.00 , 0.00 , 2.00 , <- eye
          0.00 , 0.00 , 0.00 , <- center
          0.00 , 1.00 , 0.00 ); <- up
```

Click on the arguments and move the mouse to modify values.

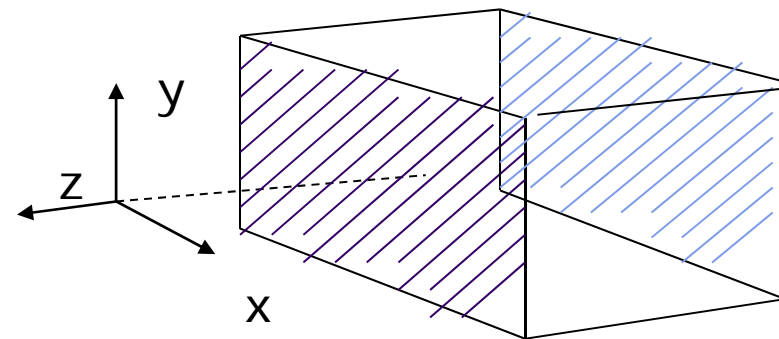


Projection Transformation

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



Perspective: **Perspective()**



Parallel: **Ortho()**



Default Projections and Normalization

- What if you user does not set up projection?
- Default OpenGL projection in eye (camera) frame is orthogonal (Ortho());
- To project points within default view volume

$$x_p = x$$

$$y_p = y$$

$$z_p = 0$$

Homogeneous Coordinate Representation



default orthographic projection

$$\begin{aligned}x_p &= x \\y_p &= y \\z_p &= 0 \\w_p &= 1\end{aligned}$$

$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

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

Default
Projection
Matrix

Vertices **before**
Projection

Vertices **after**
Projection

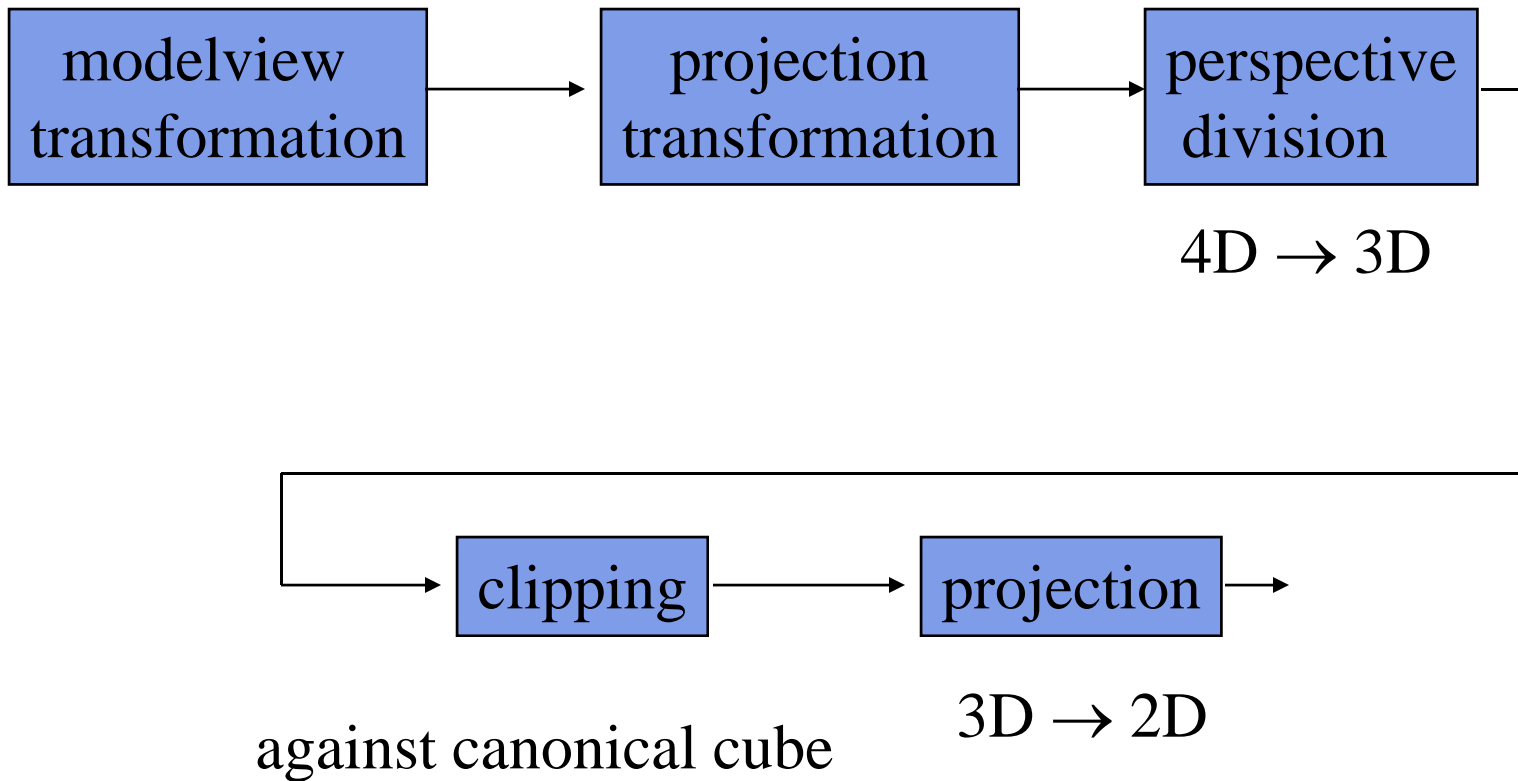
In practice, can let $\mathbf{M} = \mathbf{I}$, set the z term to zero later

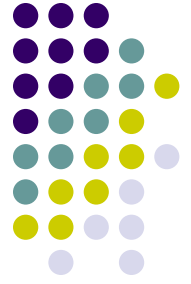


Normalization

- Most graphics systems use *view normalization*
- Instead of deriving different projection matrix for each type of projection
 - **Normalization:** convert all other projection types to orthogonal projections with the default view volume
- Specifically, **projection transform matrices** convert other projection types to default view volume
- Allows use of the same rendering pipeline for different projection types
- Later, makes for efficient clipping

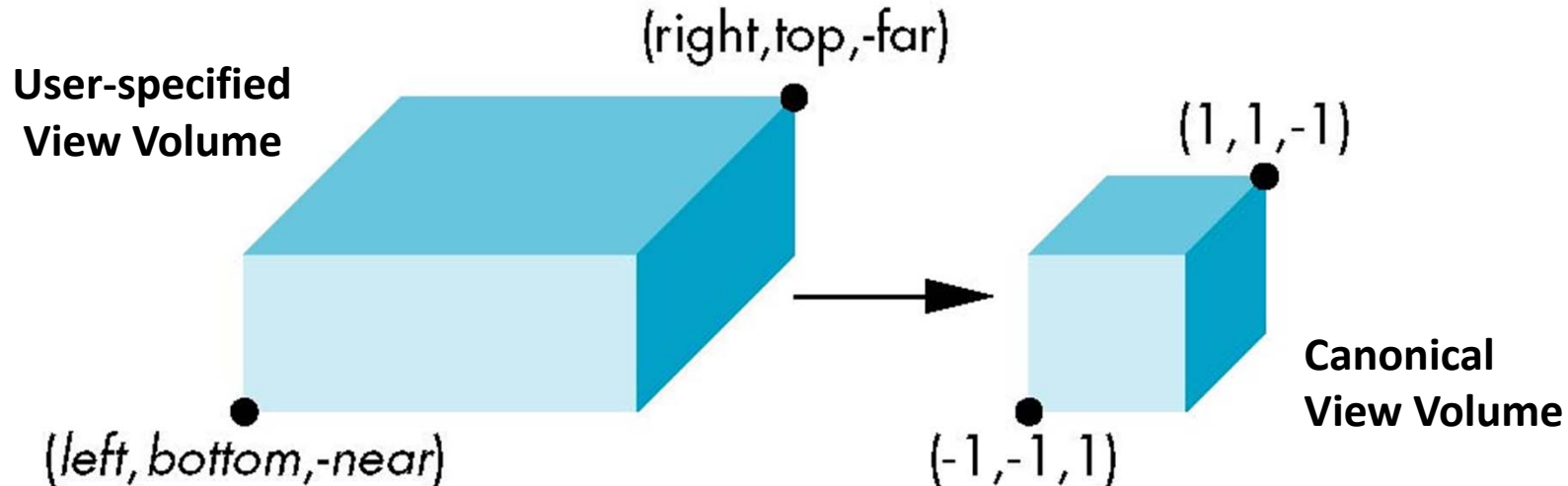
Pipeline View





Parallel Projection

- **Approach:** Project everything in the visible volume into a **canonical view volume (cube)**
- **normalization** \Rightarrow find 4x4 matrix to convert specified view volume to default



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



Parallel Projection: Ortho

- Parallel projection can be broken down into two parts
 1. **Translation:** which centers view volume at origin
 2. **Scaling:** which reduces cuboid of arbitrary dimensions to canonical cube (dimension 2, centered at origin)



Parallel Projection: Ortho

- Translation sequence moves midpoint of view volume to coincide with origin:
- E.g. midpoint of $x = (right + left)/2$
- Thus translation factors:
 $-(right + left)/2, -(top + bottom)/2, -(far+near)/2$
- And translation matrix M1:

$$\begin{pmatrix} 1 & 0 & 0 & -(right + left) / 2 \\ 0 & 1 & 0 & -(top + bottom) / 2 \\ 0 & 0 & 1 & -(far + near) / 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Parallel Projection: Ortho

- Scaling factor is ratio of cube dimension to Ortho view volume dimension
- Scaling factors:
 $2/(\text{right} - \text{left})$, $2/(\text{top} - \text{bottom})$, $2/(\text{far} - \text{near})$
- Scaling Matrix M2:

$$\begin{pmatrix} \frac{2}{\text{right} - \text{left}} & 0 & 0 & 0 \\ 0 & \frac{2}{\text{top} - \text{bottom}} & 0 & 0 \\ 0 & 0 & \frac{2}{\text{far} - \text{near}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parallel Projection: Ortho



Concatenating M1xM2, we get transform matrix used by glOrtho

$$\begin{pmatrix} \frac{2}{right - left} & 0 & 0 & 0 \\ 0 & \frac{2}{top - bottom} & 0 & 0 \\ 0 & 0 & \frac{2}{far - near} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & -(right + left) / 2 \\ 0 & 1 & 0 & -(top + bottom) / 2 \\ 0 & 0 & 1 & -(far + near) / 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right - left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & \frac{2}{near - far} & \frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Final Ortho Projection

- Set $z = 0$
- Equivalent to the homogeneous coordinate transformation

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Hence, general orthogonal projection in 4D is
 $\mathbf{P} = \mathbf{M}_{\text{orth}} \mathbf{S} \mathbf{T}$



References

- Angel and Shreiner, Chapter 4
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition