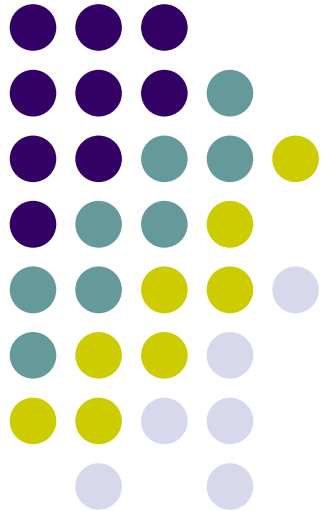
Computer Graphics (CS 543)
Lecture 6 (Part 3): Projection (Part I)

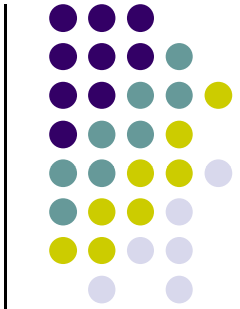
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)



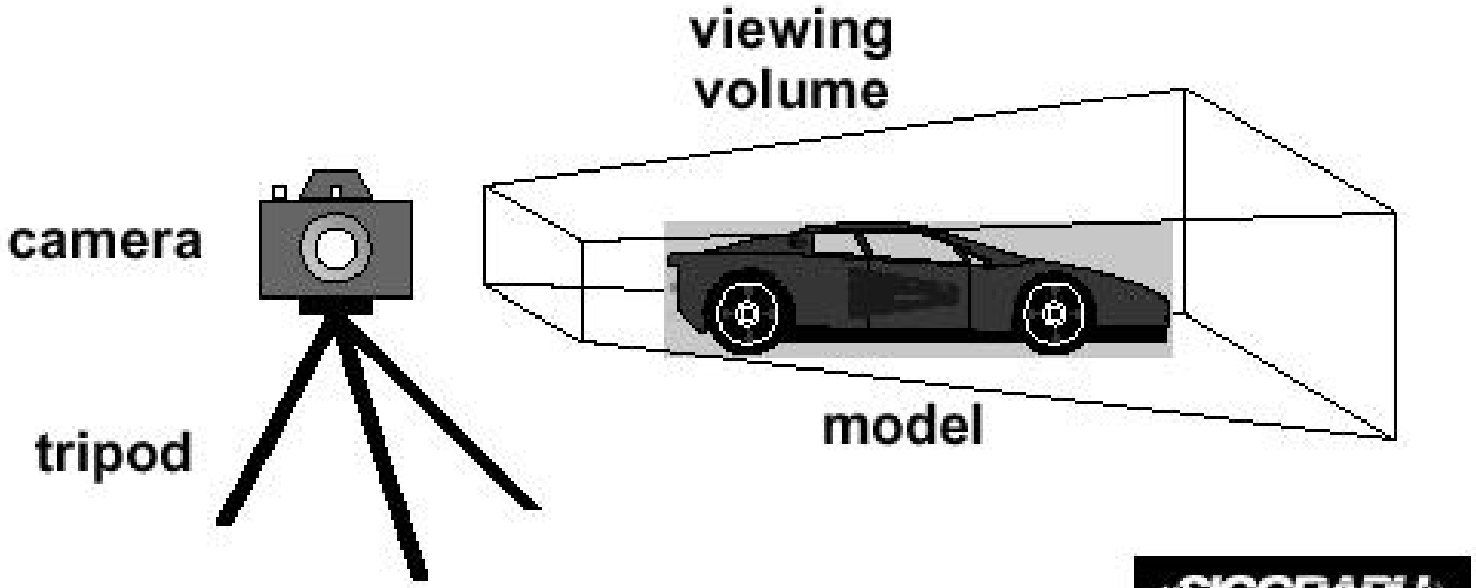
Objectives



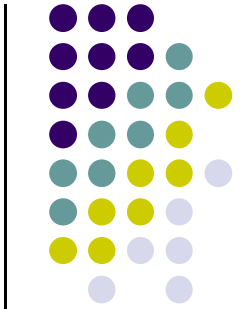
- Understand what is projection?
- Types of projection
  - Orthographic
  - Perspective Projection
- Derive projection matrices
  - Orthographic projection
  - Perspective projection
- Implementation

3D Viewing and View Volume

- Recall: 3D viewing set up



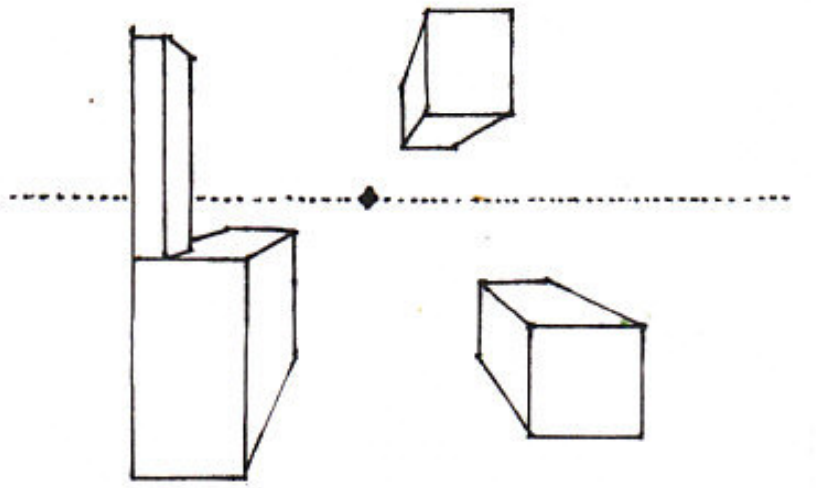
Projection Transformation



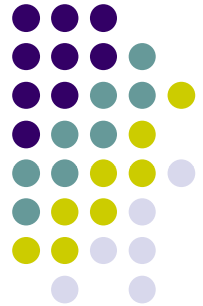
- View volume can have different shapes
- Different types of projection:
  - parallel, perspective, etc
- Control view volume parameters
  - Projection type: perspective, orthographic, etc.
  - Field of view and aspect ratio
  - Near and far clipping planes

Perspective Projection

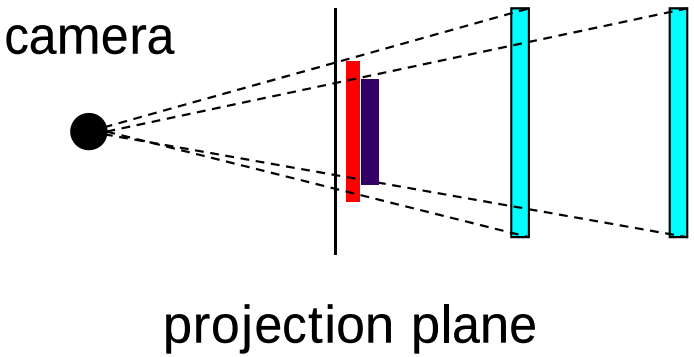
- Similar to real world
- **object foreshortening:** Objects appear larger if closer to camera

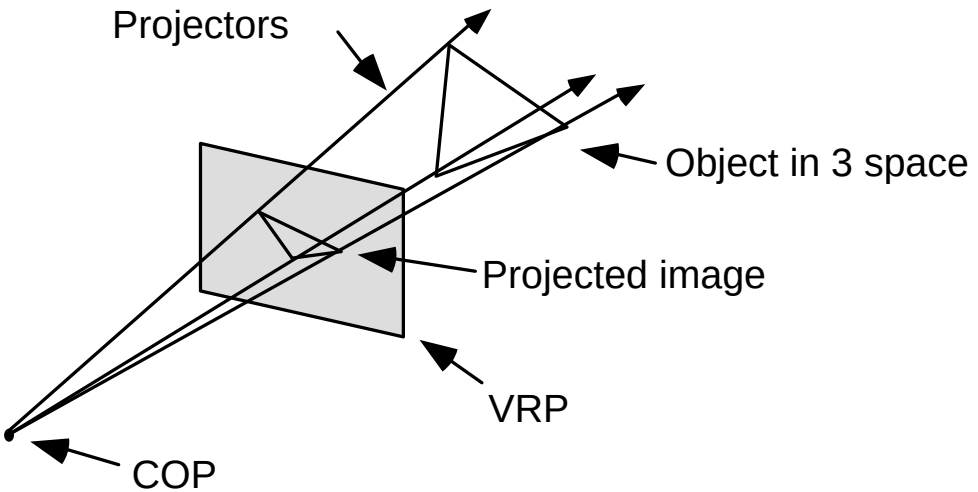


Perspective Projection



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



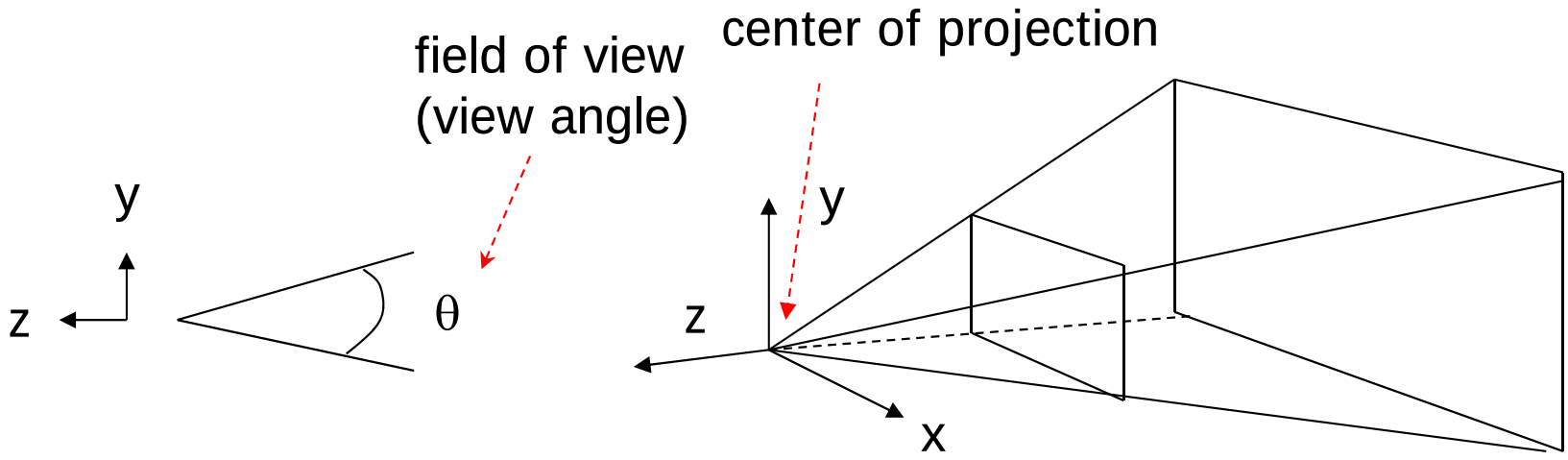


Orthographic Projection

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

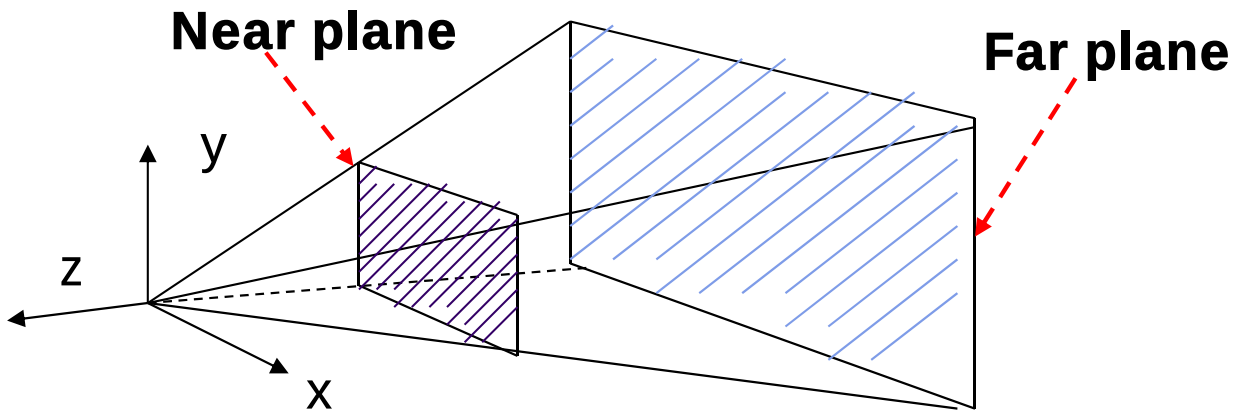
Field of View

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



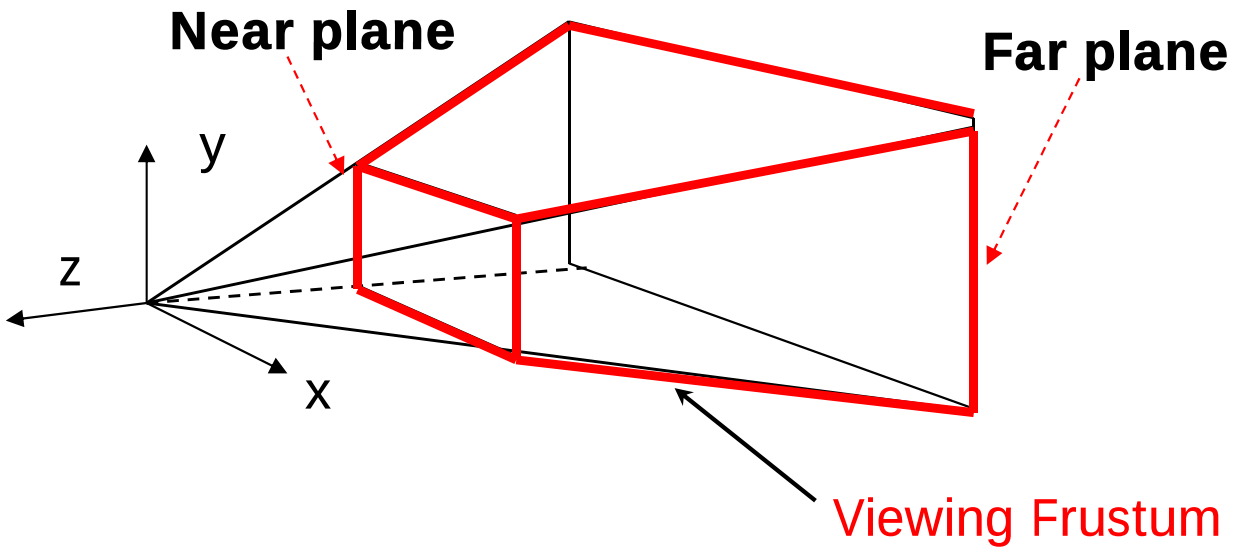
Near and Far Clipping Planes

- Only objects between near and far planes are drawn
- Near plane + far plane + field of view = **Viewing Frustum**



Viewing Frustrum

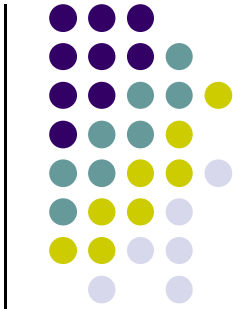
- Objects outside the frustum are clipped



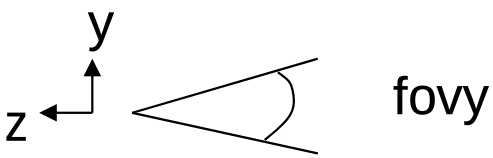
Applying Projection Transformation

- 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)

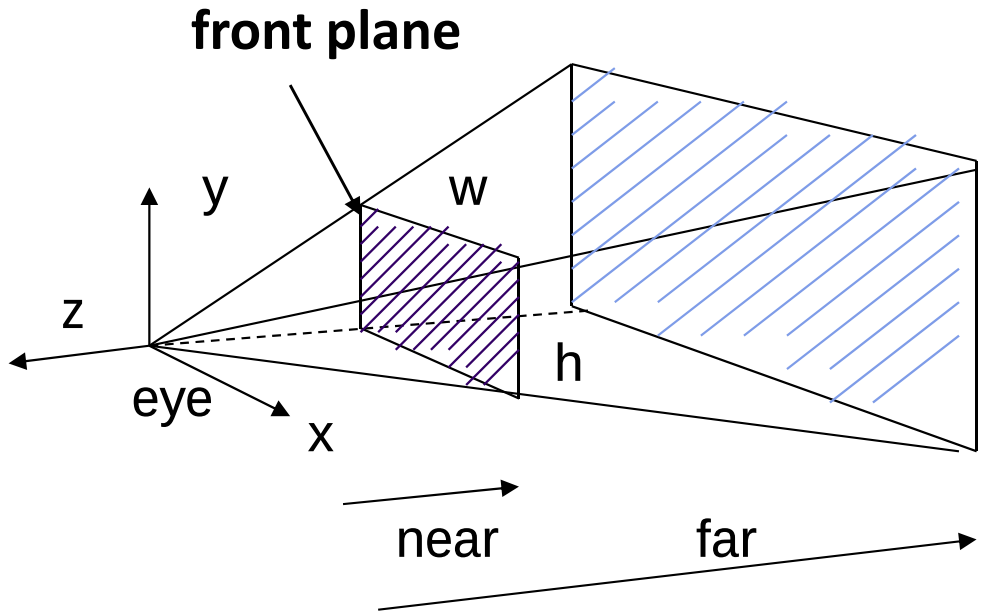
Perspective(fovy, aspect, near, far)



- Aspect ratio is used to calculate the window width

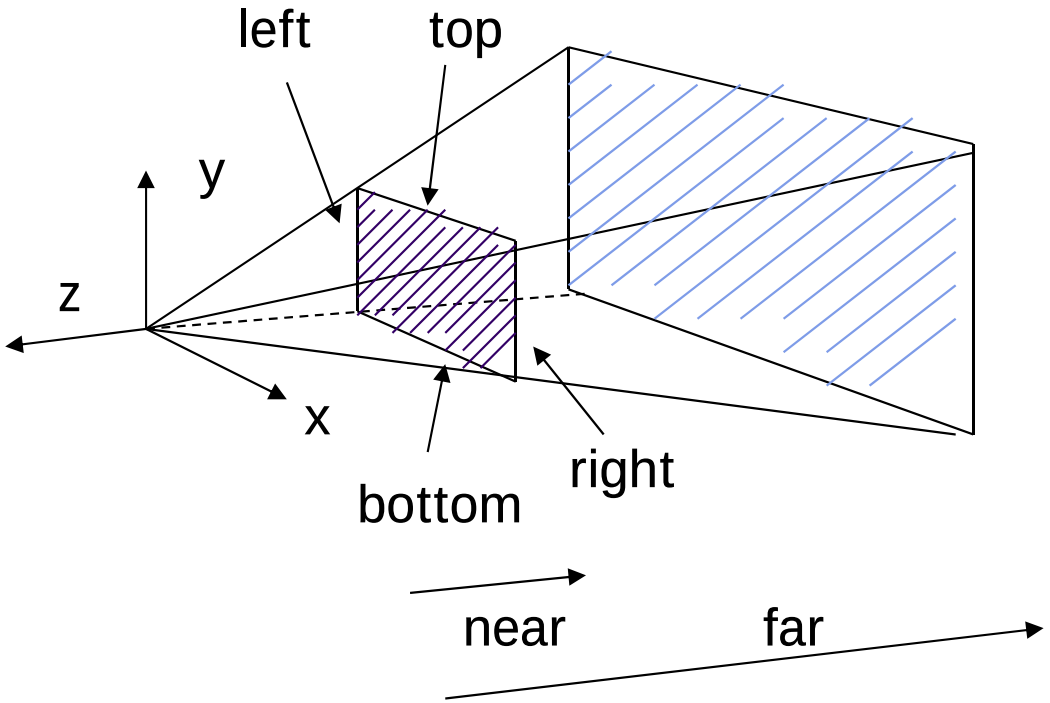


Aspect = w / h

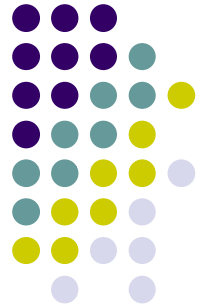


Frustum(left, right, bottom, top, near, far)

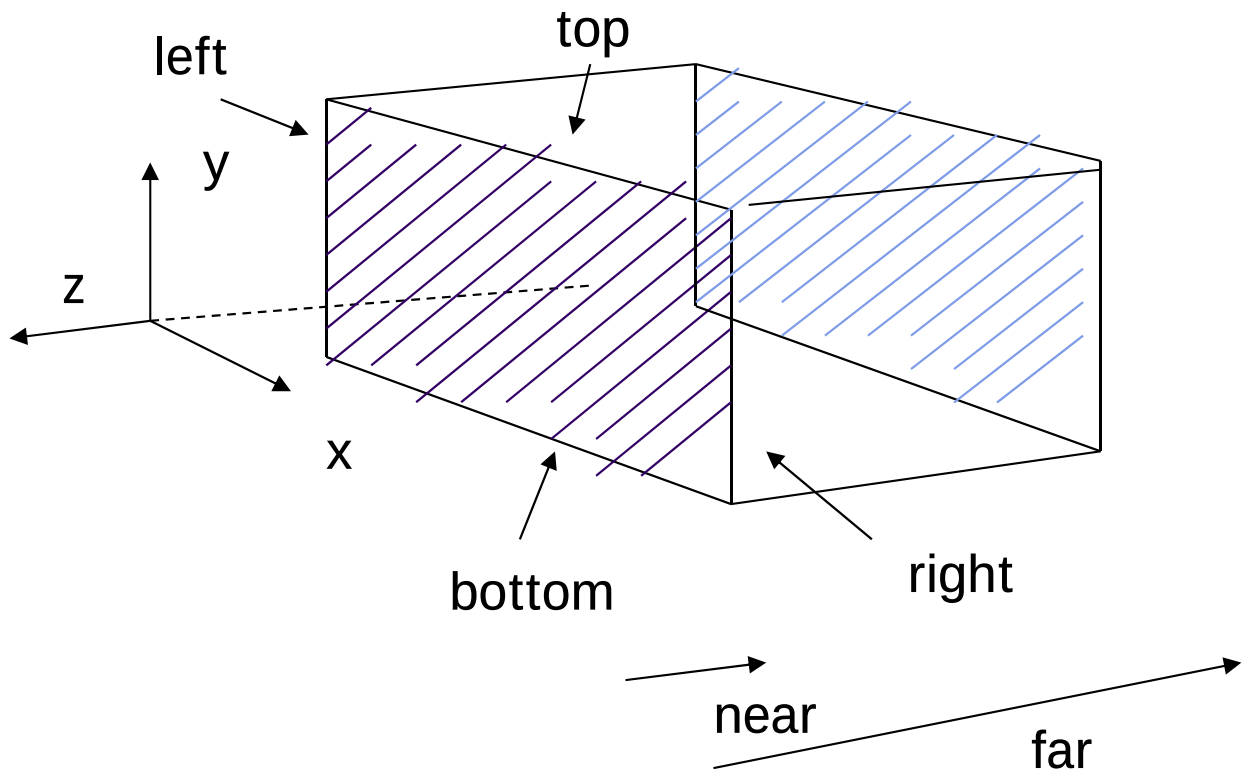
- Can use this function in place of **Perspective**()
- Same functionality, different **arguments**



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



- For orthographic projection

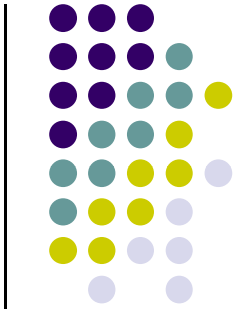


near and **far** measured from camera

Example Usage:
Setting Projection Transformation

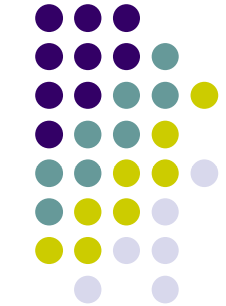
```
void display()
{
      glClear(GL_COLOR_BUFFER_BIT);
      ……..
      // Set up camera position
      LookAt(0,0,1,0,0,0,0,1,0);

      ……..
      // set up perspective transformation
      Perspective(fovy, aspect, near, far);

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

Demo

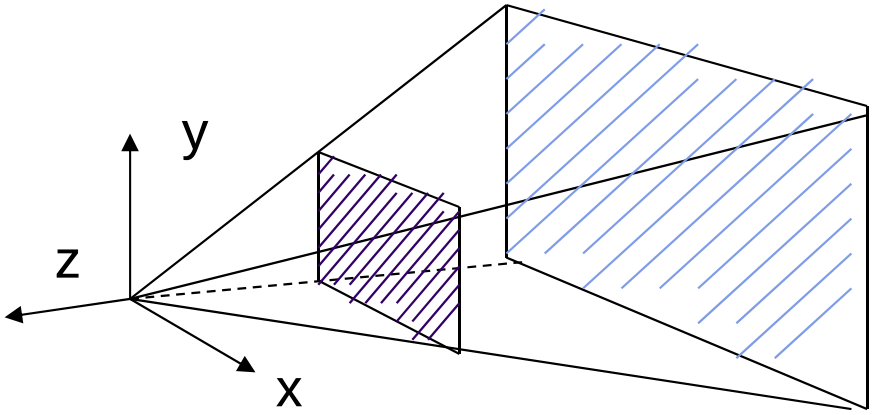


- Nate Robbins demo on projection

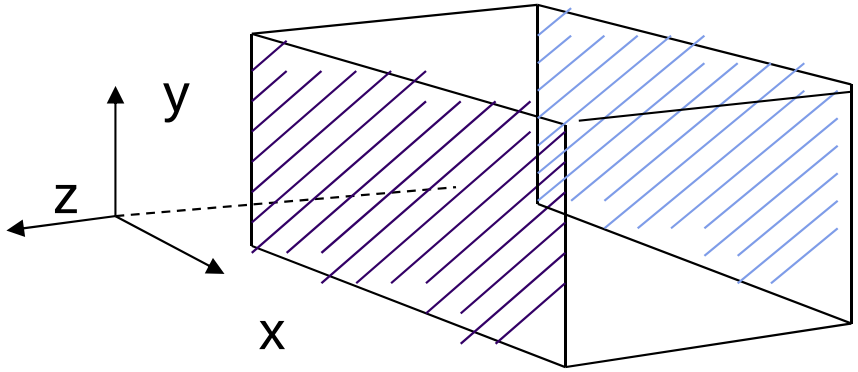
Projection Transformation



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

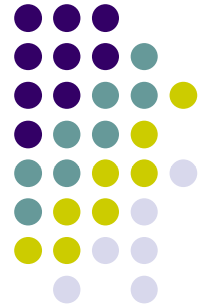


Perspective: **Perspective()**



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

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$
$$w_p = 1$$

Vertices **before** Projection

Vertices **after** Projection

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

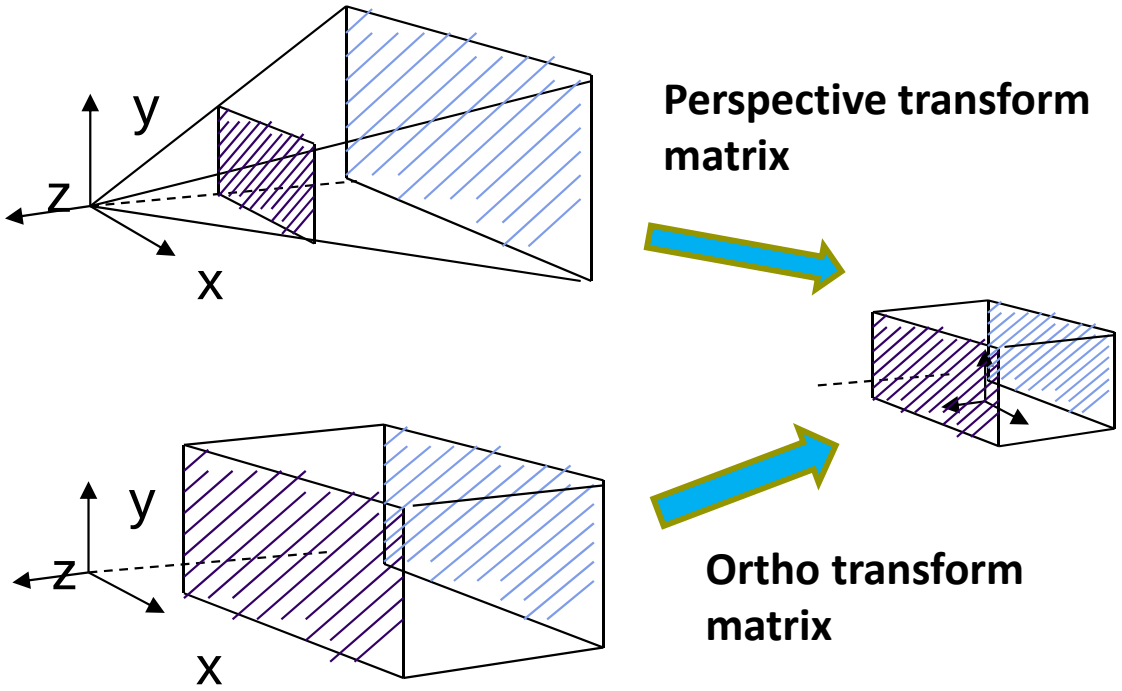
$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

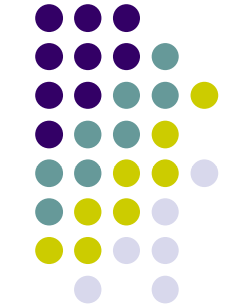
Default Projection Matrix

In practice, can let **M** = **I,** set the *z* term to zero later

Normalization

- Most graphics systems use *view normalization*
- **Normalization:** convert all other projection types to orthogonal projections with the default view volume





References

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