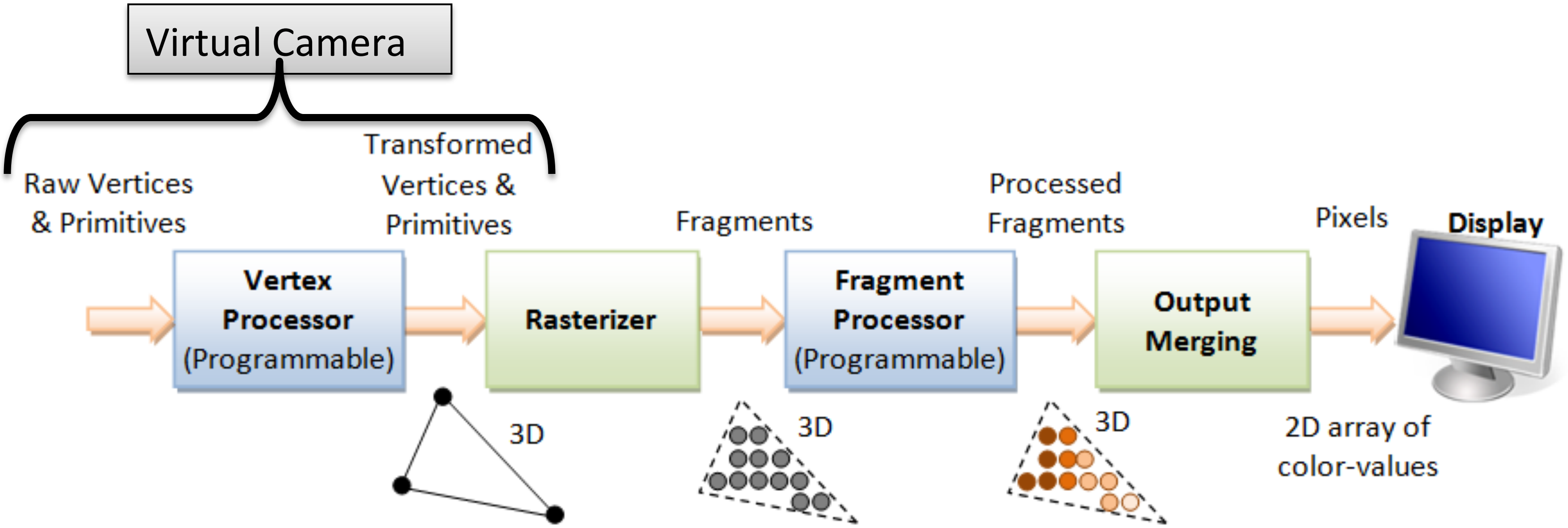
Viewing

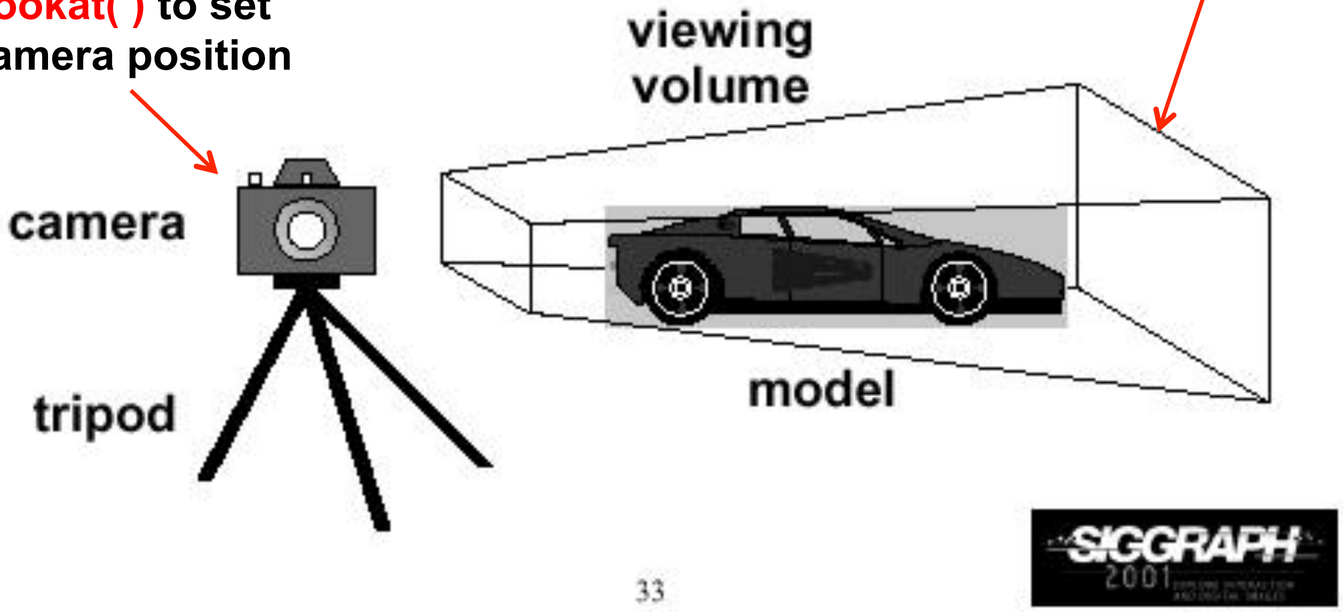
Cliff Lindsay, Ph.D.
WPI

Building Virtual Camera Pipeline

- Used To View Virtual Scene
- First Half of Rendering Pipeline Related To Camera
- Takes Geometry From Application To Rasterization Stages



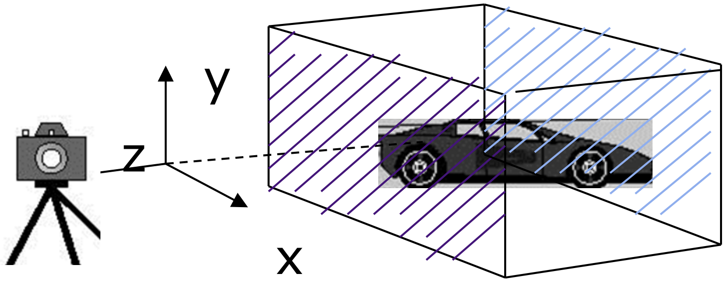
3D Viewing and View Volume



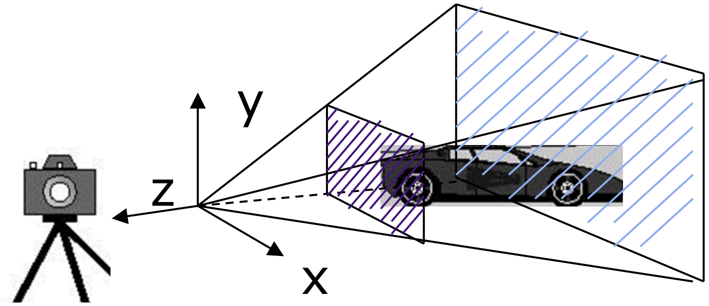
Previously:
Lookat( ) to set
camera position

Now:
Set view volume

Different View Volume Shapes



Orthogonal view volume
(no foreshortening)



Perspective view volume
(exhibits foreshortening)

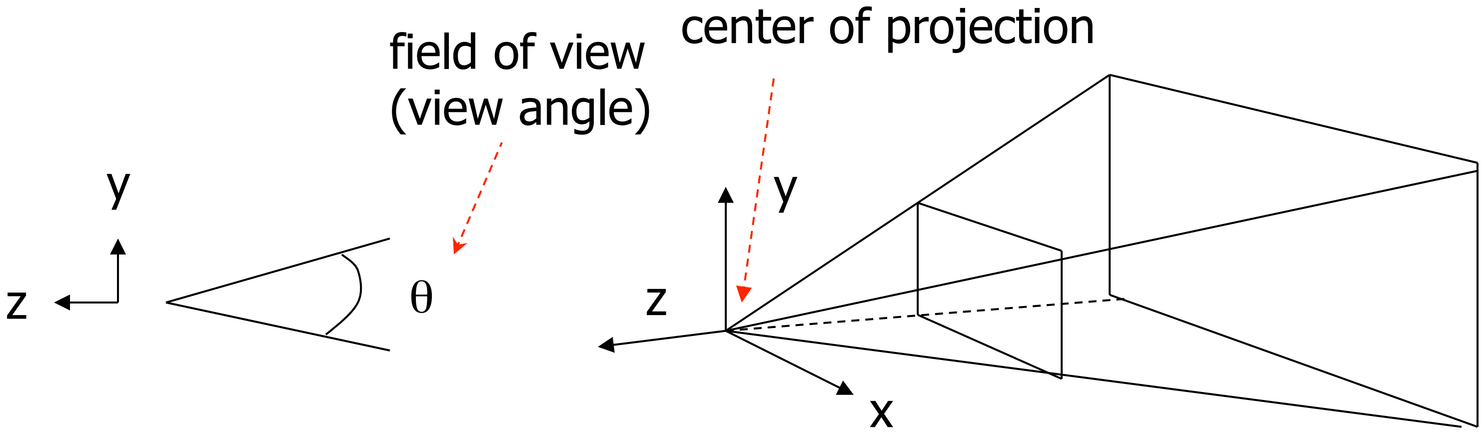
- Different view volume => different look
- **Foreshortening?** Near objects bigger

View Volume Parameters

- Need to set view volume parameters
  - **Projection type:** perspective, orthographic, etc.
  - 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 the objects are drawn

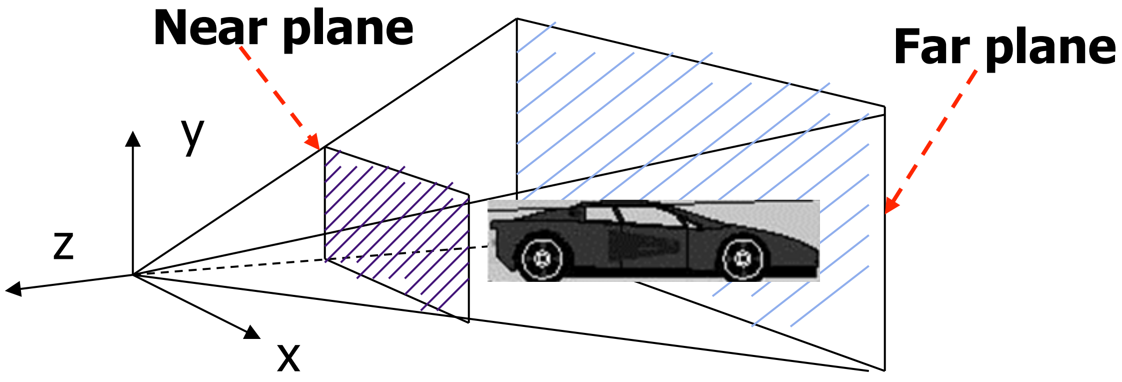


field of view
(view angle)

center of projection

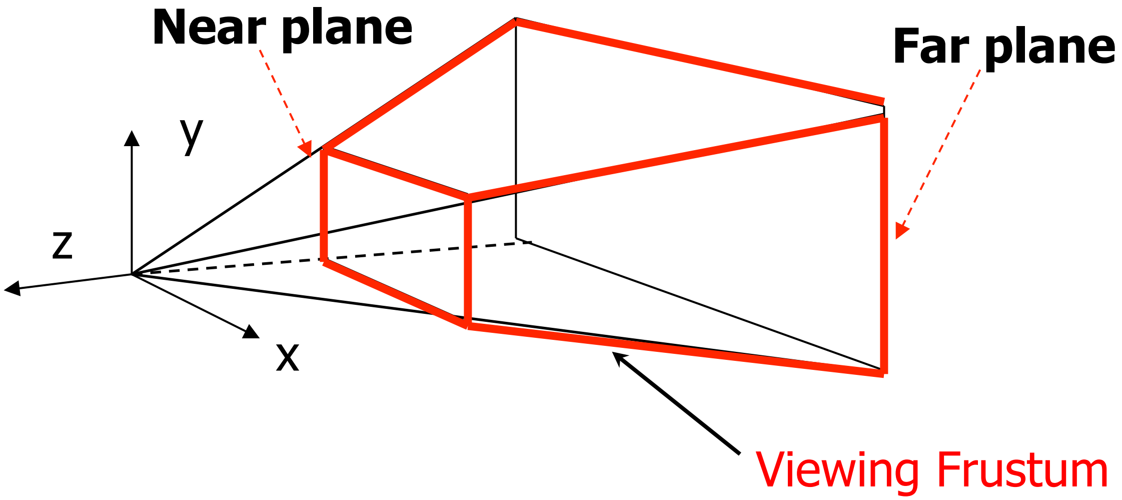
Near and Far Clipping Planes

- Only objects between near and far planes drawn



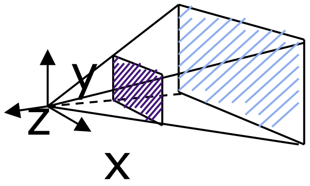
Viewing Frustrum

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

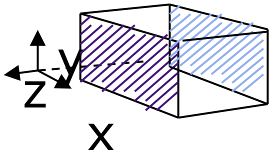


Setting up View Volume/Projection Type

- 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)

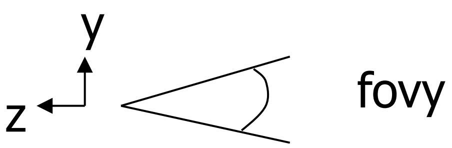


- Useful functions, so we implement similar in `mv.js`:
  - **Perspective**(fovy, aspect, near, far) **or**
  - **Frustum**(left, right, bottom, top, near, far)
  - **Ortho**(left, right, bottom, top, near, far)

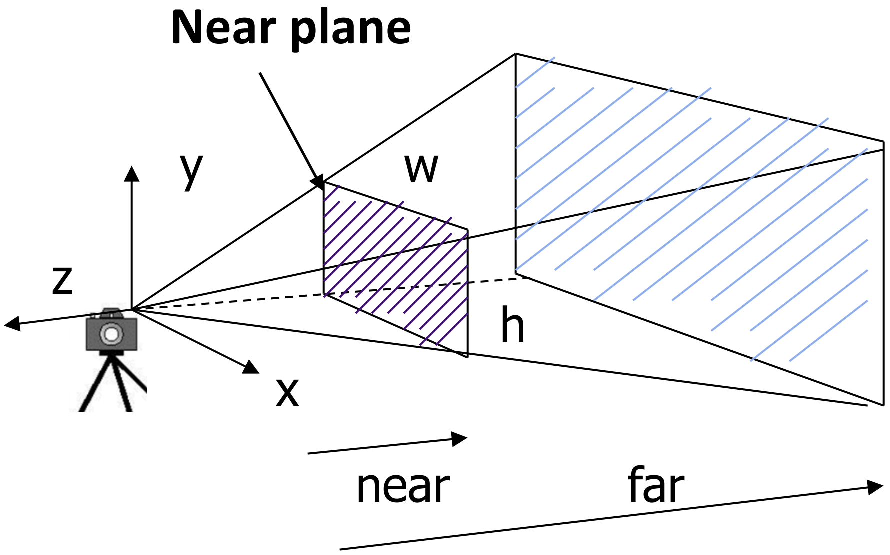
What are these arguments? Next!

Perspective(fovy, aspect, near, far)

- Aspect ratio used to calculate window width

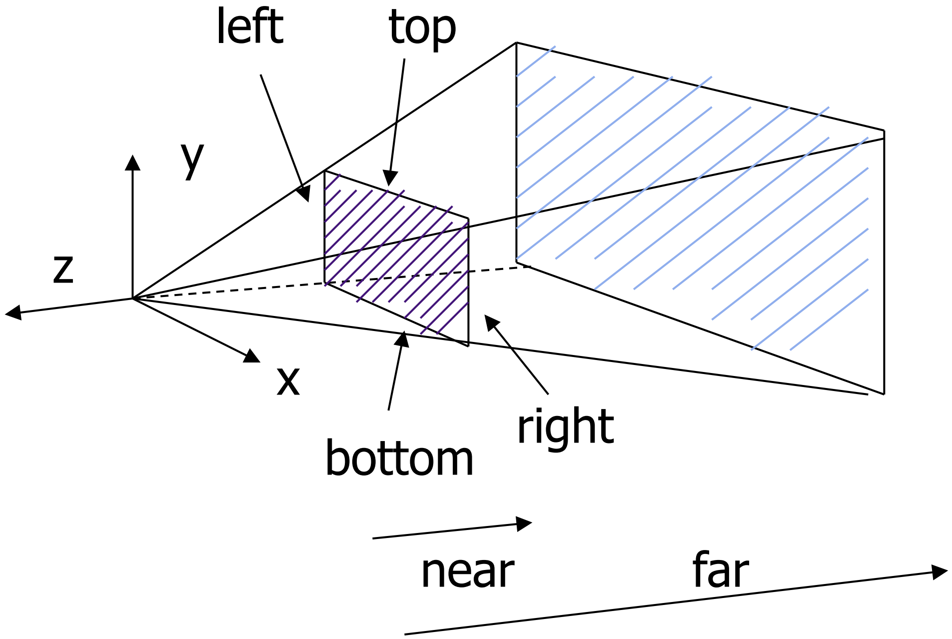


Aspect = w / h



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

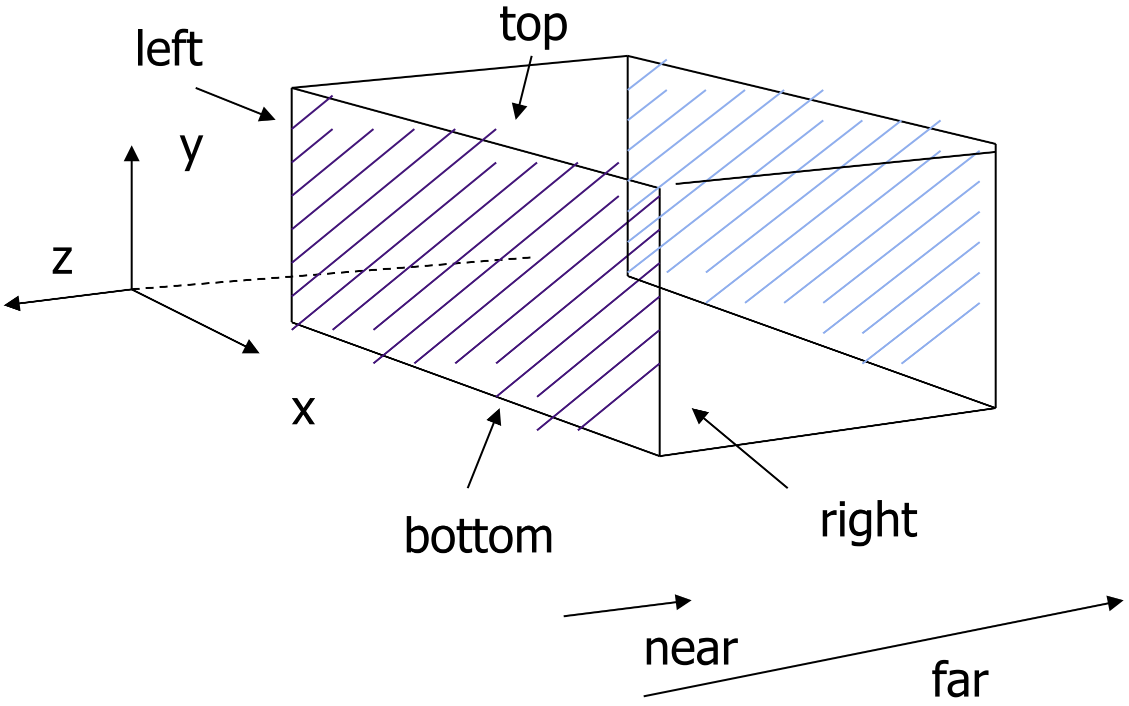
- Can use **Frustrum( )** in place of **Perspective**()
- Same view volume shape, different **arguments**



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

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

- For orthographic projection



`near` and `far` measured from camera

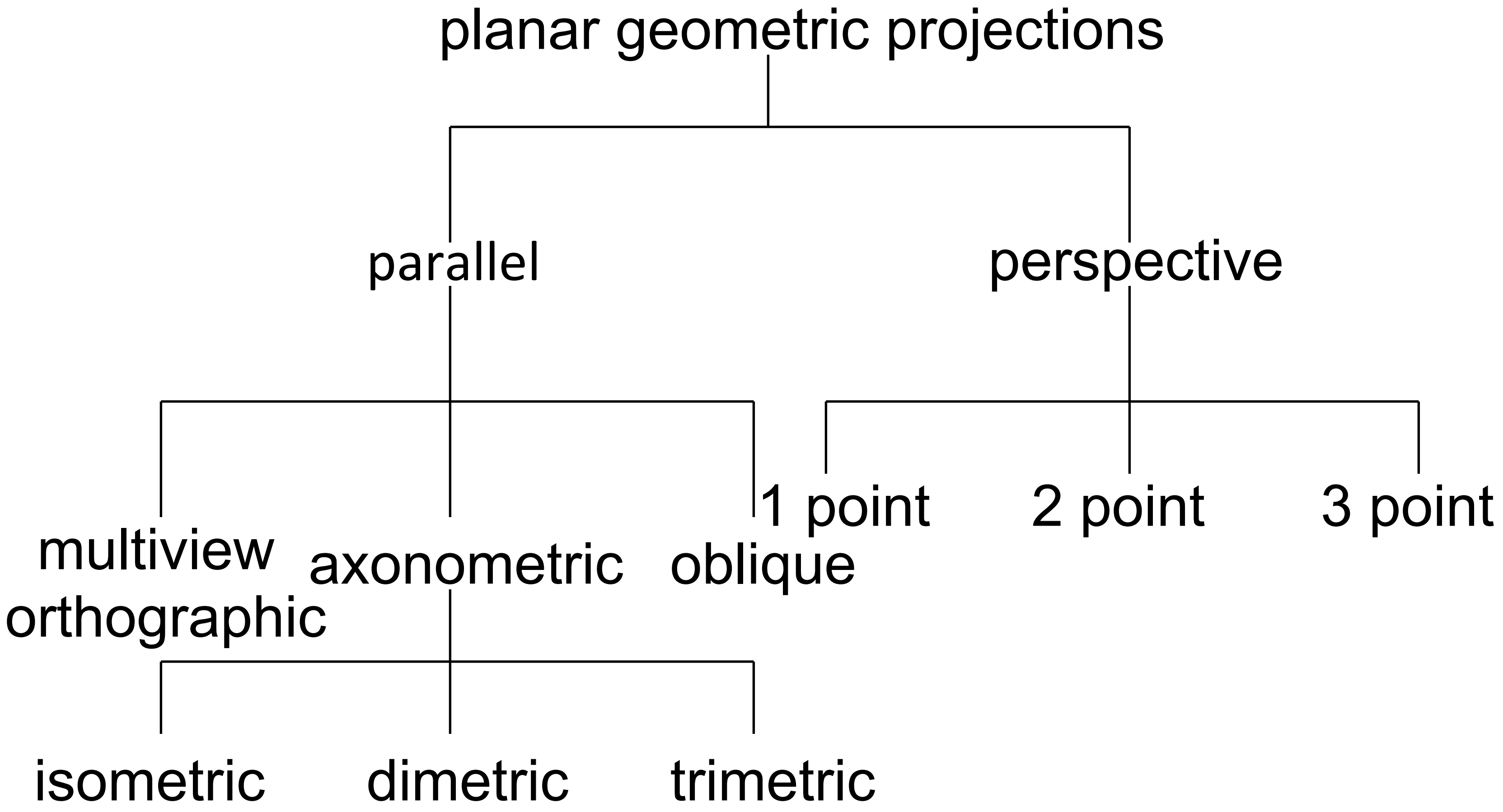
Example Usage:
Setting View Volume/Projection Type

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

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

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

Review

- Setting Up & Moving The Camera
- Look At Function
- View Volumes
- Near & Far Clipping Planes

Taxonomy of Planar Geometric Projections



planar geometric projections

parallel

perspective

multiview orthographic

axonometric

oblique

1 point

2 point

3 point

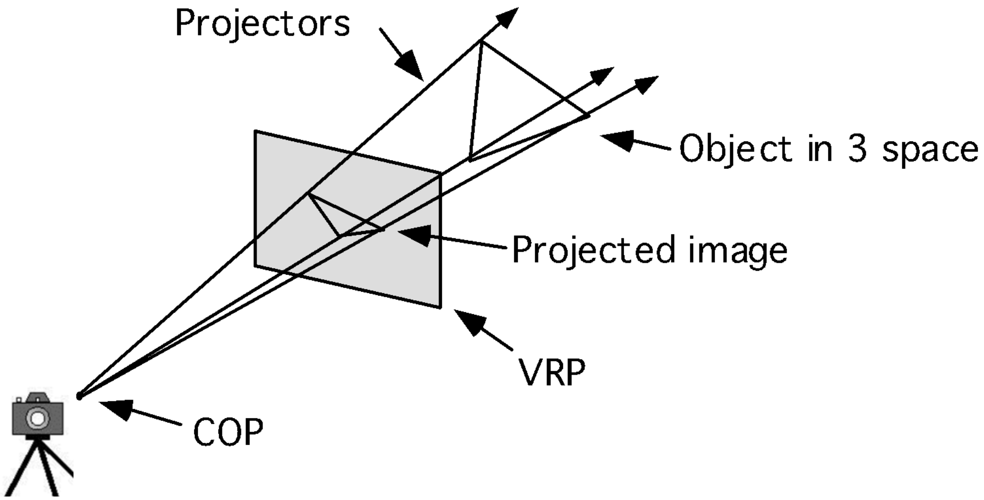
isometric

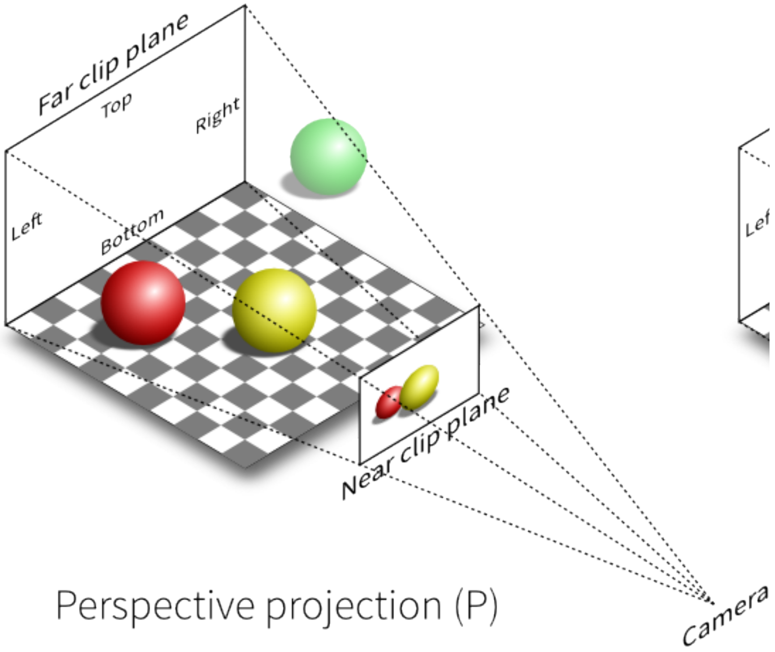
dimetric

trimetric

Perspective Projection

- After setting view volume, then projection transform
- Projection?
  - **Classic:** Converts 3D object to corresponding 2D on screen
  - How? Draw line from object to projection center
  - Calculate where each cuts projection plane

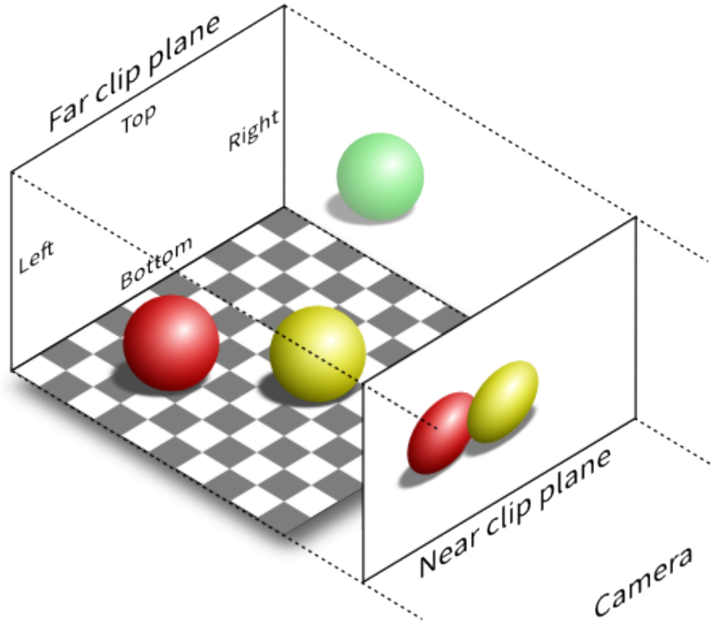




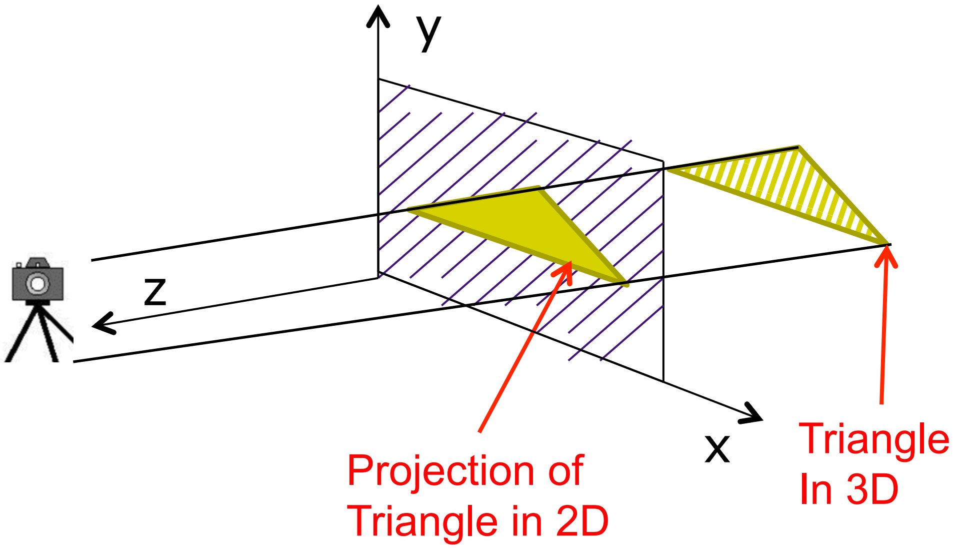
Perspective projection (P)

Orthographic Projection

- How? Draw parallel lines from each object vertex
- The projection center is at infinite
- In short, use (x,y) coordinates, just drop z coordinates



Orthographic projection (O)



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

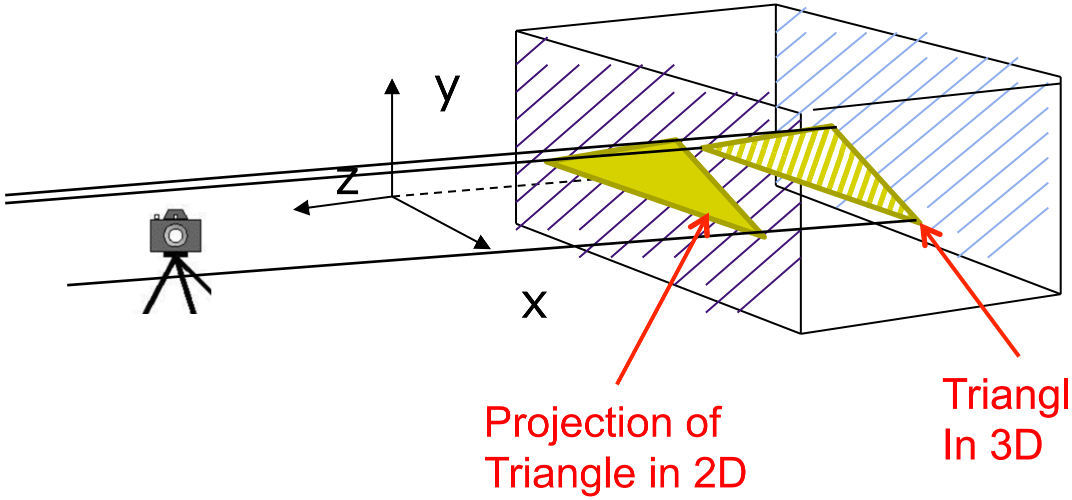
Default View Volume/Projection?

- What if you user does not set up projection?
- Default on most systems is orthogonal (Ortho( ));
- To project points within default view volume

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

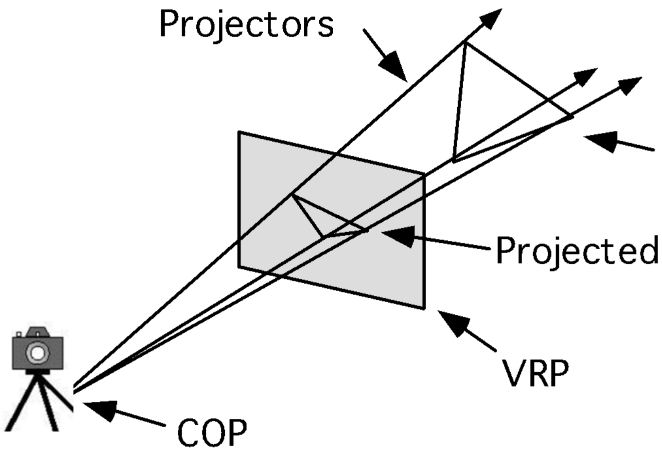
Vertices **before** Projection

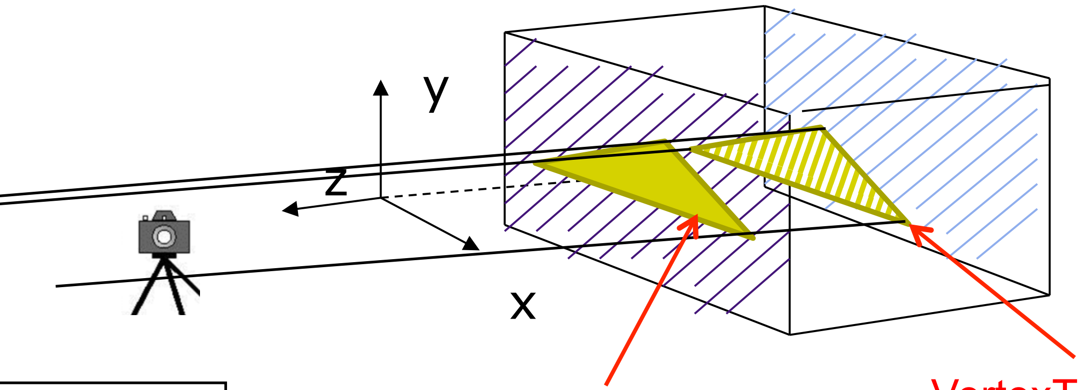
Vertices **after** Projection



The Problem with Classic Projection

- Keeps (x,y) coordintates for drawing, drops z
- We may need z. Why?





Projection of Triangle in 2D

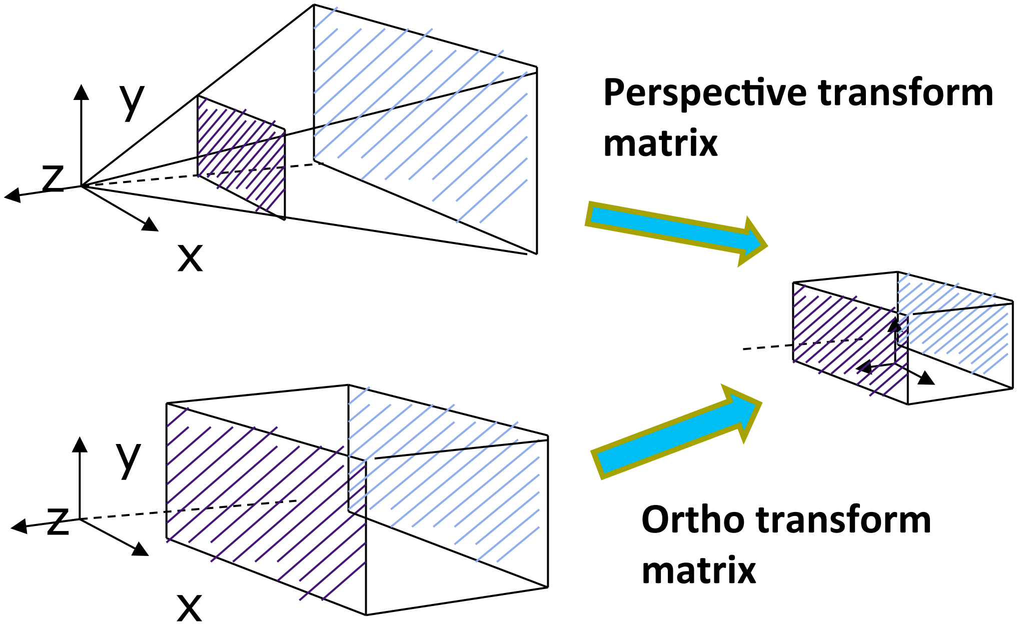
VertexTriangle In 3D

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

Classic Projection Loses z value

Normalization: Keeps z Value

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



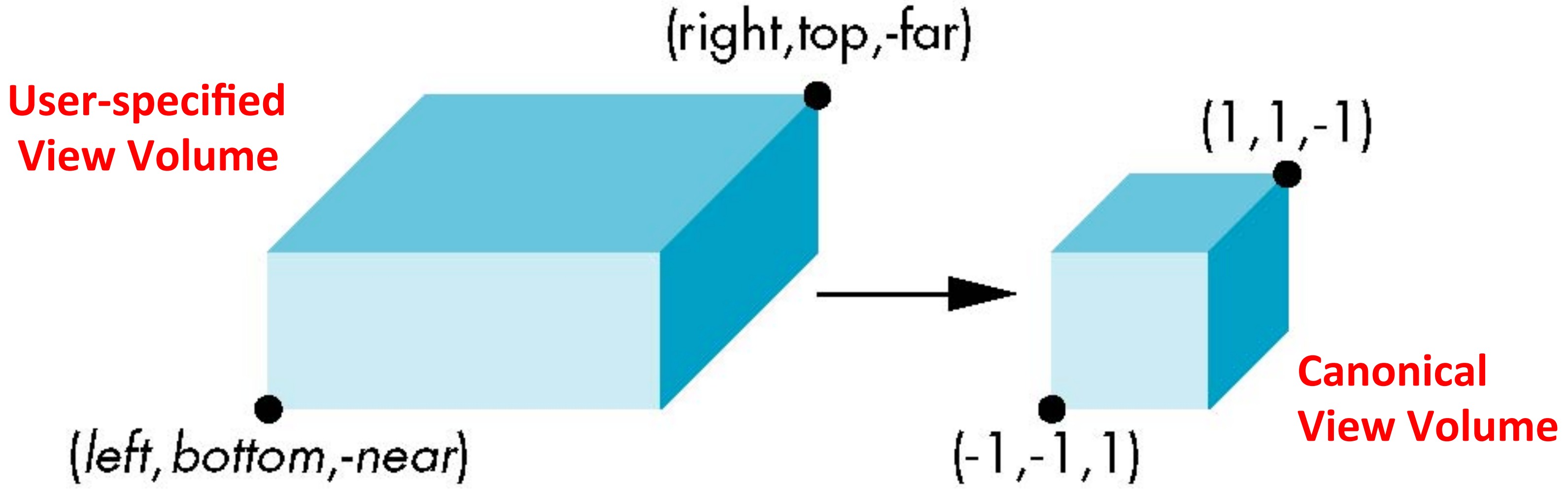
Perspective transform matrix

Ortho transform matrix

Default view volume
Clipping against it

Parallel Projection

- **normalization** ⇒ find 4x4 matrix to transform **user-specified view volume** to **canonical view volume (cube)**



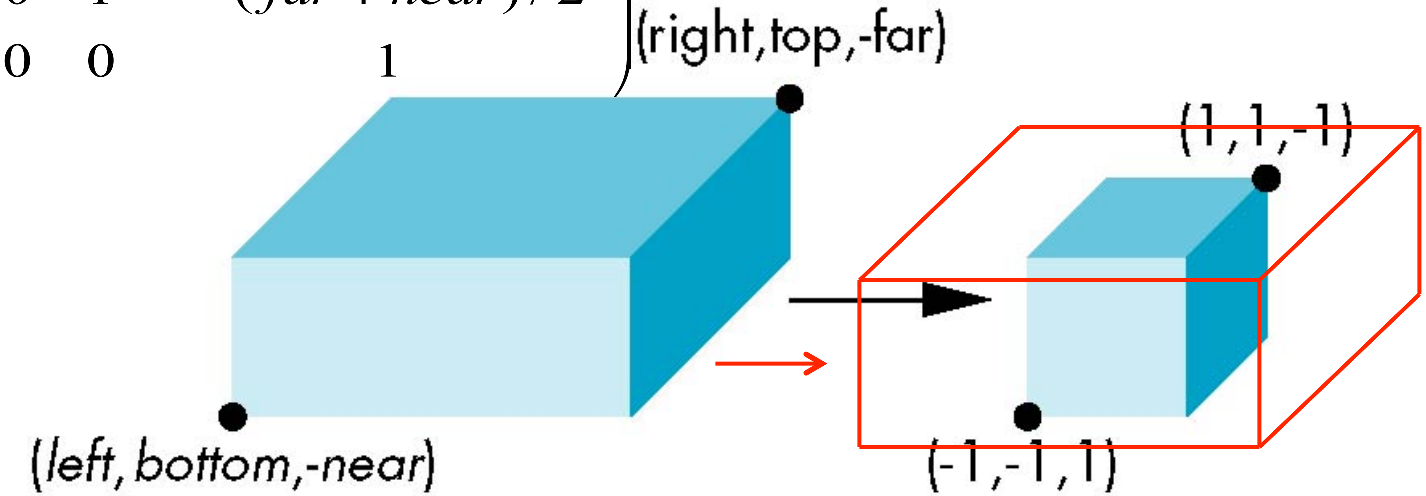
For Exampl: glOrtho(left, right, bottom, top, near, far)

Parallel Projection: Ortho

- Parallel projection: 2 parts
  1. **Translation:** centers view volume at origin
  - Thus translation factors:

    -(right + left)/2,   -(top + bottom)/2,   -(far+near)/2

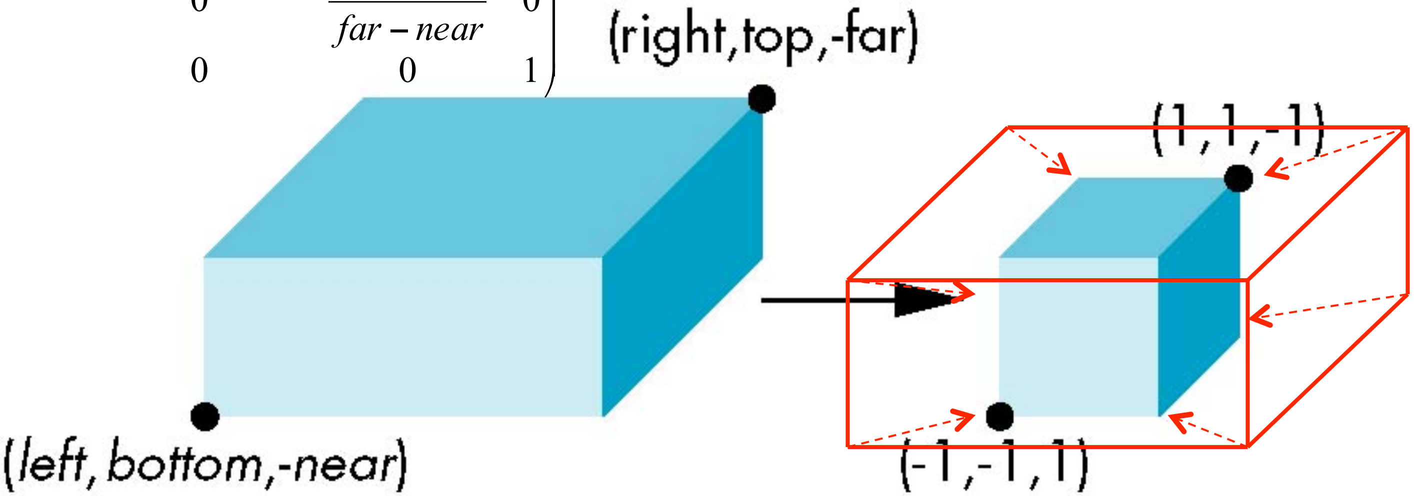
$$\begin{pmatrix} 1 & 0 & 0 & -(right+left)/2 \\ 0 & 1 & 0 & -(top+bottom)/2 \\ 0 & 0 & 1 & -(far+near)/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Parallel Projection: Ortho

2. **Scaling:** reduces user-selected cuboid to canonical cube (dimension 2, centered at origin)
   - Scaling factors: 2/(right - left), 2/(top - bottom), 2/(far - near)

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



Parallel Projection: Ortho

Concatenating **Translation** x **Scaling**, we get Ortho Projection matrix

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

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

Final Ortho Projection

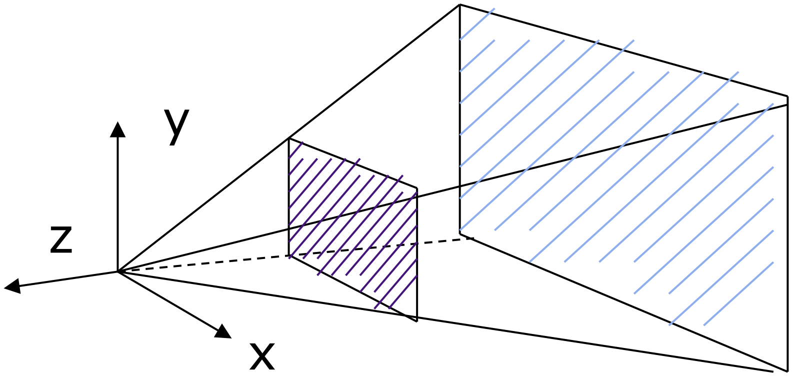
- Set $z = 0$
- Equivalent to the homogeneous coordinate transformation

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

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

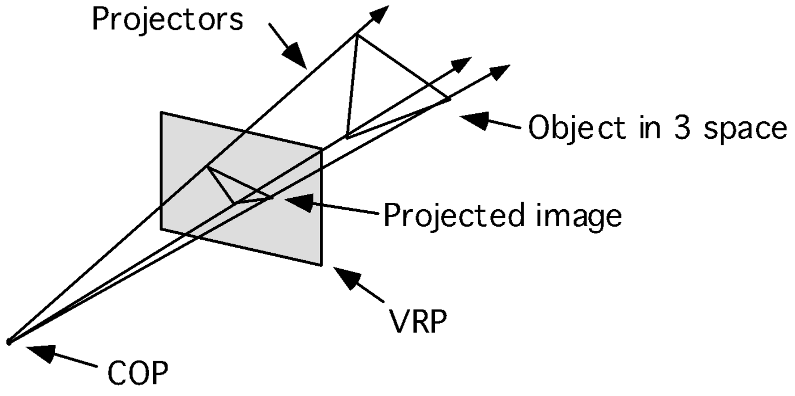
Perspective Projection

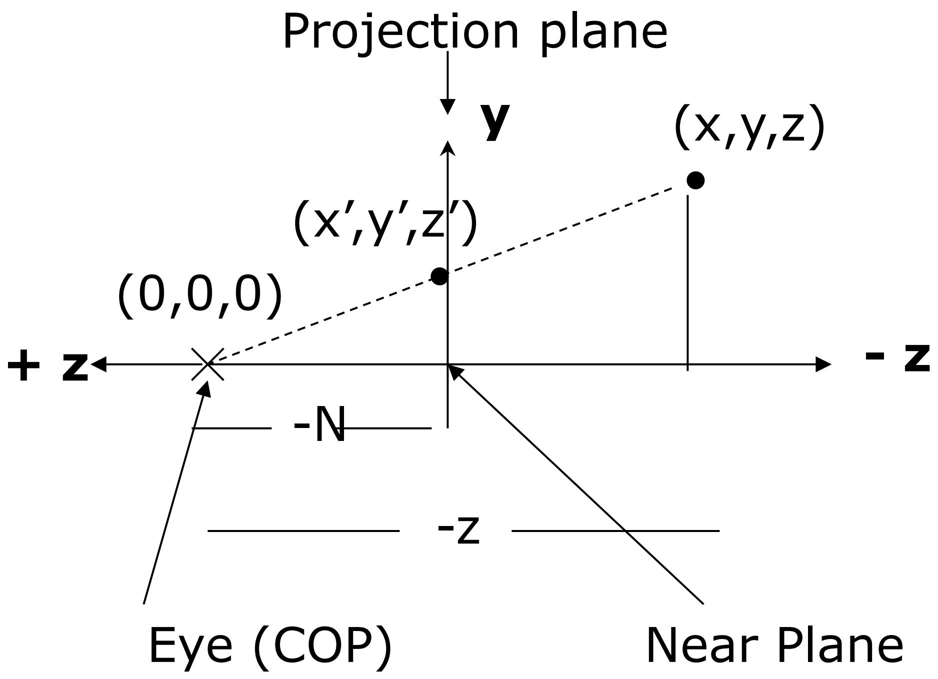
- Projection – map the object from 3D space to 2D screen



Perspective()
Frustrum( )

Perspective Projection





Based on similar triangles:

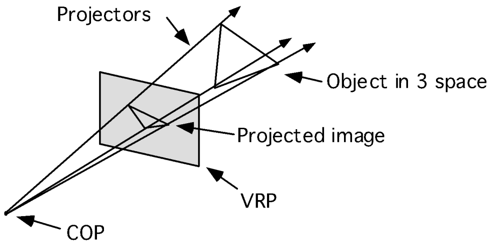
$$\frac{y'}{y} = \frac{N}{-z}$$

$$\Rightarrow \quad y' = y \ x \frac{N}{-z}$$

Perspective Projection

- So (x*,y*) projection of point, (x,y,z) unto near plane N is given as:

$$(x^*, y^*) = \left(x\frac{N}{-z}, y\frac{N}{-z} \right)$$



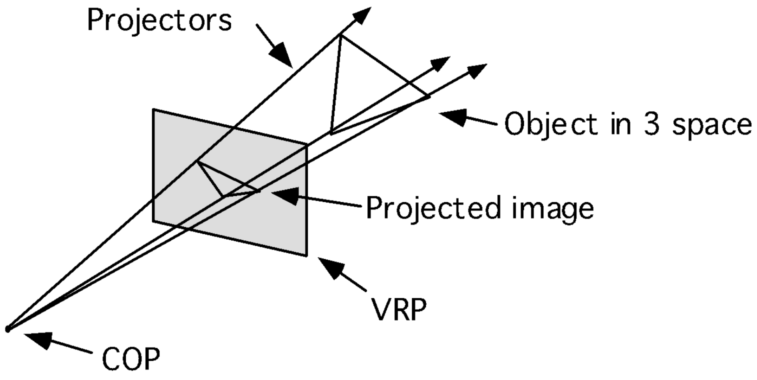
- Numerical example:

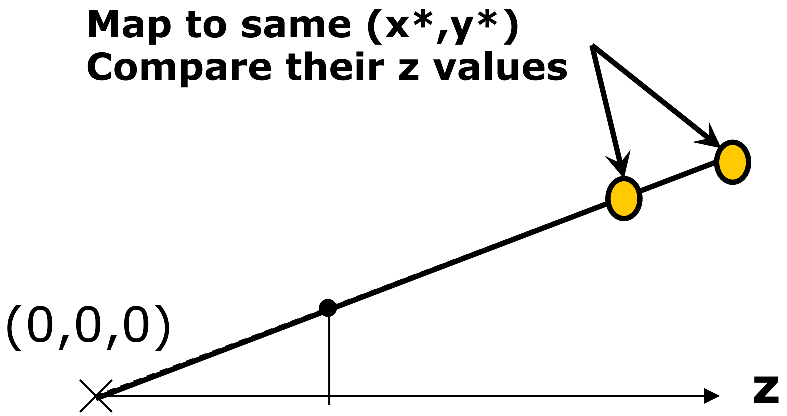
Q. Where on the viewplane does P = (1, 0.5, -1.5) lie for a near plane at N = 1?

$$(x^*, y^*) = \left(x\frac{N}{-z}, y\frac{N}{-z} \right) = \left(1\times\frac{1}{1.5}, 0.5\times\frac{1}{1.5} \right) = (0.666, 0.333)$$

Pseudodepth

- Classical perspective projection projects (x,y) coordinates to (x*, y*), drops z coordinates

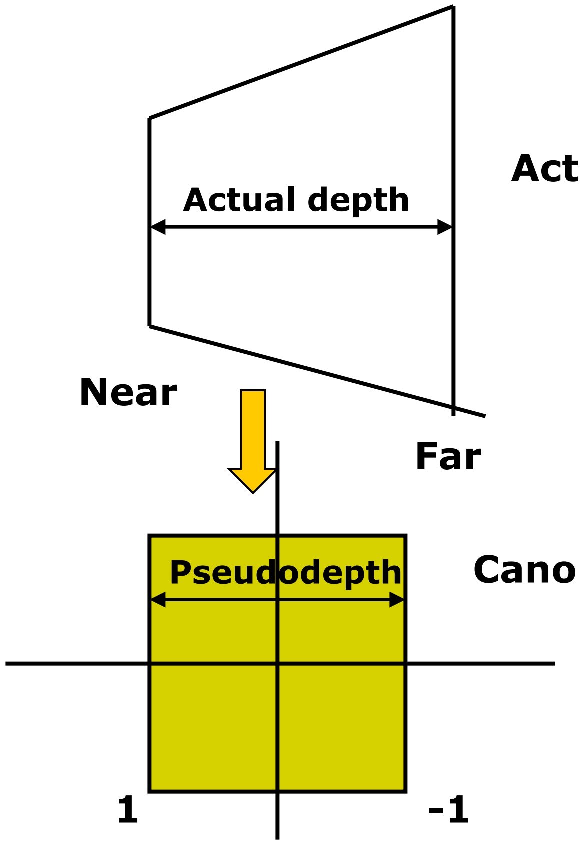




- But we need z to find closest object (depth testing)!!!

Perspective Transformation

- **Perspective transformation** maps actual z distance of perspective view volume to range [–1 to 1] (**Pseudodepth**) for canonical view volume



Actual view volume

Canonical view volume

We want perspective
Transformation and
NOT classical projection!!

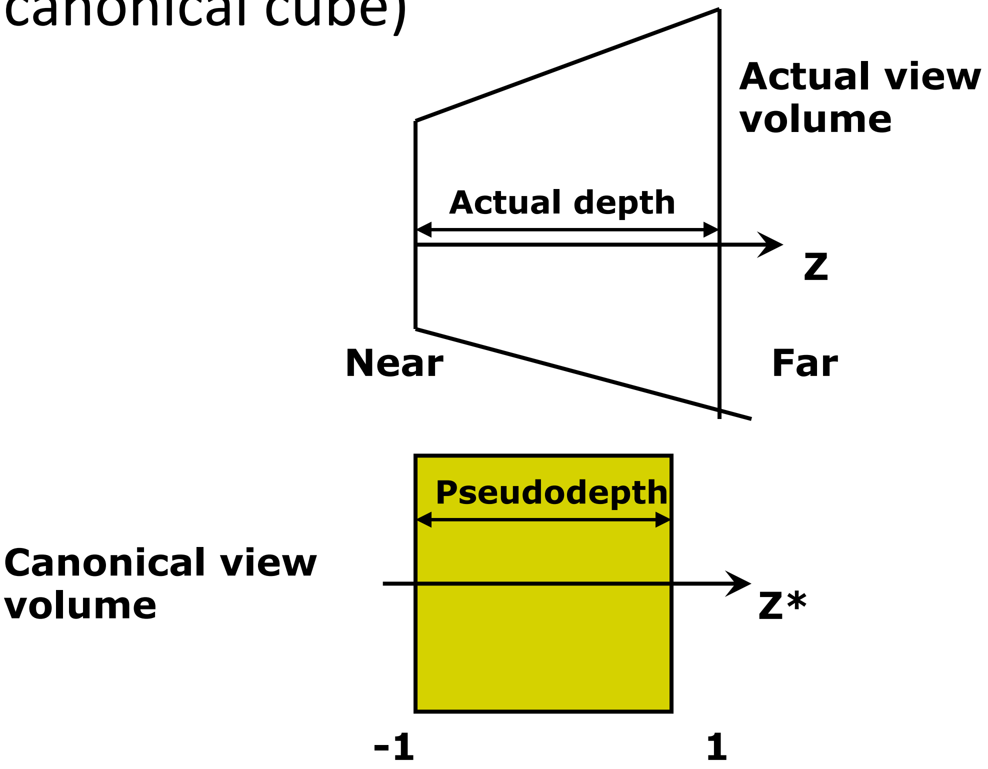
Set scaling z
Pseudodepth = az + b
Next solve for a and b

Perspective Transformation using Pseudodepth

$$\left(x^*, y^*, z^*\right) = \left(x\frac{N}{-z}, y\frac{N}{-z}, \frac{az \ + b}{-z} \right)$$

- Choose ***a, b*** so as z varies from **Near** to **Far**, pseudodepth z* varies from **–1** to **1** (canonical cube)

- Boundary conditions
  - z* = -1 when z = -N
  - z* = 1 when z = -F



Transformation of z: Solve for a and b

- Solving:

$$z^* = \frac{az + b}{-z}$$

- Use boundary conditions
  - z* = -1 when z = -N.........(1)
  - z* = 1 when z = -F...........(2)
- Set up simultaneous equations

$$-1 = \frac{-aN + b}{N} \Rightarrow -N = -aN + b........(1)$$

$$1 = \frac{-aF + b}{F} \Rightarrow F = -aF + b........(2)$$

Transformation of z: Solve for a and b

$$-N = -aN + b \ldots\ldots\ldots(1)$$

$$F = -aF + b \ldots\ldots\ldots(2)$$

- Multiply both sides of (1) by -1

$$N = aN - b \ldots\ldots\ldots(3)$$

- Add eqns (2) and (3)

$$F + N = aN - aF$$

$$\Rightarrow a = \frac{F+N}{N-F} = \frac{-(F+N)}{F-N} \ldots\ldots\ldots(4)$$

- Now put (4) back into (3)

Transformation of z: Solve for a and b

- Put solution for *a* back into eqn (3)

$$N = aN - b........(3)$$

$$\Rightarrow N = \frac{-N(F+N)}{F-N} - b$$

$$\Rightarrow b = -N - \frac{-N(F+N)}{F-N}$$

$$\Rightarrow b = \frac{-N(F-N) - N(F+N)}{F-N} = \frac{-NF - N^2 - NF + N^2}{F-N} = \frac{-2NF}{F-N}$$

- So

$$a = \frac{-(F+N)}{F-N} \qquad b = \frac{-2FN}{F-N}$$

What does this mean?

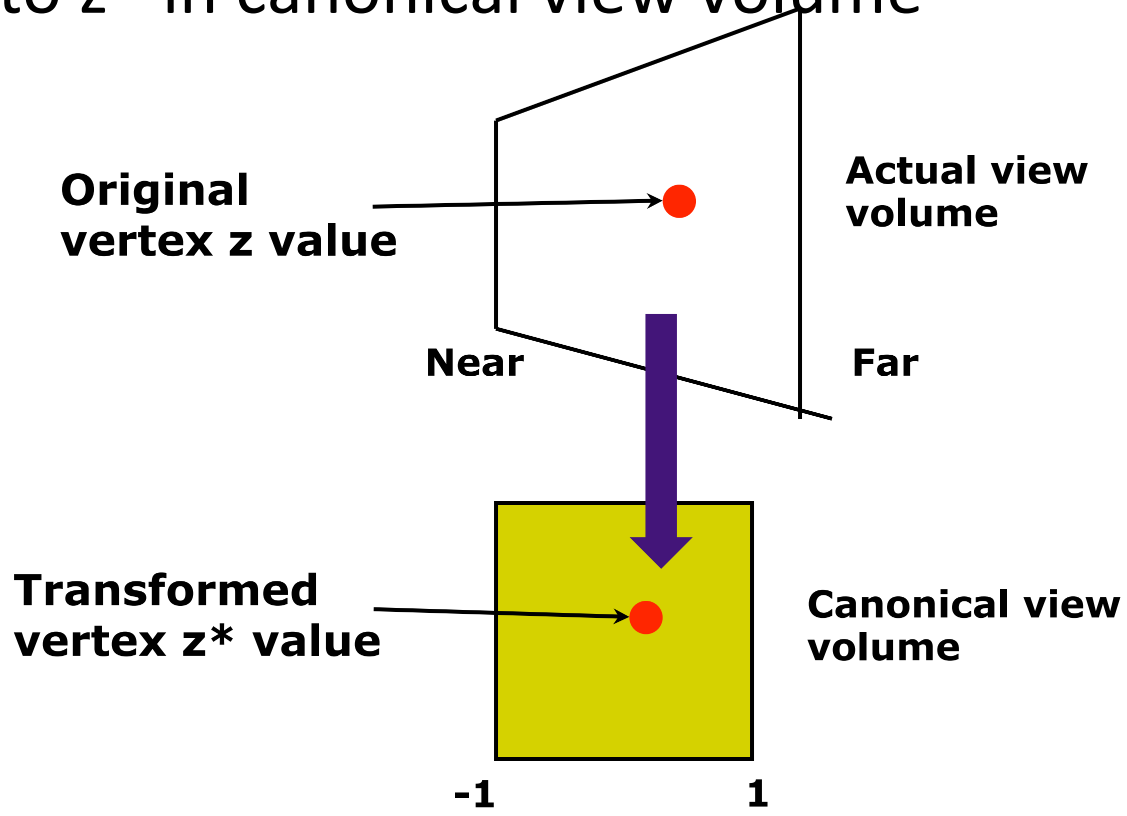
- Original point z in original view volume, transformed into z* in canonical view volume

$$z^* = \frac{az + b}{-z}$$

- where

$$a = \frac{-(F+N)}{F-N}$$

$$b = \frac{-2FN}{F-N}$$



Original vertex z value

Actual view volume

Near

Far

Transformed vertex z* value

Canonical view volume

-1

1

Homogenous Coordinates

- Want to express projection transform as 4x4 matrix
- Previously, homogeneous coordinates of

  P = (Px,Py,Pz) => (Px,Py,Pz,1)
- Introduce arbitrary scaling factor, w, so that

  P = (wPx, wPy, wPz, w) (**Note:** w is non-zero)
- For example, the point P = (2,4,6) can be expressed as
  - (2,4,6,1)
  - or (4,8,12,2) where w=2
  - or (6,12,18,3) where w = 3, or….
- To convert from homogeneous back to ordinary coordinates, first divide all four terms by **w** and discard 4th term

Perspective Projection Matrix

- Recall Perspective Transform

$$\left(x^*, y^*, z^*\right) = \left(x\frac{N}{-z}, y\frac{N}{-z}, \frac{az \ + b}{-z} \right)$$

- We have:

$$x^* = x\frac{N}{-z} \qquad y^* = y\frac{N}{-z} \qquad z^* = \frac{az + b}{-z}$$

- In matrix form:

$$\begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} = \begin{pmatrix} wNx \\ wNy \\ w(az + b) \\ -wz \end{pmatrix} \Rightarrow \begin{pmatrix} x\frac{N}{-z} \\ y\frac{N}{-z} \\ \frac{az + b}{-z} \\ 1 \end{pmatrix}$$

Perspective Transform Matrix

Original vertex

Transformed Vertex

Transformed Vertex after dividing by 4th term

Perspective Projection Matrix

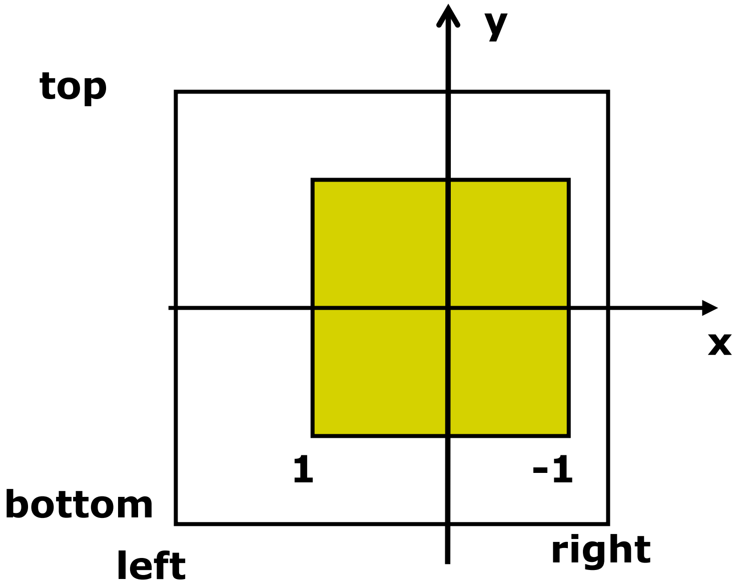
$$\begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} wP_x \\ wP_y \\ wP_z \\ w \end{pmatrix} = \begin{pmatrix} wNP_x \\ wNP_y \\ w(aP_z + b) \\ -wP_z \end{pmatrix} \Rightarrow \begin{pmatrix} x\dfrac{N}{-z} \\ y\dfrac{N}{-z} \\ \dfrac{az+b}{-z} \\ 1 \end{pmatrix}$$

$$a = \frac{-(F+N)}{F-N} \qquad b = \frac{-2FN}{F-N}$$

- In perspective transform matrix, already solved for ***a*** and ***b***:
- So, we have transform matrix to transform **z** values

Perspective Projection

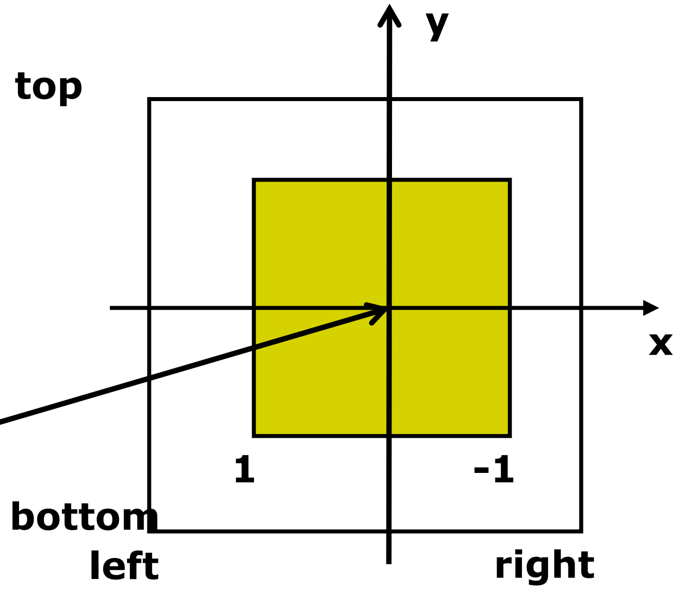
- Not done yet!! Can now transform z!
- Also need to transform the **x** = (left, right) and **y** = (bottom, top) ranges of viewing frustum to [-1, 1]
- Similar to Orthographic, we need to translate and scale previous matrix along x and y to get final projection transform matrix
- we translate by
  - –(right + left)/2 in x
  - -(top + bottom)/2 in y
- Scale by:
  - 2/(right – left) in x
  - 2/(top – bottom) in y



Perspective Projection

- Translate along x and y to line up center with origin of CVV
  - –(right + left)/2 in x
  - -(top + bottom)/2 in y
- Multiply by translation matrix:

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

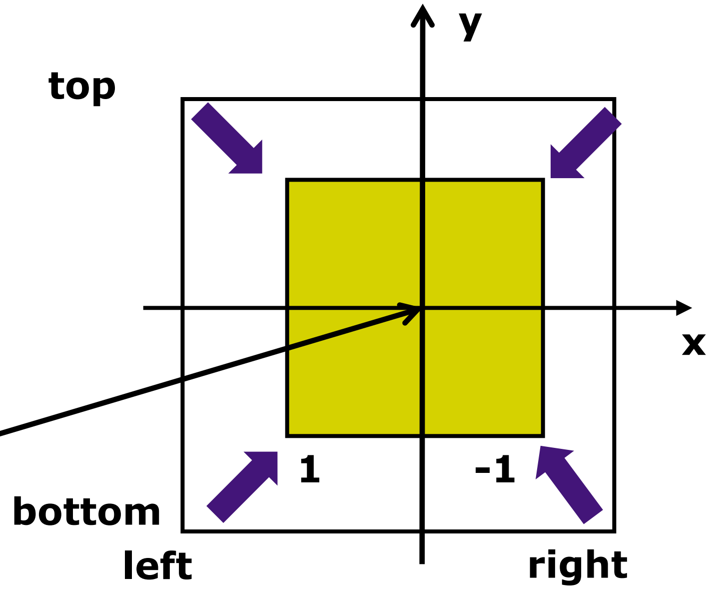


Line up centers
Along x and y

Perspective Projection

- To bring view volume size down to size of of CVV, scale by
  - 2/(right – left) in x
  - 2/(top – bottom) in y
- Multiply by scale matrix:

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



Scale size down along x and y

Perspective Projection Matrix

Scale **Translate** **Previous Perspective Transform Matrix**

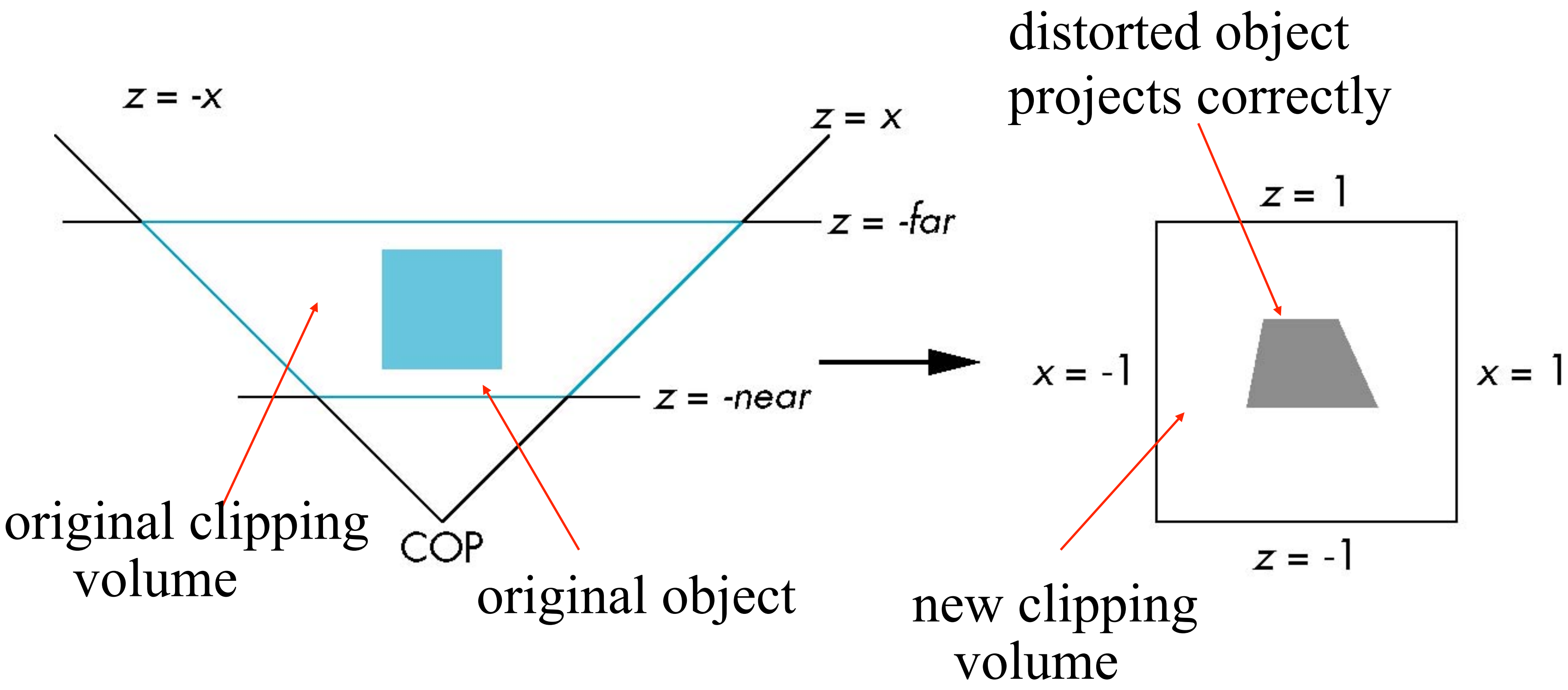
$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & -(right+left)/2 \\ 0 & 1 & 0 & -(top+bottom)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} N & 0 & 0 & 0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

$$\Rightarrow \begin{pmatrix} \frac{2N}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2N}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & \frac{-(F+N)}{F-N} & \frac{-2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Final Perspective Transform Matrix

glFrustum(left, right, bottom, top, N, F) N = near plane, F = far plane

Normalization Transformation



distorted object projects correctly

original clipping volume

original object

new clipping volume

Top View of before & after normalization

Implementation

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

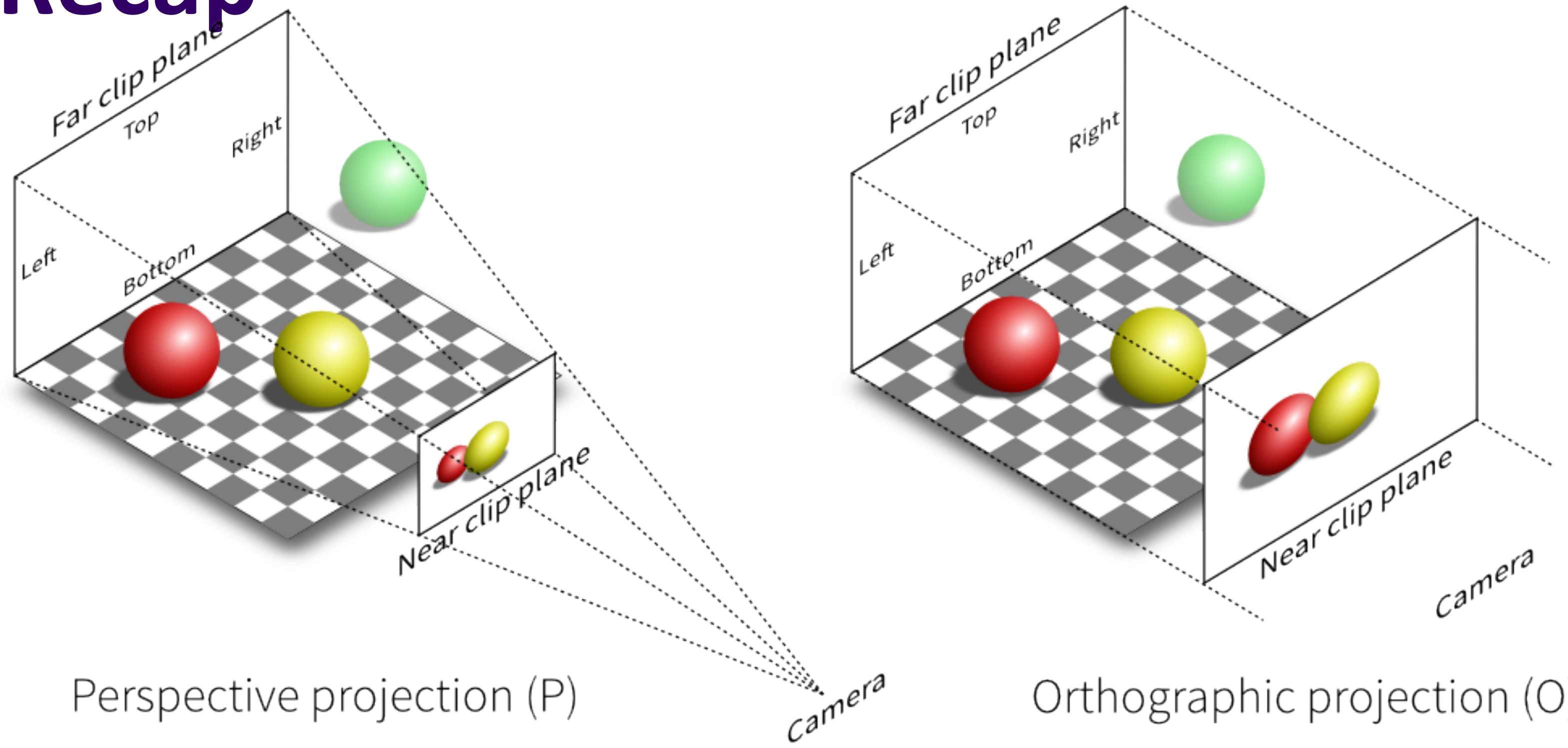
```
void display( ){
    .....
    model_view = LookAt(eye, at, up);
    projection = Ortho(left, right, bottom,top, near, far);

    // 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;
}
```

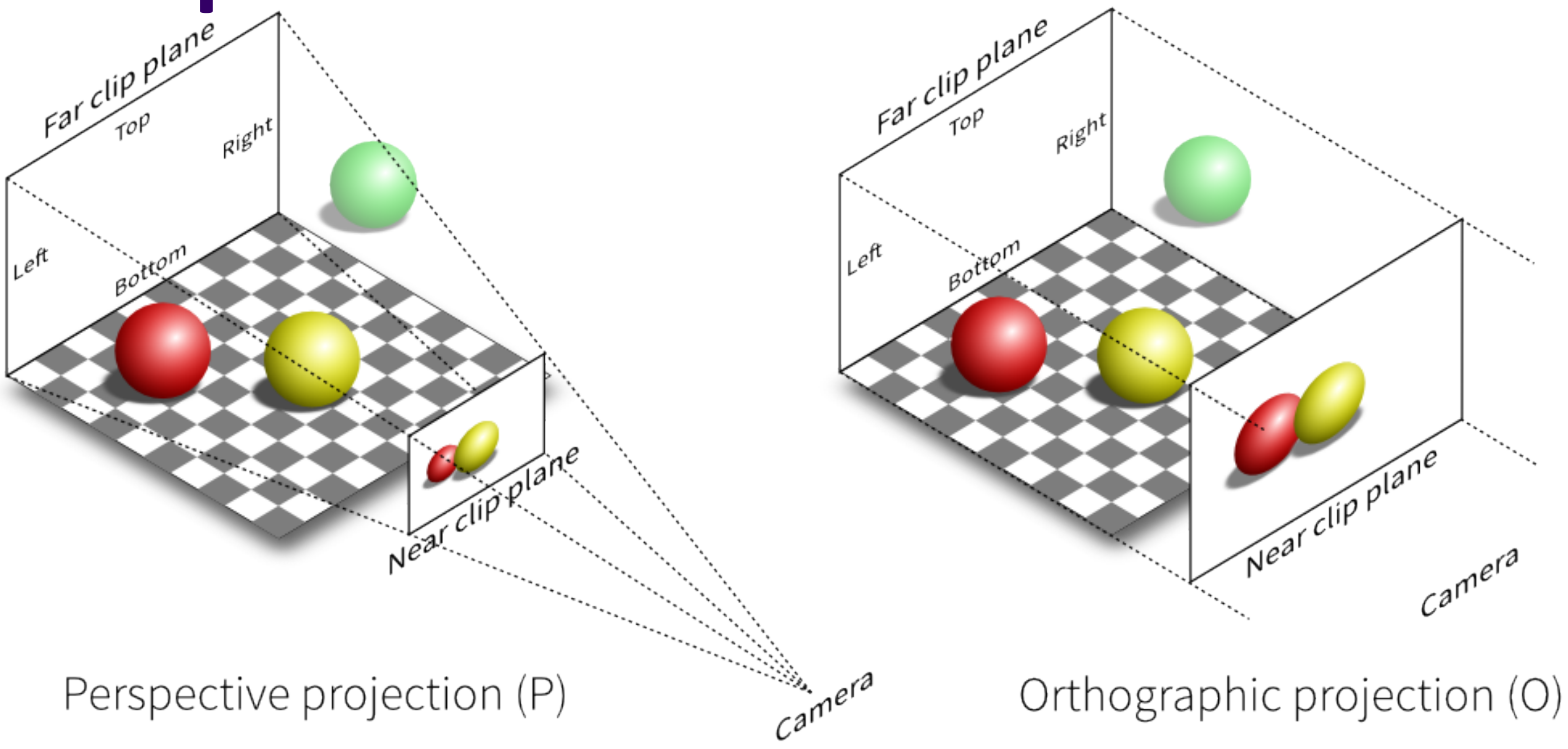
Recap



Perspective projection (P)

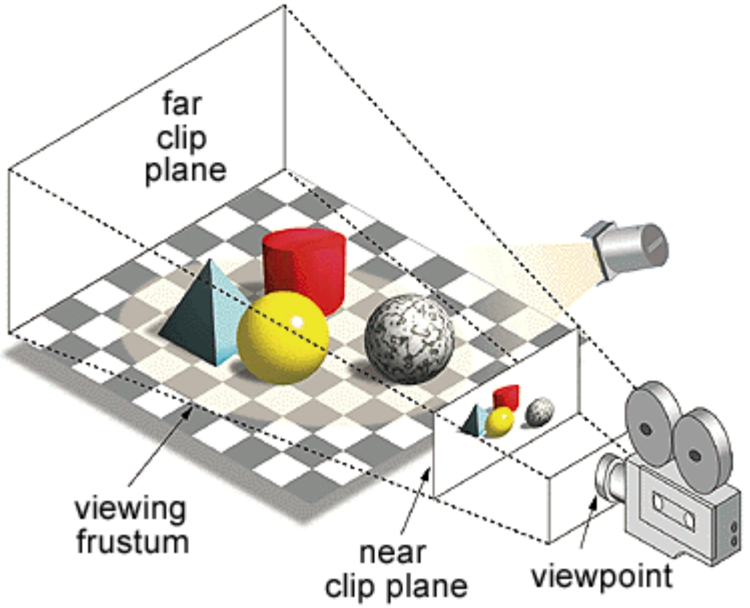
Orthographic projection (O)

Recap



Perspective projection (P)

Orthographic projection (O)



From Computer Desktop Encyclopedia
Reproduced with permission.
© 1998 Intergraph Computer Systems