



北京航空航天大学

人工智能学院(人工智能研究院)

Computer Graphics

Lecture 4: View Transformation

潘成伟 (Chengwei Pan)

Email: pancw@buaa.edu.cn

Office: Room B1021, New Main Building

北京航空航天大学, 人工智能学院

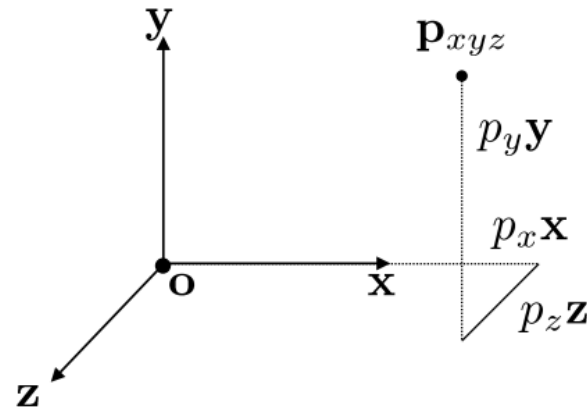
School of Artificial Intelligence, Beihang University

This Lecture

- **CG Coordinate Systems**
 - Viewing pipeline
- Camera Transformation
- Projection Transformation
 - Orthographic projection
 - Perspective projection
- Viewport Transformation

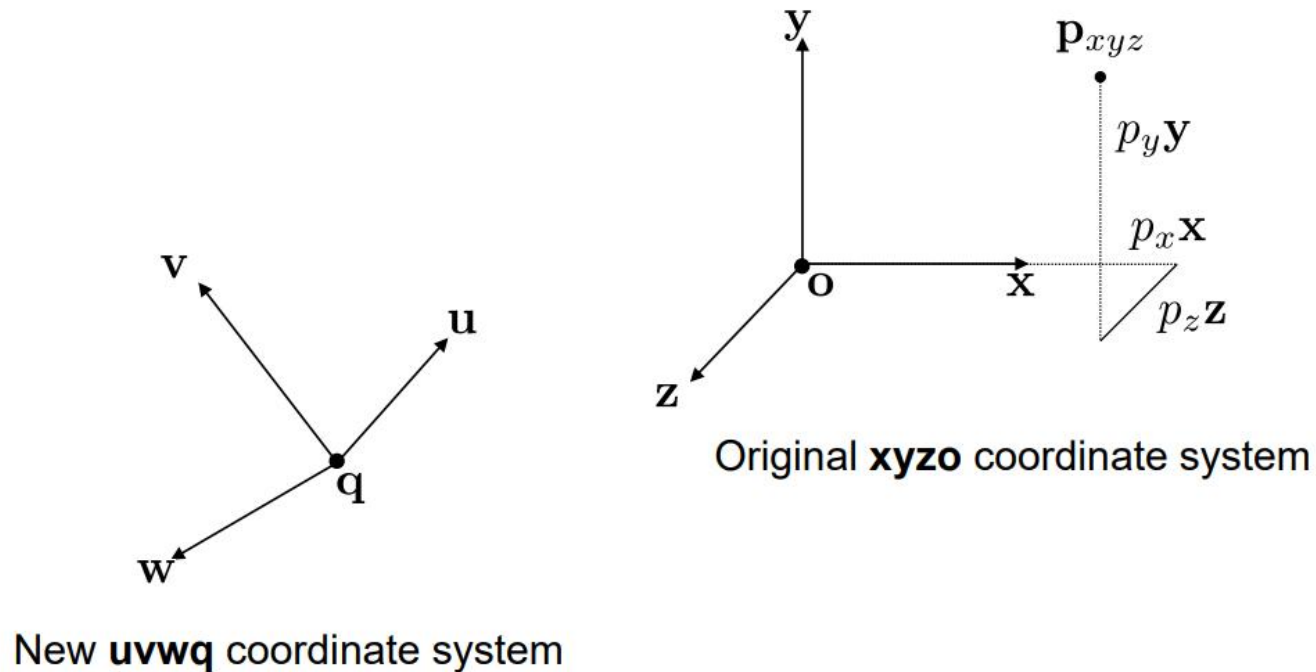
Coordinate System

- Given point \mathbf{p} in homogeneous coordinates: $\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$
- Coordinates describe the point's 3D position in a coordinate system with basis vectors \mathbf{x} , \mathbf{y} , \mathbf{z} and origin \mathbf{o} :
 - Axis are perpendicular
 - Axis are ordered



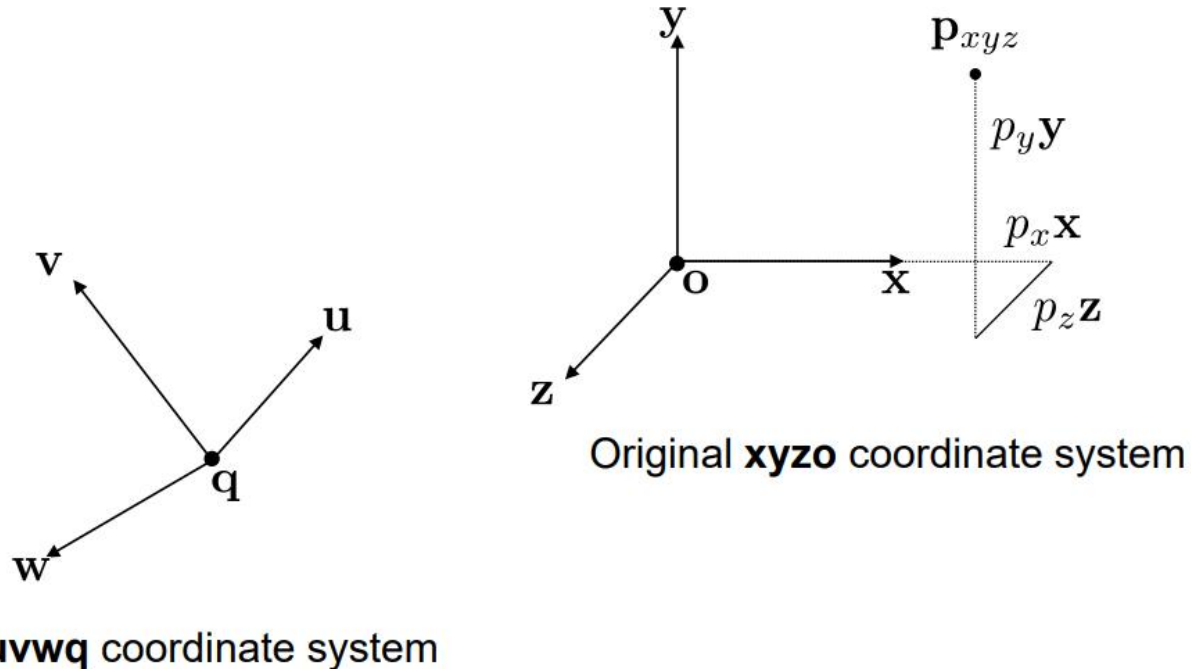
$$\mathbf{P}_{xyz} = p_x\mathbf{x} + p_y\mathbf{y} + p_z\mathbf{z} + \mathbf{o}$$

Coordinate Transformation



- Goal: Find Coordinates p_{xyz} of in new $uvwq$ coordinate system

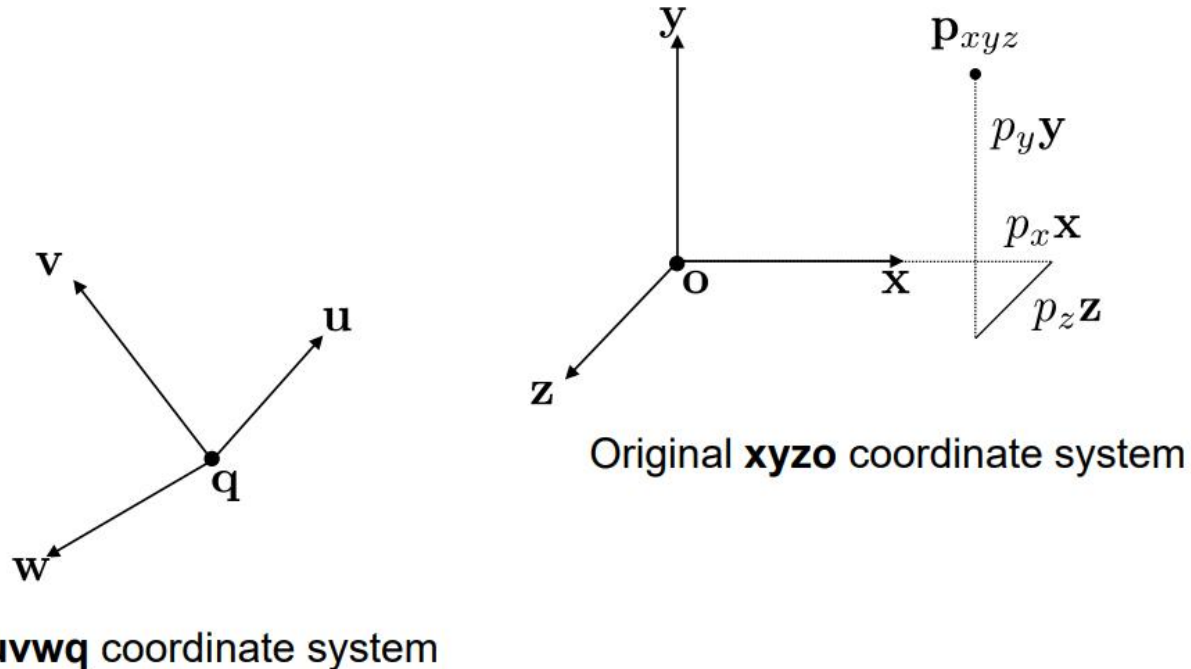
Coordinate Transformation



- Express coordinates of **xyzo** reference frame with respect to **uvwq** reference frame:

$$\mathbf{x} = \begin{bmatrix} x_u \\ x_v \\ x_w \\ 0 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_u \\ y_v \\ y_w \\ 0 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} z_u \\ z_v \\ z_w \\ 0 \end{bmatrix} \quad \mathbf{o} = \begin{bmatrix} o_u \\ o_v \\ o_w \\ 1 \end{bmatrix}$$

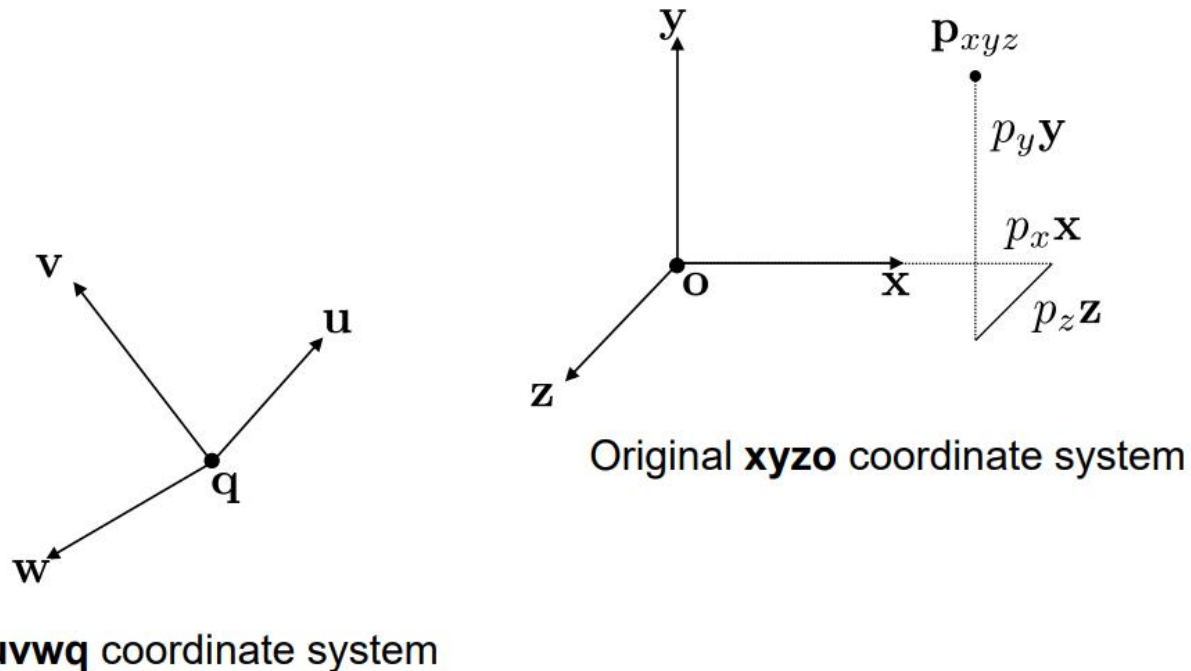
Coordinate Transformation



- Express point **p** in new **uvwq** reference frame:

$$\mathbf{p}_{uvw} = p_x \begin{bmatrix} x_u \\ x_v \\ x_w \\ 0 \end{bmatrix} + p_y \begin{bmatrix} y_u \\ y_v \\ y_w \\ 0 \end{bmatrix} + p_z \begin{bmatrix} z_u \\ z_v \\ z_w \\ 0 \end{bmatrix} + \begin{bmatrix} o_u \\ o_v \\ o_w \\ 1 \end{bmatrix}$$

Coordinate Transformation



- Express point **p** in new **uvwq** reference frame:

$$\mathbf{p}_{uvw} = \begin{bmatrix} x_u & y_u & z_u & o_u \\ x_v & y_v & z_v & o_v \\ x_w & y_w & z_w & o_w \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{o} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

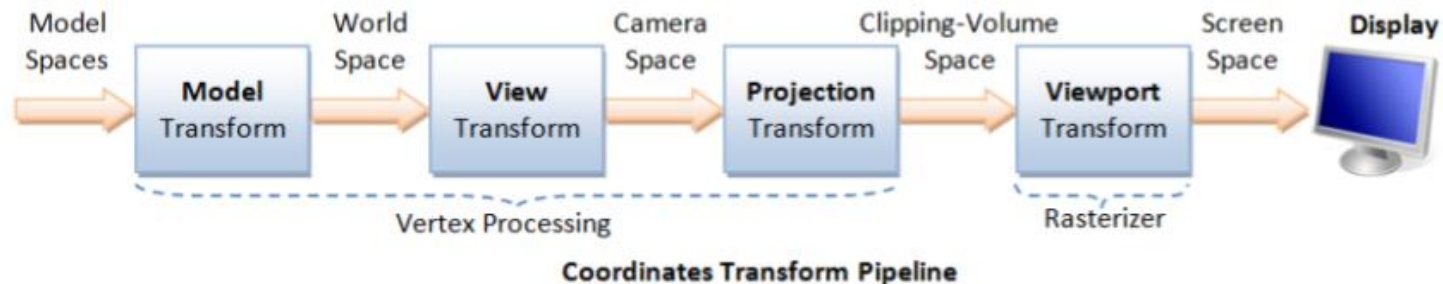
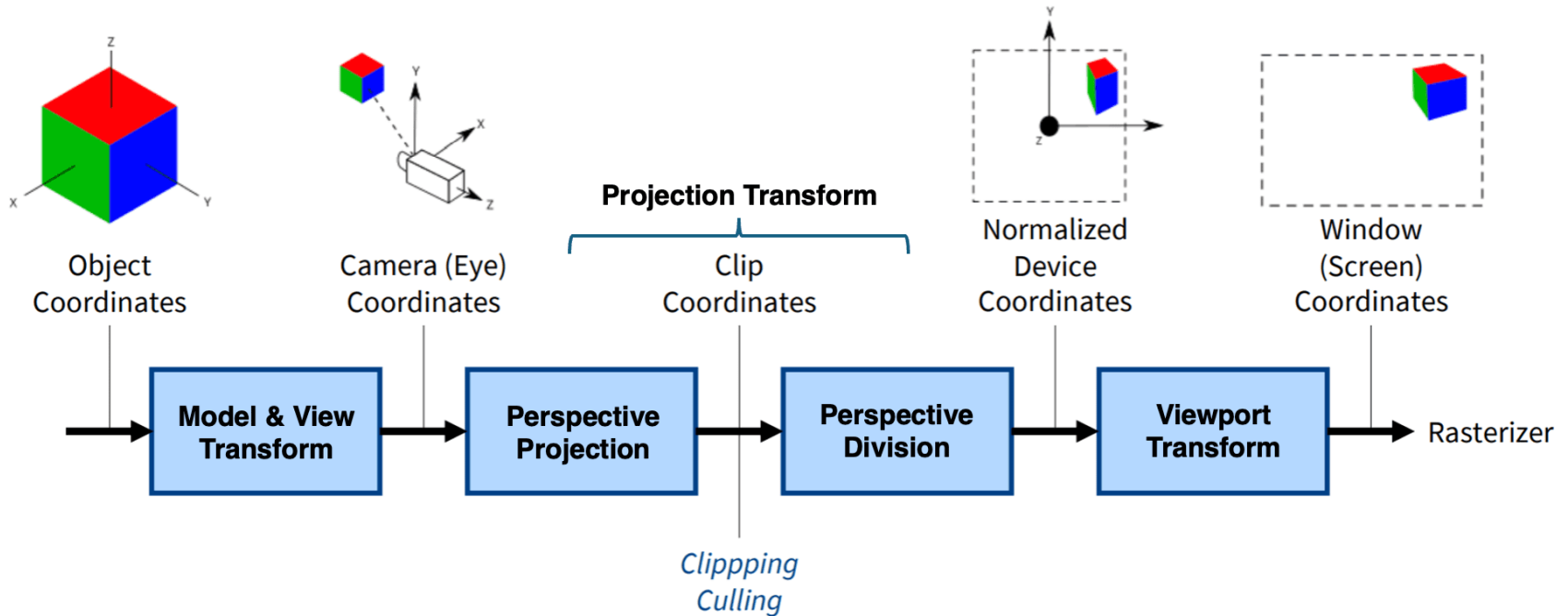
Coordinate Transformation

- Inverse transformation
 - Given point \mathbf{p}_{uvw} w.r.t reference frame \mathbf{uvwq}
 - Coordinates \mathbf{p}_{xyz} w.r.t reference frame \mathbf{xyzo} are calculated as:

$$\mathbf{p}_{xyz} = \begin{bmatrix} x_u & y_u & z_u & o_u \\ x_v & y_v & z_v & o_v \\ x_w & y_w & z_w & o_w \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} p_u \\ p_v \\ p_w \\ 1 \end{bmatrix}$$

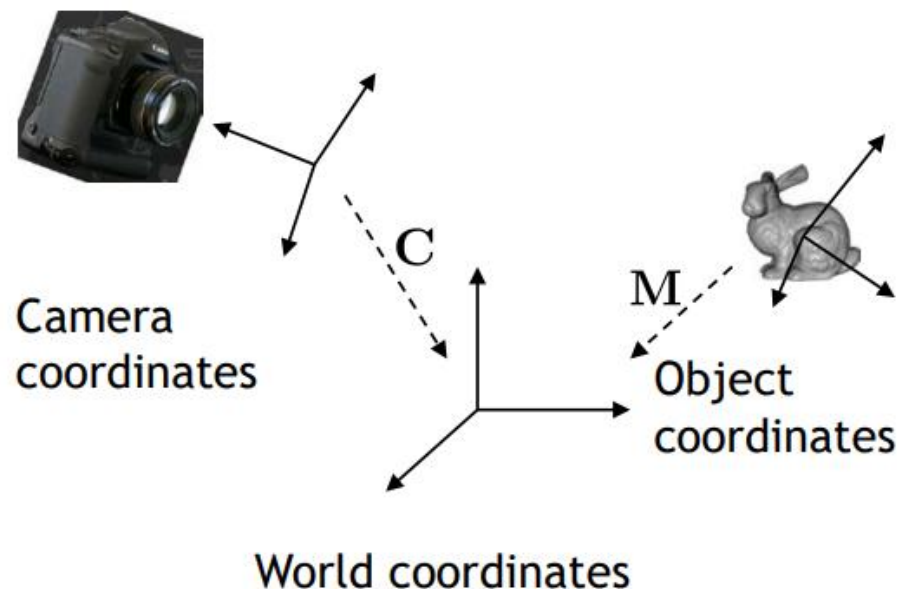
CG Coordinate Systems

- Transformation pipeline in CG



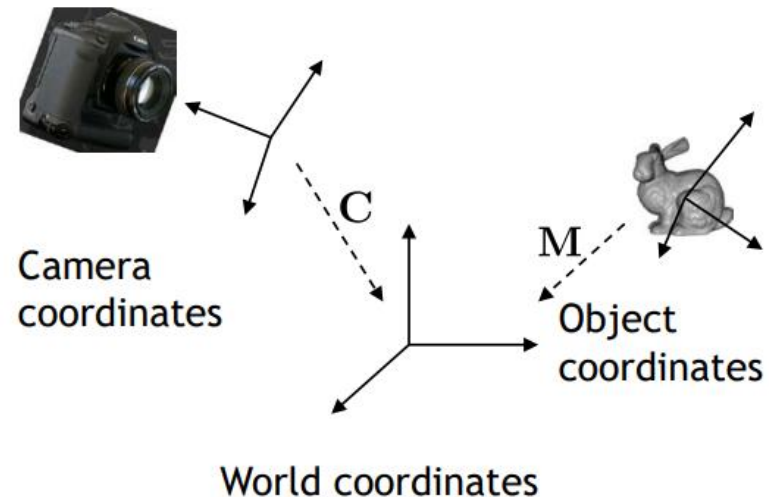
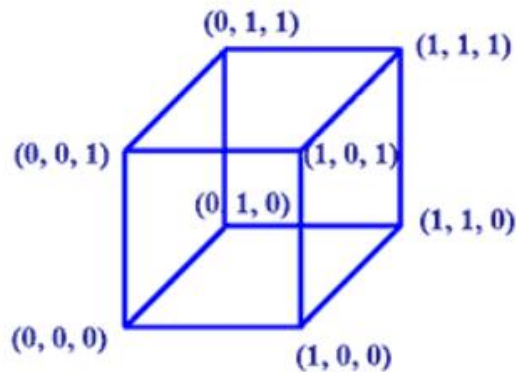
Typical Coordinate Systems

- In CG, we typically use at least three coordinate systems:
 - Object coordinate system (local coordinate)
 - World coordinate system
 - Camera coordinate system



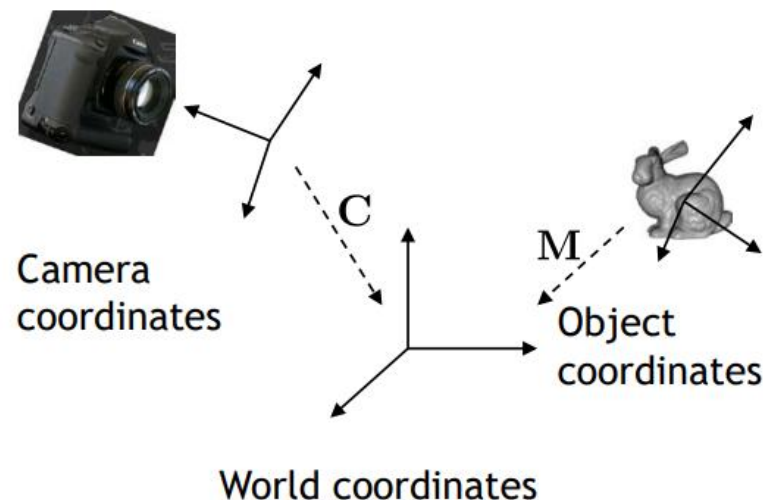
Object Coordinates

- Local coordinates in which points and other object geometry are given
- Often origin is in geometric center, on the base, or in a corner of the object
 - Depends on how an object is generated or used



World Coordinates

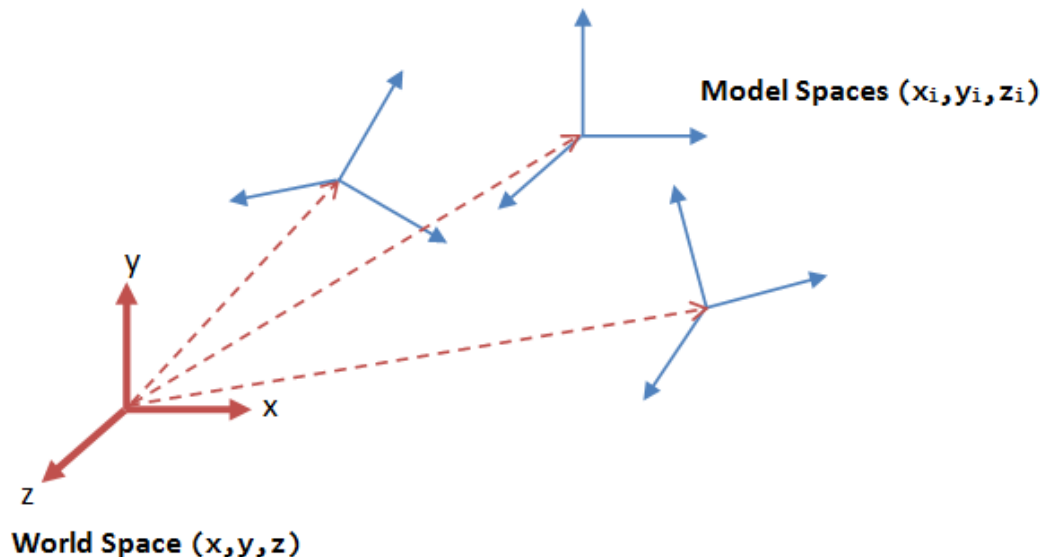
- Common reference frame for all objects in the scene
- No standard for coordinate orientation
 - If there is a ground plane, usually x/y is horizontal and z points up (height)
 - Otherwise, x/y is often screen plane, z points out of the screen



Object/Model Transformation

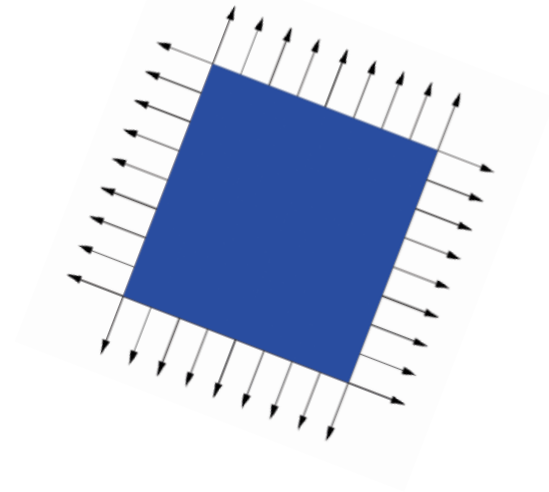
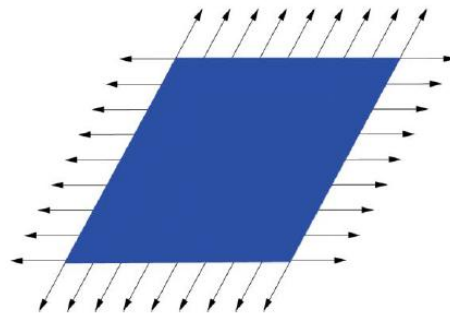
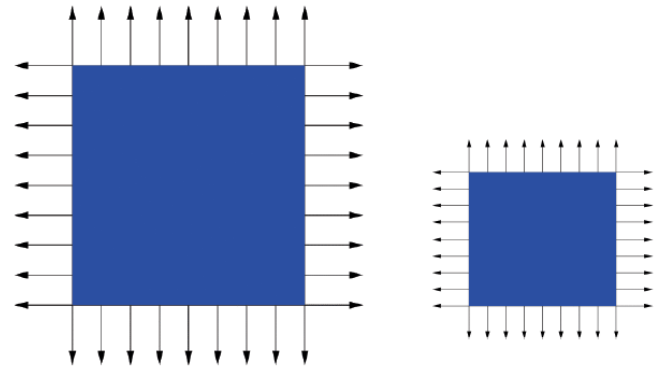
- Given by model matrix (model-to-world transformation) \mathbf{M} .
- \mathbf{M} is usually a combination of rotation, translation and scale.

$$\mathbf{p}_{world} = \mathbf{M}\mathbf{p}_{object}$$



Transform Normal like Object?

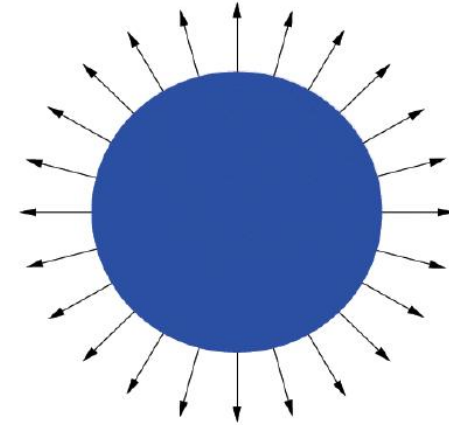
- We could find that translation, rotation and isotropic scale preserve the correct normal.
- But shear does not.



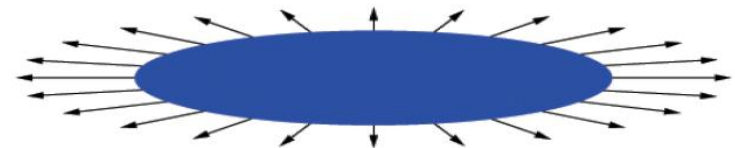
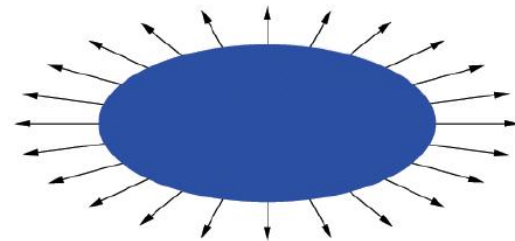
Transform Normal like Object?

- Another example

- Scale an sphere (but not isotropic scale)

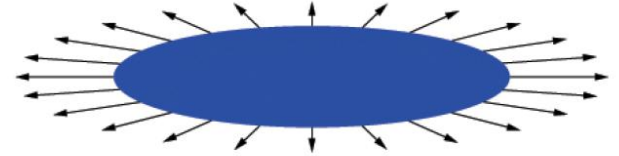
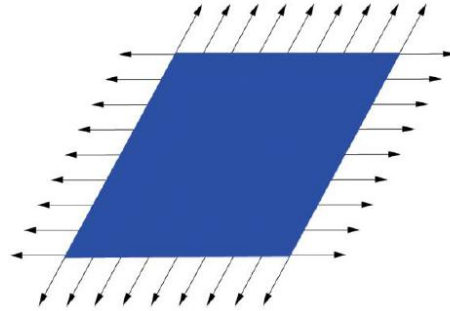


- The transformed normal is also incorrect

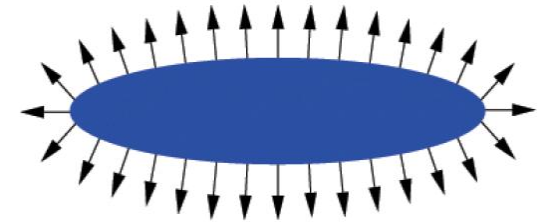
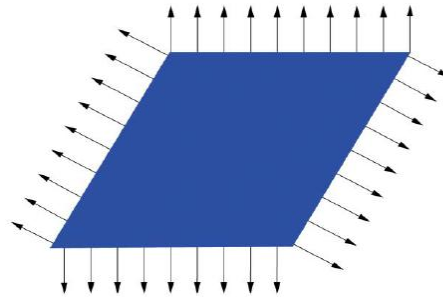


How to transform Normal?

Incorrect
Normal
Transformation

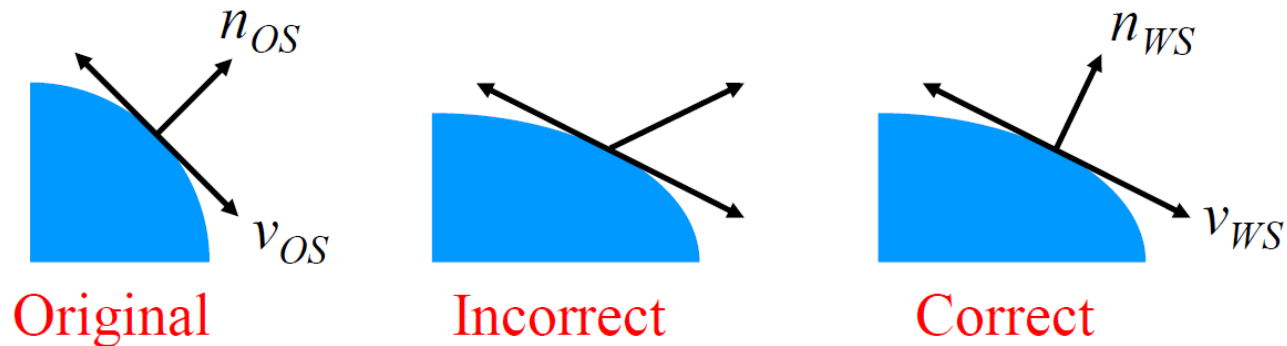


Correct
Normal
Transformation



How to transform normal?

- Transforming the tangent plane to the normal, not the normal vector directly



- Pick any vector v_{OS} in the tangent plane to transform

$$v_{WS} = \mathbf{M} v_{OS}$$

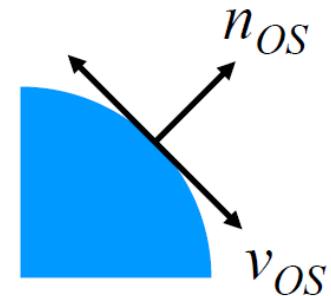
How to transform Normal?

Dot product $n_{OS}^T v_{OS} = 0$

$$n_{OS}^T (\mathbf{M}^{-1} \mathbf{M}) v_{OS} = 0$$

$$(n_{OS}^T \mathbf{M}^{-1}) (\mathbf{M} v_{OS}) = 0$$

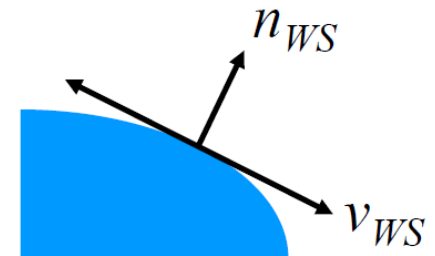
$$(n_{OS}^T \mathbf{M}^{-1}) v_{WS} = 0$$



is perpendicular to normal

$$n_{WS}^T = n_{OS}^T (\mathbf{M}^{-1})$$

$$n_{WS} = (\mathbf{M}^{-1})^T n_{OS}$$



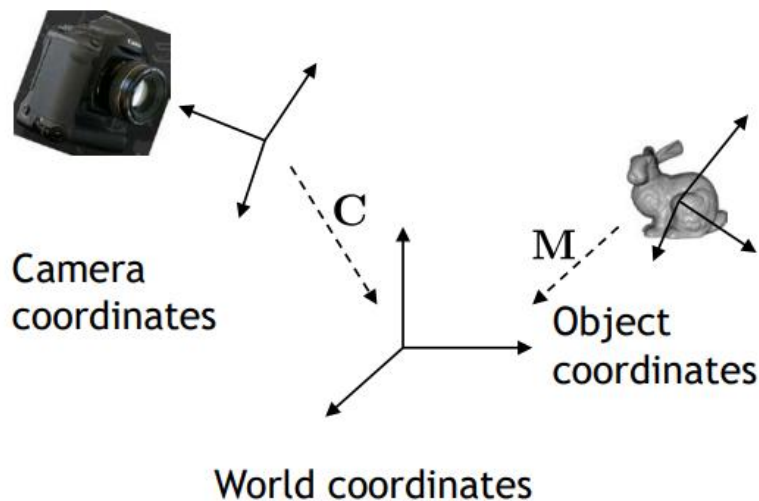
The matrix is the transpose of inverse of \mathbf{M}

This Lecture

- CG Coordinate Systems
 - Viewing pipeline
- Camera Transformation
- Projection Transformation
 - Orthographic projection
 - Perspective projection
- Viewport Transformation

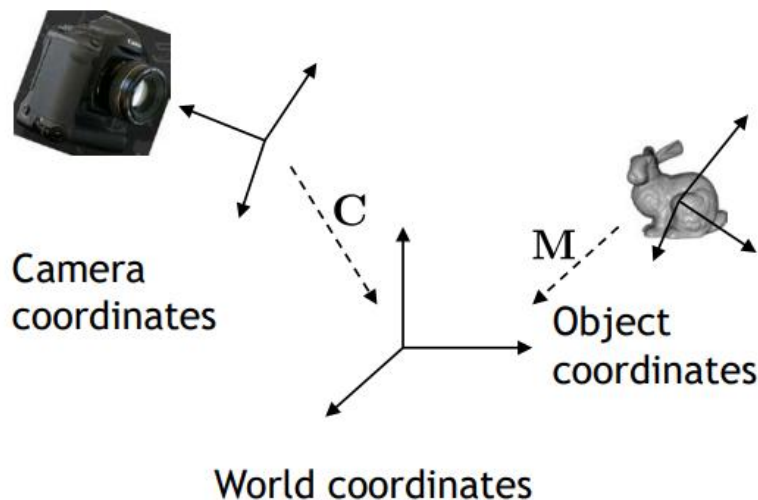
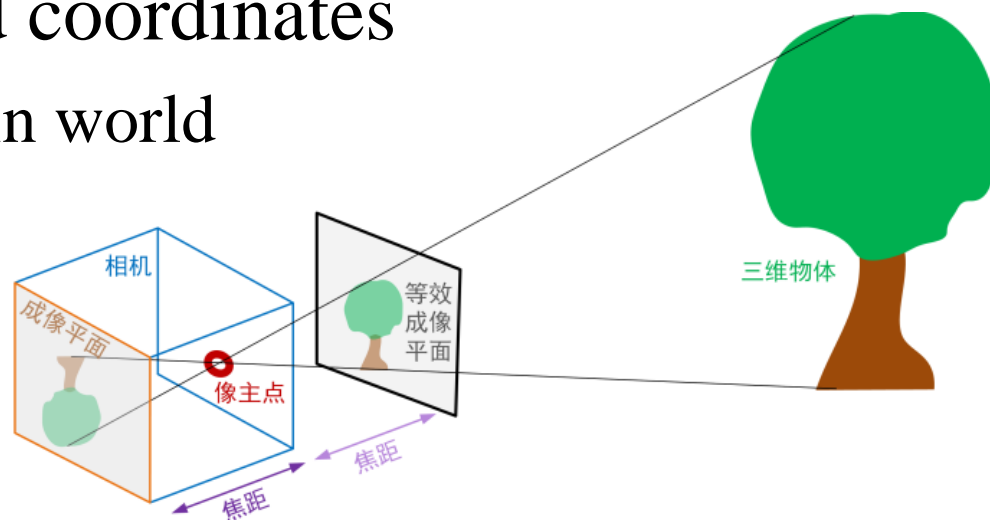
Camera Coordinate System

- Origin defines center of projection of camera
- **x-y** plane is parallel to image plane
- **z**-axis is perpendicular to image plane



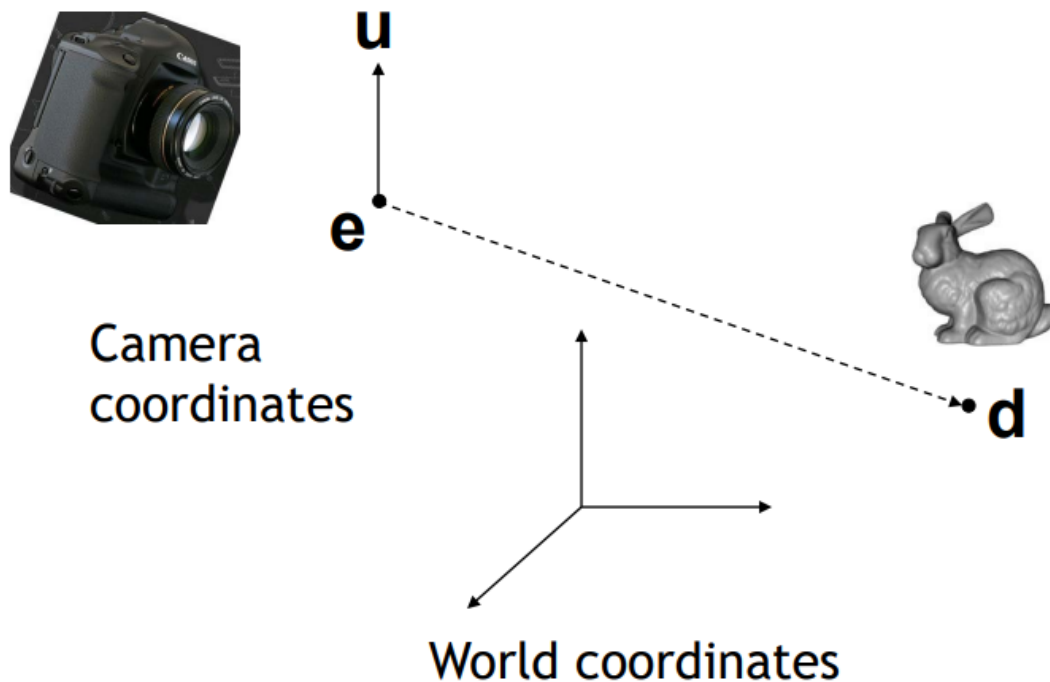
Camera Coordinate System

- The Camera Matrix defines the transformation from camera to world coordinates
 - Placement of camera in world



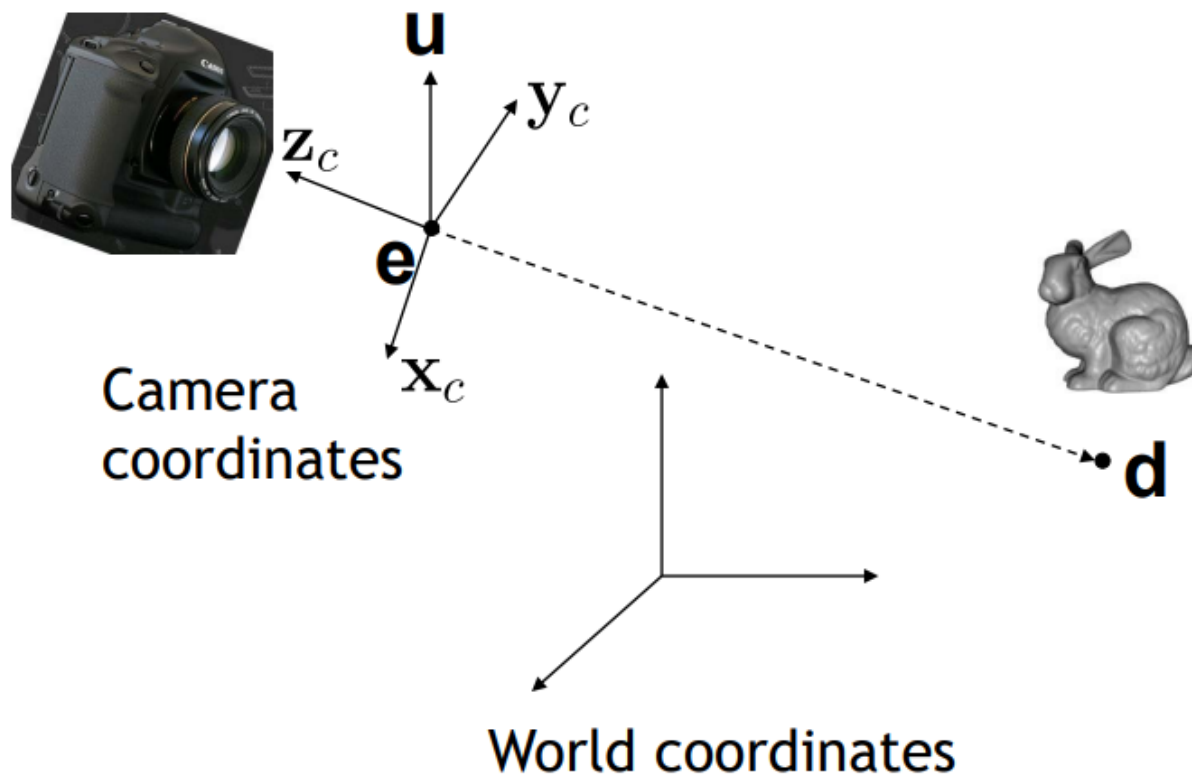
Camera Matrix

- Given:
 - Center point of projection \mathbf{e}
 - Look at point \mathbf{d}
 - Camera up vector \mathbf{u}



Camera Matrix

- Construct \mathbf{x}_c , \mathbf{y}_c , \mathbf{z}_c



Camera Matrix

- Z-axis

$$\mathbf{z}_C = \frac{\mathbf{e} - \mathbf{d}}{\|\mathbf{e} - \mathbf{d}\|}$$

- X-axis

$$\mathbf{x}_C = \frac{\mathbf{u} \times \mathbf{z}_C}{\|\mathbf{u} \times \mathbf{z}_C\|}$$

- Y-axis

$$\mathbf{y}_C = \mathbf{z}_C \times \mathbf{x}_C$$

- Camera matrix

$$\mathbf{C} = \begin{bmatrix} \mathbf{x}_C & \mathbf{y}_C & \mathbf{z}_C & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transform object to camera coordinate

- Object to world coordinates: \mathbf{M}
- Camera to world coordinate: \mathbf{C}
- Point to transform: \mathbf{p}
- Resulting transformation equation:

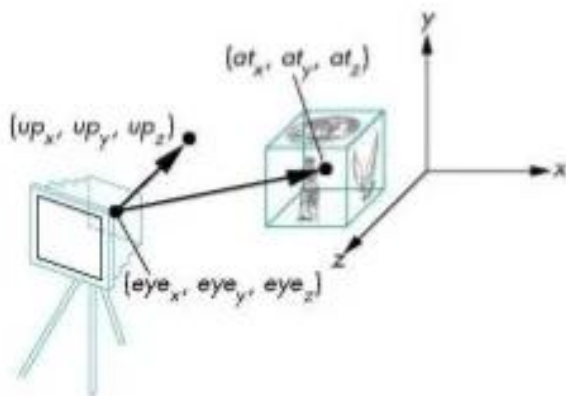
$$\mathbf{p}' = \mathbf{C}^{-1}\mathbf{M}\mathbf{p}$$

Inverse of Camera Matrix

- Generic matrix inversion is complex and compute-intensive!
- Observation
 - Camera matrix consists of translation and rotation \mathbf{TR}
- Inverse of rotation: $\mathbf{R}^{-1} = \mathbf{R}^T$
- Inverse of translation: $\mathbf{T}(\mathbf{t})^{-1} = \mathbf{T}(-\mathbf{t})$
- Inverse of camera matrix: $\mathbf{C}^{-1} = \mathbf{R}^{-1}\mathbf{T}^{-1}$

Camera Matrix

- gluLookAt



```
void gluLookAt( GLdouble eyeX,
                GLdouble eyeY,
                GLdouble eyeZ,
                GLdouble centerX,
                GLdouble centerY,
                GLdouble centerZ,
                GLdouble upX,
                GLdouble upY,
                GLdouble upZ);
```

Parameters

eyeX, eyeY, eyeZ

Specifies the position of the eye point.

centerX, centerY, centerZ

Specifies the position of the reference point.

upX, upY, upZ

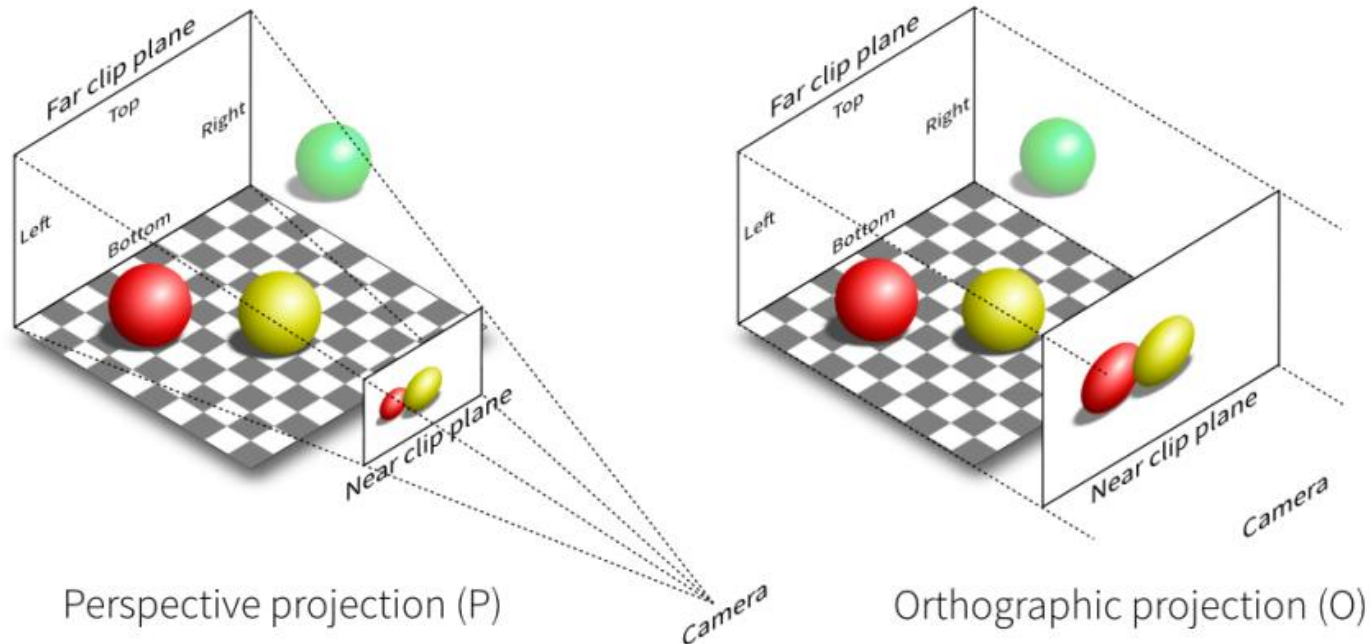
Specifies the direction of the *up* vector.

This Lecture

- CG Coordinate Systems
 - Viewing pipeline
- Camera Transformation
- Projection Transformation
 - Orthographic projection
 - Perspective projection
- Viewport Transformation

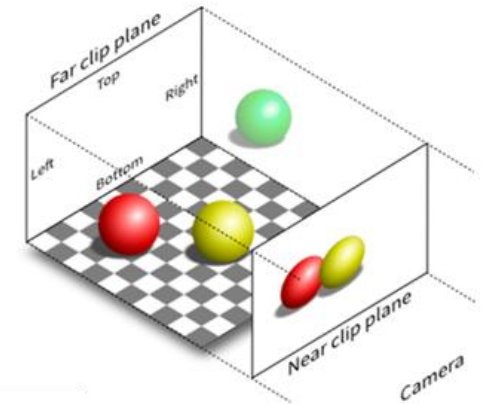
Projection Transformation

- Projection in Computer Graphics
 - 3D to 2D
 - Orthographic projection
 - Perspective projection

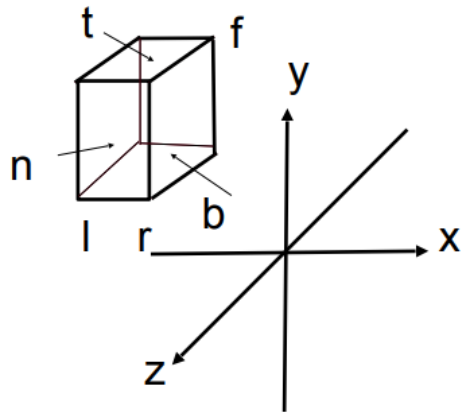


Orthographic Projection

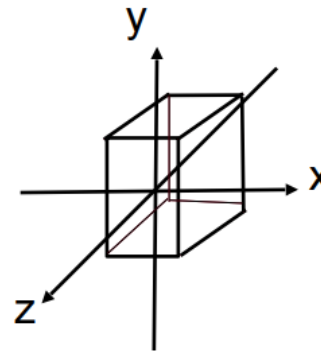
- In general
 - We want to map a cuboid $[l, r] \times [b, t] \times [f, n]$ to the canonical(正则、规范、标准) cube $[-1, 1]^3$
 - Center cuboid by translating
 - Scale into canonical cube



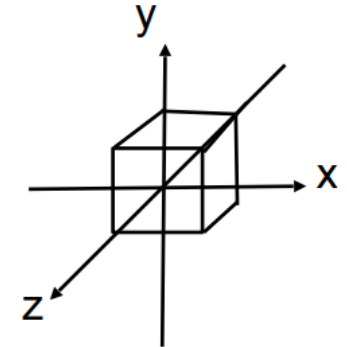
Orthographic projection (O)



Translate



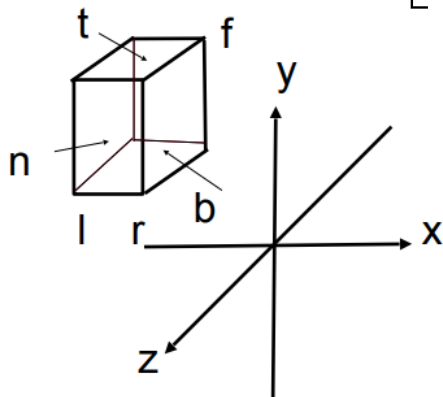
Scale



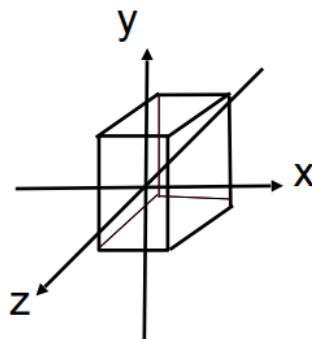
Orthographic Projection

- Transformation matrix?
 - Translate (center to origin) first, then scale (length/width/height to 2)

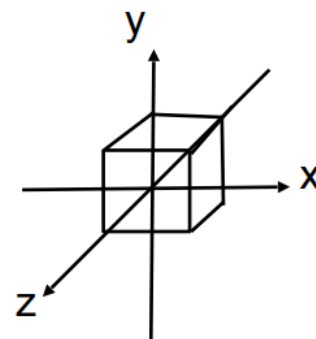
$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Translate



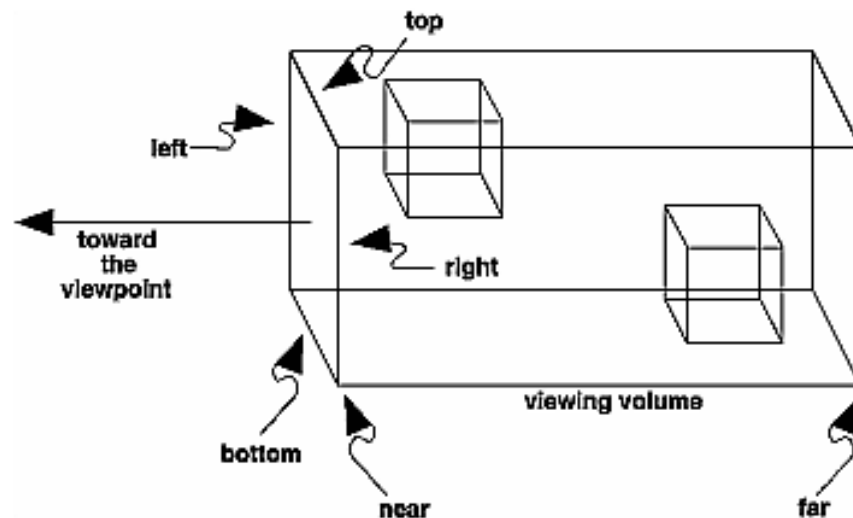
Scale



Orthographic Projection

- glOrtho

```
void glOrtho( GLdouble left,  
              GLdouble right,  
              GLdouble bottom,  
              GLdouble top,  
              GLdouble nearVal,  
              GLdouble farVal);
```



Parameters

left, right

Specify the coordinates for the left and right vertical clipping planes.

bottom, top

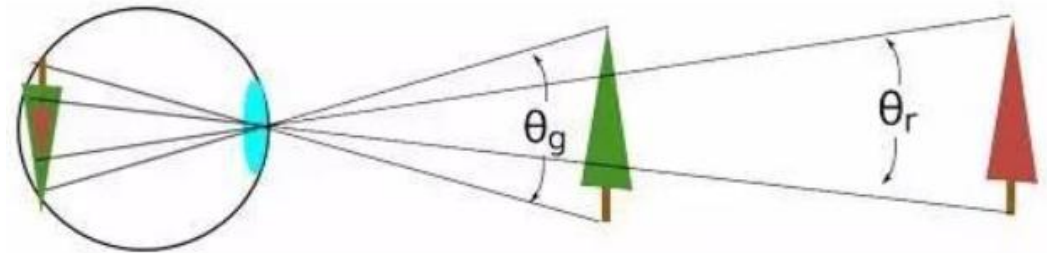
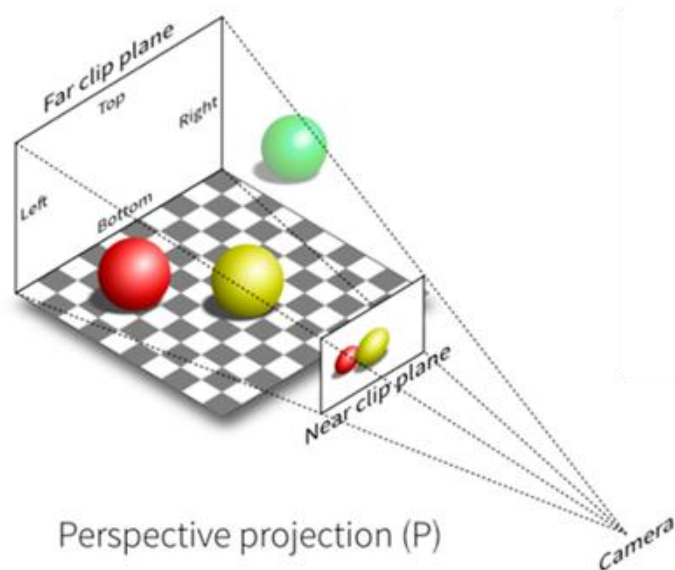
Specify the coordinates for the bottom and top horizontal clipping planes.

nearVal, farVal

Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

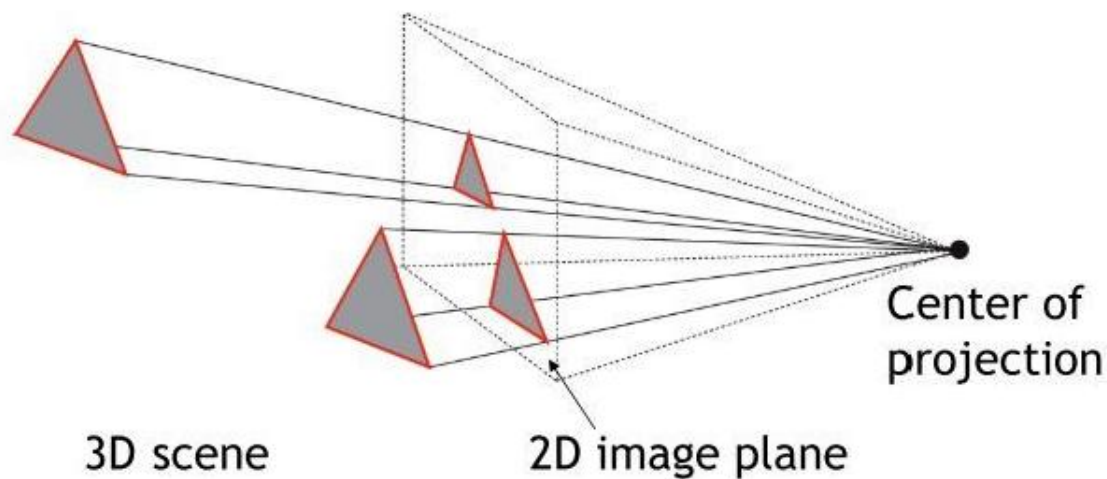
Perspective Projection

- Most common for Computer Graphics
- Simplified model of human eye, or camera lens (pinhole camera)
- Things farther away appear to be smaller



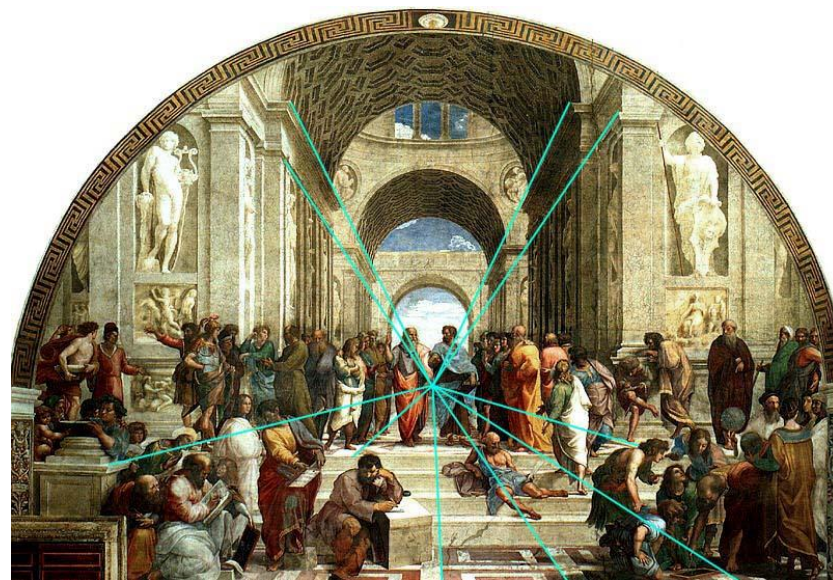
Perspective Projection

- Project along rays that converge in center of projection



Perspective projection

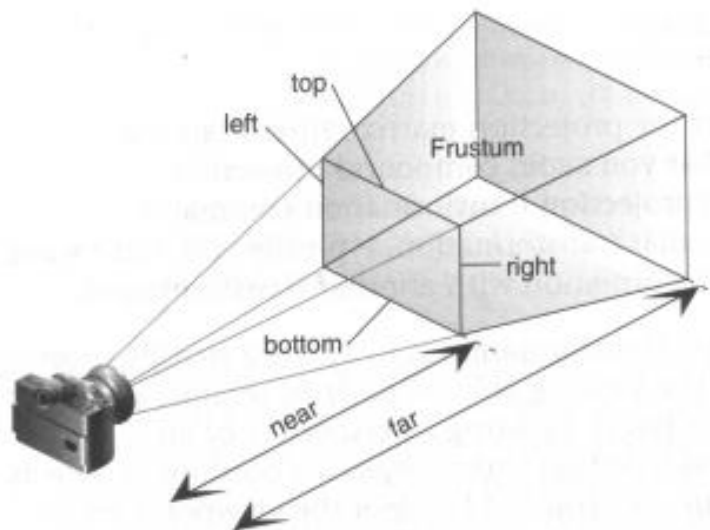
- Parallel lines are no longer parallel, converge in one point



Perspective Projection

- `glFrustum`

```
void glFrustum(GLdouble left,  
               GLdouble right,  
               GLdouble bottom,  
               GLdouble top,  
               GLdouble near,  
               GLdouble far)
```



PARAMETERS

left, right

Specify the coordinates for the left and right vertical clipping planes.

bottom, top

Specify the coordinates for the bottom and top horizontal clipping planes.

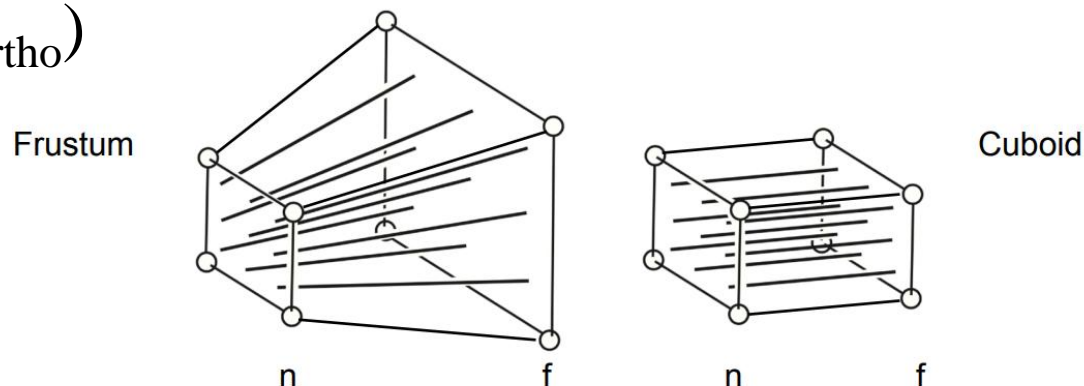
near, far

Specify the distances to the near and far depth clipping planes. Both distances must be positive.

Perspective Projection

- How to do perspective projection
 - First squish the frustum into a cuboid

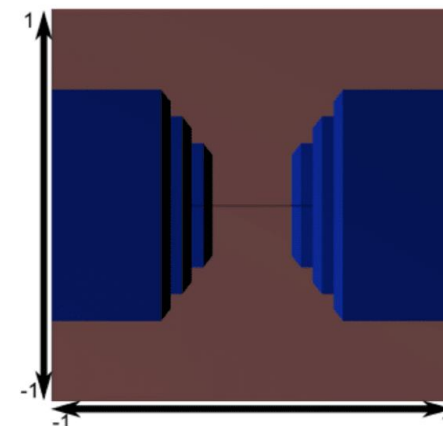
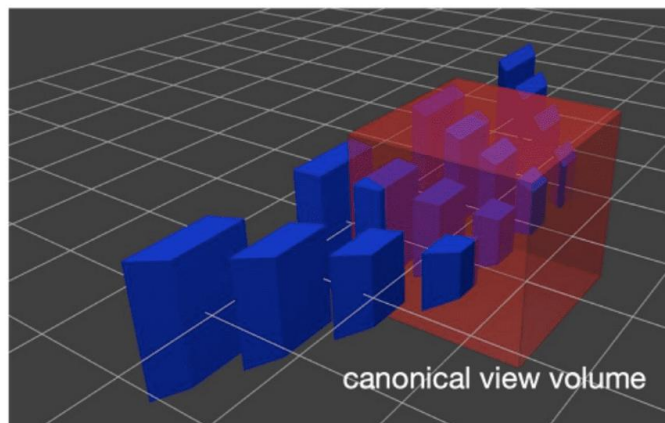
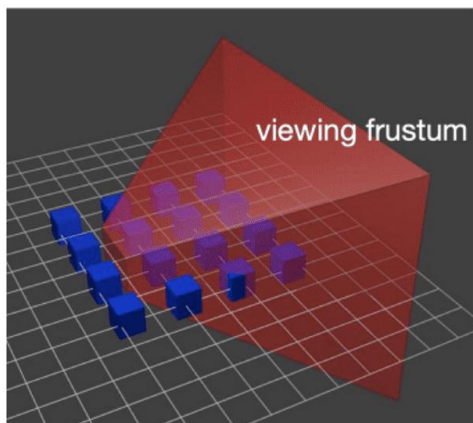
$$(n \rightarrow n, f \rightarrow f)(\mathbf{M}_{\text{persp} \rightarrow \text{ortho}})$$



Camera space (View frustum) →

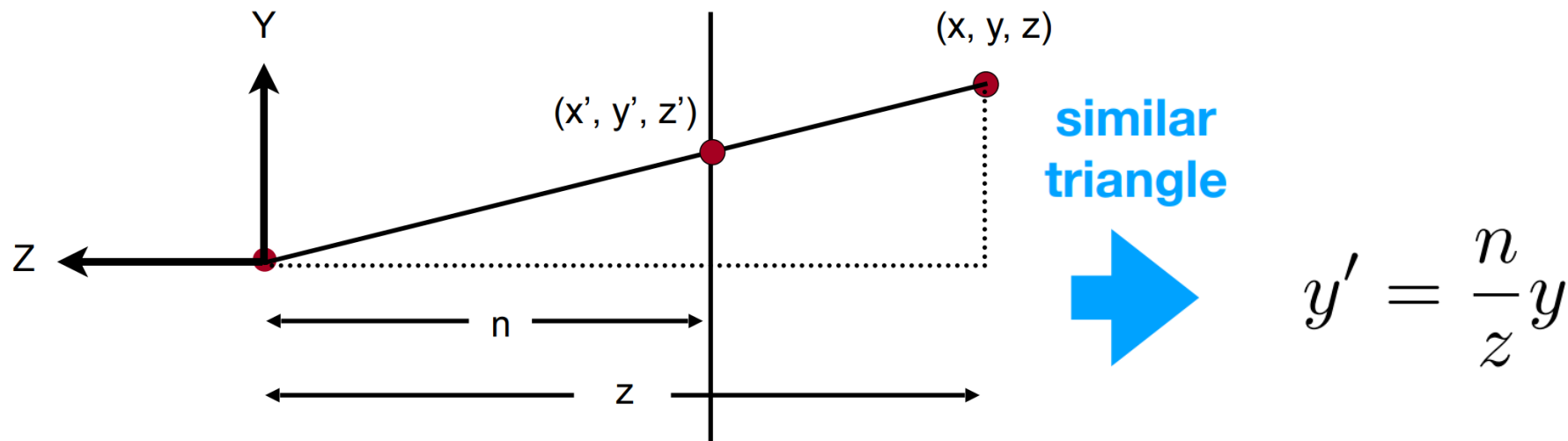
Canonical View Volume →

Screen



Perspective Projection

- In order to find a transformation
 - Recall the key idea: Find the relationship between transformed points (x', y', z') and the original points (x, y, z)



Perspective Projection

- In order to find a transformation
 - Find the relationship between transformed points (x', y', z') and the original points (x, y, z)

$$y' = \frac{n}{z}y \quad x' = \frac{n}{z}x \text{ (similar to } y')$$

- In homogeneous coordinates,

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} nx/z \\ ny/z \\ \text{unknown} \\ 1 \end{pmatrix} \xrightarrow{\text{mult. by } z} \begin{pmatrix} nx \\ ny \\ \text{still unknown} \\ z \end{pmatrix}$$

Perspective Projection

- So the squish (persp to ortho) projection does this

$$M_{persp \rightarrow ortho}^{(4 \times 4)} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

- Already good enough to figure out part of $\mathbf{M}_{persp \rightarrow ortho}$

$$M_{persp \rightarrow ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Perspective Projection

- How to figure out the third row of $\mathbf{M}_{\text{persp} \rightarrow \text{ortho}}$
 - Any information that we can use?

$$M_{\text{persp} \rightarrow \text{ortho}} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Observation: the third row is responsible for z'
 - Any point on the near plane will not change
 - Any point's z on the far plane will not change

Perspective Projection

- Any point on the near plane will not change

$$M_{persp \rightarrow ortho}^{(4 \times 4)} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix} \xrightarrow{\text{replace } z \text{ with } n} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} == \begin{pmatrix} nx \\ ny \\ n^2 \\ n \end{pmatrix}$$

- So the third row must be of the form (0,0,A,B)

$$\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2 \quad \text{n}^2 \text{ has nothing to do with } x \text{ and } y$$

Perspective Projection

- What do we have now?

$$\begin{pmatrix} 0 & 0 & A & B \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2 \quad \Rightarrow \quad An + B = n^2$$

- Any point's z on the far plane will not change

$$\begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} == \begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} \quad \Rightarrow \quad Af + B = f^2$$

Perspective Projection

- Solve for A and B

$$\begin{array}{l} An + B = n^2 \\ Af + B = f^2 \end{array} \quad \rightarrow \quad \begin{array}{l} A = n + f \\ B = -nf \end{array}$$

- Finally, every entry in $\mathbf{M}_{\text{persp} \rightarrow \text{ortho}}$ is known!
- What's next?
 - Do orthographic projection $\mathbf{M}_{\text{ortho}}$
 - $\mathbf{M}_{\text{persp}} = \mathbf{M}_{\text{ortho}} \mathbf{M}_{\text{persp} \rightarrow \text{ortho}}$

ModelView & Projection

- Specified in two parts
- First the projection
 - `glMatrixMode(GL_PROJECTION)`
 - `glLoadIdentity()`
 - `glFrustum(-4,+4, //left & right
-3, +3, //top & bottom
5, 80); // near & far`
- Second the model-view
 - `glMatrixMode(GL_MODELVIEW)`
 - `glLoadIdentity()`
 - `glTranslatef(0, 0, -14)`

Resulting projection matrix

$$\begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.667 & 0 & 0 \\ 0 & 0 & -1.1333 & -10.667 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Resulting modelview matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -14 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Modelview-Projection Transform Matrix

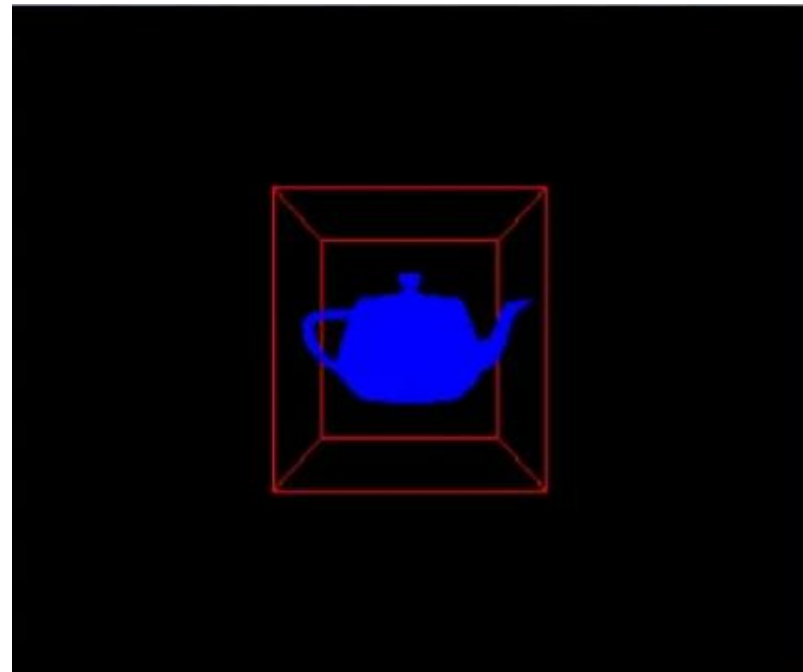
- Transform composition via matrix multiplication

$$\begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.667 & 0 & 0 \\ 0 & 0 & -1.1333 & -10.667 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -14 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 = \begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.667 & 0 & 0 \\ 0 & 0 & -1.1333 & 5.2 \\ 0 & 0 & -1 & 14 \end{bmatrix}$$

Resulting modelview-projection matrix

Draw Some Objects

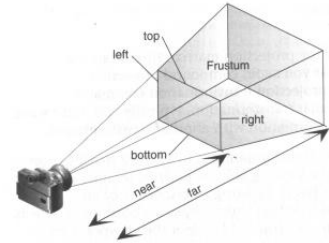
- Draw a wireframe cube
 - `glColor3f(1, 0, 0) // red`
 - `glutWireCube(6)`
- Draw a teapot in the cube
 - `glColor3f(0, 0, 1) // blue`
 - `glutSolidTeapot(2.0)`
- As given a frustum transform, the cube is in perspective



What we've accomplished

- Simple perspective

- With `glFrustum`
- Establishes how eye-space maps to clip-space



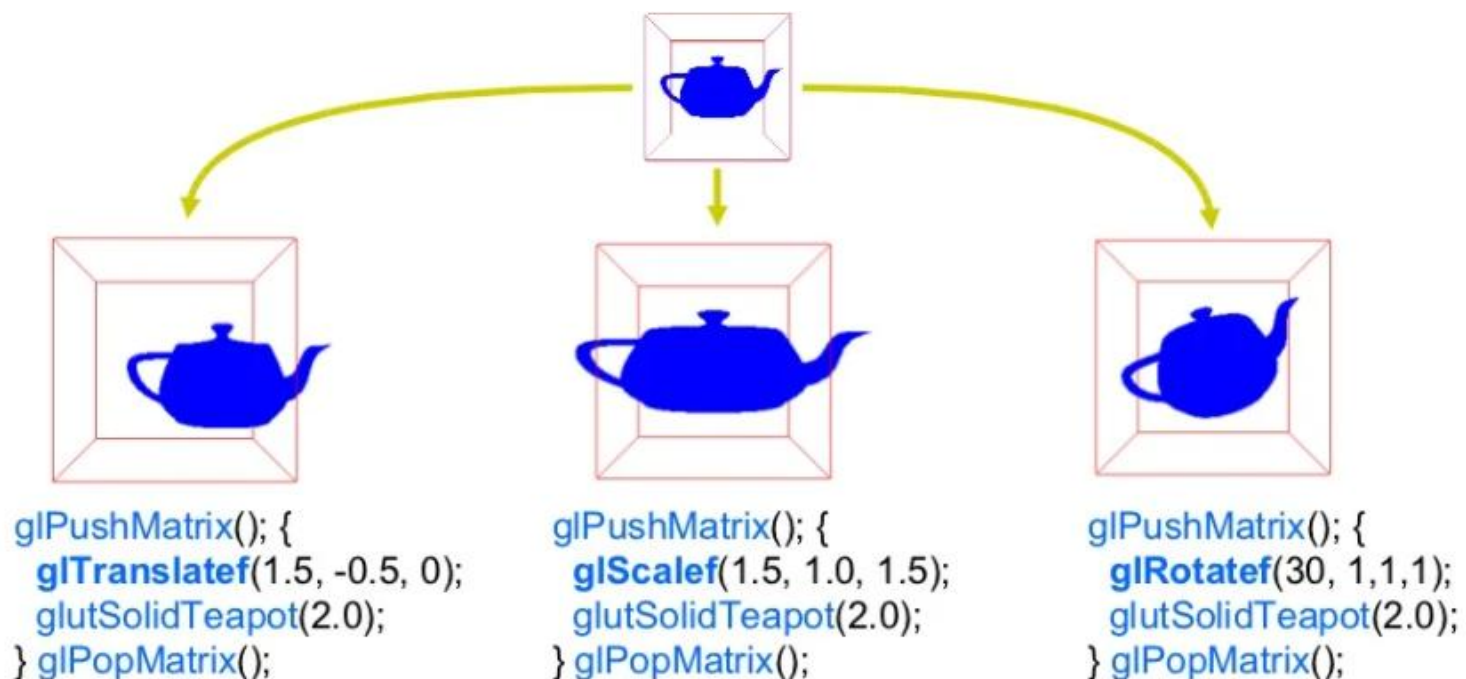
- Simple viewing

- With `glTranslatef`
- Establishes how world-space maps to eye-space
- No actual modeling transforms, just viewing
 - Modeling would be rotating, scaling, or otherwise transform the objects with the view
 - Arguably the model view matrix is really just a view matrix in this example



Add some simple Modeling

- Try some modeling transforms to move teapot
 - Leave the cube alone for reference



Notice: We "bracket" the modeling transform with `glPushMatrix/glPopMatrix` commands so the modeling transforms are "localized" to the particular object

Modelview-Projection Matrix

- Consider the combined modelview matrix with the rotation

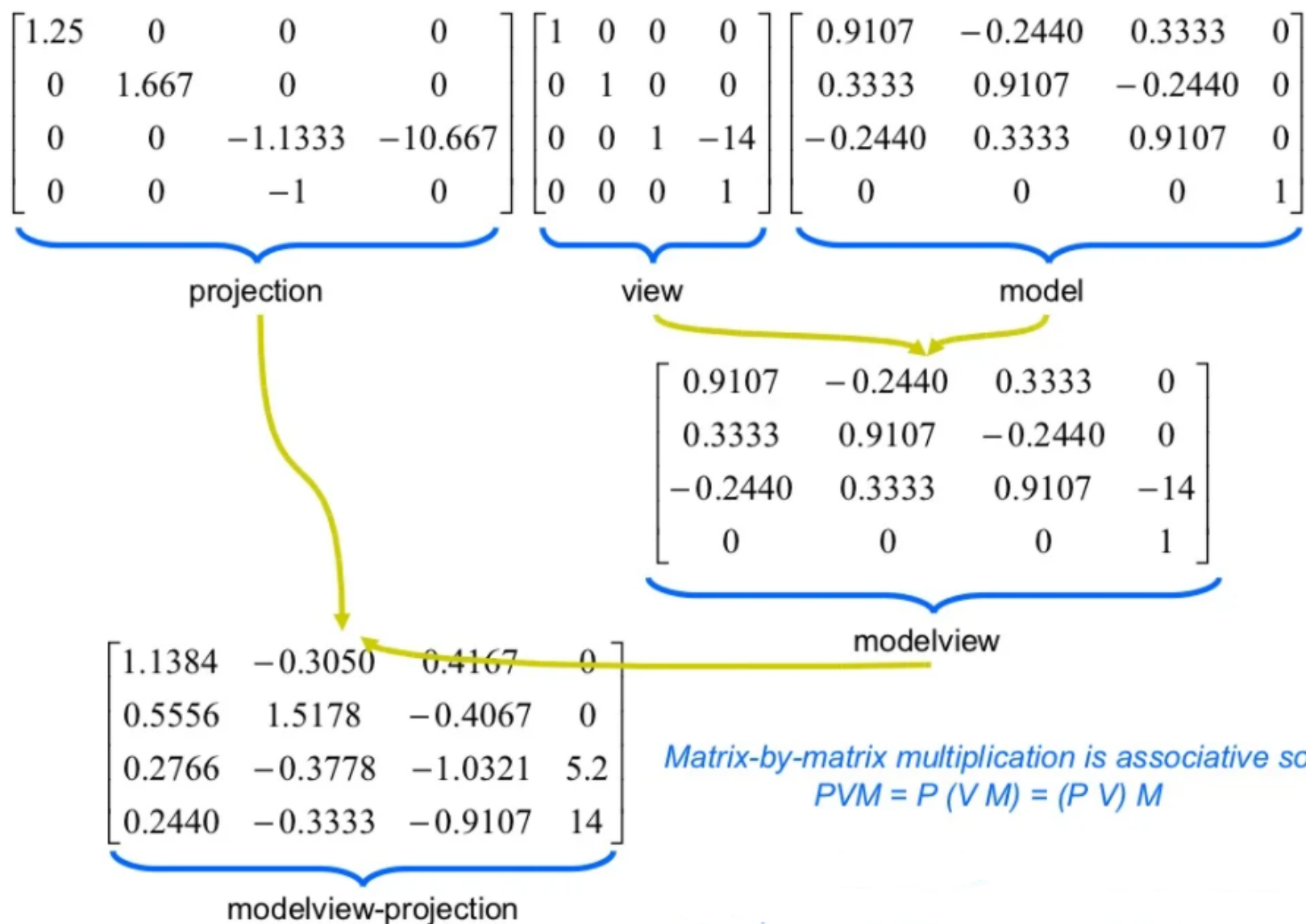
- glRotate(30, 1, 1, 1) defines a rotation matrix

- Rotating 30 degrees
- Around the axis in the (1, 1, 1) direction

$$\begin{bmatrix} 0.9107 & -0.2440 & 0.3333 & 0 \\ 0.3333 & 0.9107 & -0.2440 & 0 \\ -0.2440 & 0.3333 & 0.9107 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.667 & 0 & 0 \\ 0 & 0 & -1.1333 & -10.667 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -14 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{view}} \underbrace{\begin{bmatrix} 0.9107 & -0.2440 & 0.3333 & 0 \\ 0.3333 & 0.9107 & -0.2440 & 0 \\ -0.2440 & 0.3333 & 0.9107 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{model}}$$

Combining All Three



Math Paths from object- to Clip-space

model

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ w_{world} \end{bmatrix} = \begin{bmatrix} 0.9107 & -0.2440 & 0.3333 & 0 \\ 0.3333 & 0.9107 & -0.2440 & 0 \\ -0.2440 & 0.3333 & 0.9107 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{bmatrix}$$

view

$$\begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -14 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ w_{world} \end{bmatrix}$$

projection

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = \begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.667 & 0 & 0 \\ 0 & 0 & -1.1333 & -10.667 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix}$$

object-to-world-to-eye-to-clip

modelview

$$\begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix} = \begin{bmatrix} 0.9107 & -0.2440 & 0.3333 & 0 \\ 0.3333 & 0.9107 & -0.2440 & 0 \\ -0.2440 & 0.3333 & 0.9107 & -14 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{bmatrix}$$

projection

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = \begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.667 & 0 & 0 \\ 0 & 0 & -1.1333 & -10.667 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix}$$

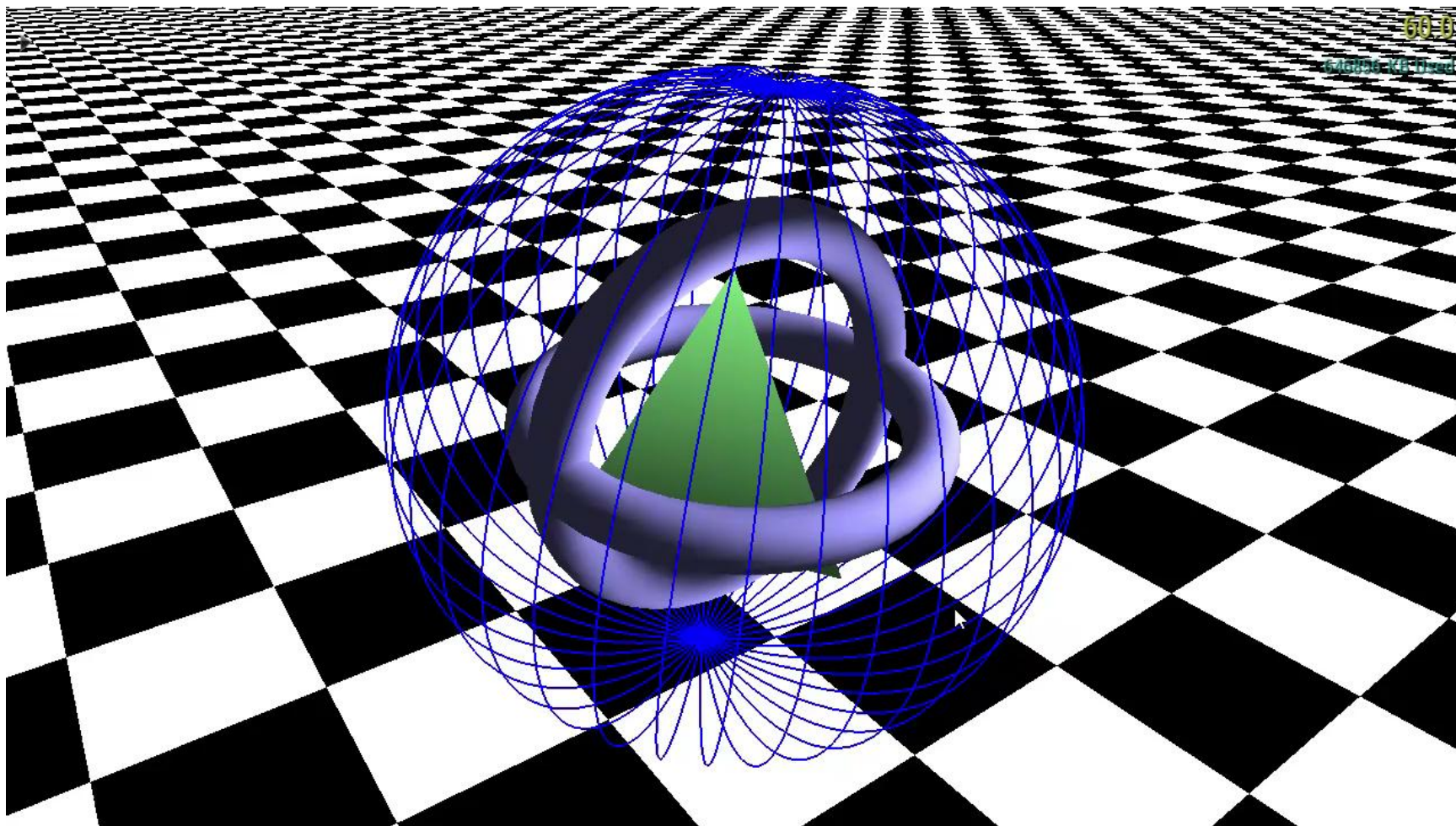
object-to-eye-to-clip

modelview-projection

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = \begin{bmatrix} 1.1384 & -0.3050 & 0.4167 & 0 \\ 0.5556 & 1.5178 & -0.4067 & 0 \\ 0.2766 & -0.3778 & -1.0321 & 5.2 \\ 0.2440 & -0.3333 & -0.9107 & 14 \end{bmatrix} \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{bmatrix}$$

object-to-clip

Projection example

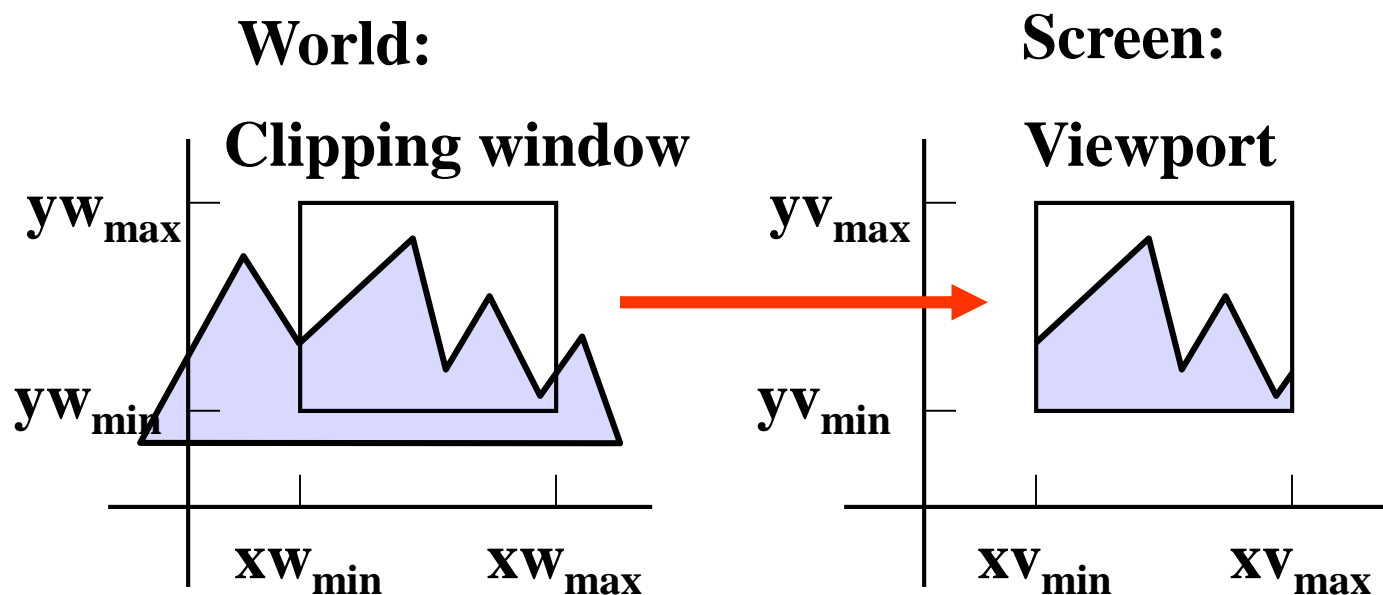


This Lecture

- CG Coordinate Systems
 - Viewing pipeline
- Camera Transformation
- Projection Transformation
 - Orthographic projection
 - Perspective projection
- Viewport Transformation

Viewport Transformation

- Transform points from world view (window) to the screen view (viewport)
 - Can be done with a translate-scale-translate sequence



Clipping window:

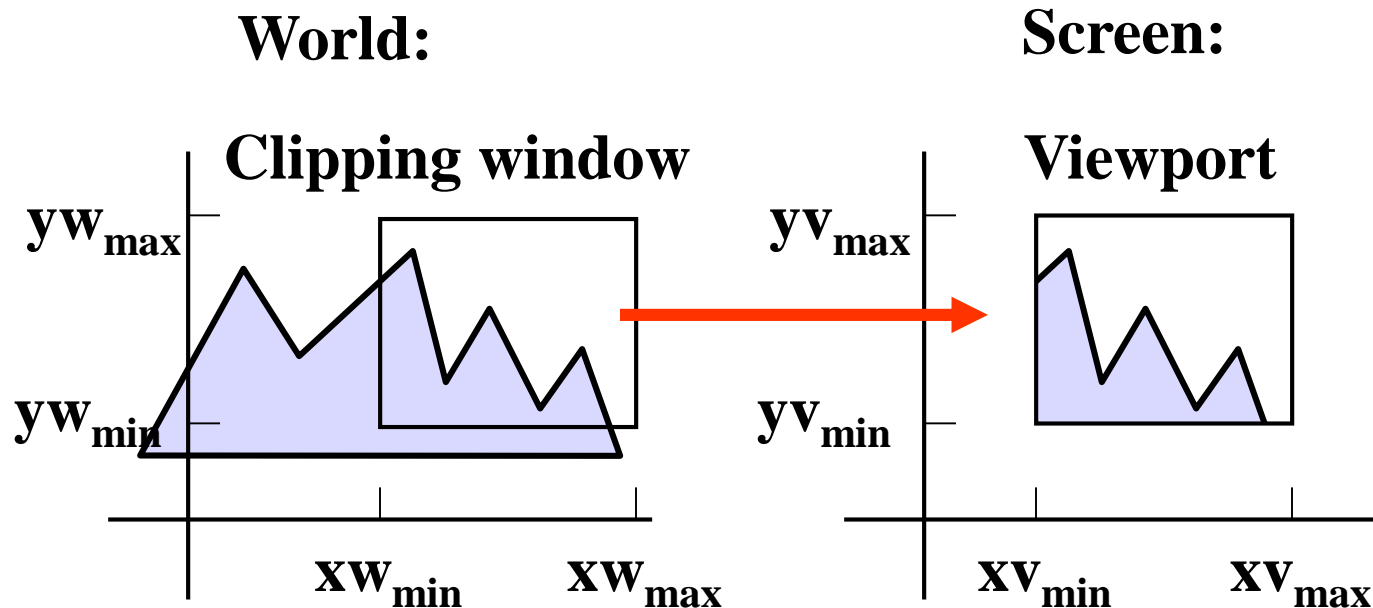
What do we want to see?

Viewport:

Where do we want to see it?

Viewport Transformation

- Transform points from world view (window) to the screen view (viewport)
 - Can be done with a translate-scale-translate sequence



Clipping window:

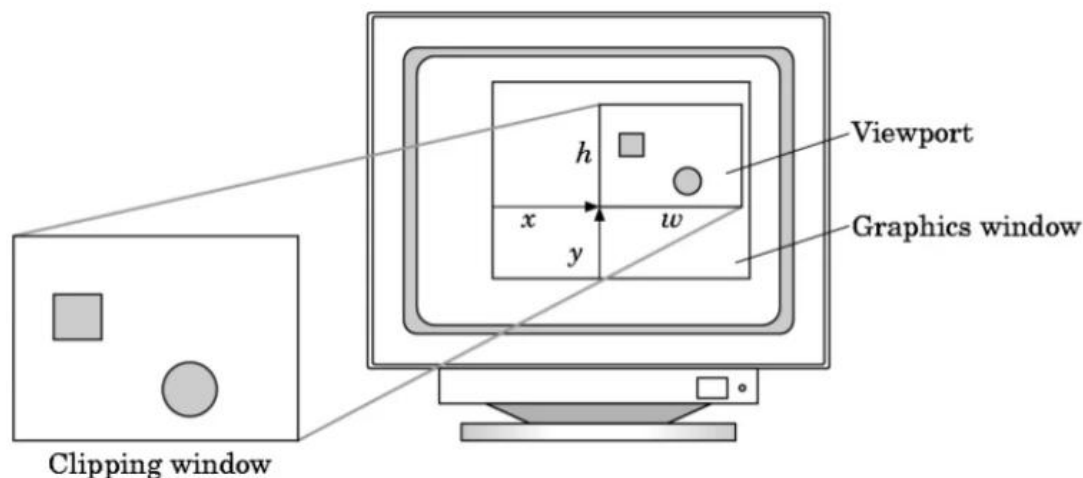
What do we want to see?

Viewport:

Where do we want to see it?

Viewport Transformation

- glViewport



```
void glViewport(GLint x,
               GLint y,
               GLsizei width,
               GLsizei height);
```

Parameters

x, y

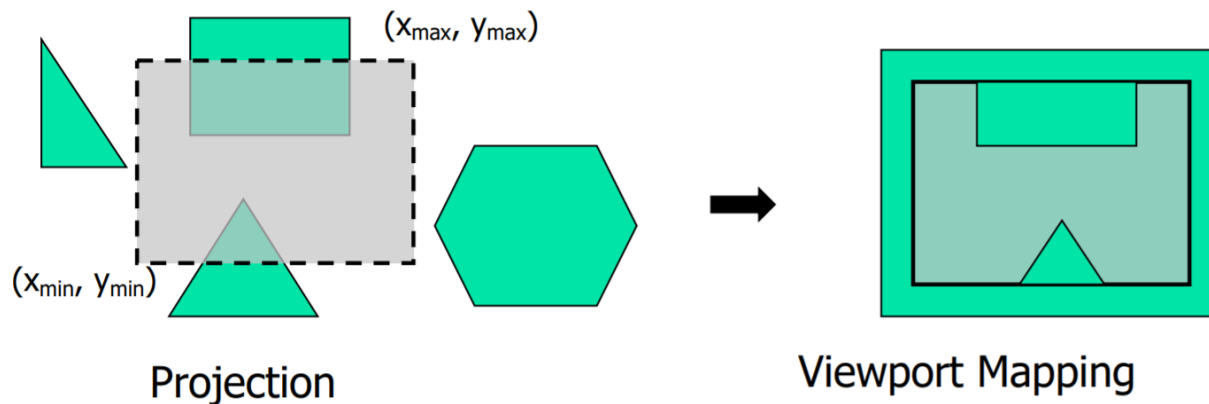
Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0).

$width, height$

Specify the width and height of the viewport. When a GL context is first attached to a window, $width$ and $height$ are set to the dimensions of that window.

From NDC to SC

- Just do a linear mapping
 - $[-1,-1] \times [1,1]$ to $[t_x, t_y] \times [t_x+W, t_y+H]$
- That is, assume (x, y) is in NDC, and (i, j) is in SC, then
 - $i = (x + 1) * 0.5 * W + t_x$
 - $j = (y + 1) * 0.5 * H + t_y$



This Lecture

- CG Coordinate Systems
 - Viewing pipeline
- Camera Transformation
- Projection Transformation
 - Orthographic projection
 - Perspective projection
- Viewport Transformation

谢谢



北京航空航天大学
人工智能学院(人工智能研究院)