



Chapter 5

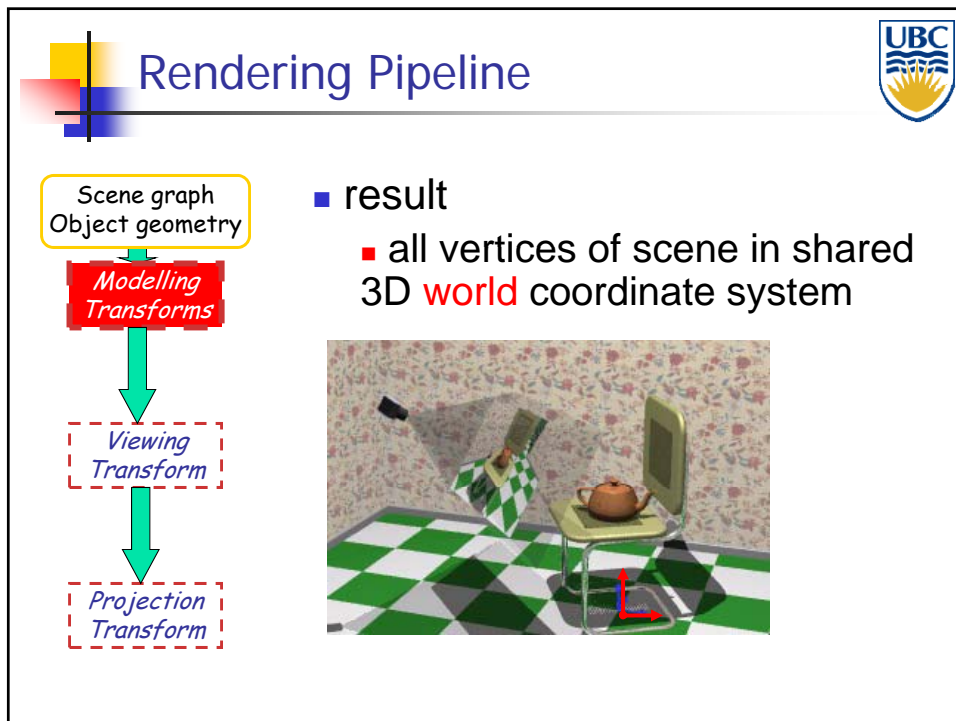
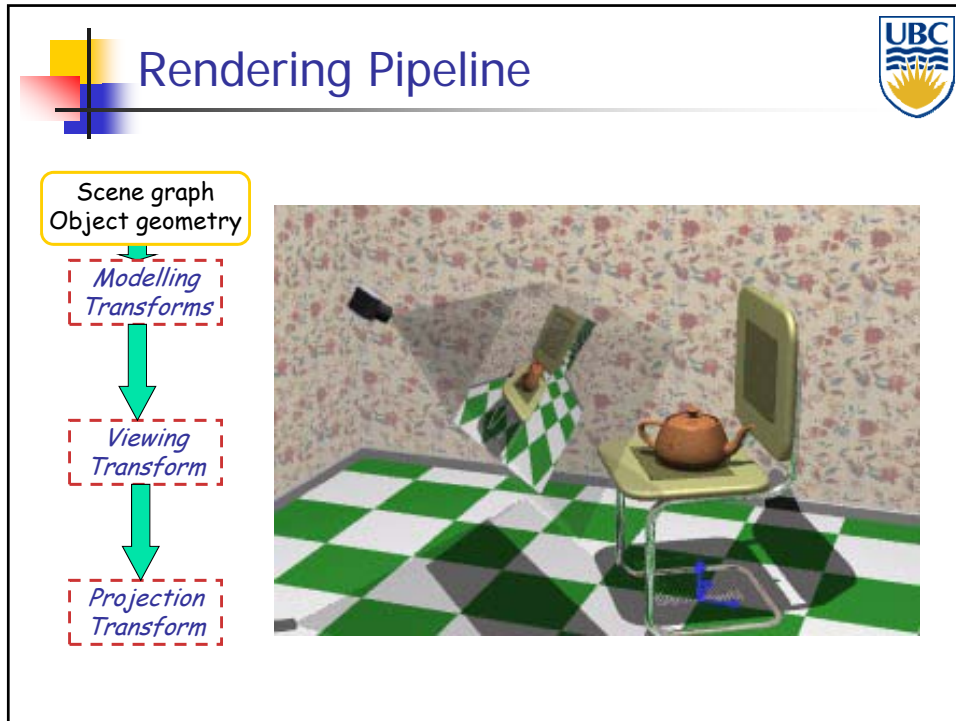
Viewing/Perspective Transformations

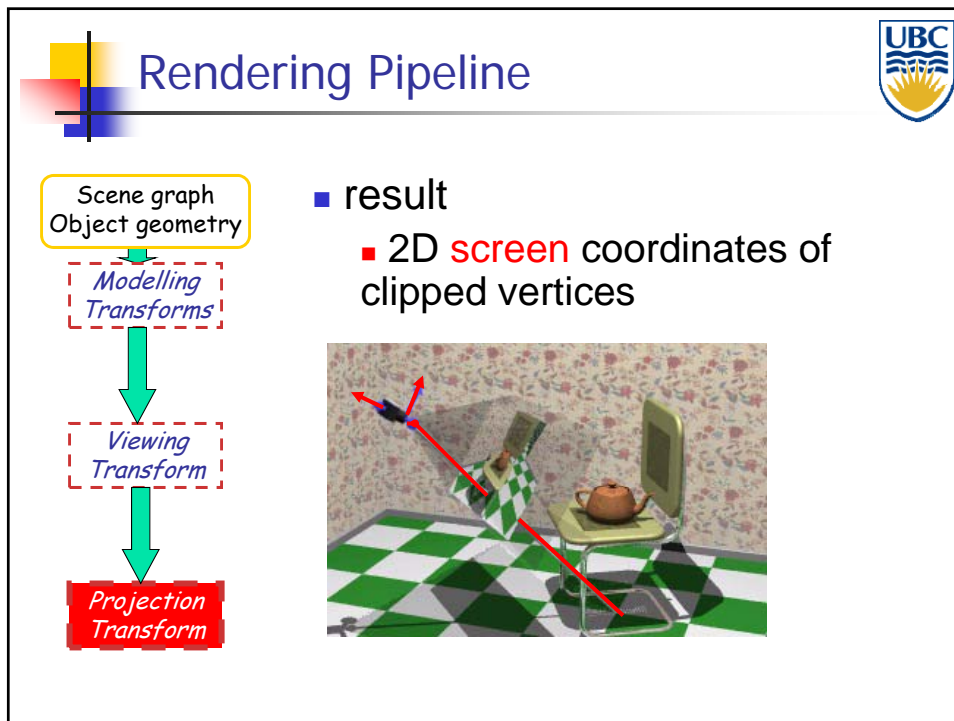
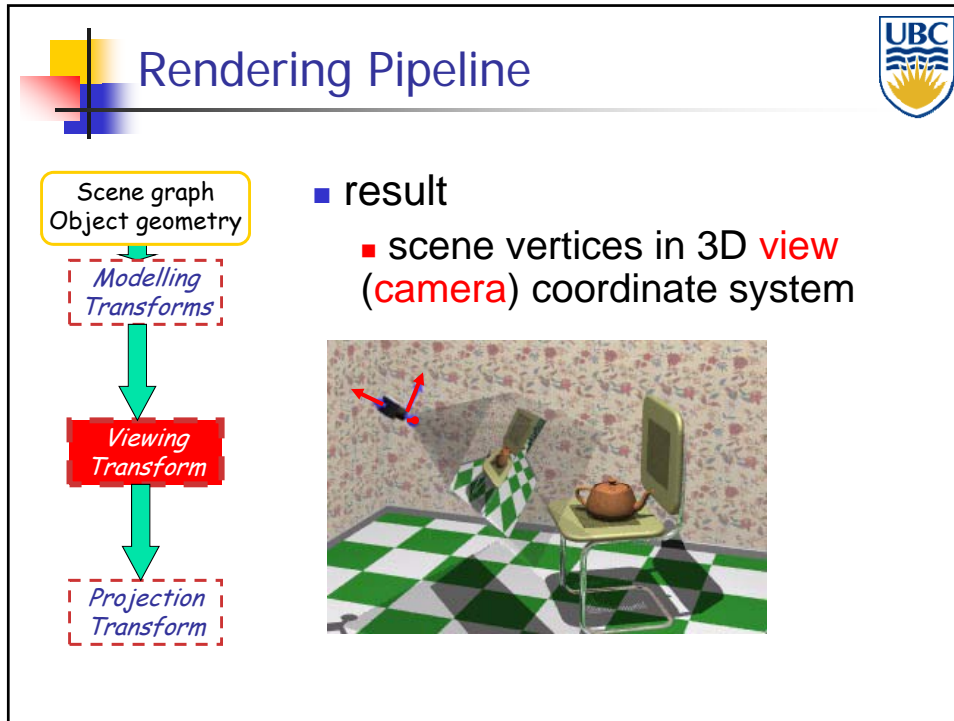


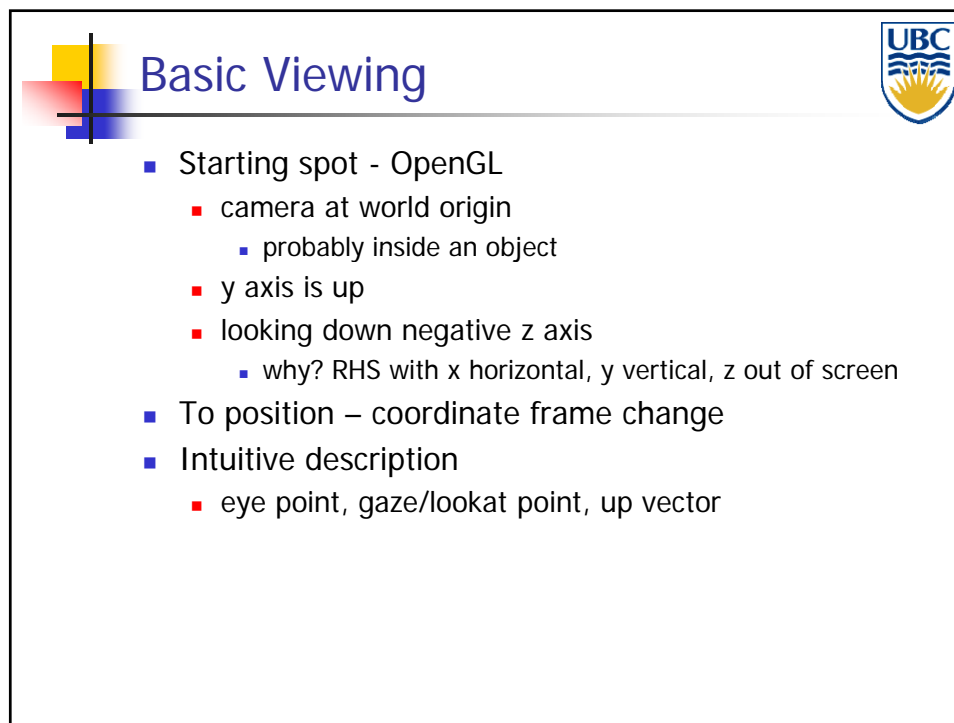
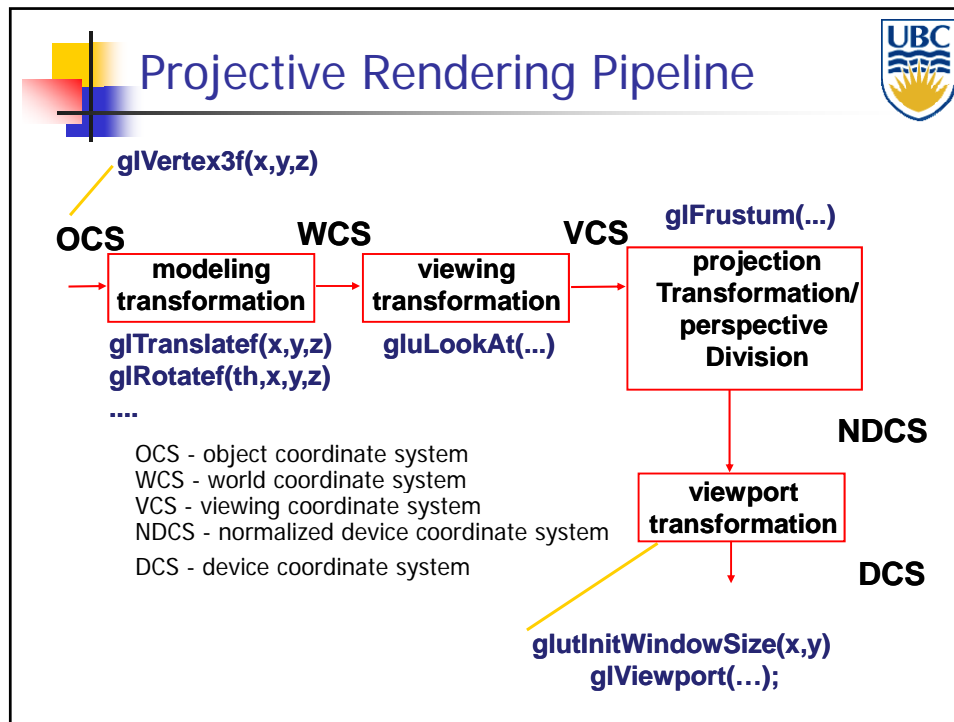
Rendering Pipeline

```
graph LR; GC((Geometric Content)) --> MVT[Model/View Transform.]; MVT --> L[Lighting]; L --> PT[Perspective Transform.]; PT --> C[Clipping]; C --> SC[Scan Conversion]; SC --> T[Texturing]; T --> DT[Depth Test]; DT --> B[Blending]; B --> FB((Frame-buffer));
```

- Specify view point (change of coordinate system)
- Project from 3D to 2D (introduce perspective)

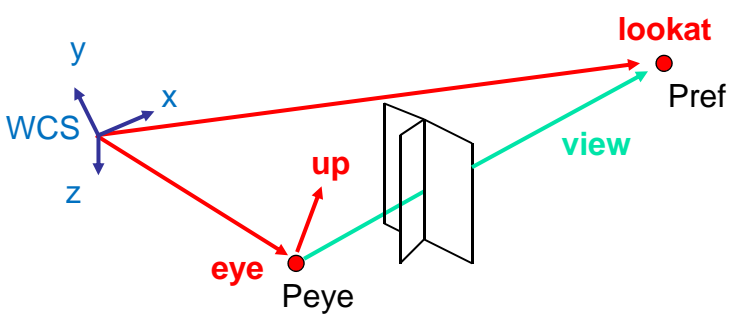






Camera Description/Motion

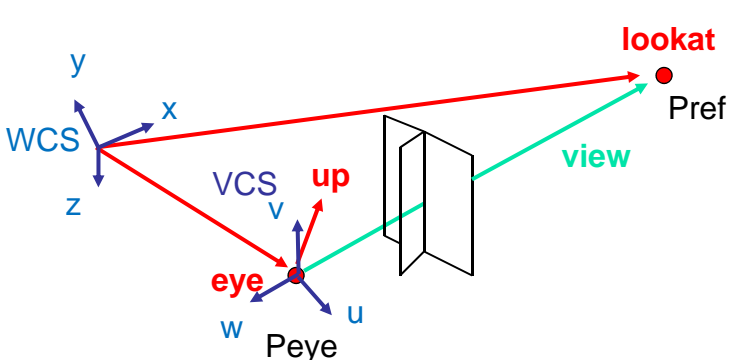
- arbitrary viewing position
 - eye point, gaze/lookat point, up vector




The diagram shows a 3D coordinate system with axes x, y, and z. The origin is labeled 'WCS'. A red dot labeled 'eye' is at position P_{eye} . A red dot labeled 'lookat' is at position P_{ref} . A red arrow points from 'eye' to 'lookat'. A red arrow labeled 'up' points upwards from the 'eye' point. A green arrow labeled 'view' points from the 'eye' point towards the 'lookat' point. A camera frustum is shown in the center, with the 'view' vector passing through its center.

From World to View Coordinates: W2V


- translate **eye** to origin
- rotate **view** vector (**lookat** - **eye**) to **w** axis
- rotate around **w** to bring **up** into **vw**-plane



The diagram shows the same scene as the previous one, but with a new coordinate system centered at the 'eye' point. The axes are labeled 'w', 'u', and 'v'. The 'w' axis points towards the 'lookat' point. The 'v' axis points upwards. The 'u' axis is perpendicular to the 'vw' plane. The 'WCS' axes are also shown for reference.



Deriving W2V Transformation



- $M = RT$


$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{\|\mathbf{t} \times \mathbf{w}\|}$$


$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

$$\mathbf{w} = -\hat{\mathbf{g}} = -\frac{\mathbf{g}}{\|\mathbf{g}\|}$$


$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{world \rightarrow view} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{e} \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{e} \\ w_x & w_y & w_z & -\mathbf{w} \cdot \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$





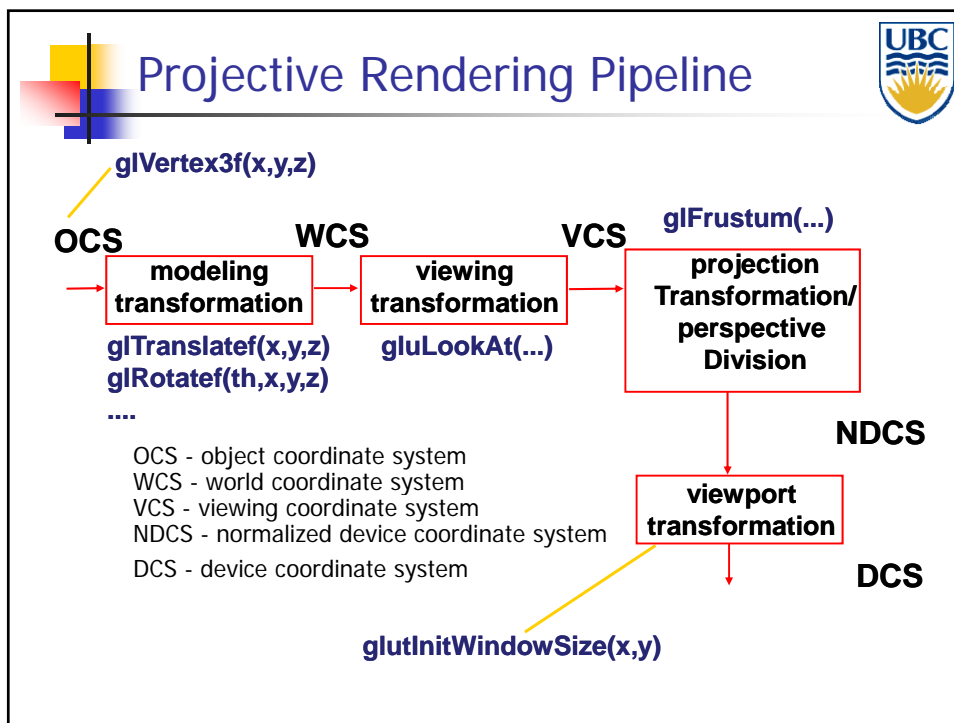
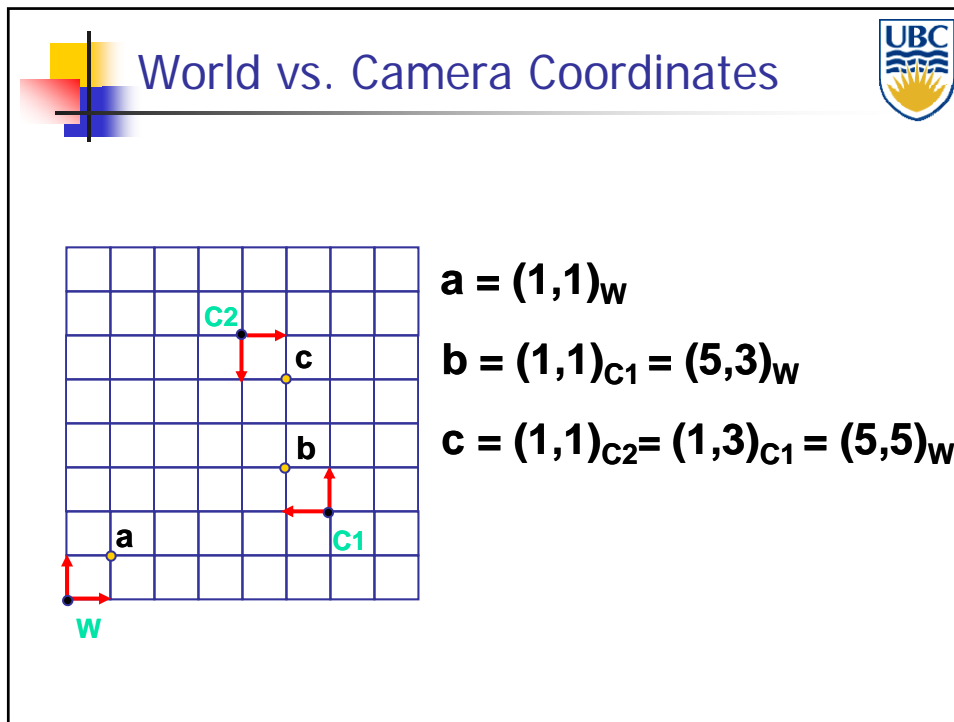
OpenGL Viewing Transformation




```
gluLookAt(ex,ey,ez,lx,ly,lz,ux,uy,uz)
```


- postmultiplies current matrix, so to be safe:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(ex,ey,ez,lx,ly,lz,ux,uy,uz)
// now ok to do model transformations
```

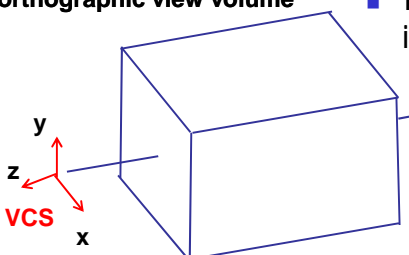





Projection Transformations




- Question: How to draw 3D object on 2D screen?
- If we ignore perspective (viewer at infinity)
 - Project transformed object along Z axis onto XY plane - and from there to screen (clipped)
 - Canonical **orthographic** projection:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- In practice “ignore” z axis – use x and y coordinates for screen coordinates

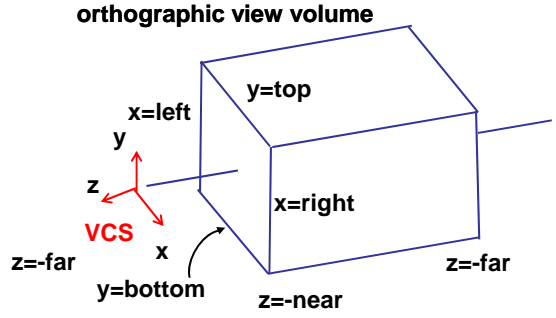





Clipping: View Volumes




- specifies field-of-view, used for clipping
- restricts domain of **z** stored for visibility test



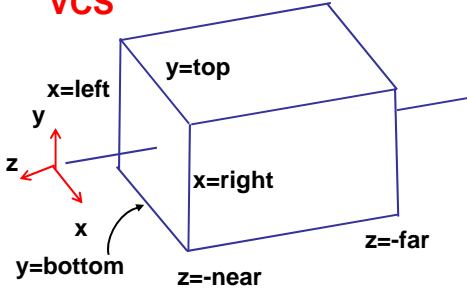


Understanding Z

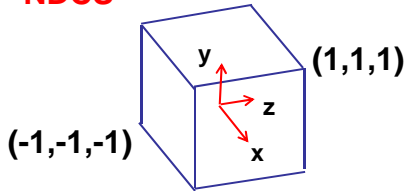



- z axis flip changes coord system handedness
 - RHS before projection (eye/view coords)
 - LHS after projection (clip, norm device coords)

VCS




NDCS







Understanding Z



- why near and far plane?
 - near plane:
 - avoid singularity for perspective projection (division by zero, or very small numbers)
 - far plane:
 - store depth in fixed-point representation (integer), thus have to have fixed range of values (0...1)
 - avoid/reduce numerical precision artifacts for distant objects



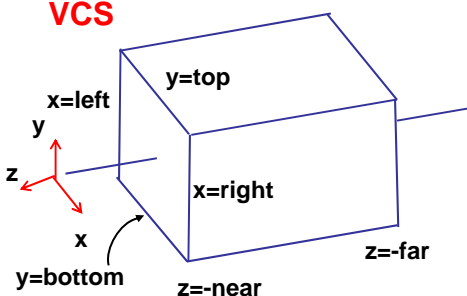
Orthographic Derivation



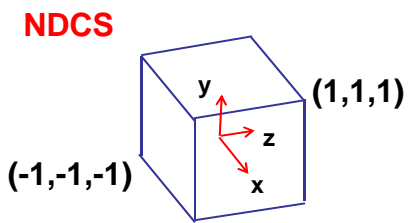
■ scale, translate, reflect for new coord sys


$$y' = a \cdot y + b \quad \begin{array}{l} y = \text{top} \rightarrow y' = 1 \\ y = \text{bot} \rightarrow y' = -1 \end{array}$$

VCS




NDCS






Orthographic Derivation




■ scale, translate, reflect for new coord sys


$$P' = \begin{bmatrix} \frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\ 0 & \frac{2}{\text{top} - \text{bot}} & 0 & -\frac{\text{top} + \text{bot}}{\text{top} - \text{bot}} \\ 0 & 0 & \frac{-2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix} P$$




Orthographic OpenGL



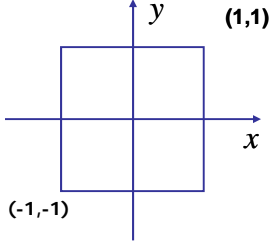
```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glOrtho(left, right, bot, top, near, far);
```



NDC to Viewport Transformation

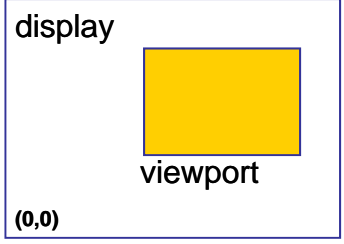


- generate pixel coordinates
 - map x, y from range $-1 \dots 1$ (NDC) to pixel coordinates on the display
 - involves 2D scaling and translation



(-1,-1) (1,1)

x y




display (w,h)

viewport


(0,0)

OpenGL


```
glViewport(x, y, a, b);
```




Origin Location



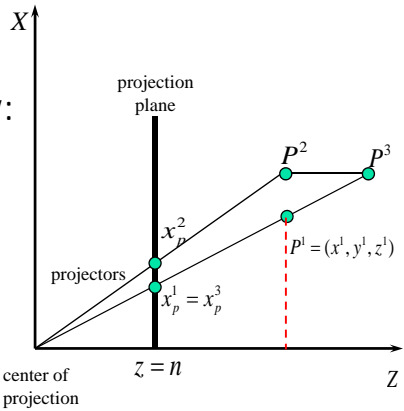
- yet more possibly confusing conventions
 - OpenGL: lower left
 - most window systems: upper left
- often have to flip your y coordinates
 - when interpreting mouse position




Perspective Projection




- Viewing is from *point at finite distance*
- Without loss of generality:
 - Viewpoint at origin
 - Viewing (near) plane is $z=n$
- Given $P=(x,y,z)$ triangle similarity gives:



$$\frac{x}{z} = \frac{x_p}{n} \text{ and } \frac{y}{z} = \frac{y_p}{n} \Rightarrow x_p = \frac{x}{z/n} \text{ and } y_p = \frac{y}{z/n}$$



Perspective Projection (cont'd)




- In matrix notation with homogeneous coordinates:

$$P(x, y, z, 1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/n & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/n \end{bmatrix}$$


- In Euclidean coordinates:

$$\left(\frac{x}{z/n}, \frac{y}{z/n}, \frac{z}{z/n} \right) = \left(\frac{x}{z/n}, \frac{y}{z/n}, n \right) = (x_p, y_p, n)$$


- P singular: $\det(P) = 0$

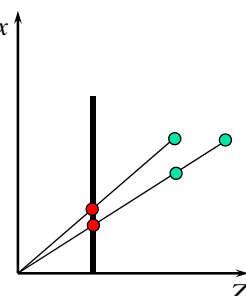
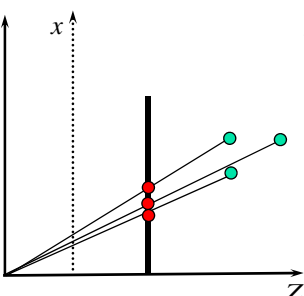
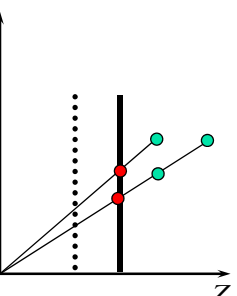



Perspective Projection (cont'd)




- What is (if any) is the difference between:
 - Moving projection plane
 - Moving viewpoint (center of projection)?


camtrans



How to make non-degenerate?




$$P(x, y, z, 1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/n & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/n \end{bmatrix}$$


$$\left(\frac{x}{z/n}, \frac{y}{z/n}, \frac{z}{z/n} \right) = \left(\frac{x}{z/n}, \frac{y}{z/n}, n \right) = (x_p, y_p, n).$$

- z' monotonically increasing function of z

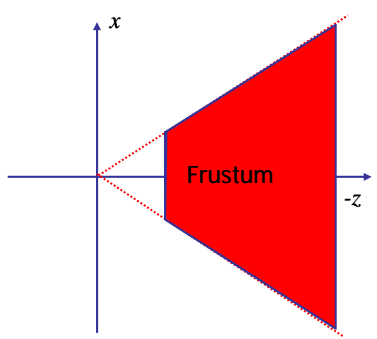
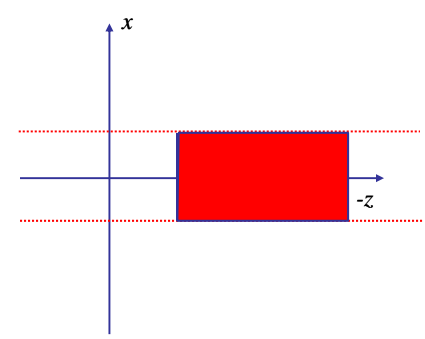
$$P(x, y, z, 1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & 1/n & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ az + b \\ z/n \end{bmatrix} \quad z' = na + \frac{b}{z}$$




Perspective Projection




- Have both near and far planes
 - Transformation well defined in-between
- Conversion to device coordinates
 - Warp view frustum to box




Introduce Far Plane




- Matrix formulation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & -f \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \frac{n+f}{n}z - f \\ z/n \end{bmatrix} \quad \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} \frac{x}{z/n} \\ \frac{y}{z/n} \\ (n+f) - \frac{fn}{z} \end{bmatrix}$$

- Preserves relative depth (third coordinate)



Alternative Formulation

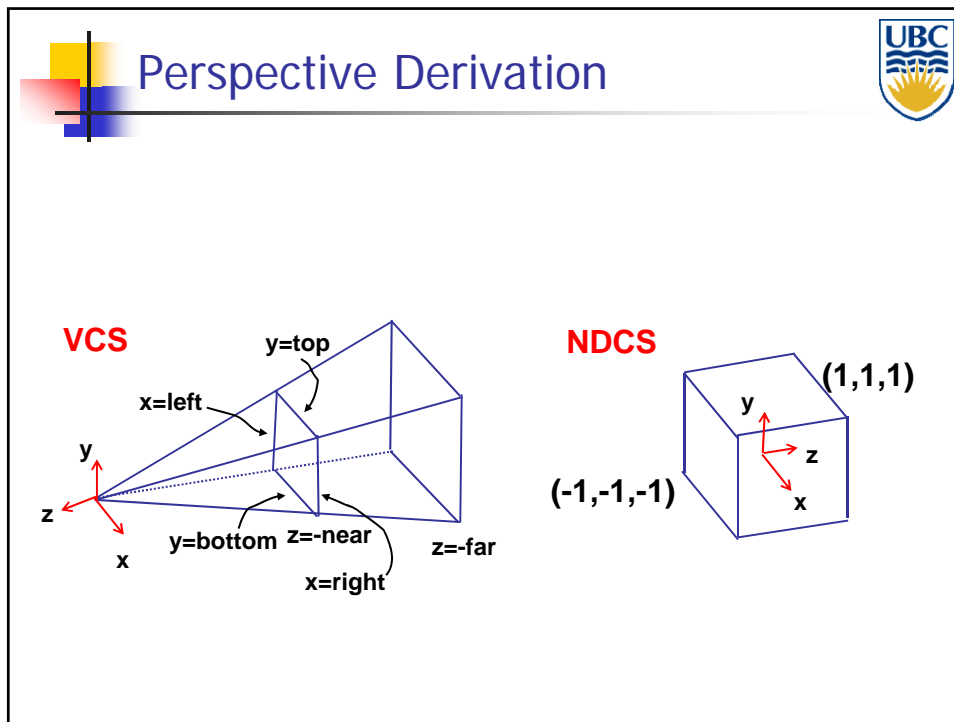


- Before

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & -f \\ 0 & 0 & \frac{1}{n} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \frac{n+f}{n}z - f \\ z/n \end{bmatrix} \quad \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} \frac{x}{z/n} \\ \frac{y}{z/n} \\ (n+f) - \frac{fn}{z} \end{bmatrix}$$

- After (cancel division by n)


$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ z \end{bmatrix} \quad \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ (n+f) - \frac{fn}{z} \end{bmatrix}$$




Perspective Derivation (full)

- Solve linear system to get A-F
- 6 planes, 6 unknowns

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} E & 0 & A & 0 \\ 0 & F & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



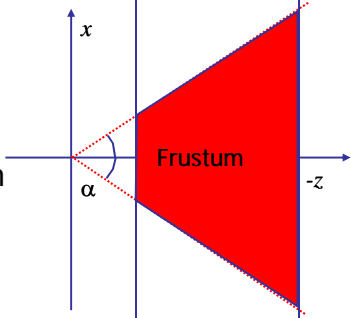
Perspective OpenGL




```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();


glFrustum(left, right, bot, top, near, far);
or
glPerspective(fov, aspect, near, far);
```

- Symmetric version using field-of-view angles
 - In x-direction (fov) α
 - In y-direction (fovy) given by aspect ratio





Another Transformations Quiz



- What does each transformation preserve?

	straight lines	parallel lines	distance	angles	normals	
scaling						
rotation						
translation						
perspective						