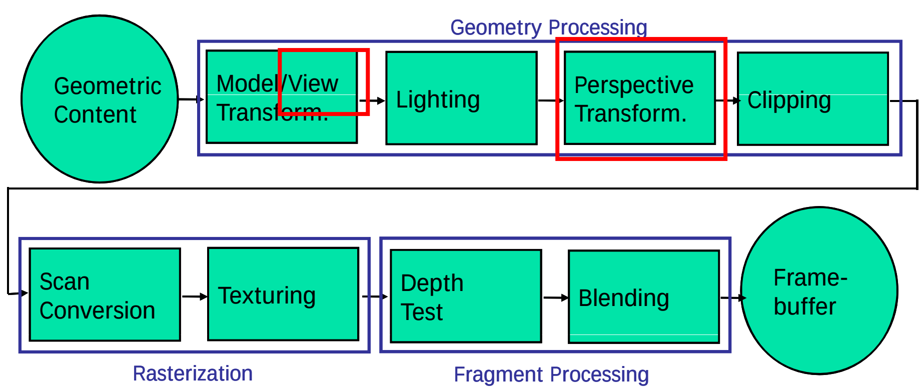
Chapter 5

Viewing/Perspective Transformations

Rendering Pipeline



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

Rendering Pipeline

Scene graph
Object geometry

Modelling Transforms

Viewing Transform

Projection Transform

Rendering Pipeline

Scene graph
Object geometry

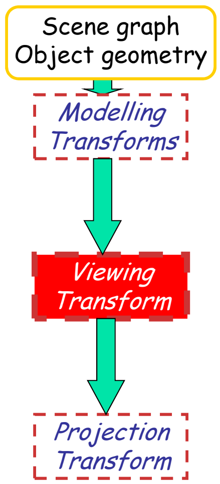
Modelling Transforms

Viewing Transform

Projection Transform

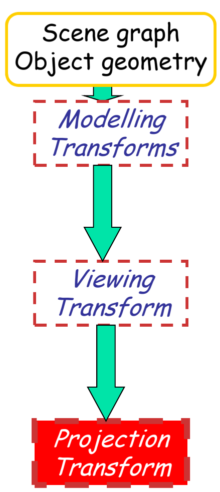
- result
  - all vertices of scene in shared 3D world coordinate system

Rendering Pipeline

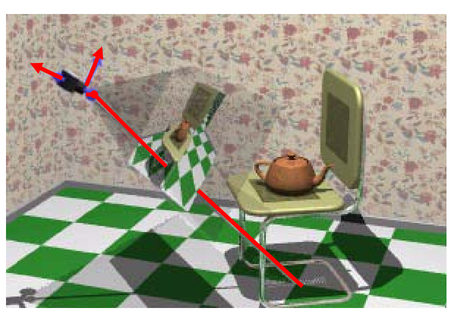


- result
  - scene vertices in 3D view (camera) coordinate system

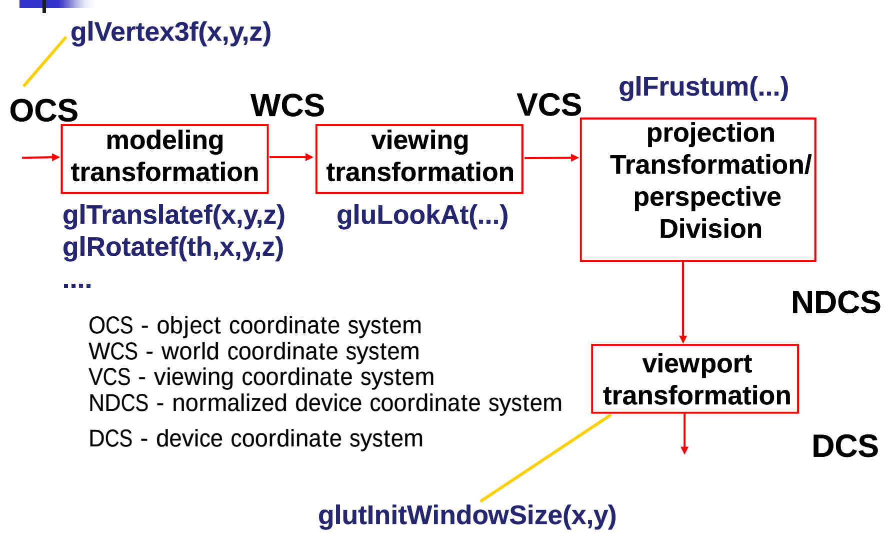
Rendering Pipeline



- result
  - 2D screen coordinates of clipped vertices



Projective Rendering Pipeline

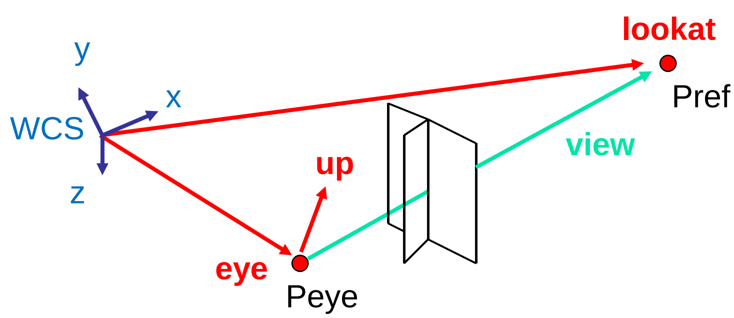


Basic Viewing

- Starting spot - OpenGL
  - camera at world origin
    - probably inside an object
  - y axis is up
  - looking down negative z axis
    - why? RHS with x horizontal, y vertical, z out of screen
- To position – coordinate frame change
- Intuitive description
  - eye point, gaze/lookat direction, up vector

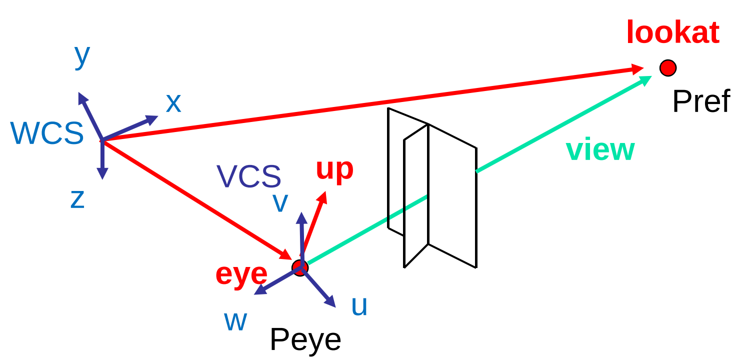
Camera Description/Motion

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



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



Deriving W2V Transformation

- M=RT

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

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

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

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

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


camtrans

Notations/derivation from the board in class

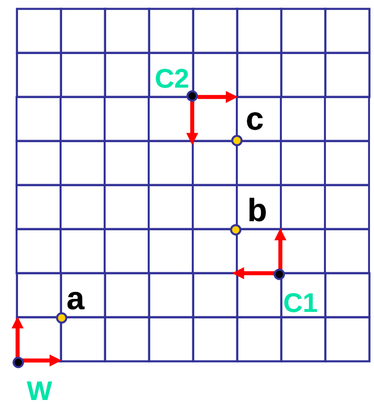
OpenGL Viewing Transformation

```
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
```

World vs. Camera Coordinates

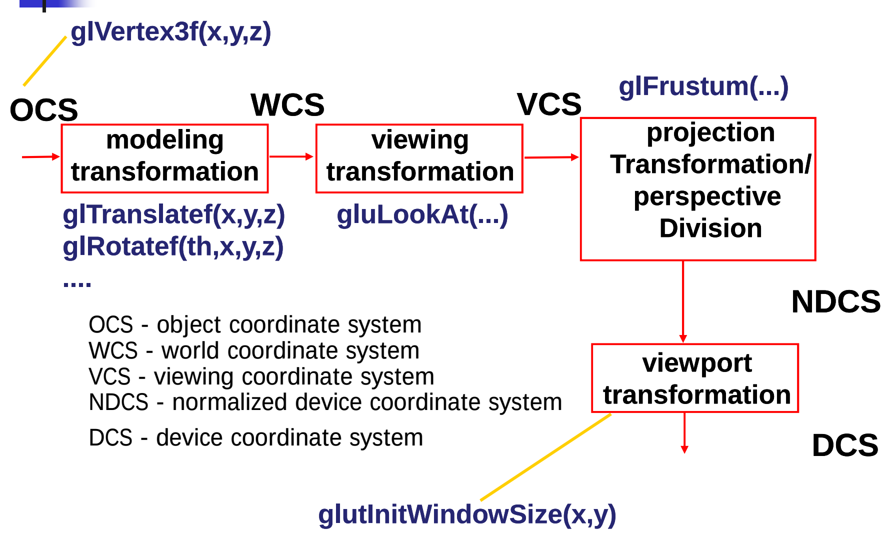


$a = (1,1)_W$

$b = (1,1)_{C1} = (5,3)_W$

$c = (1,1)_{C2} = (1,3)_{C1} = (5,5)_W$

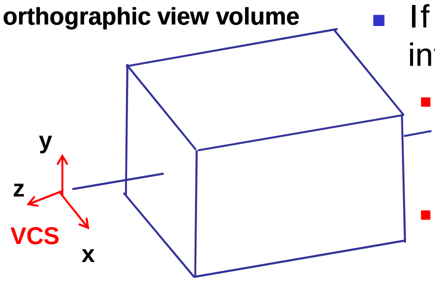
Projective Rendering Pipeline



OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

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:

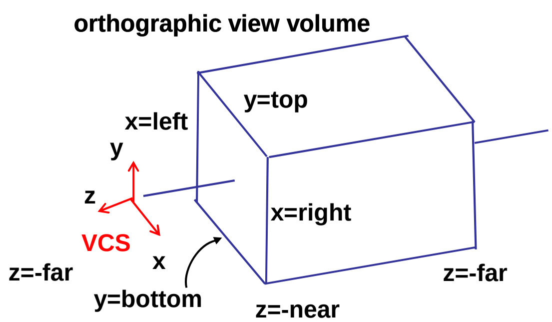


$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- In practice "ignore" *z* axis – use *x* and *y* coordinates for screen coordinates

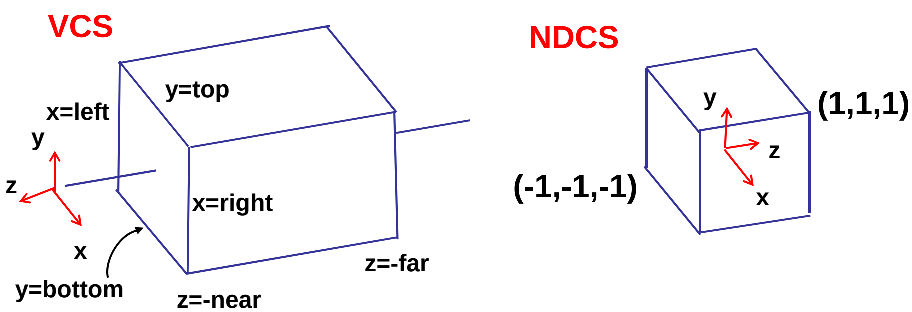
Clipping: View Volumes

- 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)



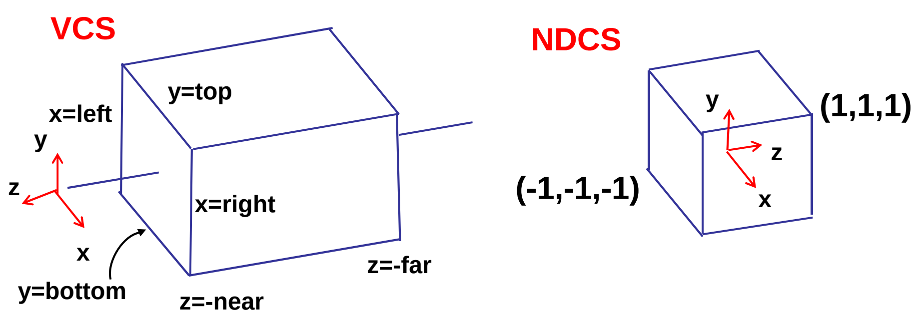
Understanding Z

- 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

$$y' = a \cdot y + b \qquad \begin{aligned} y = top &\rightarrow y' = 1 \\ y = bot &\rightarrow y' = -1 \end{aligned}$$



Orthographic Derivation

- scale, translate, reflect for new coord sys

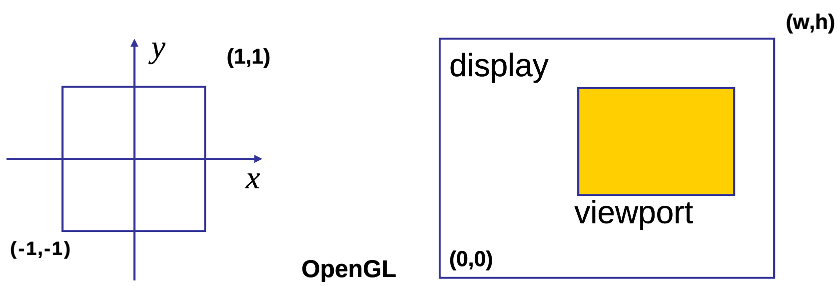
$$P' = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bot} & 0 & -\frac{top+bot}{top-bot} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix} P$$

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...1 (NDC) to pixel coordinates on the display
  - involves 2D scaling and translation



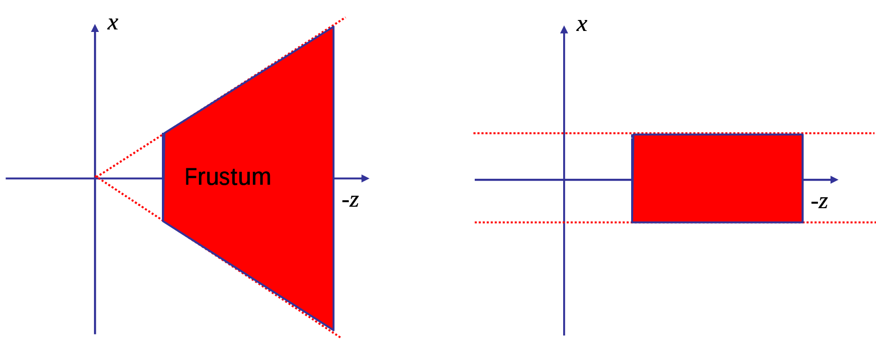

```
glViewport(x,y,a,b);
```

Origin Location

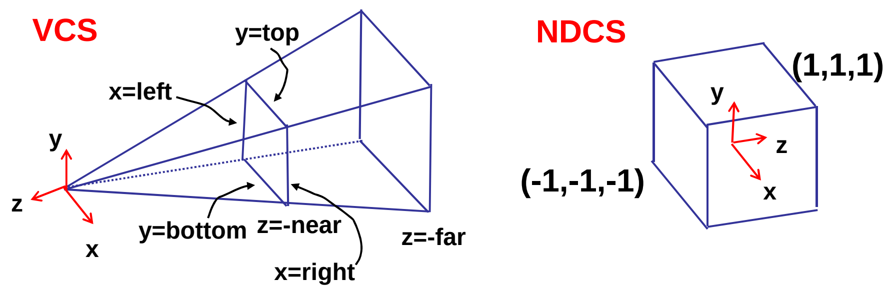
- yet more possibly confusing conventions
  - OpenGL: lower left
  - most window systems: upper left
- often have to flip your y coordinates
  - when interpreting mouse position

Perspective Projection

- Viewing is from point at finite distance (origin)
  - View volume is a frustum not a box
- Conversion to device coordinates
  - Warp view frustum to box

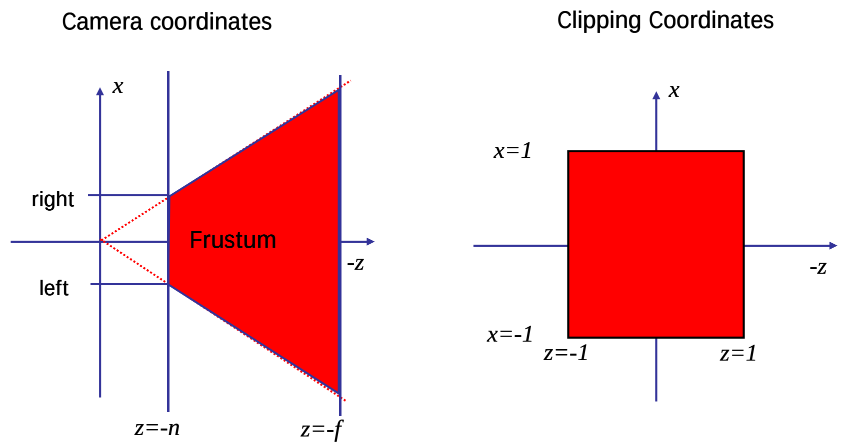


Perspective Derivation



Projective Transformations

- OpenGL Convention



Perspective Derivation

Basic (derived in class)

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, (d = -1)$$

complete: shear, scale, projection-normalization

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} E & 0 & A & 0 \\ 0 & F & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Derivation

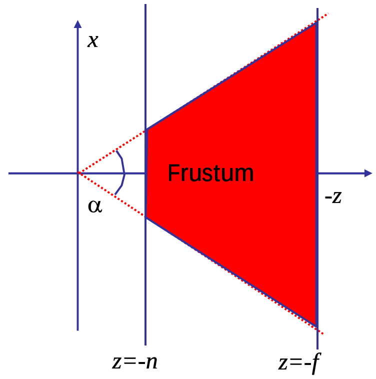
- Solve linear system to get A-F
  - 6 planes, 6 unknowns

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} E & 0 & A & 0 \\ 0 & F & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Projective Transformations

- Alternative specification of symmetric frusta
  - Field-of-view angles
    - In x-direction (fov) α
    - In y-direction (fovy) given by aspect ratio



Perspective OpenGL

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

glFrustum(left,right,bot,top,near,far);
```

or

```
glPerspective(fovy,aspect,near,far);
```

- symmetric version

Another Transformations Quiz

- What does each transformation preserve?

	lines	parallel lines	distance	angles	normals	convexity
scaling						
rotation						
translation						
shear						
perspective						