University of British Columbia
CPSC 314 Computer Graphics
May-June 2005

Tamara Munzner

Viewing, Projections I/II

Week 2, Tue May 17

http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005

News

- extra lab coverage with TAs
  - 12-2 Mondays, Wednesdays
  - for rest of term
  - just for answering questions, no presentations

Reading: Today

- FCG Chapter 6
- FCG Section 5.3.1
- RB rest of Chap **Viewing**
- RB rest of App **Homogeneous Coords**

Reading: Next Time

- FCG Section 2.11
- FCG Chap 3
  - except 3.8
- FCG Chap 17 Human Vision (pp 293-298)
- FCG Chap 18 Color pp 301-311
  - until Section 18.9 Tone Mapping

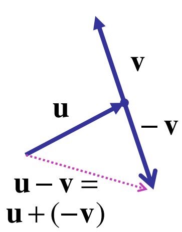
Textbook Errata

- list at http://www.cs.utah.edu/~shirley/fcg/errata
  - p 113
    - last matrix, last column denominators
      - D-a -> A-a
      - E-b -> B-b
      - F-c -> C-c
  - p 120
    - "Sometimes we will want to take the inverse of P" should be "M_p" instead of "P"

Correction²: Vector-Vector Subtraction

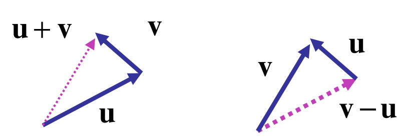
- subtract: vector - vector = vector

$$\mathbf{u}-\mathbf{v}=\begin{bmatrix} u_1-v_1 \\ u_2-v_2 \\ u_3-v_3 \end{bmatrix}$$

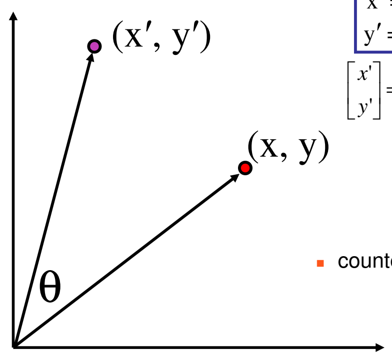


$$(3,2)-(6,4)=(-3,-2)$$
$$(2,5,1)-(3,1,-1)=(-1,4,2)$$

argument reversal



Review: 2D Rotation

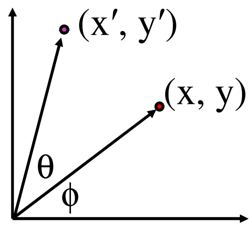


$x' = x\ \mathbf{cos}(\theta) - y\ \mathbf{sin}(\theta)$
$y' = x\ \mathbf{sin}(\theta) + y\ \mathbf{cos}(\theta)$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- counterclockwise, RHS

Review: 2D Rotation From Trig Identities

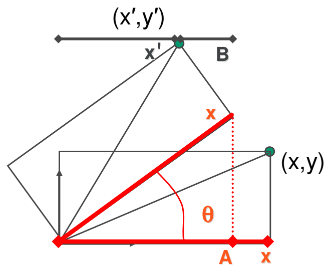


$x = r\cos(\phi)$
$y = r\sin(\phi)$
$x' = r\cos(\phi + \theta)$
$y' = r\sin(\phi + \theta)$

Trig Identity…
$x' = r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$
$y' = r\sin(\phi)\cos(\theta) + r\cos(\phi)\sin(\theta)$

Substitute…
$x' = x\ \mathbf{cos}(\theta) - y\ \mathbf{sin}(\theta)$
$y' = x\ \mathbf{sin}(\theta) + y\ \mathbf{cos}(\theta)$

Review: 2D Rotation: Another Derivation

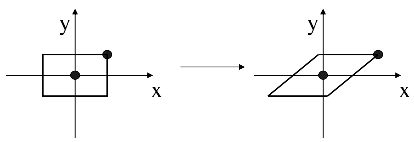


$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$x' = A - B$$
$$A = x\cos\theta$$

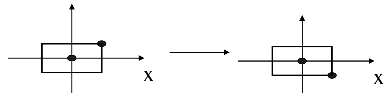
Review: Shear, Reflection

- shear along x axis
  - push points to right in proportion to height



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- reflect across x axis
  - mirror



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Review: 2D Transformations

matrix multiplication

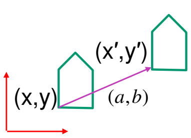
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

scaling matrix

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

rotation matrix



vector addition

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

translation multiplication matrix??

Review: Linear Transformations

- linear transformations are combinations of
  - shear
  - scale
  - rotate
  - reflect

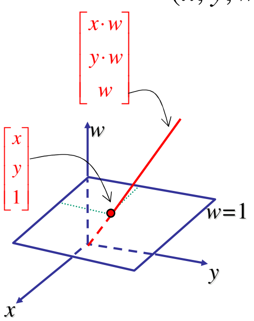
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = ax + by$$
$$y' = cx + dy$$

- properties of linear transformations
  - satisifes T(s**x**+t**y**) = s T(**x**) + t T(**y**)
  - origin maps to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

Review: Homogeneous Coordinates Geometrically

homogeneous (x, y, w) $\xrightarrow{/w}$ **cartesian** $(\frac{x}{w}, \frac{y}{w})$



- point in 2D cartesian + weight w = point P in 3D homog. coords
- multiples of (x,y,w)
  - all homogeneous points on 3D line L represent same 2D cartesian point
  - homogenize to convert homog. 3D point to cartesian 2D point
    - divide by w to get (x/w, y/w, 1)
- w=0 is direction; (0,0,0) is undefined

Review: 3D Homog Transformations

- use 4x4 matrices for 3D transformations

translate(a,b,c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & & & a \\ & 1 & & b \\ & & 1 & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

scale(a,b,c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$\text{Rotate}(x, \theta)$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & \cos\theta & -\sin\theta & \\ & \sin\theta & \cos\theta & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$\text{Rotate}(y, \theta)$

$$\begin{bmatrix} \cos\theta & & \sin\theta & \\ & 1 & & \\ -\sin\theta & & \cos\theta & \\ & & & 1 \end{bmatrix}$$

$\text{Rotate}(z, \theta)$

$$\begin{bmatrix} \cos\theta & -\sin\theta & & \\ \sin\theta & \cos\theta & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

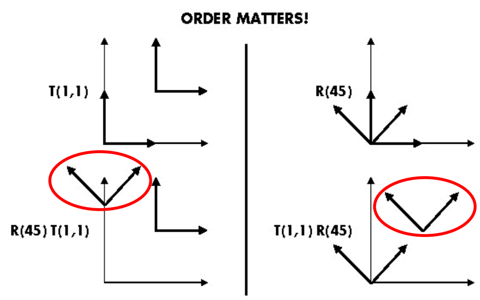
Review: Affine Transformations

- affine transforms are combinations of
  - linear transformations
  - translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- properties of affine transformations
  - origin does not necessarily map to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

Review: Composing Transformations



Ta Tb = Tb Ta, but Ra Rb != Rb Ra and Ta Rb != Rb Ta

Review: Composing Transforms

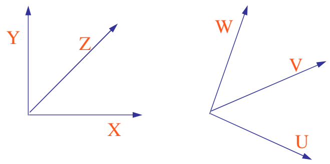
- order matters
  - 4x4 matrix multiplication not commutative!
- moving to origin
  - transformation of geometry into coordinate system where operation becomes simpler
  - perform operation
  - transform geometry back to original coordinate system

Review: Composing Transformations

$$\mathbf{p'} = \mathbf{TRp}$$

- which direction to read?
  - right to left
    - interpret operations wrt fixed coordinates
    - moving object
  - left to right **OpenGL pipeline ordering!**
    - interpret operations wrt local coordinates
    - changing coordinate system
    - OpenGL updates current matrix with postmultiply
      - glTranslatef(2,3,0);
      - glRotatef(-90,0,0,1);
      - glVertexf(1,1,1);
    - specify vector last, in final coordinate system
    - first matrix to affect it is specified second-to-last

Review: Arbitrary Rotation



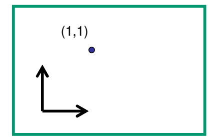
- problem:
  - given two orthonormal coordinate systems *XYZ* and *UVW*
  - find transformation from one to the other
- answer:
  - transformation matrix R whose columns are *U,V,W:*

$$R = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}$$

Review: Interpreting Transformations

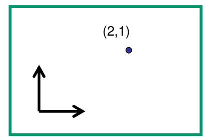
$$\mathbf{p}' = \mathbf{TRp}$$

right to left: moving object

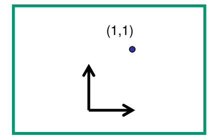


intuitive?

translate by (-1,0)



left to right: changing coordinate system

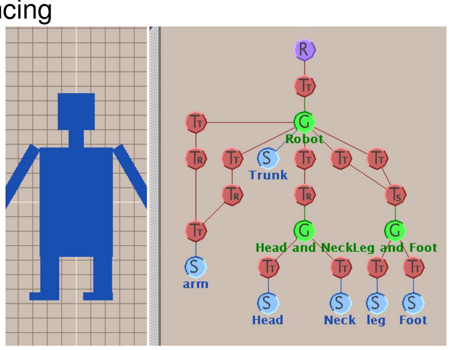


OpenGL

- same relative position between object and basis vectors

Review: Transformation Hierarchies

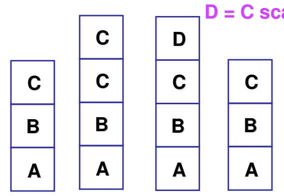
- transforms apply to graph nodes beneath them
- design structure so that object doesn't fall apart
- instancing



Review: Matrix Stacks

- OpenGL matrix calls postmultiply matrix M onto current matrix P, overwrite it to be PM
  - or can save intermediate states with stack
  - no need to compute inverse matrices all the time
  - modularize changes to pipeline state
  - avoids accumulation of numerical errors

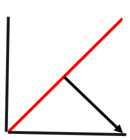
D = C scale(2,2,2) trans(1,0,0)

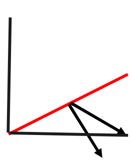


DrawSquare()
glPushMatrix()
glScale3f(2,2,2)
glTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()

Review: Transforming Normals

- shear, nonuniform scale makes normal nonperpendicular
  - need to use inverse transpose matrix instead





Review: Display Lists

- precompile/cache block of OpenGL code for reuse
  - efficiency
    - exact optimizations depend on driver
  - multiple instances of same object
  - static objects redrawn often
  - exploit hierarchical structure when possible
- set up list once with glNewList/glEndList
  - call multiple times

Viewing

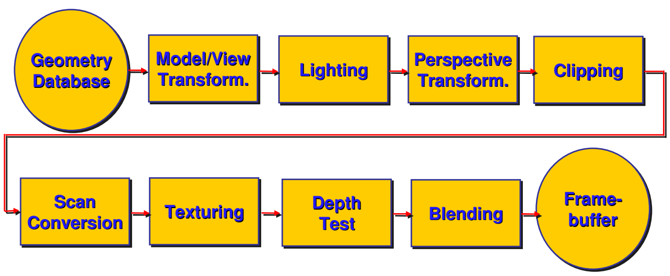
Using Transformations

- three ways
  - modelling transforms
    - place objects within scene (shared world)
  - viewing transforms
    - place camera
  - projection transforms
    - change type of camera

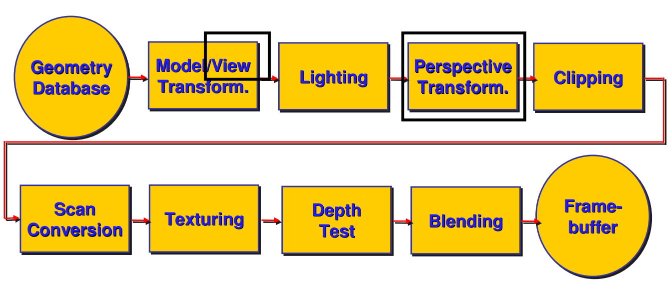
Viewing and Projection

- need to get from 3D world to 2D image
- projection: geometric abstraction
  - what eyes or cameras do
- two pieces
  - viewing transform:
    - where is the camera, what is it pointing at?
  - perspective transform: 3D to 2D
    - flatten to image

Rendering Pipeline

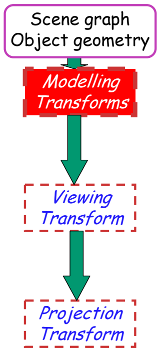


Rendering Pipeline

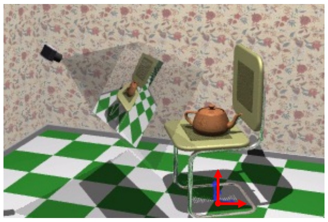


Rendering Pipeline

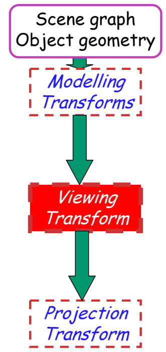
Rendering Pipeline



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

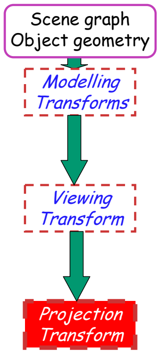


Rendering Pipeline

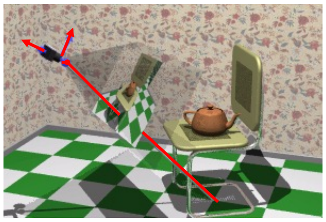


- result
  - scene vertices in 3D view (camera) coordinate system

Rendering Pipeline



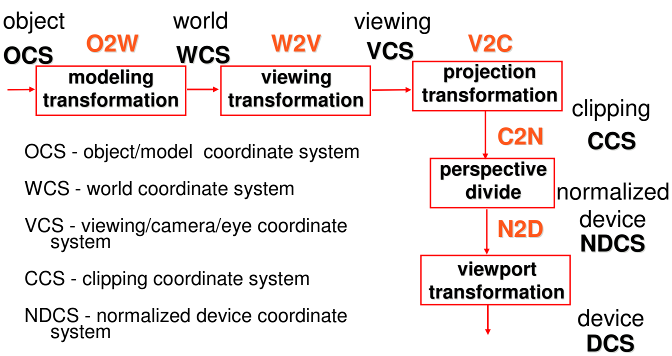
- result
  - 2D screen coordinates of clipped vertices



Coordinate Systems

- result of a transformation
- names
  - convenience
    - giraffe: neck, head, tail
  - standard conventions in graphics pipeline
    - object/modelling
    - world
    - camera/viewing/eye
    - screen/window
    - raster/device

Projective Rendering Pipeline



OCS - object/model coordinate system

WCS - world coordinate system

VCS - viewing/camera/eye coordinate system

CCS - clipping coordinate system

NDCS - normalized device coordinate system

DCS - device/display/screen coordinate system

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
- translate backward so scene is visible
  - move distance d = focal length
- can use rotate/translate/scale to move camera
  - demo: Nate Robins tutorial *transformations*

Viewing in Project 1

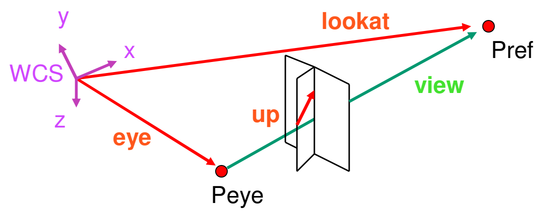
- where is camera in template code?
  - 5 units back, looking down -z axis

Convenient Camera Motion

- rotate/translate/scale not intuitive
- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector

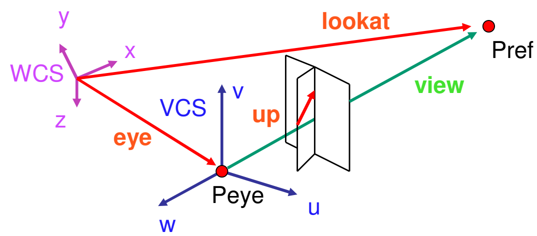
Convenient Camera Motion

- rotate/translate/scale not intuitive
- 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



OpenGL Viewing Transformation

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

- postmultiplies current matrix, so to be safe:

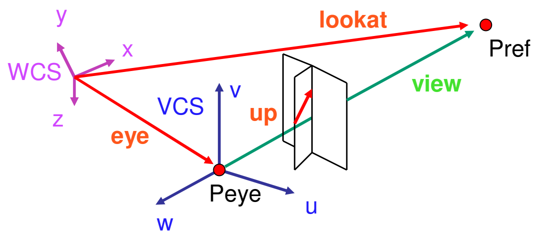
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(ex,ey,ez,lx,ly,lz,ux,uy,uz)
// now ok to do model transformations
```

- demo: Nate Robins tutorial *projection*

Deriving W2V Transformation

- translate **eye** to origin

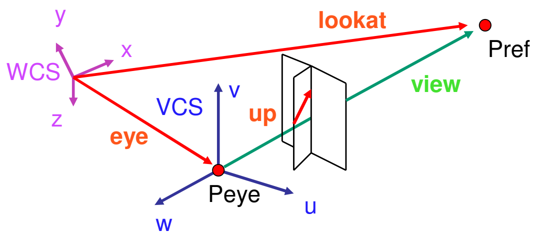
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Deriving W2V Transformation

- rotate **view** vector (**lookat** – **eye**) to **w** axis
  - **w** is just opposite of **view/gaze** vector **g**

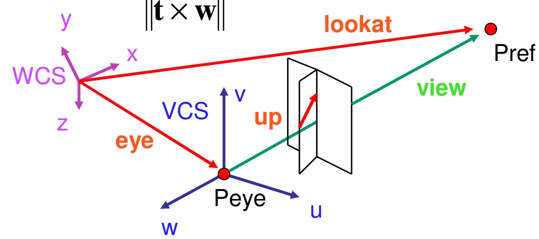
$$\mathbf{w} = -\hat{\mathbf{g}} = -\frac{\mathbf{g}}{\|\mathbf{g}\|}$$



Deriving W2V Transformation

- rotate around **w** to bring **up** into **vw**-plane
  - **u** should be perpendicular to **vw**-plane, thus perpendicular to **w** and **up** vector **t**
  - **v** should be perpendicular to **u** and **w**

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



Deriving W2V Transformation

- rotate from WCS **xyz** into **uvw** coordinate system with matrix that has rows **u**, **v**, **w**

$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{\|\mathbf{t} \times \mathbf{w}\|} \qquad \mathbf{v} = \mathbf{w} \times \mathbf{u} \qquad \mathbf{w} = -\hat{\mathbf{g}} = -\frac{\mathbf{g}}{\|\mathbf{g}\|}$$

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- reminder: rotate from **uvw** to **xyz** coord sys with matrix **M** that has columns **u,v,w**
  - rotate from **xyz** coord sys to **uvw** coord sys with matrix $\mathbf{M}^T$ that has rows **u**,**v**,**w**

Deriving W2V Transformation

- M=RT

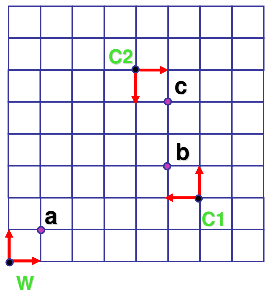
$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Moving the Camera or the World?

- two equivalent operations
  - move camera one way vs. move world other way
- example
  - initial OpenGL camera: at origin, looking along -z axis
  - create a unit square parallel to camera at z = -10
  - translate in z by 3 possible in two ways
    - camera moves to z = -3
      - Note OpenGL models viewing in left-hand coordinates
    - camera stays put, but square moves to -7
  - resulting image same either way
    - possible difference: are lights specified in world or view coordinates?

World vs. Camera Coordinates



a = $(1,1)_W$

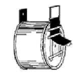
b = $(1,1)_{C1}$ = $(3,2)_W$

c = $(1,1)_{C2}$ = $(1,3)_{C1}$ = $(4,4)_W$

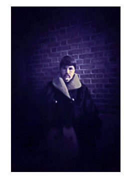
Projections I

Pinhole Camera

- ingredients
  - box
  - film
  - hole punch
- results
  - pictures!



www.kodak.com

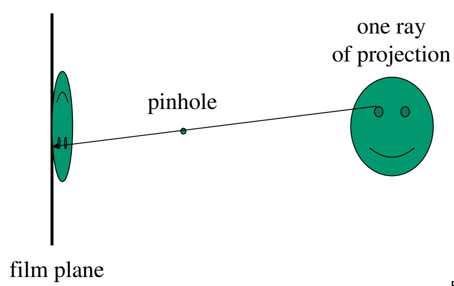


www.pinhole.org

www.debevec.org/Pinhole

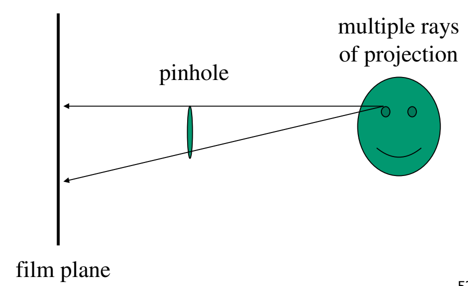
Pinhole Camera

- theoretical perfect pinhole



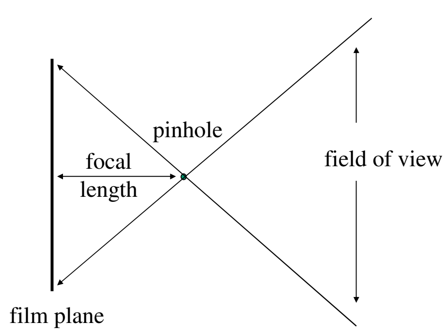
Pinhole Camera

- non-zero sized hole



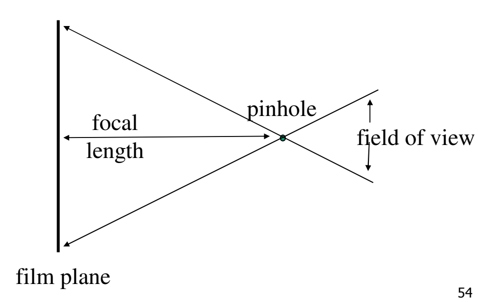
Pinhole Camera

- field of view and focal length



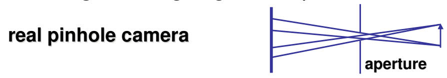
Pinhole Camera

- field of view and focal length



Real Cameras

- pinhole camera has small aperture (lens opening)
  - hard to get enough light to expose the film

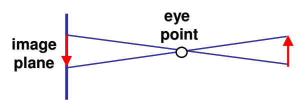


- lens permits larger apertures
- lens permits changing distance to film plane without actually moving the film plane

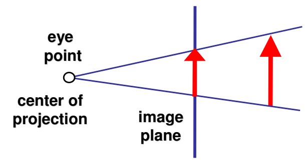
price to pay: limited depth of field

Graphics Cameras

- real pinhole camera: image inverted

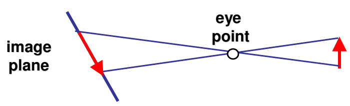


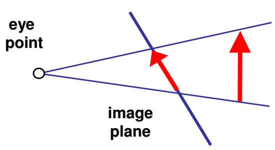
- computer graphics camera: convenient equivalent



General Projection

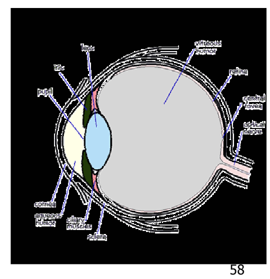
- image plane need not be perpendicular to view plane





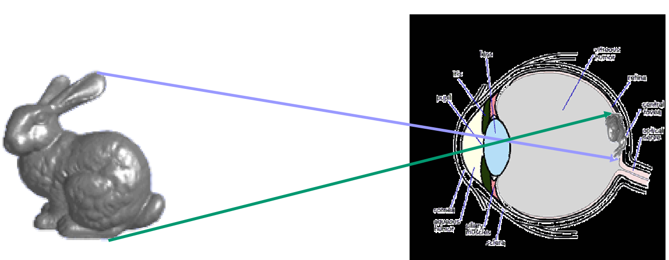
Perspective Projection

- our camera must model perspective



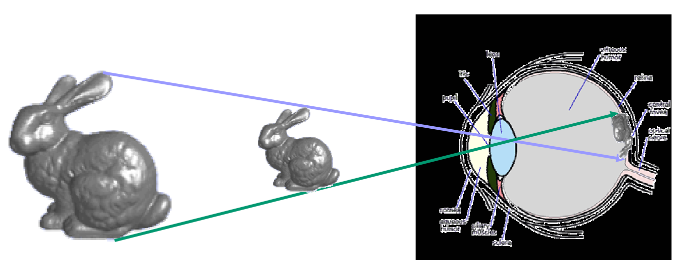
Perspective Projection

- our camera must model perspective



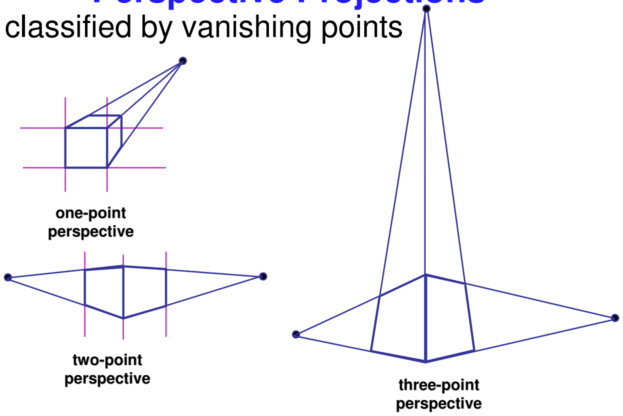
Perspective Projection

- our camera must model perspective



Perspective Projections

- classified by vanishing points



Projective Transformations

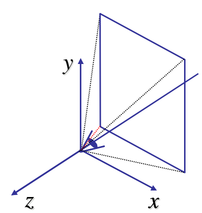
- planar geometric projections
  - planar: onto a plane
  - geometric: using straight lines
  - projections: 3D -> 2D
- aka projective mappings
- counterexamples?

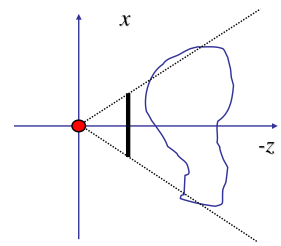
Projective Transformations

- properties
  - lines mapped to lines and triangles to triangles
  - parallel lines do NOT remain parallel
    - e.g. rails vanishing at infinity
  - affine combinations are NOT preserved
    - e.g. center of a line does not map to center of projected line (perspective foreshortening)

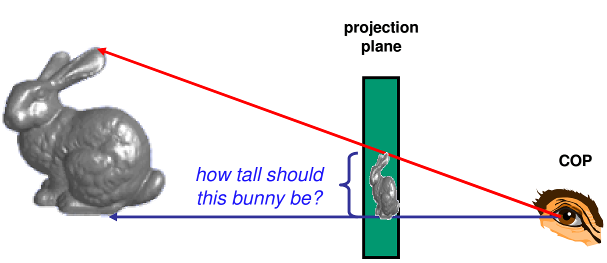
Perspective Projection

- project all geometry
  - through common center of projection (eye point)
  - onto an image plane



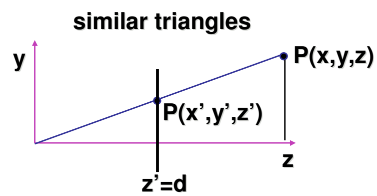


Perspective Projection



Basic Perspective Projection

similar triangles



$$\frac{y'}{d} = \frac{y}{z} \rightarrow y' = \frac{y \cdot d}{z}$$

$$\frac{x'}{d} = \frac{x}{z} \rightarrow x' = \frac{x \cdot d}{z}$$

but $z' = d$

- nonuniform foreshortening
  - not affine

Perspective Projection

- desired result for a point $[x, y, z, 1]^T$ projected onto the view plane:

$$\frac{x'}{d} = \frac{x}{z}, \quad \frac{y'}{d} = \frac{y}{z}$$

$$x' = \frac{x \cdot d}{z} = \frac{x}{z/d}, \quad y' = \frac{y \cdot d}{z} = \frac{y}{z/d}, \quad z = d$$

- what could a matrix look like to do this?

Simple Perspective Projection Matrix

$$\begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \end{bmatrix}$$

Simple Perspective Projection Matrix

$$\begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \end{bmatrix}$$ is homogenized version of $$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

where w = z/d

Simple Perspective Projection Matrix

$$\begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \end{bmatrix}$$ is homogenized version of $$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

where w = z/d

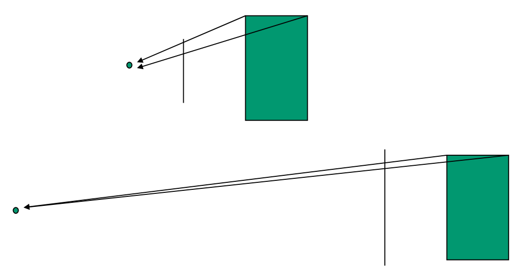
$$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Projection

- expressible with 4x4 homogeneous matrix
  - use previously untouched bottom row
- perspective projection is irreversible
  - many 3D points can be mapped to same (x, y, d) on the projection plane
  - no way to retrieve the unique z values

Moving COP to Infinity

- as COP moves away, lines approach parallel
- when COP at infinity, orthographic view



Orthographic Camera Projection

- camera's back plane parallel to lens
- infinite focal length
- no perspective convergence

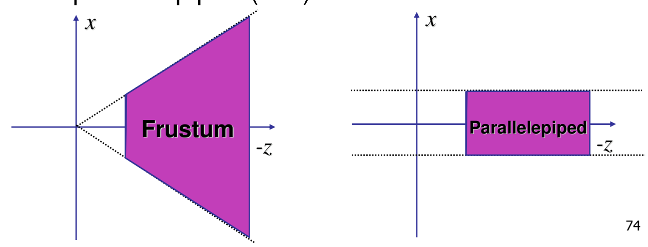
$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

- just throw away z values

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

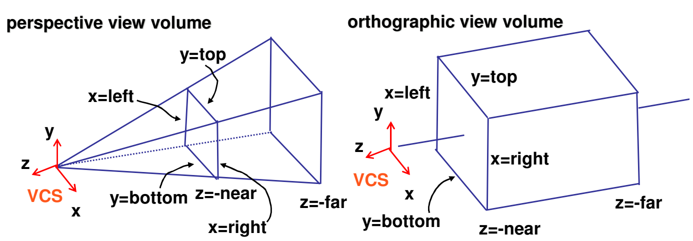
Perspective to Orthographic

- transformation of space
  - center of projection moves to infinity
  - view volume transformed
    - from frustum (truncated pyramid) to parallelepiped (box)



View Volumes

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

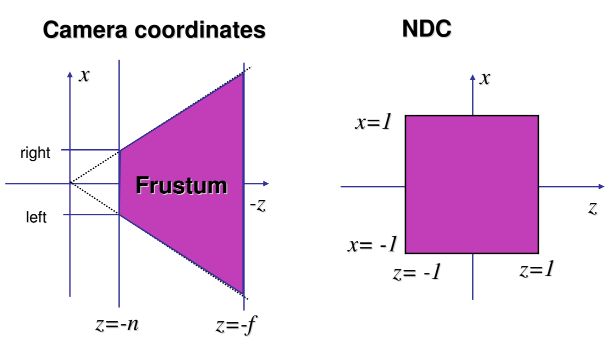


View Volume

- convention
  - viewing frustum mapped to specific parallelepiped
    - Normalized Device Coordinates (NDC)
    - same as clipping coords
  - only objects inside the parallelepiped get rendered
  - which parallelepiped?
    - depends on rendering system

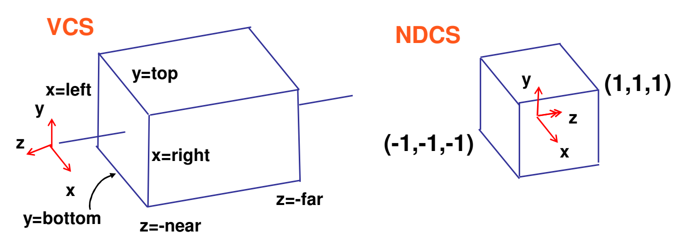
Normalized Device Coordinates

left/right *x* =+/- 1, top/bottom *y* =+/- 1, near/far *z* =+/- 1



Understanding Z

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

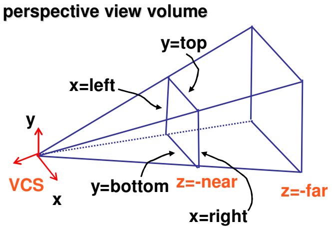


Understanding Z

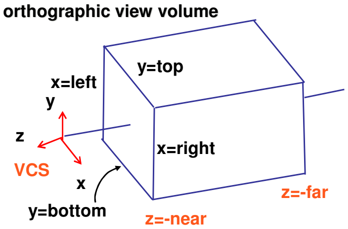
near, far always positive in OpenGL calls

glOrtho(left,right,bot,top,near,far);
glFrustum(left,right,bot,top,near,far);
glPerspective(fovy,aspect,near,far);

perspective view volume



orthographic view volume

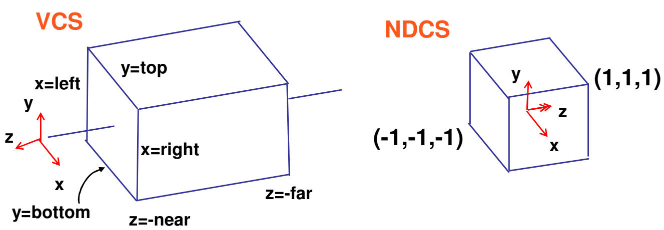


Understanding Z

- why near and far plane?
  - near plane:
    - avoid singularity (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

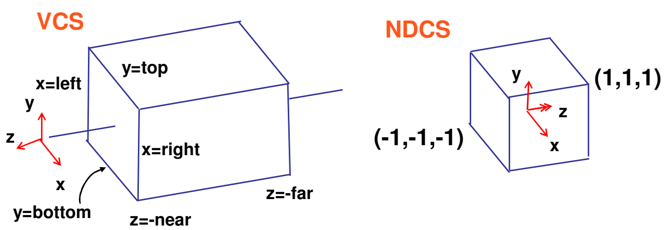
- scale, translate, reflect for new coord sys



Orthographic Derivation

- scale, translate, reflect for new coord sys

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



Orthographic Derivation

- scale, translate, reflect for new coord sys

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

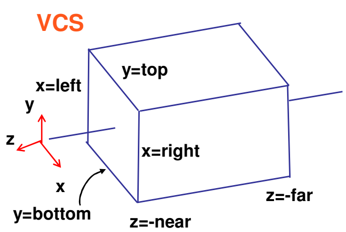
$$b = 1 - a \cdot top, b = -1 - a \cdot bot$$
$$1 - a \cdot top = -1 - a \cdot bot$$
$$1 - (-1) = -a \cdot bot - (-a \cdot top)$$
$$2 = a(-bot + top)$$
$$a = \frac{2}{top - bot}$$

$$1 = \frac{2}{top - bot} top + b$$
$$b = 1 - \frac{2 \cdot top}{top - bot}$$
$$b = \frac{(top - bot) - 2 \cdot top}{top - bot}$$
$$b = \frac{-top - bot}{top - bot}$$

Orthographic Derivation

- scale, translate, reflect for new coord sys

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



$$a = \frac{2}{top - bot}$$

$$b = -\frac{top + bot}{top - bot}$$

same idea for right/left, far/near

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 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 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 Derivation

- scale, translate, reflect for new coord sys

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

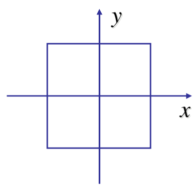
Orthographic OpenGL

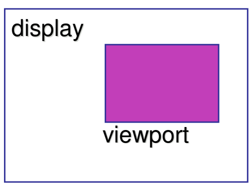
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(left,right,bot,top,near,far);
```


Projections II

NDC to Viewport Transformation

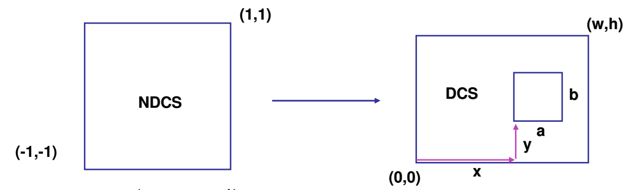
- generate pixel coordinates
  - map x, y from range –1…1 (NDC) to pixel coordinates on the display
  - involves 2D scaling and translation





NDC to Viewport Transformation

- 2D scaling and translation



$$x_{DCS} = w\frac{(x_{NDCS}+1)}{2}$$

$$y_{DCS} = h\frac{(y_{NDCS}+1)}{2}$$

$$z_{DCS} = \frac{(z_{NDCS}+1)}{2}$$

OpenGL

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

default:

```
glViewport(0,0,w,h);
```

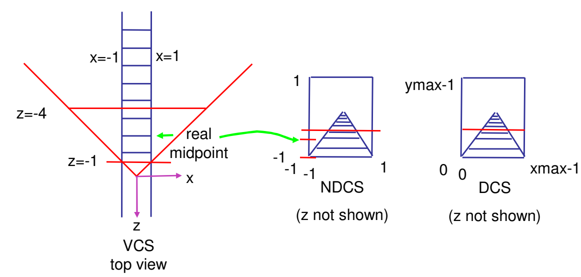

Origin Location

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

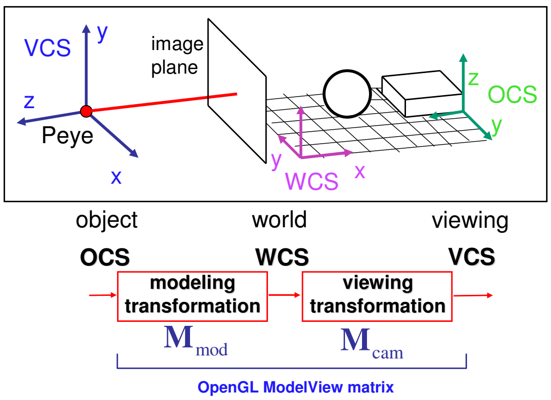
Perspective Example

tracks in VCS:
left x=-1, y=-1
right x=1, y=-1

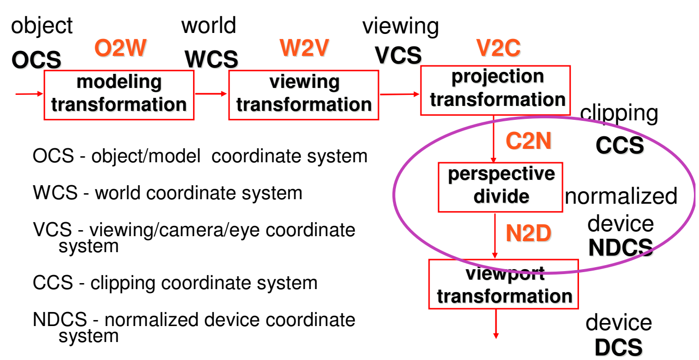
view volume
left = -1, right = 1
bot = -1, top = 1
near = 1, far = 4



Viewing Transformation



Projective Rendering Pipeline



OCS - object/model coordinate system

WCS - world coordinate system

VCS - viewing/camera/eye coordinate system

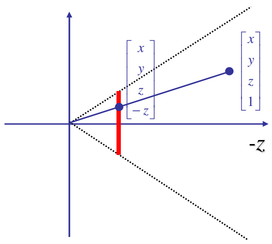
CCS - clipping coordinate system

NDCS - normalized device coordinate system

DCS - device/display/screen coordinate system

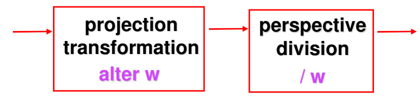
Perspective Projection

- specific example
  - assume image plane at $z = -1$
  - a point $[x,y,z,1]^T$ projects to $[-x/z,-y/z,-z/z,1]^T \equiv [x,y,z,-z]^T$



Perspective Projection

$$T\left(\begin{bmatrix}x\\y\\z\\1\end{bmatrix}\right)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&-1&0\end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x\\y\\z\\-z\end{bmatrix}\equiv\begin{bmatrix}-x/z\\-y/z\\-1\\1\end{bmatrix}$$

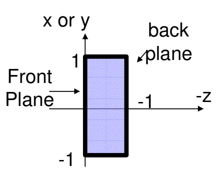


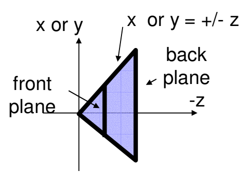
Canonical View Volumes

- standardized viewing volume representation

orthographic
orthogonal
parallel

perspective



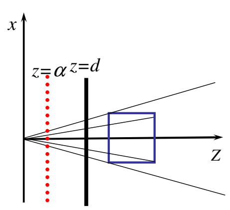


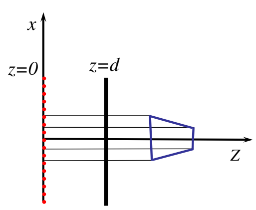
Why Canonical View Volumes?

- permits standardization
  - clipping
    - easier to determine if an arbitrary point is enclosed in volume
    - consider clipping to six arbitrary planes of a viewing volume versus canonical view volume
  - rendering
    - projection and rasterization algorithms can be reused

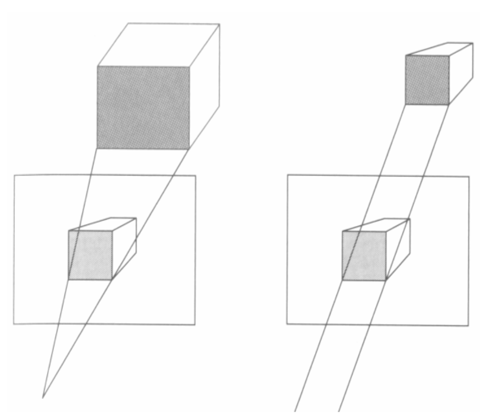
Projection Normalization

- one additional step of standardization
  - warp perspective view volume to orthogonal view volume
    - render all scenes with orthographic projection!



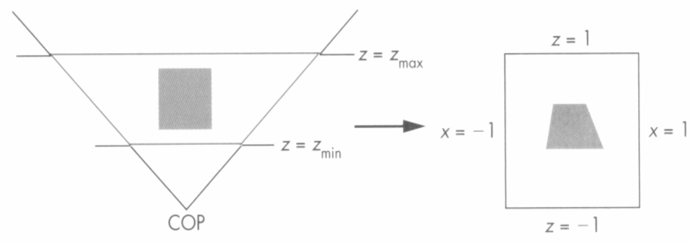


Predistortion



Perspective Normalization

- perspective viewing frustum transformed to cube
- orthographic rendering of cube produces same image as perspective rendering of original



Demos

- Tuebingen applets from Frank Hanisch
  - http://www.gris.uni-tuebingen.de/projects/grdev/doc/html/etc/AppletIndex.html#Transformationen

Perspective Warp

- matrix formulation

$$(x,y,z,1)\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&\frac{d}{d-\alpha}&\frac{1}{d}\\0&0&\frac{-\alpha\cdot d}{d-\alpha}&0\end{bmatrix}=\left(x,y,\frac{(z-\alpha)\cdot d}{d-\alpha},\frac{z}{d}\right)$$

$$(x_p,y_p,z_p)=\left(\frac{x}{z/d},\frac{y}{z/d},\frac{d^2}{d-\alpha}\left(1-\frac{\alpha}{z}\right)\right)$$

- preserves relative depth (third coordinate)
- what does $\alpha=0$ mean?

Perspective Warp

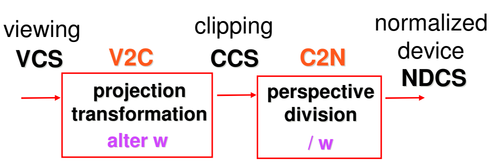
- matrix formulation

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&\frac{d}{d-\alpha}&\frac{-\alpha\cdot d}{d-\alpha}\\0&0&\frac{1}{d}&0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x\\y\\\frac{(z-\alpha)\cdot d}{d-\alpha}\\\frac{z}{d}\end{bmatrix}$$

$$(x_p,y_p,z_p)=\left(\frac{x}{z/d},\frac{y}{z/d},\frac{d^2}{d-\alpha}\left(1-\frac{\alpha}{z}\right)\right)$$

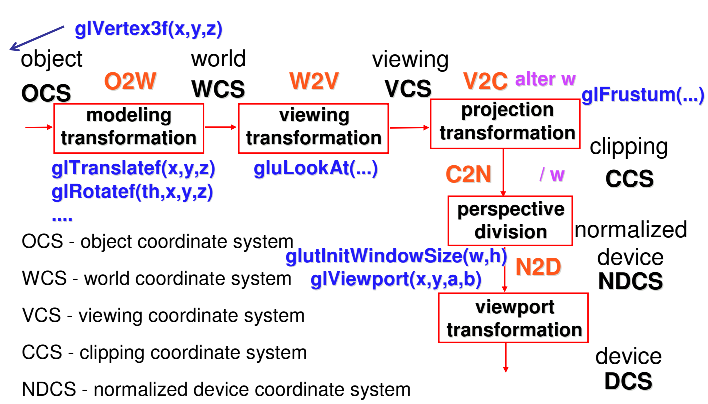
- preserves relative depth (third coordinate)
- what does $\alpha=0$ mean?

Projection Normalization



- distort such that orthographic projection of distorted objects is desired persp projection
  - separate division from standard matrix multiplies
  - clip after warp, before divide
  - division: normalization

Projective Rendering Pipeline



OCS - object coordinate system

WCS - world coordinate system

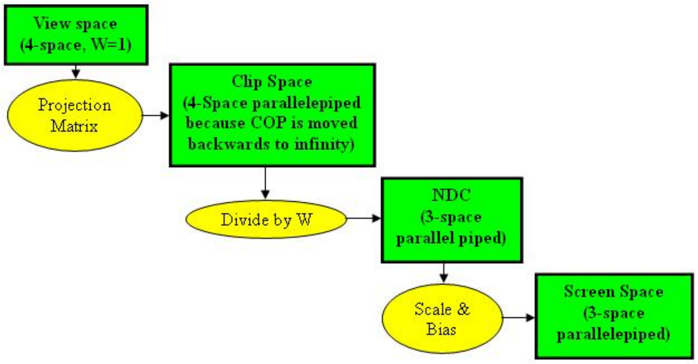
VCS - viewing coordinate system

CCS - clipping coordinate system

NDCS - normalized device coordinate system

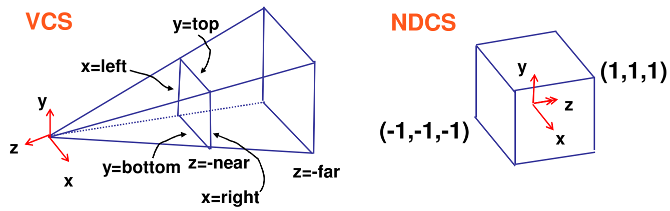
DCS - device coordinate system

Coordinate Systems



http://www.btinternet.com/~danbgs/perspective/

Perspective Derivation



Perspective Derivation

earlier:

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

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

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

$$x' = Ex + Az$$
$$y' = Fy + Bz$$
$$z' = Cz + D$$
$$w' = -z$$

$$x = left \rightarrow x'/w' = 1$$
$$x = right \rightarrow x'/w' = -1$$
$$y = top \rightarrow y'/w' = 1$$
$$y = bottom \rightarrow y'/w' = -1$$
$$z = -near \rightarrow z'/w' = 1$$
$$z = -far \rightarrow z'/w' = -1$$

$$y' = Fy + Bz, \quad \frac{y'}{w'} = \frac{Fy + Bz}{w'}, \quad 1 = \frac{Fy + Bz}{w'}, \quad 1 = \frac{Fy + Bz}{-z},$$

$$1 = F\frac{y}{-z} + B\frac{z}{-z}, \quad 1 = F\frac{y}{-z} - B, \quad 1 = F\frac{top}{-(-near)} - B,$$

$$1 = F\frac{top}{near} - B$$

Perspective Derivation

- similarly for other 5 planes
- 6 planes, 6 unknowns

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

Perspective Example

view volume

- left = -1, right = 1
- bot = -1, top = 1
- near = 1, far = 4

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -5/3 & -8/3 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspective Example

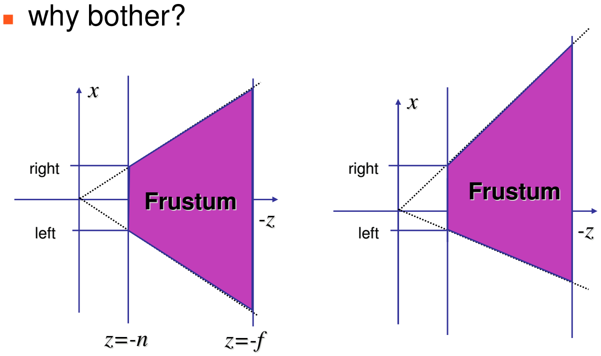
$$\begin{bmatrix} 1 \\ -1 \\ -5z_{VCS}/3-8/3 \\ -z_{VCS} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -5/3 & -8/3 \\ & & -1 & \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ z_{VCS} \\ 1 \end{bmatrix}$$

/ w

$$x_{NDCS} = -1/z_{VCS}$$
$$y_{NDCS} = 1/z_{VCS}$$
$$z_{NDCS} = \frac{5}{3} + \frac{8}{3z_{VCS}}$$

Asymmetric Frusta

- our formulation allows asymmetry
  - why bother?

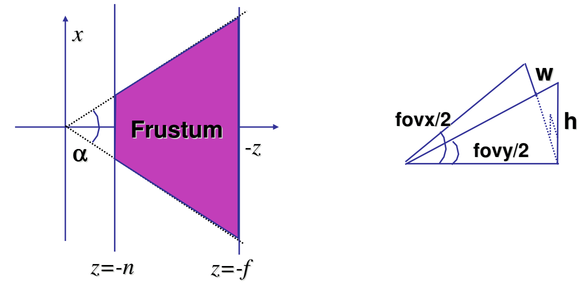


Simpler Formulation

- left, right, bottom, top, near, far
  - nonintuitive
  - often overkill
- look through window center
  - symmetric frustum
- constraints
  - left = -right, bottom = -top

Field-of-View Formulation

- FOV in one direction + aspect ratio (w/h)
  - determines FOV in other direction
  - also set near, far (reasonably intuitive)



Perspective OpenGL

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

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

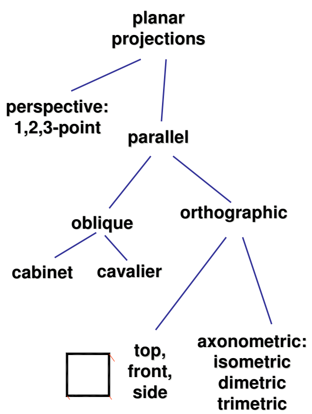
or

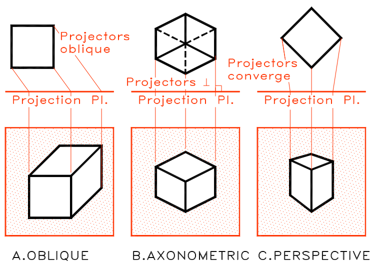
```
glPerspective(fovy,aspect,near,far);
```


Demo: Frustum vs. FOV

- Nate Robins tutorial (take 2):
  - http://www.xmission.com/~nate/tutors.html

Projection Taxonomy

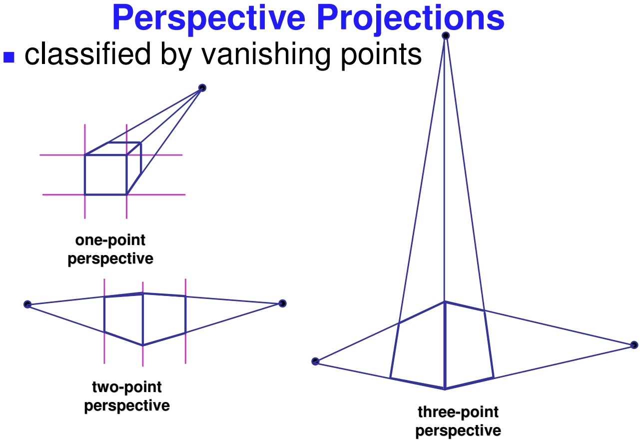




http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

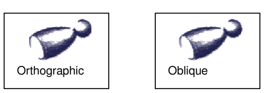
Perspective Projections

- classified by vanishing points



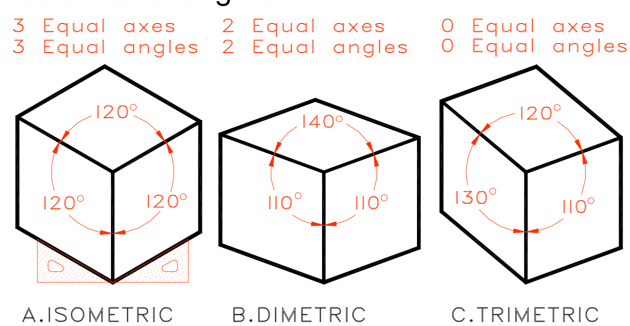
Parallel Projection

- projectors are all parallel
  - vs. perspective projectors that converge
  - orthographic: projectors perpendicular to projection plane
  - oblique: projectors not necessarily perpendicular to projection plane



Axonometric Projections

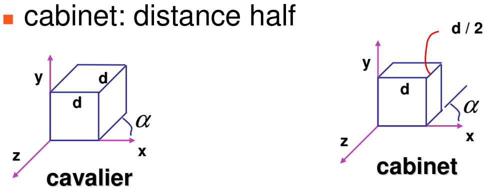
- projectors perpendicular to image plane
- select axis lengths



http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20

Oblique Projections

- projectors oblique to image plane
- select angle between front and z axis
  - lengths remain constant
- both have true front view
  - cavalier: distance true
  - cabinet: distance half



Demos

- Tuebingen applets from Frank Hanisch
  - http://www.gris.uni-tuebingen.de/projects/grdev/doc/html/etc/AppletIndex.html#Transformationen