University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2008

Tamara Munzner

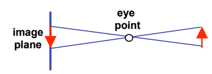
## Viewing/Projections III

## Week 4, Wed Jan 30

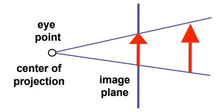
http://www.ugrad.cs.ubc.ca/~cs314/Vjan2008

## Review: Graphics Cameras

- real pinhole camera: image inverted

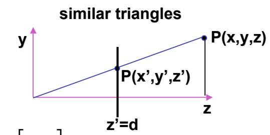



- computer graphics camera: convenient equivalent






## Review: Basic Perspective Projection




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

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

$$\begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \end{bmatrix} \xrightarrow{\text{homogeneous coords}} \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$



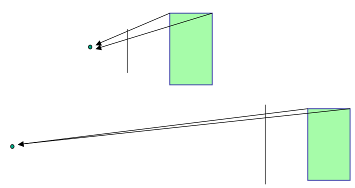
## Perspective Projection

- 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
- just throw away z values

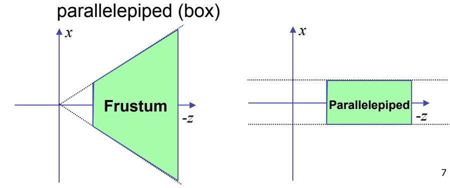
$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

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



## Perspective to Orthographic

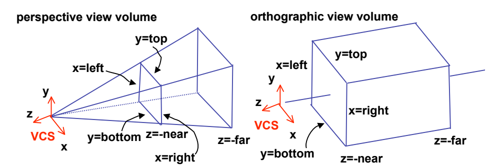
- transformation of space
  - center of projection moves to infinity
  - view volume transformed
    - from frustum (truncated pyramid) to parallelepiped (box)






## View Volumes

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



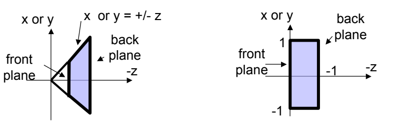



## Canonical View Volumes

- standardized viewing volume representation

perspective

orthographic
orthogonal
parallel






## Why Canonical View Volumes?

- permits standardization
  - clipping
    - easier to determine if an arbitrary point is enclosed in volume with canonical view volume vs. clipping to six arbitrary planes
  - rendering
    - projection and rasterization algorithms can be reused



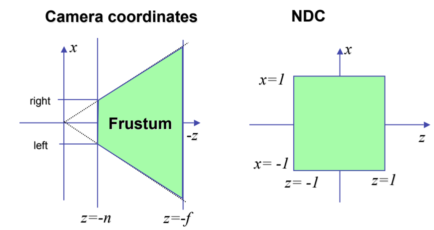
## Normalized Device Coordinates

- 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

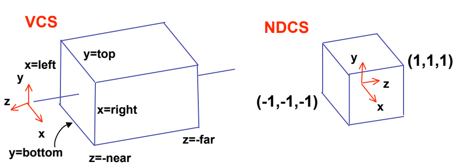
left/right $x$ =+/- 1, top/bottom $y$ =+/- 1, near/far $z$ =+/- 1






## Understanding Z

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

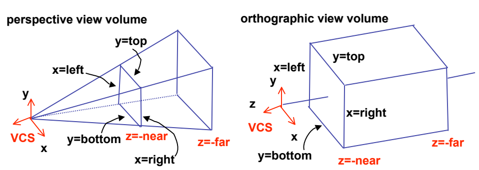





## Understanding Z

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);
```





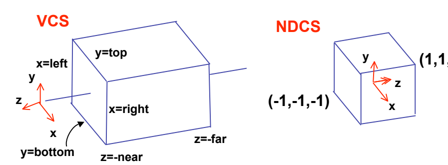

## Understanding Z

- 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

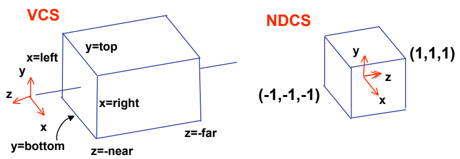
- scale, translate, reflect for new coord sys



## 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

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

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

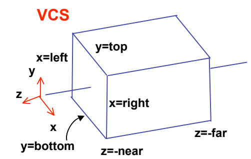
$$1 = \frac{2}{top - bot} top + b$$
$$b = 1 - \frac{2 \cdot top}{top - bot}$$
$$b = \frac{(top - bot) - 2 \cdot top}{top - bot}$$
$$b = \frac{-top - bot}{top - bot}$$



## 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

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



## Orthographic Derivation

- scale, translate, reflect for new coord sys

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



## Orthographic Derivation

- scale, translate, reflect for new coord sys

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



## Orthographic OpenGL

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



## Demo

- Brown applets: viewing techniques
  - parallel/orthographic cameras
  - projection cameras
- http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/viewing_techniques.html

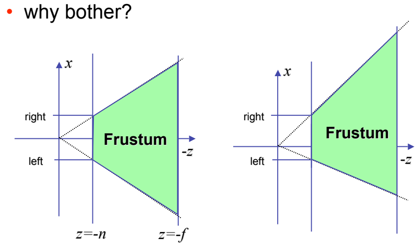


## Projections II



## Asymmetric Frusta

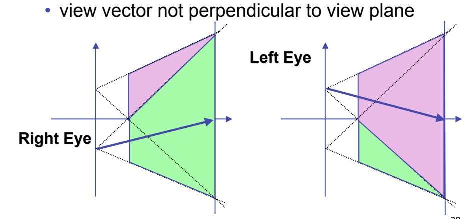
- our formulation allows asymmetry
  - why bother?






## Asymmetric Frusta

- our formulation allows asymmetry
  - why bother? binocular stereo
    - view vector not perpendicular to view plane





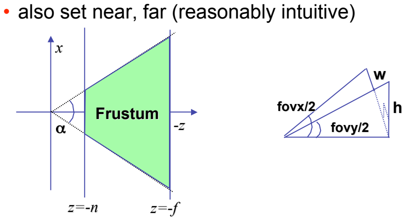

## Simpler Formulation

- 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

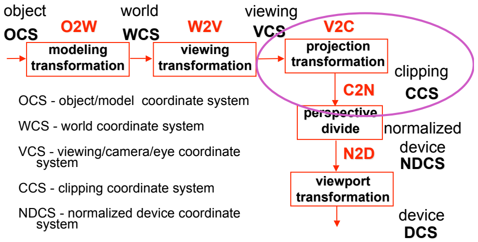
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

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

## Projective Rendering Pipeline




OCS - object/model coordinate system

WCS - world coordinate system

VCS - viewing/camera/eye coordinate system

CCS - clipping coordinate system

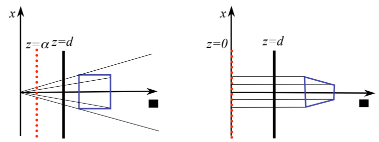
NDCS - normalized device coordinate system

DCS - device/display/screen coordinate system



## Projection Normalization

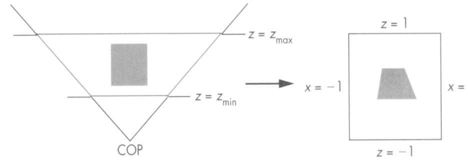
- warp perspective view volume to orthogonal view volume
  - render all scenes with orthographic projection!
  - aka perspective warp





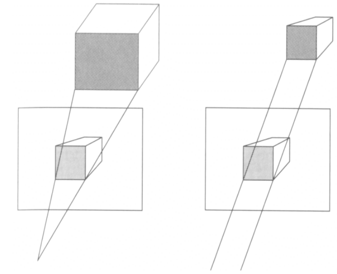

## Perspective Normalization

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






## Predistortion



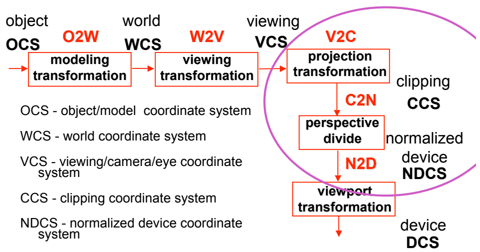


## Demos

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



## Projective Rendering Pipeline




OCS - object/model coordinate system

WCS - world coordinate system

VCS - viewing/camera/eye coordinate system

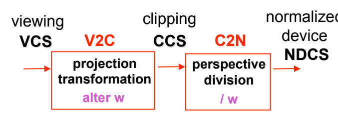
CCS - clipping coordinate system

NDCS - normalized device coordinate system

DCS - device/display/screen coordinate system



## Separate Warp From Homogenization

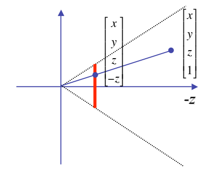



- warp requires only standard matrix multiply
  - distort such that orthographic projection of distorted objects is desired persp projection
    - w is changed
  - clip after warp, before divide
  - division by w: homogenization



## Perspective Divide Example

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

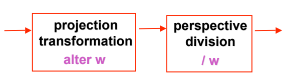





## Perspective Divide Example

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

- after homogenizing, once again w=1






## Perspective Normalization

- matrix formulation

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&\frac{d}{d-a}&\frac{-a\cdot d}{d-a}\\0&0&\frac{1}{d}&0\end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x\\y\\\frac{(z-a)\cdot d}{d-a}\\\frac{z}{d}\end{bmatrix}\qquad\begin{bmatrix}x_p\\y_p\\z_p\end{bmatrix}=\begin{bmatrix}\frac{x}{z/d}\\\frac{y}{z/d}\\\frac{d^2}{d-a}\left(1-\frac{a}{z}\right)\end{bmatrix}$$

- warp and homogenization both preserve relative depth (z coordinate)

## Demo

- Brown applets: viewing techniques
  - parallel/orthographic cameras
  - projection cameras
- http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/viewing_techniques.html