 UBC Dept. of ECE

EECE 478

Viewing and Projection

 UBC Dept. of ECE

Learning Objectives

Camera model
- Describe external and internal camera parameters
- Describe the geometry of perspective and orthogonal projections

Camera Representation
- Derive matrix representations of projections

©2005, Lee Iverson <leei@ece.ubc.ca> UBC Dept. of ECE

Learning Objectives

OpenGL Cameras
- Describe how cameras are represented in OpenGL
- Initialize OpenGL projections
- Be able to program a moving, variable zoom camera.

Vocabulary

- Orthogonal projection
- Perspective projection
- Camera frame
- View-Reference Point (VRP)
- View-Plane Normal (VPN)
- View-Up Vector (VUP)
- Roll, Pitch, Yaw
- Frustum
- Field of view
- Depth (Z) buffer
- Culling

©2005, Lee Iverson <leei@ece.ubc.ca> UBC Dept. of ECE

Viewing

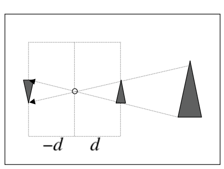
Determine 2D *view* given:
- World model (vertices and objects)
- Camera model:
  - position and orientation
  - orthographic or perspective
  - aspect ratio
  - field of view

⇒ projection from 3D to 2D

©2005, Lee Iverson <leei@ece.ubc.ca> UBC Dept. of ECE

Pinhole Camera

- Projection of object vertices through single point (*center of projection*)
- Image formed on *projection plane*
- Distance from COP to plane is *focal length* (d)



General Viewing

- Camera may be anywhere in scene
- Camera may be facing any direction
- Camera may vary focal length
  - even to infinity!

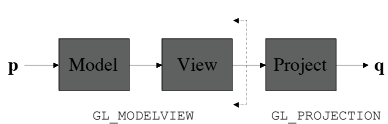
External Parameters

Camera wrt. the rest of the world
- position in world coordinates
- rotation away from world frame

Specified with a rigid-body transformation
- translation
- rotation

Moving Camera (1)

Moving camera ≡ moving world ⇒
- camera transform is inverse of instance transform
- view transform *follows* model transform



Moving Camera (2)

In matrix terms:
- instance transform for model
- *inverse* instance transform for camera
- reverse order of pipeline
- *all* combined in model-view matrix

$$\begin{aligned}\mathbf{q} &= \mathbf{P}\mathbf{V}^{-1}\mathbf{M}\mathbf{p}\\ &= \mathbf{P}(\mathbf{R}_V^{-1}\mathbf{T}_V^{-1})(\mathbf{T}_M\mathbf{R}_M\mathbf{S}_M)\mathbf{p}\end{aligned}$$

©2005, Lee Iverson <leei@ece.ubc.ca> UBC Dept. of ECE

Moving Camera (3)

Relative position, orientation determine view
- e.g. move *both* camera and model together
- same image as with no transform at all

$$\begin{aligned}\mathbf{V} = \mathbf{M} &= \mathbf{T}_M\mathbf{R}_M\\ \mathbf{q} &= \mathbf{P}(\mathbf{R}_M^{-1}\mathbf{T}_M^{-1})(\mathbf{T}_M\mathbf{R}_M)\mathbf{p}\\ &= \mathbf{P}(\mathbf{R}_M^{-1}(\mathbf{T}_M^{-1}\mathbf{T}_M)\mathbf{R}_M)\mathbf{p}\\ &= \mathbf{P}(\mathbf{R}_M^{-1}\mathbf{R}_M)\mathbf{p}\\ &= \mathbf{P}\mathbf{p}\end{aligned}$$

©2005, Lee Iverson <leei@ece.ubc.ca> UBC Dept. of ECE

Moving Camera: OpenGL

Camera position:
- COP at (p_x,p_y,p_z)
- Rotated θ degrees around axis (a_x,a_y,a_z)

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glRotatef (-theta, ax,ay,az);
glTranslatef (-px,-py,-pz);
```

Moving Camera: PHIGS

PHIGS and GKS

- separate *view* and *model* matrices
- specification of camera frame in terms of
  - VRP: view reference point
  - VPN: view plane normal
  - VUP: view up

$$\mathbf{p} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$$
$$\mathbf{n} = \begin{bmatrix} n_x & n_y & n_z & 0 \end{bmatrix}^T$$
$$\mathbf{v}_{up} = \begin{bmatrix} v_{up_x} & v_{up_y} & v_{up_z} & 0 \end{bmatrix}^T$$

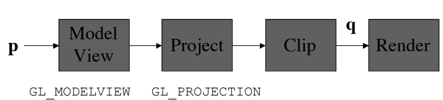
Moving Camera: LookAt

- Place camera at *eye* point
- Face towards *at* point
- Specify *up* direction

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
gluLookAt (ex,ey,ez, ax,ay,az,
           ux,uy,uz);
```


Projections

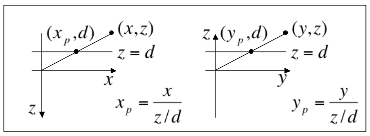
- Reduce from 3D space to 2D
  - **p** is 3D
  - **q** is 2D
    - input to renderer
    - produced by projection and division



Pinhole Algebra

- Project through origin
- Project onto line $z = d$

$\Rightarrow$ division by z/d (foreshortening)



Homogeneous Projection

Allow points to exist in unnormalized form
- final coordinate allowed to vary ($\neq 0$)
- to normalize we must divide by w
- unnormalized form temporary

$$\begin{aligned}\mathbf{p} &= \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T \\ &= \begin{bmatrix} wx & wy & wz & w \end{bmatrix}^T\end{aligned}$$

Perspective Projection

Matrix *creates* unnormalized points
- normalize by division

$$\mathbf{p} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$\begin{aligned}\mathbf{q} &= \mathbf{M}\mathbf{p} \\ &= \begin{bmatrix} x & y & z & z/d \end{bmatrix}^T \\ &= \begin{bmatrix} \frac{x}{z/d} & \frac{y}{z/d} & d & 1 \end{bmatrix}^T\end{aligned}$$

Orthogonal Projection

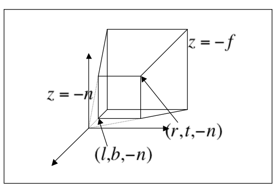
Aligned with z axis into screen
- project by collapsing to $z = 0$

$$\mathbf{p} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T \qquad \mathbf{q} = \mathbf{M}\mathbf{p}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad = \begin{bmatrix} x & y & 0 & 1 \end{bmatrix}^T$$

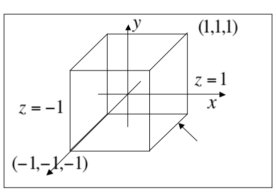
View Frustum(1)

Frustum: Volume projected onto screen
- distance varies from *near* to *far*
- (*left,right*) and (*bottom,top*)



Canonical View Volume

Both perspective and ortho collapse visible space to a *canonical view volume*
- x, y, and z all vary from -1 to 1



View Frustum(2)

Transform frustrum to
- *canonical view volume*
- x, y, and z all vary from -1 to 1

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

View Frustum(3)

Examples:

$$\mathbf{p}' = \mathbf{P}\begin{bmatrix} l \\ b \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} n(l-r)/(r-l) \\ n(b-t)/(t-b) \\ n(n-f)/(f-n) \\ n \end{bmatrix} = \begin{bmatrix} -n \\ -n \\ -n \\ n \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{P}\begin{bmatrix} r \\ t \\ -f \\ 1 \end{bmatrix} = \begin{bmatrix} n(r-l)/(r-l) \\ n(t-b)/(t-b) \\ n(f-n)/(f-n) \\ n \end{bmatrix} = \begin{bmatrix} n \\ n \\ n \\ n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

View Frustum: OpenGL

Specify frustum by providing:
- [*left,right*] and [*bottom,top*] bounds
- [*near,far*] distance range

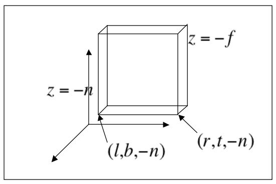
```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glFrustum (left,right, bottom,top,
           near,far);

glMatrixMode (GL_MODELVIEW);
/* Define external camera parameters */
```

Orthographic Projection(1)

Project along z axis (*not through point*)

- distance varies from *near* to *far*
- (*left,right*) and (*bottom,top*)



Orthographic Projection(2)

Transform box to

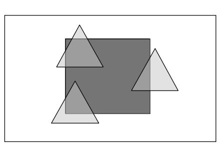
- *canonical view volume*
- x, y, and z all vary from -1 to 1

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

Canonical Clipping Volume

Canonical view volume determines what will be drawn on screen

- Throw away everything outside of CVV
- Must draw *partial* objects at edges



Z-Buffer Algorithm

From CCV (part of rasterization):

- Z values retain *relative* depth
- Clear Z buffer when draw buffer cleared
- Interpolate Z value of every polygon pixel
- If Z < *current* Z *at that pixel:*
  - Draw pixel
  - Update current Z at that pixel
- Otherwise ignore it.

©2005, Lee Iverson <leei@ece.ubc.ca> UBC Dept. of ECE

New Pipeline

- Transform by model-view
- Project to canonical view volume
- Clip primitives against CVV
- Rasterize into depth and frame buffers