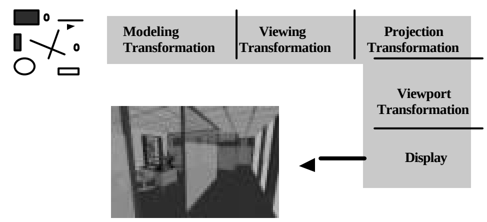
CS 4731: Computer Graphics
Lecture 13: Projection

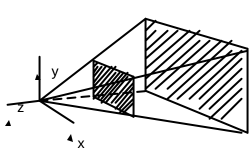
Emmanuel Agu

3D Projection

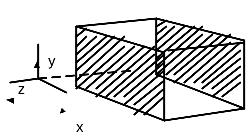


Projection Transformation

- Projection – map the object from 3D space to 2D screen



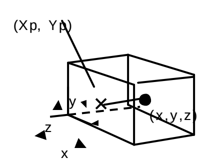
Perspective: **gluPerspective()**



Parallel: **glOrtho()**

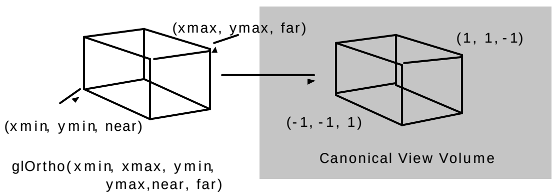
Parallel Projection

- After transforming the object to the eye space, parallel projection is relatively easy – we could just drop the Z
  - Xp = x
  - Yp = y
  - Zp = -d
- We actually want to keep Z – why?



Parallel Projection

- OpenGL maps (projects) everything in the visible volume into a canonical view volume



glOrtho(xmin, xmax, ymin, ymax,near, far)

Parallel Projection: glOrtho

- Parallel projection can be broken down into two parts
- Translation which centers view volume at origin
- Scaling which reduces cuboid of arbitrary dimensions to canonical cube (dimension 2, centered at origin)

Parallel Projection: glOrtho

- Translation sequence moves midpoint of view volume to coincide with origin:
- E.g. midpoint of x = (xmax + xmin)/2
- Thus translation factors:
  -(xmax+xmin)/2, -(ymax+ymin)/2, -(far+near)/2
- And translation matrix M1:

$$\begin{pmatrix} 1 & 0 & 0 & -(x\max + x\min)\ /\ 2 \\ 0 & 1 & 0 & -(y\max + y\min)\ /\ 2 \\ 0 & 0 & 1 & -(z\max + z\min)\ /\ 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parallel Projection: glOrtho

- Scaling factor is ratio of cube dimension to Ortho view volume dimension
- Scaling factors:
  2/(xmax-xmin), 2/(ymax-ymin), 2/(zmax-zmin)
- Scaling Matrix M2:

$$\begin{pmatrix} \frac{2}{x\max - x\min} & 0 & 0 & 0 \\ 0 & \frac{2}{y\max - y\min} & 0 & 0 \\ 0 & 0 & \frac{2}{z\max - z\min} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parallel Projection: glOrtho

Concatenating M1xM2, we get transform matrix used by glOrtho

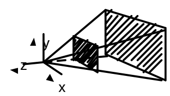
$$\begin{pmatrix} \frac{2}{x\max - x\min} & 0 & 0 & 0 \\ 0 & \frac{2}{y\max - y\min} & 0 & 0 \\ 0 & 0 & \frac{2}{z\max - z\min} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{X} \begin{pmatrix} 1 & 0 & 0 & -(x\max + x\min)/2 \\ 0 & 1 & 0 & -(y\max + y\min)/2 \\ 0 & 0 & 1 & -(z\max + z\min)/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

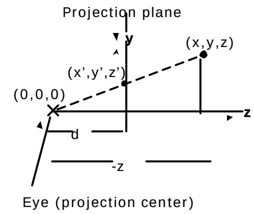
$$M2 \times M1 = \begin{pmatrix} 2/(x\max - x\min) & 0 & 0 & -(x\max + x\min)/(x\max - x\min) \\ 0 & 2/(y\max - y\min) & 0 & -(y\max + \min)/(y\max - \min) \\ 0 & 0 & 2/(z\max - z\min) & -(z\max + z\min)/(z\max - z\min) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Refer: Hill, 7.6.2

Perspective Projection: Classical

- Side view:





Based on similar triangle:

$$\frac{y}{y'} = \frac{-z}{d}$$

$$\Rightarrow y' = y \times \frac{d}{-z}$$

Perspective Projection: Classical

- So (x*,y*) the projection of point, (x,y,z) unto the near plane N is given as:

$$(x^*, y^*) = \left(N \frac{P_x}{-P_z}, N \frac{P_y}{-P_z} \right)$$

- Numerical example:

Q. Where on the viewplane does P = (1, 0.5, -1.5) lie for a near plane at N = 1?

- (x*, y*) = (1 x 1/1.5, 1 x 0.5/1.5) = (0.666, 0.333)

Pseudodepth

- Classical perspective projection projects (x,y) coordinates, drops z coordinates
- But we need z to find closest object (depth testing)
- Keeping actual distance of P from eye is cumbersome and slow

$$distance = \sqrt{\left(P_x^2 + P_y^2 + P_z^2\right)}$$

- Introduce **pseudodepth**: all we need is measure of which objects are further if two points project to same (x,y)

$$(x^*, y^*, z^*) = \left(N \frac{P_x}{-P_z}, N \frac{P_y}{-P_z}, \frac{aP_z + b}{-P_z} \right)$$

- Choose a, b so that pseudodepth varies from –1 to 1 (canonical cube)

Pseudodepth

- Solving:

$$z^* = \frac{aP_z + b}{-P_z}$$

- For two conditions, z* = -1 when Pz = -N and z* = 1 when Pz = -F, we can set up two simultaneuous equations
- Solving:

$$a = \frac{F+N}{F-N} \qquad b = \frac{-2FN}{F-N}$$

Homogenous Coordinates

- Would like to express projection as 4x4 transform matrix
- Previously, homogeneous coordinates of the point P = (Px,Py,Pz) was (Px,Py,Pz,1)
- Introduce arbitrary scaling factor, w, so that P = (wPx, wPy, wPz, w) (Note: w is non-zero)
- For example, the point P = (2,4,6) can be expressed as
  - (2,4,6,1)
  - or (4,8,12,2) where w=2
  - or (6,12,18,3) where w = 3
- So, to convert from homogeneous back to ordinary coordinates, divide all four terms by last component and discard 4th term

Perspective Projection

- Same for x. So we have:
  x' = x x d / -z
  y' = y x d / - z
  z' = -d
- Put in a matrix form:

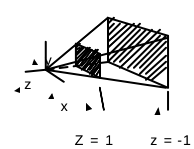
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \left(\frac{1}{-d}\right) & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w \end{pmatrix} \Rightarrow \begin{pmatrix} -d\left(\frac{x}{z}\right) \\ -d\left(\frac{y}{z}\right) \\ -d \\ 1 \end{pmatrix}$$

OpenGL assumes d = 1, i.e. the image plane is at z = -1

Perspective Projection

- We are not done yet.
- Need to modify the projection matrix to include a and b

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



We have already solved a and b

Perspective Projection

- Not done yet. OpenGL also normalizes the x and y ranges of the viewing frustum to [-1, 1] (translate and scale)
- So, as in ortho to arrive at final projection matrix
- we translate by
  - –(xmax + xmin)/2 in x
  - -(ymax + ymin)/2 in y
- Scale by:
  - 2/(xmax – xmin) in x
  - 2/(ymax – ymin) in y

Perspective Projection

- Final Projection Matrix:

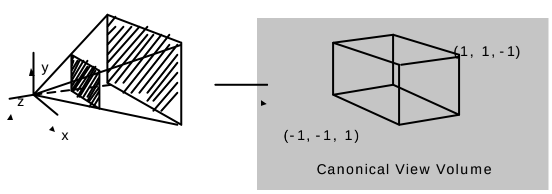
$$\begin{pmatrix} \frac{2N}{x\max - x\min} & 0 & \frac{x\max + x\min}{x\max - x\min} & 0 \\ 0 & \frac{2N}{y\max - y\min} & \frac{y\max + y\min}{y\max - y\min} & 0 \\ 0 & 0 & \frac{-(F+N)}{F-N} & \frac{-2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

⇧

glFrustum(xmin, xmax, ymin, ymax, N, F) N = near plane, F = far plane

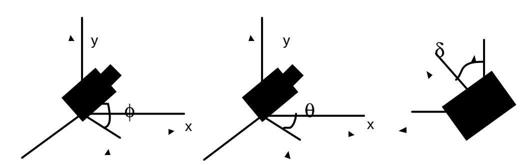
Perspective Projection

- After perspective projection, viewing frustum is also projected into a canonical view volume (like in parallel projection)



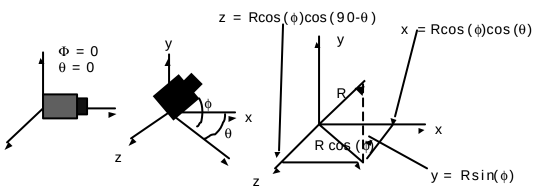
Flexible Camera Control

- Instead of provide COI, it is possible to just give camera orientation
- Just like control a airplane's orientation



Flexible Camera Control

- How to compute the viewing vector (x,y,z) from pitch(ϕ) and yaw(θ) ?



Flexible Camera Control

- gluLookAt() does not let you to control pitch and yaw
- you need to
  - User supplies θ, ϕ or roll angle
  - Compute/maintain the vector by yourself
  - Calculate COI = Eye + (x,y,z)
  - Then, call gluLookAt().

References

- Hill, chapter 7