

CSC461: Lecture 21

Perspective Projections in OpenGL

Objectives

- Derive the perspective projection matrices used for standard OpenGL projections
- Introduce shadows

Simple Perspective

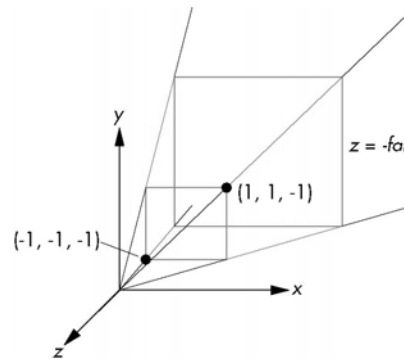
- Consider a simple perspective with the COP at the origin, the near clipping plane at $z = -1$ ($d=1$)
- Simple projection matrix in homogeneous coordinates
- Note that this matrix is independent of the far clipping plane

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

Simple Perspective View Volume

- A 90 degree field of view determined by the planes
- Intersect the projection plane at 45 degree
- This gives $x = \pm z, y = \pm z$

- Near plane $z = \text{near} (<0)$
- Far plane $z = \text{far} (<0)$
- $\text{far} > \text{near}$
- To find a projection matrix, we normalize it to default view volume



Generalization

- Consider the matrix

$$N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where α and β to be specified

- N is called *perspective normalization matrix*
- For any point on object $p = (x \ y \ z \ 1)^T$, applying N, the new point $q = (x' \ y' \ z' \ w')^T$, where

$$x' = x, \ y' = y, \ z' = \alpha z + \beta, \ w' = -z$$

- After perspective division, the point $(x, y, z, 1)$ goes to $x'' = -x/z, \ y'' = -y/z, \ z'' = -(\alpha + \beta/z)$

Projection Matrix

- Apply an orthographic projection along the z-axis to N, which projects orthogonally to the desired point regardless of a and b

$$M_{\text{orth}}N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$
- Project point p to p': $p' = M_{\text{orth}}Np = \begin{bmatrix} x \\ y \\ 0 \\ -z \end{bmatrix}$
- Do perspective division: $p' = \begin{bmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ 0 \\ 1 \end{bmatrix}$
- Conclusion: to obtain the perspective projection, to three steps: applying N, orthographic projection, and perspective division

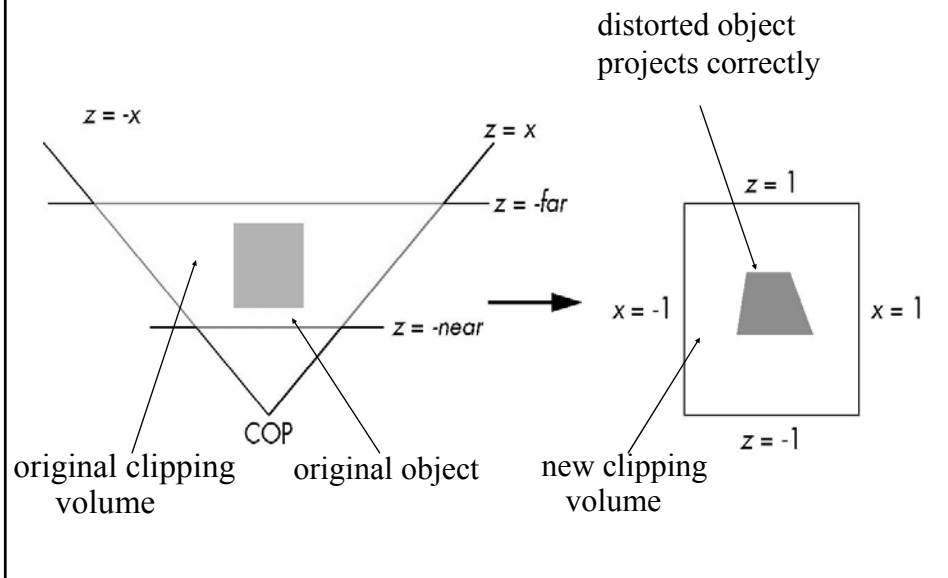
Picking α and β

- If we pick

$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

$$\beta = \frac{2\text{near} * \text{far}}{\text{near} - \text{far}}$$
 - the near plane is mapped to $z = -1$
 - the far plane is mapped to $z = 1$
 - and the sides are mapped to $x = \pm 1, y = \pm 1$
- Hence the new clipping volume is the default clipping volume

Normalization Transformation

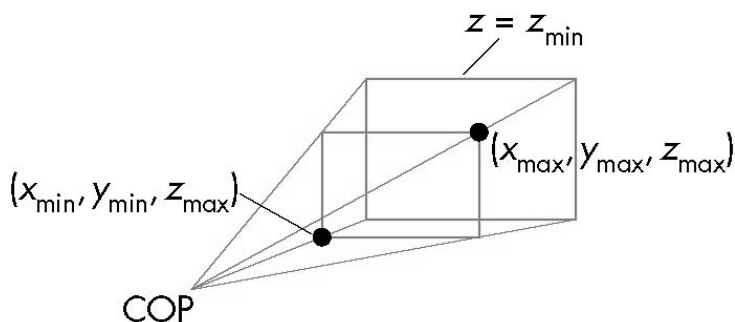


Normalization and Hidden-Surface Removal

- Although our selection of the form of the perspective matrices may appear somewhat arbitrary, it was chosen so that if $z_1 > z_2$ in the original clipping volume then the for the transformed points $z_1' > z_2'$ \rightarrow preserves the ordering of depths
- Thus we hidden surface removal works if we first apply the normalization transformation
- However, the formula $z'' = -(\alpha + \beta/z)$ implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small

OpenGL Perspective

- `glFrustum` allows for an unsymmetric viewing frustum
- `gluPerspective` does not



OpenGL Perspective Matrix

- The normalization in `glFrustum` requires
 - an initial shear to form a right viewing pyramid,
 - followed by a scaling to get the normalized perspective volume.
 - Finally, the perspective matrix results in needing only a final orthogonal transformation

$$P = NSH$$

our previously defined perspective matrix shear and scale

Shear and Scaling

- Shear the point $((x_{\text{far}}+x_{\text{near}})/2, (y_{\text{far}}+y_{\text{near}})/2)$ to $(0,0,\text{near})$
→ Shear matrix:
$$H(\theta, \phi) = H(\cot^{-1}((x_{\text{far}}+x_{\text{near}})/(2\text{far})), \cot^{-1}((y_{\text{far}}+y_{\text{near}})/(2\text{far})))$$
- Results: $x = \pm (x_{\text{far}}-x_{\text{near}})/(2\text{far})$
 $y = \pm (y_{\text{far}}-y_{\text{near}})/(2\text{far})$
 $z = \text{near}, z = \text{far}$
- Scale the sides of the frustum to $x = y = \pm z$ without changing the near and the far plane → scaling matrix:
$$S(2\text{far}/(x_{\text{far}}-x_{\text{near}}), 2\text{far}/(y_{\text{far}}-y_{\text{near}}), 1)$$

Why do we do it this way?

- Normalization allows for a single pipeline for both perspective and orthogonal viewing
- We keep in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading
- We simplify clipping

Projections and Shadows

- Default light source is assumed at the COP
- With default light source, all shadows are invisible (behind the visible objects)
- General lighting will be the topic of the next chapter.
- Some special cases related to projections are discussed here

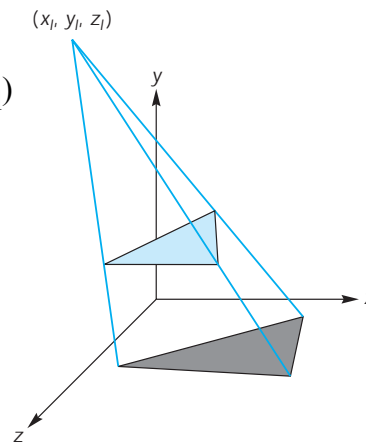
Shadow Polygon

- Shadow falls on the ground $y = 0$, forming a *shadow polygon*
- Shadow polygon is the projection of the object onto the ground with the COP at the light source
- To draw the shadow polygon

- Move the origin to the light source \rightarrow translation $T(-x_l, -y_l, -z_l)$
- Perspective projection with the projection plane $y = 0 \rightarrow M =$

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{y_l} & 0 \end{bmatrix}$$

- Translate back $\rightarrow T(x_l, y_l, z_l)$



OpenGL Code

```
GLfloat m[16] = {...};           //move back
//set the dadow projection matrix  glTranslatef(xl,yl,zl);
glColor3fv(polygon_color)        // project
glBegin(GL_POLYGON)              glMultMatrixf(m);
...                               // move light to origin
...// draw polygon              glTranslatef(-xl,-yl,-zl);
...                               glColor3f(shadow_color);
glEnd(GL_POLYGON)                glBegin(GL_POLYGON);
glMatrixMode(GL_MODELVIEW);     ... //re-draw polygon
glPushMatrix();                 glEnd);
                                glPopMatrix()
```