

The LookAt Function

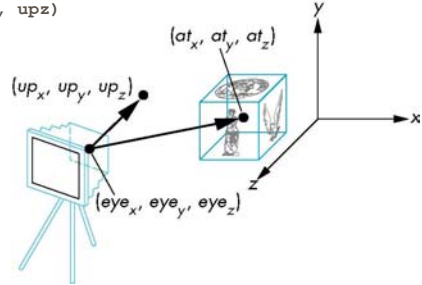
- The GLU library contains the function `glLookAt` to from the required modelview matrix through a simple interface
- Note the need for setting an up direction
- Still need to initialize
 - Can concatenate with modeling transformations
- Example: isometric view of cube aligned with axes

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0., 1.0, 0.0);
```



Slides by Edward Angel © 2002

```
gluLookAt(eyex, eyeey, eyez, atx, aty, atz, upx,
          upy, upz)
```



Slides by Edward Angel © 2002

Other Viewing APIs

- The LookAt function is only one possible API for positioning the camera
- Others include
 - View reference point, view plane normal, view up (PHIGS, GKS-3D)
 - Yaw, pitch, roll
 - Elevation, azimuth, twist
 - Direction angles



Slides by Edward Angel © 2002

Projections and Normalization

- The default projection in the eye (camera) frame is orthogonal
- For points within the default view volume

$$\begin{aligned} x_p &= x \\ y_p &= y \\ z_p &= 0 \end{aligned}$$
- Most graphics systems use *view normalization*
 - All other views are converted to the default view by transformations that determine the projection matrix
 - Allows use of the same pipeline for all views



Slides by Edward Angel © 2002

Homogeneous Coordinate Representation

$$\begin{aligned} x_p &= x \\ y_p &= y \\ z_p &= 0 \\ w_p &= 1 \end{aligned} \quad \mathbf{p}_p = \mathbf{M}\mathbf{p}$$

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

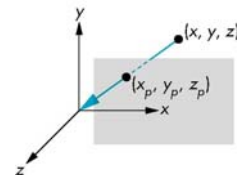
In practice, we can let $\mathbf{M} = \mathbf{I}$ and set the z term to zero later



Slides by Edward Angel © 2002

Simple Perspective

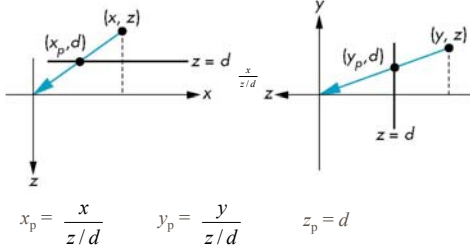
- Center of projection at the origin
- Projection plane $z = d, d < 0$



Slides by Edward Angel © 2002

Perspective Equations

Consider top and side views



Slides by Edward Angel © 2002

Homogeneous Coordinate Form

consider $q = Mp$ where

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

$$q = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow p = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$



Slides by Edward Angel © 2002

Perspective Division

- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates
- This *perspective division* yields

$$x_p = \frac{x}{z/d} \quad y_p = \frac{y}{z/d} \quad z_p = d$$

the desired perspective equations

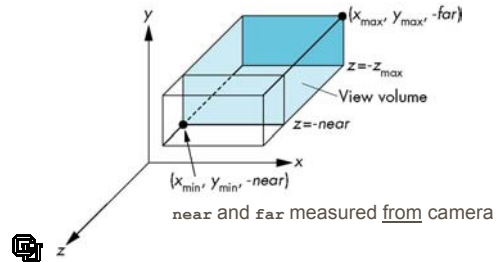
- We will consider the corresponding clipping volume with the OpenGL functions



Slides by Edward Angel © 2002

OpenGL Orthogonal Viewing

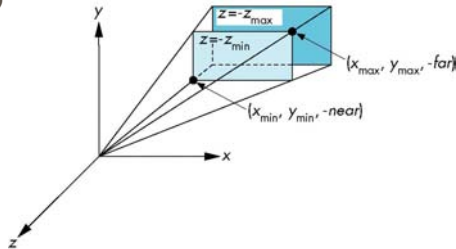
`glOrtho(xmin, xmax, ymin, ymax, near, far)`
`glOrtho(left, right, bottom, top, near, far)`



Slides by Edward Angel © 2002

OpenGL Perspective

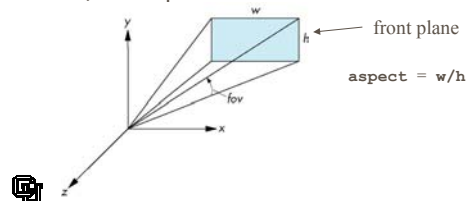
`glFrustum(xmin, xmax, ymin, ymax, near, far)`



Slides by Edward Angel © 2002

Using Field of View

- With `glFrustum` it is often difficult to get the desired view
- `gluPerspective(fovy, aspect, near, far)` often provides a better interface



Slides by Edward Angel © 2002

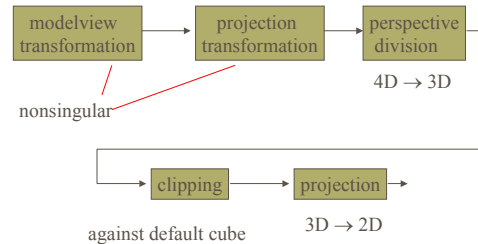
Normalization

- Rather than derive a different projection matrix for each type of projection, we can convert all projections to orthogonal projections with the default view volume
- This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping



Slides by Edward Angel © 2002

Pipeline View



Slides by Edward Angel © 2002

Notes

- We stay in four-dimensional homogeneous coordinates through both the modelview and projection transformations
 - Both these transformations are nonsingular
 - Default to identity matrices (orthogonal view)
- Normalization lets us clip against simple cube regardless of type of projection
- Delay final projection until end
 - Important for hidden-surface removal to retain depth information as long as possible

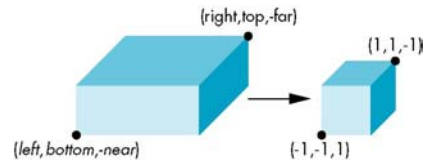


Slides by Edward Angel © 2002

Orthogonal Normalization

`glOrtho(left, right, bottom, top, near, far)`

normalization \Rightarrow find transformation to convert specified clipping volume to default



Slides by Edward Angel © 2002

Orthogonal Matrix

- Two steps
 - Move center to origin
 $T(-(\text{left}+\text{right})/2, -(\text{bottom}+\text{top})/2, (\text{near}+\text{far})/2)$
 - Scale to have sides of length 2
 $S(2/(\text{left}-\text{right}), 2/(\text{top}-\text{bottom}), 2/(\text{near}-\text{far}))$

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



Slides by Edward Angel © 2002

Final Projection

- Set $z=0$
- Equivalent to the homogeneous coordinate transformation

$$M_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Hence, general orthogonal projection in 4D is

$$P = M_{\text{orth}}ST$$



Slides by Edward Angel © 2002

Oblique Projections

- The OpenGL projection functions cannot produce general parallel projections such as

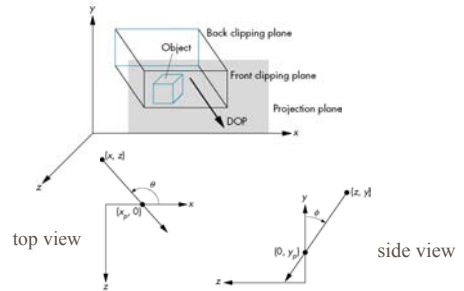


- However if we look at the example of the cube it appears that the cube has been sheared
- Oblique Projection = Shear + Orthogonal Projection



Slides by Edward Angel © 2002

General Shear



Slides by Edward Angel © 2002

Shear Matrix

xy shear (z values unchanged)

$$H(\theta, \phi) = \begin{bmatrix} 1 & 0 & -\cot \theta & 0 \\ 0 & 1 & -\cot \phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection matrix
 $P = M_{\text{orth}} H(\theta, \phi)$

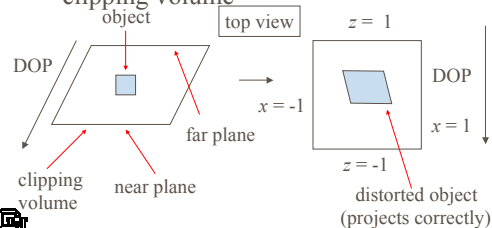
General case: $P = M_{\text{orth}} STH(\theta, \phi)$



Slides by Edward Angel © 2002

Effect on Clipping

- The projection matrix $P = STH$ transforms the original clipping volume to the default clipping volume

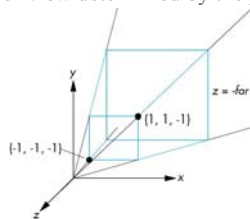


Slides by Edward Angel © 2002

Simple Perspective

Consider a simple perspective with the COP at the origin, the near clipping plane at $z = -1$, and a 90 degree field of view determined by the planes

$$x = \pm z, y = \pm z$$



Slides by Edward Angel © 2002

Perspective Matrices

Simple projection matrix in homogeneous coordinates

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

Note that this matrix is independent of the far clipping plane



Slides by Edward Angel © 2002

Generalization

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

after perspective division, the point $(x, y, z, 1)$ goes to

$$\begin{aligned} x'' &= x/z \\ y'' &= y/z \\ z'' &= -(\alpha + \beta/z) \end{aligned}$$

which projects orthogonally to the desired point regardless of α and β



Slides by Edward Angel © 2002

Picking α and β

If we pick

$$\begin{aligned} \alpha &= \frac{\text{near} + \text{far}}{\text{far} - \text{near}} \\ \beta &= \frac{2\text{near} * \text{far}}{\text{near} - \text{far}} \end{aligned}$$

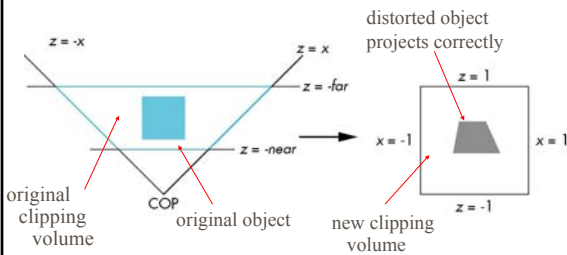
the near plane is mapped to $z = -1$
the far plane is mapped to $z = 1$
and the sides are mapped to $x = \pm 1, y = \pm 1$

Hence the new clipping volume is the default clipping volume



Slides by Edward Angel © 2002

Normalization Transformation



Slides by Edward Angel © 2002

Normalization and Hidden-Surface Removal

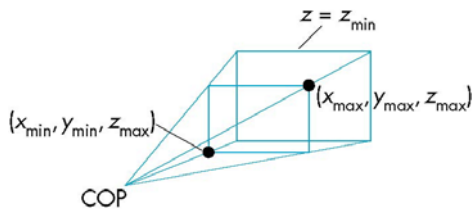
- Although our selection of the form of the perspective matrices may appear somewhat arbitrary, it was chosen so that if $z_1 > z_2$ in the original clipping volume then the for the transformed points $z_1' > z_2'$
- Thus we hidden surface removal works if we first apply the normalization transformation
- 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



Slides by Edward Angel © 2002

OpenGL Perspective

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



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002



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

