Perspective transformations, transformation pipeline

Coordinate systems

Eye coordinates - coordinate system attached to the camera; in this system camera looks down negative Z-axis

World coordinates - fixed initial coord system; everything is defined with respect to it

Positioning the camera

Viewing transformation is the inverse of the camera positioning transformation:

Camera positioning: translate by \((t_x, t_z)\)

Viewing transformation (world to eye):

\[ x_{\text{eye}} = x_{\text{world}} - t_z \]
\[ z_{\text{eye}} = z_{\text{world}} - t_z \]

Transformation pipeline

Modelview: model (position objects) + view (position the camera)
Projection: map viewing volume to a standard cube
Perspective division: project 3D to 2D
Viewport: map the square \([-1,1] \times [-1,1]\) in normalized device coordinates to the screen

Positioning the camera

- Modeling transformation: reshape the object, orient the object, position the object with respect to the world coordinate system
- Viewing transformation: transform world coordinates to eye coordinates
- Viewing transformation is the inverse of the camera positioning transformation
- Viewing transformation should be rigid: rotation + translation
- Steps to get the right transform: first, orient the camera correctly, then translate it

Look-at positioning

Find the viewing transform given the eye (camera) position, point to look at, and the up vector
- Need to specify two transforms: rotation and translation.
- Translation is easy
- Natural rotation: define implicitly using a point at which we want to look and a vector indicating the vertical in the image (up vector)
can easily convert the eye point to the direction vector of the camera axis; can assume up vector perpendicular to view vector
Look-at positioning

Problem: given two pairs of perpendicular unit vectors, find the transformation mapping the first pair into the second

\[ u \] (Eye coords)
\[ v = \begin{bmatrix} \frac{1}{\sqrt{1 + u \cdot u}} \\ \frac{1}{\sqrt{1 + u \cdot u}} \\ 0 \end{bmatrix} \] (World coords)

Look-at positioning

Recall the matrix for translation:

\[ T = \begin{bmatrix} 1 & 0 & 0 & \alpha \\ 0 & 1 & 0 & \beta \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

Now we have the camera positioning matrix, \( T_{HR} \)

To get the viewing transform, invert: \( (T_{HR})^{-1} = R^{-1}T^{-1} \)

For rotation the inverse is the transpose:

\[ R^{-1} = [v \times u]^{T} = \begin{bmatrix} (v \times u)^{T} \\ u^{T} \\ -w^{T} \end{bmatrix} \]

Positioning the camera in OpenGL

- imagine that the camera is an object and write a sequence of rotations and translations positioning it
- change each transformation in the sequence to the opposite
- reverse the sequence
- Camera positioning is done in the code before modeling transformations
- OpenGL does not distinguish between viewing and modeling transformation and joins them into the modelview matrix

Space to plane projection

In eye coordinate system

\[ e = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \]

Image plane

\[ v = \begin{bmatrix} \frac{-n_{y}}{n_{z}} \\ \frac{-n_{y}}{n_{z}} \\ 0 \\ -1 \end{bmatrix} \]

\[ p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

Look-at positioning

Determine rotation first, looking how coord vectors change:

\[ R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = v \quad R \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = u \times v \quad R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = w \]

\[ R \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = R = [v \times u \ u - v] \]
Visibility

Objects that are closer to the camera occlude the objects that are further away

- All objects are made of planar polygons
- A polygon typically projects 1 to 1
- Idea: project polygons in turn; for each pixel, record distance to the projected polygon
- When writing pixels, replace the old color with the new one only if the new distance to camera for this pixel is less than the recorded one

Z-buffering idea

- Problem: need to compare distances for each projected point
- Solution: convert all points to a coordinate system in which (x,y) are image plane coords and the distance to the image plane increases when the z coordinate increases
- In OpenGL, this is done by the projection matrix

Viewing frustum

Volume in space that will be visible in the image

- f is the aspect ratio of the image width/height
- Viewing frustum

Projection transformation

Maps the viewing frustum into a standard cube extending from -1 to 1 in each coordinate (normalized device coordinates)

3 steps:
- Change the matrix of projection to keep z: result is a parallelepiped
- Translate: parallelepiped centered at 0
- Scale in all directions: cube of size 2 centered at 0

Projection transformation

Change

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

So that we keep z:

$$\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1 & 1
\end{bmatrix} \begin{bmatrix}
p_x \\
p_y \\
p_z \\
p_w
\end{bmatrix} = \begin{bmatrix}
p_x \\
p_y \\
p_z \\
-1
\end{bmatrix}
$$

The homogeneous result corresponds to:

$$\begin{bmatrix}
p_z/p_w \\
p_y/p_w \\
p_z \\
p_w
\end{bmatrix}$$

The last component increases monotonically with z!
Projection transformation

Combined matrix, mapping frustum to a cube:

\[
P \rightarrow \mathcal{F} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & -1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & f \cdot n
\end{bmatrix}
\]

To get normalized image plane coordinates (valid range [-1,1] both), just drop \( z \) in the result and convert from homogeneous to regular.

To get pixel coordinates, translate by 1, and scale \( x \) and \( y \) (Viewport transformation)

Transformation pipeline

Modelview: model (position objects) + view (position the camera)

Projection: map viewing volume to a standard cube

Perspective division: project 3D to 2D

Viewport: map the square \([-1,1] \times [-1,1]\) in normalized device coordinates to the screen