Review of the Graphics Pipeline

3D Clipping

OpenGL Pipeline

The Graphics Pipeline

Object Space → World Space → Eye Space

Object Transform → Viewing Transform → Projection

Clip → Clip Space → Depth Resolution

Viewport Transform

Image → Image Space with Depth → Normalized View Volume

Divide by w

Object Transformation Matrix (review, see tutorial 3 and class notes):
series of rotations, scales, shears, and translations as a single matrix

Camera Transform (review, see tutorial 3 and class notes):
inverse of the matrix that brings camera to its world space position, followed by a z reflection, since the camera has a left handed coordinate system. This is equivalent to translation and followed by a change of basis to the camera’s space.

Projection (review, see tutorial 3 and class notes):
warps viewing frustum to a rectangular volume
Orthographic: scale in three dimensions
Perspective: depth dependent scale for x and y, nonlinear change in z
end result for visible points: \[ x \in [-w, w] \]
\[ y \in [-w, w] \]
\[ z \in [0, w] \]

Clipping: clip all polygons to rectangular view volume

straightforward extension of 2D clipping (see next section)

Divide by w: change to normalized view volume
\[ x \in [-1, 1] \]
\[ y \in [-1, 1] \]
\[ z \in [0, 1] \]

Viewport Transform: translation and scale to fit window

Resolve Visibility (see class notes and tutorial 2):

z-buffer, polygon scan conversion, bsp tree, etc.

3D Clipping:

Why not before projection?
If projection is orthographic, it would require clipping by axis
aligned planes, no additional expense.
If projection is perspective, it would require clipping by arbitrary
planes in space, more expensive
Choose after projection for general efficiency.

Why not after divide by w?
After this operation, points behind us are reflected through eye;
many can become unintentionally visible.
Consider the visible eye space point \( p_v = [-x, -y, 1, 1]^{\top} \) and the eye
space point \( p_n = [x, y, -1, 1]^{\top} \), behind us, along with the simple
perspective projection matrix:

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

After projection
\( p_v' = [-x, -y, 1/d, 1]^{\top} \)
\( p_n' = [x, y, -1, -1/d]^{\top} \)
After divide by w, we have the same point:
\[ p_v' = [-x_d, -y_d, d, 1]^T \]
\[ p_b' = [-x_d, -y_d, d, 1]^T \]

Extension of 2D Clipping:
- Bitwise labeling of points: Extend to six bits. Additional two bits represent \( z < 0 \) and \( z > w \). Use the same rejection and acceptance tests.
- Clip against six axis-aligned planes instead of 4:
  \[ x = -w, x = +w, y = -w, y = w, z = 0, z = +w \]

OpenGL Pipeline:
- Modelview Matrix: object transform and eye space transform, see tutorial 3 notes for OpenGL commands
- Projection Matrix: projection only, see tutorial 3 notes.
- Clipping: internal to OpenGL
- Divide by w: internal to OpenGL
- Viewport Transform: calculated internally based on call to glViewport
- Visibility Resolution: internal, typically z-buffer