Perspective Projections, OpenGL Viewing, 3D Clipping

CS116A
Chris Pollett
Dec 1, 2004.
Outline

- Perspective projections
- OpenGL Viewing
- 3D Clipping
Vanishing Points for Perspective Projections

• A point is a perspective scene where all lines not parallel to the view plane intersect is called a vanishing point.

• When the set of lines is parallel to one of the axes then vanishing point is called a principle vanishing point.

• Can have 1, 2, 3 vanishing points and we can control this by the position of the projection plane.
View Volume

- We can create a view volume by specifying a rectangular clipping window on the view plane.
- Bounding planes are now not parallel. Get a shape called a pyramid of vision and can truncate this by specifying near and far planes to get a frustrum.
Perspective Projection Transformation Matrix

• Want to use our equations for $x_p$, $y_p$ and $z_p$ of last day to get a matrix.
• Need to use homogeneous coordinates:
  • Let $x_p = x_h/h$, and $y_p = y_h/h$ where $h = z_{prp} - z$.
  • Here $x_h$ and $y_h$ are obtained from $x$ and $y$ as
      • $x_h = x(z_{prp} - z_{vp}) + x_{prp}(z_{vp} - z)$
      • $y_h = y(z_{prp} - z_{vp}) + y_{prp}(z_{vp} - z)$. 
Matrix

- Now can map $(x, y, z, 1)$ to the projected point in homogeneous coordinates $(x_h, y_h, z_h, 1)$ with the matrix:

$$\begin{bmatrix}
  z_{prp} - z_{vp} & 0 & -x_{prp} & x_{prp}\cdot z_{prp} \\
  0 & z_{prp} - z_{vp} & -y_{prp} & y_{prp}\cdot z_{prp} \\
  0 & 0 & z_{sz} & t_{z} \\
  0 & 0 & -1 & z_{prp}
\end{bmatrix}$$

- Here $z_{sz}$ and $t_{z}$ are scaling and translation factors to go to normalized cube
Symmetric Perspective Projection Frustrum

- The line from the projection reference point through the center of the clipping window, and through the view volume, is called the centerline.
- If this is perpendicular to the view plane then we have a symmetric frustrum.
- In this case, can say have a field of view angle. Have $z_{prp} - z_{vp} = \text{width} \times \cot(\theta/2)/2 \times \text{aspect}$
Normalized Perspective-Projection Transformation Coordinates

• To get into our normalized cube we have already done a scaling in z axis. To get the x and y coordinates in range need to apply a scaling.

\[
\begin{bmatrix}
\text{sx} & 0 & 0 & 0 \\
0 & \text{sy} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

• The equations for these scaling are \( \text{sx} = 2/\text{xwmax-xwmin} \) for x, similar for y. \( \text{sz} = z\text{near}+z\text{far}/z\text{near}-z\text{far} \) and \( \text{tz} = 2*z\text{near}*z\text{far}/(z\text{near}-z\text{far}) \)
3D Screen Coordinates

- We now want to map info to screen coordinates.
- However, still want to keep z-info around (now normalized between 0 and -1), so it can be used in surface removal, etc.
- Map to viewport is thus kept as 3D and given by:

\[
\begin{bmatrix}
\frac{x_{\text{max}} - x_{\text{min}}}{2} & 0 & 0 & \frac{x_{\text{max}} + x_{\text{min}}}{2} \\
0 & \frac{y_{\text{max}} - y_{\text{min}}}{2} & 0 & \frac{y_{\text{max}} + y_{\text{min}}}{2} \\
0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
OpenGL 3D Viewing Functions

• To specify where in world coordinates to look at a scene need to enter view matrix mode:
  
glMatrixMode(GL_MODELVIEW);
• Then can do:
  
gluLookAt(x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
• The last three are the direction of view-up.
OpenGL Orthogonal Projection Function

- Must be in projection mode to set up projection matrices. So do:
  \[ \text{glMatrixMode(GL\_PROJECTION);} \]
- To set up an orthogonal projection can do:
  \[ \text{glOrtho(xwmin, xwmax, ywmin, ywmax, dnear, dfar);} \]
  each parameter is a double.
- Near plane is also the view plane.
- If dfar is 55 then point with z value < -55 clipped.
- Default parameters are -1 or 1 for each of the parameters listed above.
- No OpenGL function for oblique projections
OpenGl Symmetric Perspective-Projection Function

- The GLU function:
  
  \[ \text{gluPerspective}(\text{theta}, \text{aspect}, \text{dnear}, \text{dfar}); \]
  
  with each parameter a double sets up a symmetric, perspective projection.

- The angle can be between 0 and 180.

- The aspect specifies the width/height rations
OpenGL General Perspective Projection Function

- To specify a perspective projection can use:
  \texttt{glFrustrum(xwmin, xwmax, ywmin, ywmax, dnear, dfar);}
- Numbers are double precision floats.
- Near and far clipping distances must be positive.
- The first four parameters say the coordinates of the clipping window on near plane.
- The clipping window can be specified anywhere on the near plane. So if xwmin = -xwmax and ywmin = - ywmax then get symmetric frustrum.
- If do not invoke projection command get orthogonal projection.
OpenGL Viewports and Display Windows

• Finally setting the size of the viewport that projected points will appear in is specified in the same way as in the 2D case:
  
  \[
  \text{glViewport(xvmin, yvmin, xvmax, yvmax);} \]

3D Clipping

• As in the 2D case, in the 3D there are advantages to having normalized cube before clipping:
  – All device independent transformations are carried out before applying any clipping.
  – Each clip plane is parallel to one of the 3 axes regardless of the original shape of the view volume so can be optimized.

• Common choice of cubes are the **unit cube** which has extents between 0 and 1 and the **symmetric cube** has extents between -1 and 1.
Clipping in 3D Homogeneous Coordinates

• In homogeneous coordinates \((x,y,z)\) gets converted to \((x,y,z, 1)\).

• After all our transformations and projections might have \((x_h, y_h, z_h, h)\) where \(h\) is not 1. (Might happen because of perspective transformation).

• If divided away the factor \(h\), would lose precision, so this is why want to do clipping in homogeneous coordinates
3D Regions Codes

• The concept of region code used in Cohen-Sutherland clipping can be extended to 3D. Need to use a 6 bit number now for all the regions:
  – bit 6 - far, bit 5 - near, bit 4 - top, bit 3-bottom, bit 2 -right, bit 1 -left

• Conditions for setting bit same as in 2D case but now have 2 extra bits to set for each point.
Assigning Bit Values

• Suppose we have a point \((x,y,z,h)\). Then
  
  bit 1 = 1 if \(h+x < 0\)
  
  bit 2 = 1 if \(h-x < 0\)
  
  bit 3 = 1 if \(h+y < 0\)
  
  bit 4 = 1 if \(h-y < 0\)
  
  bit 5 = 1 if \(h+z < 0\)
  
  bit 6 = 1 if \(h-z < 0\)
3D Point and Line Clipping

- A point is within the view volume if its region code is 000000. So this gives us an easy way to clip points.
- For lines, we can clip the whole line if when we AND the endpoint codes we get a 1 in the same bit position. We can accept the whole line if when we OR the codes we get 000000.
- Otherwise, we need to analyze the part of the line that needs to be saved.
More line clipping

• Suppose the endpoint of our line are: P1=(x1,y1,z1,h1) and P2=(x2,y2,z2,h2).
• Can write points on line segment with P=P1+(P2-P1)u for u between 0 and 1.
• Look for bits in the region code that are not the same. Know boundary crossed, and also in which coordinates. For example, maybe x.
• Then can solve for u to find point of intersection.
• If for instance crosed xmax = 1. Then know intersection point x/h must equal 1. So get: u=x1-h/((x1-h1) - (x2-h2))
3D Polygon Clipping

• Say want to intersect a tetrahedron with our view volume.
• First check if its coordinate extents lie completely within the view volume or if its coordinates lie completely outside one of the clipping boundaries.
• If not, go through each edge in the object, clip and to obtain a new vertex lists, edge lists for the clipped object.
• Then might have to add new faces to our face list
• If object is made of triangle strips process easier as can then use Sutherland-Hodgman
3D Curve Clipping

- First check if the coordinate extents of curved object are completely inside the view volume.
- Then check if object is completely outside any one of the six clipping planes.
- If this accept/reject test fails, then we locate intersections with clipping plane.
- This involves solving simultaneous surface and clipping plane equations.
- This can be hard so polygon patches are often used to approximate curved surfaces.
Arbitrary Clipping Planes

• Might also want to clip to arbitrary planes.
• Might be used for cross-sectional view.
• Can specify a plane with $Ax+By+Cz +D =0$
• Objects behind the plane, for instance point $(x,y,z)$ with $Ax+By+Cz+D <0$ are the ones that are usually clipped.
OpenGL Optional Clipping Planes

- One can specify additional clipping planes for a scene than those of the view volume with:
  ```
  GLfloat planeCoeffs[] = {1.0, 2.0, 3.0, 4.0};
  glClipPlane(GL_CLIP_PLANE0, planeCoeffs);
  ```

- To use this plane can use `glEnable(GL_CLIP_PLANE0);` and `glDisable` to stop using.

- There are also planes 1, 2… To find out how many use:
  ```
  glGetIntegerv(GL_MAX_CLIP_PLANES, numPlanes);
  ```