Completing the graphics pipeline and 3D clipping

Richard (Hao) Zhang

Introduction to Computer Graphics
CMPT 361 – Lecture 9
Perspective normalization

- Frustum() maps objects from the perspective view volume to the **left-handed** canonical view volume.

- The projection matrix is:

\[
p' = \begin{bmatrix}
\frac{2 \cdot \text{near}}{\text{right} - \text{left}} & 0 & \frac{\text{right} + \text{left}}{\text{top} + \text{bottom}} & 0 \\
0 & \frac{2 \cdot \text{near}}{\text{top} - \text{bottom}} & \frac{\text{top} - \text{bottom}}{\text{far} - \text{near}} & 0 \\
0 & 0 & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} & -1 \\
0 & 0 & \frac{2 \cdot \text{far \cdot near}}{\text{far} - \text{near}} & 0
\end{bmatrix}
\]

- The midpoint \( M \) at \(-\text{(near+far)}/2\) is mapped to \((\text{far–near})/(\text{far+near})\), not 0, for foreshortening.
Don’t forget perspective divide

Expression for the $y$ (respectively, $x$) component in the canonical view volume is really $-y'/z$ (respectively, $-x'/z$)

Need division of $-z$ to normalize $w$ component — this is called perspective division

Points with $z = 0$ are mapped to point at infinity
Where did the view plane go?

- We never specified it …
- Function LookAt() specifies the camera position
- Ortho() and Frustrum() do projection normalization
- When we are in the canonical view volume, just project by ignoring the z values (after visibility)
- Important: LookAt() can be superseded by projection
  - It depends on where the near clipping plane is placed
  - One may “see” behind the camera
Example

LookAt(0.0, 0.0, 0.0, 0.0, 0.0, −10.0, 0.0, 1.0, 0.0); ...

Ortho(−1.0, 1.0, −1.0, 1.0, 1.0, 30.0); ...

with a square in the plane \( z = 1.0 \) which is behind the camera — it is clipped out by near and far planes at −1.0 and −30.0 — **do not see**

\[
\begin{array}{c}
\text{z} \\
\hline
\text{square at 1.0} \\
\hline
\text{eye} \\
\hline
\text{near plane −1.0} \\
\hline
\text{far plane −30.0}
\end{array}
\]
Seeing behind the camera

- Ortho\((-1.0, 1.0, -1.0, 1.0, -2.0, 30.0)\)

  The near clipping plane +2.0 is behind the camera and also the square.

  ** Although the square is also behind the camera, it is mapped into the canonical cube nevertheless and we *can “see” it*. 

  If we change \(-2.0\) to \(-0.5\), we will not be able to see the square since it is clipped out by the near clipping plane. 

Now change Ortho() all to Frustum(), strange behavior!

- No at 1.0
- Eye
- Near plane –1.0
- Far plane –30.0
  - NO

- Near plane +2.0
- Eye
- Far plane –30.0
  - NO!

- Near plane +0.5
- Eye
- Far plane –30.0
  - Yes!
Positioning of the near plane

- OpenGL has problems in **clipping things behind the camera with perspective projection**

- It is your responsibility to make sure that the near clipping plane is **in front of** the camera (positive “near”)

- Avoid placing near clipping plane behind the camera: why mess it up? 😊
The graphics pipeline

- We have been focusing on the first three boxes
- An expanded view ...
An expanded view

Objects in OCS → Modeling transform → WCS (4D) → Viewing transform → VCS (4D)

Pixels = SCS (2D)

WinCS (3D) → Viewport transform → NDCS (3D) → clipping then perspective ‘/’ → CCS (4D)

OCS: object coordinate system
WCS: world coordinate system
VCS: viewing coordinate system
CCS: clip coordinate system
NDCS: normalized device CS
WinCS: window coordinate system
SCS: Screen coordinate system

February 19, 2018
## OpenGL transformation functions

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Functions/Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modeling transform</strong></td>
<td>Translate(), RotateX(), RotataY(), RotateZ(), Scale() in mat.h</td>
</tr>
<tr>
<td><strong>Viewing transform</strong></td>
<td>LookAt() in mat.h</td>
</tr>
<tr>
<td><strong>Projection transform</strong></td>
<td>Frustum(), Ortho(), Perspective() in mat.h</td>
</tr>
<tr>
<td><strong>Viewport transform</strong></td>
<td>glViewport(x, y, width, height)</td>
</tr>
</tbody>
</table>
Viewport transformation

- Do this after orthographic projection in the canonical view volume: NDCS $\rightarrow$ WinCS

\[
\frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} = \frac{x_v - x_{v\text{min}}}{x_{v\text{max}} - x_{v\text{min}}}
\]
3D clipping

- The Cohen-Sutherland line clipping algorithms and the Sutherland-Hodgeman polygon clipping algorithm work in pretty much the same way

  - 6-bit outcodes instead of 4-bit
  - Line-line intersection in 2D → line-plane intersection in 3D
Line-plane intersection

- Line: \( \mathbf{p}(\alpha) = (1 - \alpha)\mathbf{p}_1 + \alpha \mathbf{p}_2 \)
- Plane: \( \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0 \)
- To intersect: \( \mathbf{n} \cdot (\mathbf{p}(\alpha) - \mathbf{p}_0) = 0 \)
- So we have
  \[
  \alpha = \frac{\mathbf{n} \cdot (\mathbf{p}_0 - \mathbf{p}_1)}{\mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1)}
  \]

- Six multiplications and one division in general
- How about clipping against canonical view volume? – only one division and more efficient
When can clipping be done?

- Clipping in VCS: correct but expensive computations
  - Clipping in CCS (before perspective division)
    - Clip in homogeneous coordinates – correct but rather tricky
  - Clipping in NDCS (after perspective division)
    - Most efficient but be aware of case where \( w < 0 \)
Problem with $w < 0$

- Seeing behind the camera with Frustum()

```
z
square at 1.0

z
near plane +2.0

z
near plane +0.5
```

- eye
- near plane $-1.0$
- far plane $-30.0$

NO

- NO!

- YES!
Situation 2 (aside)

near plane +2.0

\[ z \quad \text{eye} \quad \text{far plane } -30.0 \quad \text{NO!} \]

\[
\begin{bmatrix}
- & - & - & - & - \\
- & - & - & - & - \\
0 & 0 & -\frac{28}{32} & \frac{120}{32} & 0 \\
0 & 0 & -1 & \frac{32}{32} & 0 \\
0 & 0 & -1 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
- \\
- \\
1 \\
1 \\
1 \\
\end{bmatrix}
=
\begin{bmatrix}
- \\
- \\
\frac{92}{32} \\
\frac{32}{32} \\
-1 \\
\end{bmatrix}
=
\begin{bmatrix}
- \\
- \\
-2.875 \\
1 \\
1 \\
\end{bmatrix}
\]

-2.875 is outside of canonical cube: z in [-1, +1]
Situation 3 (aside)

near plane +0.5

z

eye

far plane −30.0

YES!

\[-0.0164\] is \textbf{inside} of canonical cube: \(z\) in \([-1, +1]\)

\[
\begin{bmatrix}
- & - & - & - & - \\
- & - & - & - & - \\
0 & 0 & -29.5 & 30 & 30.5 \\
0 & 0 & 30.5 & 30.5 & 0 \\
0 & 0 & -1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
- \\
- \\
1 \\
1
\end{bmatrix}
= \begin{bmatrix}
- & - & - & - & - \\
0.5 & 30.5 & -1 & -1 & -1
\end{bmatrix}
\begin{bmatrix}
- \\
- \\
- \\
- \\
- \end{bmatrix}
= \begin{bmatrix}
- \\
-0.0164 \\
1
\end{bmatrix}
\]
Problem with line clipping in NDCS

- Perspective division may create a problem: depth information may get lost if w component $< 0$

- Example:

<table>
<thead>
<tr>
<th></th>
<th>VCS</th>
<th>CCS</th>
<th>NDCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>(1, 0, -2)</td>
<td>(1, 0, 2/3, 2)</td>
<td>(1/2, 0, 1/3)</td>
</tr>
<tr>
<td>$P_2$</td>
<td>(0, 0, 2)</td>
<td>(0, 0, -6, -2)</td>
<td>(0, 0, 3)</td>
</tr>
</tbody>
</table>
Various solutions

- Clip all in VCS – “safest” but expensive
- Clip in CCS using J. Blinn’s technique [78 paper]
- Alternatively, note that
  - Negative \( w \) values occur only for points with \( z > 0 \)
  - Depth order preserved in NDCS for points with negative \( z \) in VCS

So clip out points behind COP in VCS and then project and clip in NDCS. Or do not create content behind eye.
Application of projection: shadows

- Essential component for realistic rendering
- Naturally use projections
  - Can produce **hard** shadows
  - Only handles shadows on a plane
  - To shadow on a polygonal face, need **clipping**
- More advanced shadow algorithms exist, e.g., soft shadows are not easy to do
Shadow polygon: parallel projection

- **Shadow polygon** is obtained via projection
  the center of projection is a light source
- Project shadow on \( z = 0 \)
- Light at \( \propto \) (directional light)
- Derive the projection matrix

\[
\begin{bmatrix}
 x_s \\
 y_s \\
 0 \\
 1 \\
\end{bmatrix}
= 
\begin{bmatrix}
 x_p \\
 y_p \\
 z_p \\
 1 \\
\end{bmatrix}
\]
Shadow polygon: parallel projection

- Project shadow on $z = 0$ with light at $\alpha$ (**directional light**)

$$(x_p, y_p, z_p) - (x_s, y_s, 0) = t \, (x_L, y_L, z_L), \text{ then solve}$$
Shadows: perspective projection

- Project on plane $z = 0$
- **Point light source**
- Derive the matrix

\[
\begin{bmatrix}
    x_s \\
    y_s \\
    0 \\
    1
\end{bmatrix}
= \begin{bmatrix}
    \cdot & \cdot & \cdot & \cdot \\
    \cdot & \cdot & \cdot & \cdot \\
    \cdot & \cdot & \cdot & \cdot \\
    \cdot & \cdot & \cdot & \cdot \\
\end{bmatrix}
\begin{bmatrix}
    x_p \\
    y_p \\
    z_p \\
    1
\end{bmatrix}
\]
Shadows: perspective projection

- Project on plane $z = 0$
- **Point light source**
- Let us derive the matrix

\[
\begin{bmatrix}
  x_s \\
  y_s \\
  0 \\
  1
\end{bmatrix} =
\begin{bmatrix}
  -z_L & 0 & x_L & 0 \\
  0 & -z_L & y_L & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 1 & -z_L
\end{bmatrix}
\begin{bmatrix}
  x_p \\
  y_p \\
  z_p \\
  1
\end{bmatrix}
\]

\[
(x_p, y_p, z_p) - (x_s, y_s, 0) = t \left[(x_L, y_L, z_L) - (x_p, y_p, z_p)\right], \text{ then solve}
\]

We have

\[
t = z_p / (z_L - z_p), \quad x_s = (1 + t)x_p - tx_L
\]
Shadow of a teapot