CS380: Computer Graphics

Clipping and Culling

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/CG/
Class Objectives

- Understand clipping and culling
- Understand view-frustum, back-face culling, and hierarchical culling methods
- Know various possibilities to perform culling and clipping in the rendering pipeline
Culling and Clipping

- **Culling**
  - Throws away entire objects and primitives that cannot possibly be visible
  - An important rendering optimization (esp. for large models)

- **Clipping**
  - “Clips off” the visible portion of a primitive
  - Simplifies rasterization
  - Also, used to create “cut-away” views of a model
Culling Example

Power plant model
(12 million triangles)
Culling Example

Full model
12 Mtris

View frustum culling
10 Mtris

Occulsion culling
1 Mtris
Implicit equation for line (plane):

\[ n_x x + n_y y - d = 0 \]

\[
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\begin{bmatrix}
  n_x & n_y & -d
\end{bmatrix}
= 0 \implies \mathbf{T} \cdot \mathbf{p} = 0
\]

If \( \mathbf{n} \) is normalized then \( d \) gives the distance of the line (plane) from the origin along \( \mathbf{n} \).
Lines and Planes

- Lines (planes) partition 2D (3D) space:
  - Positive and negative *half-spaces*

- The intersection of negative half-spaces defines a convex region
Testing Objects for Containment

Outside

Straddling

Inside
Conservative Testing

- Use cheap, conservative bounds for trivial cases
- Can use more accurate, more expensive tests for ambiguous cases if needed
Hierarchical Culling

- Bounding volume hierarchies accelerate culling by rejecting/accepting entire sub-trees at a time

- Bounding volume hierarchies (BVHs)
  - Object partitioning hierarchies
  - Uses axis-aligned bounding boxes
Hierarchical Culling

- Simple algorithm:
  while (node is indeterminate) recurse on child
View Frustum Culling

- Test objects against planes defining view frustum
- How do you compute them?

- Other planes can be computed similarly
Back-Face Culling

- Special case of occlusion - convex self-occlusion
  - For closed objects (has well-defined inside and outside) some parts of the surface must be blocked by other parts of the surface
- Specifically, the backside of the object is not visible
Face Plane Test

- Compute the plane for the face:
  \[ n = (\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0) \]
  \[ d = n \cdot \mathbf{v}_0 \]

- Cull if eye point in the negative half-space
Back-Face Culling in OpenGL

- Can cull front faces or back faces
- Back-face culling can sometimes double performance

```python
if (cull):
    glEnable(GL_CULL_FACE)  # enable Culling
    glCullFace(GL_BACK)      # which faces to cull
    glFrontFace(GL_CCW)      # define winding order
else:
    glDisable(GL_CULL_FACE)
```

You can also do front-face culling!
Clipping a Line Segment against a Line

- First check endpoints against the plane
  - If they are on the same side, no clipping is needed
- Interpolate to get new point
  \[ p' = p_0 + t(p_1 - p_0) \quad T \cdot p' = 0 \]

\[ T \cdot (p_0 + t(p_1 - p_0)) = 0 \]

\[ t = \frac{-(T \cdot p_0)}{T \cdot (p_1 - p_0)} \]

- Vertex attributes interpolated the same way
Clipping a Polygon against a Line

- Traverse edges
- Keep edges that are entirely inside
- Create new point when we exit
- Throw away edges entirely outside
- Create new point and new edge when we enter
Clipping against a Convex Region

- Sutherland-Hodgman
  - Just clip against one edge at a time
Outcodes

- The Cohen-Sutherland clipping algorithm uses **outcodes** to quickly determine the visibility of a primitive
- An outcode is created for each vertex
  - It is a bit vector with a bit set for each plane the vertex is outside of
- Works for any convex region
Outcode for Lines

(outcode1 OR outcode2) == 0
- line segment is inside

(outcode1 AND outcode2) != 0
- line segment is totally outside

(outcode1 AND outcode2) == 0
- line segment potentially crosses clip region at planes indicated by set bits in
  (outcode1 XOR outcode2)

● False positive
  ● Some line segments that are classified as potentially crossing the clip region actually don’t
Outcodes for Triangles

Combine outcodes from vertices

(outcode1 OR outcode2 OR outcode3) == 0
triangle is inside

(outcode1 AND outcode2 AND outcode3) != 0
triangle is outside

(outcode1 AND outcode2 AND outcode3) == 0
triangle potentially crosses clip region
Clipping in the Pipeline

Clip space
View Frustum Clipping

- Points in projective space need to be clipped before projection.
- Primitives that straddle the $z=0$ plane "flip" around infinity when projected.
**Clipping in the Clip Space**

- **NDC simplify view frustum clipping**
- **Clip after applying projection matrix, but before the divide by w**
  - clip coordinates

\[-W < X < W\]

\[W = -X\]

\[W = X\]

\[v_0\]  

\[v_1\]

\[v_i = [x_i \ w_i \ 1]^T\]

\[T_{x^+} = [1 \quad -1 \quad 0]\]

\[t = \frac{w_0 - x_0}{(w_0 - x_0) - (w_i - x_i)}\]

- **Easy in/out test and interpolation**
Culling and Clipping in the Rendering Pipeline

View frustum culling

View frustum clipping and back-face culling can be done here

Back-face culling done in setup phase of rasterization
Class Objectives were:

- Understand clipping and culling
- Understand view-frustum, back-face culling, and hierarchical culling methods
- Know various possibilities to perform culling and clipping in the rendering pipeline
Reading Assignment

- Read the chapter “Raster Algorithms”
Next Time

- Triangulating a polygon
- Rasterizing triangles
- Interpolating parameters