Clipping

Cohen-Sutherland Line Clipping

- outcodes
- 4 flags encoding position of a point relative to top, bottom, left, and right boundary
- $OC(p_1) = 0010$
- $OC(p_2) = 0000$
- $OC(p_3) = 1001$

Trivial Accepts

- big optimization: trivial accept/rejects
- Q: how can we quickly determine whether a line segment is entirely inside the viewport?
- A: test both endpoints

Trivial Rejects

- Q: how can we know a line is outside viewport?
- A: if both endpoints on wrong side of same edge, can trivially reject line

Cohen-Sutherland Line Clipping

- assign outcode to each vertex of line to test
- line segment: $(p_1, p_2)$
- trivial cases
- $OC(p_1) = 0 \& OC(p_2) = 0$
- both points inside window, thus line segment completely visible (trivial accept)
- $(OC(p_1) \& OC(p_2)) = 0$
- there is (at least) one boundary for which both points are outside the same flag set in both outcodes
- thus line segment completely outside window (trivial reject)
- if line cannot be trivially accepted or rejected, subdivide so that one or both segments can be discarded
- pick an edge that the line crosses (how?)
- intersect line with edge (how?)
- discard portion on wrong side of edge and assign outcode to new vertex
- apply trivial accept/reject tests; repeat if necessary

Why Clip?

- bad idea to rasterize outside of framebuffer bounds
- also, don’t waste time scan converting pixels outside window
- could be billions of pixels for very close objects!

Line Clipping

- 2D
- determine portion of line inside an axis-aligned rectangle (screen or window)
- 3D
- determine portion of line inside axis-aligned parallelepiped (viewing frustum in NDC)
- simple extension to 2D algorithms

Clipping Lines To Viewport

- combining trivial accepts/rejects
- trivially accept lines with both endpoints inside all edges of the viewport
- trivially reject lines with both endpoints outside the same edge of the viewport
- otherwise, reduce to trivial cases by splitting into two segments

Clipping

- naïve approach to clipping lines:
  - for each line segment
  - find intersection point
  - pick “nearest” point
  - if anything is left, draw it
- what do we mean by “nearest”?
- how can we optimize this?

Next Topic: Clipping

- we’ve been assuming that all primitives (lines, triangles, polygons) lie entirely within the viewport
- in general, this assumption will not hold:

Reading for Clipping

- FCG Sec 8.1.3-8.1.6 Clipping
- FCG Sec 8.4 Culling
- (12.1-12.4 2nd ed)
Cohen-Sutherland Line Clipping

- intersect line with edge

Sutherland-Hodgeman Clipping

- basic idea:
- consider each edge of the viewport individually
- clip the polygon against the edge equation
- after doing all edges, the polygon is fully clipped

Line Clipping in 3D

- approach
- clip against parallelepiped in NDC
- after perspective transform
- means that clipping volume always the same
- $x_{min} = y_{min} = -1, x_{max} = y_{max} = 1$ in OpenGL
- boundary lines become boundary planes
- but outcodes still work the same way
- additional front and back clipping plane
- $z_{min} = -1, z_{max} = 1$ in OpenGL

Polygon Clipping

- objective
- 2D: clip polygon against rectangular window
- or general convex polygons
- extensions for non-convex or general polygons
- 3D: clip polygon against parallelepiped

Viewport Intersection Code

- $(x_1, y_1), (x_2, y_2)$ intersect vertical edge at $x_{right}$
- $y_{intersect} = y_1 + m(x_{right} - x_1)$
- $m = (y_2 - y_1) / (x_2 - x_1)$

- $(x_1, y_1), (x_2, y_2)$ intersect horiz edge at $y_{bottom}$
- $x_{intersect} = x_1 + (y_{bottom} - y_1) / m$
- $m = (y_2 - y_1) / (x_2 - x_1)$

Why Is Clipping Hard?

- a really tough case:
- concave polygon to multiple polygons
- classes of polygons
- triangles
- convex
- concave
- holes and self-intersection

Sutherland-Hodgeman Clipping

- basic idea:
- consider each edge of the viewport individually
- clip the polygon against the edge equation
- after doing all edges, the polygon is fully clipped

Why Is Clipping Hard?

- what happens to a triangle during clipping?
- some possible outcomes:
- triangle to triangle
- triangle to quad
- triangle to 5-gon
- how many sides can result from a triangle?
- seven

Sutherland-Hodgeman Clipping

- basic idea:
- consider each edge of the viewport individually
- clip the polygon against the edge equation
- after doing all edges, the polygon is fully clipped

Polygon Clipping

- not just clipping all boundary lines
- may have to introduce new line segments

Cohen-Sutherland Discussion

- key concepts
- use opcodes to quickly eliminate/include lines
- best algorithm when trivial accepts/rejects are common
- must compute viewport clipping of remaining lines
- non-trivial clipping cost
- redundant clipping of some lines
- basic idea, more efficient algorithms exist
Sutherland-Hodgeman Clipping

- basic idea:
  - consider each edge of the viewport individually
  - clip the polygon against the edge equation
  - after doing all edges, the polygon is fully clipped

Sutherland-Hodgeman Algorithm

- input/output for whole algorithm
  - input: list of polygon vertices in order
  - output: list of clipped polygon vertices consisting of old vertices (maybe) and new vertices (maybe)
- input/output for each step
  - input: list of vertices
  - output: list of vertices, possibly with changes
- basic routine
  - go around polygon one vertex at a time
  - decide what to do based on 4 possibilities
    - is vertex inside or outside?
    - is previous vertex inside or outside?

Clipping Against One Edge

- \( p[i] \) inside: 2 cases
  - \( p[0] \) inside \( \Rightarrow \) \( p[i] \) \( \Rightarrow \) \( p[i] \)
  - \( p[i-1] \) inside \( \Rightarrow \) \( p[i] \) \( \Rightarrow \) \( p[i] \)
- \( p[i] \) outside: 2 cases
  - \( p[0] \) outside \( \Rightarrow \) \( p[i] \) \( \Rightarrow \) \( p[i] \)
  - \( p[i-1] \) inside \( \Rightarrow \) \( p[i] \) \( \Rightarrow \) \( p[i] \)

Clipping Against One Edge

\[
\text{clipPolygonToEdge}( p[n], \text{edge} ) \{
    \text{for( } i= 0 ; i< n ; i++ \text{) } \{
        \text{if( } p[i] \text{ inside edge } \) \{
            \text{if( } p[i-1] \text{ inside edge } \) \{
                \text{output } p[i];
            \text{else (} \text{p= intersect( } p[i-1], p[i], \text{edge}) \text{)}
                \text{output } p, p[i];
        \text{else (} \text{p= intersect( } p[i-1], p[i], \text{edge}) \text{)}
            \text{output nothing}.
        \}
    \}
\}
\]