Clipping

CMPT 361
Introduction to Computer Graphics
Torsten Möller
Rendering Pipeline

Hardware

Modelling → Transform → Visibility

Illumination + Shading

Perception, Interaction

Color

Texture/Realism
Reading

• Angel – Chapter 6.1 – 6.7
• Foley et al – Chapter 3
Clipping

• Clipping is important in order to avoid expensive computations, like
  – illumination
  – visibility determination
  – rasterization, etc.
Clipping (2)

• Can be done
  – before scan conversion: analytically (floating point intersections), so all generated pixels will be visible
  – during scan conversion: scissoring
    • simplest, brute-force technique
    • scan-convert entire primitive and write only visible pixels
    • may be fast in certain circumstances, if bounds check can be done quickly
Clipping (3)

• Can be done
  – after scan conversion: create the entire canvas in memory and display only visible parts (used for panning large static scenes, text)
Clipping Lines

- Let's consider clipping lines against a rectangle

![Diagram](image)
Lines - Endpoints

- If the endpoints of a line are within the clipping rectangle, then the line is completely inside (trivially accepted)
- If 1 endpoint is inside and the other is outside then we MUST compute the point of intersection
- If both endpoints are outside then the line may or may not be inside
Lines - Endpoints (2)

• good approach will eliminate trivial acceptances/rejections quickly and devote time to those lines which actually intersect the clipping rectangle

• Consider the following methods:
  – Analytical (solve simultaneous equations)
  – Cohen-Sutherland algorithm
  – Cyrus-Beck algorithm (Liang-Barsky)
Simultaneous Equations

• Brute force:
  – intersect the line with each of the 4 clip edges \((x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}})\)
  – test these intersection points to see if they occur on the edges of the clipping rectangle
Simultaneous Equations (2)

• done with a parametric line equation

\[ p = p_0 + t(p_1 - p_0) \]

• if \( t \) is between [0,1] then an intersection exists.

• Be careful of parallel lines!

• This method is inefficient!
Cohen-Sutherland

- performing initial tests on the endpoints
- avoid expensive intersection calculations
- If a line cannot be trivially accepted nor rejected, it is subdivided.
Cohen-Sutherland (2)

- trivially accepted -- both endpoints are inside the clipping rectangle
- trivially rejected -- both points are in the outside halfplane of a particular clip edge

• tests are facilitated by assigning 4-bit codes to the clipping rectangle and the surrounding regions
Cohen-Sutherland (3)

• Code ABRL assigned according to:
  - $A := y > y_{\text{max}}$
  - $B := y < y_{\text{min}}$
  - $R := x > x_{\text{max}}$
  - $L := x < x_{\text{min}}$
Cohen-Sutherland (4)

• For each line:
  – Assign codes to the endpoints
  – Trivially accept if both codes are 0000, display line
  – Perform bitwise AND of codes
  – Trivially reject if result is not 0000, return
  – Choose an endpoint outside the clipping rectangle
  – Test its code to determine which clip edge was crossed and find the intersection of the line and that clip edge (test the edges in a consistent order)
  – Replace endpoint (selected above) with intersection point
  – Repeat
Cohen-Sutherland (5)
• Stay in parametric space (as opposed to x,y-space) to do all computations
Cyrus-Beck (2)

- Point of intersection:

\[ N_i \cdot (P(t) - P_{E_i}) = 0 \]
\[ N_i \cdot (P_0 + t(P_1 - P_0) - P_{E_i}) = 0 \]
\[ N_i \cdot (P_0 - P_{E_i}) + tN_i \cdot (P_1 - P_0) = 0 \]

- if \( D = P_1 - P_0 \):

\[ t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot D} \]

- problem, if \( N_i \cdot D = 0 \) (\( D \) and \( N_i \) are parallel)
Cyrus-Beck (3)

• Applying this algorithm for each edge of a rectangle yields 4 values of the parameter $t$. 

© Machiraju/Zhang/Möller
Cyrus-Beck (4)

• Values outside [0,1] can be ignored,
• classify endpoints:
  – PE (potentially entering)
    • if moving from $P_0$ to $P_1$, an intersection occurs such that the inside halfplane of the edge is entered
  – PL (potentially leaving)
    • if moving from $P_0$ to $P_1$, an intersection occurs such that the outside halfplane of the edge is entered
Cyrus-Beck (5)

• Formally, the intersections can be classified based on the sign of the dot product $N \cdot D$
• Note that $t_E < t_L$
• then we can find the clipped line by $\max(t_E)$ … $\min(t_L)$
Clipping Circles

- test if trivially accept or reject the circle using the circle's extent (a square the size of the circle's diameter)
  - trivial tests fail at the circle level: divide the circle into quadrants and test again
  - trivial tests fail at the quadrant level: divide the quadrants into octants and test again
  - trivial tests still fail, analytically calculate remaining intersection points and scan convert the resulting arcs.
Clipping Polygons

- Can get complex - must handle:
  - multiple components,
  - concave with many exterior edges,
  - simple convex
Clipping Polygons (2)

• Must test each edge against each clipping edge
  – new edges must be added
  – existing edges must be discarded, retained or divided

• can result in one or more polygons
Sutherland-Hodgman

• divide and conquer strategy
• simple problem -- clip a polygon against a single infinite clip edge
• Successively clip polygon against each of the edges comprising the clip rectangle
• algorithm is more general: it can be used to clip a polygon against any convex clipping polygon
Sutherland-Hodgman (2)

• Input: a list of vertices which define (in 2D) the polygon edges

• output: After each pass (for each edge), the algorithm returns a list of vertices defining the (partially) clipped polygon.
Sutherland-Hodgman (3)

(a) Clip rectangle
(b) Right clip boundary
(c) Bottom clip boundary
(d)
(e)

© Machiraju/Zhang/Möller
Sutherland-Hodgman (4)

- examine successive pairs of vertices (labelled s and p)
- 0, 1, or 2 vertices can be added to the output at each step, according to 4 cases
Sutherland-Hodgman (5)

- The algorithm can add extra vertices along the clipping rectangle, which can be removed in a post-processing phase.
Clipping Characters

- Defined as higher-order curves: clip the curve’s
- defined as bitmap fonts: clip on pixel-level
- or faster - clip on character level (draw the whole character or not)