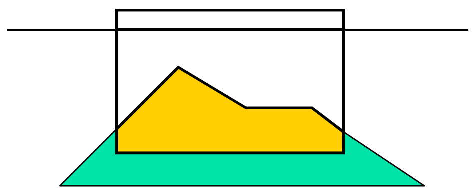
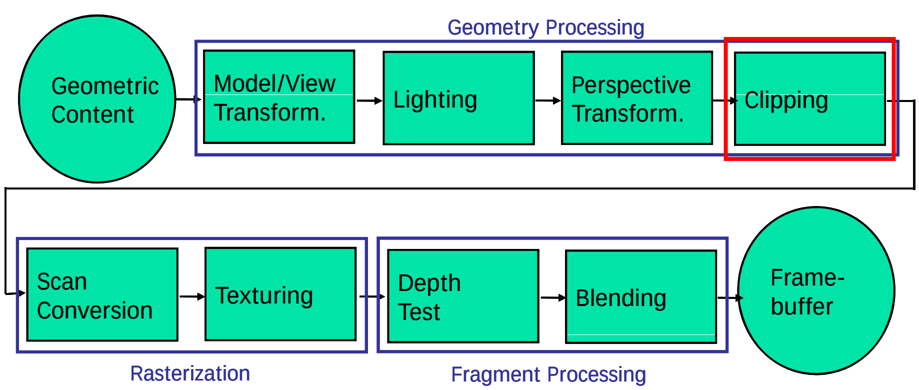
Chapter 7

Clipping



Clipping - 1

Rendering Pipeline



- Discard geometry outside viewport window

Line/Polygon Clipping (2D)

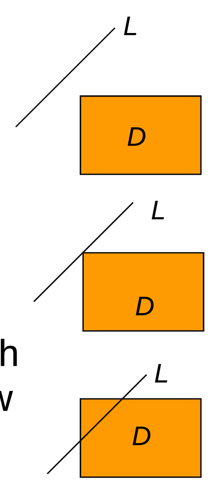
Problem:
Given a 2*D* line/polygon and a window, clip the line/polygon to their regions that are *inside* the window.

- Objectives
  - Efficiency
  - (Parallelization)
- Two approaches
  - Explicit (continuous setting)
  - Implicit (discrete setting) – part of scan conversion

3

Explicit Solution: Line Segments

- *Intersection* of convex regions is convex
  - Why?
- *L* & *D* are *convex* - intersection is convex
  - single connected segment of *L*
- Clipping uses intersections of *L* with four boundary segments of window *D*

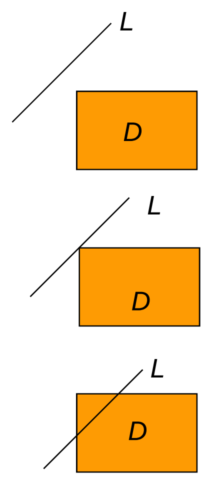


4

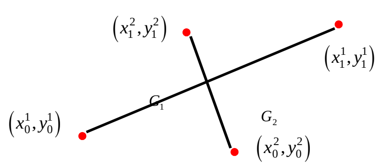
Basic Method

Clip($P_0, P_1, x_{min}, x_{max}, y_{min}, y_{max}$)
if ((P_0 and P_1 inside window) then draw(P_0, P_1);
test if segment (P_0, P_1) intersects any of the edges
if not, return;
else let P_2 be the first intersection found
Clip($P_0, P_2, x_{min}, x_{max}, y_{min}, y_{max}$);
Clip($P_2, P_1, x_{min}, x_{max}, y_{min}, y_{max}$);
end

- Works, but inefficient for lines OUTSIDE D
  - Four intersection tests
- Note: need special care for vertices ON window edges



Segment-Segment Intersection

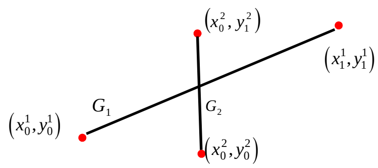


$$G_1 = \begin{cases} x^1(t) = x_0^1 + (x_1^1 - x_0^1)t \\ y^1(t) = y_0^1 + (y_1^1 - y_0^1)t \end{cases} \quad t \in [0,1] \qquad G_2 = \begin{cases} x^2(r) = x_0^2 + (x_1^2 - x_0^2)r \\ y^2(r) = y_0^2 + (y_1^2 - y_0^2)r \end{cases} \quad r \in [0,1]$$

Intersection: *x* & *y* values equal in both representations - two linear equations in two unknowns (*r*,*t*)
test if resulting r & t are inside the [0,1] range

$$x_0^1 + (x_1^1 - x_0^1)t = x_0^2 + (x_1^2 - x_0^2)r$$
$$y_0^1 + (y_1^1 - y_0^1)t = y_0^2 + (y_1^2 - y_0^2)r$$

Intersection with axis-aligned lines



$$G_1 = \begin{cases} x^1(t) = x_0^1 + (x_1^1 - x_0^1)t \\ y^1(t) = y_0^1 + (y_1^1 - y_0^1)t \end{cases} t \in [0,1], G_2 = \begin{cases} x^2(r) = x_0^2 \\ y^2(r) = y_0^2 + (y_1^2 - y_0^2)r \end{cases} r \in [0,1]$$

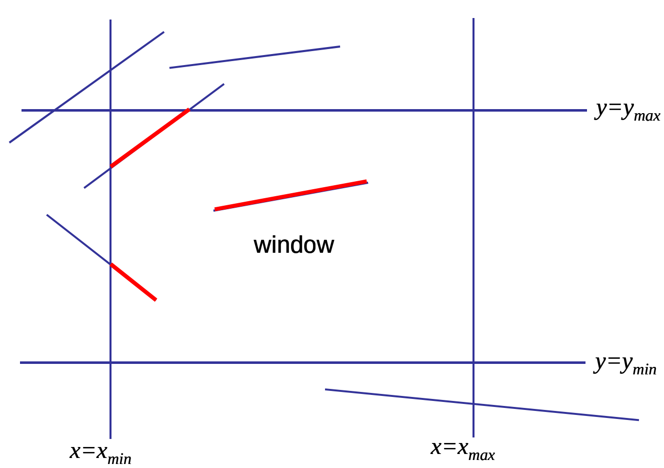
Intersection: *x* & *y* values equal in both representations - two linear equations in two unknowns (*r*,*t*)

$$x_0^1 + (x_1^1 - x_0^1)t = x_0^2$$

$$t = \frac{x_0^2 - x_0^1}{x_1^1 - x_0^1}, \text{if } t < 0 \text{ or } t > 1 \text{ no intersection}$$

$$y_0^1 + (y_1^1 - y_0^1)t = y_0^2 + (y_1^2 - y_0^2)r, \text{(relevant only for segments)}$$

Line Clipping



Cohen-Sutherland Algorithm

Purpose:
Fast treatment of line segments that are trivially inside/outside window.

P = (*x*,*y*) - point to be classified against window *D*

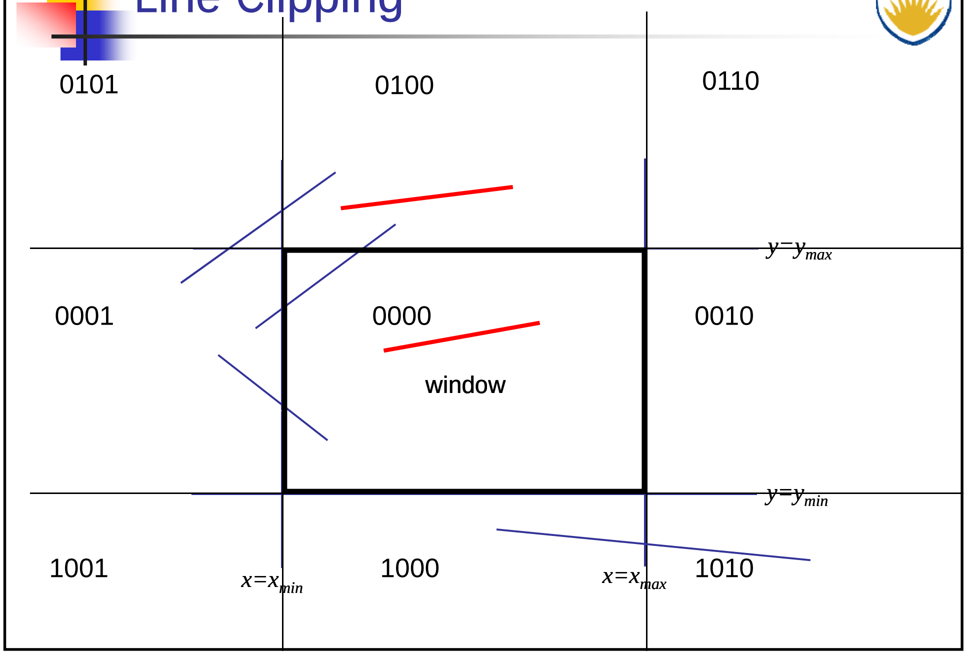
0101	0100	0110
0001	0000	0010
1001	1000	1010

Idea: Assign to P a binary code consisting of a bit for each edge of *D*, using lookup table:

bit	1	0
1	$y < y_{\min}$	$y \geq y_{\min}$
2	$y > y_{\max}$	$y \leq y_{\max}$
3	$x > x_{\max}$	$x \leq x_{\max}$
4	$x < x_{\min}$	$x \geq x_{\min}$

9

Line Clipping



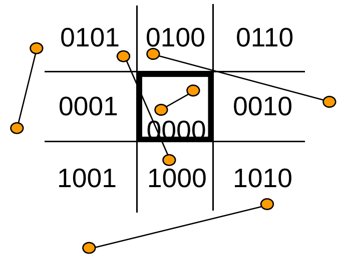
Cohen-Sutherland Algorithm (cont'd)

Given *L* from (x_0, y_0) to (x_1, y_1)

& rectangle D.

If bitwise **and** of the codes of (x_0, y_0) and (x_1, y_1) is not zero, or the bitwise **or** is zero,

then *L* can be trivially handled (it is either totally outside or totally inside D).

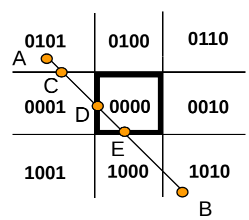


Why?

11

Cohen-Sutherland Algorithm (cont'd)

$\mathbf{C\text{-}S\text{-}Clip}(P_0 = (x_0, y_0), P_1 = (x_1, y_1), x_{\min}, x_{\max}, y_{\min}, y_{\max})$
(assumes $x_0 < x_1$)
$C_0 \Leftarrow \text{code}(P_0)$; $C_1 \Leftarrow \text{code}(P_1)$;
if $((C_0 \text{ and } C_1) \mathrel{!=} 0)$ then return;
if $((C_0 \text{ or } C_1) == 0)$ then draw(P_0, P_1);
else if (OutsideWindow(P_0)) then
begin
$Edge \Leftarrow$ Window boundary of leftmost non-zero bit of C_0;
$P_2 \Leftarrow \overline{P_0, P_1} \cap Edge$;
$\mathbf{C\text{-}S\text{-}Clip}(P_2, P_1, x_{\min}, x_{\max}, y_{\min}, y_{\max})$;
end
else
$Edge \Leftarrow$ Window boundary of leftmost non-zero bit of C_1;
$P_2 \Leftarrow \overline{P_0, P_1} \cap Edge$;
$\mathbf{C\text{-}S\text{-}Clip}(P_0, P_2, x_{\min}, x_{\max}, y_{\min}, y_{\max})$;
end



AB → CB → DB → DE

bit	1	0
1	$y < y_{\min}$	$y \geq y_{\min}$
2	$y > y_{\max}$	$y \leq y_{\max}$
3	$x > x_{\max}$	$x \leq x_{\max}$
4	$x < x_{\min}$	$x \geq x_{\min}$

12

3D clipping

- Determine portion of line inside axis-aligned parallelpiped (viewing frustum in NDC)
- Simple extension to 2D algorithms
- After perspective transform
  - means that clipping volume always the same
    - xmin=ymin= -1, xmax=ymax= 1 in OpenGL
  - boundary lines become boundary planes
    - but bit-codes still work the same way
    - additional front and back clipping plane
      - zmin = -1, zmax = 1 in OpenGL

13

Triangle Clipping

- How does intersection of rectangle & triangle looks like?
  - How many sides?
- How to expand clipping to triangles?
  - Hint: it is convex
  - Will develop on the board...

14

Cohen-Sutherland Algorithm for convex polygons

$\textbf{C-S-Clip}(\text{poly} = P_0,.., P_n, x_{\min}, x_{\max}, y_{\min}, y_{\max})$
for i = 1 to n $C_i \Leftarrow \text{code}(P_i)$;
if $((C_0 \text{ and } C_1 \text{ and} \dots \text{and } C_n) != 0)$ then return;
if $((C_0 \text{ or } C_1 \text{ or} \dots \text{or } C_n) == 0)$ then draw(poly);
else
for i = 1 to n if (OutsideWindow(P_i)) then
begin
$\quad Edge \Leftarrow$ Window boundary of leftmost non-zero bit of C_i;
$\quad P_{i-1,i} \Leftarrow \overline{P_{i-1}, P_i} \cap Edge$;
$\quad P_{i,i+1} \Leftarrow \overline{P_i, P_{i+1}} \cap Edge$;
$\quad \textbf{C-S-Clip}(P_0,.., P_{i-1}, P_{i-1,i}, P_{i,i+1} P_{i+1},..., P_n, x_{\min}, x_{\max}, y_{\min}, y_{\max})$;
end

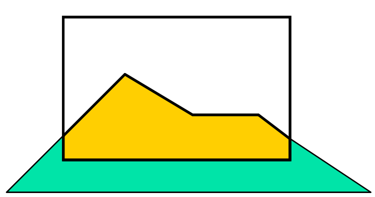
0101	0100	0110
0001	0000	0010
1001	1000	1010

bit	1	0
1	$y < y_{\min}$	$y \geq y_{\min}$
2	$y > y_{\max}$	$y \leq y_{\max}$
3	$x > x_{\max}$	$x \leq x_{\max}$
4	$x < x_{\min}$	$x \geq x_{\min}$

15

Other Geometric Problems

- Questions: How can these ideas be used to design an algorithm for checking if:
  - a point is inside a (convex) polygon?
    - E.g. For collision detection
  - A (convex) polygon is inside/intersects/outside a (convex) polygon ?



16