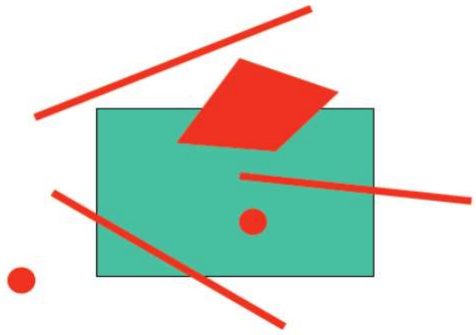
Clipping

Clipping

- Clipping is a fundamental task in graphics.
- It keeps those parts of an object that lie outside a given region from being drawn.
- A large number of clipping algorithms have been developed.
- In OpenGL each object is automatically clipped.
- But the ideas that are used to develop a clipper are basic and arise in diverse situations. So we need to study them.

Clipping

- What to do with the geometry lying outside the view volume

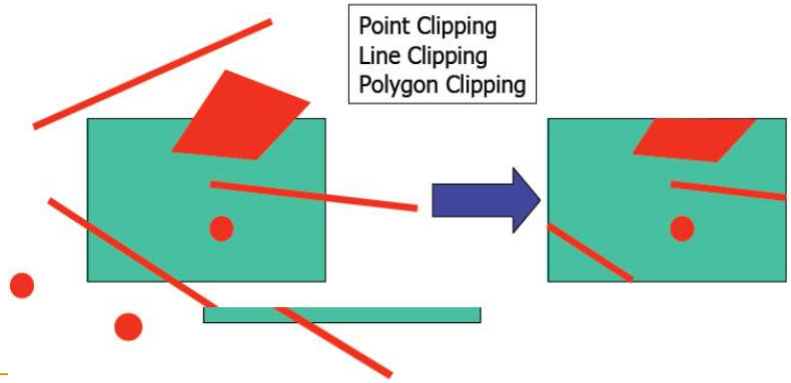


3

3

Clipping Algorithm

- A procedure to identify the portions of the picture that are either inside or outside the clipping window



4

4

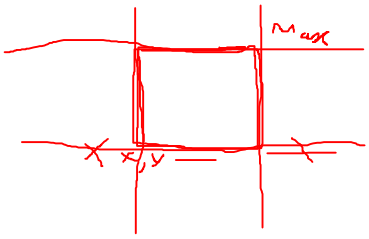
Clipping..

- Point clipping:
  - Remove points outside window.
  - A point is either entirely inside the region or not.
- Line clipping
  - Remove portion of line segment outside window.
- Polygon clipping:
  - Remove portion of polygon outside window.

Why Clip?

- No need to rasterize (store) outside of framebuffer bounds
- Also, don't waste time scan converting pixels outside window

Point Clipping



- **P** = (*x*, *y*) is displayed if:

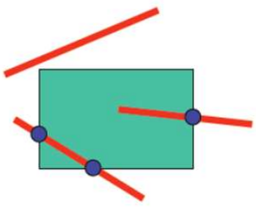
  $x_{wmin} \le x \le x_{wmax}$ and

  $y_{wmin} \le y \le y_{wax}$

Line Clipping

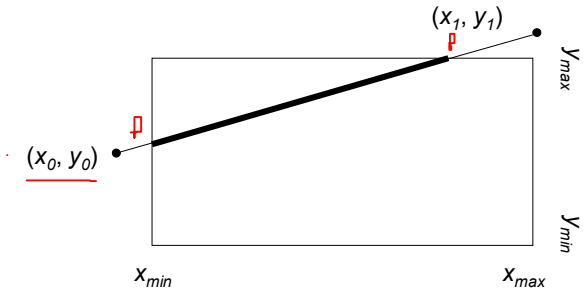
- Algorithm

```
for each line segment
   for each edge of the viewport
      find the intersection
      Remove part lying on the 'wrong' side
   if anything is left, draw it
```



Clipping a Line

- Line clipping against rectangles



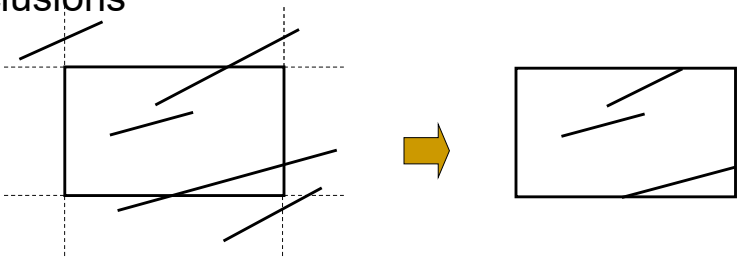
- **The problem**: Given a set of 2D lines or polygons and a window, clip the lines or polygons to their regions that are *inside* the window.

9

9

Motivations

- Efficiency
- Display in portion of a screen
- Occlusions

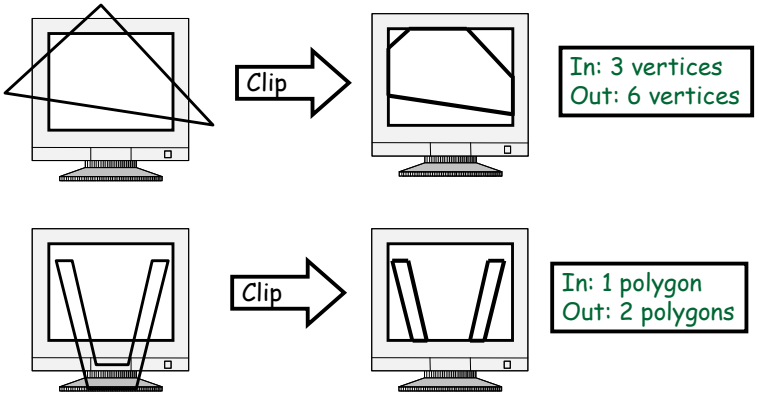


Clip rectangle

10

10

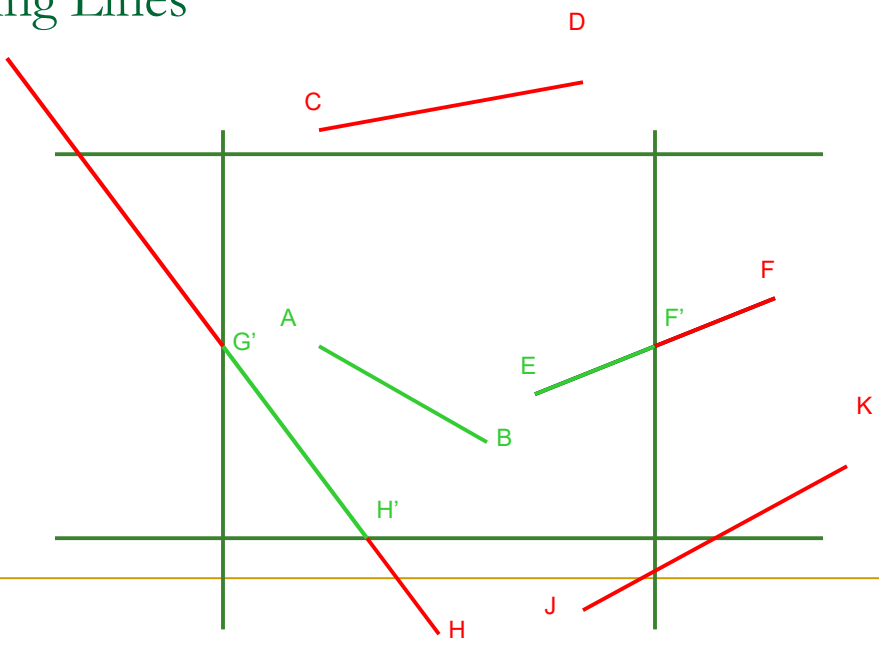
Polygon Clipping is tricky!



11

11

Clipping Lines

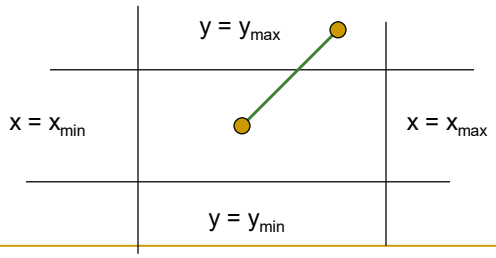


12

12

Cohen-Sutherland Algorithm

- Idea: eliminate as many cases as possible without computing intersections
- Start with four lines that determine the sides of the clipping window



13

13

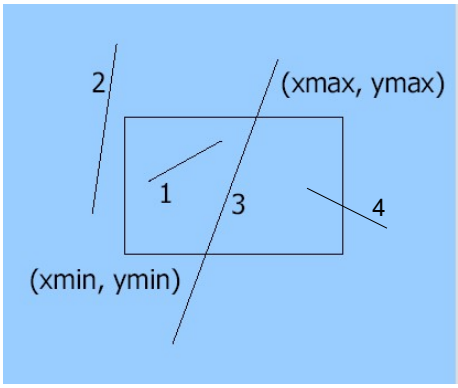
Four Cases

Case 1: All of the line in

Case 2: All of the line out

Case 3: Part in, part out (two end)

Case 4: Part in, part out (One end)

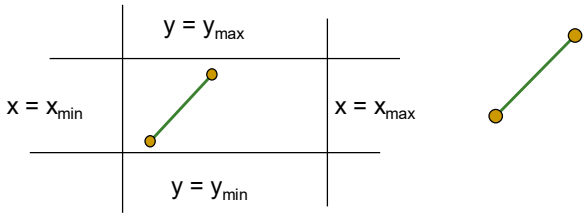


14

14

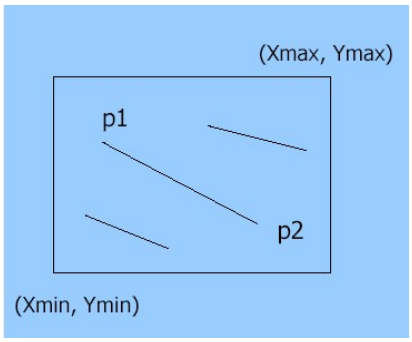
The Cases

- Case 1: both endpoints of line segment inside all four lines
  - Draw (accept) line segment as is



- Case 2: both endpoints outside all lines and on same side of a line
  - Discard (reject) the line segment

The Cases: Trivial Accept

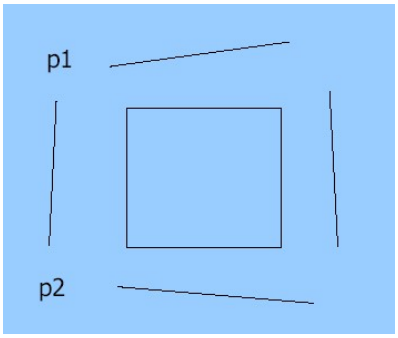


- Case 1: All of line in
- Test Line endpoints:

Xmin<= P1.x,P2.x <= Xmax
and
Ymin <= P1.y, P2.y <= Ymax

- Result: trivially accept.
- Draw line in completely

The Cases: Trivial Reject



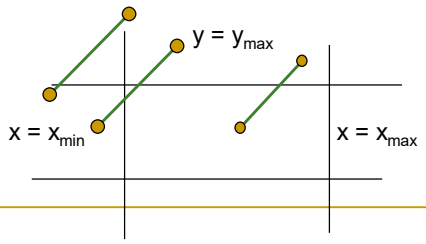
- Case 2: All of line out
- Test Line endpoints:

- p1.x, p2.x <=Xmin OR
- p1.x, p2.x >=Xmax OR
- p1.y, p2.y <= ymin OR
- p1.y, p2.y >= ymax

•Result: trivially reject.
- Don't draw line in.

The Cases: Part in Part out

- Case 3: One endpoint inside, one outside
  - Must do at least one intersection
- Case 4: Both outside
  - May have part inside
  - Must do at least one intersection



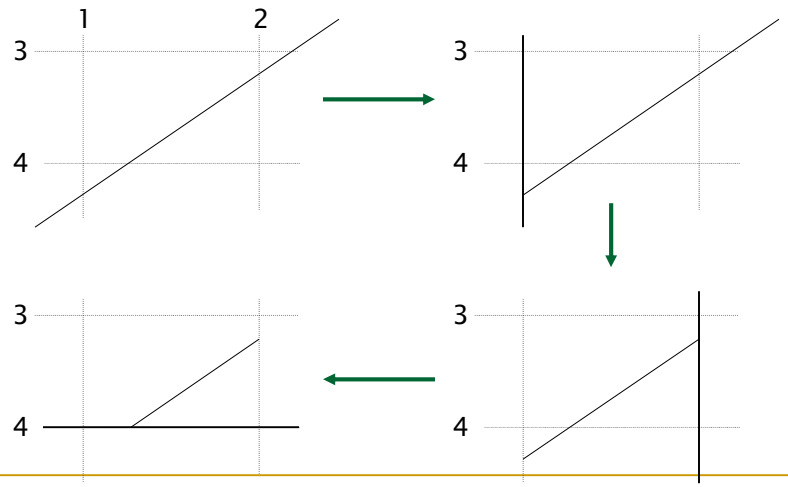
Cohen-Sutherland Algorithm

- Clip line against each edge of clip region in turn
  - If both endpoints outside, discard line and stop (based on outcodes)
  - If both endpoints in, continue to next edge (or finish)
  - If one in, one out, chop line at crossing point and continue
- Works in both 2D and 3D for convex clipping regions

19

19

Cohen-Sutherland Algorithm

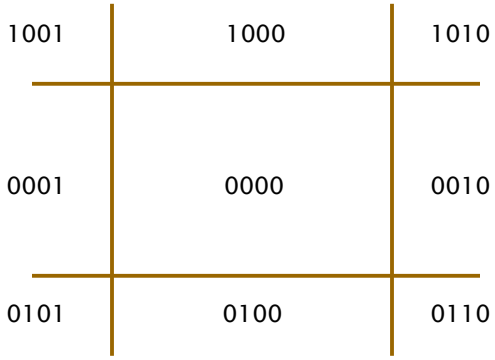


20

20

Cohen-Sutherland Clipping Algorithm

Region and outcodes



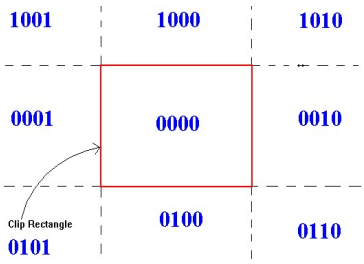
First bit:	above top edge	$y > y_{max}$
Second bit:	below bottom edge	$y < y_{min}$
Third bit:	to right of right edge	$x > x_{max}$
Fourth bit:	to left of left edge	$x < x_{min}$

21

21

Cohen-Sutherland Clipping Algorithm

- Checking for trivial acceptance or rejection using outcodes
  1. Each endpoint of a line segment is assigned an outcode;
  2. If both 4-bit codes are zero, the line can be trivially accepted;
  3. A logical **and** is performed on both outcodes;
  4. If the result is nonzero, the line can be trivially rejected

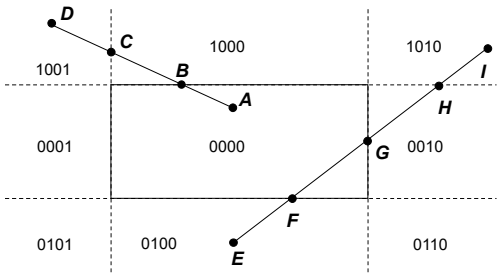


22

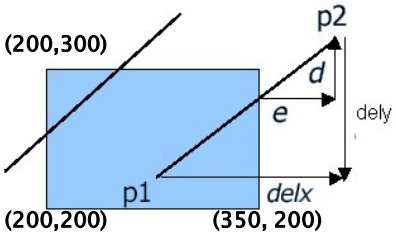
22

Cohen-Sutherland Clipping Algorithm

- End-points pairs are checked for trivial acceptance or rejection using outcode;
- If not trivially accepted or rejected, divide the line segment into two at a clip edge;
- Iteratively clipped by test trivial-acceptance or trivial-rejection, and divided into two segments until completely inside or trivial-rejection.



Clipping Lines: Calculation example



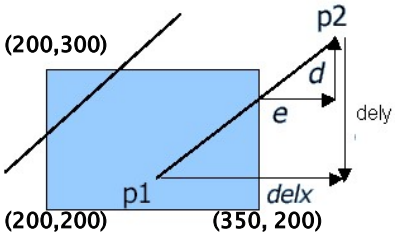
- If chopping window has (left, right, bottom, top) = (200, 350, 200,300),
- What happens when the following lines are chopped?

$$\frac{d}{dely} = \frac{e}{delx}$$

p1 = (310,210),

p2 = (370,310)

Clipping Lines: Calculation example



$$\frac{d}{dely} = \frac{e}{delx}$$

p1 = (310,210), p2 = (370,310)

x1=p1.x=310; y1=p1.y=210;
x2=p2.x=370; y2=p2.y=310;

xmin=200; ymin=200;
xmax=350; ymax=300;

$m=\frac{x_2-x_1}{y_2-y_1}$

P2.x=xmax

p2.y=y1+m(x.max-x1)

p.y=y1+m(X-x1)

p.x=x1+$\frac{1}{m}$(Y-y1)

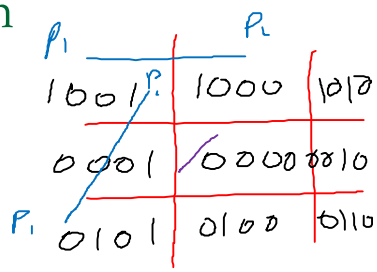
X= x.min or x.max
Y= y.min or y.max

Cohen-Sutherland Clipping Algorithm

```
int clipSegment(Point2 &p1, Point2 &p2, RealRect W)
// Point2 holds a 2D point and RealRect holds an aligned rectangle
{
   do {
        if (trivial accept) return 1;  // some portion survives
        if (trivial reject) return 0;   // no portion survives
        if (p1 is outside) {
                if (p1 is to the left) chop against the left edge
                if (p1 is to the right) chop against the right edge
                if (p1 is below) chop against the bottom edge
                if (p1 is above) chop against the top edge
        } else {                                  // p2 is outside
                if (p2 is to the left) chop against the left edge
                if (p2 is to the right) chop against the right edge
                if (p2 is below) chop against the bottom edge
                if (p2 is above) chop against the top edge
        }
   } while (1);
}
```

Cohen-Sutherland Clipping Algorithm

```
int clipSegment(Point2 &p1, Point2 &p2, RealRect W)
// Point2 holds a 2D point and RealRect holds an aligned rectangle
{
    do {
        if (trivial accept) return 1;          // some portion survives
        if (trivial reject) return 0;          // no portion survives
        if (p1 is outside) {
            if (p1 is to the left) chop against the left edge
                p1.x=x.min
                p1.y=y1+m(x.min-x1)
            if (p1 is to the right) chop against the right edge
                p1.x=x.max
                p1.y=y1+m(x.max-x1)
            if (p1 is below) chop against the bottom edge
                p1.y=y.min
                p1.x=x1+(y.min-y1)/m
            if (p1 is above) chop against the top edge
                p1.y=y.max
                p1.x=x1+(y.max-y1)/m
        } else {                                // p2 is outside
            clip for P2                }
    } while (1);
}
```



P_1 1001
P_2 1000
P_1 0000
P_2 0000
Accept

27

$P_1 = 00\,0\,10$
$P_2 = 10\,1\,0$
$0\,0\,0\,0$

$P_1(60, 40)$
$P_2(160, 340)$

$x_{min} = 40$
$x_{min} = 10$
$x_{max} = 90$
$y_{max} = 310$

$m = \frac{y_2 - y_1}{x_2 - x_1}$

$P_1 x$
$P_2 x$

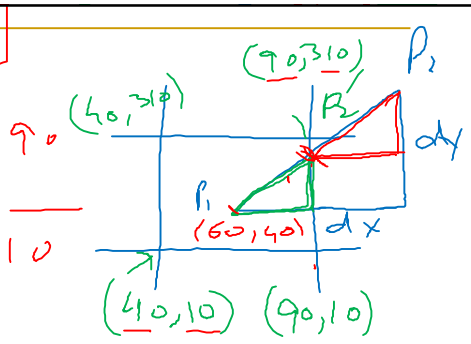
$m = \frac{340 - 40}{160 - 60}$

$m = 3$

$P_2' x = x_{max} = 90$

$P_2' y = y_1 + m(x_{max} - x_1)$
$= 40 + 3(90 - 60)$
$= 130$

$P_2'(90, 130)$



$y' = y_1 + m(x - x_1)$

$x' = x_1 + \frac{1}{m}(y - y_1)$

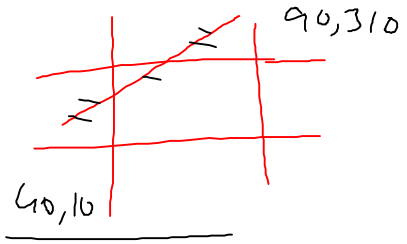
28

$P_1 = (10, 10)$

$P_2 = (60, 310)$

$y' = y_1 + m(X - x_1)$

$x' = x_1 + \frac{1}{m}(Y - y_1)$



29