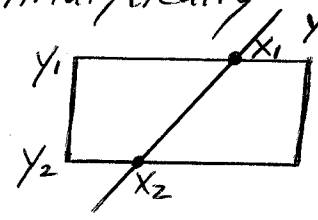
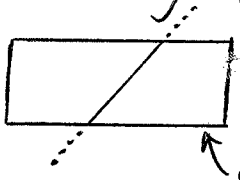
Clipping

Analytically



Solve for x_1 and x_2

Scissoring (on the fly during scan conversion)



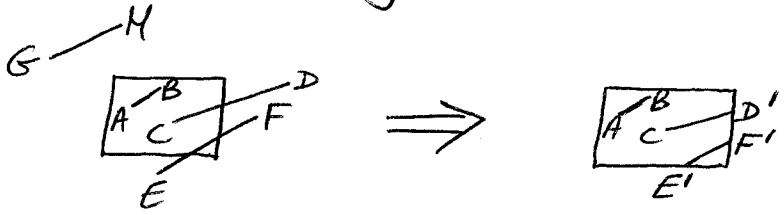
pixels are written conditionally

clip rectangle

Display all pixels in clip rectangle

We will look at different analytic clipping algs.

Examples of clipping:



To clip endpt (x,y) check that

$$x_{min} \le x \le x_{max} \quad AND \quad y_{min} \le y \le y_{max}$$

To clip line AB, check endpts A and B only (not interior pts)

3 cases:

1) A and B are both inside the clip rect ⇒ trivially accepted
2) A inside, B outside ⇒ AB intersects clip rect; compute intersection
3) A AND B lie outside ⇒ AB may or may not intersect; compute intersections

Def. of line:

$y = mx + b \Rightarrow$ slope-intercept formula (describes ~~infinite~~ lines; doesn't deal w/ vertical lines)

(slope: m; y-intercept: b)

✓ $\left.\begin{aligned} x &= x_0 + t(x_1 - x_0) \\ y &= y_0 + t(y_1 - y_0) \end{aligned}\right\} \Rightarrow$ parametric formulation (deals with line segments and vertical lines) *

$0 \le t \le 1$

The parametric formulation for lines will be used for line clipping because it handles line segments and vertical lines.

Cohen-Sutherland Line Clipping Algorithm

1) Endpts are checked for trivial acceptance
2) If line can't be trivially accepted, region checks are done for trivial rejection.
3) If line is not trivially accepted or rejected, it is divided into 2 segments at a clip edge so that one segment can be trivially rejected.

Thus, a segment is iteratively clipped by testing for triv. acc. or rej., + is then subdivided if neither test is successful until what remains is completely inside or can be trivially rejected.

1001	1000	1010
0001	0000	0010
0101	0100	0110

bit #1 $\Rightarrow y > y_{max}$
#2 $\Rightarrow y < y_{min}$
#3 $\Rightarrow x > x_{max}$
#4 $\Rightarrow x < x_{min}$

} used to perform efficient trivial accept + reject tests

All points in clip rect have code 0000
If code(A)=0 AND code(B)=0 $\Rightarrow$ line AB can be trivially accepted

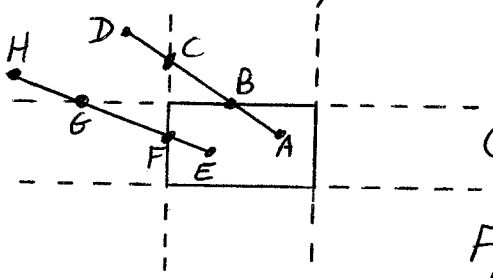
Note: bit 1 is sign bit of $y_{max} - y$
bit 2 . . . . . $y - y_{min}$
bit 3 . . . . . $x_{max} - x$
bit 4 . . . . . $x - x_{min}$

If both endpts lie in an outside halfplane of an edge, then the same bit is 1 in both codes

$\Rightarrow$ if((code(A) & code(B)) != 0) trivial rejection

(& = logical AND)

Else subdivide line $\rightarrow$ find endpt that lies outside (code != 0) and test code to find edge that is crossed to calculate intersection pt.



$$B_x = A_x + \frac{(D_x - A_x)(y_{max} - A_y)}{(D_y - A_y)}$$

$$G_x = E_x + \frac{(H_x - E_x)(y - E_y)}{(H_y - E_y)}$$

$$F_y = E_y + \frac{(G_y - E_y)(x_{min} - E_x)}{(G_x - E_x)}$$

Ex #1:

code(A) = 0000

code(D) = 1001 $\rightarrow$ outside point

1 : line crosses top edge

1 : line crosses left edge

Resolve 1 bits from left-to-right.

Thus, first use top edge to clip AD to AB.

Since code(B) = 0000, display AB

Ex #2:

Code(E) = 0000

Code(H) = 1001

Use top edge to clip EH to EG

Since code(G) = 0001 clip EG to EF

Since code(F) = 0000, display EF

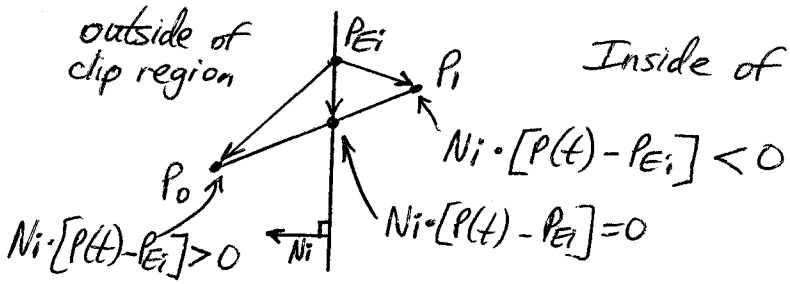
Note: because testing + clipping are done in a fixed order, the alg will sometimes perform needless clipping (external intersections). For ex, compare the clipping of AD and EH

Parametric Line Clipping

Cyrus-Beck (1978) alg efficiently clips a 2D line against a rectangle or a convex polygon, or a 3D line against a convex polyhedron in 3D space.

Liang-Barsky (1984) alg is more efficient, especially for upright 2D and 3D clip regions.

Cyrus-Beck alg:



N_i is edge's outward normal

1) $P(t) = P_0 + (P_1 - P_0)t$, $0 \le t \le 1$ ← parametric rep. of line segment P_0P_1
2) Pick an arbitrary pt. P_{Ei} on E_i
3) Consider the 3 vectors $P(t) - P_{Ei}$ from P_{Ei} to P_0, P_1, and pt of intersection.
   We know that $P_{Ei}P_0$ is outside because $N_i \cdot [P_0 - P_{Ei}] > 0$

Note: $\overleftarrow{A}^{B} = \overleftarrow{B}$ $A \cdot B = |A||B|\cos\theta > 0$ in this ex. (Project one vector onto second)

Since we are interested in pt. of intersection, we solve for t in

$$N_i \cdot [P(t) - P_{Ei}] = 0$$

$$N_i \cdot [P_0 + (P_1 - P_0)t - P_{Ei}] = 0$$

$$N_i \cdot [P_0 - P_{Ei}] + N_i \cdot [P_1 - P_0]t = 0$$

$$t = \frac{N_i \cdot [P_0 - P_{Ei}]}{-N_i \cdot \underbrace{[P_1 - P_0]}_{D}}$$

Make sure that $-N_i \cdot D \neq 0$

Check that:

$N_i \neq 0$ ($N_i = 0$ is a mistake only)

$D \neq 0$ (that is, $P_0 \neq P_1$)

$N_i \cdot D \neq 0$ (E_i and P_0P_1 are not parallel. If so, the alg moves onto next case)

4) Solve for 4 t's: one for each edge.

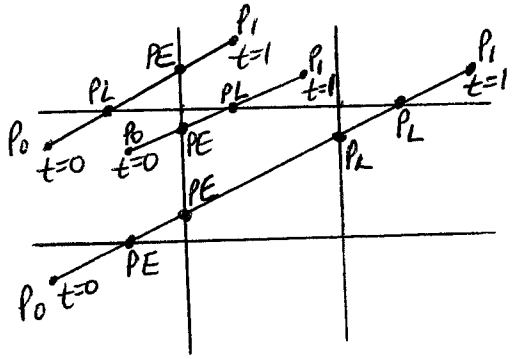
Discard t if $t < 0$ or $t > 1$.

However, even if $0 \leq t \leq 1$, we can still have them be pts outside of clip rect.

Which ones do we choose?

Soln: characterize intersections as PE or PL.

(potentially entering) (pot. leaving)



Note: 2 interior intersection pts of a line intersecting the clip rect have opposing labels.

$N_i \cdot D < 0 \Rightarrow$ PE ($\theta > 90°$) outward normal is in opp. dir of incoming line P_0P_1

$N_i \cdot D > 0 \Rightarrow$ PL ($\theta < 90°$)

Since $N_i \cdot D$ is denominator of t, we categorize intersection while solving for it.

The clipped line is defined by the range (t_E, t_L) where t_E is PE with largest t, t_L is PL with smallest t, and $0 \le t_E < t_L$, $t_L \le 1$

We use t_E and t_L in $P(t) = P_0 + (P_1 - P_0)t$ to calculate the x and y coordinates.

Summary:

Cohen-Sutherland alg. is efficient when outcode testing can be done cheaply (bitwise ops) and most line segments can be trivially accepted or rejected. Parametric line clipping is efficient when many lines need to be clipped, since testing is done on parameter values, and (x, y) are computed only when needed (once).

Note: Liang-Barsky alg. is more efficient than Cyrus-Beck because of additional trivial rejection testing.

Cyrus-Beck alg.

$t_E = 0$
$t_L = 1$
for all 4 sides {
    Compute t
    use sign of $N_i \cdot D$ to categorize as PE or PL
    if PE then $t_E = \max(t_E, t)$
    if PL then $t_L = \min(t_L, t)$
}
if ($t_E > t_L$) return 0
else return ($P(t_E)$, $P(t_L)$)

Liang-Barsky alg

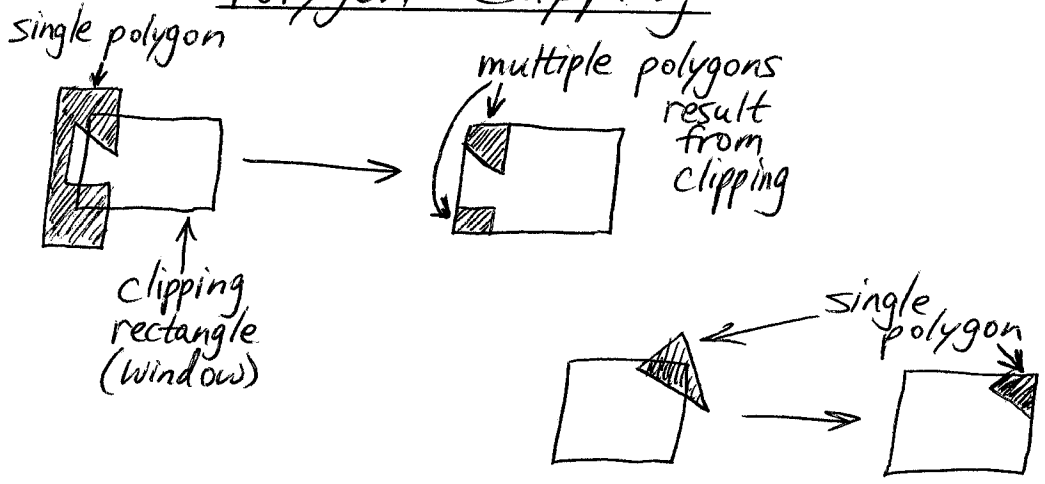
$t_E = 0$
$t_L = 1$ ← updates t_E if $(-N_i \cdot D) > 0$ (PE); returns 0 if crossover
" t_L if $(-N_i \cdot D) < 0$ (PL)

looks for trivial rejection test to exit sooner

```
if(CLIPt( ))        /* inside wrt left edge */
  if(CLIPt( ))      /*  . . . . . right  . . */
    if(CLIPt())     /*  . . . . . bottom " */
      if(CLIPt())   /*  " . . . . . top  " */
        return(P(tE), P(tL))
```

Both Cyrus-Beck and Liang-Barsky generalize to 3D

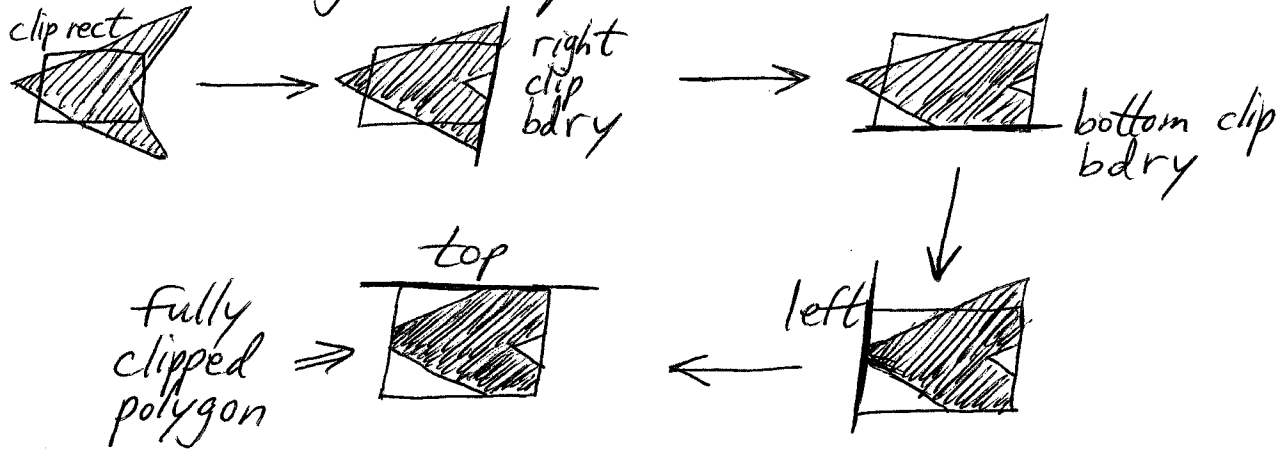
Polygon Clipping



Clipping a polygon to a clip rectangle requires each polygon edge to be tested against each edge of the clip rectangle

Sutherland-Hodgman Alg.

Divide-and-conquer strategy ⇒ solve a series of simple problems: clip polygon against a single infinite clip edge; 4 in succession for a rectangular clip window

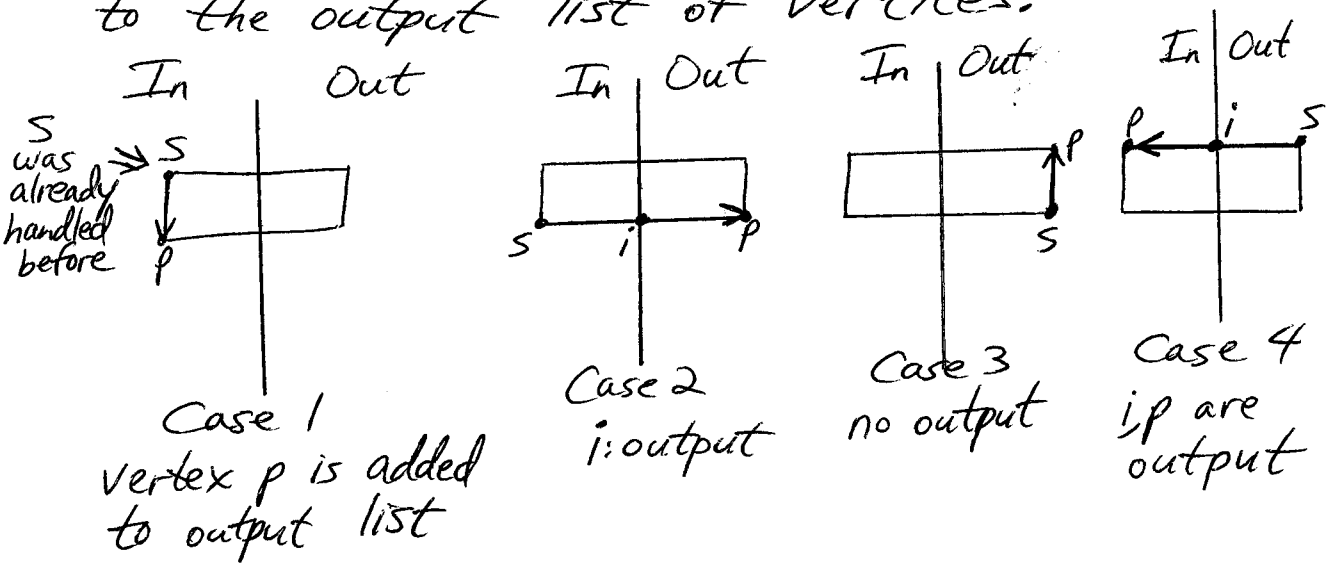


Polygon is given as a list of vertices $V_1, V_2, V_3, \ldots V_n$

Its edges are $V_1V_2, V_2V_3, \ldots V_{n-1}V_n, V_nV_1$

The alg. moves around polygon, from edge to edge, clipping to a clip edge.

$\Rightarrow$ 0, 1, or 2 vertices are added to the output list of vertices.



The Sutherland-Hodgman alg is also reentrant: as soon as a vertex is output, the clipper calls itself with that vertex (to clip against next clip bdry). No intermediate storage for partially clipped polygon.