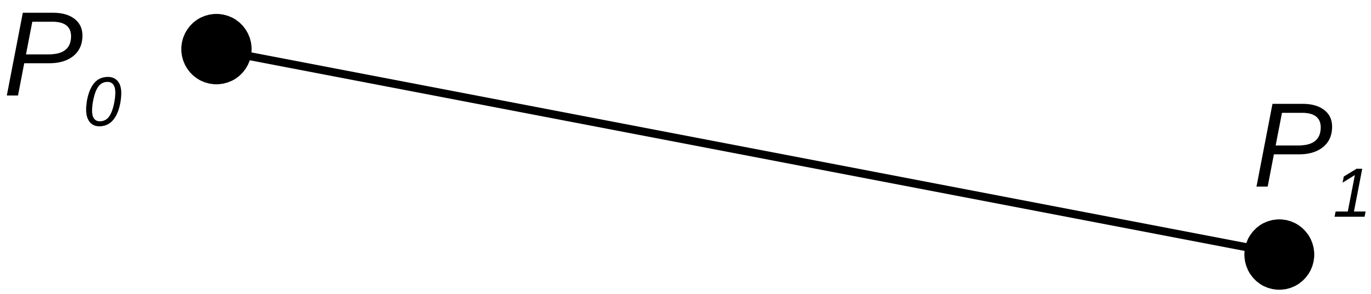
Advanced Clipping Algorithms

Prof. Subhadip Basu
Dept. of CSE, JU

Parametric Line Equation



- Line: $P(t) = P_0 + t(P_1 - P_0)$
- *t value defines a point on the line going through P_0 and P_1*
- *$0 <= t <= 1$ defines line segment between P_0 and P_1*
- $P(0) = P_0$ $\quad\quad$ $P(1) = P_1$

The Cyrus-Beck Technique

- Cohen-Sutherland algorithm computes *(x,y)* intersections of the line and clipping edge
- Cyrus-Beck finds a value of parameter *t* for intersections of the line and clipping edges
- Simple comparisons used to find actual intersection points
- Liang-Barsky optimizes it by examining *t* values as they are generated to reject some line segments immediately

Finding the Intersection Points

Line $P(t) = P_0 + t(P_1 - P_0)$

Point on the edge P_{ei}

N_i ➔ Normal to edge i

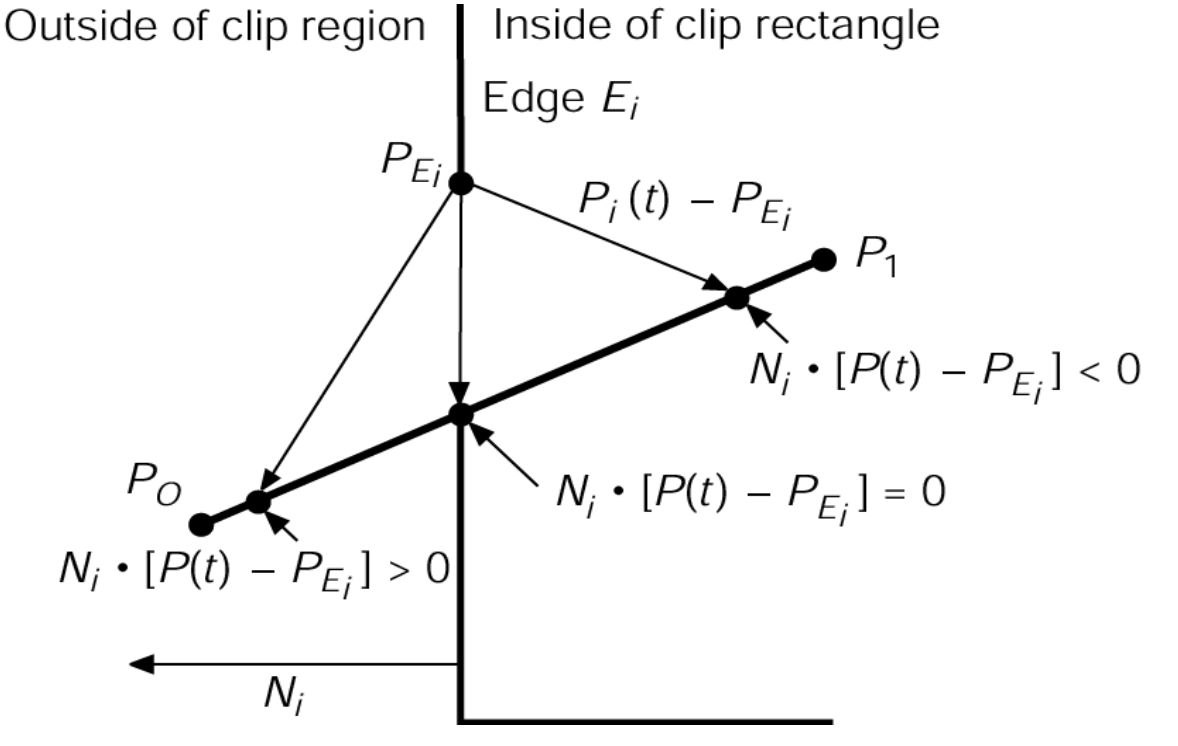
$$N_i \bullet [P(t) - P_{Ei}] = 0$$

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

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

$$\text{Let } D = (P_1 - P_0)$$

$$t = \frac{N_i \bullet [P_0 - P_{Ei}]}{-N_i \bullet D}$$



Make sure

1. *$D \neq 0$, or $P_1 \neq P_0$*
2. *$N_i \bullet D \neq 0$, lines are not parallel*

Calculating N_i

N_i for window edges

- WT: (0,1)   WB: (0, -1)   WL: (-1,0)   WR: (1,0)

N_i for arbitrary edges

- Calculate edge direction
  - $E = (V_1 - V_0) / |V_1 - V_0|$
  - Be sure to process edges in CCW order
- Rotate direction vector -90°

  $N_x = E_y$

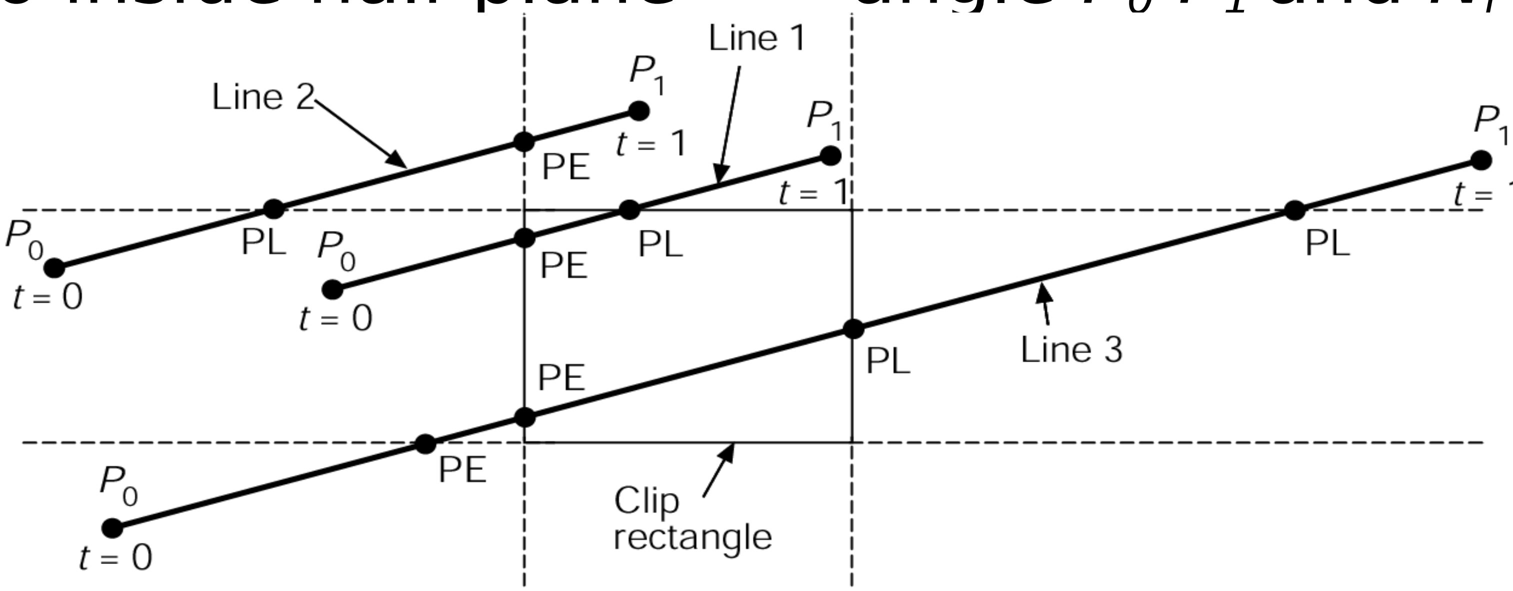
  $N_y = -E_x$

TABLE 3.1 CALCULATIONS FOR PARAMETRIC LINE CLIPPING ALGORITHM*

Clip edge$_i$	Normal N_i	P_{E_i}	$P_0 - P_{E_i}$	$t = \dfrac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot D}$
left: $x = x_{\min}$	$(-1, 0)$	$(x_{\min}, y)$	$(x_0 - x_{\min}, y_0 - y)$	$\dfrac{-(x_0 - x_{\min})}{(x_1 - x_0)}$
right: $x = x_{\max}$	$(1, 0)$	$(x_{\max}, y)$	$(x_0 - x_{\max}, y_0 - y)$	$\dfrac{(x_0 - x_{\max})}{-(x_1 - x_0)}$
bottom: $y = y_{\min}$	$(0, -1)$	$(x, y_{\min})$	$(x_0 - x, y_0 - y_{\min})$	$\dfrac{-(y_0 - y_{\min})}{(y_1 - y_0)}$
top: $y = y_{\max}$	$(0, 1)$	$(x, y_{\max})$	$(x_0 - x, y_0 - y_{\max})$	$\dfrac{(y_0 - y_{\max})}{-(y_1 - y_0)}$

Finding the Line Segment

- Calculate intersection points between line and every window line
- Classify points as potentially entering (PE) or leaving (PL)
- PE if crosses edge into inside half plane => angle P_0 P_1 and N_i greater 90° => $N_i \bullet D$
- PL otherwise.
- Find $T_e = max(t_e)$
- Find $T_l = min(t_l)$
- Discard if $T_e > T_l$
- If $T_e < 0$, $T_e = 0$
- If $T_l > 1$, $T_l = 1$
- Use T_e, T_l to compute intersection coordinates (x_e, y_e), (x_l, y_l)



1994 Foley/VanDam/Finer/Huges/Phillips ICG

Cyrus-Beck Algorithm

```
precalculate N_i and select a P_Ei for each edge;

for (each line segment to be clipped) {
    if (P1 == P0)
        line is degenerate so clip as a point;
    else {
        tE = 0; tL = 1;
        for (each candidate intersection with a clip edge) {
            if (N_i • D != 0) {        /* Ignore edges parallel to line for now */
                calculate t;
                use sign of N_i • D to categorize as PE or PL;
                if (PE) tE = max (tE, t);
                if (PL) tL = min (tL, t);
            }
        }
        if (tE > tL)
            return NULL;
        else
            return P(tE) and P(tL) as true clip intersections;
    }
}
```

Liang-Barsky Line Clipping

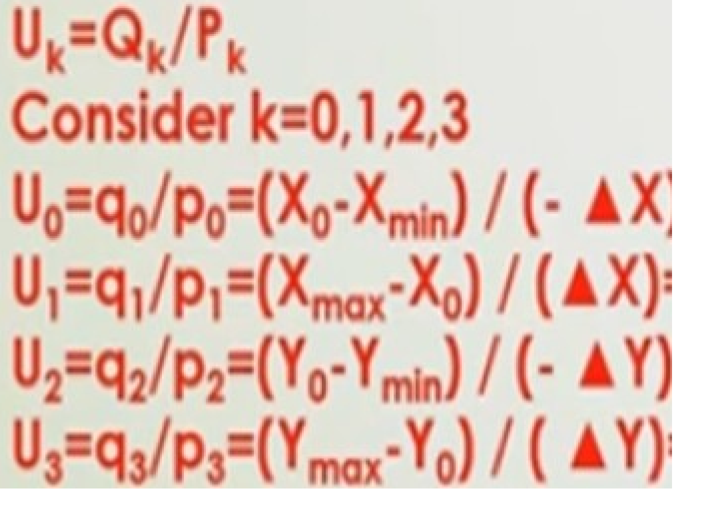
- Consider the parametric definition of a line:
  - $x = x_1 + u\Delta x$
  - $y = y_1 + u\Delta y$
  - $\Delta x = (x_2 - x_1)$, $\Delta y = (y_2 - y_1)$, $0 \leq (u) \leq 1$

- What if we could find the range for u in which both x and y are inside the viewport?

Liang-Barsky Line Clipping

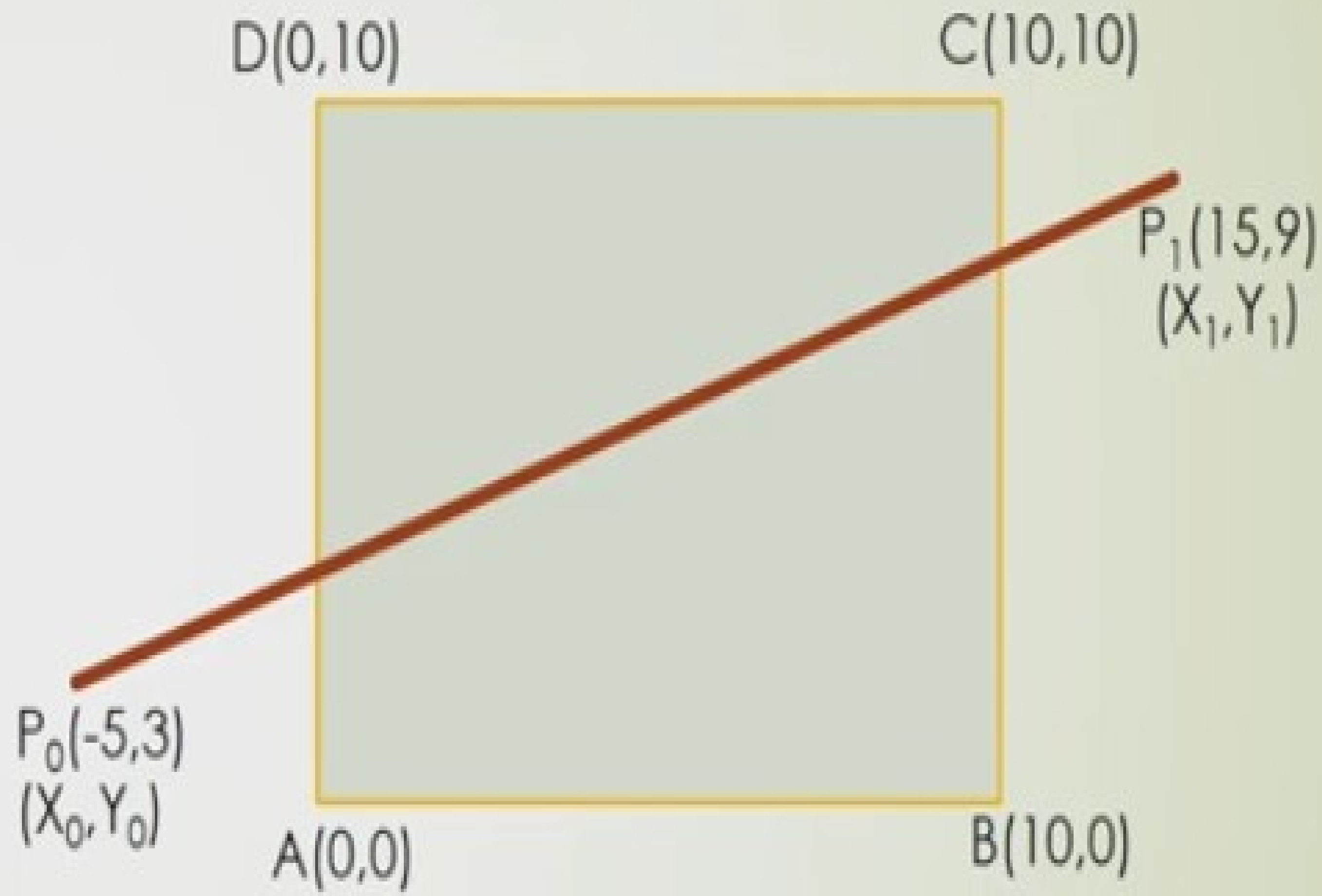
- Mathematically, this means
  - $x_{min} \leq x_1 + u\Delta x \leq x_{max}$
  - $y_{min} \leq y_1 + u\Delta y \leq y_{max}$
- Rearranging, we get
  - $-u\Delta x \leq (x_1 - x_{min})$
  - $u\Delta x \leq (x_{max} - x_1)$
  - $-u\Delta y \leq (y_1 - y_{min})$
  - $u\Delta y \leq (y_{max} - y_1)$

  – In general: $u_k * p_k \leq q_k$



Q

Let ABCD be the Rectangular window with A(0,0) B(10,0) C(0,10) and D(0,10) Use Liang Barsky Algorithm to clip the line P_0P_1 with P0(-5,3) P1(15,9)



Step 1 : Plot the points

$X_{min}=0, X_{max}=10$

$Y_{min}=0, Y_{max}=10$

Step 2 :

$\blacktriangle X = X_1 - X_0 = (15-(-5)=20$

$\blacktriangle Y = Y_1 - Y_0 = 9-3=6$

Step 3 :

$U_k = Q_k / P_k$

Consider k=0,1,2,3

$U_0 = q_0/p_0 = (X_0 - X_{min}) / (-\blacktriangle X) = (-5-0)/-20 = 0.25$

$U_1 = q_1/p_1 = (X_{max} - X_0) / (\blacktriangle X) = (10+5)/20 = 0.75$

$U_2 = q_2/p_2 = (Y_0 - Y_{min}) / (-\blacktriangle Y) = (3-0)/-6 = -0.5$

$U_3 = q_3/p_3 = (Y_{max} - Y_0) / (\blacktriangle Y) = (10-3)/6 = 1.16$

Step 4 :

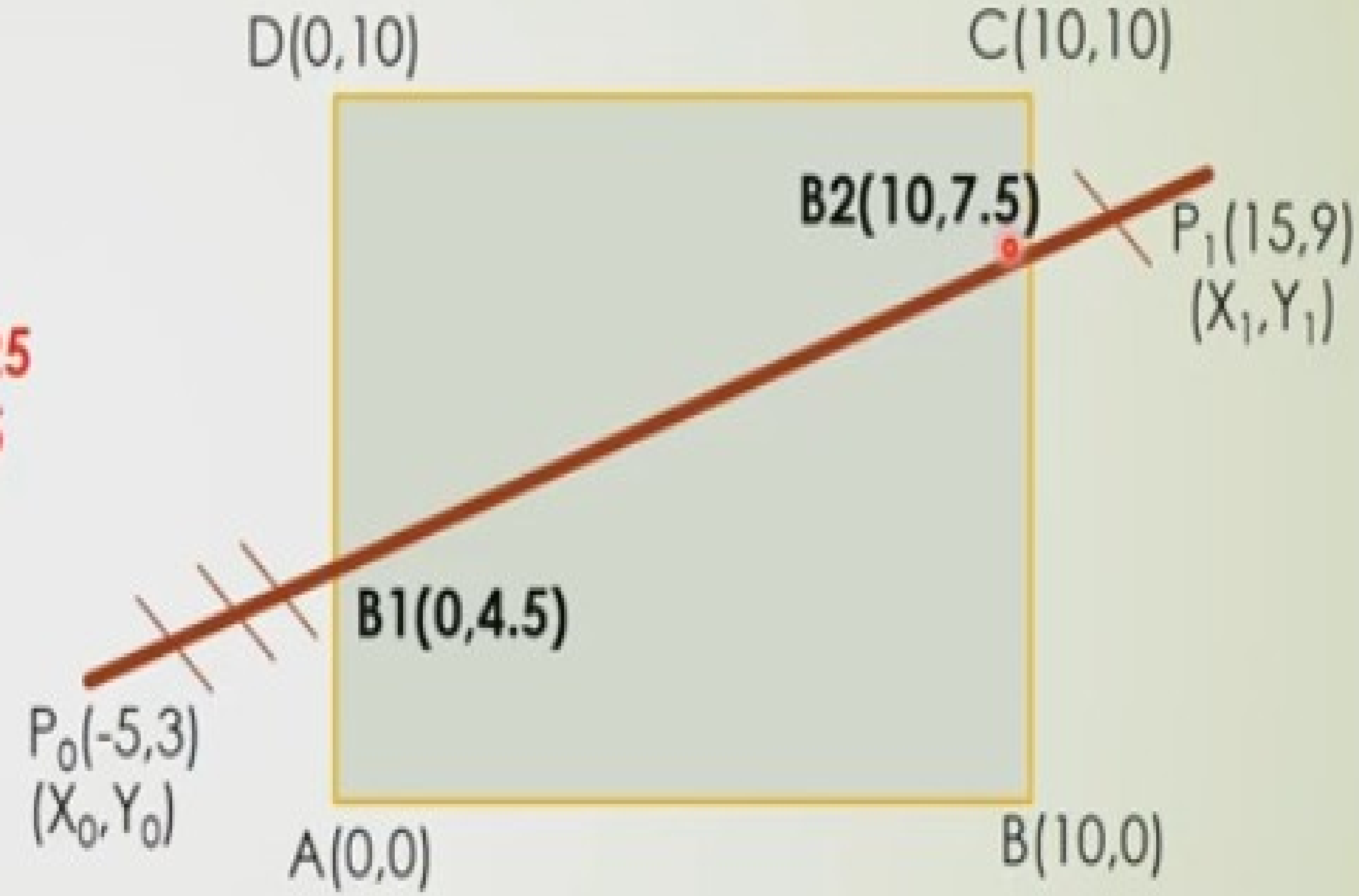
Consider values of Uk if it satisfies

Umin<=Uk<=Umax ie 0<=Uk<=1

We consider **U0=0.25 and U1=0.**

Calculate Intersection Points

Let ABCD be the Rectangular window with A(0,0) B(10,0) C(0,10) and D(0,10) Use Liang Barsky Algorithm to clip the line P_0P_1 with P0(-5,3) P1(15,9)



Step 3 :

$U_k=Q_k/P_k$

Consider k=0,1,2,3

$U_0=q_0/p_0=(X_0-X_{min}) / (-\blacktriangle X)=(-5-0)/-20 =0.25$

$U_1=q_1/p_1=(X_{max}-X_0) / (\blacktriangle X)=(10+5)/20 =0.75$

$U_2=q_2/p_2=(Y_0-Y_{min}) / (-\blacktriangle Y)=(3-0)/-6=-0.5$

$U_3=q_3/p_3=(Y_{max}-Y_0) / (\blacktriangle Y)=(10-3)/6=1.16$

Step 4 :

Consider values of Uk if it satisfies

Umin<=Uk<=Umax ie 0<=Uk<=1

We consider **U0=0.25 and U1=0.75**

Calculate Intersection Points

(x,y)=(0,4.5)

1.When u=0.25

X=X0+U ▲X

X=-5+0.25*20=0, Y=Y0+U▲Y=3+0.25*6=4.5

(x,y)=(10,7.5)

2..When u=0.75

X=X0+U ▲X

X=-5+0.75*20=10,

Y=Y0+U▲Y=3+0.75*6=7.5

Liang-Barsky Line Clipping

- Cases:
  1. $p_k = 0$
     - Line is parallel to boundaries
       - If for the same k, $q_k < 0$, reject
       - Else, accept
  2. $p_k < 0$
     - Line starts outside this boundary
       - $r_k = q_k / p_k$
       - $u_1 = \max(0, r_k, u_1)$
  3. $p_k > 0$
     - Line starts inside this boundary
       - $r_k = q_k / p_k$
       - $u_2 = \min(1, r_k, u_2)$
  4. If $u_1 > u_2$, the line is completely outside

Liang-Barsky Line Clipping

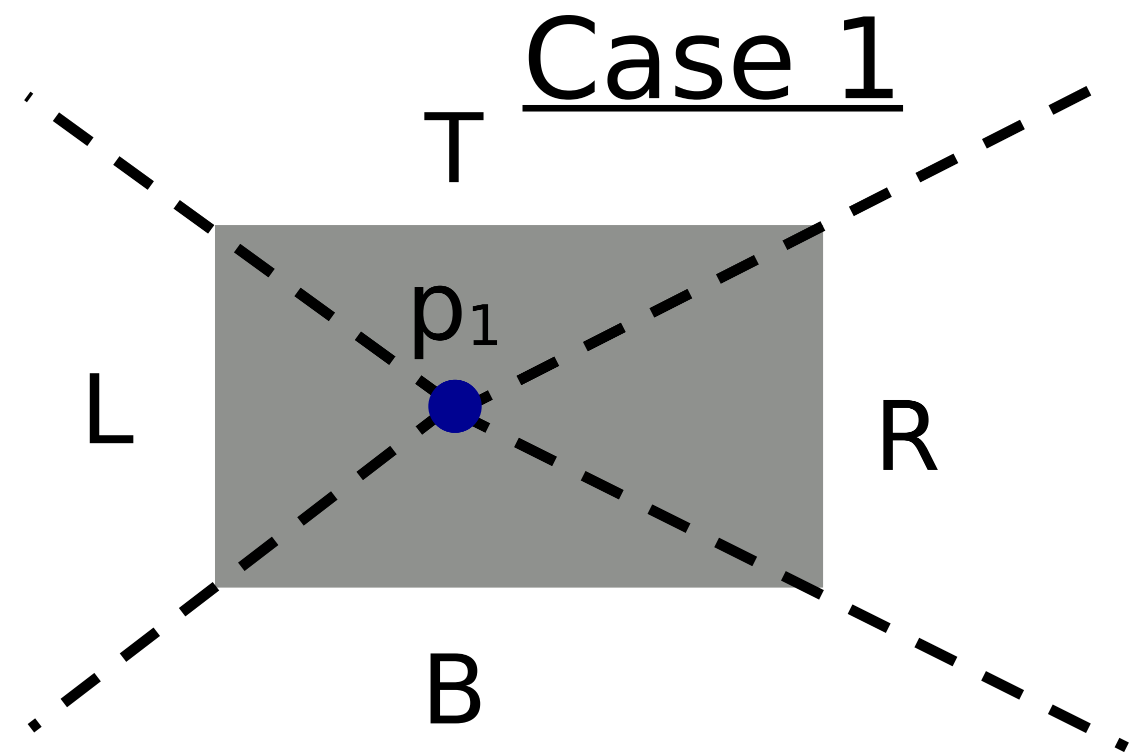
- In most cases, Liang-Barsky is slightly more efficient
  - Avoids multiple shortenings of line segments
- However, Cohen-Sutherland is much easier to understand
  - An important issue if you’re actually implementing

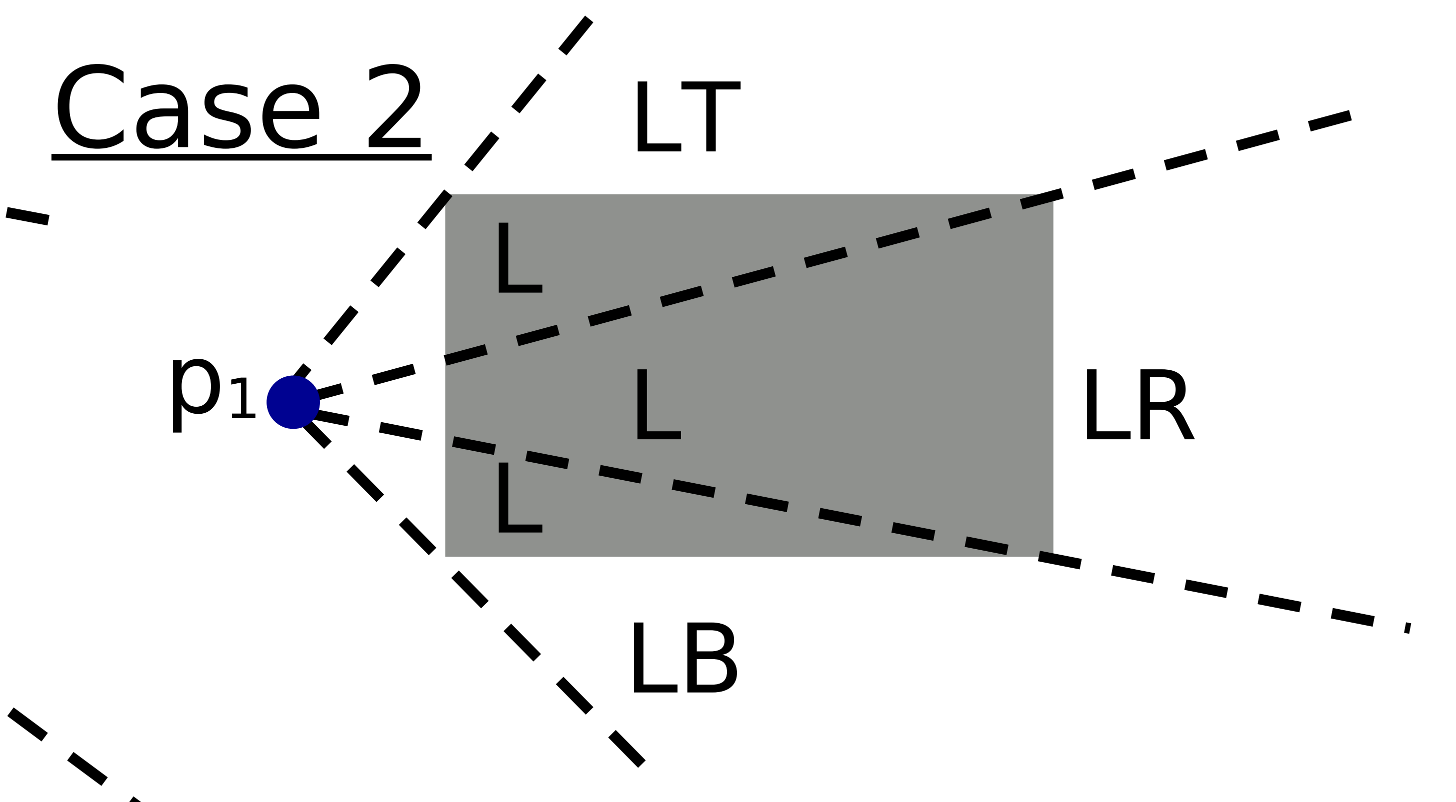
Nicholl-Lee-Nicholl Line Clipping

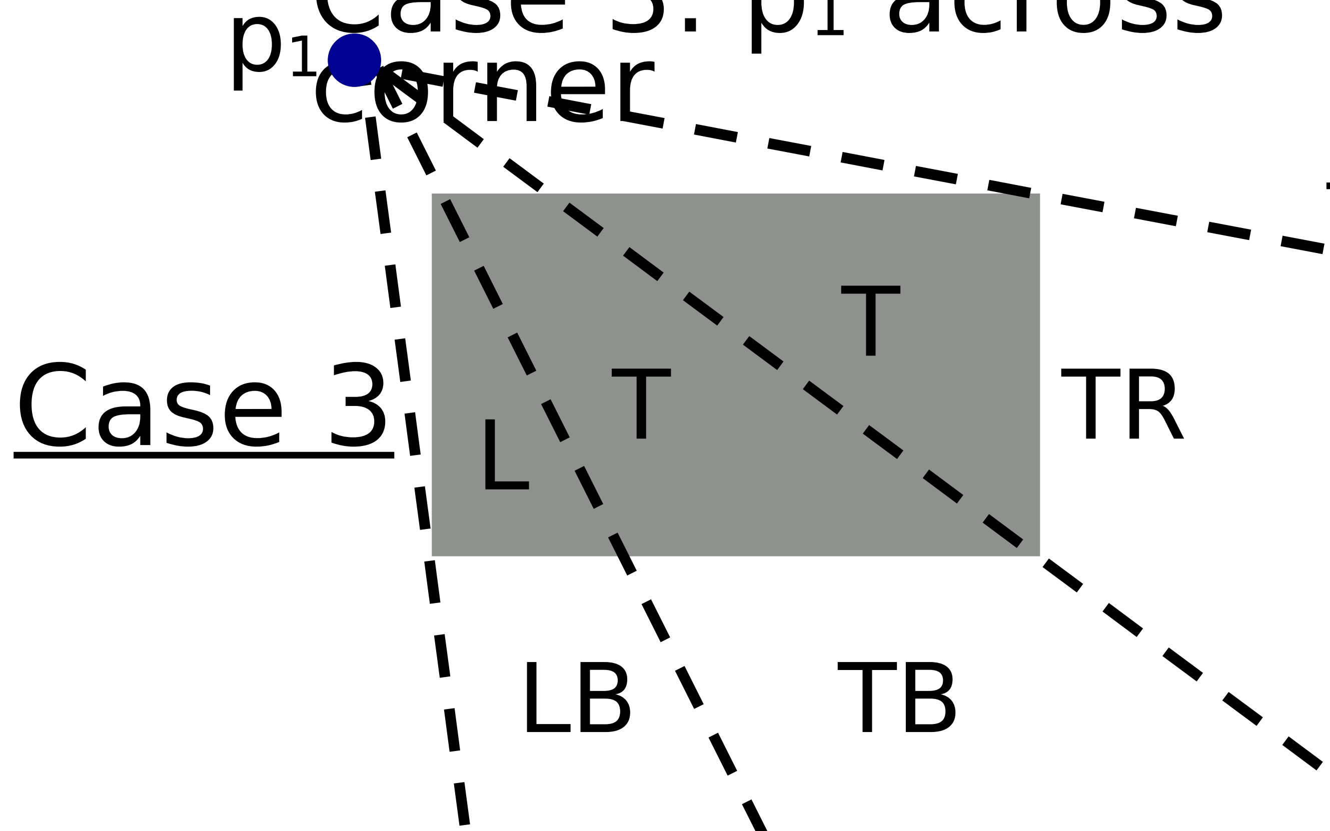
- This is a theoretically optimal clipping algorithm (at least in 2D)
  - However, it only works well in 2D
- More complicated than the others
- Just do an overview here

Nicholl-Lee-Nicholl Line Clipping

- Partition the region based on the first point (p_1):
  - Case 1: p_1 inside region
  - Case 2: p_1 across edge
  - Case 3: p_1 across corner







Nicholl-Lee-Nicholl Line Clipping

- Can use symmetry to handle all other cases
- “Algorithm” (really just a sketch):
  - Find slopes of the line and the 4 region bounding lines
  - Determine what region p_2 is in
    - If not in a labeled region, discard
    - If in a labeled region, clip against the indicated sides

A Note on Redundancy

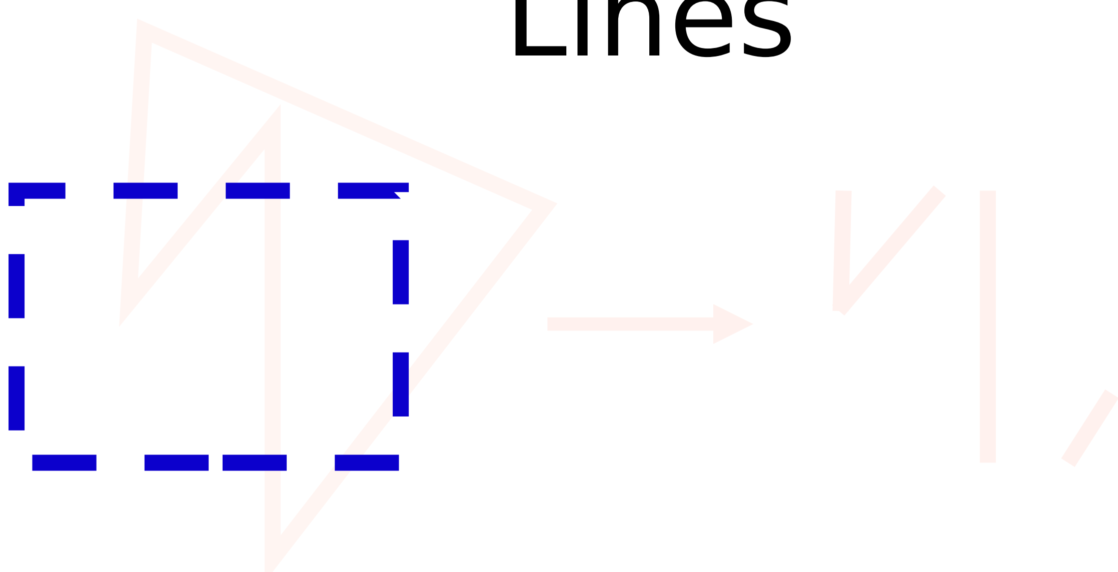
- Why am I presenting multiple forms of clipping?
  - Why do you learn multiple sorts?
    - Fastest can be harder to understand / implement
    - Best for the general case may not be for the specific case
      - Bubble sort is really great on mostly sorted lists
    - “History repeats itself”
      - You may need to use a similar algorithm for something else; grab the closest match

Polygon Clipping

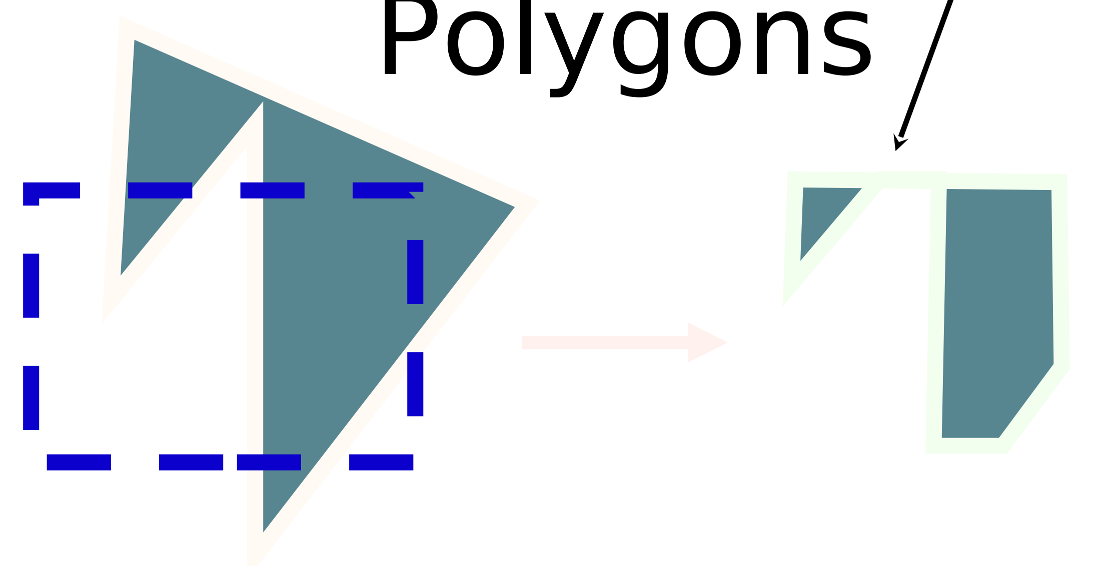
- Polygons are just composed of lines. Why do we need to treat them differently?
  - Need to keep track of what is inside

NOTE:

Lines



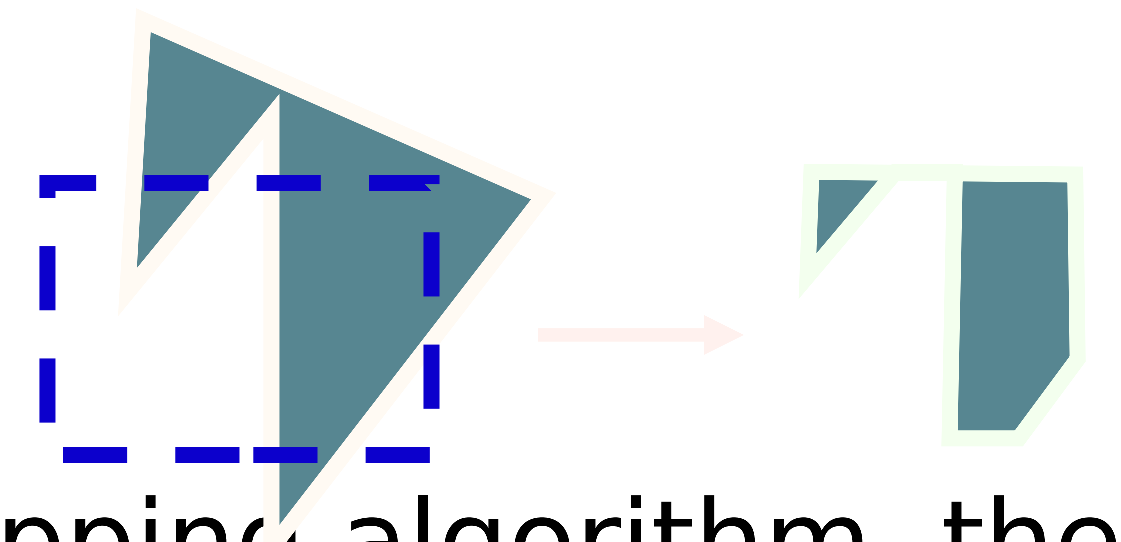
Polygons



Weiler-Atherton Polygon Clipping

- When using Sutherland-Hodgeman, concavities can end up linked

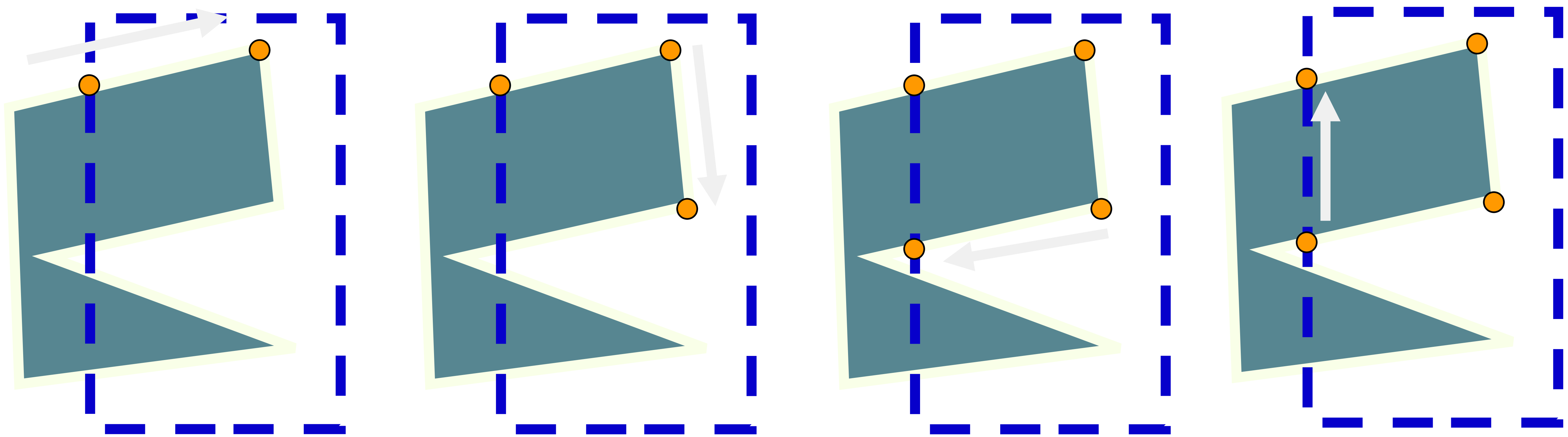
Remember this?



- A different clipping algorithm, the Weiler-Atherton algorithm, creates separate polygons

Weiler-Atherton Polygon Clipping

- Example:



Out -> In
Add clip vertex
Add end vertex

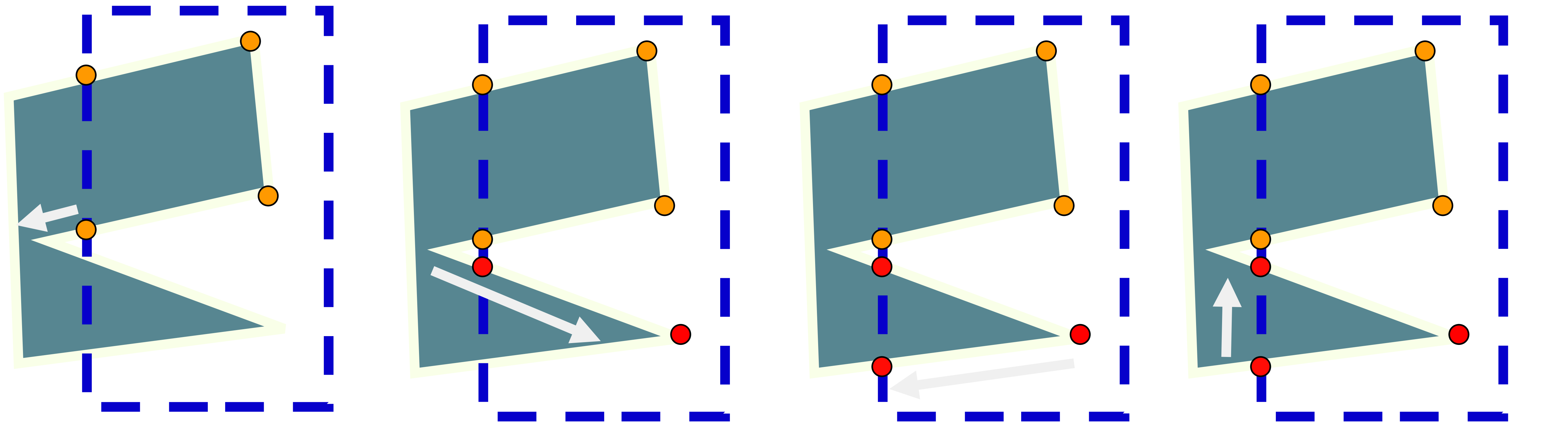
In -> In
Add end vertex

In -> Out
Add clip vertex
Cache old direction

Follow clip edge until
(a) new crossing found
(b) reach vertex already added

Weiler-Atherton Polygon Clipping

- Example (cont'd):



Continue from cached vertex and direction

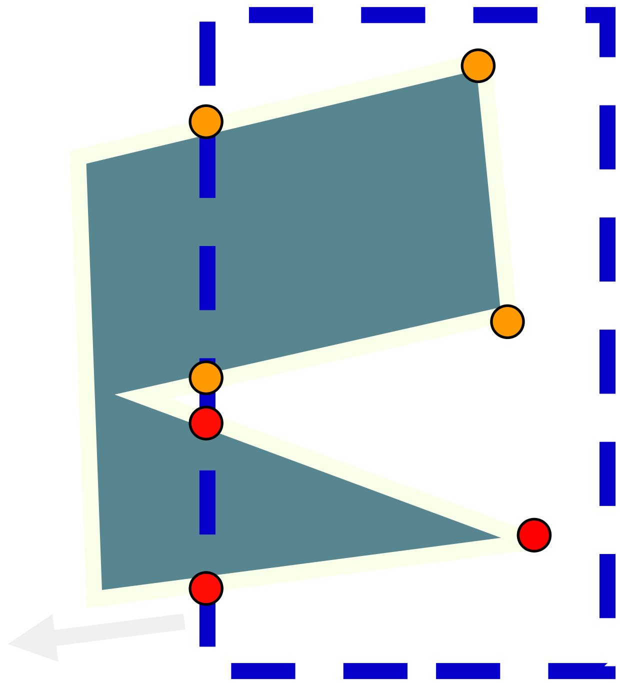
Out -> In
Add clip vertex
Add end vertex

In -> Out
Add clip vertex
Cache old direction

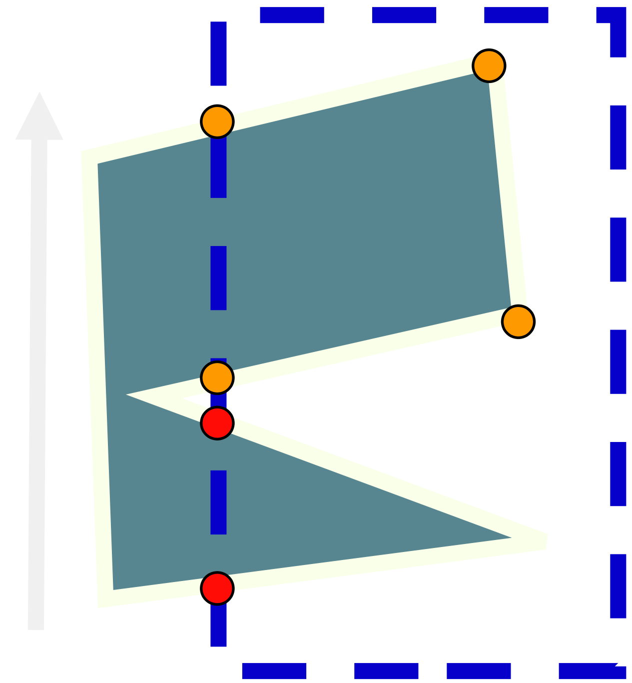
Follow clip edge until
(a) new crossing found
(b) reach vertex already added

Weiler-Atherton Polygon Clipping

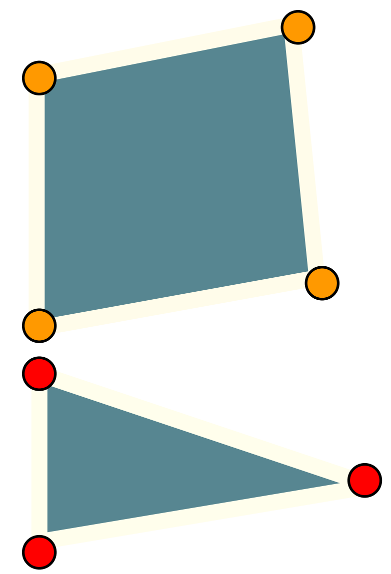
- Example (cont’d):



Continue from cached vertex and direction

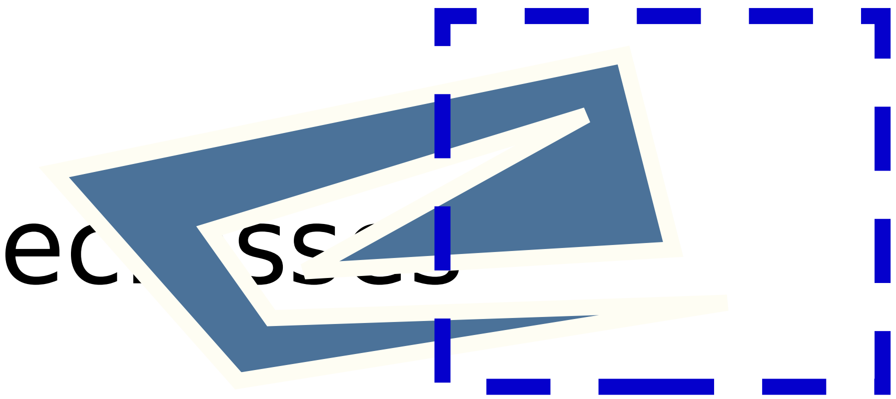


Nothing added
Finished

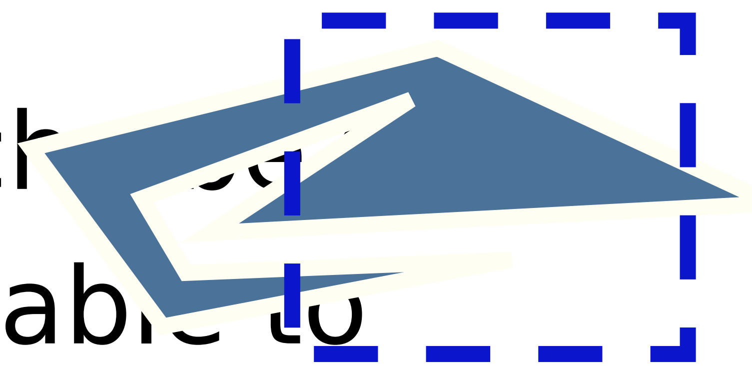


Final Result:
2 unconnected polygons

Weiler-Atherton Polygon Clipping



- Difficulties:
  - What if the polygon recrosses an edge?
  - How big should your cache be?
  - Geometry step must be able to create new polygons
    - Not 1 in, 1 out



Done with Clipping

- Point Clipping (really just culling)
  - Easy, just do inequalities
- Line Clipping
  - Cohen-Sutherland
  - Cyrus-Beck
  - Liang-Barsky
  - Nicholl-Lee-Nicholl
- Polygon Clipping
  - Sutherland-Hodgeman
  - Weiler-Atherton

Any Questions?