Advance Clipping Algorithms

Prof. Subhadip Basu

Liang-Barsky Line Clipping

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

- 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
  - $x_{min} \le x_1 + u\Delta x \le x_{max}$
  - $y_{min} \le y_1 + u\Delta y \le y_{max}$
- Rearranging, we get
  - $-u\Delta x \le (x_1 - x_{min})$
  - $u\Delta x \le (x_{max} - x_1)$
  - $-u\Delta y \le (y_1 - y_{min})$
  - $u\Delta y \le (y_{max} - y_1)$
    - In general: $u * p_k \le q_k$

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
     - $r_k = q_k / p_k$
     - $u_1 = \max(0, r_k, u_1)$

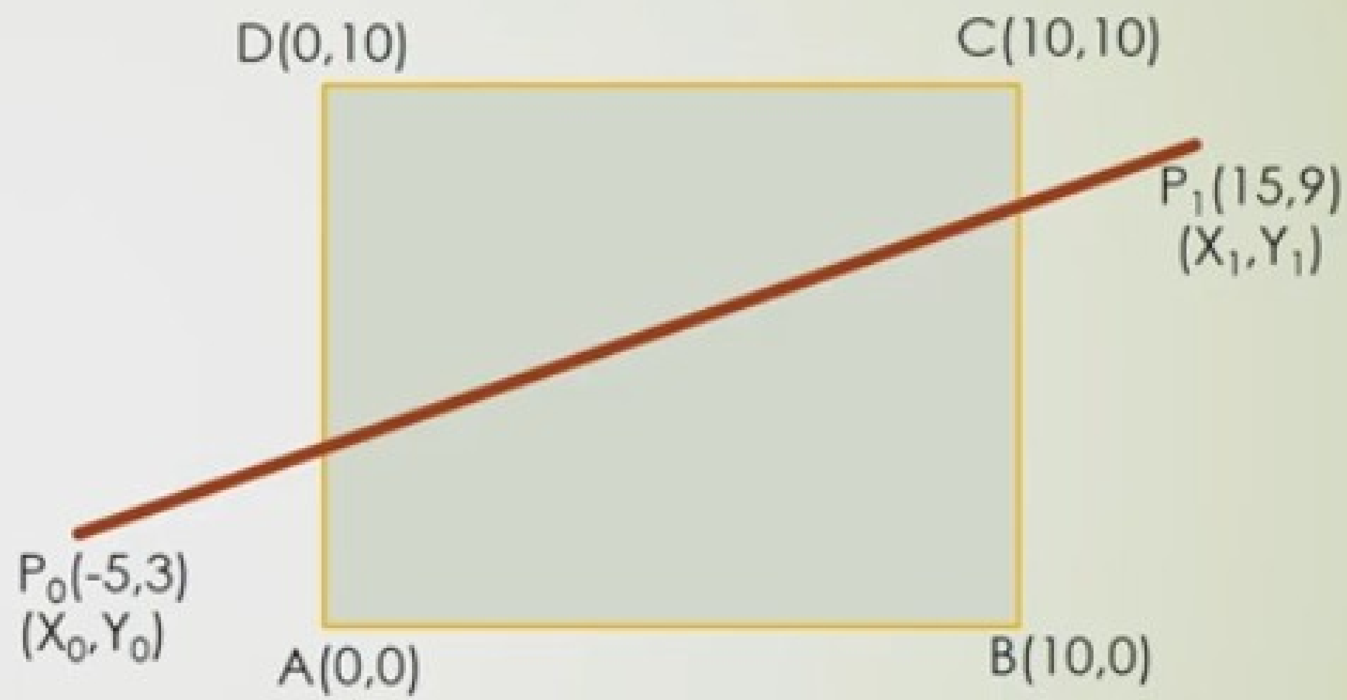
Liang-Barsky Line Clipping

- Cases: (cont'd)

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

Example

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

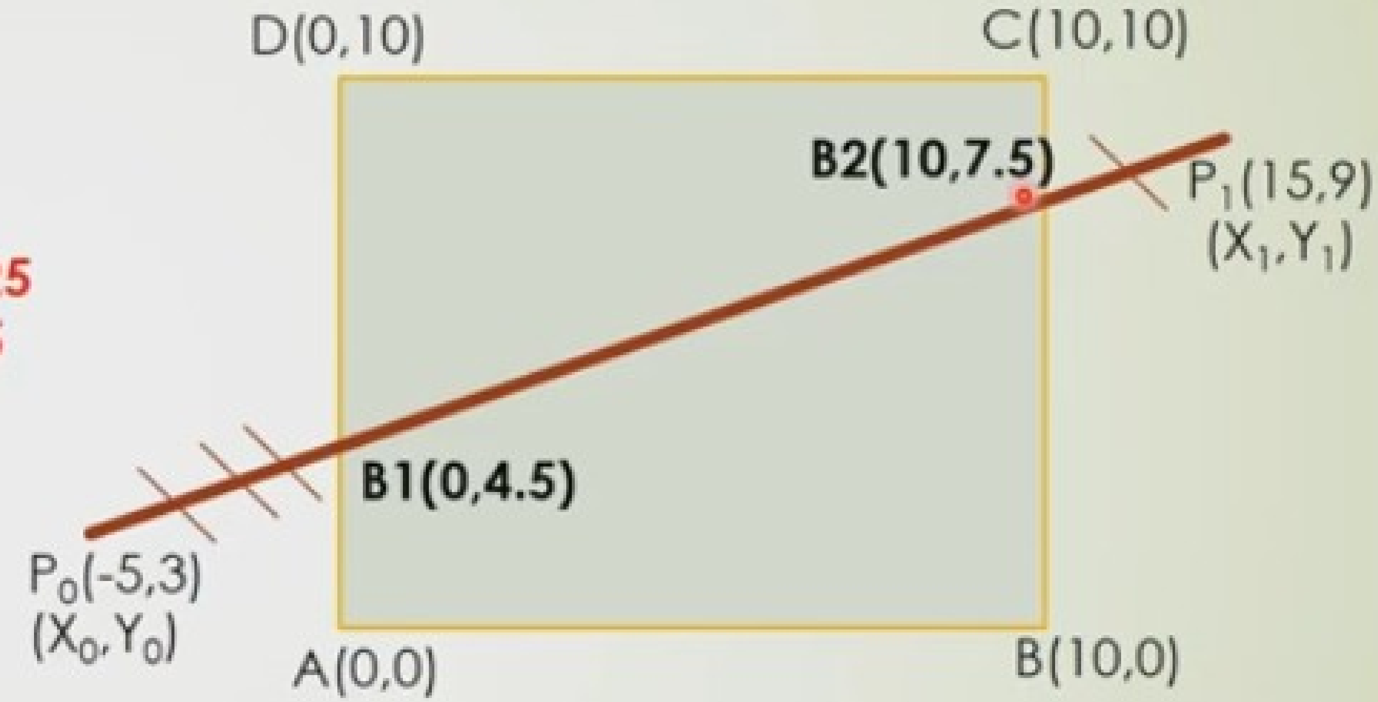
We consider **U0=0.25 and U1=0**

Calculate Intersection Points

SUBSCRIBE

Example

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 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

1.When u=0.25

(x,y)=(0,4.5)

X=X0+U ▲X

X=-5+0.25*20=0, Y=Y0+U▲Y=3+0.25*6=4.5

2..When u=0.75

(x,y)=(10,7.5)

X=X0+U ▲X

X=-5+0.75*20=10,

Y=Y0+U▲Y=3+0.75*6=7.5

SUBSCRIBE

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

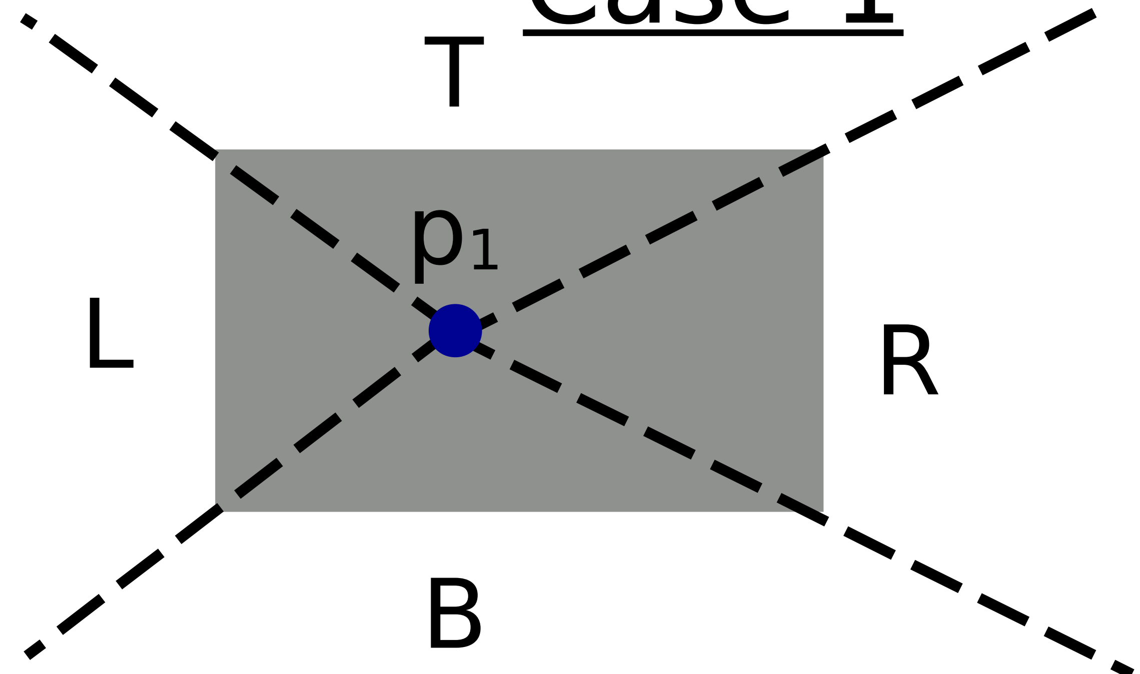
Nicholl-Lee-Nicholl Line Clipping

- 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

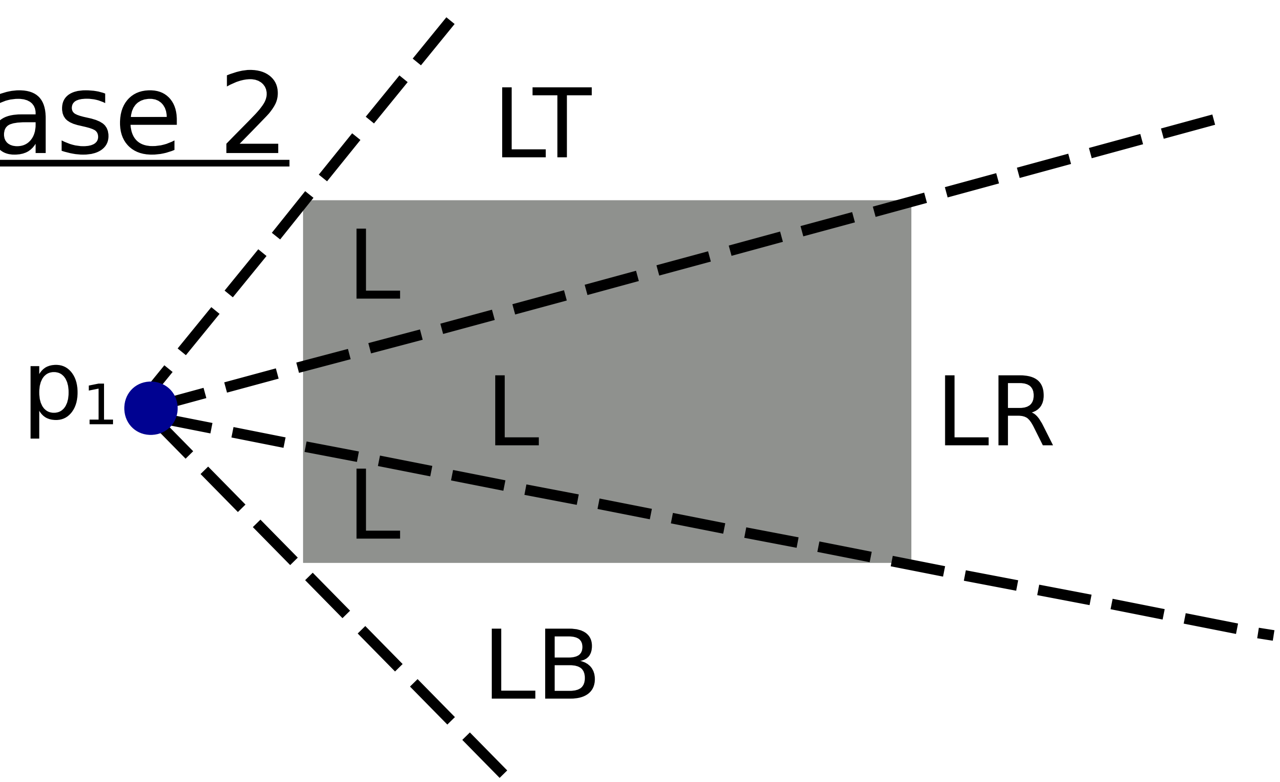
Nicholl-Lee-Nicholl Line Clipping

- 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

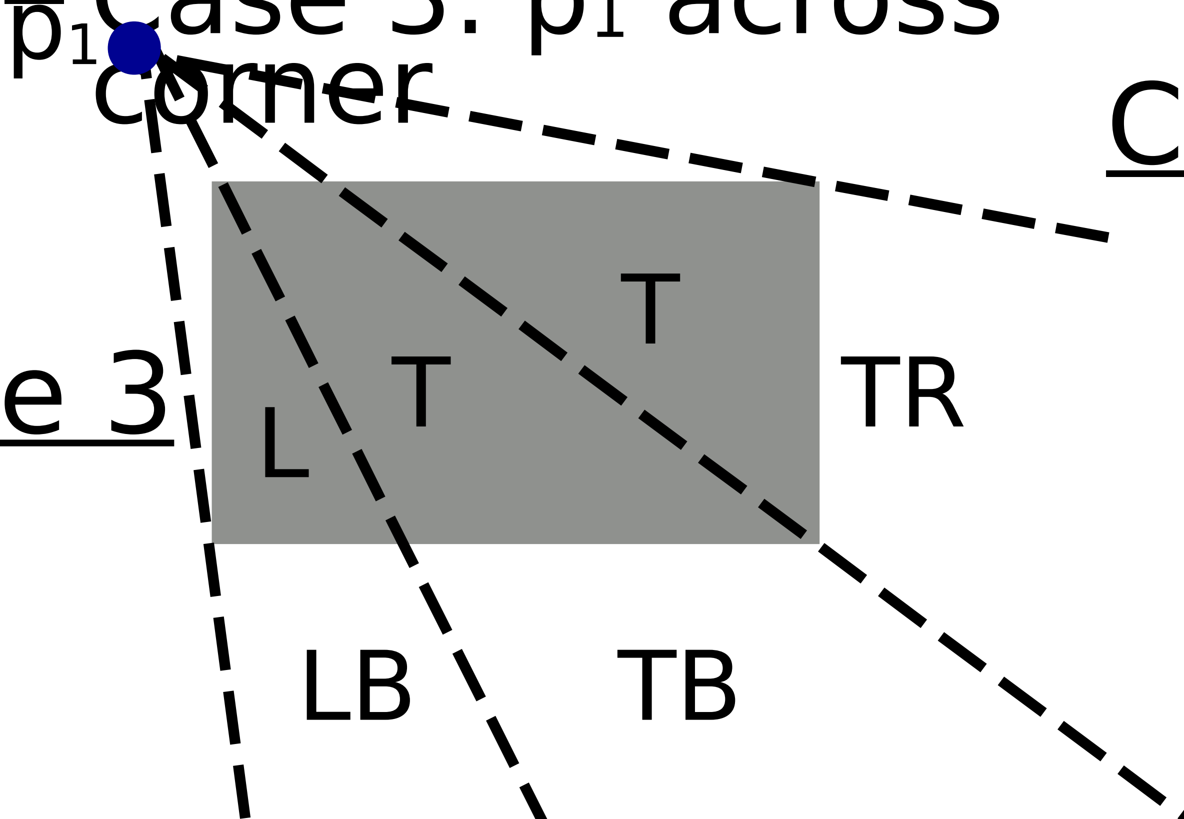
Case 1



Case 2



Case 3



Nicholl-Lee-Nicholl Line Clipping

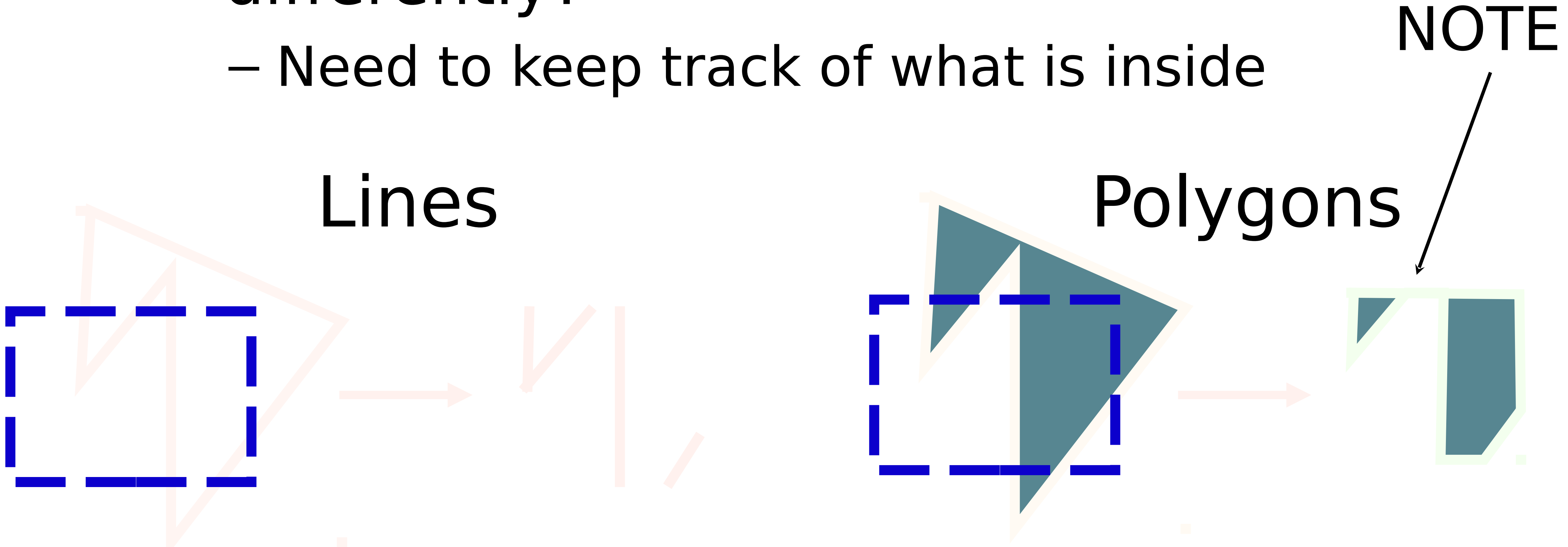
- Can use symmetry to handle all other cases
- “Algorithm” (really just a sketch):
  - Find slopes of the line and the 4 region bounding lines
  - Determine what region p_2 is in
    - If not in a labeled region, discard
    - If in a labeled region, clip against the indicated sides

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

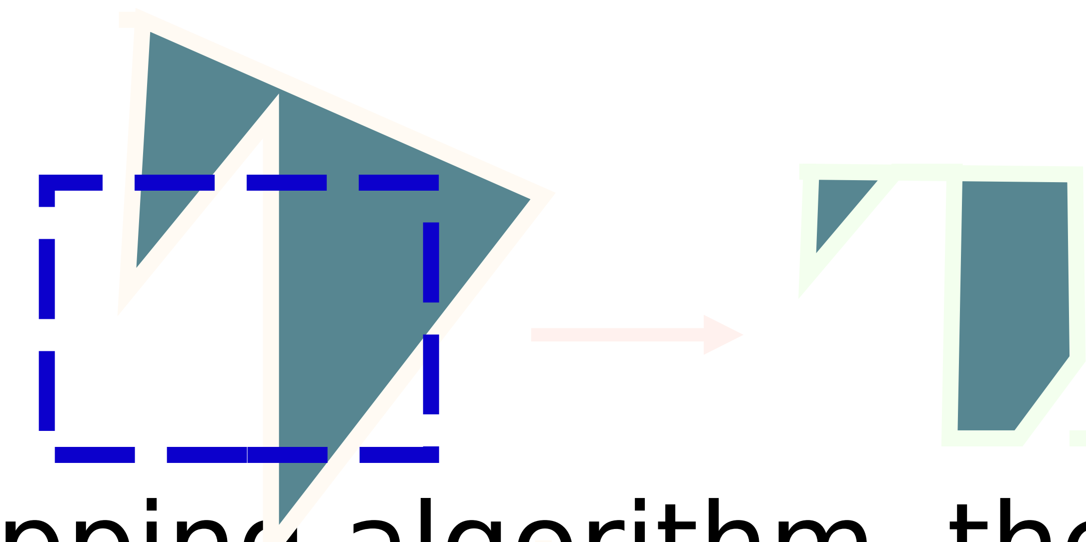
- Polygons are just composed of lines. Why do we need to treat them differently?
  - Need to keep track of what is inside



Weiler-Atherton Polygon Clipping

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

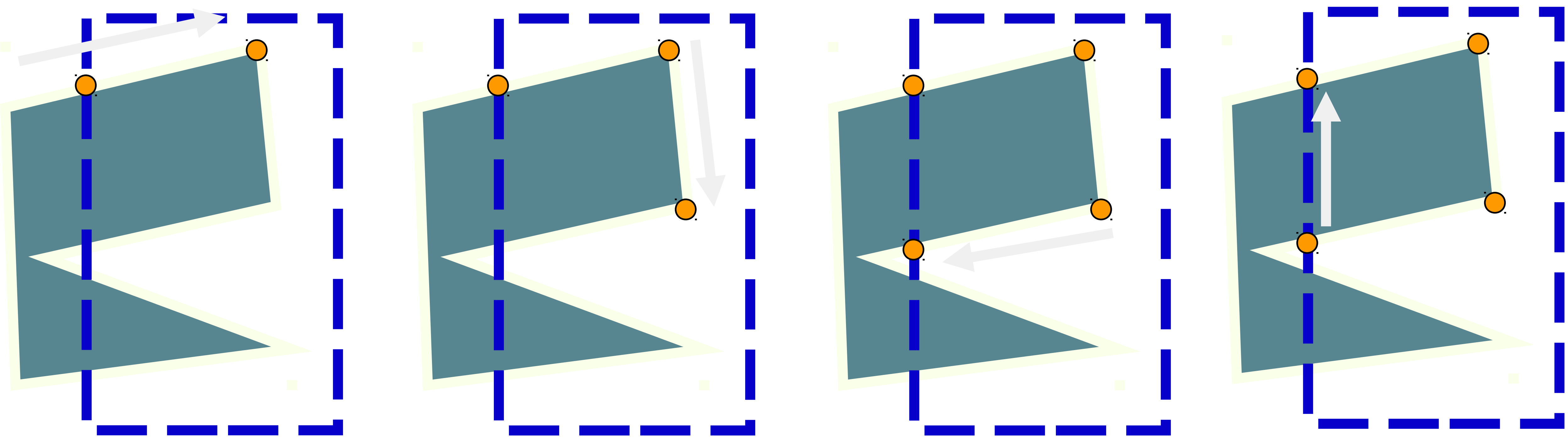
Remember this?



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

Weiler-Atherton Polygon Clipping

- Example:



Out -> In
Add clip vertex
Add end vertex

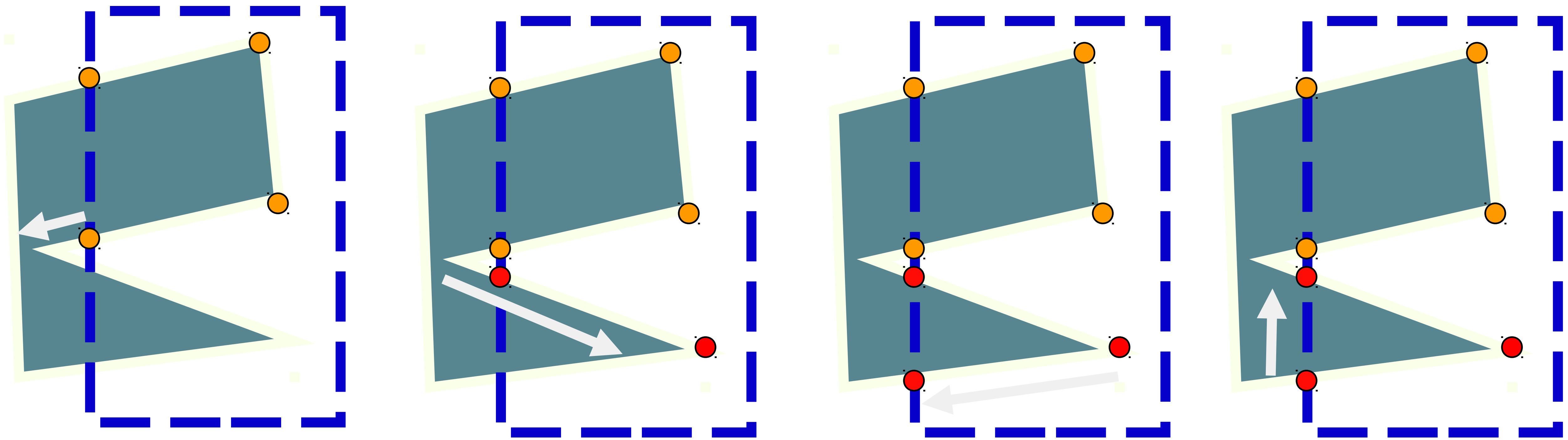
In -> In
Add end vertex

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

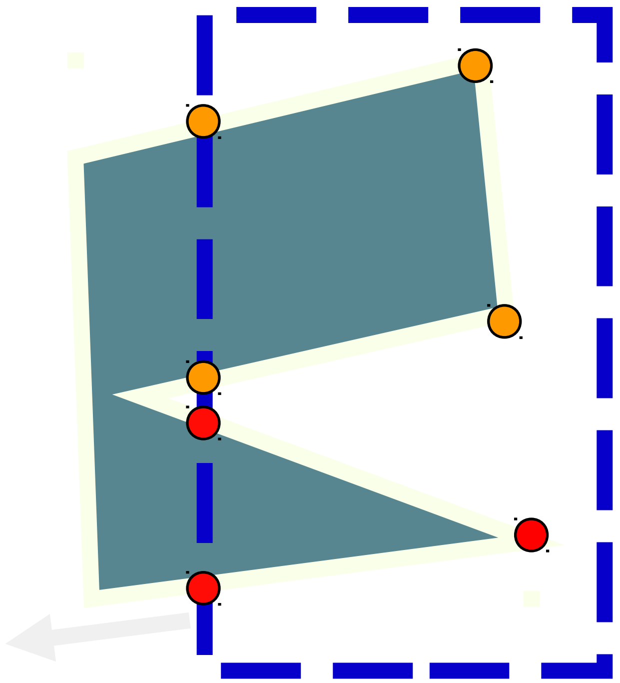
Out -> In
Add clip vertex
Add end vertex

In -> Out
Add clip vertex
Cache old direction

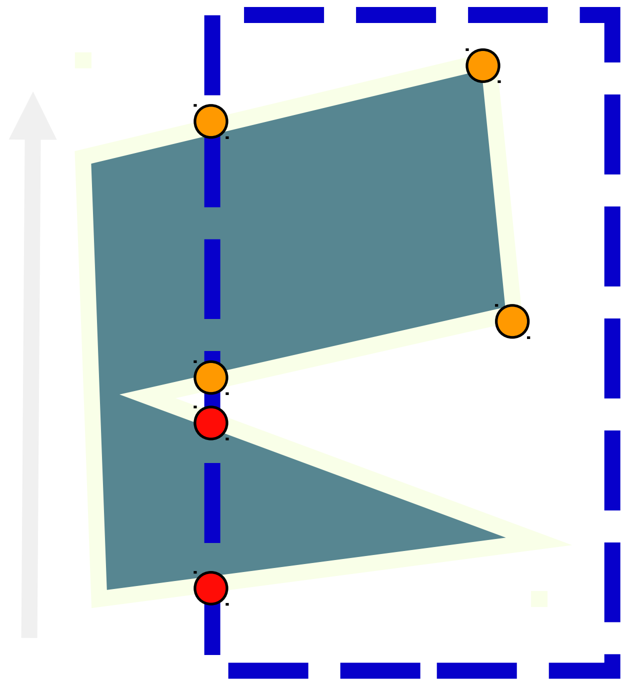
Follow clip edge until
(a) new crossing found
(b) reach vertex already added

Weiler-Atherton Polygon Clipping

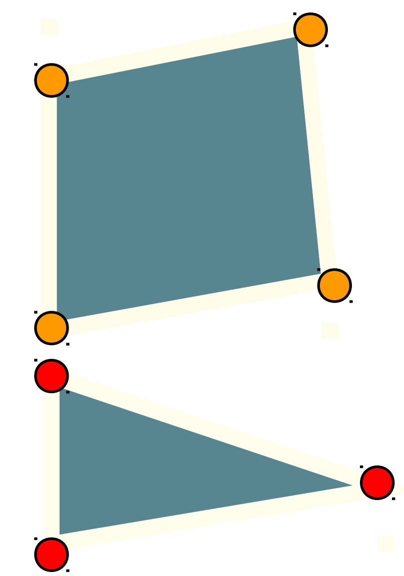
- Example (cont'd):



Continue from cached vertex and direction



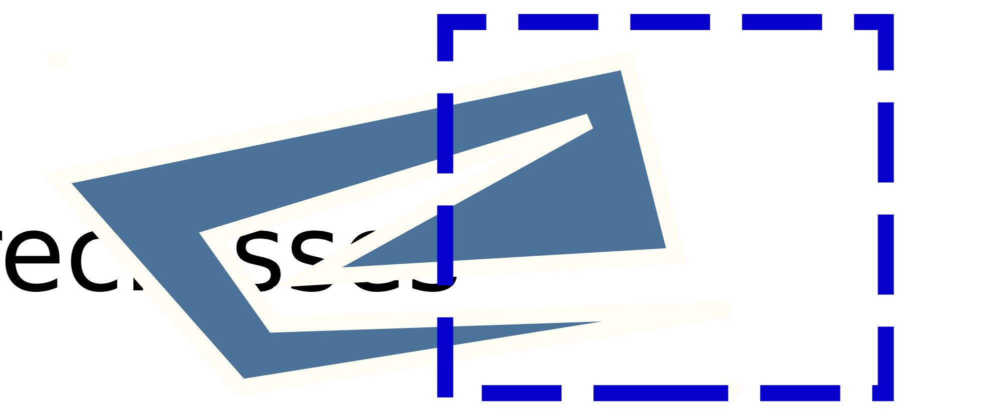
Nothing added
Finished

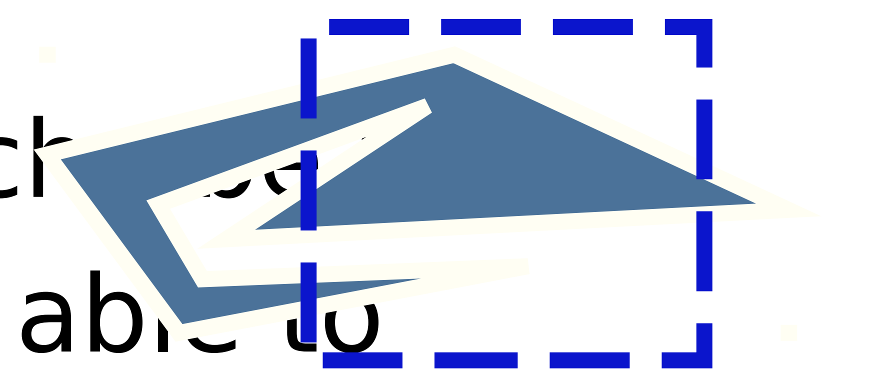


Final Result:
2 unconnected polygons

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
  - Liang-Barsky
  - Nicholl-Lee-Nicholl
- Polygon Clipping
  - Sutherland-Hodgeman
  - Weiler-Atherton

Any Questions?