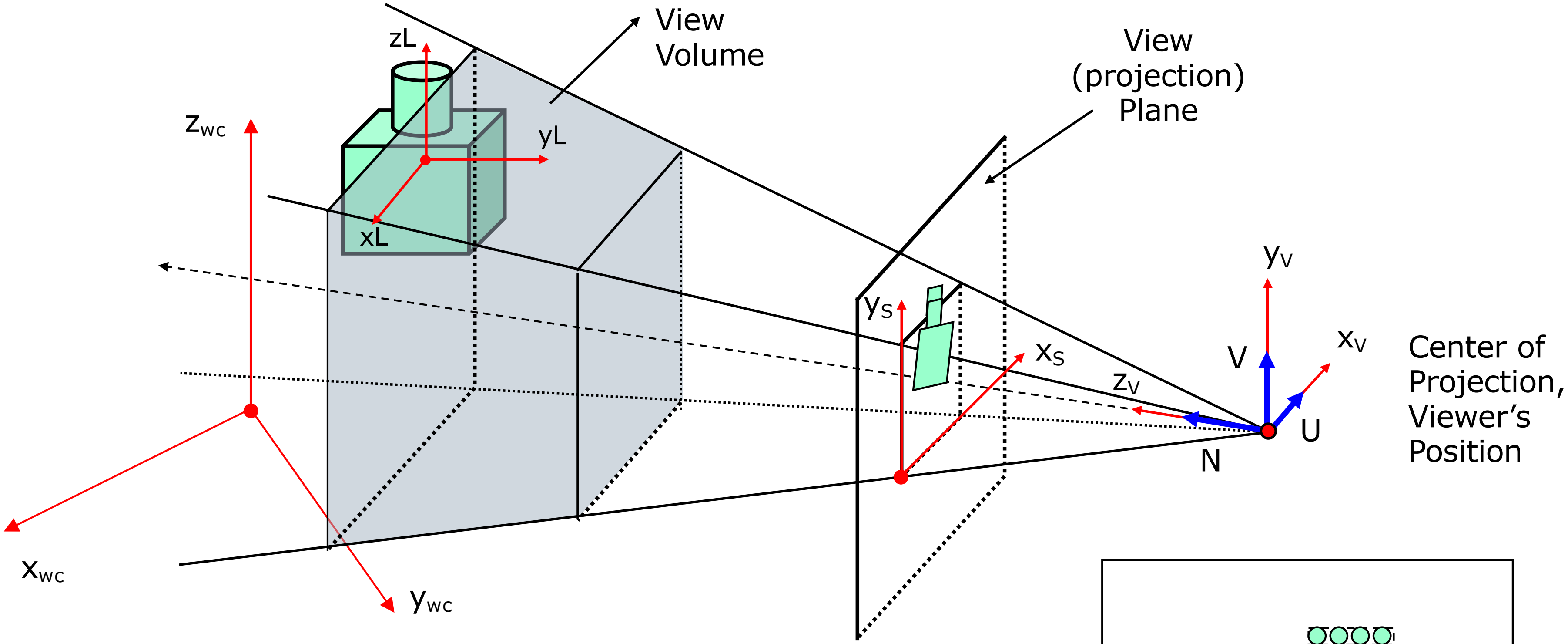
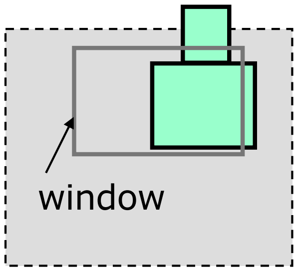
2D Clipping

Contents

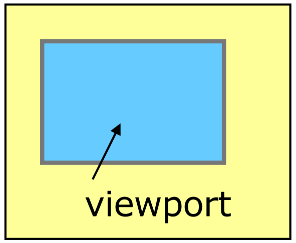
- Point
- Line Clipping
- Polygon Clipping
- Text Clipping

Real objects to image on the screen

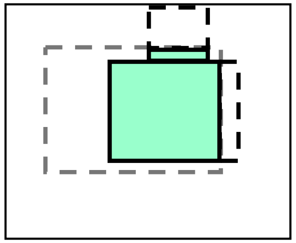




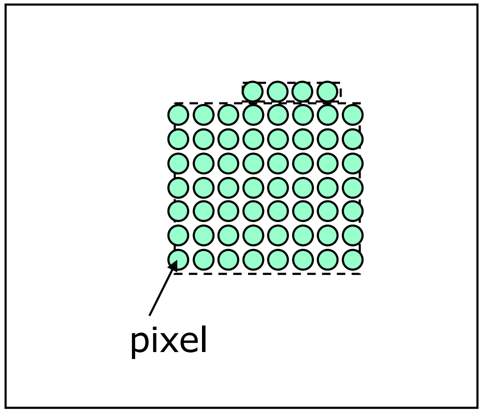
Window definition in projection plane



Viewport definition in projection plane

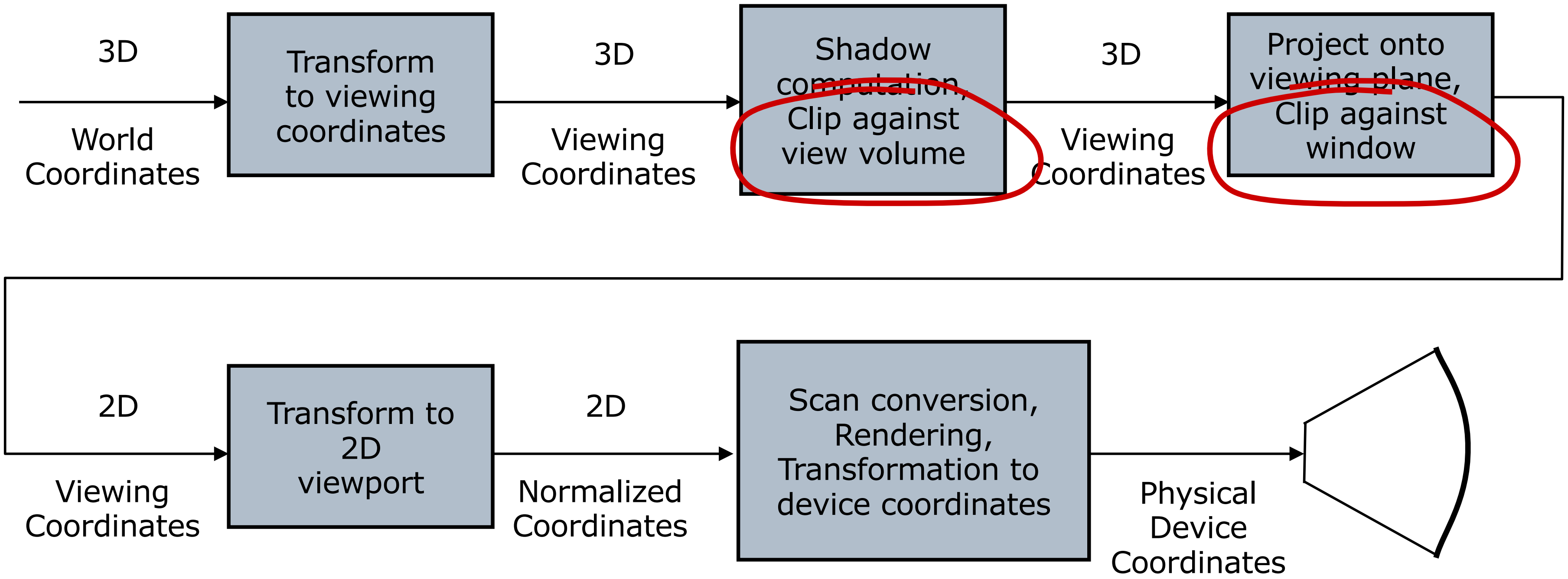


Window's content inside the viewport

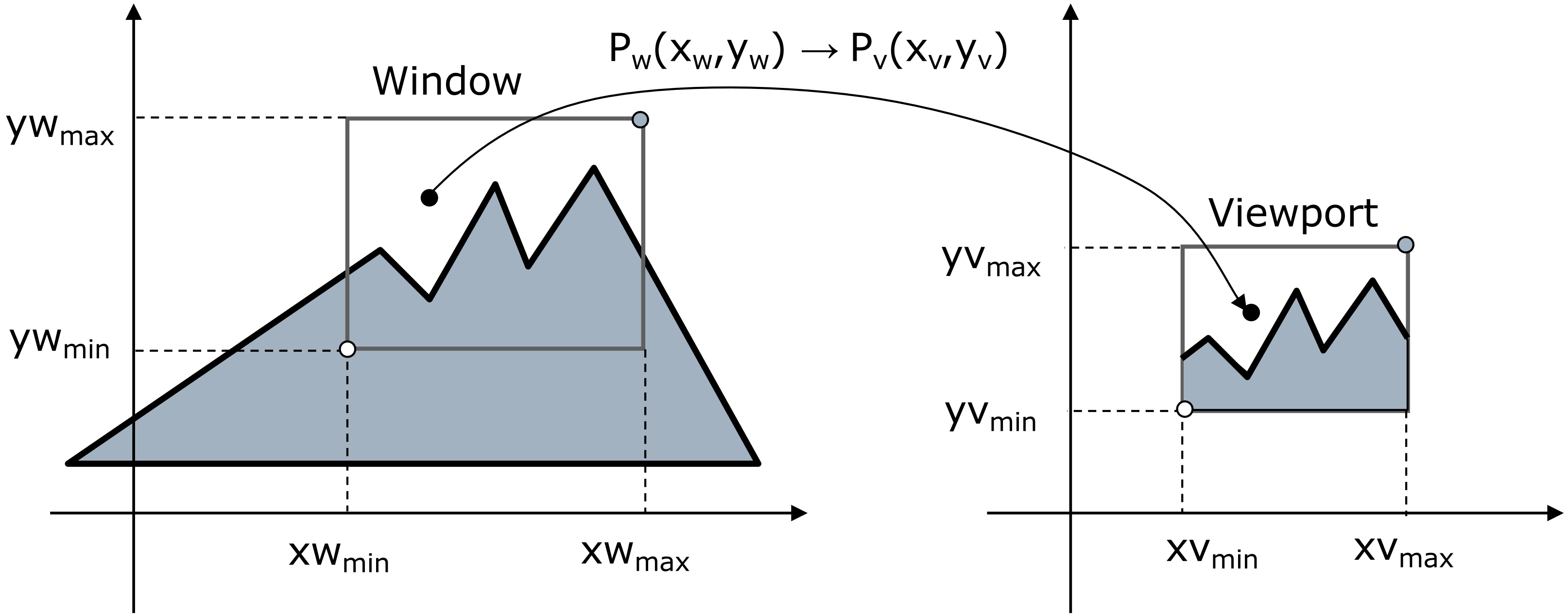


Scan conversion onto the raster screen

Viewing transformations pipeline



Window and viewport



World Coordinates

Normalized Device Coordinates

Window definition: $(x_{wmin}, x_{wmin}, x_{wmax}, y_{wmax})$

Viewport def: $(x_{vmin}, x_{vmin}, x_{vmax}, y_{vmax})$

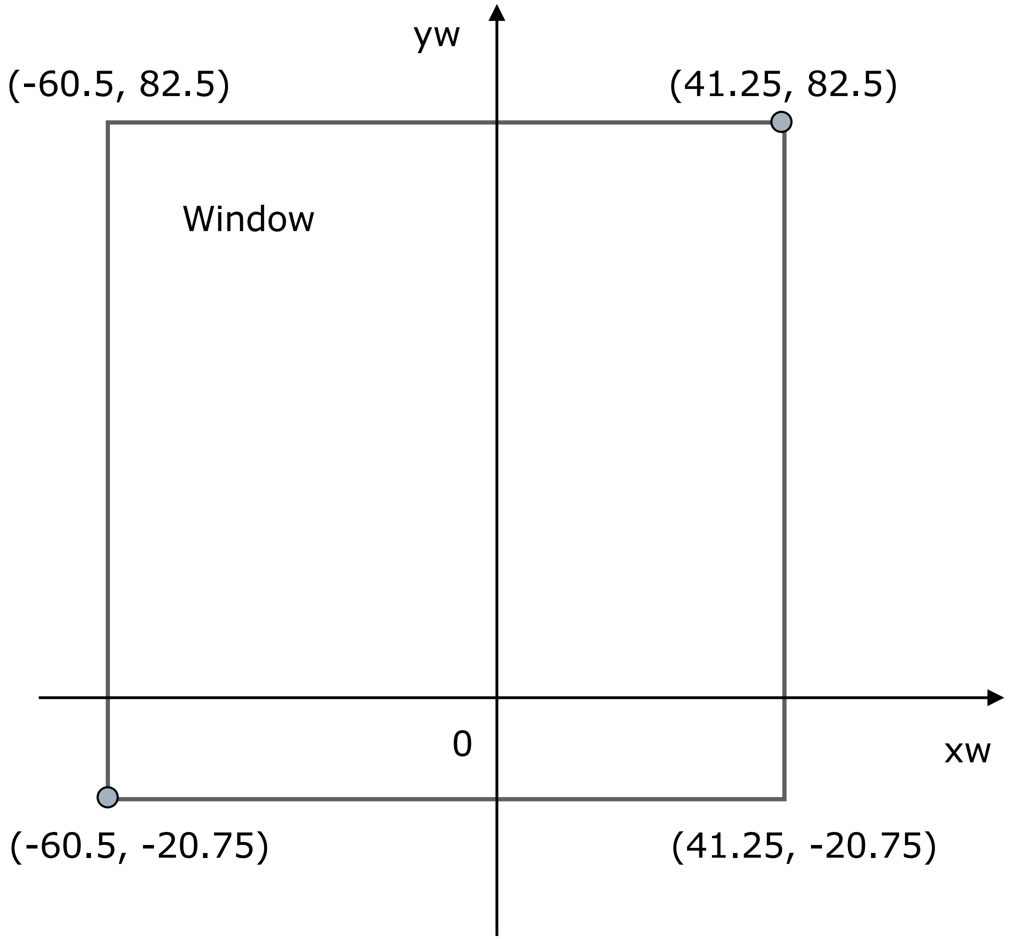
$$\frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}} = \frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}}$$

$$x_v = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}} (x_w - x_{wmin}) + x_{vmin}$$

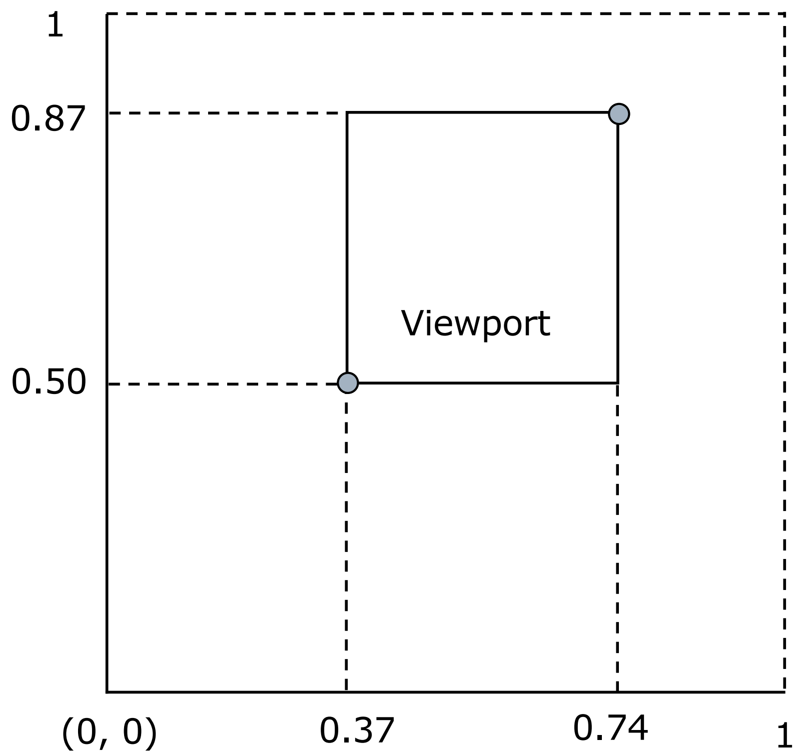
$$x_v = sx\ (x_w - x_{wmin}) + x_{vmin}$$
$$y_v = sy\ (y_w - y_{wmin}) + y_{vmin}$$

$$y_v = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}} (y_w - y_{wmin}) + y_{vmin}$$

Window and viewport - specification

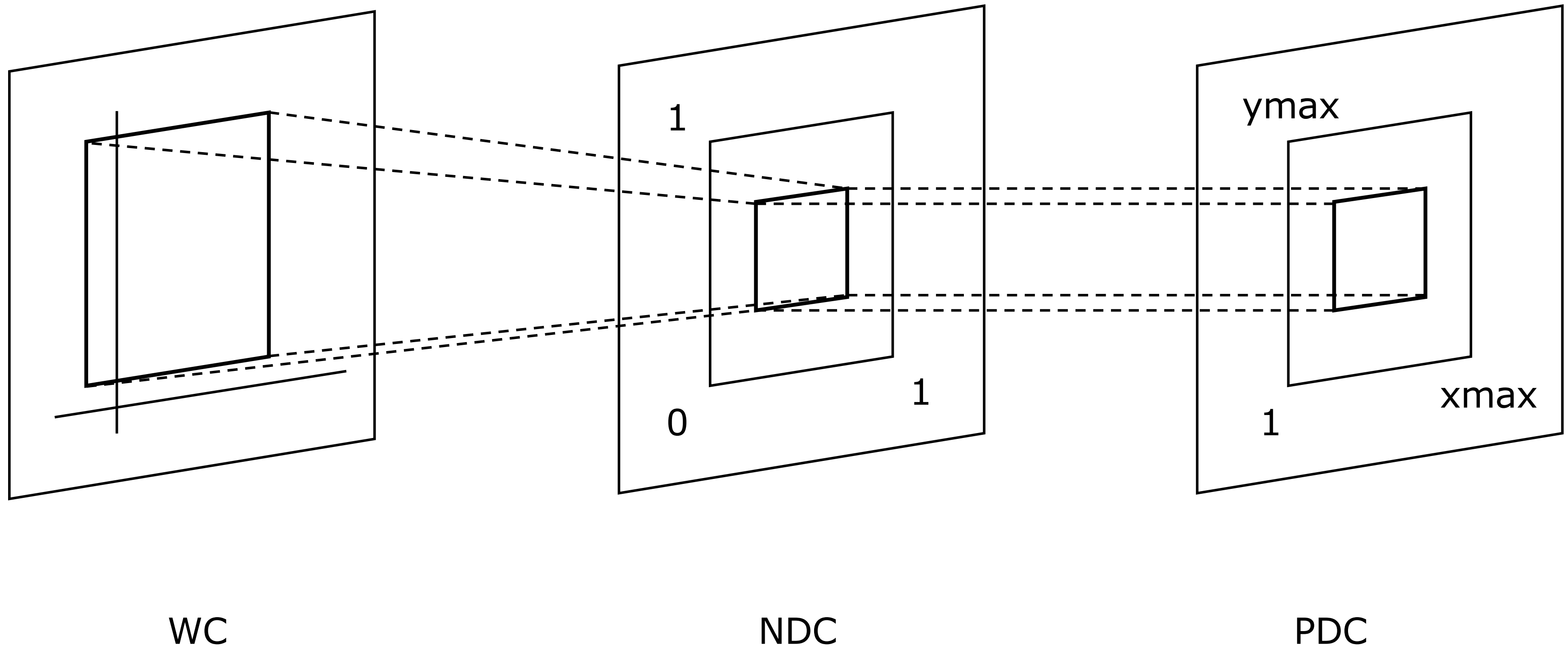


World Coordinates



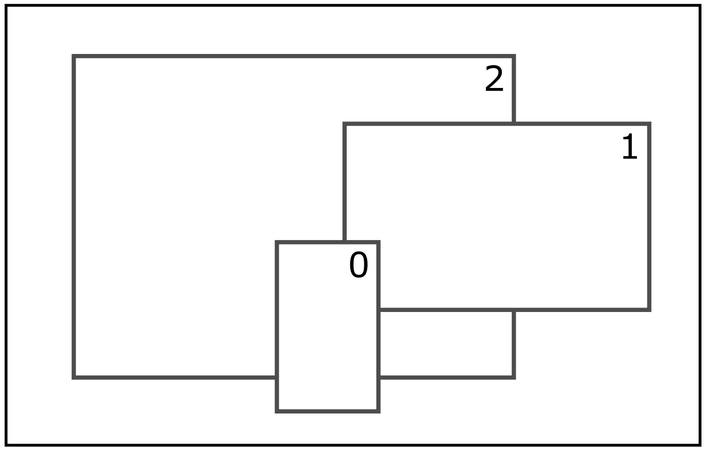
Normalized Coordinates

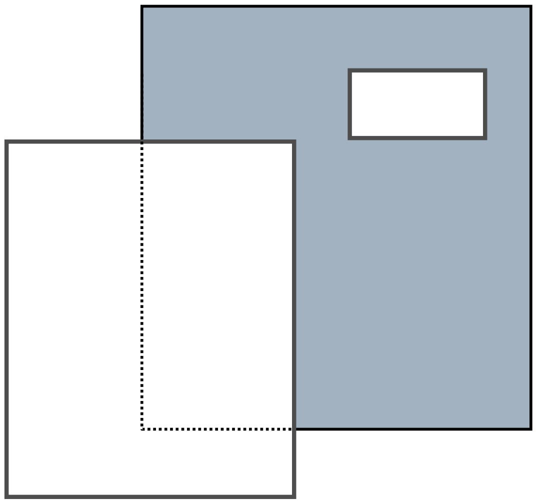
Window and viewport transformations

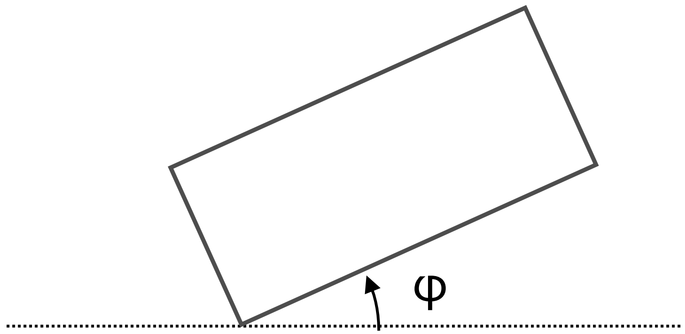


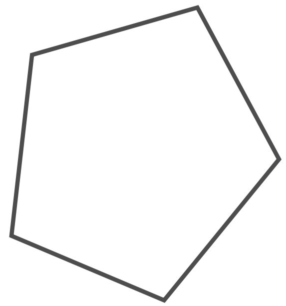
Window and viewport types

- multiple windows / viewports
- multiple workstations
- polygonal windows / viewports
- zooming / panning
- blanking







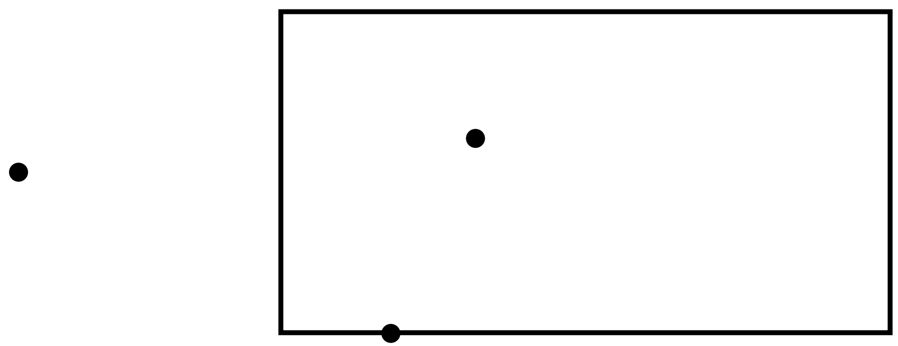


Point clipping

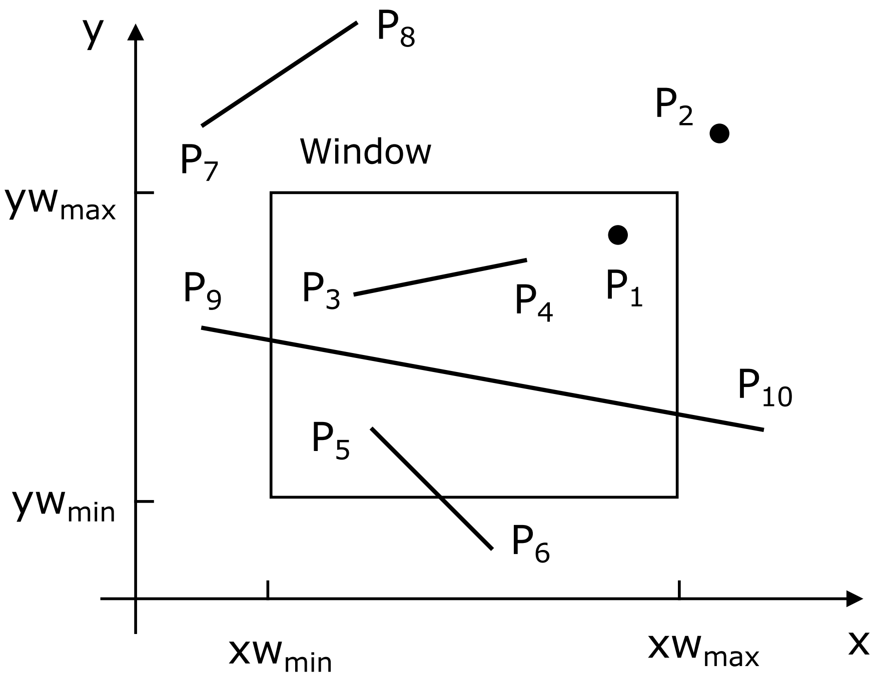
- Inside outside test:

  $xmin \le x \le xmax$

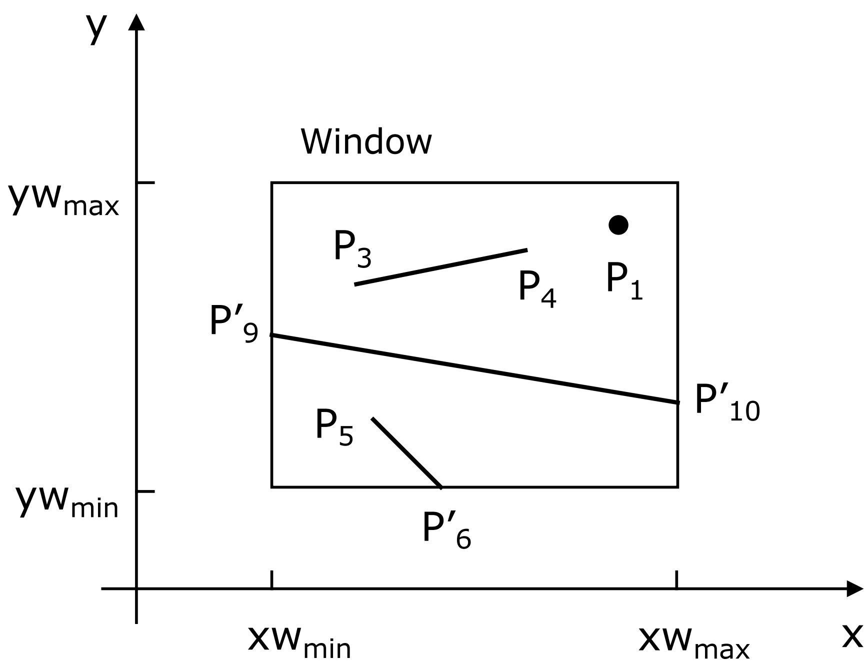
  $ymin \le y \le ymax$



Line clipping



Before clipping



After clipping

Line Clipping

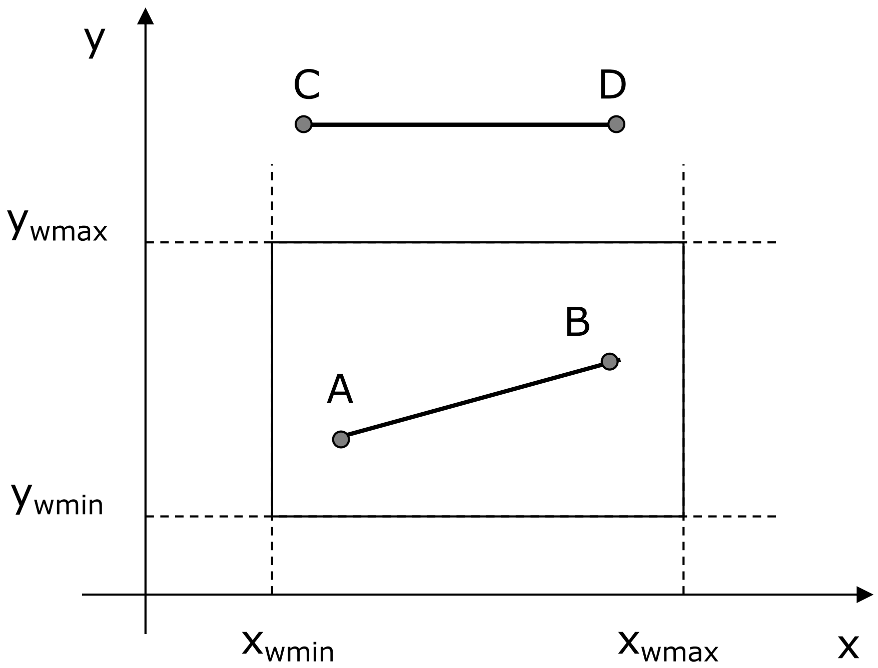
Line definitions

- Set of points ->
  - point clipping
- Vector ->
  - endpoints clipping
  - binary clipping
  - parametric line clipping
  - Cohen-Sutherland algorithm

Endpoints clipping

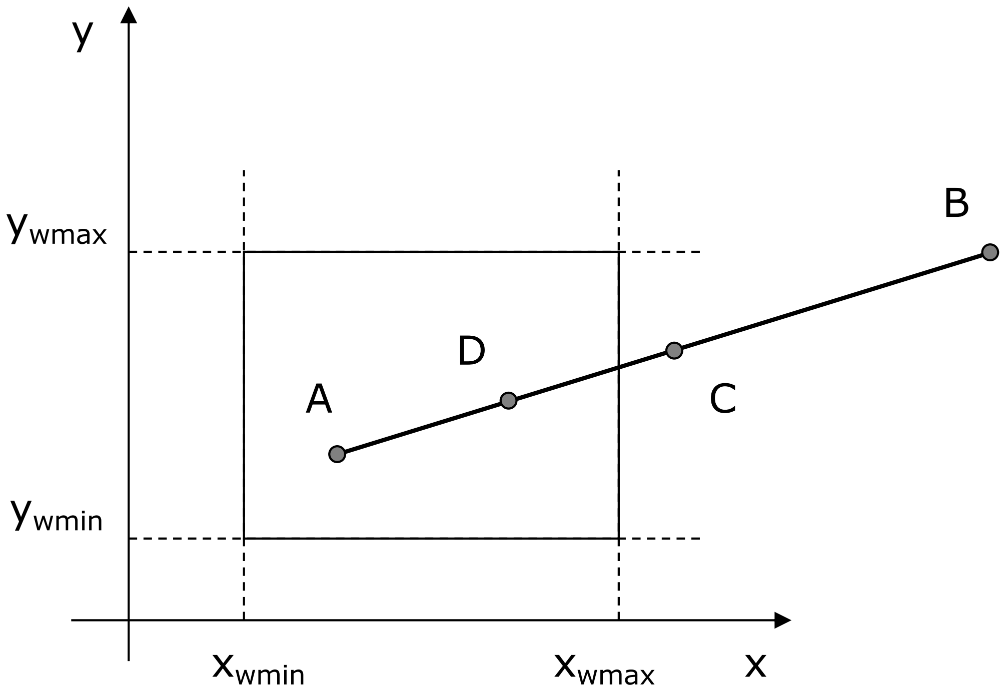
Could be two cases:

1. trivially accepted (AB)
2. trivially rejected (CD)



Binary clipping

1. step: AB
2. step: AC, CB (tr. rejected)
3. 3 step: AD (tr. acc), DC
4. . . .



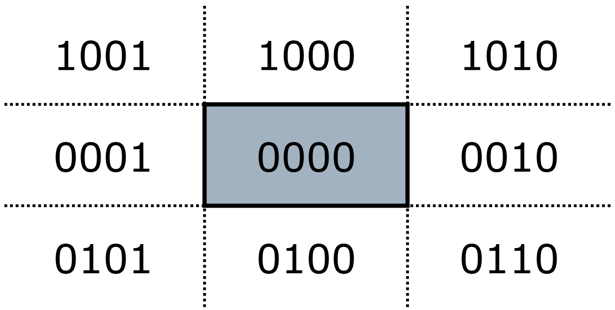
Cohen-Sutherland clipping algorithm

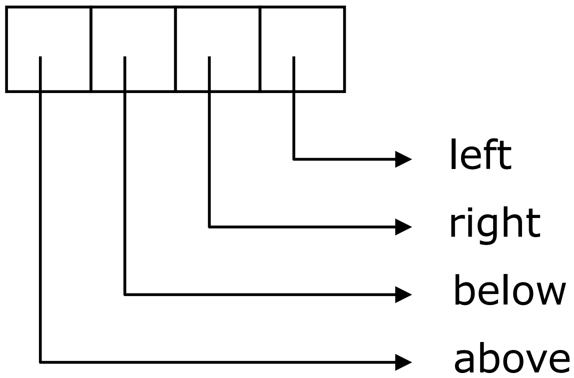
Basic Idea

- Encode the line endpoints
- Successively divide the line segments so that they are completely contained in the window or completely lies outside window
- Division occurs at the boundary of window

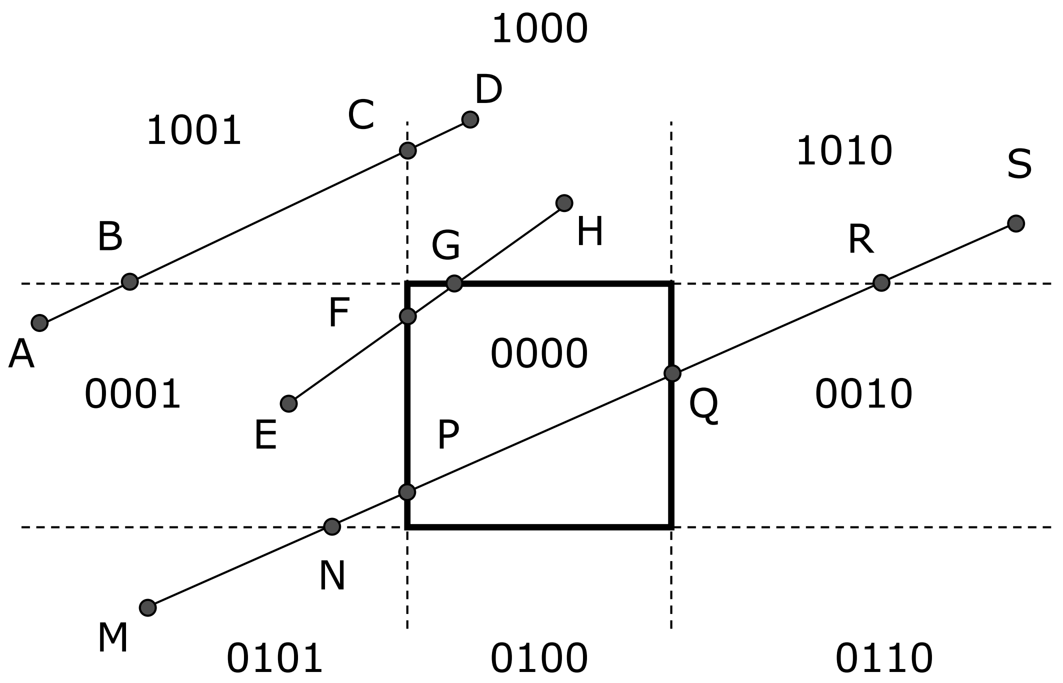
Cohen-Sutherland clipping algorithm

Region outcodes:





Cohen-Sutherland clipping algorithm



Cohen-Sutherland Algorithm

```
procedure CSDecLinie( x0, y0, x1, y1,
          min, xmax, ymin, ymax: real;
          color: integer);
{ Cohen-Sutherland clipping algorithm for  line P0(x0,y0) to  P1(x1,y1), and clip
   rectangle with diagonal from (xmin, ymin) to (xmax, ymax) }
type      edge = (LEFT, RIGHT, BOTTOM, TOP);
          outcode = set of edge;

var       accept, done : boolean;
          outcod0, outcod1, outcodt : outcode;   {outcodes for  P0, P1, and
                                                  whichever point lies outside
                                                  the clipping rectangle}
          x, y : real;

procedure CompOutCode( x, y: real; var code: outcode);
{ computes outcode for the point (x,y)}
begin
   code := [];
   if y > ymax then code:= [TOP]
   else if y < ymin then code:= [BOTTOM];
   if x > xmax then code:= code+[RIGHT]
   else if x < xmin then code:= code+[LEFT]
end;
```

```
Begin
    accept := false; done := false;
    CompOutCode(x0, y0, outcod0);
    CompOutCode(x1, y1, outcod1);
    repeat
        if (outcod0 = []) and (outcod1 = []) then  {trivially accepted}
                begin accept := true; done := true end {end exit}
        else if (outcod0 * outcod1) <> [] then
                done := true  {Logical intersection is true, so trivial reject and exit}
        else
                {Failed both tests, so calculate the line segment to clip:
                from an outside point to an intersection with clip edge}
                begin {at least one endpoint is outside the clip rectangle; pick it.}
                        if outcod0 <> [] then
                                outcodt := outcod0 else outcodt := outcod1;
                        {now find intersection point; use formulas
                                y = y0 + slope * (x- x0),
                                x = x0 + (1/slope) * (y-y0)}
                        if TOP in outcodt then
                                begin
                                {divide line at top of clip rectangle}
                                x := x0+(x1-x0)*(ymax-y0)/(y1-y0);
                                y := ymax
                                end
                        else if BOTTOM in outcodt then
                                begin
                                { divide line at bottom of clip rectangle }
                                x := x0+(x1-x0)*(ymin-y0)/(y1-y0);
                                y := ymin
                                end
```

```
                    else if RIGHT in outcodt then
                            begin
                            { divide line at right of clip rectangle }
                            y := y0+(y1-y0)*(xmax-x0)/(x1-x0);
                            x := xmax
                            end
                    else if LEFT in outcodt then
                            begin
                            { divide line at left of clip rectangle }
                            y := y0+(y1-y0)*(xmin-x0)/(x1-x0);
                            x := xmin
                            end;
                    {Now we move outside point to intersection point to clip,
                    and get ready for next pass. }
                    if (outcodt = outcod0) then
                            begin
                            x0 := x; y0 := y;
                            CompOutCode(x0, y0, outcod0)
                            end
                    else
                            begin
                            x1 := x; y1 := y;
                            CompOutCode(x1, y1, outcod1)
                            end
            end {Subdivide}
    until done;
    if accept then MidpointLineReal(x0, y0, x1, y1, color)
        {draw a line with real endpoint coordinates}
End; {CSDecLinie Cohen Sutherland clipping algorithm}
```

Parametric line clipping

Basic approach:

1. Oriented line segment and window edge,

   e.g. line from (x_0,y_0) to (x_1, y_1):

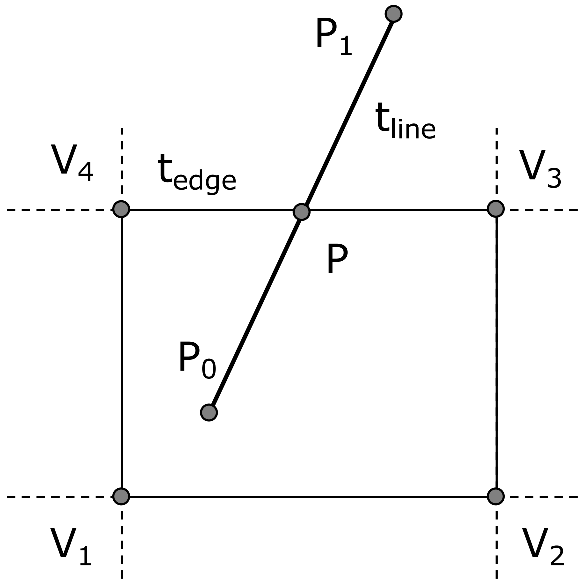
   $P = P_0 + t_{line} \cdot (P_1 - P_0)$

or

   $x = x_0 + t_{line} \cdot (x_1 - x_0)$

   $y = y_0 + t_{line} \cdot (y_1 - y_0)$

where $t_{line} \in [0, 1]$



2. Solve the system of two equations:

   $P = P_0 + t_{line} \cdot (P_1 - P_0)$

   $P = V_i + t_{edge} \cdot (V_{i+1} - V_i)$

3. The line and edge intersects each other if both t_{edge} and $t_{line} \in [0, 1]$

4. Solution: the clipped line is $P_0P(t)$, $t_{line} \in [0, 1]$

Other parametric line clipping algorithms:

1978 Cyrus-Beck algorithm

1984 Liang-Barsky algorithm

   (improved form of the Cyrus-Beck algorithm)

Cyrus-Beck algorithm

- Published in 1978 by Cyrus, M., Beck, J, "Generalized Two- and Three-Dimensional Clipping", Computers and Graphics, vol 3(1).

Dot product:

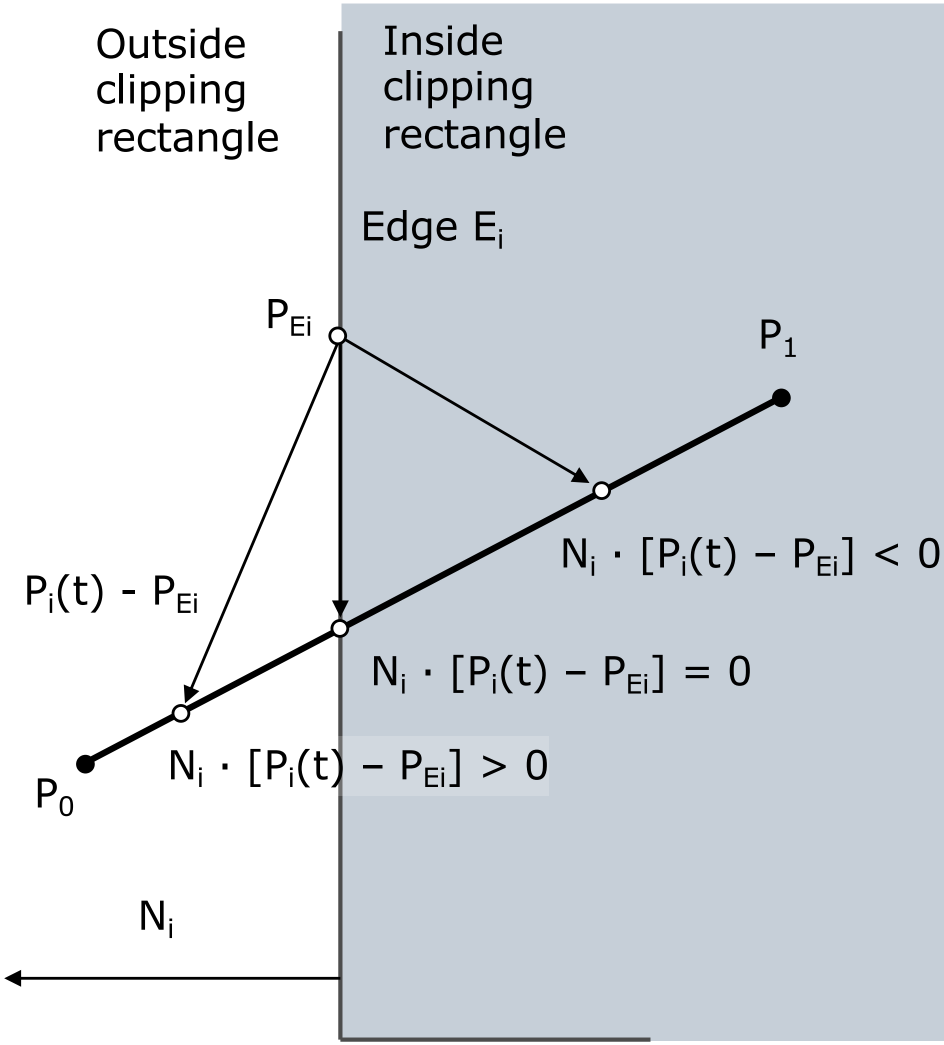
$N_i \cdot [P_i(t) - P_{Ei}] =$

$\quad |N_i| \cdot |P_i(t) - P_{Ei}| \cdot \cos \varphi$

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

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

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



Cyrus-Beck algorithm

Solve the equation

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

Substitute $P(t) = P_0 + (P_1 - P_0) \cdot t$

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

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

Let $D = (P_1 - P_0)$ be the vector from P_0 to P_1, then

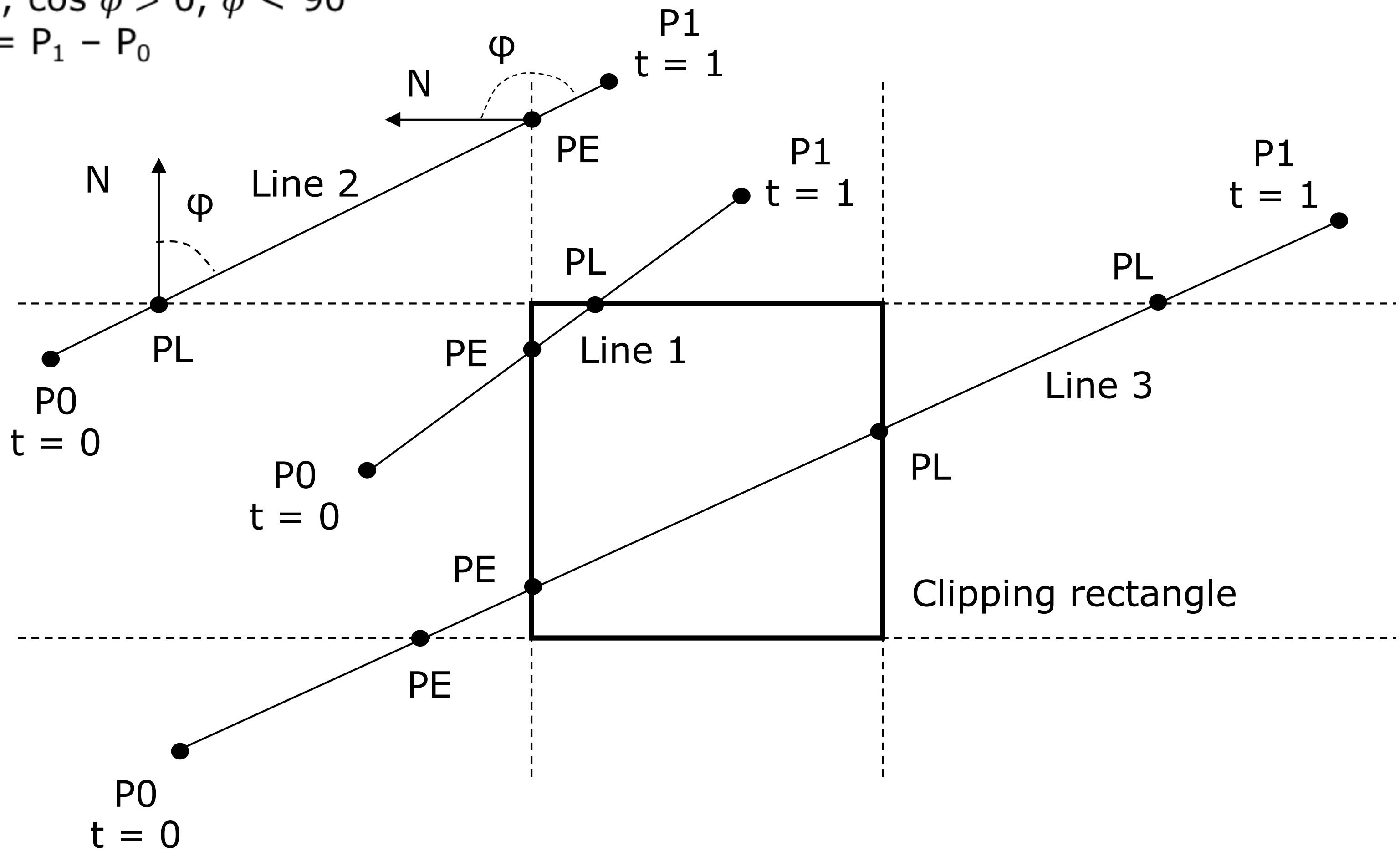
$t = N_i \cdot [P_0 - P_{Ei}] / (- N_i \cdot D)$

The algorithm checks that

$N_i \neq 0$	the normal should not be 0 (error case)
$D \neq 0$	$P_1 \neq P_0$
$N_i \cdot D \neq 0$	the edge Ei and the line P_0 to P_1 are not parallel

Cyrus-Beck algorithm

PE = entering point
  $N_i \cdot D < 0$, $\cos\varphi < 0$, $\varphi > 90°$
PL = leaving point
  $N_i \cdot D > 0$, $\cos\varphi > 0$, $\varphi < 90°$
  where $D = P_1 - P_0$



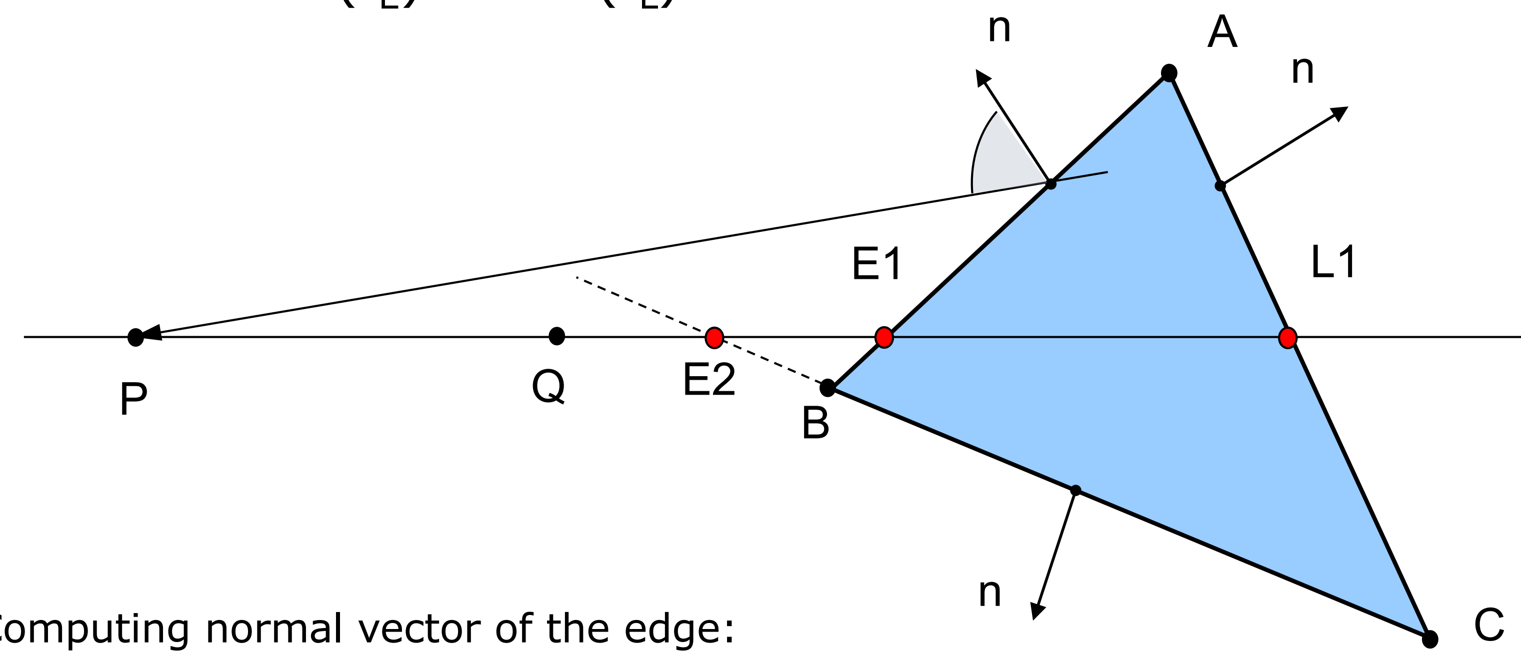
Cyrus-Beck algorithm - pseudocode

```
Begin
    precalculate Ni and select a PEi for each edge;
   for each line segment to be clipped
      if P1 = P0 then
         line degenerates to a point, so clip as a point;
      else
         begin
             tE = 0; tL = 1;
             for each candidate compute intersection with a clipping edge
                if Ni * D <> 0 then { ignore edges parallel to line for now}
                   begin
                       calculate t;
                       use sign of Ni * D to categorize as
                             PE (potentially entering) or PL (potentially leaving);
                       if PE then tE = max (tE , t);
                       if PL then tL = min (tL , t);
                   end;
             if tE > tL then
                return  nil
             else
                return P(tE ) and  P(tL ) as true clip intersections
         end
End
```

Intersection: line–convex polyhedron

Obs: triangle is always convex polygon

$\max(t_E) < \min(t_L)$



Computing normal vector of the edge:

1. Compute the normal vector of the plane

   $N = AB \times BC$

(cross product of two consecutive edges)

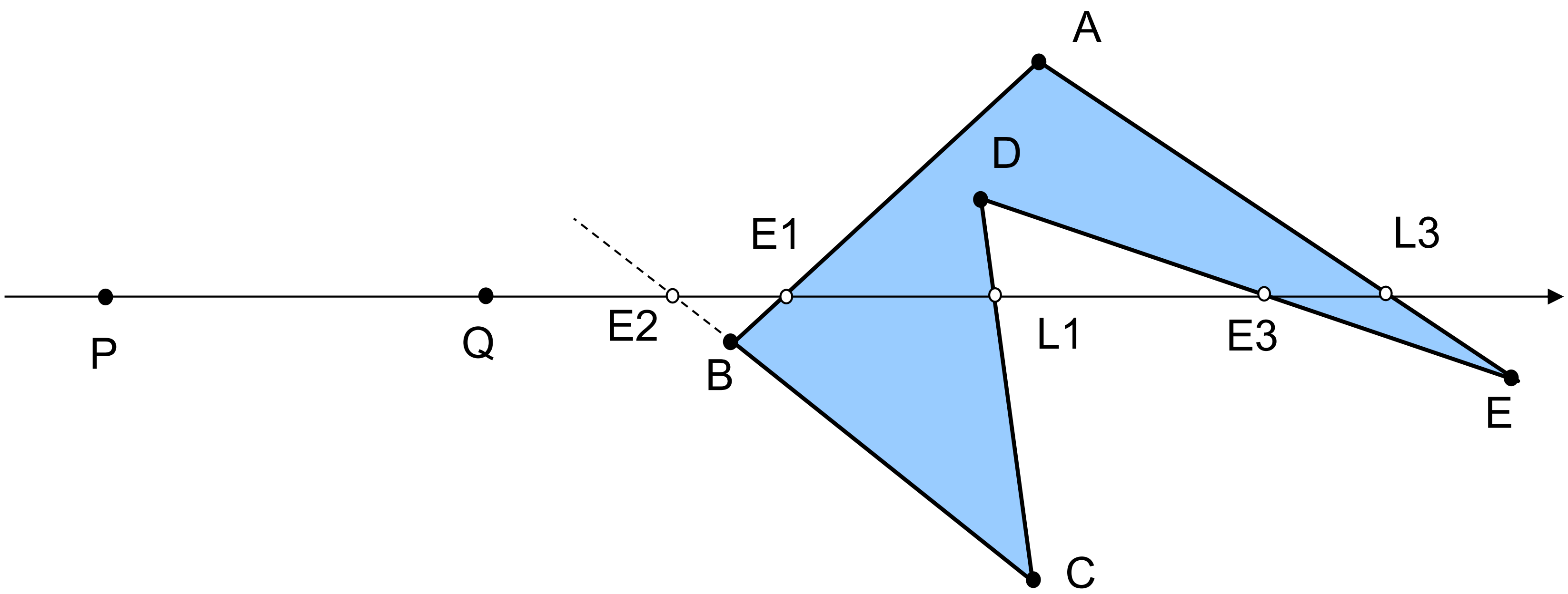
2. Normal vector of the edge

   $n = AB \times N$

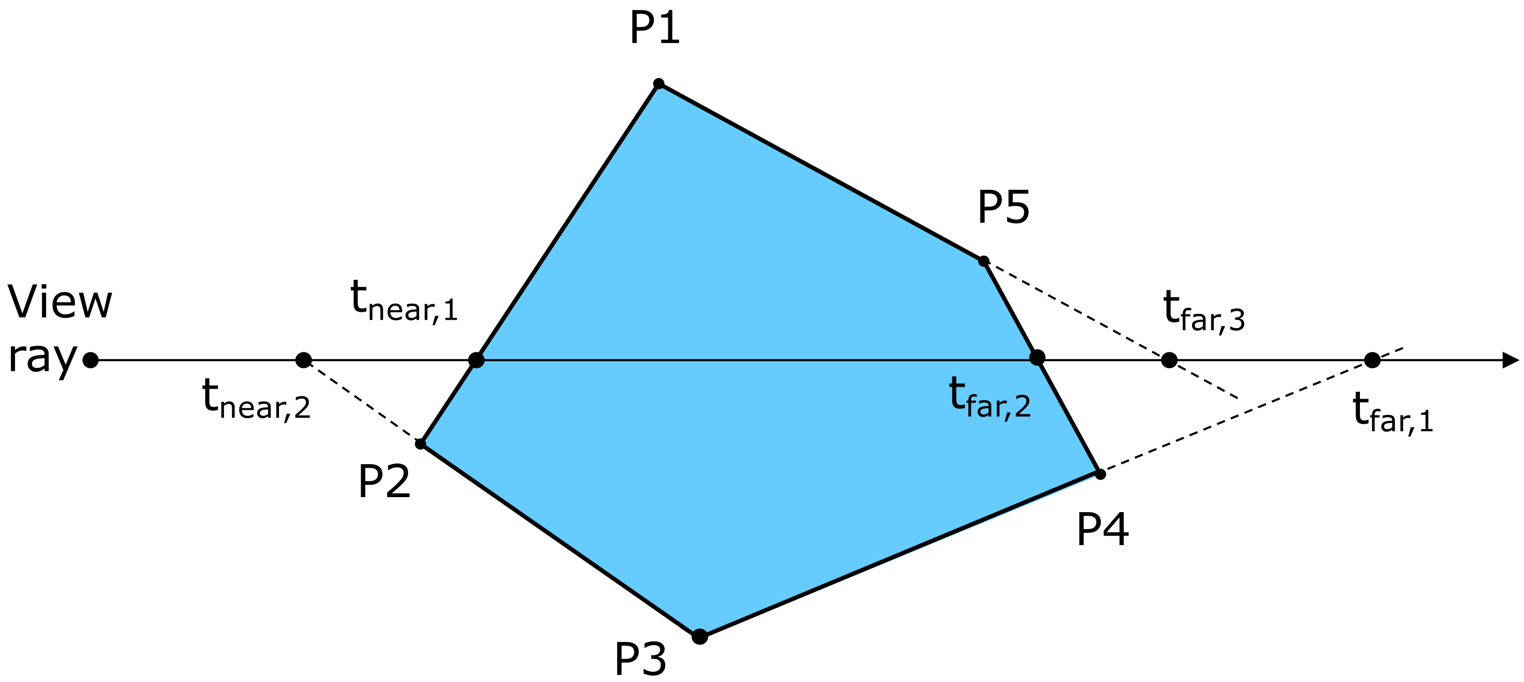
(cross product of the edge and normal vector of the plane)

Intersection: line–convex polyhedron

requires a convex polygon!



Intersection: line–convex polyhedron



Cyrus-Beck algorithm:

Classify the edge/face by frontface and backface;

For each edge/face compute intersection with view ray;

Compute max(t_{near}) and max(t_{far});

If max(t_{near}) < max(t_{far}) there is intersection;

Liang-Barsky line clipping algorithm

- Published in 1984 by Liang, Y-D., Barsky, B.A., “A New Concept and Method for Line Clipping”, ACM TOG, vol 3(1).

- Simplifies the Cyrus-Beck algorithm for the particular case of clipping window as an horizontal rectangle:

  1. One coordinate of each normal is 0.

  2. The dot product $N_i \cdot (P_0 - P_{Ei})$ determining whether the endpoint P_0 lies inside or outside a specific edge, reduces to the directed horizontal or vertical distance from the point to the edge.

  3. The denominator dot product $N_i \cdot D$, which determines whether the intersection is potentially entering or leaving, reduces to ±dx or dy:

     If dx >0, the line moves from left to right and is PE for the left edge, PL for the right edge, and so on.

  4. The parameter t, the ratio of numerator and denominator, reduces to the distance to an edge divided by dx or dy, exactly the constant of proportionality we could calculate directly from the parametric line formulation.

Liang-Barsky algorithm - parameters

Calculation for parametric line clipping algorithm.

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

Liang-Barsky algorithm - pseudocode

```
Procedure Clip2D (var x0,y0,x1,y1: real; var visible: boolean);
{clip a line with endpoints (x0,y0) and (x1,y1), against the rectangle with corners (xmin,ymin) and
        (xmax,ymax). The flag visible is true if a clipped segment is returned in the var endpoint
        parameters. If the line is rejected, the endpoints are not changed and visible is set to false}
var    tE,tL,dx,dy: real;

    function ClipPoint(denom, num:real; var tE,tL: real): boolean;
    {computes a new value of tE or tL for an interior intersection of a line segment and edge.
        Parameter denom is −(Ni · D), which reduces to ±Δx, Δy for upright rectangle. Its sign
        determines whether the intersection is PE or PL. parameter num is Ni · D(P0-PEi) for a particular
        edge/line combination, which reduces to directed horizontal and vertical distances from P0 to
        an edge; If the line segment can be trivially rejected, false is returned; if it cannot be, true is
        returned and the value of tE or tL is adjusted, if needed, for the portion of the segment that is
        inside the edge}
        var    t: real; accept: boolean;          {accept will be returned as value of ClipPoint;
                                                    accept line until find otherwise}
```

Liang-Barsky algorithm - pseudocode

```
begin
    accept:=true;
    if denom>0 then                       {PE intersection}
       begin
          t:=num/denom;                   {value of t at the intersection }
          if t>tL then                    {tE and tL crossover}
             accept:=false;               {so prepare to reject line}
          else if t>tE then               {a new tE has been found }
             tE:=t
       end
    else if denom<0 then                  {PL intersection}
       begin
          t:=num/denom;                   {value of t at the intersection }
          if t<tE then                    {tE and tL crossover}
             accept:=false;               {so prepare to reject line}
          else if t<tL then               {a new tL has been found }
             tL:=t
       end
    else
       if num>0 then                      {line on outside of edge }
          accept:=false;
    ClipPoint:=accept
end; {ClipPoint }
```

Liang-Barsky algorithm - pseudocode

```
Begin
   dx:=x1-x0; dy:=y1-y0;
   visible ;= false;                    { output is generated only if line is inside all four edges}
   { first test for degenerate line and clip the point; ClipPoint returns true if the point lies inside the
         clip rectangle.}
    if (dx=0) and (dy=0) and ClipPoint(x0,y0) then
        visible:=true
    else
        begin
            tE = 0;
            tL = 1;
            if ClipPoint(dx, xmin-x0, tE, tL) then                {inside against left edge}
                if ClipPoint(-dx, x0-xmax, tE, tL) then           {inside against right edge}
                    if ClipPoint(dy, ymin-y0, tE, tL) then        {inside against bottom edge}
                        if ClipPoint(-dy, y0-ymax, tE, tL) then   {inside against top edge}
                            begin
                                visible:=true;
                                if tL<1 then
                                    begin                 {compute PL intersection, if tL has moved}
                                        x1:=x0+tL*dx;
                                        y1:=y0+tL*dy
                                    end;
                                if tE>0 then
                                    begin                 {compute PE intersection, if tE has moved}
                                        x0:=x0+tE*dx;
                                        y0:=y0+tE*dy
                                    end;
                            end
        end
End; {Clip2D}
```

Conclusions

- Cohen-Sutherland algorithm is efficient when:
  - Outcode testing can be done cheaply, e.g. by doing bitwise operations in assembly language
  - Trivial acceptance or rejection is applicable to the majority of line segments
- Parametric line clipping is efficient when:
  - Many line segments need to be clipped, since the actual calculations of the coordinates of the intersection points is postponed until needed, and testing can be done on parameter values.
- Liang-Barsky algorithm is more efficient than Cyrus-Beck
  - Trivial rejection testing can avoid calculation of all four parameter values for lines that do not intersect the clip rectangle.

Polygon Clipping

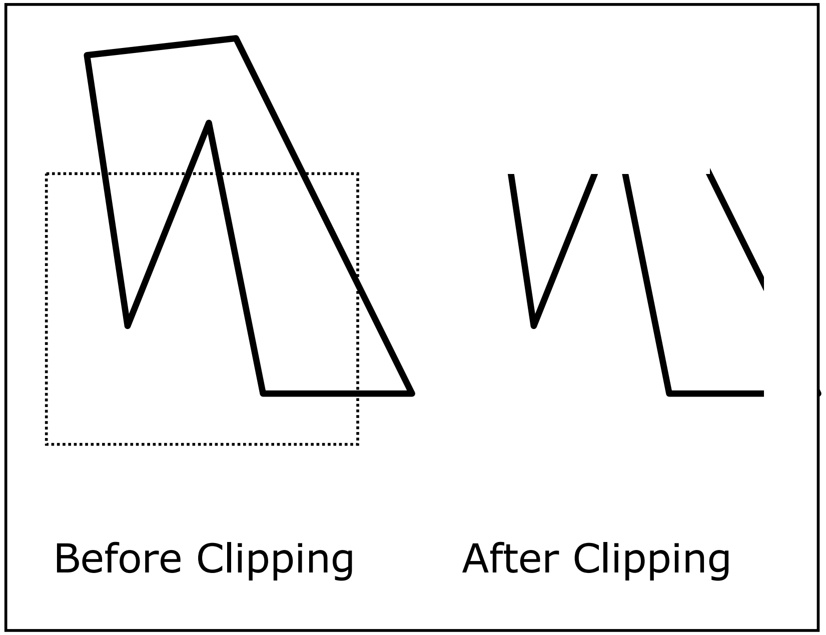
- Polygon definition:
  - set of points → point clipping
  - set of lines → line clipping
  - sequence of lines →
    Sutherland – Hodgman algorithm
    Weiler-Atherton algorithm

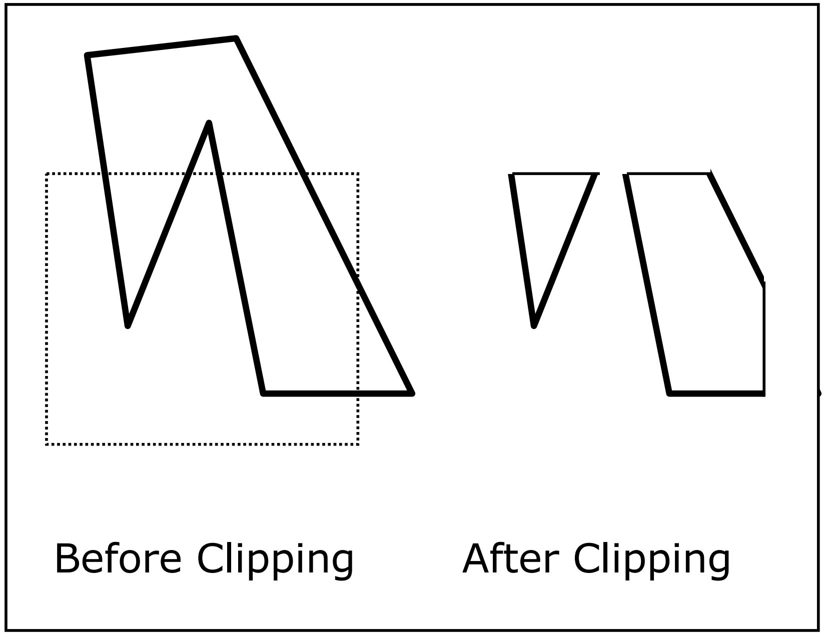
- General definition:
  - Sequence of vertices with implicit relationships
  - By clipping: 1 polygon (v1, v2, …., vn) →
    p polygons (A11, A12, …, A1k; …Ap1, Ap2, …,Apq)

- Display:
  - set of points
  - set of lines (wire-frame)
  - closed multicolor polyline
  - set of polygons
  - filled polygons

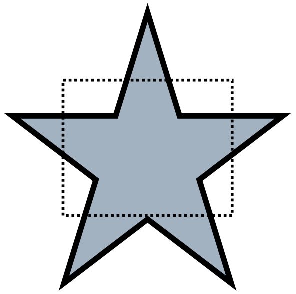
Sutherland Hodgman algorithm

- Reentrant polygon clipping
- Published in 1974 by Sutherland, I.E., Hodgman, G.W., "Reentrant Polygon Clipping", CACM, vol 17(1).

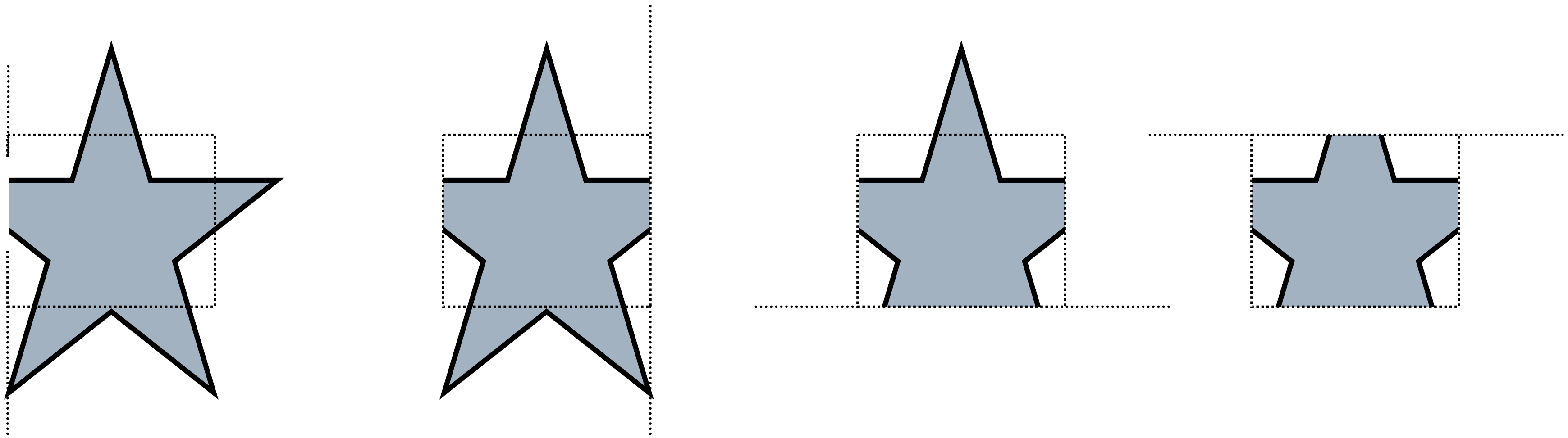




Sutherland Hodgman algorithm

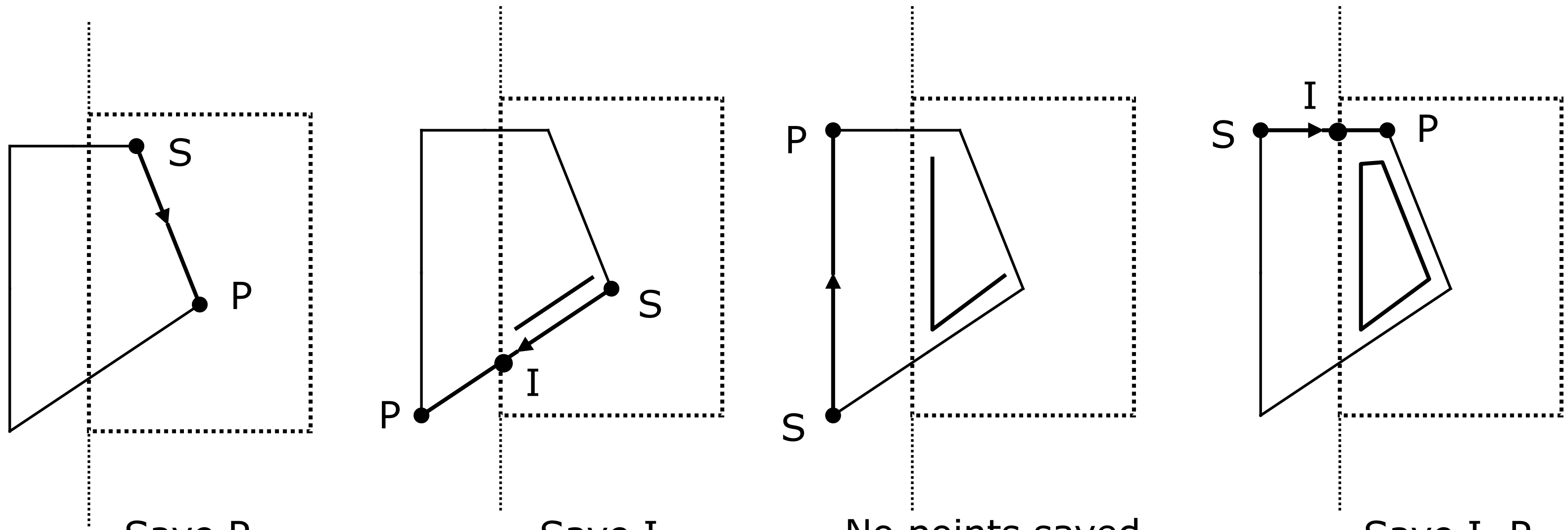


Original
Polygon



Clip Left | Clip Right | Clip Bottom | Clip Top

Window – polygon edge relationships



Save P
(a)

Save I
(b)

No points saved
(c)

Save I, P
(d)

Sutherland Hodgman algorithm

Input vertex list: inv[] = {v1, v2,..., vn}
Output vertex list: outv[ ] = {v1', v2',..., vp'}

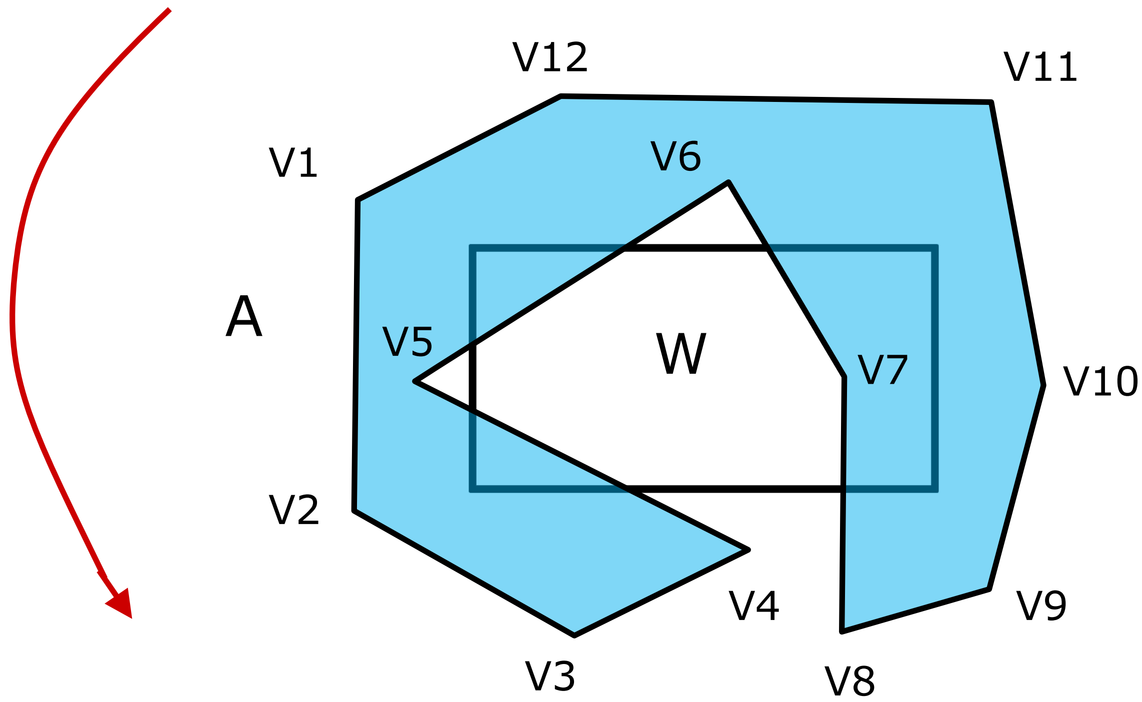
```
algorithm  ClippingSH(    vertex_list      inv, outv;
                          window_edge      clipping_edges[4])
{
    for( j = 0; j < 4; j++){
            EdgeClippingSH( inv, outv, clipping_edges[j]);
            inv = outv;          /* update the current edge list */
    }
}
```

Sutherland Hodgman algorithm

```
algoritm EdgeClippingSH(  vertex_list  inv, outv;
                             window_edge    clipping_edge)
{
    vertex   i,p,s;
    s = the last vertex in inv;
    for( each vertex p from  inv) {
    if( ToWindow(p, clipping_edge))              /* case a and d */
        if( ToWindow(s, clipping_edge))          /* case a */
            write vertex p into outv;
        else{
            i = Intersection(s, p, clipping_edge);
            write intersection i into outv;
            write vertex p into outv;
        }
    else if( ToWindow(s, clipping_edge)){        /* case b */
        i = Intersection(s, p, clipping_edge);
        write intersection i into outv;
    }                                            /* for case c don't write anything */
    s = p;                                       /* update the start vertex */
    }
}   /* end of the EdgeClippingSH algorithm*/
```

Sutherland Hodgman algorithm

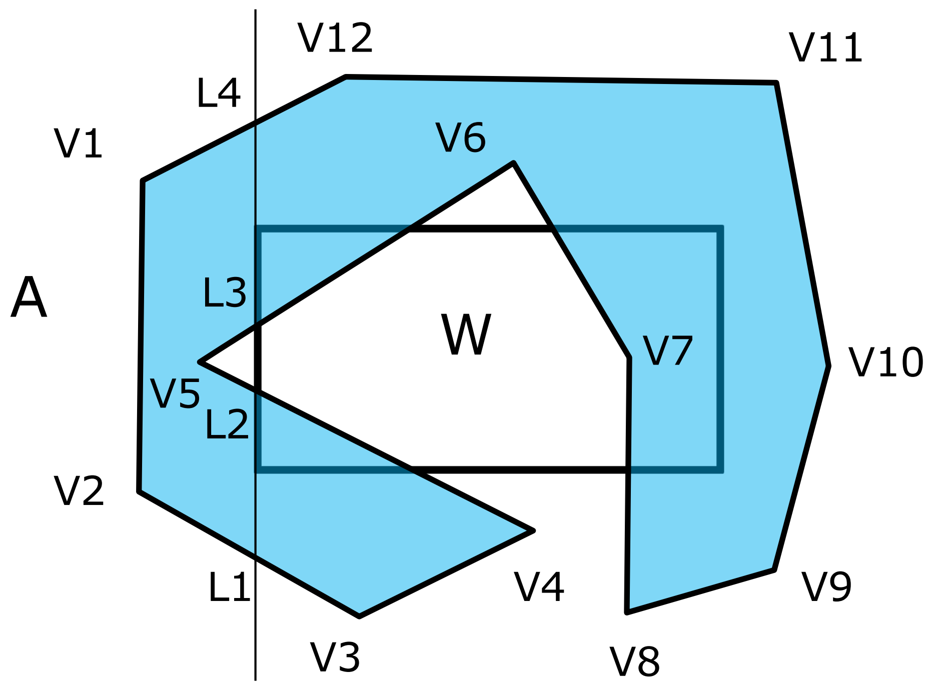
Step 1. Let us consider the polygon A defined with vertices V1, V2, …, V12 in counterclockwise direction, and the rectangular window W.



Sutherland Hodgman algorithm

Step 2. Clip the polygon A against the left edge of the window W. We analyze each edge of the polygon and classify its relationship against the window edge.

Input polygon: A; Output polygon: OPL (initially no any vertex)



Start from V1

V1V2 – case c: none saved

V2V3 – case d: save L1, V3

V3V4 – case a: save V4

V4V5 – case b: save L2

V5V6 – case d: save L3, V6

Any other edge, till V12 is in case a: save V7, …, V12

V12V1 – case b: save L4

Finally the output polygon:

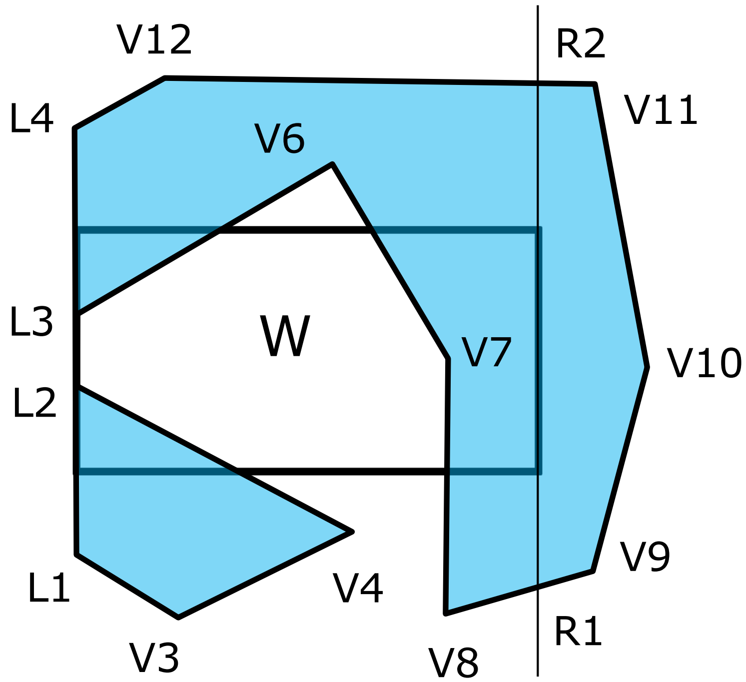
OPL is L1, V3, V4, L2, L3, …, V12, L4

Sutherland Hodgman algorithm

Step 3. Clip the polygon A against the right edge of the window W. We analyze again each edge of the polygon and classify its relationship against the window edge.

Input polygon: OPL: L1, V3, V4, L2, L3, …, V12, L4;

Output polygon: OPR (initially no any vertex)



Start from L1

All edges, till V8 are in case a: save V3, …,L2, L3,…, V8

V8V9 – case b: save R1

V9V10, V10V11 – case c: none

V11V12 – case d: save R2, V12

Any other edge, till L1 is in case a: save L4, L3, L2, L1

Finally the output polygon:

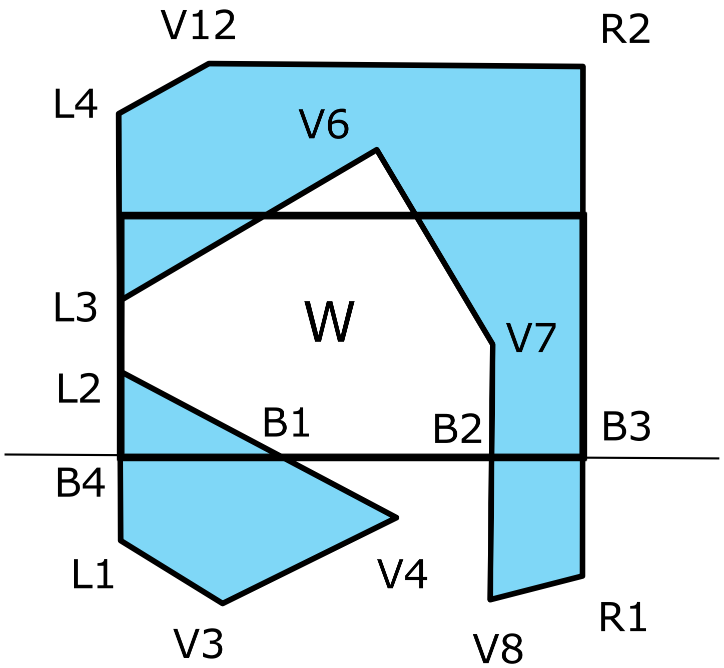
OPR is L1, V3, V4, L2, L3, …, V8, R1, R2, V12, L4

Sutherland Hodgman algorithm

Step 4. Clip the polygon A against the bottom edge of the window W. We analyze again each edge of the polygon and classify its relationship against the window edge.

Input polygon: OPR: L1, V3, V4, L2, L3, …, V8, R1, R2, V12, L4;

Output polygon: OPB (initially no any vertex)



Start from L1

L1V3, V3V4, V8R1 – case c: none

V4L2 – case d: save B1, L2

V7V8 – case b: save B2

R1R2 – case d: save B3, R2

L4L1 – case b: save B4

All edges, till V8 are in case a: save L3, …, V12

The output polygon:

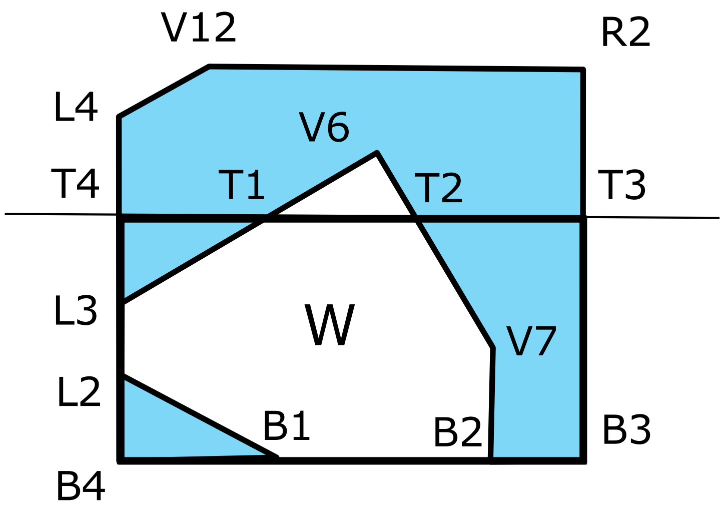
OPB is B1, L2, L3, V6, V7, B2, B3, R2, V12, L4, B4

Sutherland Hodgman algorithm

Step 5. Clip the polygon A against the top edge of the window W. We analyze again each edge of the polygon and classify its relationship against the window edge.

Input polygon: OPB: B1, L2, L3, V6, V7, B2, B3, R2, V12, L4, B4;

Output polygon: OPT (initially no any vertex)



Start from B1

The new vertices T1, T2, T3 and T4 are introduced into the OPB list.

L3V6, B3R2 – case b: save T1, T3

V6V7, L4B4 – case d: save T2, V7 and T4, B4

The output polygon:

OPT is B1, L2, L3, T1, T2, V7, B2, B3, T3, T4, B4

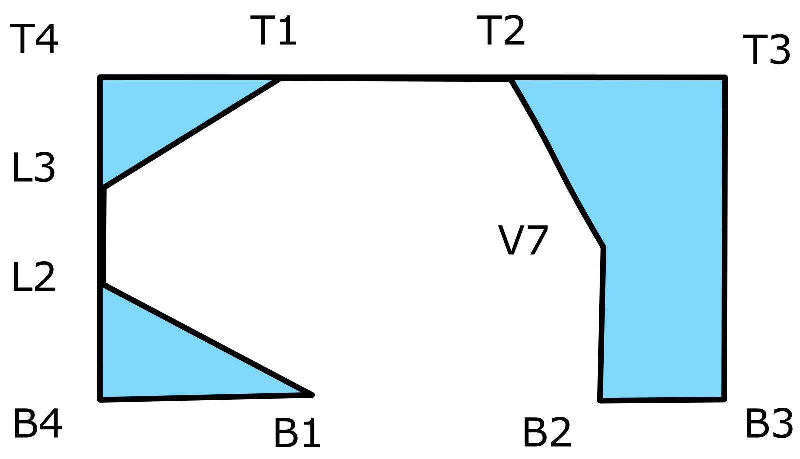
Sutherland Hodgman algorithm

Step 6. The final polygon clipped against the window W is

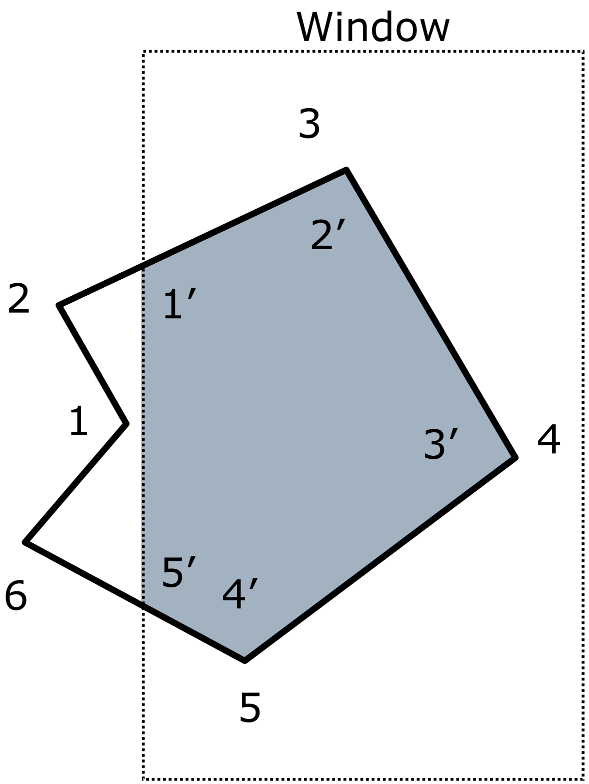
A’: B1, L2, L3, T1, T2, V7, B2, B3, T3, T4, B4

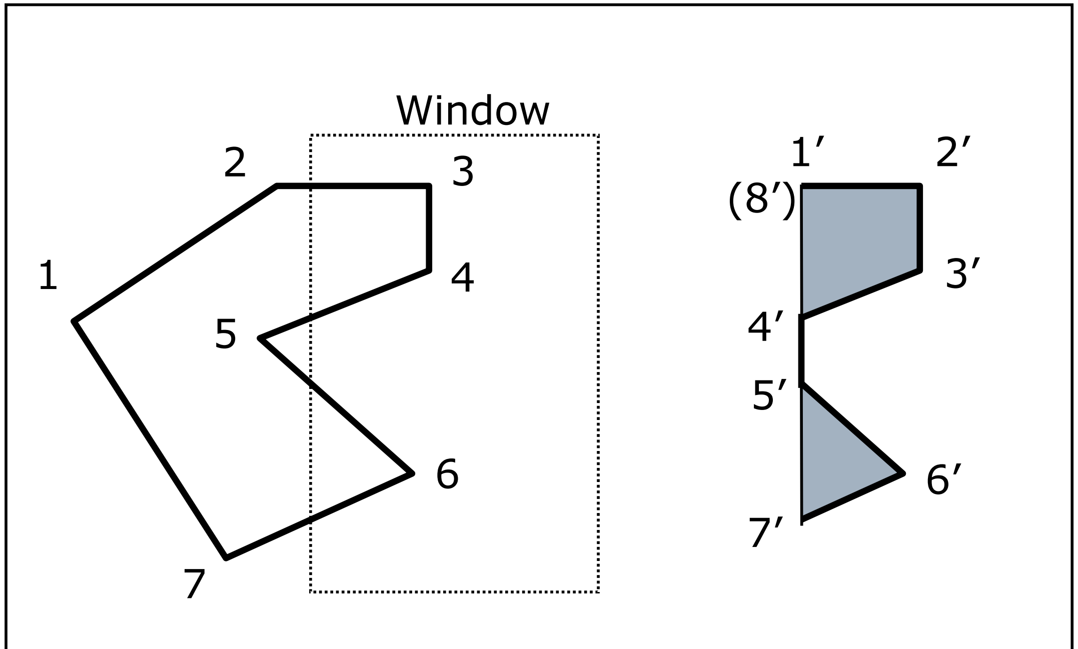
Observation:

- There is just one polygon, consisting of three connected polygons
- There are false edges
- The clipped polygon includes the vertices of the window W as new vertices within the clipped polygon



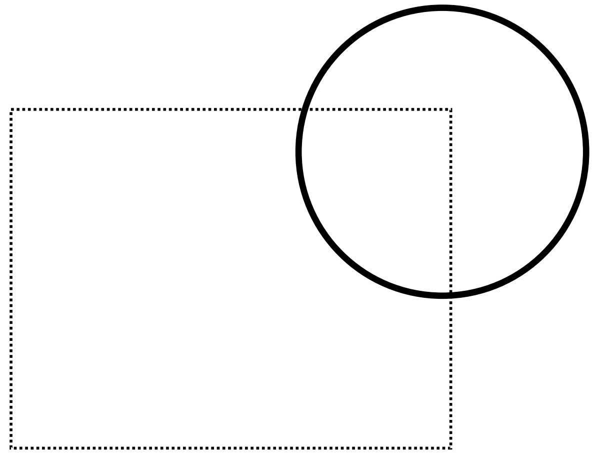
SH clipping - Examples



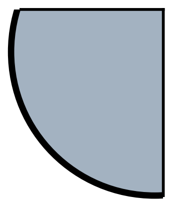


(a) (b)

SH clipping - Examples



Before Clipping

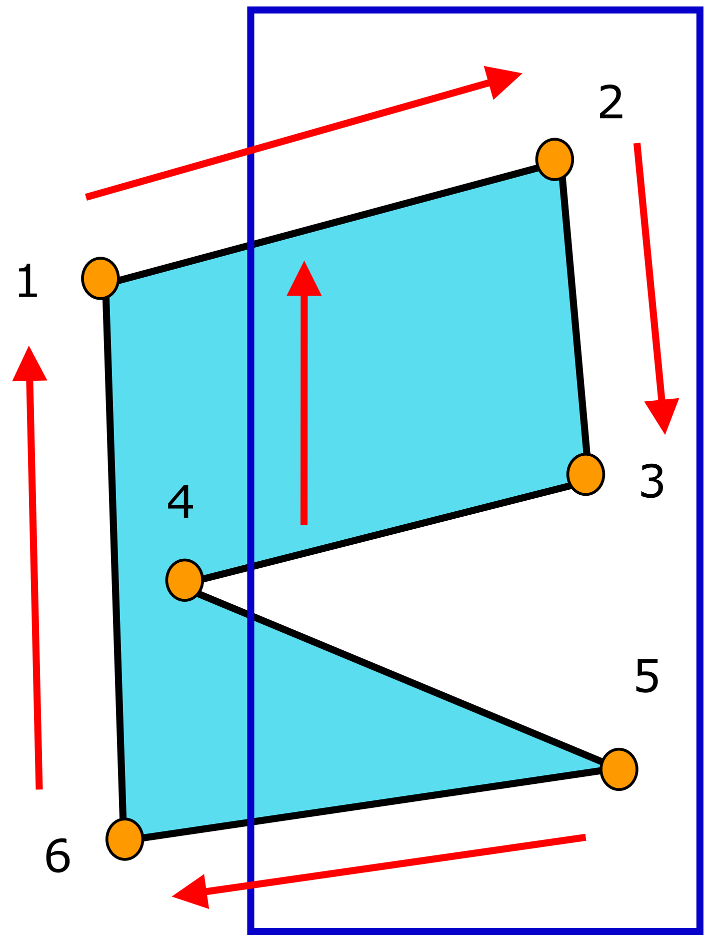


After Clipping

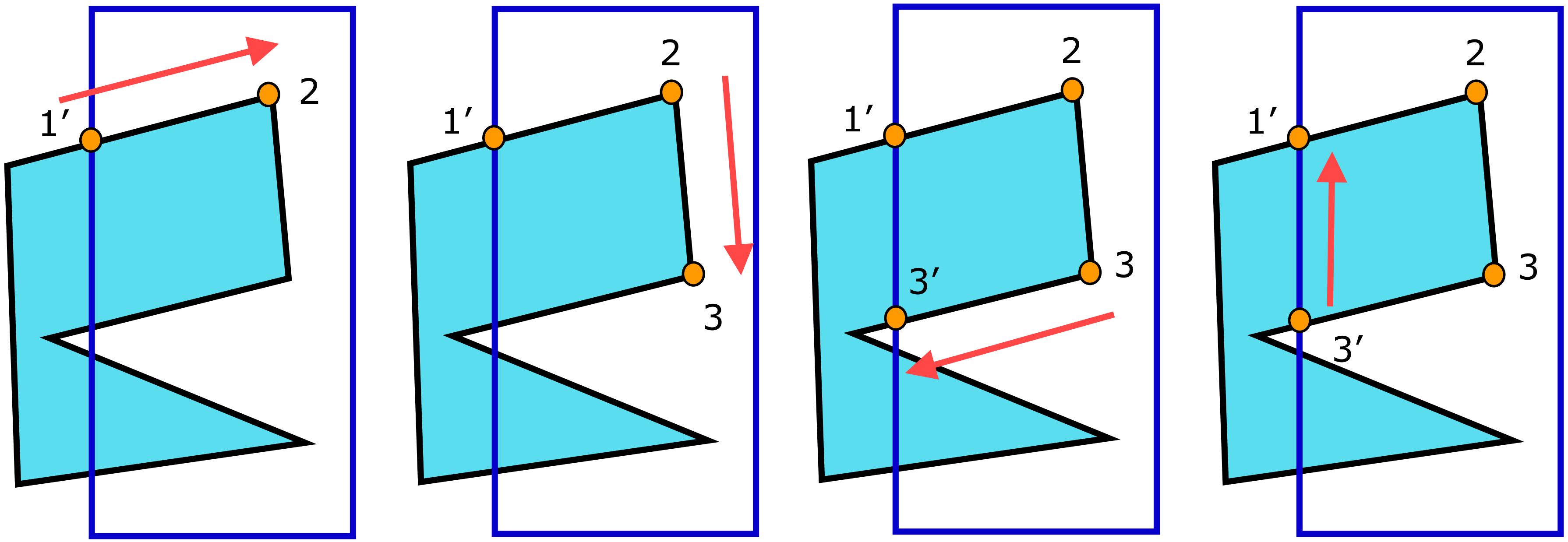
Weiler-Atherton clipping algorithm

- The Weiler-Atherton algorithm produces separate polygons for each visible fragment

- 4 cases:
    a) out - > in
    b) in -> in
    c) in -> out
    d) out - > out
    e) follow clip edge



Weiler-Atherton clipping algorithm



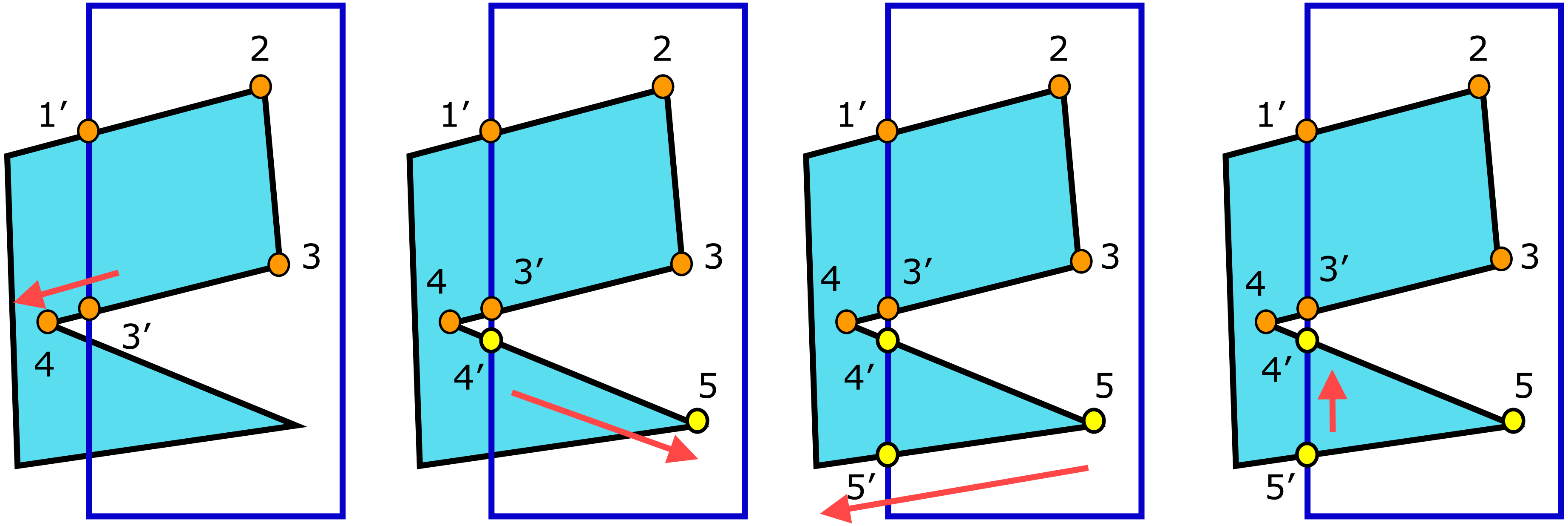
Case: Out -> In
Add clip vertex (1′)
Add end vertex (2)

Case: In -> In
Add end vertex (3)

Case: In -> Out
Add clip vertex (3′)
Cache old direction

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

Weiler-Atherton clipping algorithm



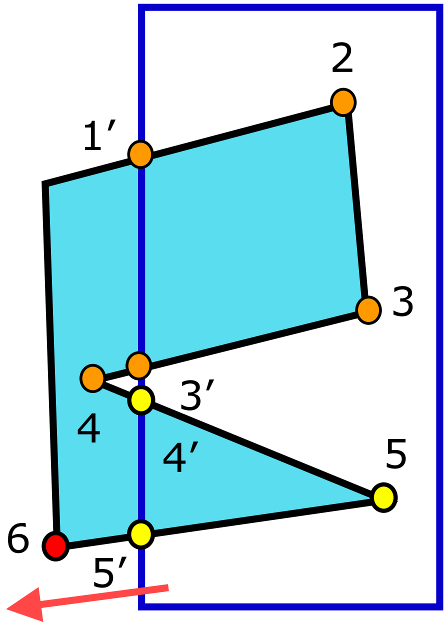
Continue from cached vertex and direction

Case: Out -> In
Add clip vertex (4′)
Add end vertex (5)

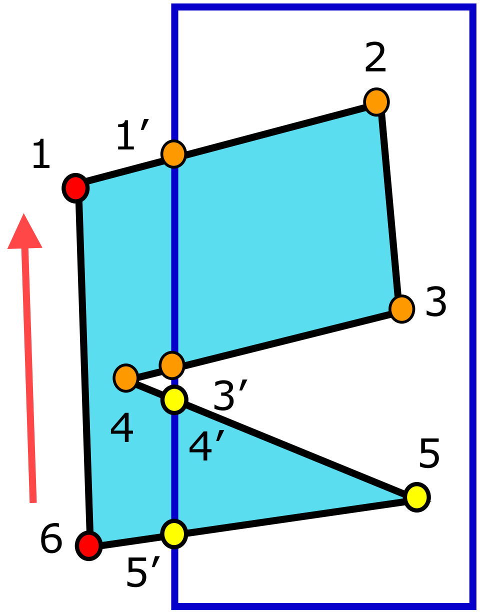
Case: In -> Out
Add clip vertex (5′)
Cache old direction

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

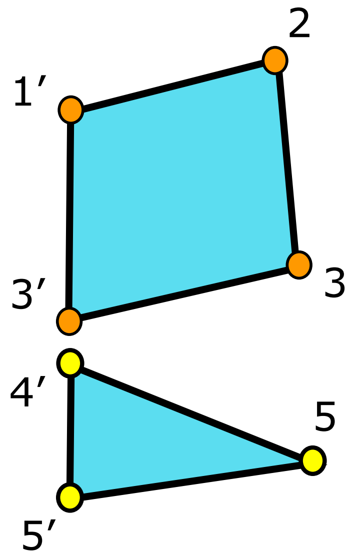
Weiler-Atherton clipping algorithm



Continue from cached vertex and direction



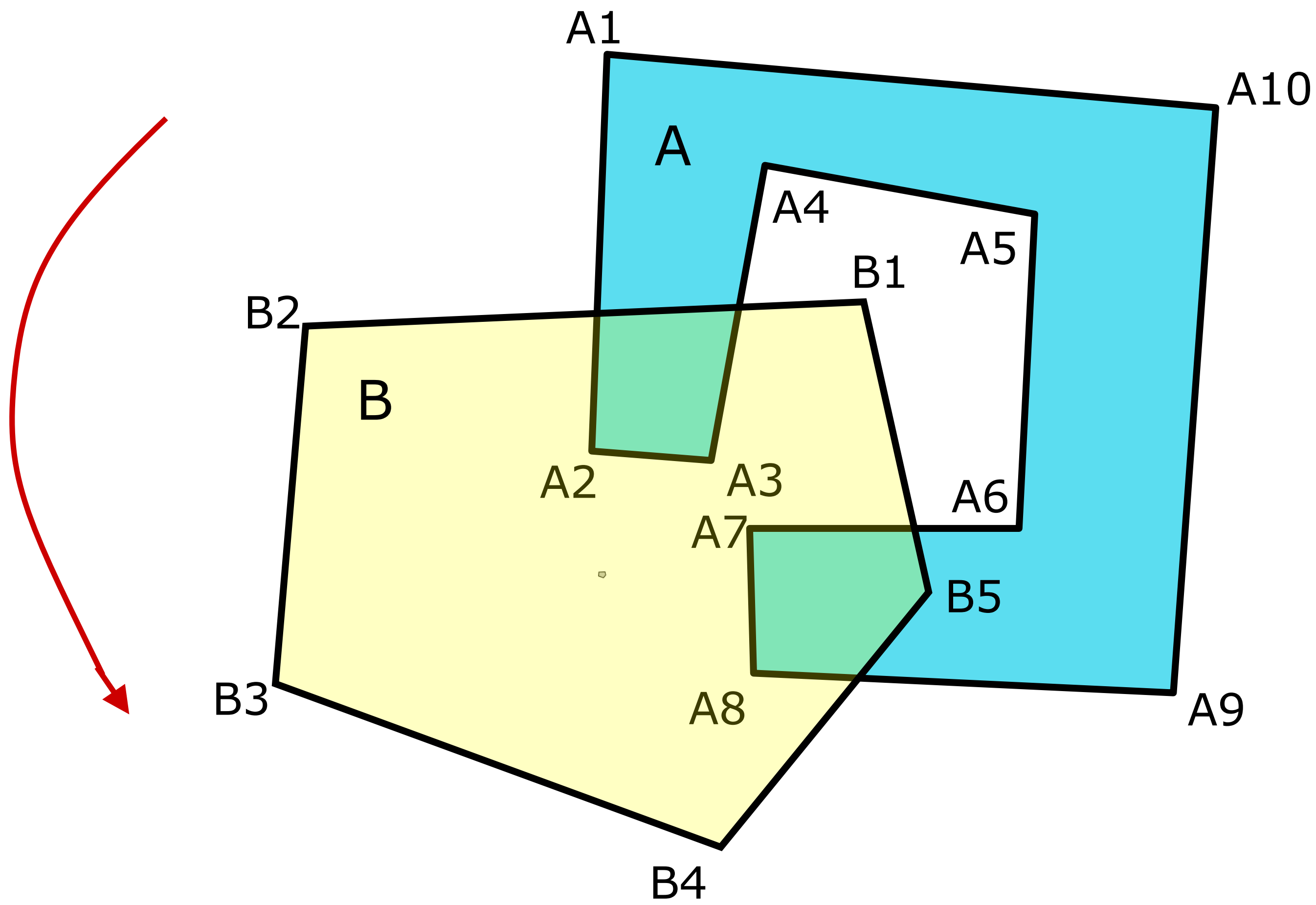
Nothing added. Finished



Final Result: 2 unconnected polygons

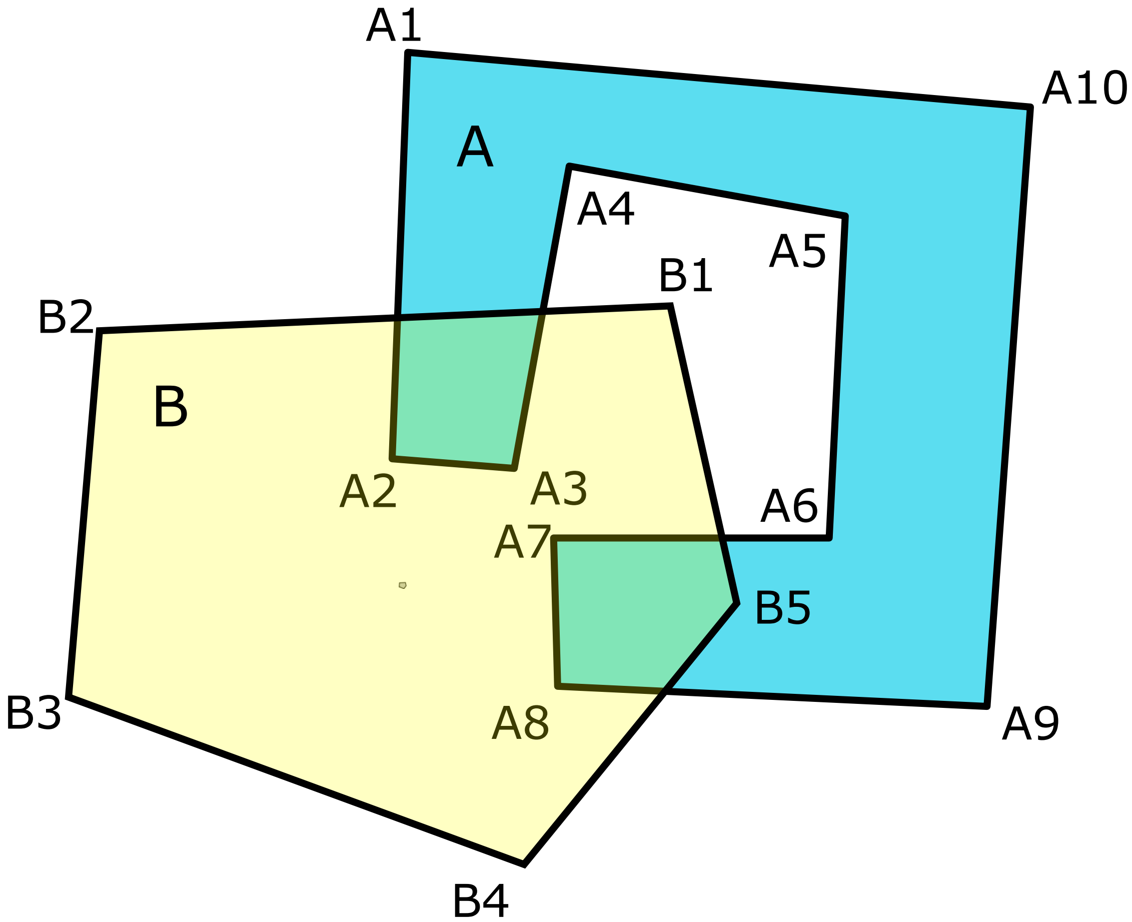
Weiler-Atherton alg. implementation

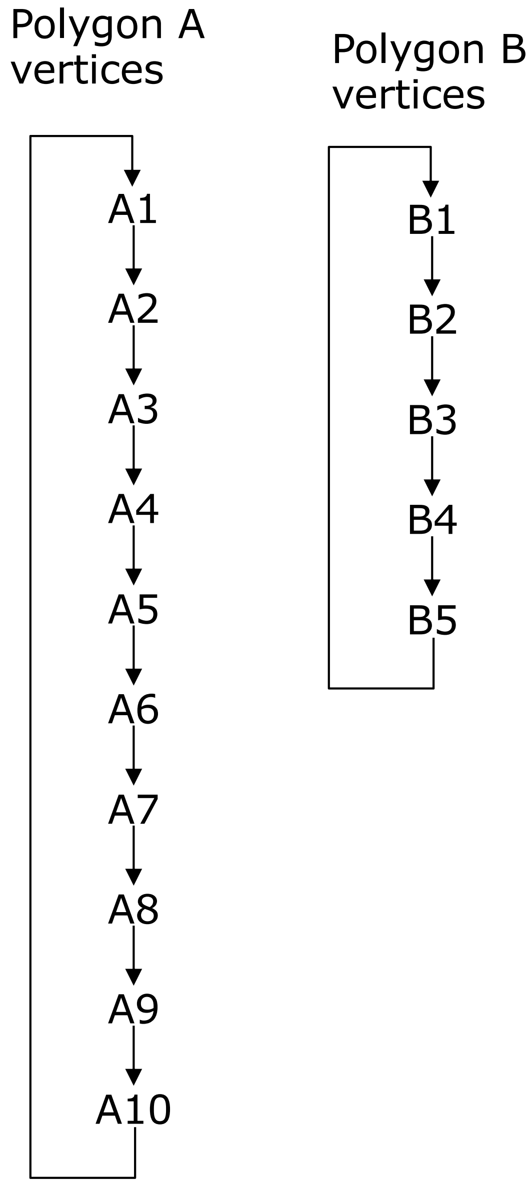
Step 1. Let be two polygons A and B, with vertices A1, A2, …, A10 and B1, …, B5, given in counterclockwise direction. Let us consider the polygon A to be clipped by polygon B (window)



Weiler-Atherton alg. implementation

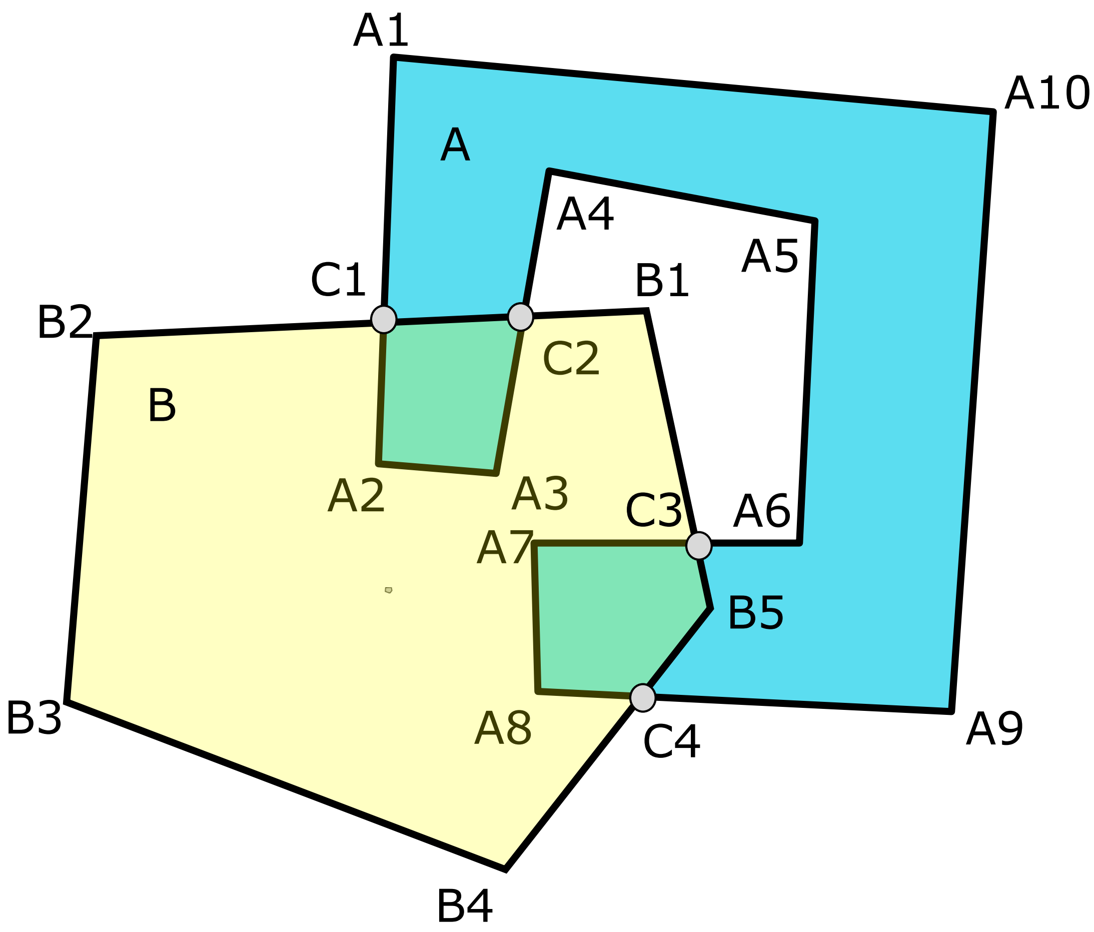
Step 2. Build up the circular lists of vertices for both polygons in counterclockwise direction.

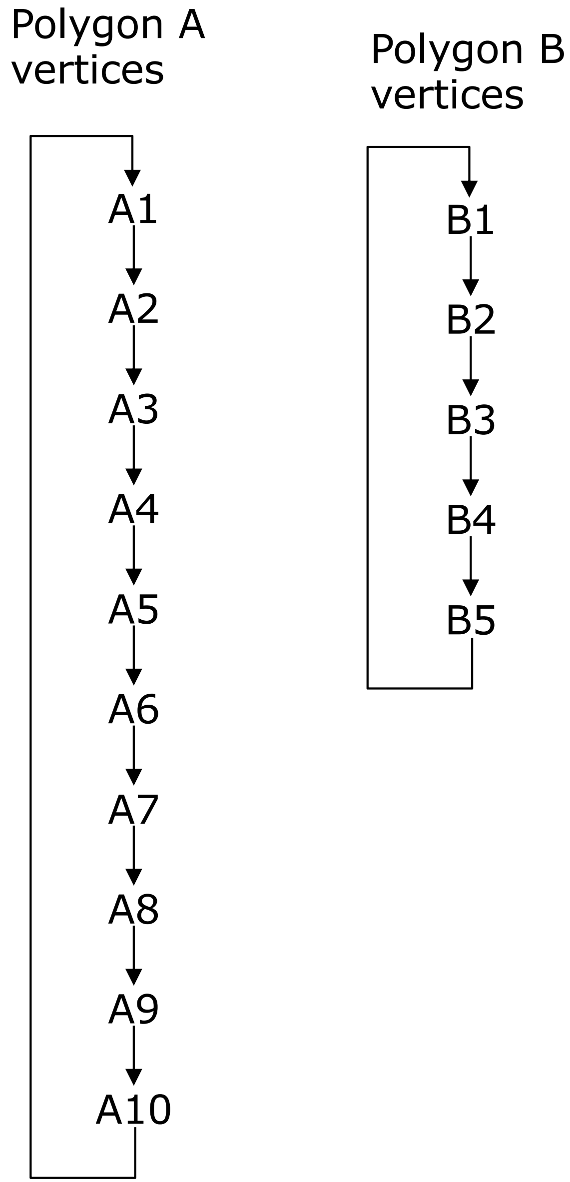




Weiler-Atherton alg. implementation

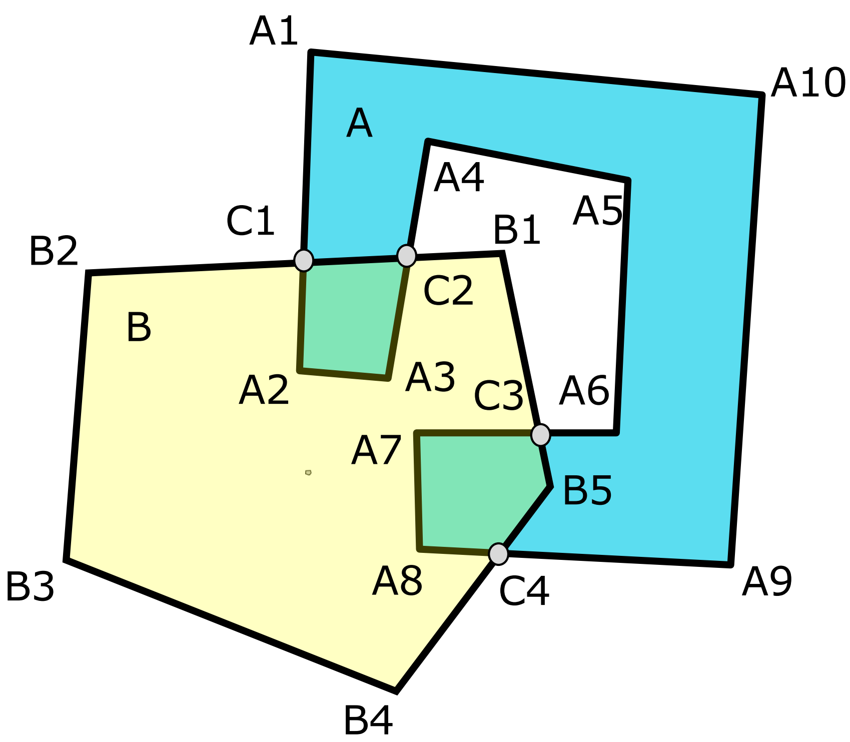
Step 3. Compute the intersection points between polygon A and the window B, along A in counterclockwise direction: C1, C2, C3 and C4.



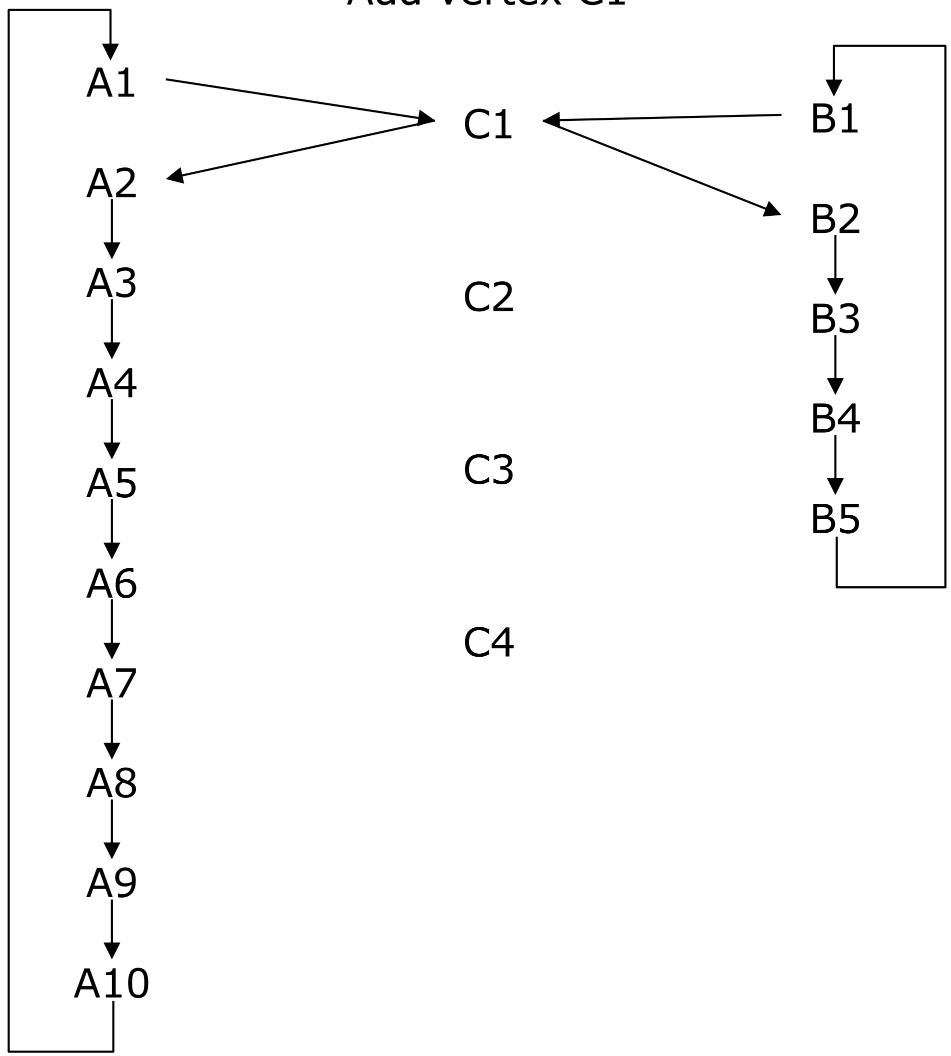


Weiler-Atherton alg. implementation

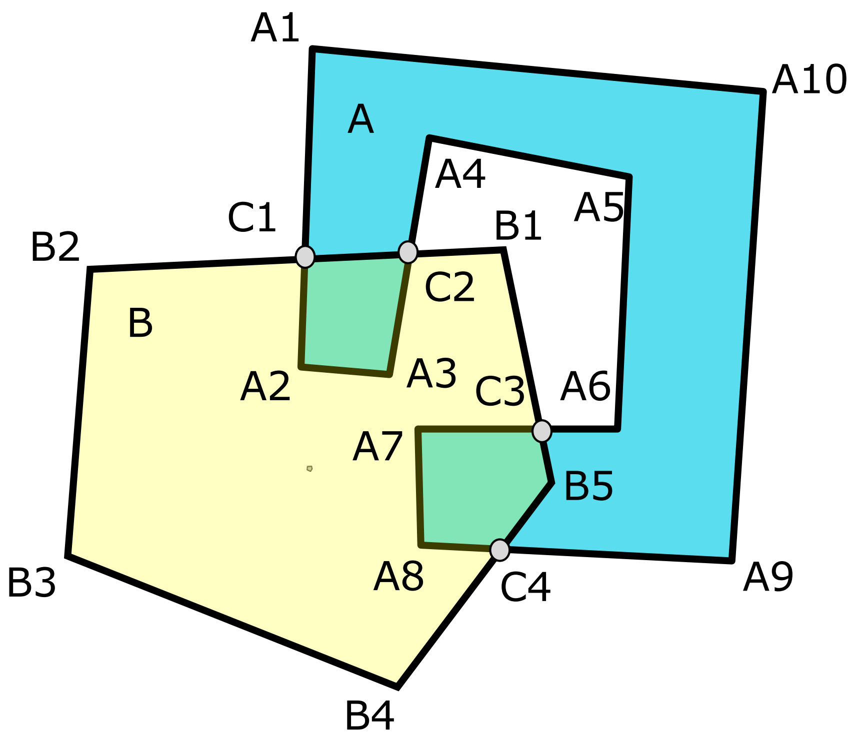
Step 4. Insert the intersection points C1, C2, C3 and C4 into the lists A and B, considering the conventional direction of each polygon.

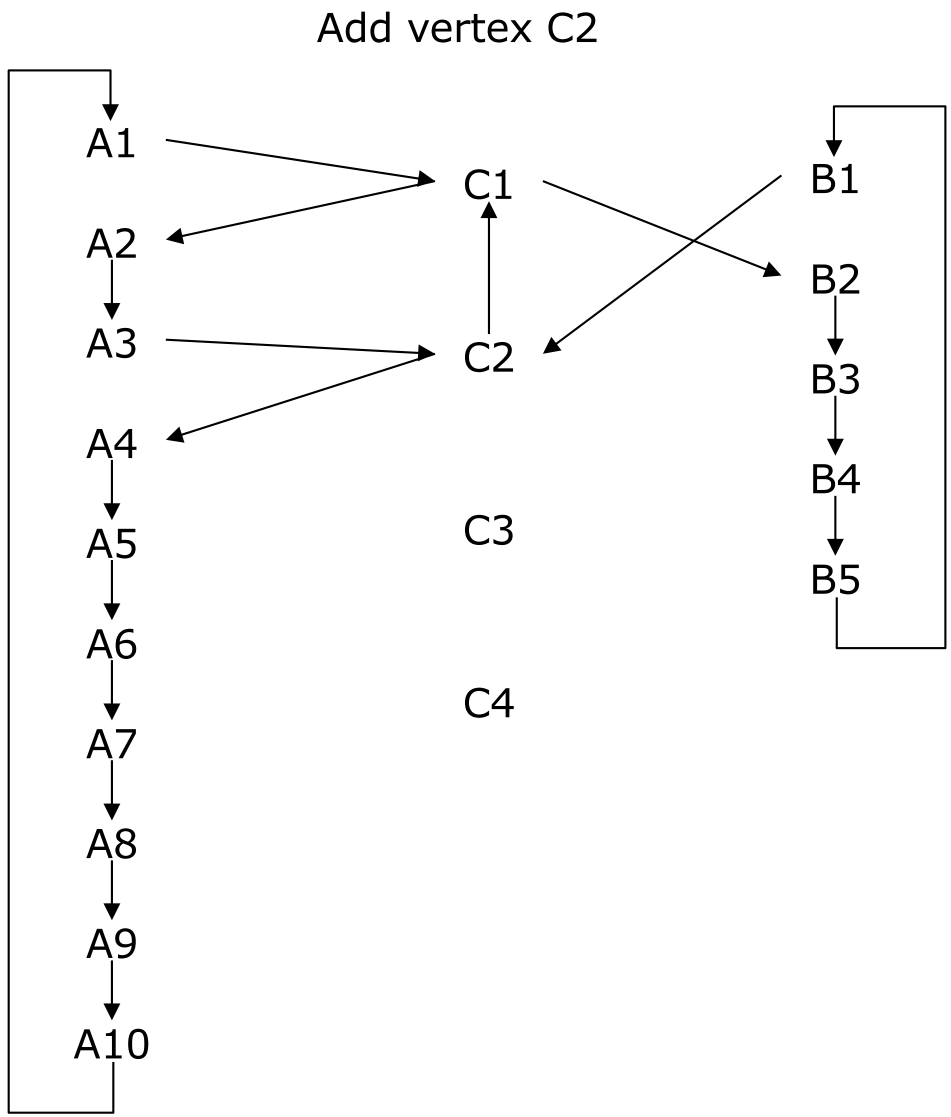


Add vertex C1

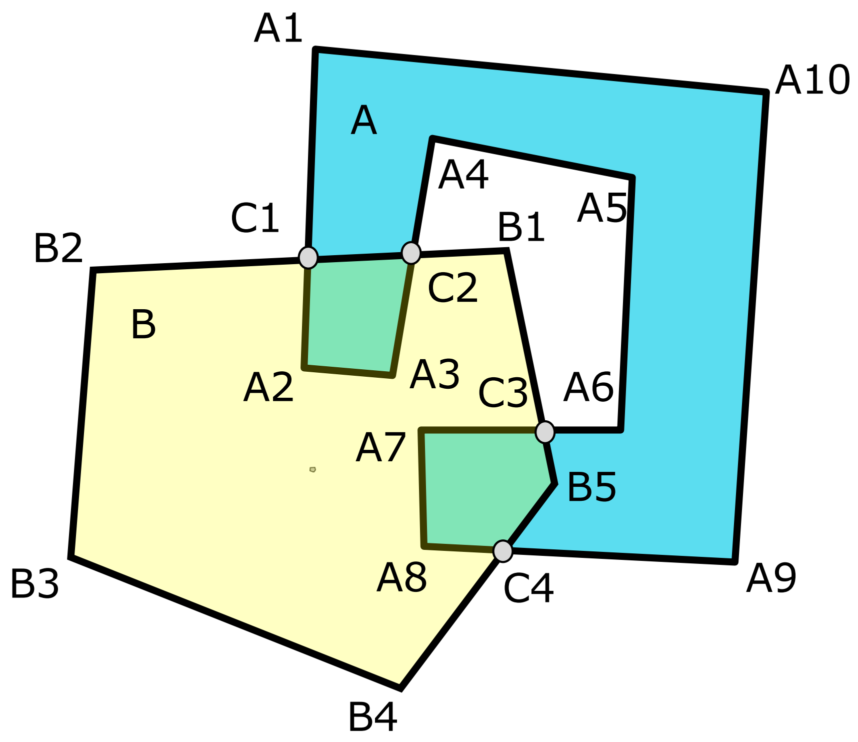


Weiler-Atherton alg. implementation

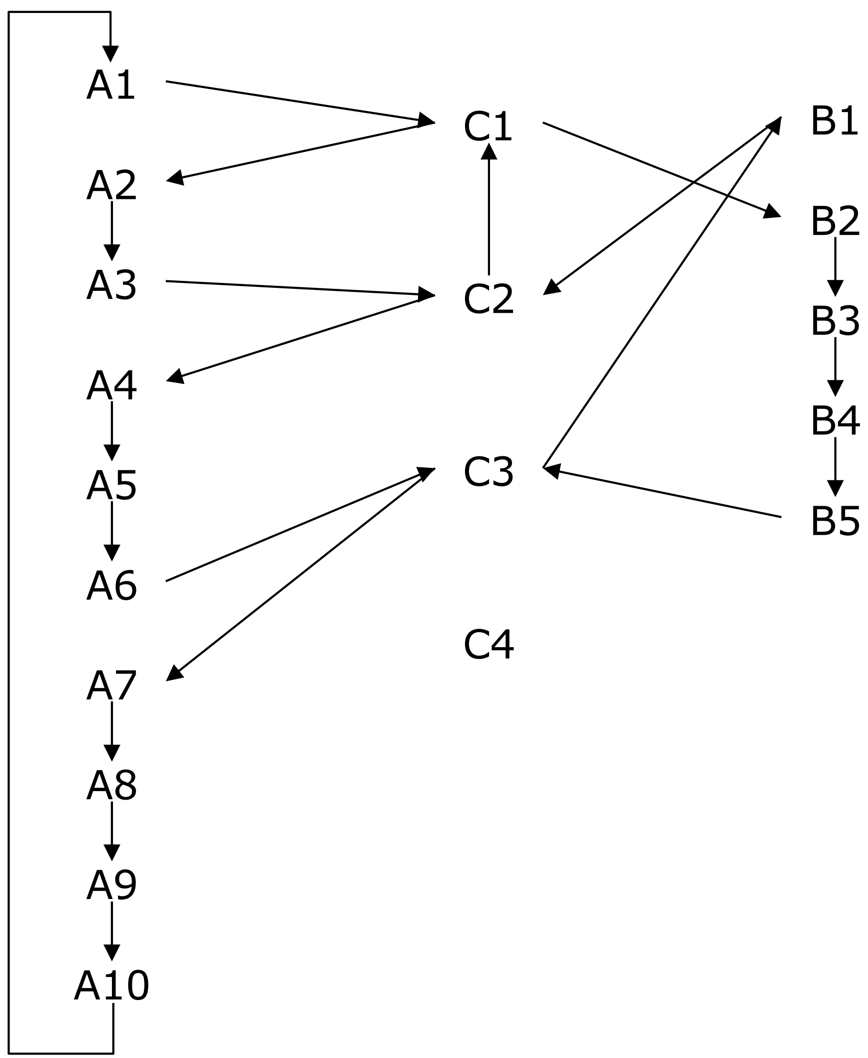




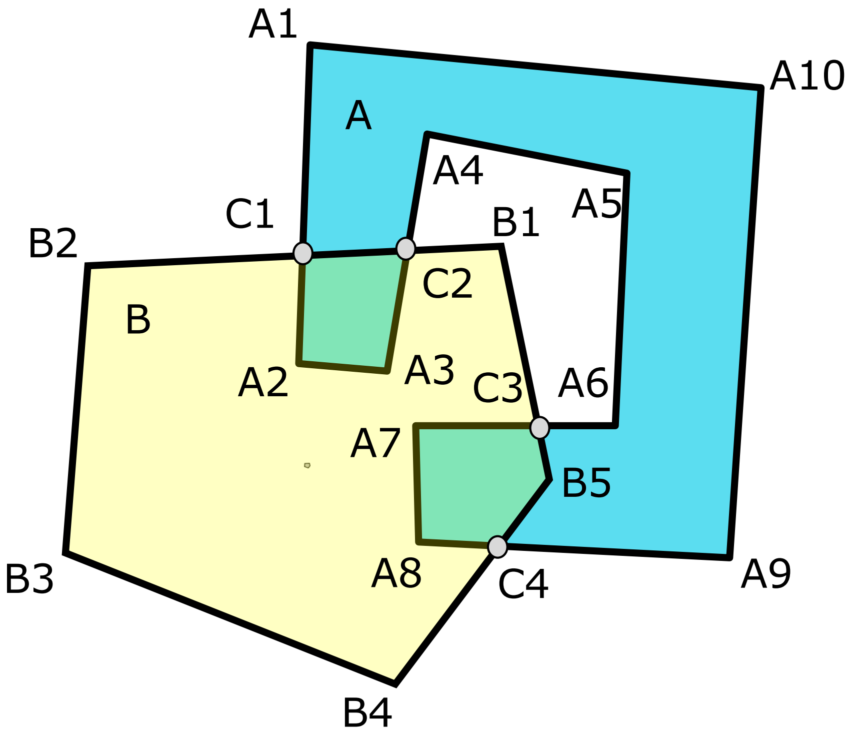
Weiler-Atherton alg. implementation



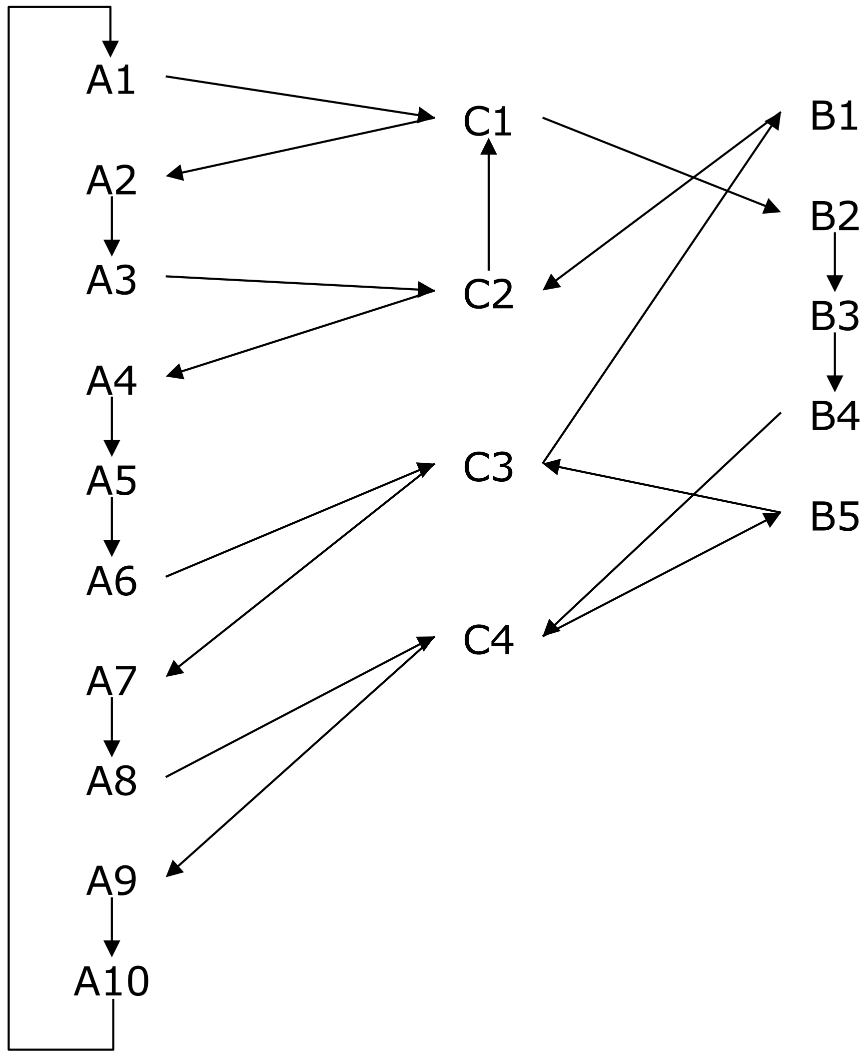
Add vertex C3



Weiler-Atherton alg. implementation



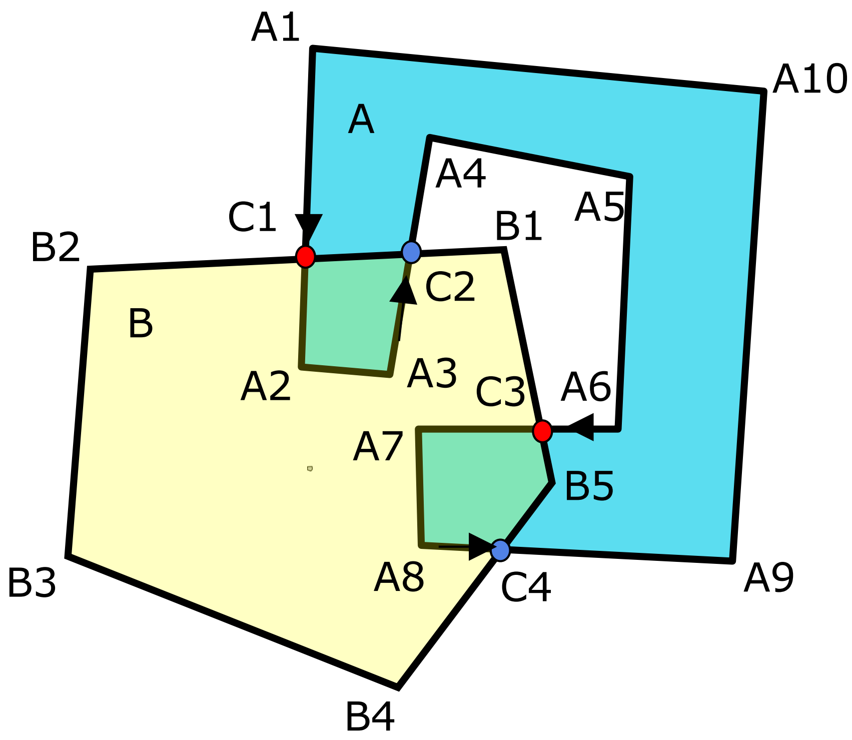
Add vertex C4

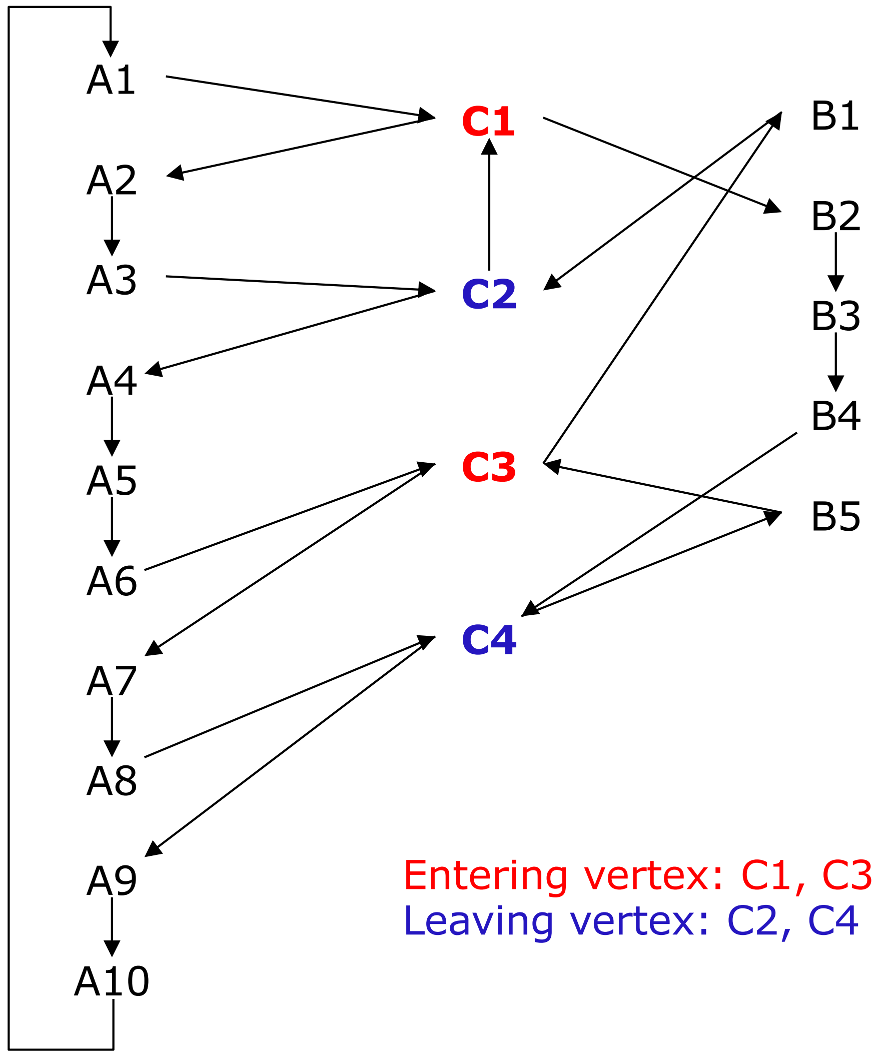


Weiler-Atherton alg. implementation

Step 5. Identify the entering and leaving intersection points into the window polygon (B).

Entering: C1, C3; Leaving: C2, C4



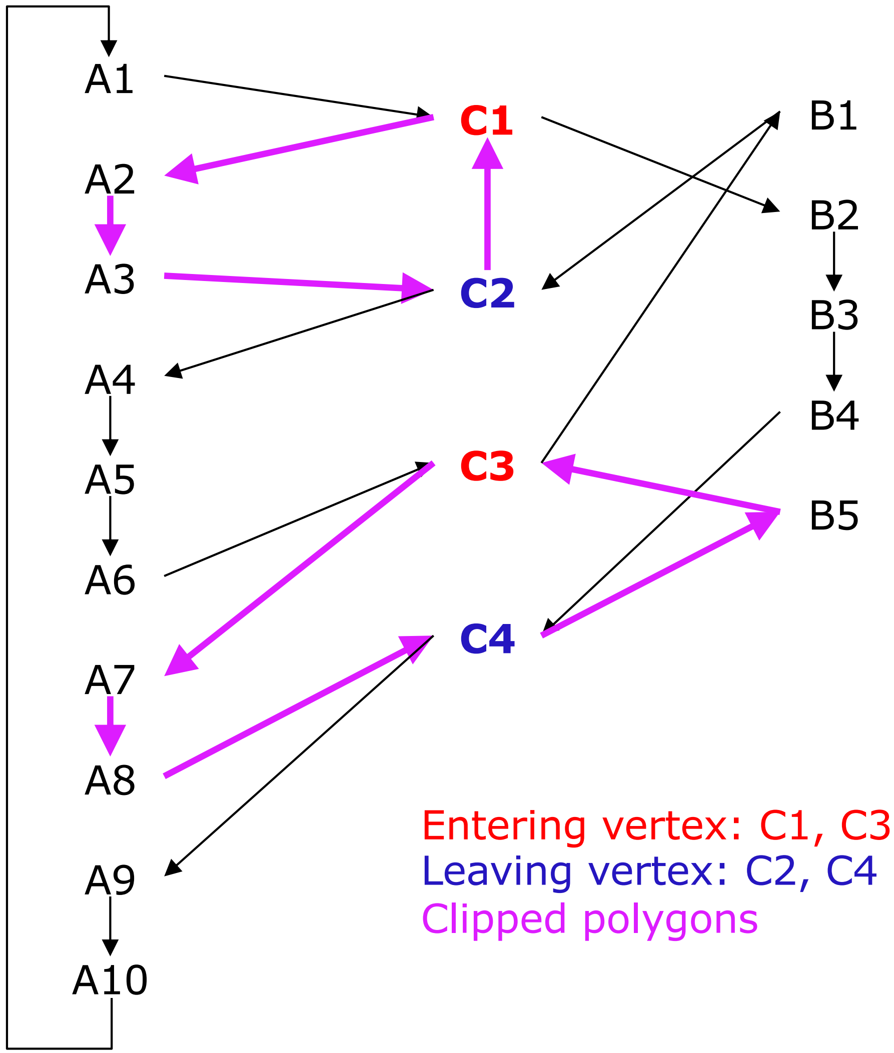


Weiler-Atherton alg. implementation

Step 6. Identify the clipped polygons.

Rules for capturing the clipped polygons:

1. Start at an entering intersection point (e.g. C1) and follow the connecting arrow on polygon A
2. If you encounter an entering intersection point (e.g. C1) swap to left hand loop (i.e. polygon A, e.g. toward A2)
3. If you encounter a leaving intersection point (e.g. C2) swap to right hand loop (i.e. window polygon B, e.g. toward C1 on B)
4. A loop is finished when you arrive back at start (e.g. back on C1)
5. Repeat 1-4 for all entering points



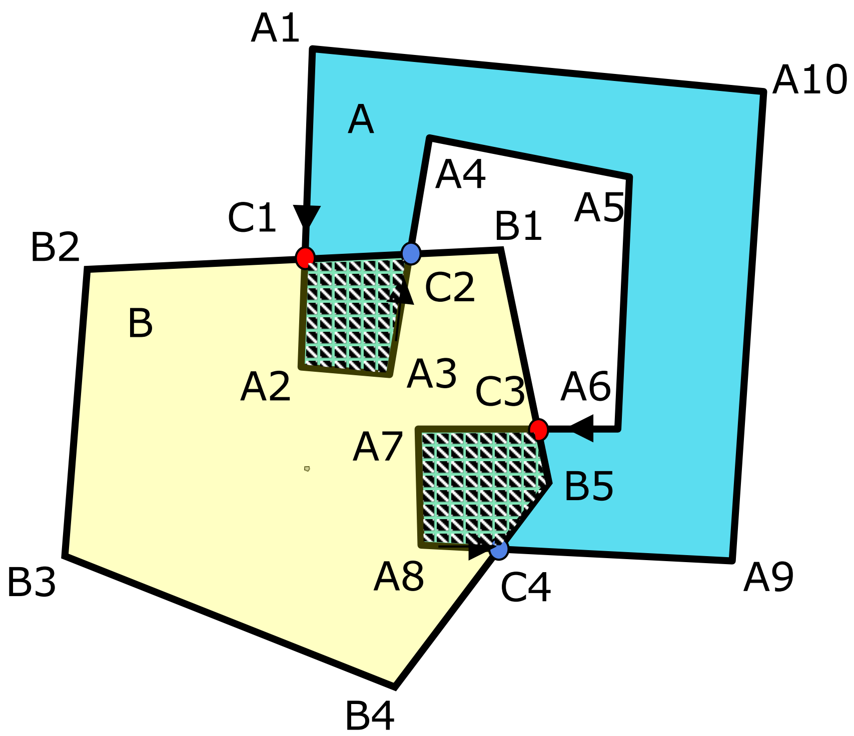
Weiler-Atherton alg. implementation

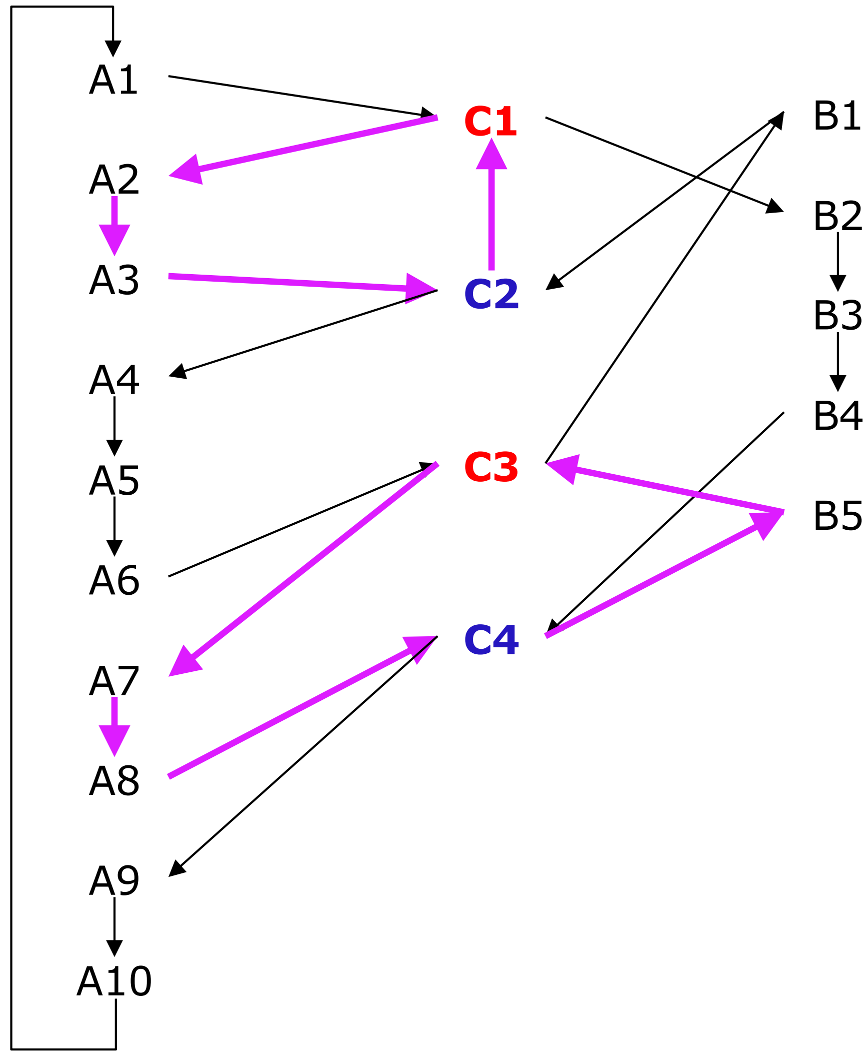
Step 7. Resulted clipped polygons are:

P1: C1, A2, A3, C2

P2: C3, A7, A8, C4, B5

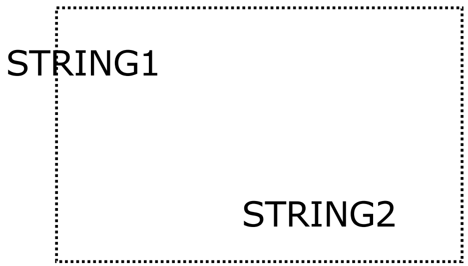
Obs. The resulted polygons are given in counterclockwise direction as the initial polygons.





Text Clipping

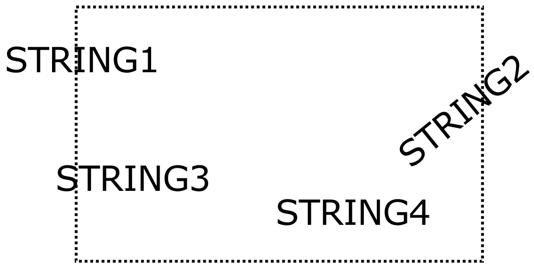
All or none
text clipping



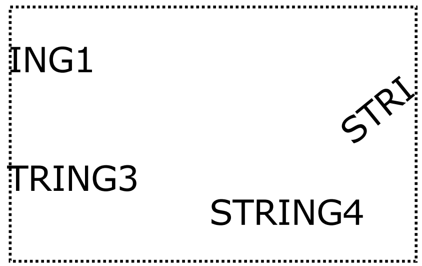
Before Clipping

After Clipping

All or none
character clipping

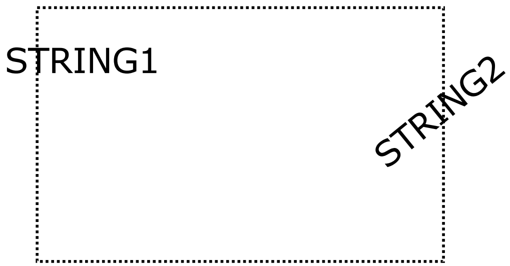


Before Clipping



After Clipping

Clipping individual
character



Before Clipping

After Clipping

Questions and proposed problems

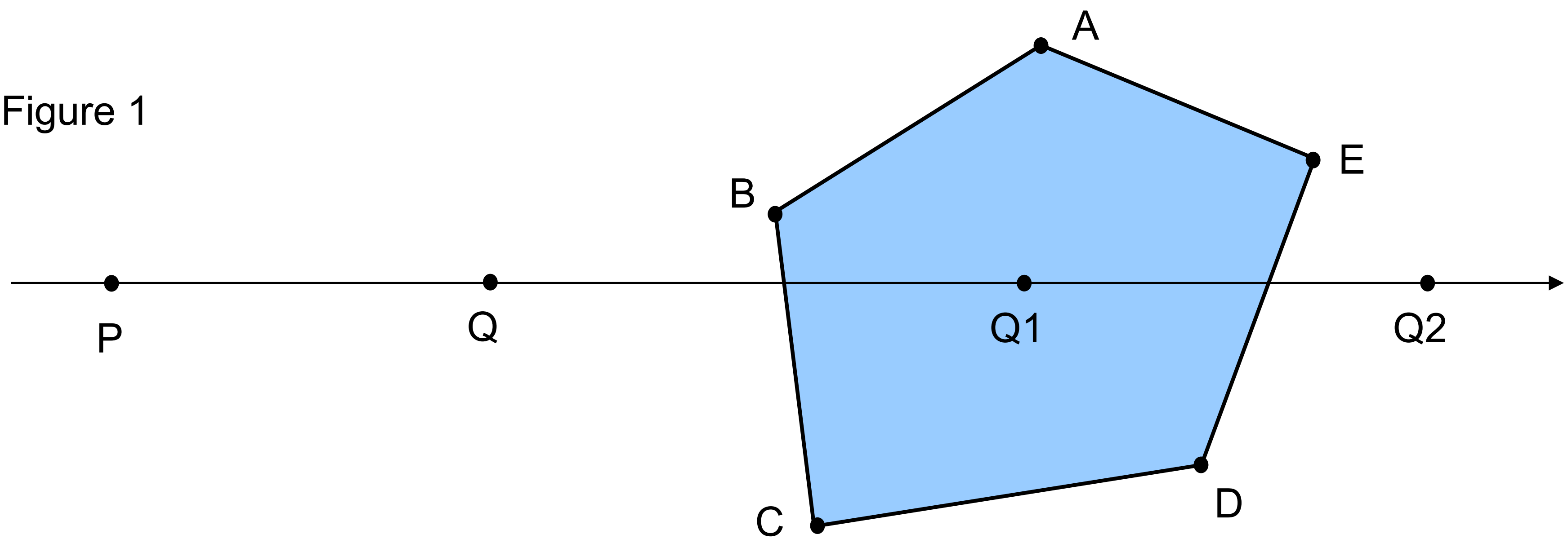
1. What are the main differences between clipping against view volume and clipping against window?
2. How is modified the image is the viewport is defined in such a way that $xv_{min} > xv_{max}$? Similarly for $yv_{min} > yv_{max}$. What is the result if both conditions are true?
3. How can be used the relation between window and viewport to achieve operations such as zoom in, zoom out, symmetry. What other operations could be achieved?
4. Give examples of using the window as drawing area, and as mask.
5. Explain how the endpoints clipping algorithm can be used for clipping the alphanumeric characters.
6. Give an example where the binary clipping algorithm is more efficient than other line clipping algorithms.
7. Describe a line clipping algorithm similar with Cohen-Sutherland algorithm if the window is a triangle.

Questions and proposed problems

8. Describe a line clipping algorithm similar with Cohen-Sutherland algorithm if the window is a rectangle with edge not parallel with coordinate axes.
9. Describe a line clipping algorithm similar with Cohen-Sutherland algorithm if the window is a rectangle with edge not parallel with coordinate axes.
10. How could you apply the Cohen-Sutherland algorithm to clip a triangle? How may you display the result? Can it be filled in?
11. Let us consider the convex polygon in figure 1, in the plain, and a line given by two points P and Q, oriented from P toward Q. Specify an approach to compute the normal vectors to the polygon’s edges.
12. Classify the edges of the polygon in figure 1, as entering and leaving type.
13. Specify the mathematical approach to compute the intersections between the PQ line and the polygon’s edges.

Questions and proposed problems

Figure 1



14. Determine by the Cyrus-Beck algorithm if the line PQ intersects the polygon, in figure 1.
15. Determine by the Cyrus-Beck algorithm what is the intersection between the line PQ and the polygon in figure 1.
16. Generalize the previous problem for any line and any convex polygon, in the plane.

Questions and proposed problems

17. Determine by the Cyrus-Beck algorithm what is the visible intersection point between the line PQ and the polygon in figure 1. Viewer lies on P.
18. Determine by the Cyrus-Beck algorithm what is the intersection between the line segment PQ and the polygon in figure 1.
19. Analyze the previous problem for different line segments: (a) PQ1; (b) PQ2.
20. Exemplify and explain why the Cyrus-Beck algorithm does not work for concave polygons.
21. Explain the Sutherland - Hodgman polygon clipping algorithm on the following polygon A and rectangular window (Figure 2).

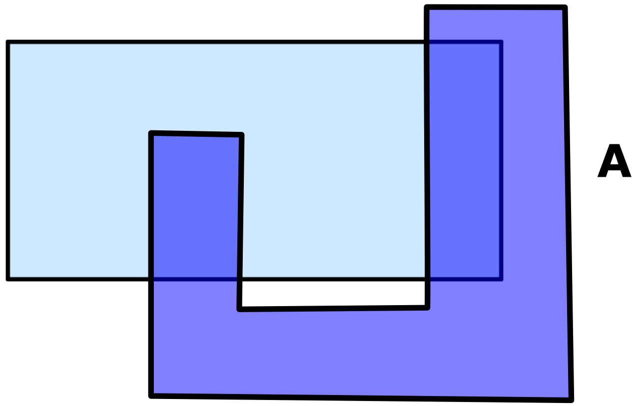


Figure 2

Questions and proposed problems

22. Explain the Sutherland - Hodgman polygon clipping algorithm on the following particular cases of polygon A and rectangular window (Figure 3).

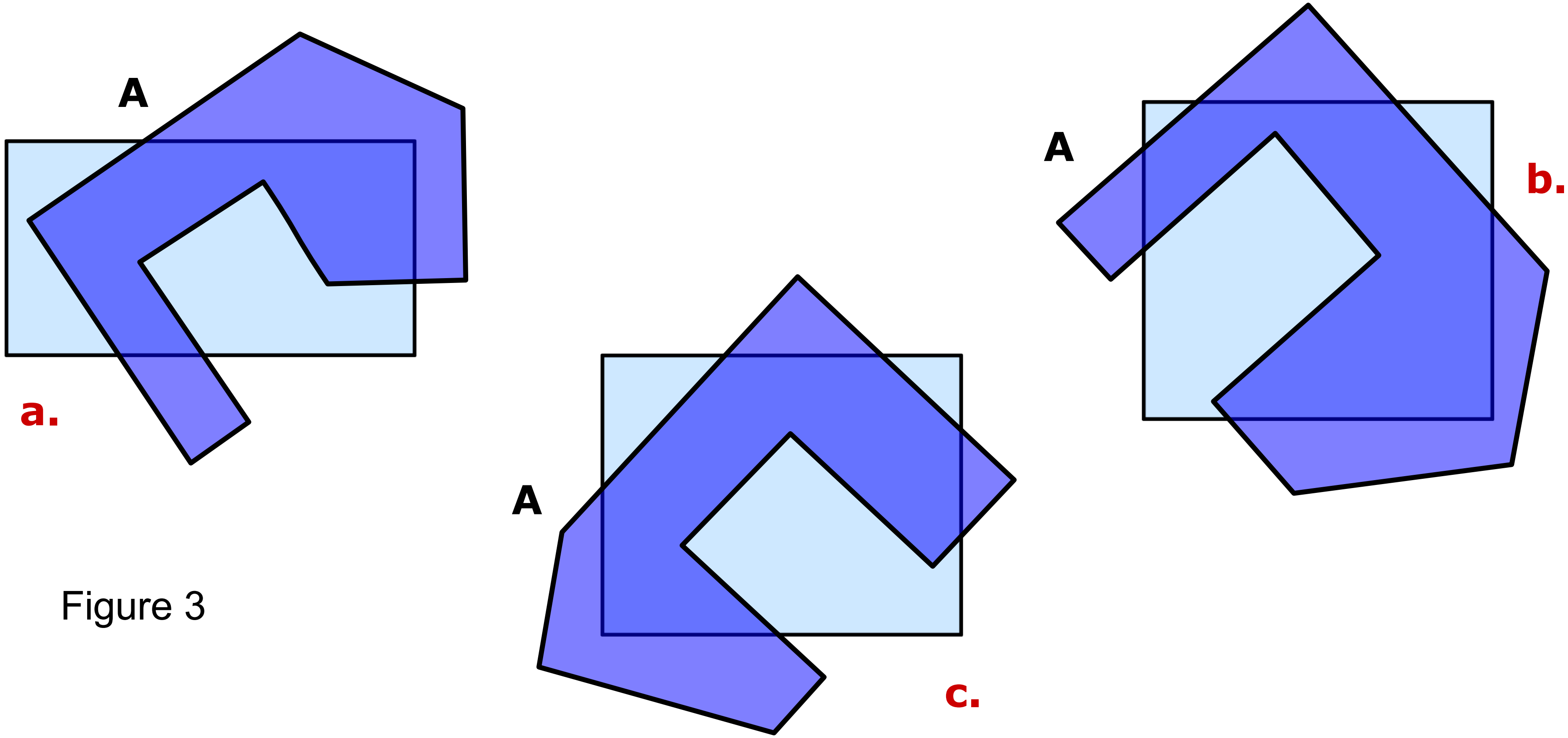


Figure 3

Questions and proposed problems

23. Explain the Sutherland - Hodgman polygon clipping algorithm on the following particular cases of polygon A and rectangular window (Figure 4).

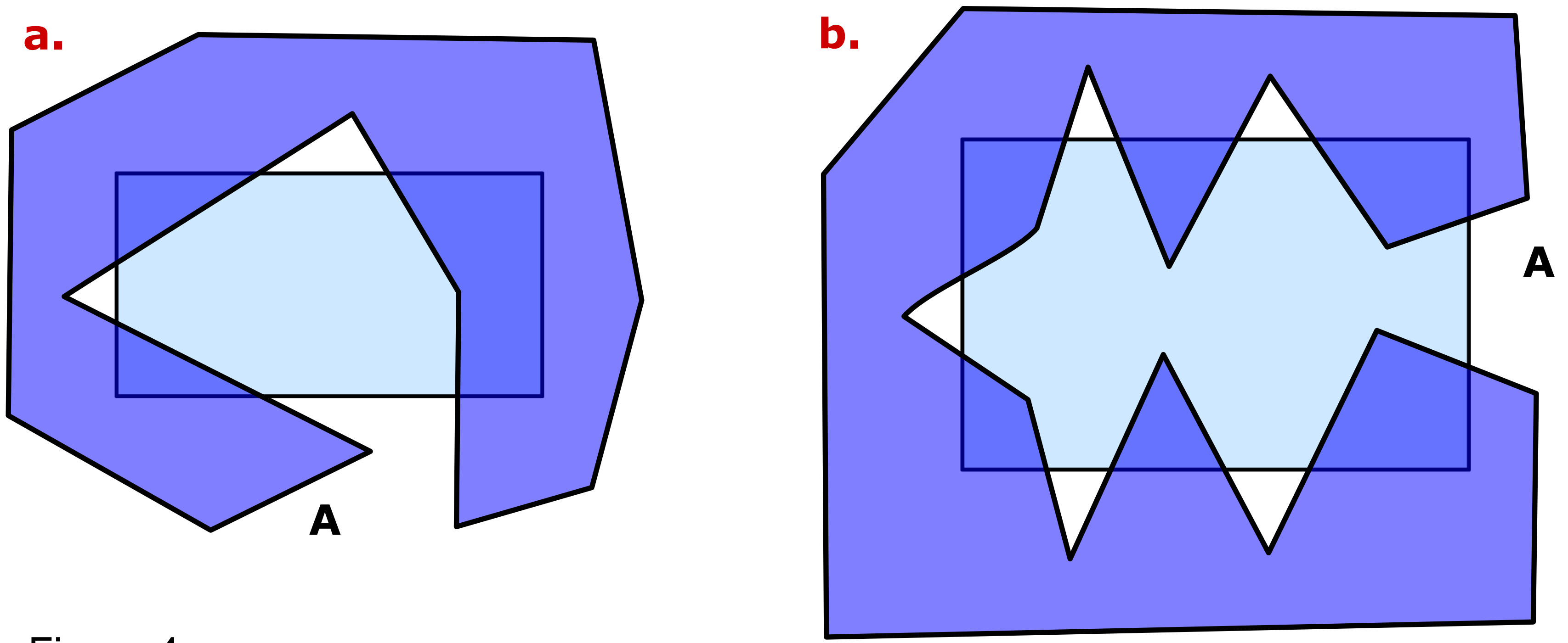


Figure 4

Questions and proposed problems

24. Explain the Weiler-Atherton polygon clipping algorithm on the following particular cases of polygon A and rectangular window (Figure 5).

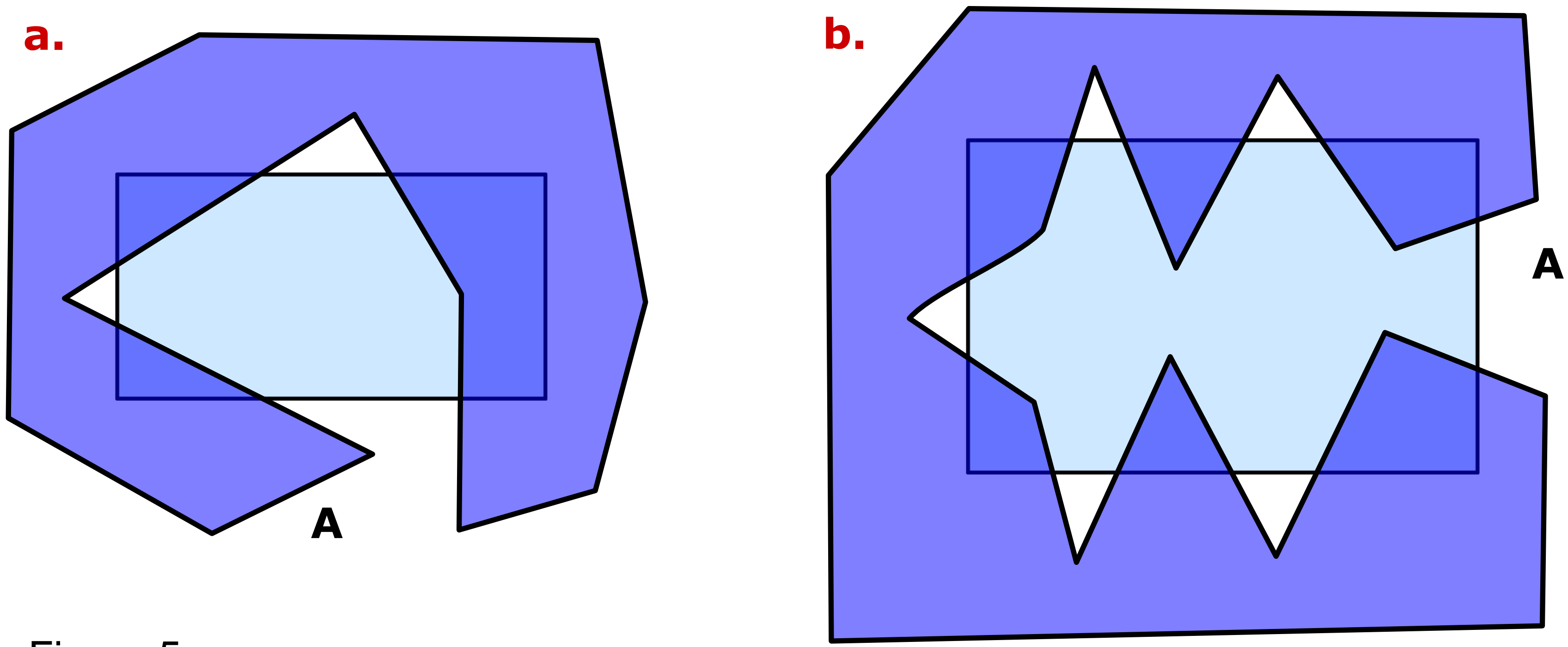


Figure 5

Questions and proposed problems

25. Explain the Weiler-Atherton polygon clipping algorithm on the following particular cases of polygons A and B (Figure 6).

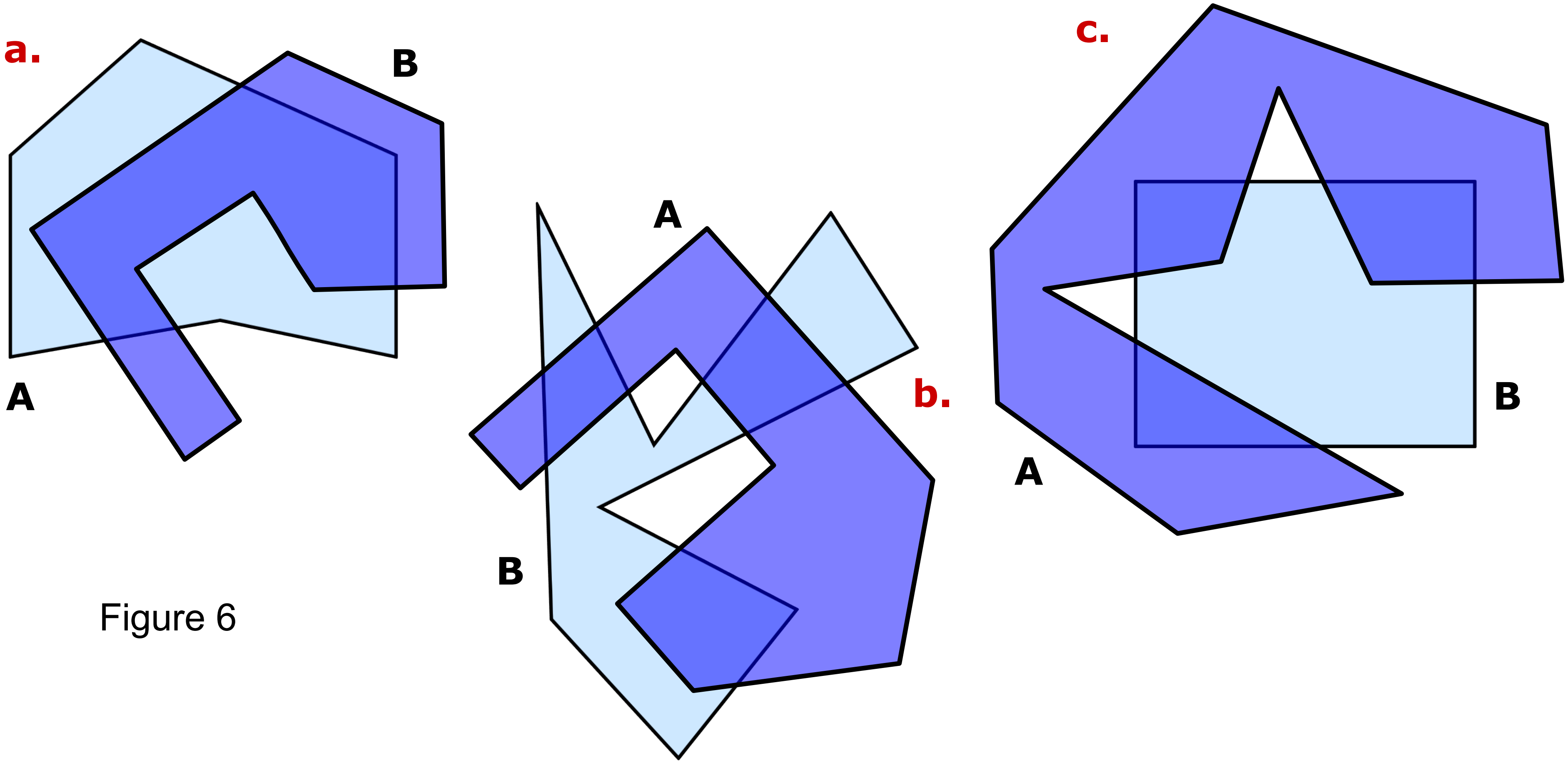


Figure 6

Questions and proposed problems

26. Explain an extension of the Weiler-Atherton polygon clipping algorithm on the following three polygons A, B and C (Figure 7).

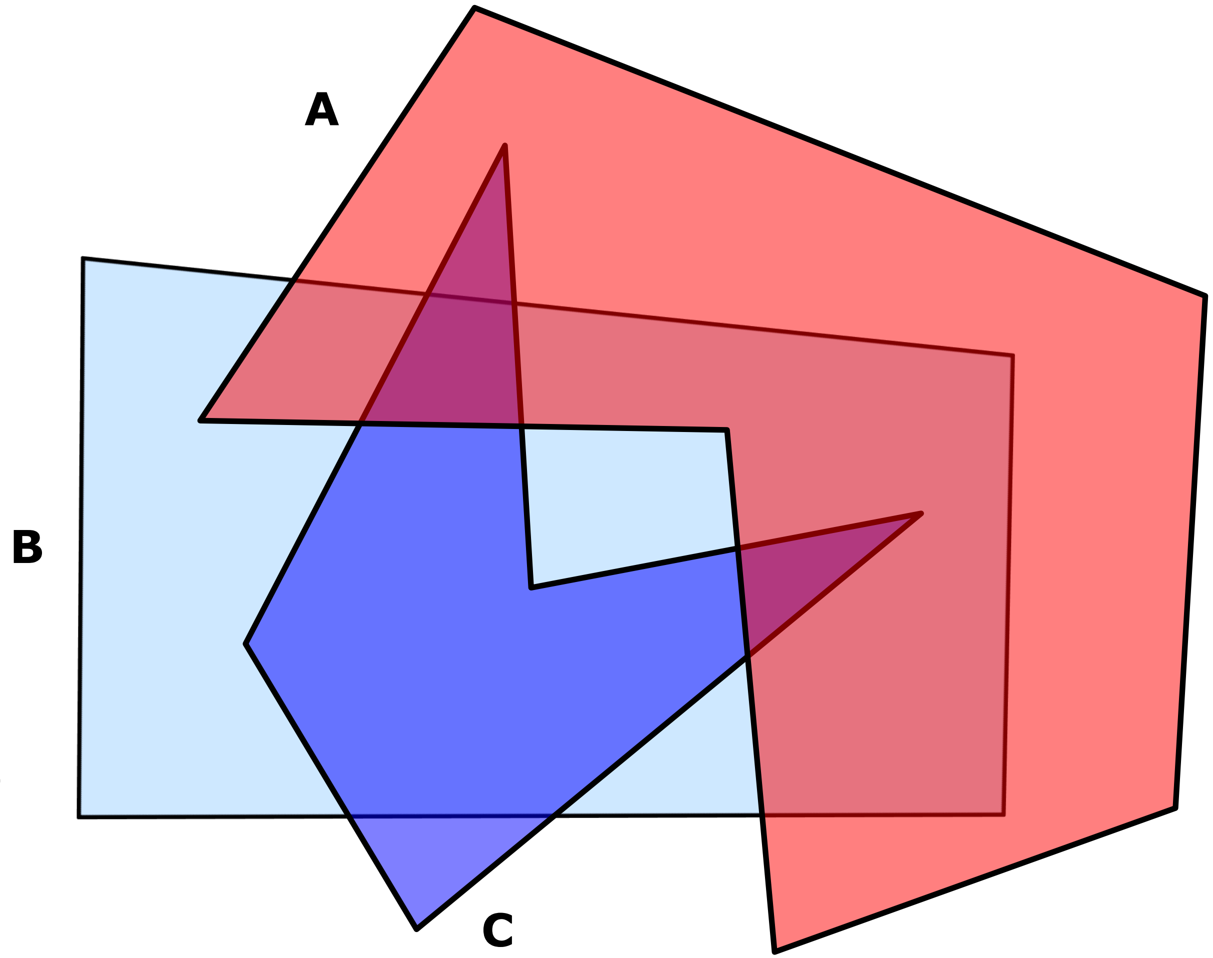


Figure 7

Questions and proposed problems

27. Explain an extension of the Weiler-Atherton polygon clipping algorithm on the following three polygons A, B and C (Figure 8).

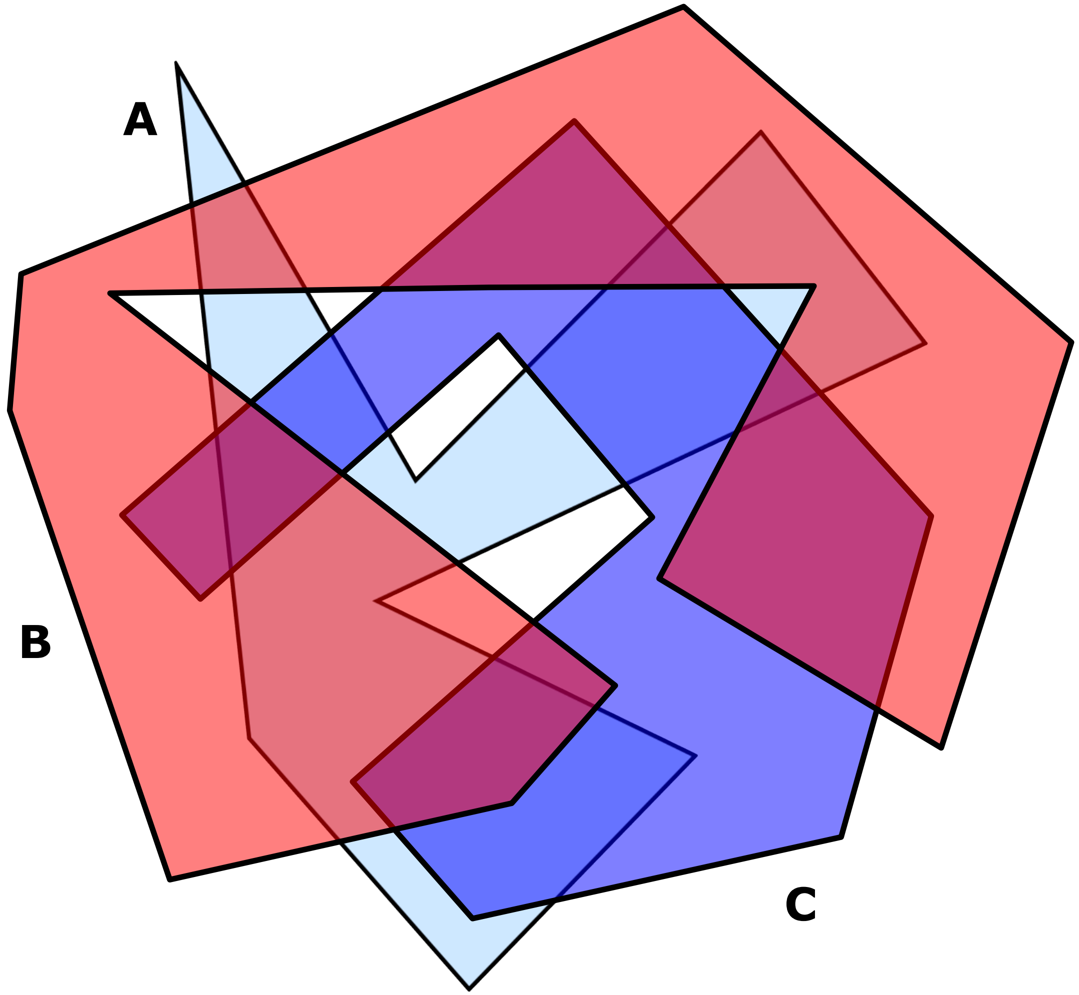


Figure 8