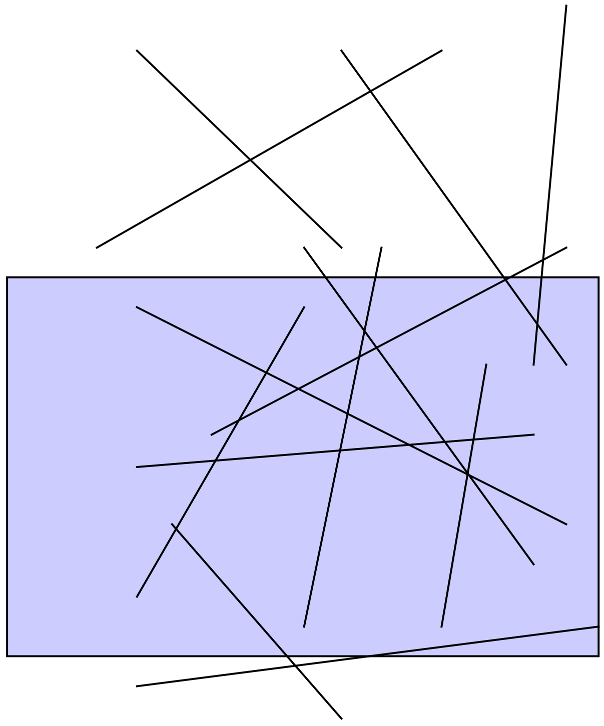
Computer Graphics: 7-Polygon Rasterization, Clipping

Prof. Dr. Charles A. Wüthrich,
Fakultät Medien, Medieninformatik
Bauhaus-Universität Weimar
caw AT medien.uni-weimar.de

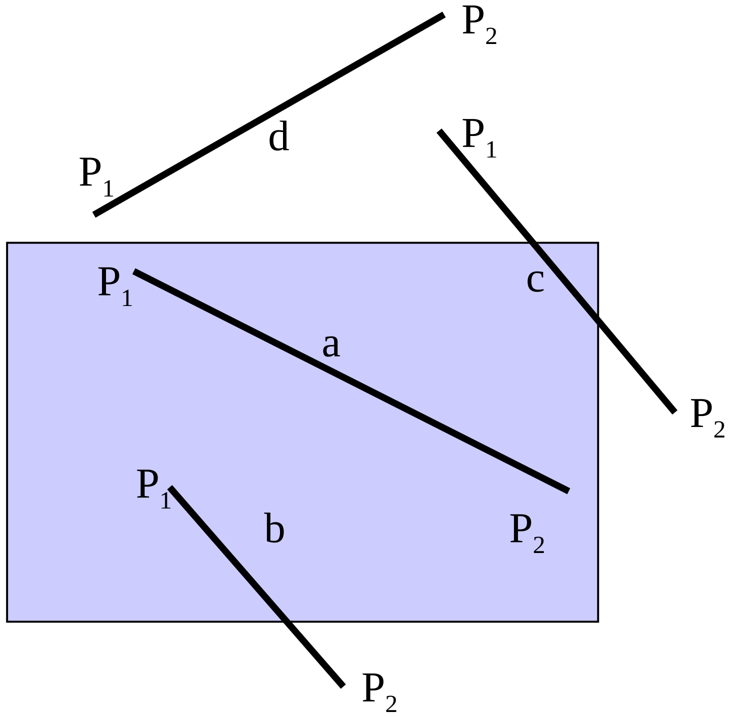
Clipping: motivation

- Often in 2D we have drawings that are bigger than a screen
- To save drawing complexity, it is good to be able to cut the drawings so that only screen objects are drawn
- Also, one needs to protect other (invisible) regions while working on a complex drawing
- The question is how is this done
- Problem: Given a segment in the plane, clip it to a rectangular segment



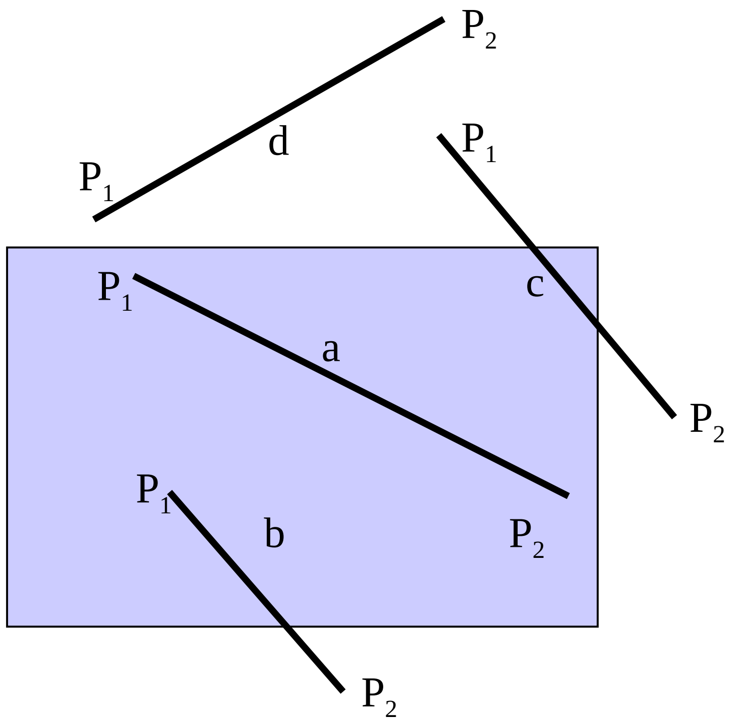
Line clipping

- Let B be the screen, and let P_1P_2 be the endpoints of the segment to be drawn
- There are four possible cases available:
  a) Whole line is visible
     $P_1,P_2 \in B$
  b) Line is partially visible
     $P_1 \in B$, $P_2 \in B$, P_1P_2 intersects screen borders
  c) Line partially visible
     P_1, $P_2 \notin B$, but P_1P_2 intersects screen borders
  d) Line not visible
     P_1, $P_2 \notin B$



Line clipping Algorithm

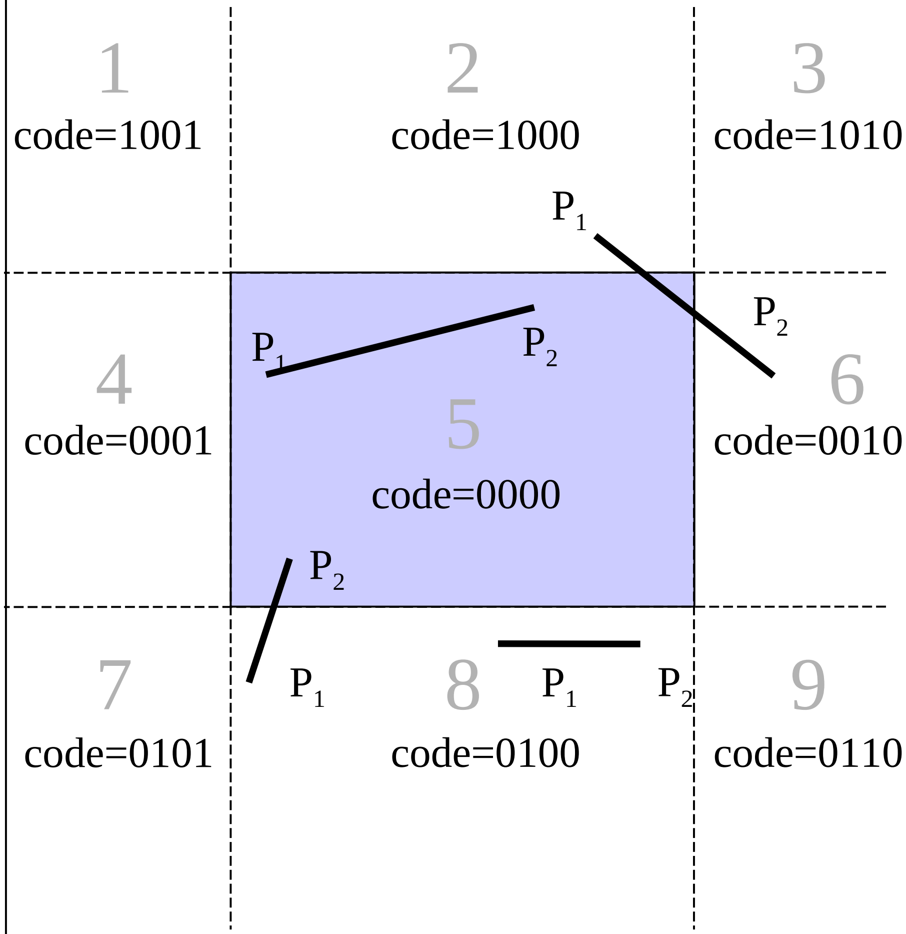
```
IF (P1,P2∈B)          /* a */
   DrawLine(P1,P2)
ELSE IF               /* b */
   (((P1∈B)AND NOT(P2∈B)) OR
   ((P2∈B)AND NOT(P1∈B)))
   compute I=(P1P2∩borders)
   IF(P1ÎB)
      Drawline(I,P1)
   ELSE
      DrawLine(I,P2)
ELSE                  /* c,d */
   compute I1,I2=
           (P1P2∩ borders)
   IF I1,I2 exist
      Drawline (I1,I2)
END
```



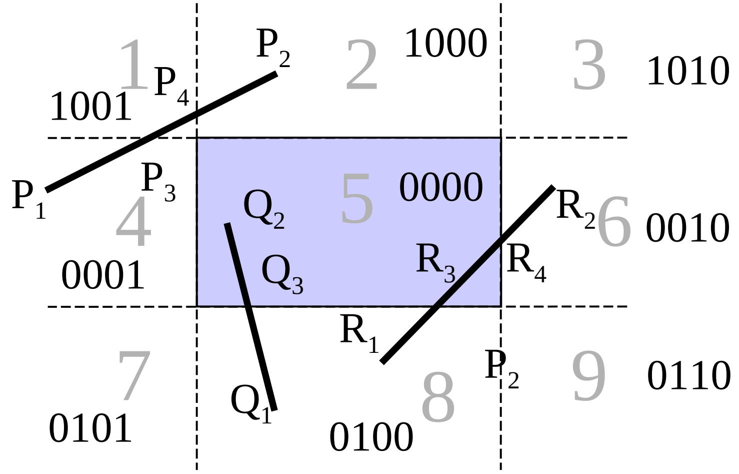
Examples: Cohen-Sutherland algo.

```
Code points according to
     characteristics:
  Bit 0=1 if xP<xmin else 0
  Bit 1=1 if xP>xmax else 0
  Bit 2=1 if yP<ymin else 0
  Bit 3=1 if yP>ymax else 0

Use bitwise operations:
  code(P1) AND code(P2)!= 0
       trivial case, line not
       on screen
  code(P1) OR code(P2) == 0
       trivial case, line
       on screen
  ELSE
  - compute line-borders intersection
     (one at time) and set their code as
     above
  - redo clipping with shortened line
Note: before new intersection, at least
     one endpoint is outside WRT the
     border you clipped against, thus
     one subseg is trivially out (all
     left or right or up or down of
     screen)
```



Algorithm Examples



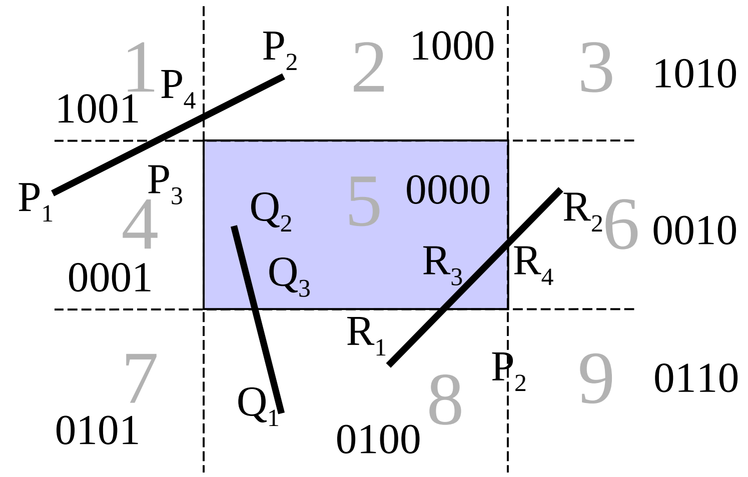
Algorithm examples

P_1P_2: P_1=0001, P_2=1000
P_1 AND P_2= 0000
P_1 OR P_2=1001
Subdivide against left,
Pick P_2, find P_4
new line P_2P_4
P_2P_4: P_2=1000, P_4=1000
P_2 AND P_4: 1000 outside!
Draw nothing

Q_1Q_2: Q_1=0100, Q_2=0000
Q_1 AND Q_2:0000
Q_1 OR Q_2: 0100
Subdivide, Pick Q_2, find Q_3
new line Q_2Q_3
Q_2Q_3: Q_2=0000, Q_3=0000
Q_2 AND Q_3: 0000
Q_1 OR Q_3: 0000 inside!
Draw Q_3Q_2
Q_3Q_2: Q_3=0100

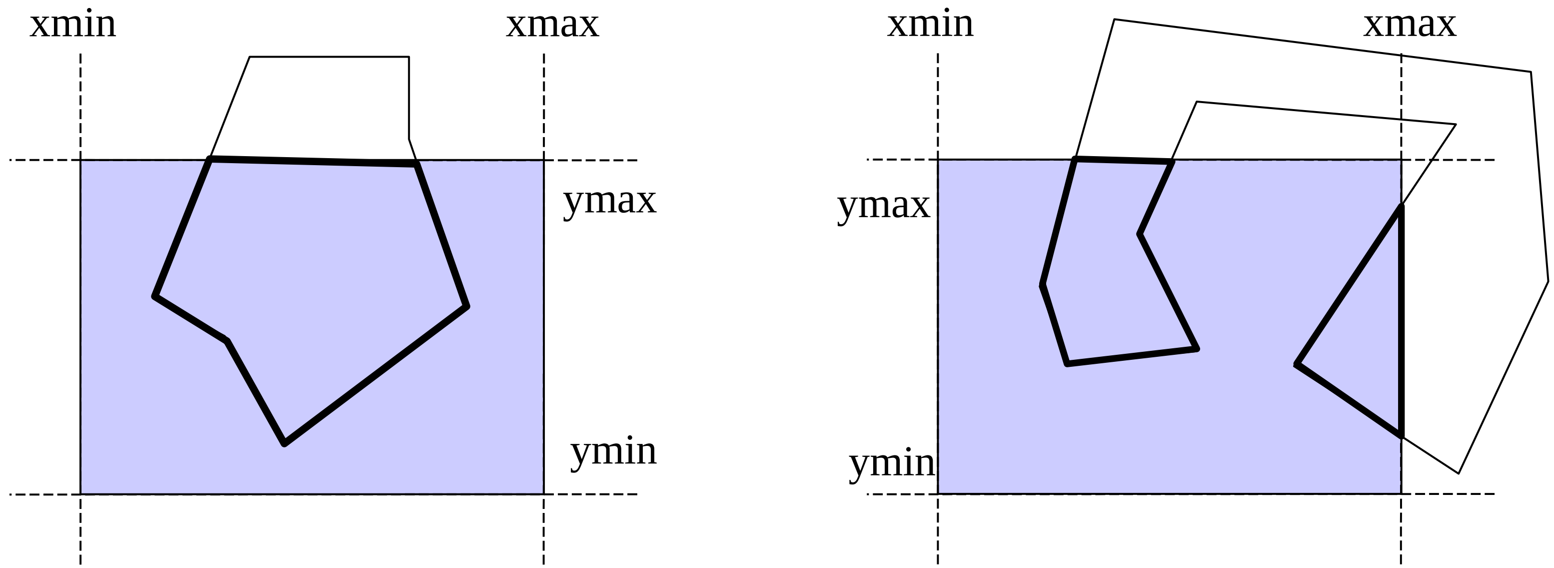
~

R_1R_2: R_1=0100, R_2=0010
R_1 AND R_2= 0000
R_1 OR R_2= 0110
Subdivide, Pick R_1, find R_4
new line R_1R_4
R_1=0100, R_4=0000
R_1 AND R_4= 0000
R_1 OR R_4= 0100
Subdivide, Pick R_4, find R_3
new line R_3R_4
R_3=0000 R_4=0000
R_3 AND R_4=0000
draw R_3R_4



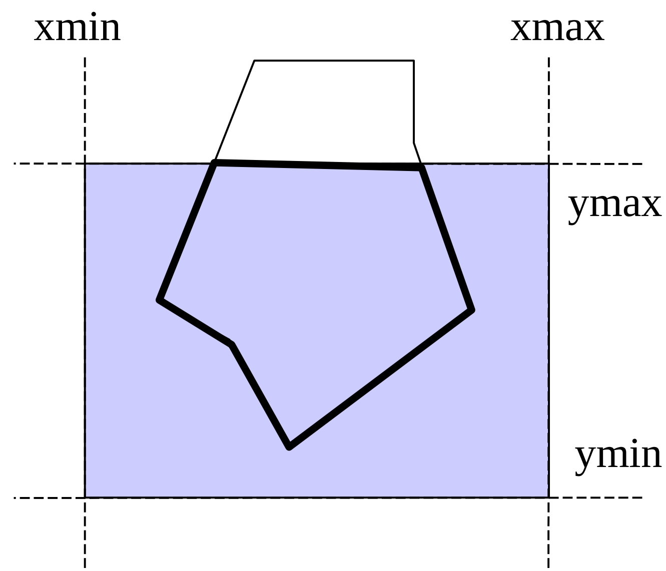
Clipping polygons

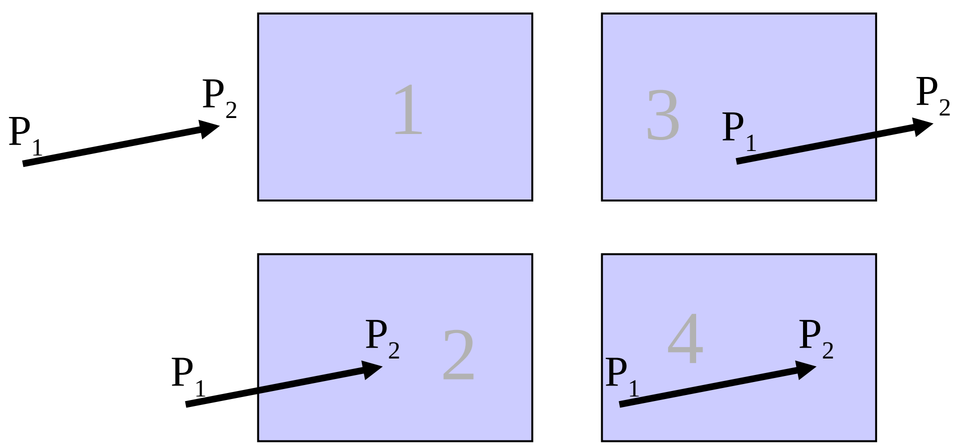
- The task is similar, but it is more complicated to achieve
- Polygon clipping may result into disjunct polys



Sutherland Hodgeman Algorithm

- Clearly, drawing polygons is a more complicated issue
- Idea: one could follow the polygon border, and switch to following the border when the polygon leaves the screen until it re-enters it
- This means creating a new polygon, which is trimmed to the screen
- While following an edge, four cases are possible:

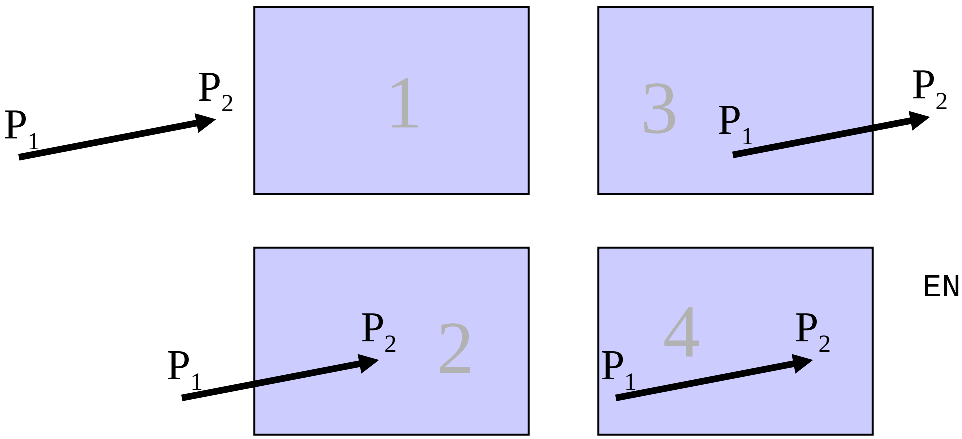




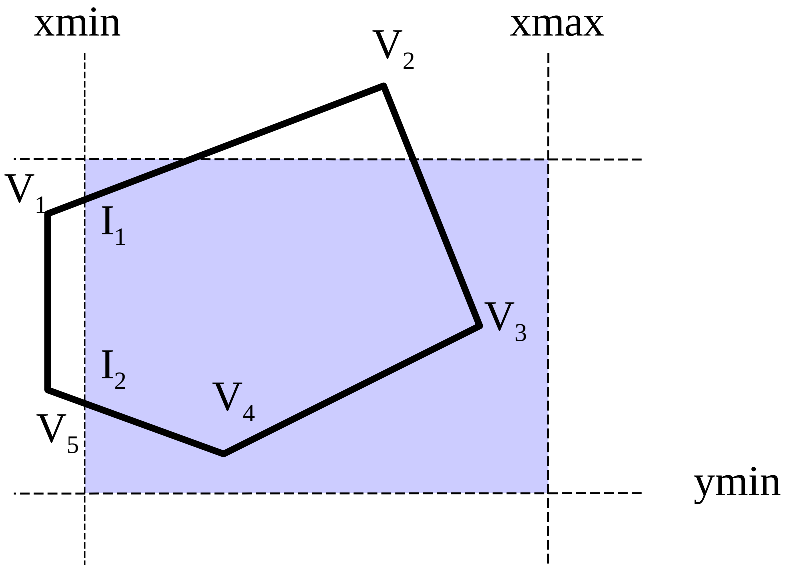
Sutherland-Hodgeman Algorithm

- The algorithm works considering polygons as lists of edges
- Input is a list L of polygon edges
- Output wil be a new list L´ of polygon edges
- The polygon is clipped against ALL screen borders one at a time

```
FOR all screen borders DO:
  FOR all lines in polygons
   DO:
    FOR all points P in L DO
      Compute intersection I
       of line with current
       border
      IF (case 1):
        Do Nothing
      IF (case 2):
        Add (I,Succ(P))to L´
      IF (case 3):
        Add (I) to L´
      IF (case 4):
        Add (succ(P)) to L´
    END
  END
END
```



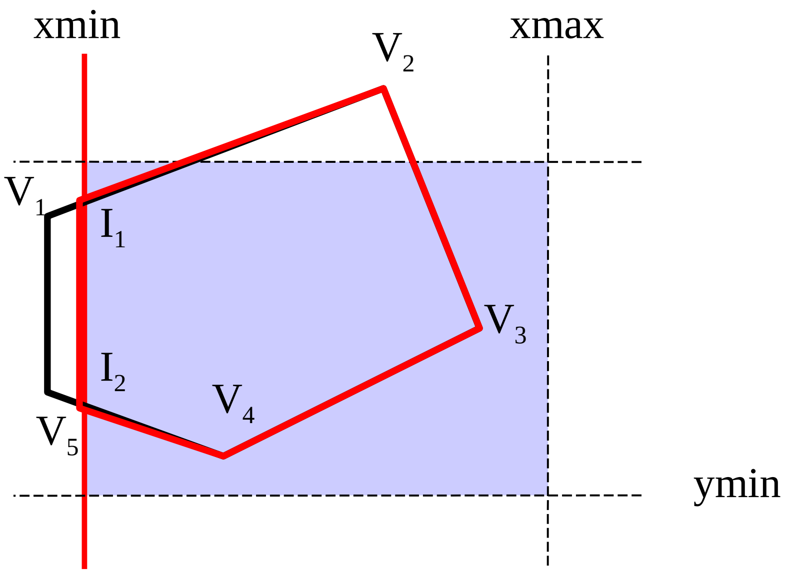
Example

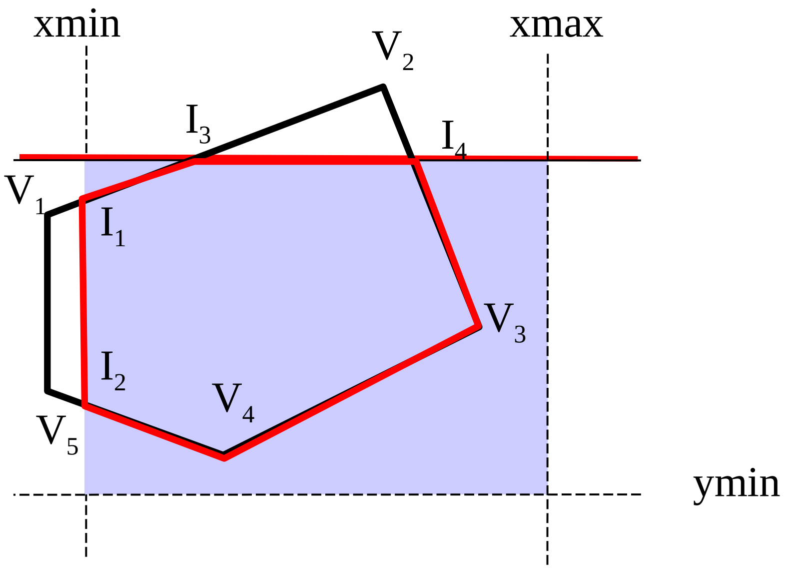


Example

- Left border
  Input: $\{V_1, V_2, V_3, V_4, V_5\}$
  Output: $\{I_1, V_2, V_3, V_4, \acute{I}_2\}$

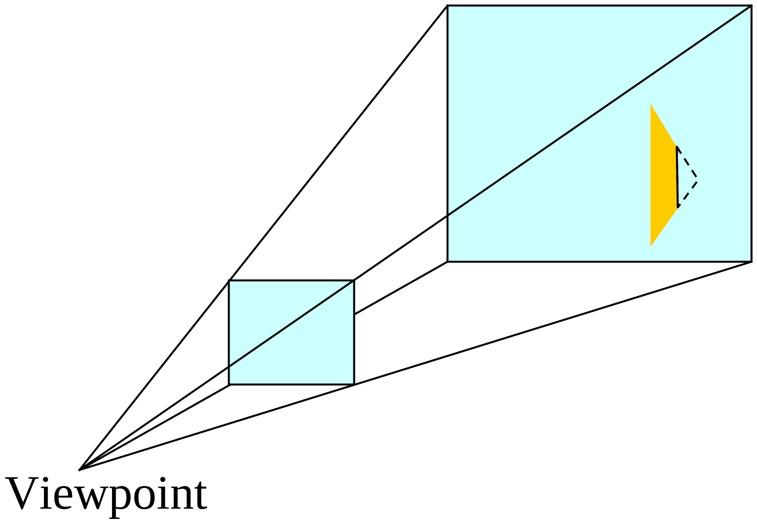
- Top Border
  Input: $\{I_1, V_2, V_3, V_4, I_2\}$
  Output: $\{I_1, I_3, I_4, V_3, V_4, I_2\}$





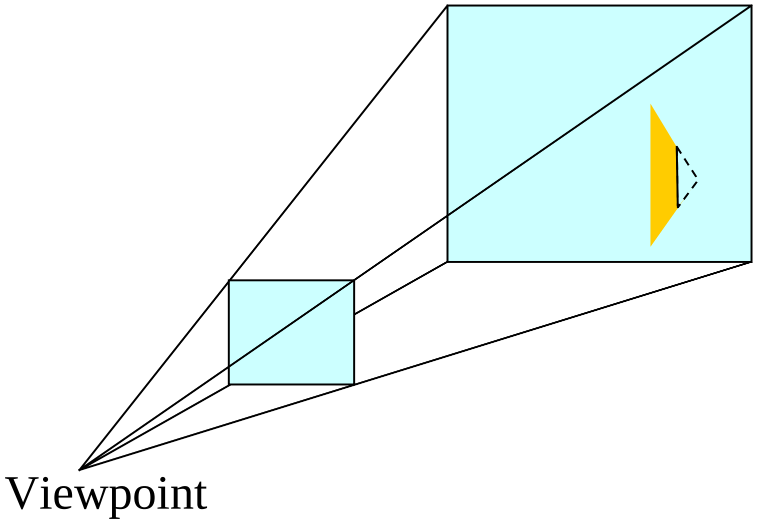
Clipping in 3D

- Remember the near and far clipping planes of the view frustum?
- How do I clip a polygon against them?



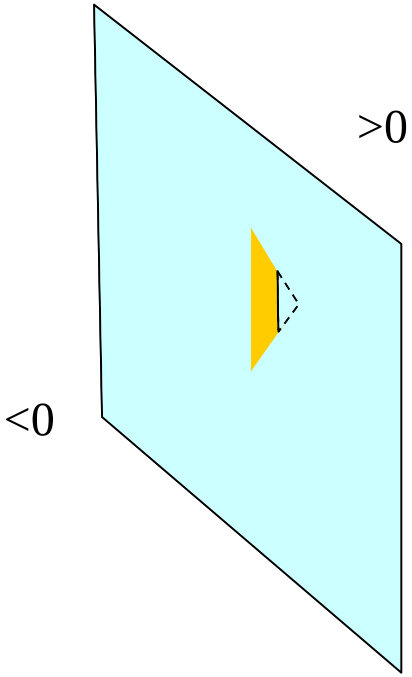
Clipping in 3D

- Remember the near and far clipping planes of the view frustum?
- How do I clip a polygon against them?
- As a matter of fact, it is not so different!
- The problem can be reduced to the same as in 2D, with a few differences



Clipping in 3D

- Let us consider a the far plane and a polygon
- Substitute the coordinates of the vertices of the triangle into the plane equation:
  - Front: <0
  - Back: >0
  - Plane: =0
- So we can follow the vertices exactly like in Cohen-Sutherland to clip against the plane
- A similar method can be applied for an arbitrary plane
- For the frustum planes one can do clipping one plane at a time, like in 2D (except they are 6 now)



End

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++