9.6.1. Cohen-Sutherland Algorithm

(UPTU B.Tech, 2013-14)

In computer graphics, the **Cohen-Sutherland** algorithm is a line clipping algorithm. The algorithm divides a 2D space into 9 parts, using the infinite extensions of the four linear boundaries of the window. Assign a bit pattern to each region as shown below:

The numbers in the figure above are called **outcodes.** An outcode is computed for each of the two points in the line. The bits in the outcode represent: **Top, Bottom, Right, Left.**

Each bit in the code is set to either a 1 (true) or a 0 (false). If the region is to the *left* of the window, the ***first*** bit of the code is set to 1. If the region is to the *Right* of the window, the *second* bit of the code is set to 1. If to the ***Bottom***, the ***third*** bit is set, and if to the ***Top***, the ***fourth*** bit is set. The 4 bits in the code then identify each of the nine regions.

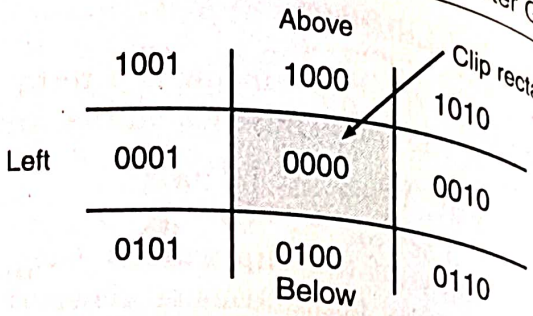


Fig. 9.7.

For any endpoint (x, y) of a line, the code can be determined that identifies which region the endpoint lies. The code's bits are set accordi to the following conditions:

- First bit set 1: Point lies to **left** of window $x < x_{min}$
- Second bit set 1: Point lies to **right** of window $x > x_{max}$
- Third bit set 1: Point lies **below** (bottom) window $y < y_{min}$
- Fourth bits set 1: Point lies **above** (top) window $y > y_{max}$

For example,

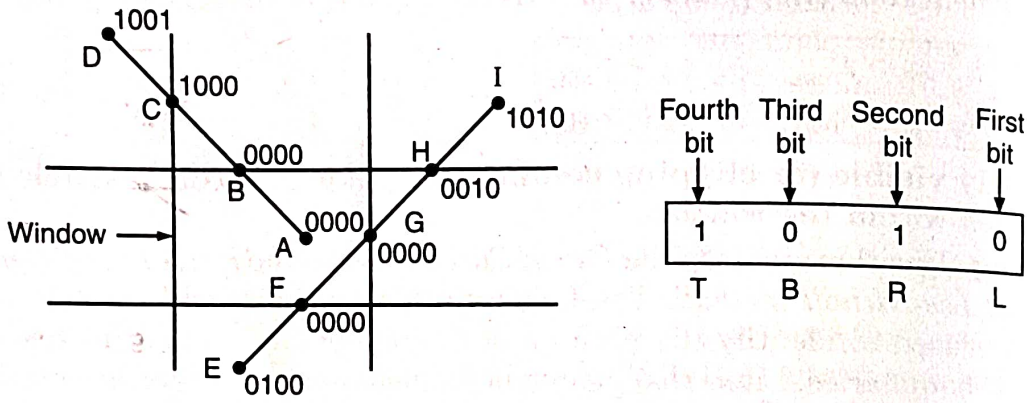


Fig. 9.8.

The outcode 1010 represents a point that is top-right of the viewport. Note that t outcodes for endpoints **must** be recalculated on each iteration after the clipping occurs.

The Algorithm

The Cohen-Sutherland algorithm uses a divide-and-conquer strategy. The line segmen endpoints are tested to see if the line can be trivially accepted or rejected. If the line can be trivially accepted or rejected, an intersection of the line with a window edge is determin and the trivial reject/accept test is repeated. This process is continued until the line accepted.

To perform the trivial acceptance and rejection tests, we extend the edges of the wind to divide the plane of the window into the nine regions. Each endpoint of the line segme is then assigned the code of the region in which it lies.

1. Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2. Compute the 4-bit codes for each endpoint.
   If both codes are **0000,** (bitwise OR of the codes yields 0000) line lies complete **inside** the window: pass the endpoints to the draw routine.
   If both codes have a 1 in the same bit position (bitwise AND of the codes is **not** 000 the line lies **outside** the window. It can be trivially rejected.

3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be **clipped** at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P_1 = (x_1, y_1)$. Read P_1's 4-bit code in order: **Left-to-Right, Bottom-to-Top.**
5. When a set bit (1) is found, compute the **intersection I** of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with I and repeat the algorithm.

> **Note:** If both endpoints of a line entirely to one side of the window the line must lie entirely outside of the window. It is trivally rejected and if both end points of a line lie4 inside the window, the entire lie lies inside the window. It is trivially accepted.

Before Clipping

1. Consider the line segment **AD**.

Point **A** has an outcode of **0000** and point **D** has an outcode of **1001**. The logical AND of these outcodes is zero; therefore, the line cannot be trivially rejected. Also, the logical OR of the outcodes is no zero; therefore, the line cannot be trivially accepted. The algorithm then chooses **D** as the outside point (its outcode contiain 1's). By our testing order, we first use the top edge to clip **AD** at **B**. The algorithm then recomputes **B's** outcode as **0000.** With the next iteration of the algorithm, **AB** is tested and is trivially accepted and displayed.

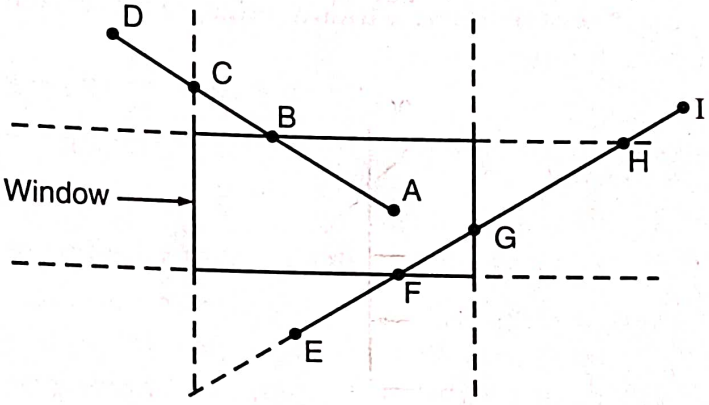


Fig. 9.9.

2. Consider the line segment **EI**

Point **E** has an outcode of **0100,** while point **I's** outcode is **1010.** The results of the trivial tests show that the line can neither be trivially rejected or accepted. Point **E** is determined to be an outside point, so the algorithm clips the line against the bottom edge of the window. Now line **EI** has been clipped to be line **FI**. Line **FI** is tested and cannot be trivially accepted or rejected. Point **F** has an outcode of **0000,** so the algorithm chooses point **I** as an outside point since its outcodde is **1010.** The line **FI** is clipped against the window's top edge, yielding a new line **FH.** Line **FH** cannot be trivially accepted or rejected. Since **H's** outcode is **0010,** the next iteration of the algorthm clips against the window's right edge, yielding line **FG**. The next iteration of the algorithm tests **FG**, and it is trivially accepted and display.

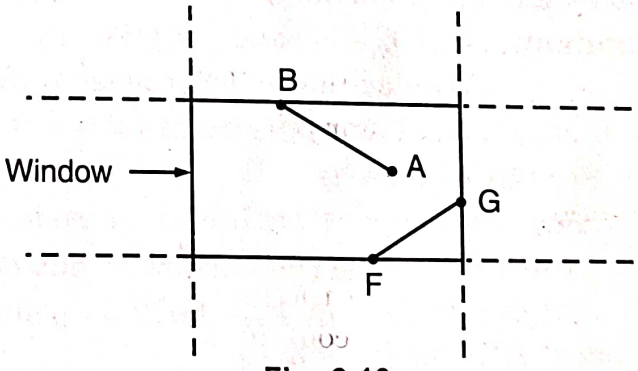


Fig. 9.10.

After Clipping

After clipping the segments **AD** and **EI**, the result is that only the line segments **AB** and FG can be seen in the window.

9.6.2. Line Intersection and Clipping

We determine the intersection points of the lines in category 3 *i.e.* clipping condidate with the boundaries of the window. There intersection points then subdivide the line segment into several line segments which can belong only to visible or nonvisible category *i.e.* category 1 or category 2. The segment in category 1 will be clipped line segment.

The intersection points are found by solving the equations representing the line segment and the boundary lines.

- For left window edge, intersection point will be (x_L, y)

$$y = m(x_L - x_1) + y_1,\ m \neq \infty$$

- For right window edge, intersection point will be (x_R, y)

$$y = m(x_R - x_1) + y_1,\ m \neq \infty$$

- For top window edge, intersection point will be (x, y_T)

$$y = x_1 + \frac{1}{m}(y_T - y_1),\ m \neq 0$$

- For bottom window edge, intersection point will be (x, y_B)

$$x = x_1 + \frac{1}{m}(y_B - y_{1m}),\ m \neq 0$$

Note:

In all above equations

x_L is x-value of left edge of clipping window

x_R is x-value of right edge of clipping window

y_B is y-value of bottom edge of clipping window

y_T is y-value of top edge of clipping window.

and acceptable value of x and y are

$$x_L \leq x \leq x_R \qquad y_B \leq y \leq y_T$$

EXAMPLE 9.3.

Use the cohen Sutherland algorithm to clip line P_1 (70, 20) and P_2 (100, 10) against a window lower left hand corner (50, 10) and upper right hand corner (80, 40).

Solution: $P_1(70, 20)$ and $P_2(100, 10)$

Window lower left corner = (50, 10)

Window upper right corner = (80, 40)

So, the window is

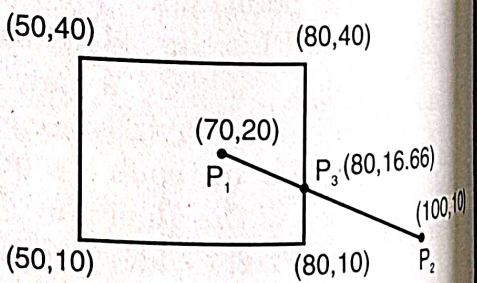


Fig. 9.11.

Now, we assign 4 bit binary outcode.

Point P_1 is inside the window so outcode of $P_1 = 0000$

and the outcode of $P_2 = 0010$ as point P_2 is right of window AND of P_1 and P_2

$$\begin{array}{r} 0\ 0\ 0\ 0 \\ 0\ 0\ 1\ 0 \\ \hline 0\ 0\ 0\ 0 \end{array}$$

The result of AND operation is zero so line is partially visible. Slop of line P_1P_2 is $m =$

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{10 - 20}{100 - 70} = \frac{-10}{30} = -\frac{1}{3}$$

We have to find intersection of line $P_1 P_2$ with right edge of window *i.e.* Point P_3.
Let the intersection point be (x, y)
here $x = 80$, we have to find the value of y.
We use point $P_2(x_2, y_2) = P_2(100, 10)$

$$m = \frac{y - y_2}{x - x_2}$$

$$-\frac{1}{3} = \frac{y - 10}{80 - 100}$$

$$y - 10 = \frac{20}{3} \Rightarrow y = \frac{20}{3} + 10 = 16.66.$$

$$y = 16.66$$

∴ Thus the intersection point $P_3 = (80, 16.66)$

So, after clipping line P_1P_2 against window, new line is P_1P_3 with co-ordinates $P_1(70, 20)$ and $P_3(80, 16.66)$.

EXAMPLE 9.4.

Given a clipping window A(20, 20) B(60, 20) C(60, 40), D(20, 40). Using Sutherland Cohen algorithm find the visible portion of line segment joining the points P(40, 80) Q(120, 30).

Solution:

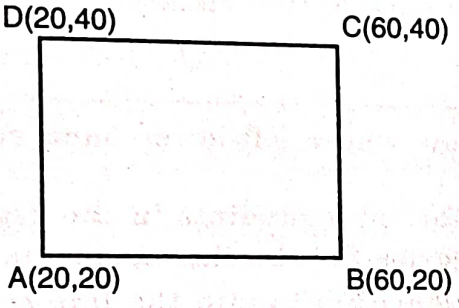


Fig. 9.12.

Here $x_L = 20 \quad y_B = 20$
$x_R = 60 \quad y_T = 40$

As we know, the outcodes can be calculated as

Bit 1 = sign of $(y - y_T)$
Bit 2 = sign of $(y_B - y)$
Bit 3 = sign of $(x - x_R)$
Bit 4 = sign of $(x_L - x)$

and sign = 1 if value is +ve and sign = 0 if value is –ve.

Thus, the outcodes of $P(40, 80)$ is 1000
and $Q(120, 30)$ is 0010.

Both endpint codes are not zero and their logical AND is zero. Hence, line cannot be rejected as invisible.

$$\text{Slope of } PQ(m) = \frac{y_2 - y_1}{x_2 - x_1} = \frac{30 - 80}{120 - 40} = -\frac{5}{8}$$

and intersections with window edge are

- **Left** $y = m(x_L - x_1) + y_1 = -\frac{5}{8}(20 - 40) + 80$

  $= 92.5$, which is greater than y_T and hence rejected.

- **Right** $y = m(x_R - x_1) + y_1$

  $= -\frac{5}{8}(60 - 40) + 80 = 67.5$, which is greater than y_T

  so, it is rejected.

- **Top** $x = x_1 + \frac{1}{m}(y_T - y_1) = 40 + \frac{1}{-\frac{5}{8}}(40 - 80) = 104,$

  which is greater than x_R and hence rejected.

- **Bottom** $x = x_1 + \frac{1}{m}(y_B - y_1) = 40 + \frac{1}{-\frac{5}{8}}(20 - 80) = 136,$

  which is greater than x_R and hence rejected.

Since both the values of y are greater than y_T and both are also greater than x_R. Theref the line segment PQ completely outside the window.

EXAMPLE 9.5.

Let R be the rectangular window whose left-lower hand corner is at $L(-3, 1)$ and upp right-hand corner is at $R(2, 6)$.

(a) Find the region codes for the endpoints in the Fig.

(b) Find the clipping categories for the line segments

(c) Use cohen-sutherland algorithm to clip the line segments.

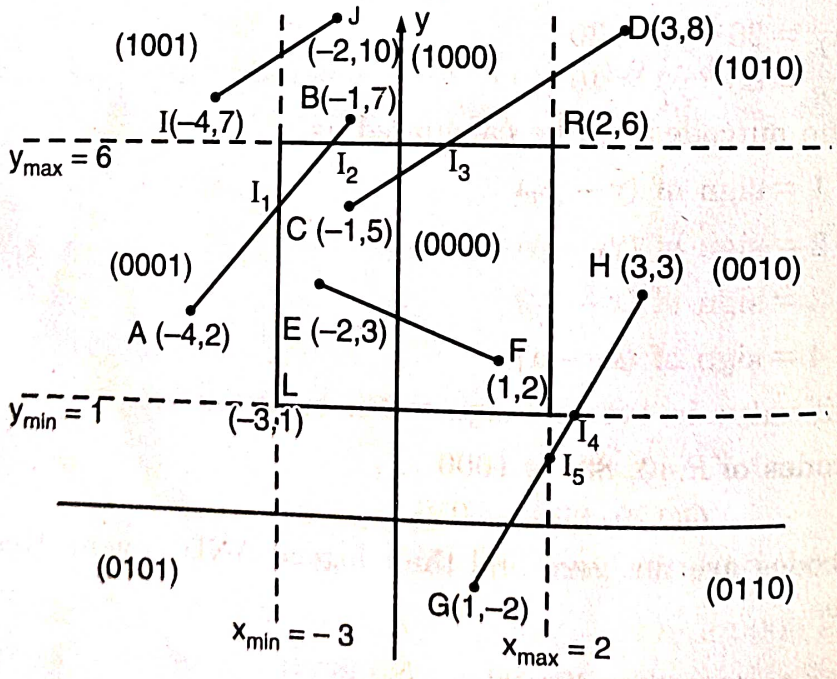


Fig. 9.13.

Solution: (a) The region code or outcode for point (x, y) is set according to

$$\left.\begin{aligned}\text{Bit 1} &= \text{sign}(y - y_{max}) = \text{sign}(y - 6)\\ \text{Bit 2} &= \text{sign}(y_{min} - y) = \text{sign}(1 - y)\\ \text{Bit 3} &= \text{sign}(x - x_{max}) = \text{sign}(x - 2)\\ \text{Bit 4} &= \text{sign}(x_{min} - x) = \text{sign}(-3 - x)\end{aligned}\right\} \text{ and sign } (a) = \begin{cases}1, & \text{if a is + ve}\\ 0 & \text{otherwise}\end{cases}$$

So, $A(-4, 2) \rightarrow 0001$ $\quad F(1, 2) \rightarrow 0000$

$B(-1, 7) \rightarrow 1000$ $\quad G(1, -2) \rightarrow 0100$

$C(-1, 5) \rightarrow 0000$ $\quad H(3, 3) \rightarrow 0010$

$D(3, 8) \rightarrow 1010$ $\quad I(-4, 7) \rightarrow 1001$

$E(-2, 3) \rightarrow 0000$ $\quad J(-2, 10) \rightarrow 1000$

(b) Now by testing the outcodes, we get the appropriate categories.

Category 1 (Visible): $\overline{EF}$ line segment because the outcodes for both endpoints is 0000

Category 2 (Not Visible): $\overline{IJ}$ line segment, because

$$\begin{array}{r} 1\ 0\ 0\ 1 \\ \text{AND}\ 1\ 0\ 0\ 0 \\ \hline 1\ 0\ 0\ 0 \end{array}$$ (which is not 0000)

Category 3 (Candidates of Clipping):

(1) $\overline{AB}$ line segment since (0001) AND (1000) = 0000

(2) $\overline{CD}$ line segment since (0000) AND (1010) = 0000

(3) $\overline{GH}$ line segment since (0100) AND (0010) = 0000

(c) The clipping lines are $\overline{AB}$, $\overline{CD}$ and $\overline{GH}$.

For Line $\overline{AB}$:

$$m = \frac{7-2}{-1+4} = \frac{5}{3}$$

$$\frac{5}{3} = \frac{y - y_2}{x - x_2}$$

$$\Rightarrow \quad \frac{5}{3} = \frac{y - 7}{-3 + 1} \qquad [\because \quad x = -3]$$

$$\Rightarrow \quad y = \frac{11}{3} = 3\frac{2}{3}$$

Thus, the resulting intersection point is $I_1\left(-3, 3\frac{2}{3}\right)$.

Now, we clip $\overline{AI_1}$ segment and work on $\overline{I_1B}$.

The code of I_1 is 0000 and category of $\overline{I_1B}$ is 3 since 0000 AND 1000 = 0000. Now B is outside the window its outcode is 1000, so we push the 1 to 0 by clipping against the line $y_{max} = 6$.

Now $$\frac{5}{3} = \frac{y - y_2}{x - x_2}$$

$$\Rightarrow \quad \frac{5}{3} = \frac{6 - \frac{11}{3}}{x + 3} \quad [\because \ y = 6]$$

$$\Rightarrow \quad 5x + 15 = 7$$

$$\Rightarrow \quad x = -\frac{8}{5} = -1\frac{3}{5}.$$

So, the resulting intersection point I_2 is $\left(-1\frac{3}{5}, 6\right)$

Thus $\overline{I_2B}$ is clipped. The code for I_2 is 0000. The remaining segment $\overline{I_1I_2}$ is displa since both endpoints lie in the window.

For Line $\overline{CD}$

The outcode of D is 1010 *i.e.* it is outside the window. We push the first 1 to 0 by clipp against the line $y_{max} = 6$.

Here $$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8 - 5}{3 + 1} = \frac{3}{4}$$

and $$y = 6$$

$$\frac{3}{4} = \frac{6 - 5}{x + 1}$$

$$\Rightarrow \quad x = \frac{1}{3}.$$

So, the resulting intersection point I_3 is $\left(\frac{1}{3}, 6\right)$ and its code is 0000. Thus $\overline{I_3D}$ clipped and the remaining segment $\overline{CI_3}$ is displayed since it has both endpoints outcod 0000.

For Line $\overline{GH}$

We can start with either G or H. The code for G is 0100 we push the 1 to a 0 by clippi against the line $y_{min} = 1$. The resulting intersection point is $I_4\left(2\frac{1}{3}, 1\right)$ and its outcode is 00 we clip $\overline{GI_4}$ and now $\overline{I_4H}$. But segment $\overline{I_4H}$ is not displayed since (0010) AND (0010) 0010 (which is not 0000).

EXAMPLE 9.6.

Suppose that in an implementation of the Cohen-Sutherland algorithm we choose boundar lines in the top-bottom-right left order to clip a line in category 3, draw a picture to sho a worst case scenario, *i.e.*, one that involves the highest number of iterations.

Solution:

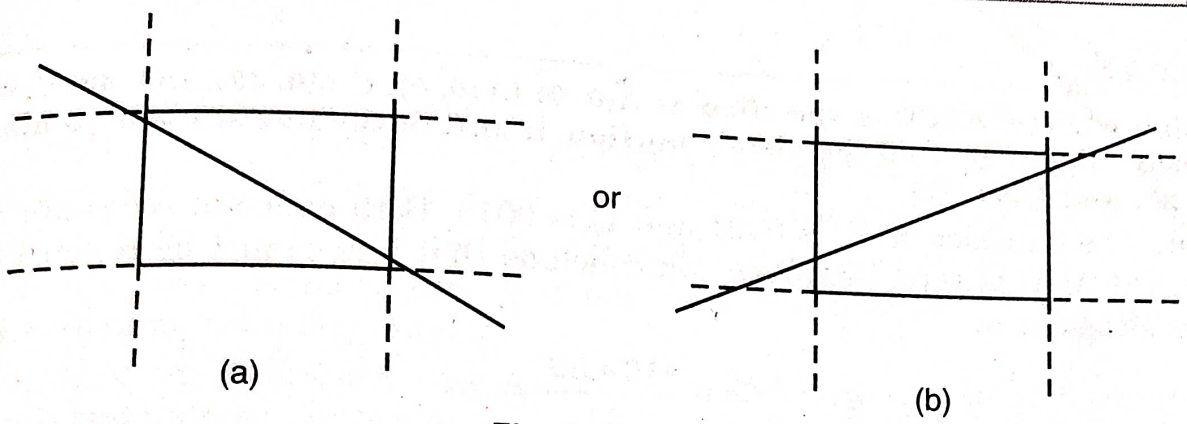
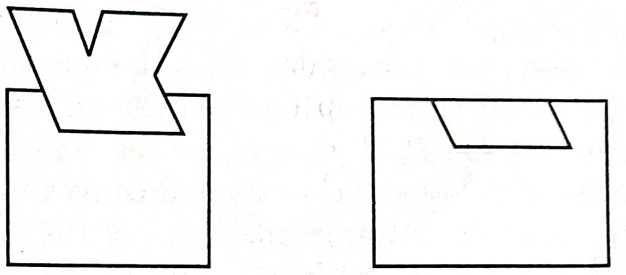


Fig. 9.14.

9.7. Polygon Clipping

Line clipping algorithm connot be used on a polygon because we must generate new edges along the window boundaries as well as clip the original edges. For example.

A polygon is called **convex** if the line joining any two interior points of the polygon lies completely inside the polygon. A non-convex polygon is said to be **concave.**



(a) Before clipping (b) After clipping

Fig. 9.21.

A polygon with vertices $P_1, P_2 \ldots\ldots P_N$ is said to be **positively oriented**, if a tour of the vertices in the given order produces a **counterclockwise circuit.**

Let $A(x_1, y_1)$ and $B(x_2, x_2)$ be the endpoints of a directed line segment A point $P(x, y)$ will be to the left of the line segment if the expression

$$C = (x_2 - x_1)\ (y - y_1) - (y_2 - y_1)\ (x - x_1) \text{ is positive.}$$

We say that the point is to the right of the line segment if this quantity is negative. If a point P is to the right of any one edge of a positively oriented, convex polygon, it is outside the polygon. If it is to the left of every edge of the polygon, it is inside the polygon.

A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments. For example display of a polygon processed by a line-clipping algorithm is as follows:

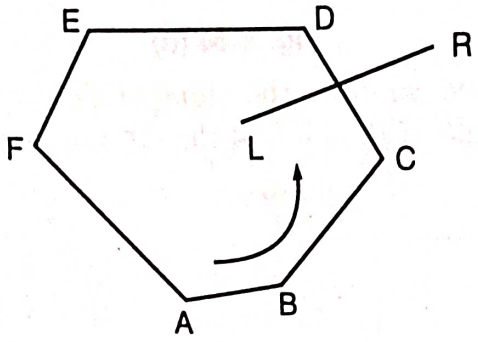


Fig. 9.22.

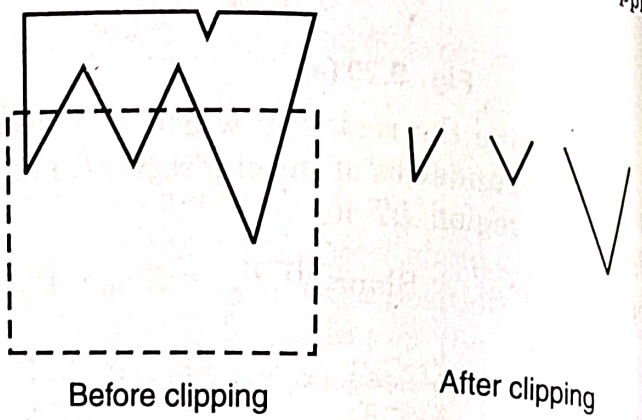


Fig. 9.23.

For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill. Thus, the output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries.

In polygon clipping, each edge of the polygon must be tested against each edge of the clip rectangle; new edges must be added and existing edges must be discarded, retained or divided. Multiple polygons may result from clipping a single polygon. We need an organized way to deal with all these cases.

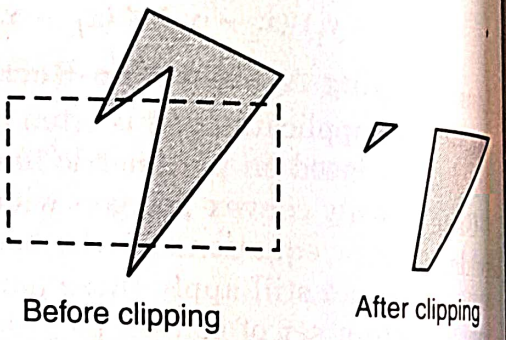


Fig. 9.24. Display of a Correctly Clipped Polygon

9.7.1. The Sutherland-Hodgman Polygon-Clipping Algorithm *(UPTU 2004)*

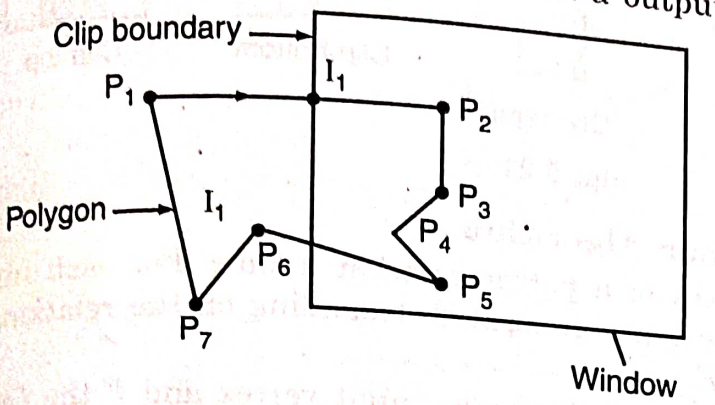
Sutherland and Hodgman's polygon-clipping algorithm uses a *divide-and-conquer strategy*. It solves a series of simple and identical problems that, when combined, solve the overall problem.

Note the difference between this strategy for a polygon and the Cohen-Sutherland algorithm for clipping a line: The polygon clipper clips against four edges in succession, whereas the line clipper tests the outcodes to see which edge is crossed and clips only when necessary. In Sutherland-Hodgeman algorithm a polygon consists of an ordered sequence of vertices. Let $P_1, P_2 \ldots\ldots P_N$ be the vertex list of the polygon to be clipped. Let edge E be any edge of the positively oriented convex clipping polygon. The edges will be processed by the clipping algorithm in the order of the vertix pairs.

The clipping algorithm will be called once for each vertex of the polygon. For each call, the algorithm will return either no vertex at all, the original vertex without change or one or more vertices.

When we are clipping a polygon with respect to any particular edge of the window at that time we have to consider following four different cases:

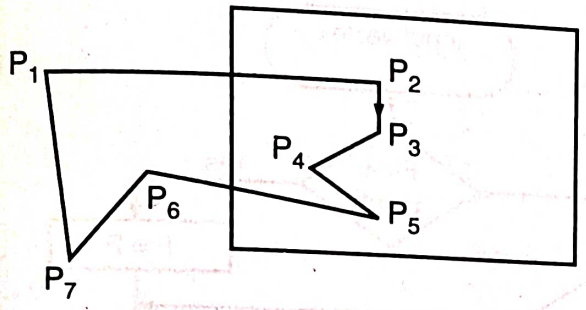
Case-1: If the first vertex is outside the window boundary and the second vertex is inside the window; then the intersection point of polygon with boundary edge of window and the vertex which is inside the window is stored in a output vertex list *i.e.*, P_1P_2 edge.



Store I_1 and P_2 in output vertex list.

Fig. 9.25.

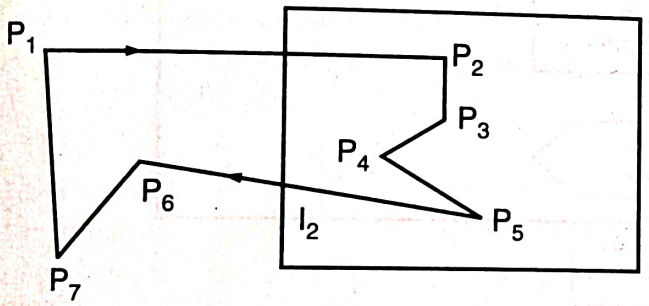
Case-2: If both *i.e.* first and second vertex are inside the window boundary then we have to store the second vertex only in output vertex list, *i.e.*, P_2P_3 edge.



Store only P_3 in output vertex list.

Fig. 9.26.

Case-3: If the first vertex is inside the window and second vertex is outside the window boundary then we have to store only intersection point in output vertex list *i.e.*, P_5P_6 edge.



Store only I_2 vertex in output vertex list.

Fig. 9.27.

Case-4: If both first and second vertex of a polygon are lieing outside the window boundary then no vertex is stored in output vertex list *i.e.*, P_6P_7 edge nothing is stored in output vertex list.

Once all vertices have been considered for one clip window boundary, the output list of vertices is clipped against the next window boundary.

The block diagram for this algorithm is as follows:

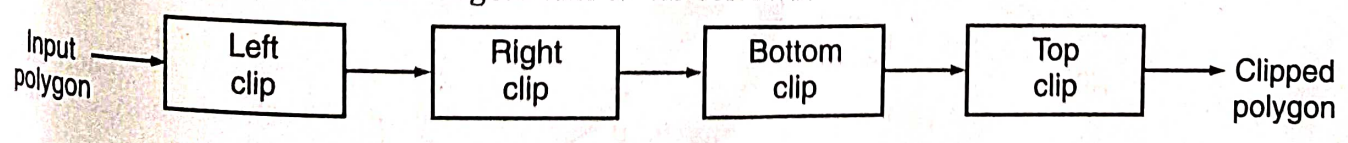


Fig. 9.28.

For example

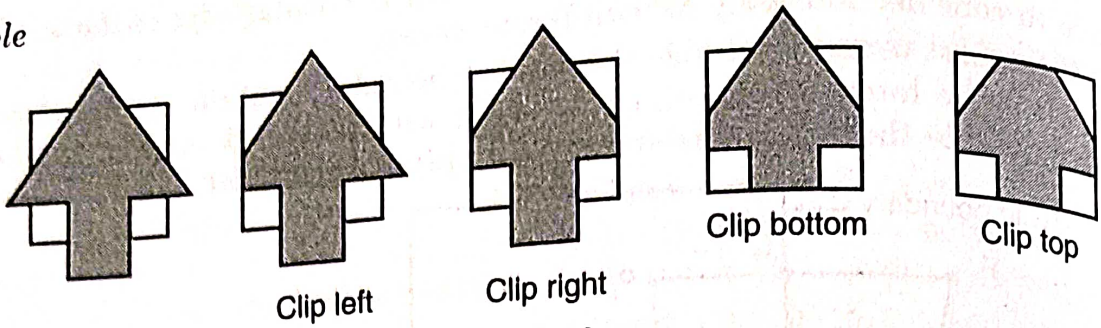


Fig. 9.29.

Flow-chart of Sutherland-Hodgman Algorithm

This algorithm inputs the vertices of a polygon one at a time. For each input vertex, either zero, one or two output vertices are generated depending on the relationship of the input vertices to the clipping edge E.

We denote P as the input vertex, S the previous input vertex and F the first arriving input vertex. The vertex or vertices to be output are determined according to the flow-chart below:

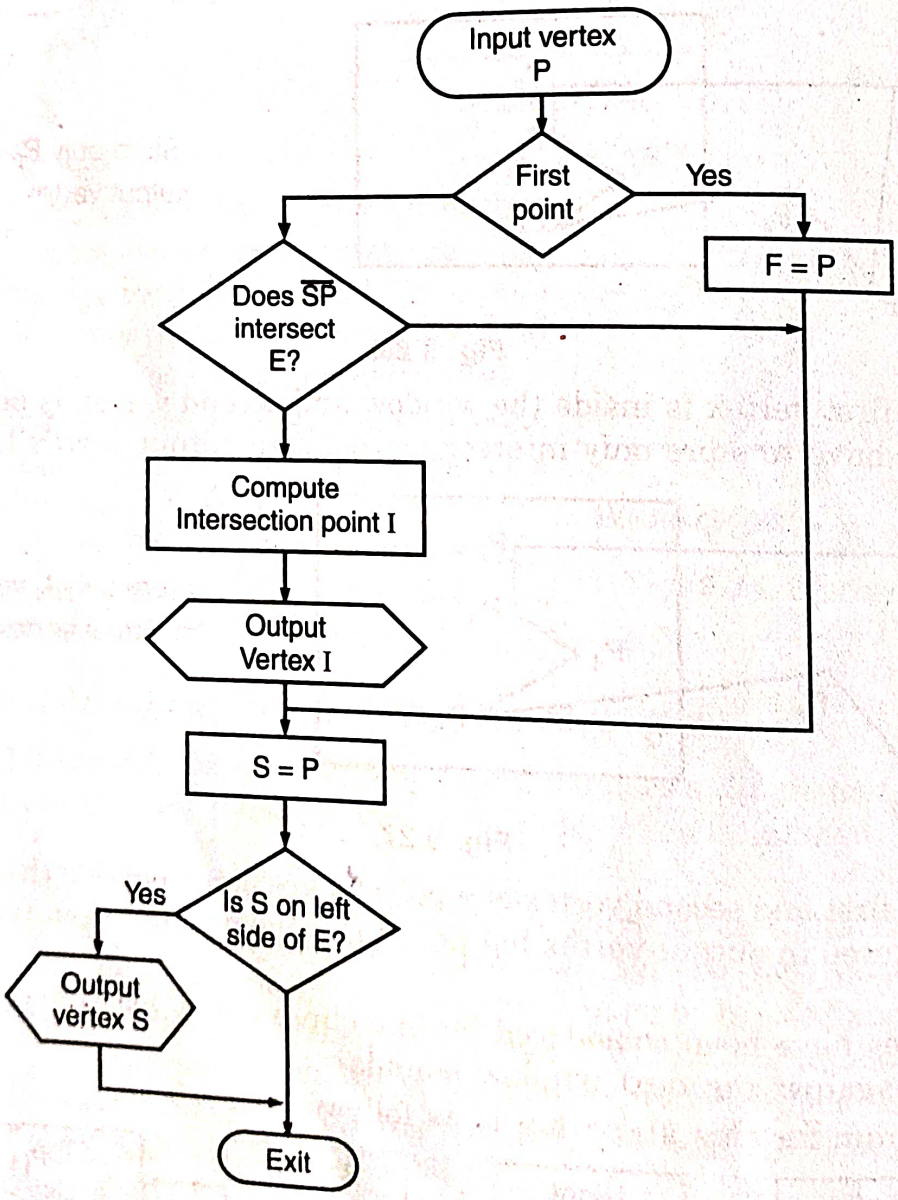


Fig. 9.30.

If the polygon has n edges then the edge $\overline{P_n P_1}$ is closing the polygon. In order to avoid the need to duplicate the input of P_1 as the final input vertex the closing logic shown in the flow chart below is called after processing the final input vertex P_n.

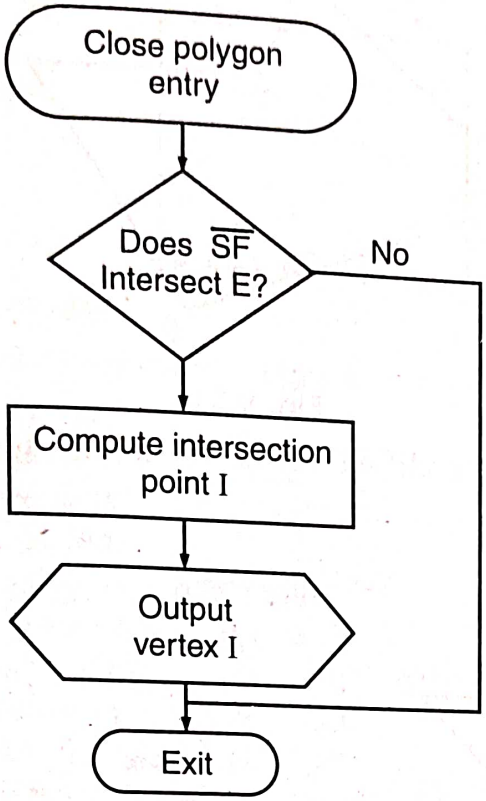


Fig. 9.31.

Example:

Let us take an example to understand the polygon clipping by Sutherland Hodgman algorithm. Suppose we are having polygon. (See figure 9.32). *ABCDE* which we want to clip against a rectangular window.

Here vertex list will be *A*, *B*, *C*, *D*, *E* and edges will be *AB*, *BC*, *CD*, *DE* and *EA*.

Step 1. Clip left: We will consider all edges of polygon with respect to left boundary of window. Diagrammatic representation of left clipped polygon is

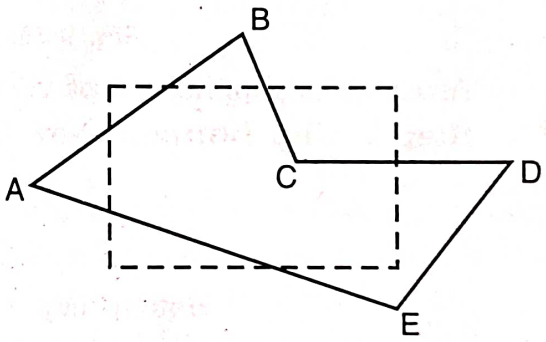


Fig. 9.32.

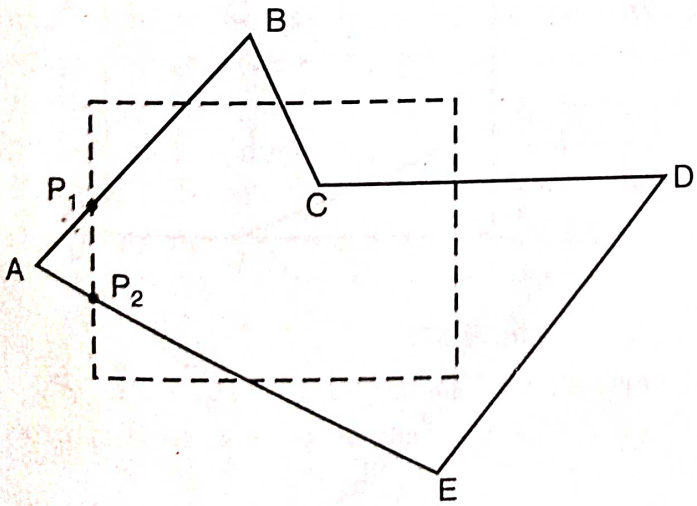


Fig. 9.33.

Left clip

$AB \rightarrow P_1 B$

$BC \rightarrow C$

$CD \rightarrow D$

$DE \rightarrow E$

$EA \rightarrow P_2$

So after left clipping our output vertex list will become
$\{P_1, B, C, D, E, P_2\}$ *i.e.*

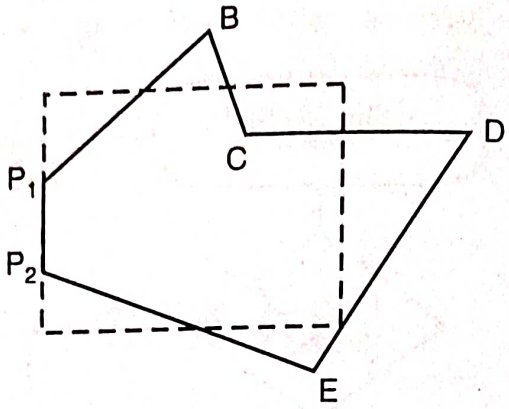
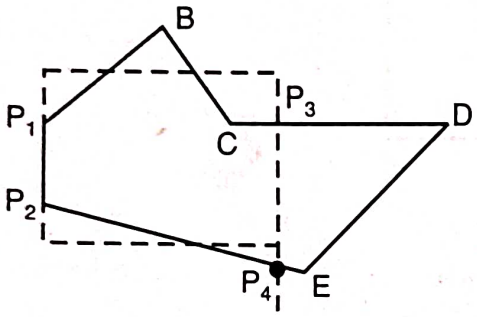


Fig. 9.34.

Step 2. Clip Right: Now modified list of vertices is passed to clip right procedure.



Right clip

$P_1 B \rightarrow B$
$BC \rightarrow C$
$CD \rightarrow P_3$
$DE \rightarrow$ No vertex
$EP_2 \rightarrow P_4, P_2$
$P_2 P_1 \rightarrow P_1$

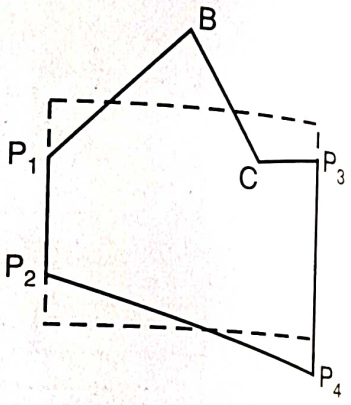


Fig. 9.35. **Fig. 9.36.**

After right clipping set of vertices will be $\{B, C, P_3, P_4, P_2, P_1\}$

Step-3. Clip Bottom: Now modified list of vertices is passed to this procedure.

Bottom clip

$P_2 P_1 \rightarrow P_1$
$P_1 B \rightarrow B$
$BC \rightarrow C$
$CP_3 \rightarrow P_3$
$P_3 P_4 \rightarrow I_1$
$P_4 P_2 \rightarrow I_2, P_2$

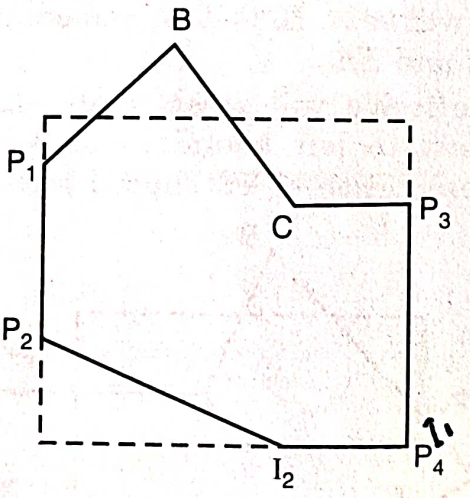


Fig. 9.37.

After bottom clipping set of vertices will be $\{P_1, B, C, P_3, I_1, I_2, P_2\}$

Step 4: Clip top: The modified list of vertices is passed to this procedure.

Top clip

$P_1 B \rightarrow I_3$
$BC \rightarrow I_4, C$
$CP_3 \rightarrow P_3$
$P_3 I_1 \rightarrow I_1$
$I_1 I_2 \rightarrow I_2$
$I_2 P_2 \rightarrow P_2$
$P_2 P_1 \rightarrow P_1$

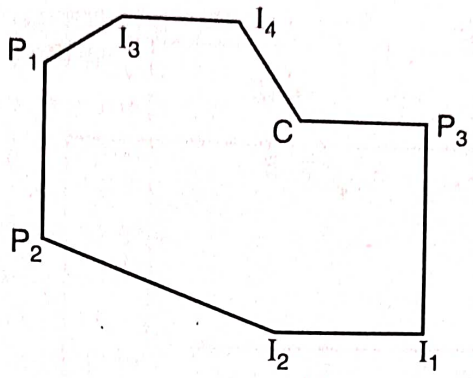


Fig. 9.38.

This is the final clipped polygon.

Limitations with Sutherland-Hodgeman Algorithm

All convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm, but concave polygons may be displayed with **extraneous lines.** This occurs when the clipped polygon should have two or more separate sections. But since there is only one output vertex list, the last vertex in the list is always joined to the first vertex *i.e.* we are forming an edge between last and first vertex. For example:

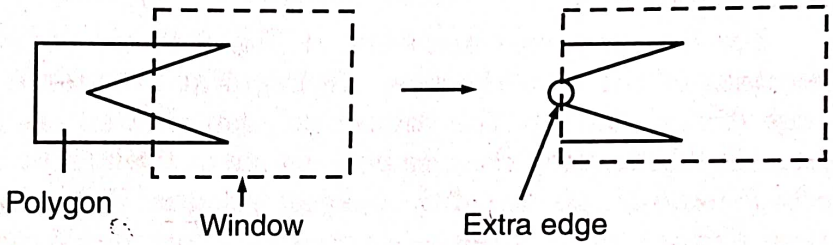


Fig. 9.39.

To overcome this problem one way is to split the concave polygon into two or more convex polygons and process each eonvex polygon separately. Another possibility is to modify the Sutherland-Hodgeman approach to check the final vertex list for multiple vertex points along any clip window boundary and correctly join pairs of vertices. Finally, we could use a more general polygon clipping algorithm such as. Weiler-Atherton polygon clipping algorithm.