UNIT 3:

Clipping and Filling Rectangles and Polygons

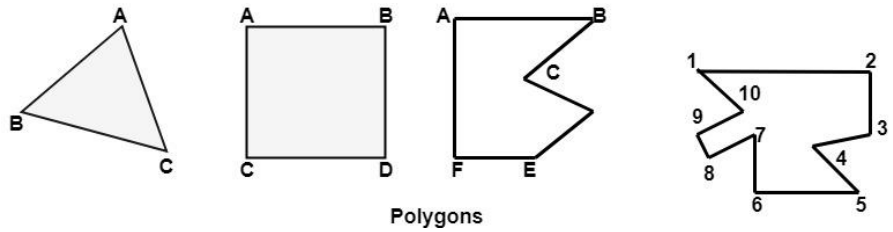
Polygon:

Polygon is a representation of the surface. It is primitive which is closed in nature. It is formed using a collection of lines. It is also called as many-sided figure. The lines combined to form polygon are called sides or edges. The lines are obtained by combining two vertices.

Example of Polygon:

1. Triangle
2. Rectangle
3. Hexagon
4. Pentagon

Following figures shows some polygons.



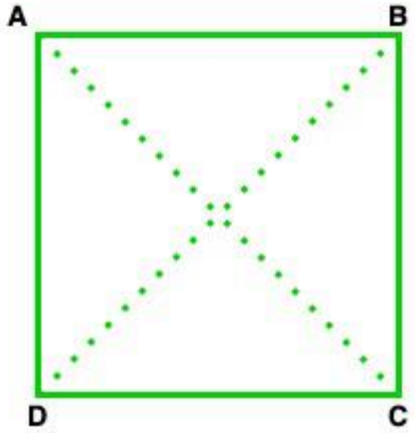
Polygons

Properties of Polygon:

- The measure of each exterior angle of an n-sided regular polygon will be 360°/n.
- The measure of each interior angle of n-sided regular polygon will be [(n – 2) × 180°]/n.
- The number of triangles formed by joining the diagonals from one corner of a polygon will be n – 2.
- The number of diagonals in a polygon with n sides will be n(n – 3)/2.
- The sum of all the interior angles of an n-sided polygon will be (n – 2) × 180°.

Terminology:

1. Diagonals: A line segment joining two non-consecutive vertices of a polygon is known as a diagonal. For example, in the given figure, AC and BD are the two diagonals of the ABCD square.



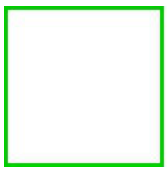
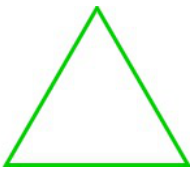
2. Adjacent Sides: In a polygon, if two sides share a common vertex then such type of sides is known as adjacent sides. For example, in the above figure, AD and DC are the adjacent sides.

3. Adjacent Vertex: In a polygon, if two endpoints or vertex of the same side, then such type of vertex is known as the adjacent vertex. For example, in the above figure vertex, A and B are the adjacent vertex of side AB.

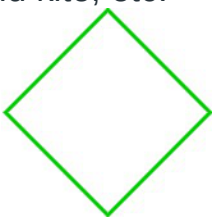
Types of Polygons

There are 4 types of Polygon :

- **Regular Polygon:** If all the sides and interior angles of the polygon are equal or if a polygon is equiangular and equilateral, then the polygon will be known as a regular polygon. Example square, rhombus, equilateral triangle, etc.

- **Irregular Polygon:** If all the sides and the interior angles of the polygon are of different measure, then the polygon will be known as an irregular polygon. Example scalene triangle, rectangle, and kite, etc.

- **Convex Polygon:** If all the interior angles of a polygon are strictly less than 180° or if a line segment between two points on the boundary does not go outside the polygon, then the polygon will be known as a convex polygon.


- **Concave Polygon:** If one or more interior angles of a polygon are more than 180° degrees or a polygon contains at least one reflex interior angle, then the polygon will be known as a concave polygon. This polygon can have at least four sides.

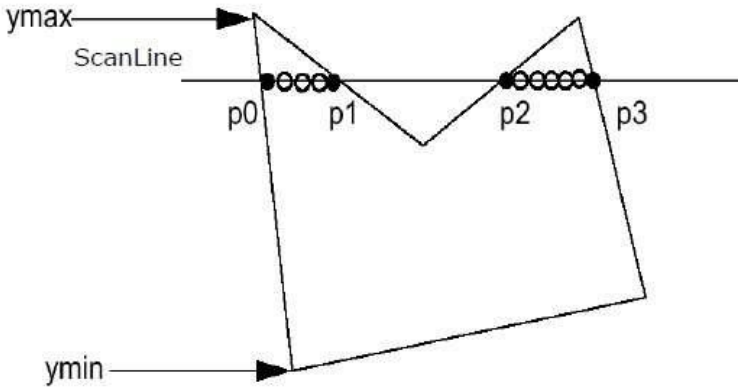

Polygon Filling Algorithm

Polygon is an ordered list of vertices as shown in the following figure. For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon. In this chapter, we will see how we can fill polygons using different techniques.

Scan Line Algorithm

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections. The following steps depict how this algorithm works.

Step 1 − Find out the Ymin and Ymax from the given polygon.



Step 2 − ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

Step 3 − Sort the intersection point in the increasing order of X coordinate i.e. p0,p10,1, p1,p21,2, and p2,p32,3.

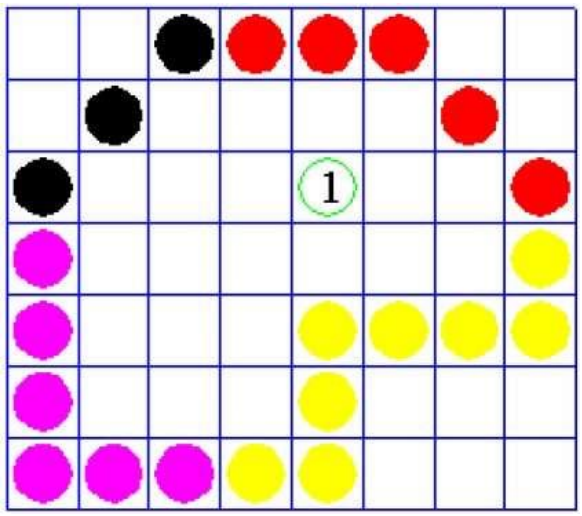
Step 4 − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

Flood Fill Algorithm

Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm.

Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.



Boundary Fill Algorithm

The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill should be different for this algorithm to work.

In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

Seed fill

The seed is a point known to be inside the polygon
and highlight outward from this
point (i.e.) neighbouring pixels until the boundary
pixel is encountered. This approach is called seed ûll
because colour üows from the seed pixel until
reaching the polygon boundary.

Line Clipping:

It is performed by using the line clipping algorithm. The line clipping algorithms are:

1. Cohen Sutherland Line Clipping Algorithm
2. Midpoint Subdivision Line Clipping Algorithm
3. Liang-Barsky Line Clipping Algorithm

Cohen Sutherland Line Clipping Algorithm:

In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

1. Visible
2. Not Visible
3. Clipping Case

1. Visible: If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.

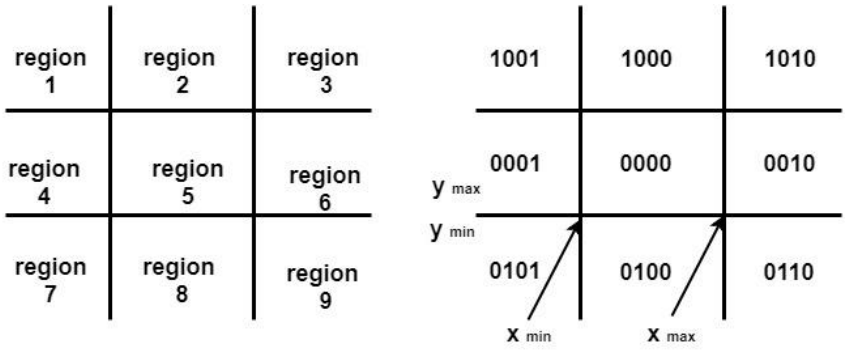
2. Not Visible: If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible. Let A (x_1,y_2) and B (x_2,y_2) are endpoints of line.

x_{min},x_{max} are coordinates of the window.

y_{min},y_{max} are also coordinates of the window.

$x_1 > x_{max}$
$x_2 > x_{max}$
$y_1 > y_{max}$
$y_2 > y_{max}$
$x_1 < x_{min}$
$x_2 < x_{min}$
$y_1 < y_{min}$
$y_2 < y_{min}$

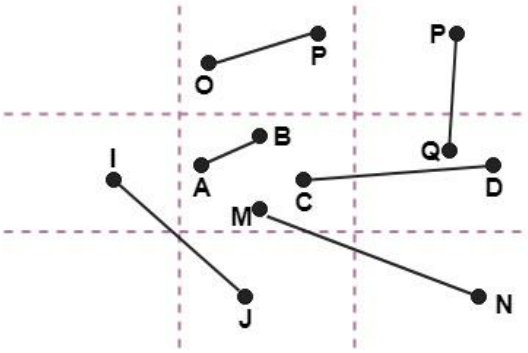
3. Clipping Case: If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible.



9 region bits assigned to 9 regions

The center area is having the code, 0000, i.e., region 5 is considered a rectangle window.

Following figure show lines of various types



Line AB is the visible case
Line OP is an invisible case
Line PQ is an invisible line
Line IJ are clipping candidates
Line MN are clipping candidate
Line CD are clipping candidate

Advantage of Cohen Sutherland Line Clipping:

1. It calculates end-points very quickly and rejects and accepts lines quickly.
2. It can clip pictures much large than screen size.

Algorithm of Cohen Sutherland Line Clipping:

Step1: Calculate positions of both endpoints of the line

Step2: Perform OR operation on both of these end-points

Step3: If the OR operation gives 0000
Then
line is considered to be visible
else
Perform AND operation on both endpoints
If And ≠ 0000
then the line is invisible
else
And=0000
Line is considered the clipped case.

Step4: If a line is clipped case, find an intersection with boundaries of the window
$m=(y_2-y_1)(x_2-x_1)$

(a) If bit 1 is "1" line intersects with left boundary of rectangle window
$y_3=y_1+m(x-X_1)$
where $X = X_{wmin}$
where $X_{wmin}$is the minimum value of X co-ordinate of window

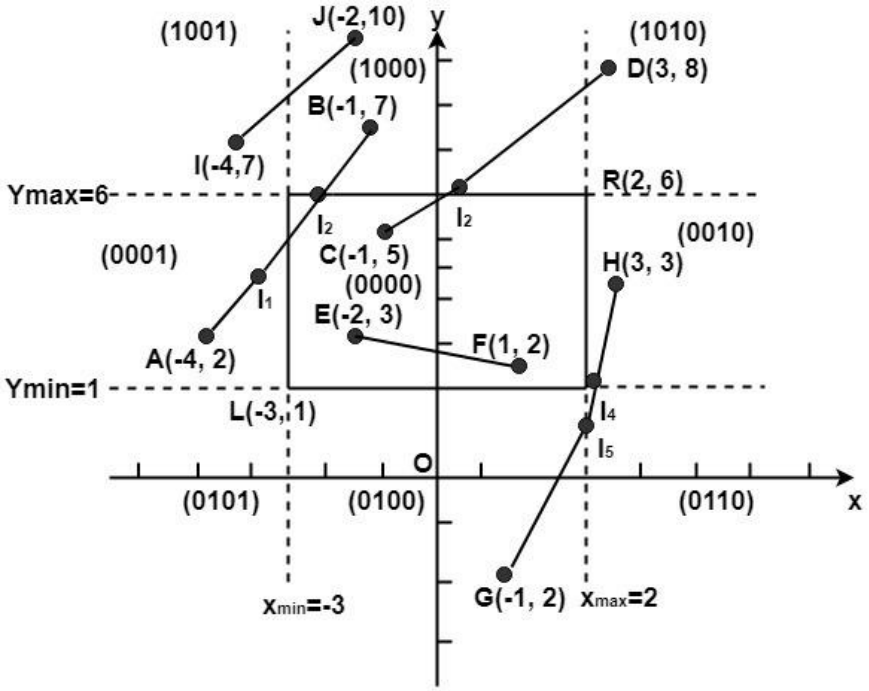
(b) If bit 2 is "1" line intersect with right boundary
$y_3=y_1+m(X-X_1)$
where $X = X_{wmax}$
where X more is maximum value of X co-ordinate of the window

(c) If bit 3 is "1" line intersects with bottom boundary
$X_3=X_1+(y-y_1)/m$
where $y = y_{wmin}$
y_{wmin} is the minimum value of Y co-ordinate of the window

(d) If bit 4 is "1" line intersects with the top boundary
$X_{3=X}1+(y-y_1)/m$
where $y = y_{wmax}$
y_{wmax} is the maximum value of Y co-ordinate of the window

Example of Cohen-Sutherland Line Clipping Algorithm:

Let R be the rectangular window whose lower left-hand corner is at L (-3, 1) and upper right-hand corner is at R (2, 6). Find the region codes for the endpoints in fig:



The region code for point (x, y) is set according to the scheme

Bit 1 = sign $(y-y_{max})$=sign (y-6) Bit 3 = sign $(x-x_{max})$= sign (x-2)

Bit 2 = sign $(y_{min}-y)$=sign(1-y) Bit 4 = sign $(x_{min}-x)$=sign(-3-x)

Here

$$\text{sign}(a) = \begin{cases} 1 & \text{if a is positive} \\ 0 & \text{otherwise} \end{cases}$$

So

A (-4, 2)→ 0001 F (1, 2)→ 0000

B (-1, 7) → 1000 G (1, -2) →0100

C (-1, 5)→ 0000 H (3, 3) → 0100

D (3, 8) → 1010 I (-4, 7) → 1001
E (-2, 3) → 0000 J (-2, 10) → 1000

We place the line segments in their appropriate categories by testing the region codes found in the problem.

Category1 (visible): EF since the region code for both endpoints is 0000.

Category2 (not visible): IJ since (1001) AND (1000) =1000 (which is not 0000).

Category 3 (candidate for clipping): AB since (0001) AND (1000) = 0000, CD since (0000) AND (1010) =0000, and GH. since (0100) AND (0010) =0000.

The candidates for clipping are AB, CD, and GH.

In clipping AB, the code for A is 0001. To push the 1 to 0, we clip against the boundary line x_{min}=-3. The resulting intersection point is I_1 (-3,3 $\frac{2}{3}$). We clip (do not display) AI_1 and I_1 B. The code for $I_1$is 1001. The clipping category for I_1 B is 3 since (0000) AND (1000) is (0000). Now B is outside the window (i.e., its code is 1000), so we push the 1 to a 0 by clipping against the line y_{max}=6. The resulting intersection is I_2 (-1 $\frac{3}{5}$,6). Thus I_2 B is clipped. The code for I_2 is 0000. The remaining segment I_1 I_2 is displayed since both endpoints lie in the window (i.e., their codes are 0000).

For clipping CD, we start with D since it is outside the window. Its code is 1010. We push the first 1 to a 0 by clipping against the line y_{max}=6. The resulting intersection I_3 is ($\frac{3}{5}$,6),and its code is 0000. Thus I_3 D is clipped and the remaining segment CI_3 has both endpoints coded 0000 and so it is displayed.

For clipping GH, we can start with either G or H since both are outside the window. The code for G is 0100, and we push the 1 to a 0 by clipping against the line y_{min}=1.The resulting intersection point is I_4 (2 $\frac{1}{5}$,1) and its code is 0010. We clip GI_4 and work on I_4 H. Segment I_4 H is not displaying since (0010) AND (0010) =0010.

Generalized Clipping with Cyrus-beck Algorithm

Background:
Cyrus Beck is a line clipping algorithm that is made for convex polygons. It allows line clipping for non-rectangular windows, unlike Cohen Sutherland or **Nicholl Le Nicholl**. It also removes the repeated clipping needed in Cohen Sutherland.

Input:

1. **Convex area of interest**
   which is defined by a set of coordinates
   given in a clockwise fashion.
2. **vertices** which are an array of coordinates:
   consisting of pairs (x, y)
3. **n** which is the number of vertices
4. A **line** to be clipped
   given by a set of coordinates.
5. **line** which is an array of coordinates:
   consisting of two pairs, (x0, y0) and (x1, y1)

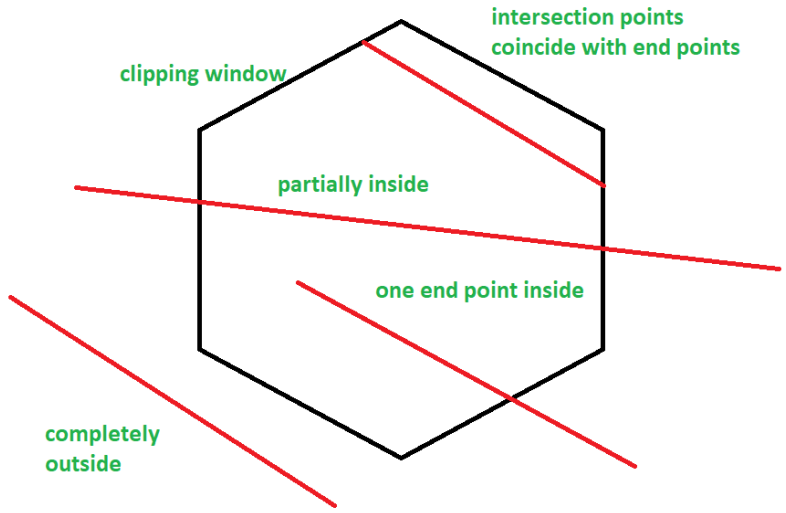
Output:

1. Coordinates of line clipping which is the **Accepted clipping**
2. Coordinates (-1, -1) which is the **Rejected clipping**

Algorithm:

- Normals of every edge is calculated.
- Vector for the clipping line is calculated.
- Dot product between the difference of one vertex per edge and one selected end point of the clipping line and the normal of the edge is calculated (for all edges).
- Dot product between the vector of the clipping line and the normal of edge (for all edges) is calculated.
- The former dot product is divided by the latter dot product and multiplied by -1. This is ‘t’.
- The values of ‘t’ are classified as entering or exiting (from all edges) by observing their denominators (latter dot product).
- One value of ‘t’ is chosen from each group, and put into the parametric form of a line to calculate the coordinates.
- If the entering ‘t’ value is greater than the exiting ‘t’ value, then the clipping line is rejected.

Cases:



1. **Case 1:** The **line is partially inside** the clipping window:
2. $0 < t_E < t_L < 1$
3.
4. where t_E is 't' value for entering intersection point
5. t_L is 't' value for exiting intersection point
6. **Case 2:** The **line has one point inside or both sides inside** the window **or the intersection points are on the end points of the line**:
   $0 \leq t_E \leq t_L \leq 1$
7. **Case 3:** The line is **completely outside** the window:
   $t_L < t_E$

Pseudocode:

First, calculate the parametric form of the line to be clipped and then follow the algorithm.

- Choose a point called P_1 from the two points of the line (P_0P_1).
- Now for each edge of the polygon, calculate the normal pointing away from the centre of the polygon, namely N_1, N_2, etc.
- Now for each edge choose P_{Ei} (i -> i^{th} edge) (choose any of the vertices of the corresponding edge, eg.: For polygon ABCD, for side AB, P_{Ei} can be either point A or point B) and calculate
  $P_0 - P_{Ei}$
- Then calculate
  $P_1 - P_0$
- Then calculate the following dot products for each edge:
  $N_i . (P_0 - P_{Ei})$
  $N_i . (P_1 - P_0)$

where i -> i^{th} edge of the convex polygon

- Then calculate the corresponding 't' values for each edge by:
  $N_i . (P_0 - P_{Ei})$

$$t = \frac{}{-(N_i \,.\, (P_1 - P_0))}$$

- Then club the 't' values for which the $\mathbf{N_i \,.\, (P_1 - P_0)}$ came out to be negative and take the minimum of all of them and 1.
- Similarly club all the 't' values for which the $N_i \,.\, (P_1 - P_0)$ came out to be positive and take the maximum of all of the clubbed 't' values and 0.
- Now the two 't' values obtained from this algorithm are plugged into the parametric form of the 'to be clipped' line and the resulting two points obtained are the clipped points.

Liang-Barsky Algorithm

The **Liang-Barsky algorithm** is a line clipping algorithm. This algorithm is more efficient than Cohen–Sutherland line clipping algorithm and can be extended to 3-Dimensional clipping. This algorithm is considered to be the faster parametric line-clipping algorithm. The following concepts are used in this clipping:

1. The parametric equation of the line.
2. The inequalities describing the range of the clipping window which is used to determine the intersections between the line and the clip window.

The parametric equation of a line can be given by,

$$X = x_1 + t(x_2 - x_1)$$
$$Y = y_1 + t(y_2 - y_1)$$

Where, t is between 0 and 1.

Then, writing the point-clipping conditions in the parametric form:

$$xw_{min} <= x_1 + t(x_2 - x_1) <= xw_{max}$$
$$yw_{min} <= y_1 + t(y_2 - y_1) <= yw_{max}$$

The above 4 inequalities can be expressed as,

$$tp_k <= q_k$$

Where k = 1, 2, 3, 4 (correspond to the left, right, bottom, and top boundaries, respectively).

The p and q are defined as,

$$p_1 = -(x_2 - x_1), \quad q_1 = x_1 - xw_{min} \text{ (Left Boundary)}$$
$$p_2 = (x_2 - x_1), \quad q_2 = xw_{max} - x_1 \text{ (Right Boundary)}$$
$$p_3 = -(y_2 - y_1), \quad q_3 = y_1 - yw_{min} \text{ (Bottom Boundary)}$$
$$p_4 = (y_2 - y_1), \quad q_4 = yw_{max} - y_1 \text{ (Top Boundary)}$$

When the line is parallel to a view window boundary, the p value for that boundary is zero.
When $p_k < 0$, as t increase line goes from the outside to inside (entering).
When $p_k > 0$, line goes from inside to outside (exiting).
When $p_k = 0$ and $q_k < 0$ then line is trivially invisible because it is outside view window.

When $p_k = 0$ and $q_k > 0$ then the line is inside the corresponding window boundary.
Using the following conditions, the position of line can be determined:

Condition	Position of line
$p_k = 0$	parallel to the clipping boundaries
$p_k = 0$ and $q_k < 0$	completely outside the boundary
$p_k = 0$ and $q_k >= 0$	inside the parallel clipping boundary
$p_k < 0$	line proceeds from outside to inside
$p_k > 0$	line proceeds from inside to outside

Parameters t_1 and t_2 can be calculated that define the part of line that lies within the clip rectangle.
When,
1. $p_k < 0$, maximum(0, q_k/p_k) is taken.
2. $p_k > 0$, minimum(1, q_k/p_k) is taken.

If $t_1 > t_2$, the line is completely outside the clip window and it can be rejected.
Otherwise, the endpoints of the clipped line are calculated from the two values of parameter t.

Algorithm –

1. Set $t_{min}=0$, $t_{max}=1$.
2. Calculate the values of t (t(left), t(right), t(top), t(bottom)),
   (i) If $t < t_{min}$ ignore that and move to the next edge.
   (ii) else separate the t values as entering or exiting values using the inner product.
   (iii) If t is entering value, set $t_{min} = t$; if t is existing value, set $t_{max} = t$.
3. If $t_{min} < t_{max}$, draw a line from $(x_1 + t_{min}(x_2-x_1), y_1 + t_{min}(y_2-y_1))$ to $(x_1 + t_{max}(x_2-x_1), y_1 + t_{max}(y_2-y_1))$
4. If the line crosses over the window, $(x_1 + t_{min}(x_2-x_1), y_1 + t_{min}(y_2-y_1))$ and $(x_1 + t_{max}(x_2-x_1), y_1 + t_{max}(y_2-y_1))$ are the intersection point of line and edge.

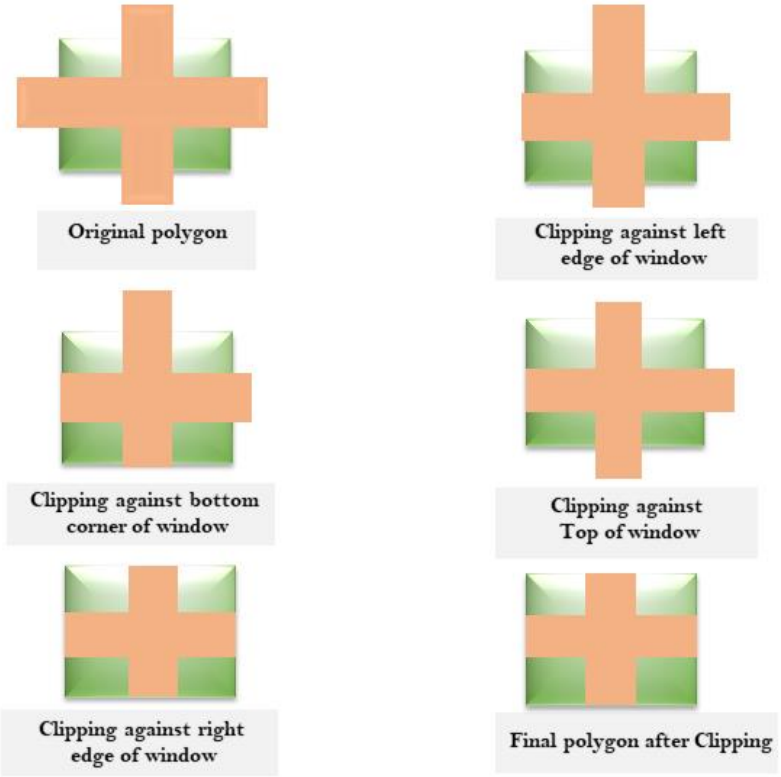
Sutherland-Hodgeman Polygon Clipping:

It is performed by processing the boundary of polygon against each window corner or edge. First of all entire polygon is clipped against one edge, then resulting polygon is considered, then the polygon is considered against the second edge, so on for all four edges.

Four possible situations while processing

1. If the first vertex is an outside the window, the second vertex is inside the window. Then second vertex is added to the output list. The point of intersection of window boundary and polygon side (edge) is also added to the output line.
2. If both vertexes are inside window boundary. Then only second vertex is added to the output list.
3. If the first vertex is inside the window and second is an outside window. The edge which intersects with window is added to output list.
4. If both vertices are the outside window, then nothing is added to output list.

Following figures shows original polygon and clipping of polygon against four windows.



Disadvantage of Cohen Hodgmen Algorithm:

This method requires a considerable amount of memory. The first of all polygons are stored in original form. Then clipping against left edge done and output is stored. Then clipping against right edge done, then top edge. Finally, the bottom edge is clipped. Results of all these operations are stored in memory. So wastage of memory for storing intermediate polygons.

Sutherland Hodgemen Algorithm: