3.1. POLYGON REPRESENTATION

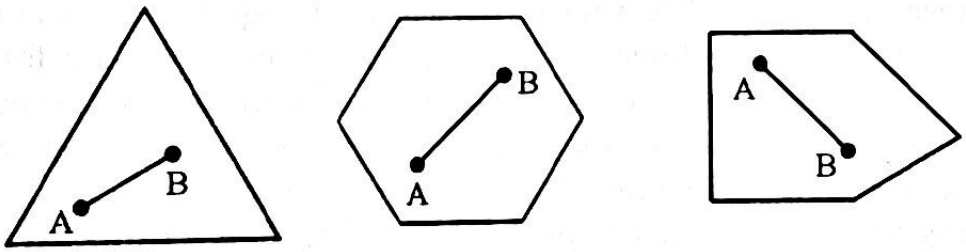
A many-sided-figure is termed as a polygon. Any polygon can be represented by using various line segments which are connected end to end or we can say that any polygon can be represented by series of points which make line segments which are connected and these line segments are called sides of polygon or edges and the end points are called vertices. The simplest polygon is a triangle which have minimum line segments i.e. three.

There are two types of polygons

(i) Convex Polygon

(ii) Concave Polygon

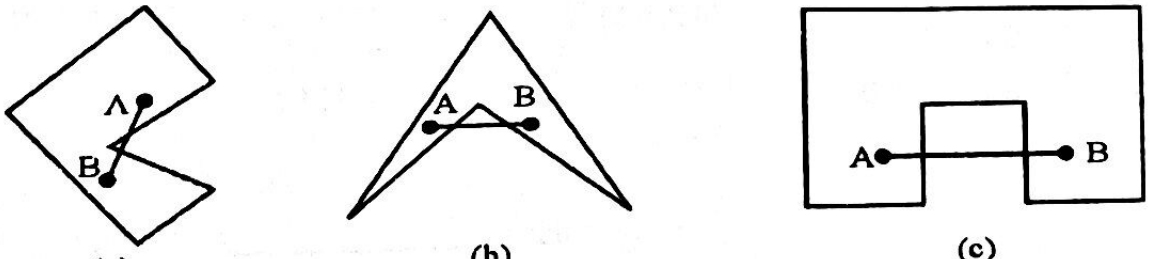
(i) Convex Polygon : If the line connecting two interior points of the polygon lies completely inside the polygon, is said to be convex as shown in the figure 3.1.



(a) (b) (c)

Fig. 3.1. Convex Polygon.

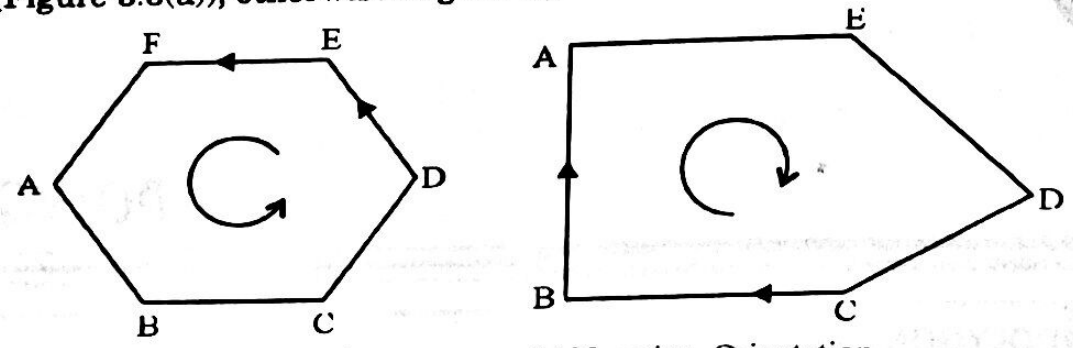
(ii) Concave Polygon : The polygon that are not convex, are called concave polygon *i.e.*, line joining any two interior point of the polygon is not completely inside the polygon as shown in the following figure 3.2.



(a) (b) (c)

Fig. 3.2. Concave Polygon

Note : A polygon having vertices $P_1, P_2, \ldots P_N$, is said to be positively oriented if a tour of the vertices in the given order produces an anticlockwise loop (Figure 3.3(a)), otherwise negatively oriented (Figure 3.3(b)).



(a) Positive Orientation (b) Negative Orientation

Fig. 3.3.

There are various method to represent polygons. Some graphical devices supply a polygon drawing primitive to directly image polygon shapes and on these devices polygon can be saved as a unit. And some devices provide a trapezoid primitive. Trapezoids can be formed by two scan lines and two line segments and can be drawn by stepping down the line segments with two vector generators and, for each scan line, filling in all the pixels between them. Every polygon can be broken up into trapezoids and we can say any polygon can be represented by the series of trapezoids and polygon representation is better as a unit.

We can store our polygons also in display file by just using the line commands for each of the edges into the display file, but there are two problems with this technique. First, there is no way to distinguish which line commands out of all those in display file should be grouped together to form polygon display unit and second, we have not correctly positioned the pen for drawing the initial sides. We can overcome these problems by prefacing the commands for drawing the polygon sides by a new commands. This new command describes that how many sides are in the polygon, so we will know how many of the following line commands are part of the polygon and this new command will act like a move to correctly position the pen for drawing the first side. We

DF–OP	DF–X	DF–Y
5	0.2	0.2
2	0.4	0.2
2	0.6	0.5
2	0.4	0.7
2	0.2	0.5
2	0.2	0.2

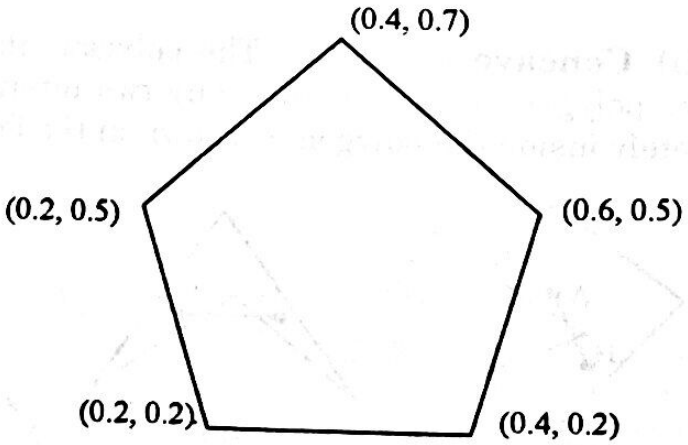


Figure 3.4. Polygon Represented by display file structure.

have used operation codes 1 and 2 till now and now we will use the operation code 3 or greater in order to represent polygon. The value of code will tell the number of sides used in polygon *i.e.*, polygon of sides no more than maximum possible value of operation code can be represented. The X and Y operands of the polygon command will be co-ordinates of the points where the first side to be drawn begins. Because the polygons are closed curves, therefore it will also be the final endpoint of the last side to be drawn. A polygon represented by the display is shown in Fig. 3.4.

3.2. ENTERING POLYGONS

There are two basic informations to specify a polygon and are required to enter polygon into display file *i.e.*, number of sides of polygon and the vertex points. We know that there are two types of primitive operation for line drawing as we have discussed in previous chapter in detail, i.e.

LINE – ABS2 (X, Y) and
LINE – REL2 (DX, DY),
MOVE – ABS2 (X, Y) and
MOVE – REL2 (DX, DY).

In this way there are two ways to implement the Algorithm for entering the polygon into the display file — One for absolute co-ordinates and other for relative co-ordinates, when we use absolute command.

Algo for Absolute Polygon :

1. Input the array containing the vertices of the polygon
   - the number of sides of polygon
   - co-ordinates of current pen position and a variable for stepping through the polygon sides.
2. Check the number of sides that is 3 or greater if less than 5 retum invalid polygon.
3. Enter the polygon instructions i.e. current pen position and number of sides.
4. And finally enter the instructions for the sides and retum.

Algo for Relative Polygon

1. Input the arrays containing the relative offsets of the vertices
   - the number of sides, the current pen position, variable for stepping through the polygon sides and variable for storing the starting point which will be the final point of polygon.
2. Check the number of sides that is 3 or greater, if less than 3 retum invalid polygon.
3. Increment the current pen position.
4. Save the starting point for closing the polygon.
5. Enter the polygon instruction i.e. number of sides
6. Enter the instruction for sides and close the polygon then retum.

3.3. AN INSIDE AND OUTSIDE TEST

Polygons can be represented in two ways – outlined form using move commands and as a solid objects by setting the pixels high inside the polygon including pixels on the boundary. There is an important issue how we can determine whether or not a point is inside of a polygon. There are two methods for the same :

- Even-odd method
- Winding-numbere Method.

3.3.1. Even-Odd Method (Odd Parity Rule)

In this rule, we draw a line segment between the point in question and a point known to be outside the polygon. Just choose a point with an x co-ordinate smaller than the smallest x co-ordinate of the polygon's vertices, then draw a line from any position to the distant point chosen outside the co-ordinate extents of the object and counting the number of edges crossing along the line. If the number of polygon edges crossed by this line is odd, then this point is an interior point. Otherwise, this point is an exterior point. To get the accurate count of edges, we should choose the path that does not intersect the polygon vertices (Figure 3.5).

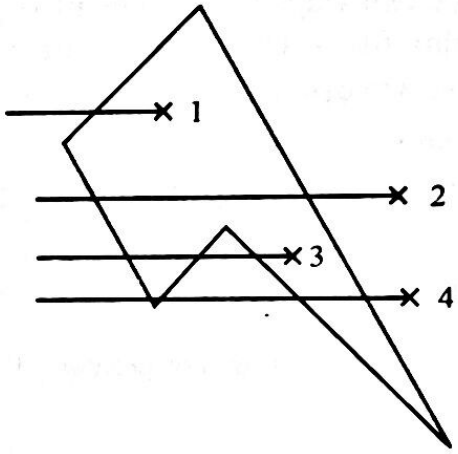


Fig. 3.5. Even-Odd Rule.

In above example, line segment from point '1' crosses single edge, hence point is inside the polygon (Since 1 is an odd number). And line segment from point '2' crosses two edges then the point is outside the polygon. Similarily point '3' is inside (Since intersects three edges) and point '4' is outside (Since intersects four edges).

Note : When line segment intersects the vertex of the polygon, then the counting is considered as follows :

1. Count is even if the other endpoints of the two segments that meet at the intersecting vertex. (Fig. 3.6).

2. Count is odd if both the endpoints lie on opposite sides of the constructed line. (Fig. 3.6).

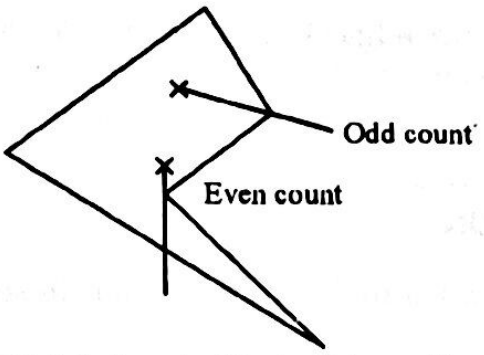


Fig. 3.6. Count of Vertex Intersection.

3.3.2. Winding Number Method or Nonzero Winding Number Rule

There is an alternative method to define interior region, called winding number method. In this method, we count how many number of times the polygon edges wind around a particular point in the counterclockwise direction and we give each boundary line crossed a *direction number*, and we sum these direction numbers. The direction number indicates the direction the polygon edge was drawn relative to the line segment we constructed for the test. For example, to test a point (x_1, y_1). Let us consider a horizontal line segment $y = y_1$ that sums from the outside polygon to (x_1, y_1). We find all of the sides which cross this line segment. There are two ways for a side to cross. The side could be drawn starting below the line, cross it, and end above the line (*e.g.*, edge no. 1). In this case, we give the direction number of '–1' to the side. Or edge could start above the line and finish below it (*e.g.* edge no. 2, 3), this case is given the direction of '1'. *The sum of direction numbers of the sides that cross the constructed horizontal line segment yields the winding number* for the point. If the winding number is nonzero, then the point is in interior region, otherwise, in the exterior region *i.e.*, outside the polygon.

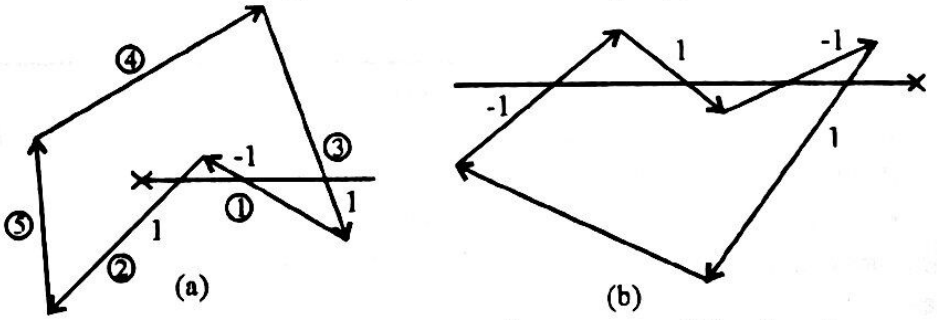


Fig. 3.7. Calculating winding number as sum of the direction numbers for the sides crossed.

In our example, (see figure 3.7(a)) line segment crosses three edges having direction numbers 1, –1, 1 respectively, then

$$\text{Winding Number} = (1) + (-1) + (1)$$
$$= 1$$

which non-zero hence, point is inside the polygon.

Similarly, in figure 3.7 (b), line segment crosses the four edges having direction numbers 1, – 1, 1, – 1 respectively, then

$$\text{Winding Number} = (1) + (-1) + (1) + (-1)$$
$$= 0$$

which is zero, hence point is outside the polygon.

Note : In figure 3.7 (a), edge (1) has direction number '–1' because it starts from below the line segment constructed and finishes above and similarity edge (2) has direction number 1 because it starts from above and finishes below the line segment.

3.4. FILLING POLYGON

There are various ways to fill the area and these methods depend on the nature of the boundary as follows,

- Boundary-Fill Algorithm
- Flood-Fill Algorithm
- Scanline Algorithm

3.4.1. Boundary-Fill Algorithm

In this method, area filling starts from a random point, called a seed, inside a region and continues painting towards the boundary. If boundary is monocolour, then the fill algorithm proceeds outward pixel by pixel until the boundary colour is encountered. Such a method is called the boundary-fill algorithm

There may be two methods for proceeding to neighbouring pixels from the seed point :

- Four adjacent point may be tested
- Eight adjacent point may be tested

In first case, left, right, above and below the seed point area tested and known as 4-connected and is used for partical filling of areas.

In second case, also includes four diagonal pixels and used to fill more complex figures. This method is called 8-connected.

Algo.

1. A Random point is considered *i.e.*, seed.
2. Each pixel to the left, right, above and below are tested (in case of 4-connected).
   - (If 8-connected, diagonal pixels are also tested).
3. Repeat process until all pixels up to the boundary color for the area have been tested (*i.e.* when boundary colour is hit.)

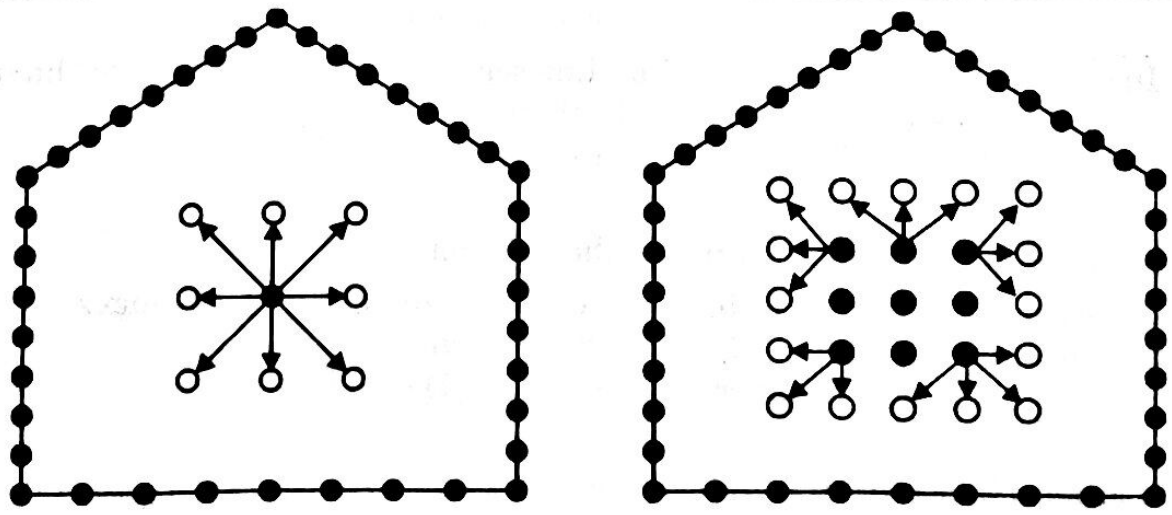


Fig. 3.8. Boundary filling for polygon for 8-connected.

This method can be employed more efficiently using stack for adjacent pixels. In this method, horizontal pixels are considered across scan line, instead of proceeding 4-connected or 8-connected adjacent pixels. Then we need only stack a beginning position for each horizontal pixel in scanline, instead of stacking all adjacent points around the seed. In this way all scanlines present in the polygon are considered respectively for filling the bounded area by the single colour boundary.

3.4.2. Flood-Fill Algorithm

This is the method, which is used to fill the area bounded by different color boundaries. Such area can be painted by replacing a specified interior color instead of reaching for a boundary color value. This method is called a flood-fill algorithm. We start from a specified interior point and reassign and pixel values that are currently set to a given interior color with the desired fill color. If the area we want to paint has more than one color, we can first reassign pixel values so that all interior points have the same color and we then step through pixel positions untill all interior points have been repainted.

This method also can be employed more efficiently using stack implementation similarity as in boundary. Fill algorithm.

Algo.

1. Choose a random interior point.
2. Replace the interior color with desired colour.
3. If area to be painted has more than one color then reassign all pixel values to the same color.
4. Repeat the process untill all interior points have been repainted.

3.4.3. Scanline Algorithm

The scan-line algorithm includes following steps :

Algo.

1. Sort the polygon sides on the largest y value.
2. Start with the largest y value and scan down the polygon.
3. For each y, determine which sides can be intersected and find the x values of these intersection point.
4. Sort, pair and pass the x values to the line-drawing routing.

This algorithm is implemented in the following example.

It is assumed that the vertex list describing the polygon is in order and that the polygon already has been seated to viewport and window, as shown in the following matrix,

Vertex	1	2	3	4	5	6	7	8	9	10	11	12	13
x	1	1	2	2	5	5	7	7	5	5	3	3	1
y	1	5	5	7	7	5	5	1	1	2	2	1	1

The first step of the scan conversion algorithm is to create an edge list (EL) which contains one cell for each y scan line. Therefore if vertical resolution of

screen is n pixels then edge entries in the edge list will be n. Here we assume $n = 8$. The edge list entry values are found by the $1/m$ where m is the slope of each edge and finding the maximum y value for the edge and minimum x value. These values are sorted one by one into the edge list by minimum value. In the following example, there are 13 vertices and 12 edges and edges 2, 4, 6, 8, 10 and 12 are horizontal. Therefore, filling between vertical edges will account for them, so they need not be included in the edge list. Sorting by minimum y value in the edge list as shown in table 3.2. Each edge list entry contains y_{min}, y_{max}, $1/m$ and x_A for y_{min} and y_{max} both. Now polygon can be filled by stepping y in increments of 1, because $n = 8$. So y will increment from 1 to 8. As y is increased by 1, edges will become active. An edge becomes active when the scan line value y equals the edge's cell number and the edge remains active until the scanline value y equals the edge's y_{max} value. For example, when $y = 1$, edges E_1, E_7, E_9 and E_{11} will become active i.e. the line $y = 1$ intersects E_1, E_7, E_9 and E_{11}. Active edges are then sorted by the x_A value to form the active edge list i.e. E_1, E_{11}, E_9, E_7. The actual fill is performed by filling the pixels, one scan line at a time between edge pairs. After the pixels are filled, the edges whose y_{max} value equals the current value of y are removed from active edge-list. The x_A values for remaining edges are incremented by $1/m$ end y is incremented by 1. The new edges that become active are added to EL and process is repeated until y equals maximum cell number.

There are some side effects of scan-conversion given as follows, (Refer Section 1.7 for detail).

1. Aliasing
2. Unequal intensity
3. Overstrike
4. Picket Fence Problem

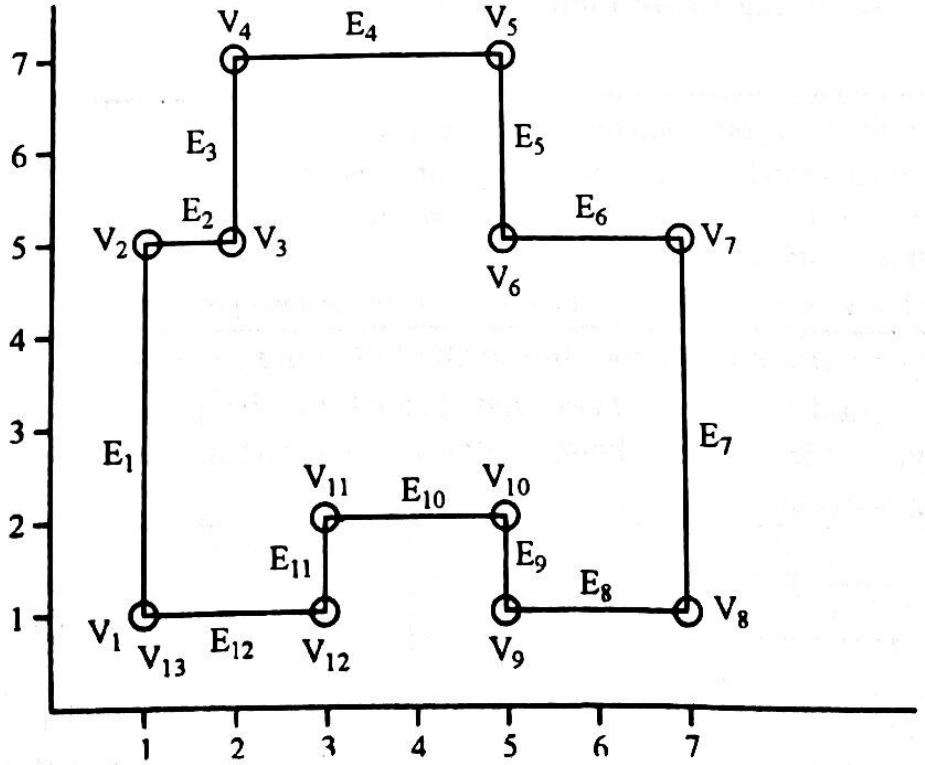


Fig. 3.9. Scan conversion of polygon.

Table 3.1.

Edge	Y_{min}	y_{max}	x_A for y_{min}	x_A for y_{max}	$1/m$
E_1	1	5	1	1	0
E_2	5	5	1	2	—
E_3	5	7	2	2	0
E_4	7	7	2	5	—
E_5	5	7	5	5	0
E_6	5	5	5	7	—
E_7	1	5	7	7	0
E_8	1	1	5	7	—
E_9	1	2	5	5	0
E_{10}	2	2	3	5	—
E_{11}	1	2	3	3	0
E_{12}	1	1	1	3	—

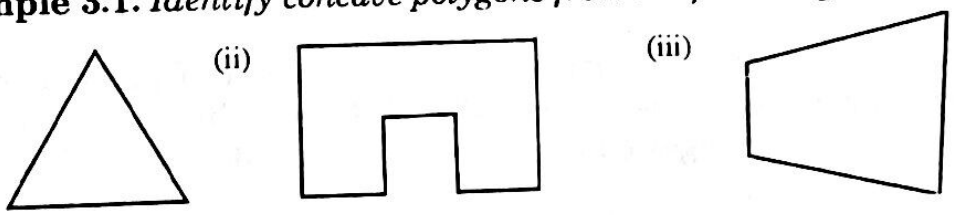
Table 3.2.

El	Edge Number
Cell 8	—
Cell 7	—
Cell 6	—
Cell 5	3, 5
Cell 4	—
Cell 3	—
Cell 2	—
Cell 1	1, 7, 9, 11

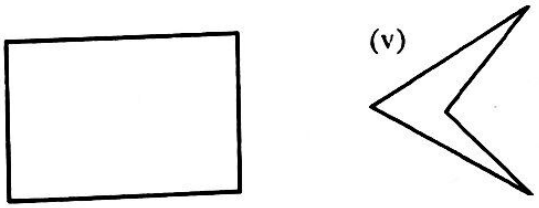
SOLVED EXAMPLES

Example 3.1. *Identify concave polygons from the following :*

(i) (ii) (iii)

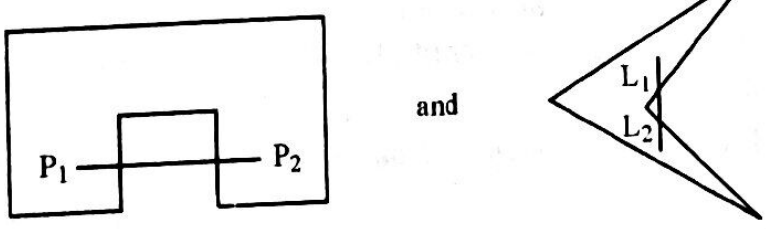


(iv) (v)



[B.Tech. (IE)-2003 (CO) UPTU]

Sol. We know that if line joining any two interior points of the polygon is not completely inside the polygon, then polygon is concave.



In (ii) & (v), line segments P_1, P_2 and L_1 L_2 are not completely polygons inside the polygon, hence polygons (ii) and (v) are concave.

Example 3.2. *What are the different types of polygons ? Categorize the following according to their types :*

[MCA-2003 UPTU]