3 Polygon clipping

Line clipping is acceptable when the output can be a set of disconnected lines segments. There are, however, situation in which a polygon clipped against a window should result in one or more polygons.
For example a region that is to be shaded must have a closed boundary.

The Sutherland-Hodgman polygon clipping algorithm clips a polygon against each edge of the window, one at a time. Specifically for each window edge it inputs a list of vertices and outputs a new list of vertices which is submitted to the algorithm for clipping against the next window edge.
The first window edge has as input the original set of vertices.
After clipping against the last edge, the output list consists of the vertices describing the clipped polygon.

In the algorithm, for each window edge, we tack the input list of vertices for that edge, tests a pair of consecutive vertices against a window edge to produce the output list of vertices. There are four possible cases of testing as illustrated in figure - 3 - Where testing is performed against the left edge

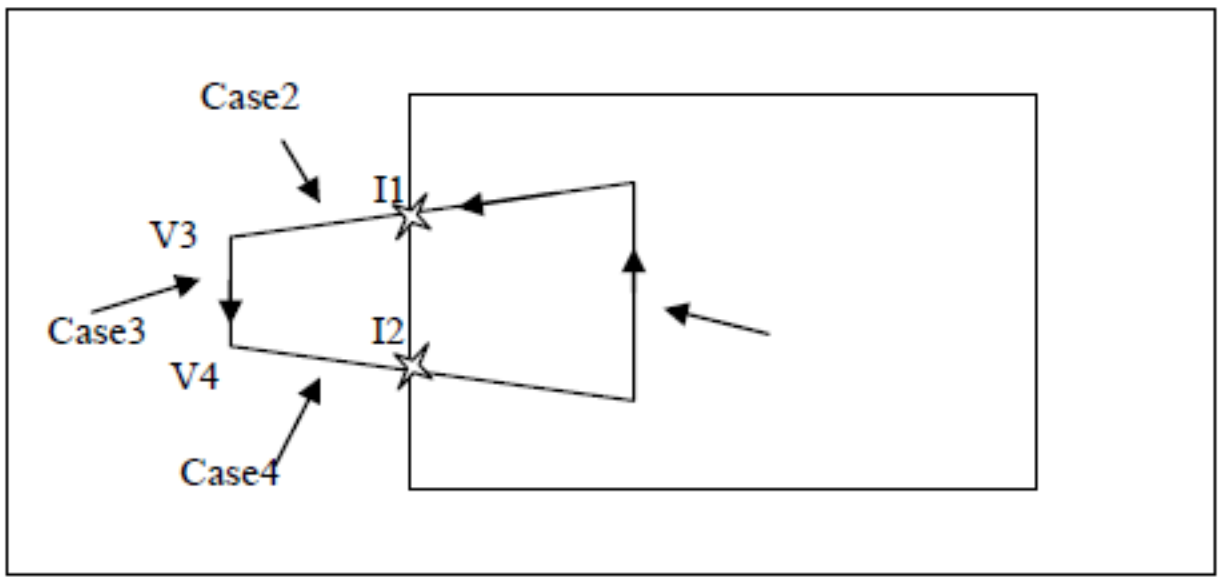


Figure - 3 -

The sample polygon has vertices { V1 , V2 , V3 , V4 } .
Case 1 has the first and second vertices V1 and V2 inside the window, so the second vertex V2 is send to the output list.

Case2 has the first vertex V2 inside and the second vertex V3 outside the window. The point of intersection, I1 , of the side of the polygon joining the vertices and the edge is added to the output list.

Case3 has both vertices, V3 and V4 outside the window and no point is output.

Case4 has the first vertex V4 outside and the second vertex V1 inside the window, the point of intersection I2 and the second vertex V1 are added to the output list.

The result of this left clipping is the transformation of the input list { V1 , V2 , V3 , V4 } to the output list { V2 , I1 , I2 , V1 }.

Programming the polygon clipping:

The declarations:

1- We need to define three arrays to hold the list of polygon vertices, each array is of two dimension, for the X and Y values of each vertex.

```
ORG ( n , 2 ) / n is the number of vertices in the polygon
                This matrix to store the coordinate of the vertices

POLYON   (25,2) / the vertices input list {maximum 25 vertex}

POLYOUT (25,2) / the vertices output list{maximum 25 vertex }
```

2- We need to define a number of variables :

```
Vertices-no : integer ; counter of the input vertices
              {same as n above}

Outlin      : integer ; counter of the output vertices
```

Xmin,Xmax
Ymin,Ymax
} Integer ; the boundary of the window

V1 (1,2) : array for the first vertex
V2 (1,2) : array for the second vertex

Edge : string ; for the edge code

3- We need two functions

V3=***Intersection*** (V1 , V2 , Edge) : used to determine the intersection point of the line between two vertices and the clip edge

Boolean=***Inside*** (V , Edge) : return true if the vertex V is inside the edge , or return false if V is outside the edge

3- the algorithm

1- Store the values of the vertices coordinate in the ORG array

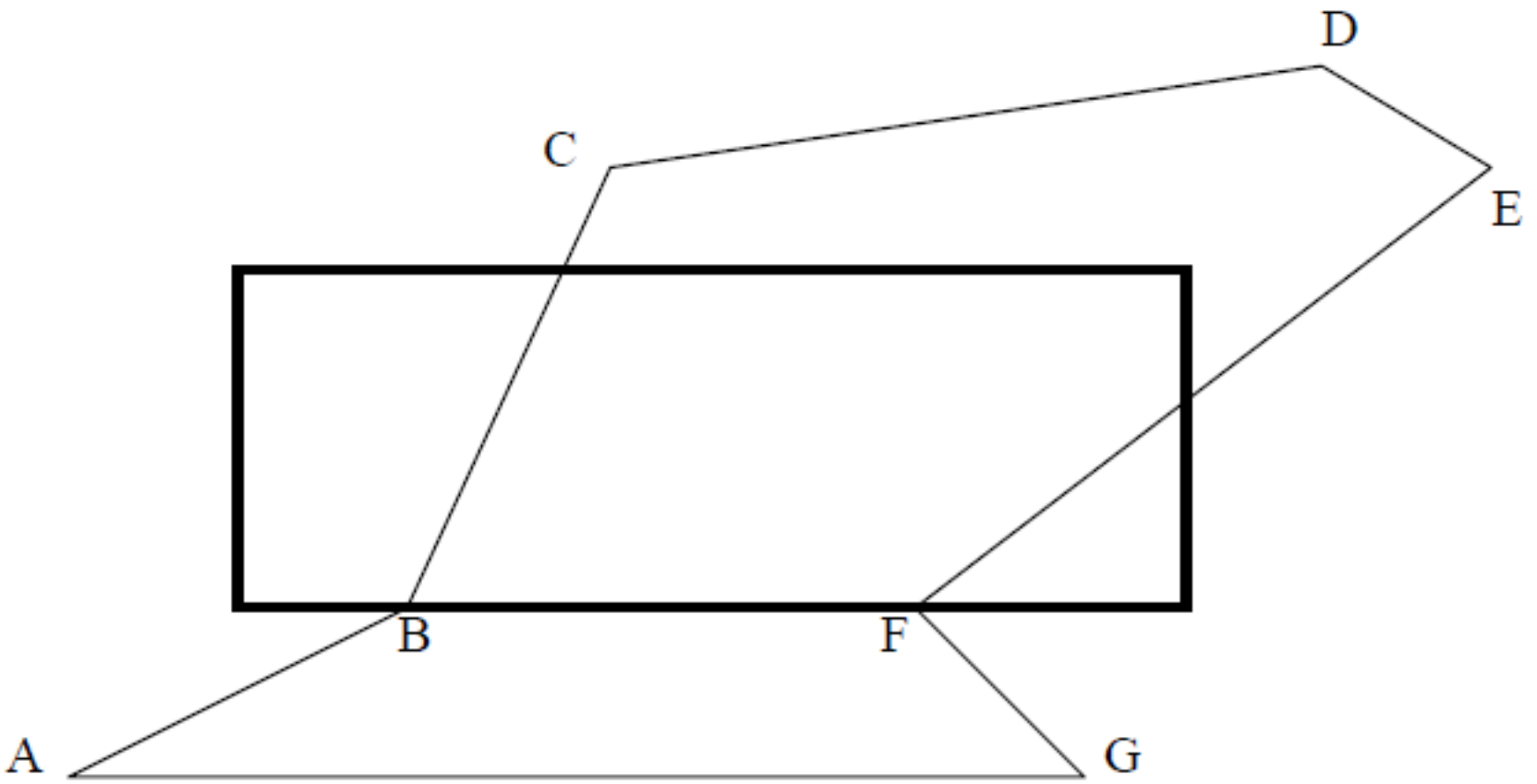
2- Transfer the values in ORG to POLYIN { the initial set of vertices}

```
For I=1 to Vertices-no
  POLYIN  (I,1) = ORG (I,1)  { X values }
  POLYIN  (I,2) = ORG (I,2)  { Y values }
Next
```

3- Edge = "L" { start from the left edge of the window }

```
For  EG=1 to 4 { we have four edges in the window }
      Outlin = 0 { at first there are no output vertices }
       Take the first vertex from POLYIN and put it in V1
        For J=1 to Vertices-no
          Take the second vertex from POLYIN and put it in V2

          If Inside (V2,Edge) then  { case 1 or case 4 }
             If Inside (V1,Edge) then  { case 1}
                Outlin=Outlin +1
                 POLYOUT (Outlin)=V2          ] output
             Else
                V3= calculate the Intersection (V1,V2,Edge)
                 Outlin=Outlin +1
                  POLYOUT (Outlin)=V3         ] output

                 Outlin=Outlin +1
                 POLYOUT (Outlin)=V2          ] output
             End IF

          Else  { case 2 or case 3 }

             If  Inside (V1, Edge) Then  { case 2 }
               V3=Intersection (V1,V2,Edge)

              Outlin=Outlin +1
               POLYOUT (Outlin)=V3            ] output
             End if
          End IF

          V1=V2

        Next J
              - Set Edge to the next Edge
              - Transfer the values in POLYOUT to POLYIN
              - Vertices-no=Outlin

         Next EG
```

Example: Clip the polygon against the window



The equation for *t* becomes,

$$t = \frac{N_i \cdot [P_0 - P_{Ei}]}{-N_i \cdot D}$$

It is valid for the following conditions:

1. $N_i \neq 0$ (error cannot happen)
2. $D \neq 0$ ($P_1 \neq P_0$)
3. $N_i \cdot D \neq 0$ (P_0P_1 not parallel to E_i)

Polygon Clipping (Sutherland Hodgman Algorithm)

A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman polygon clipping algorithm is used for polygon clipping. In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.

First the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window as shown in the following figure.

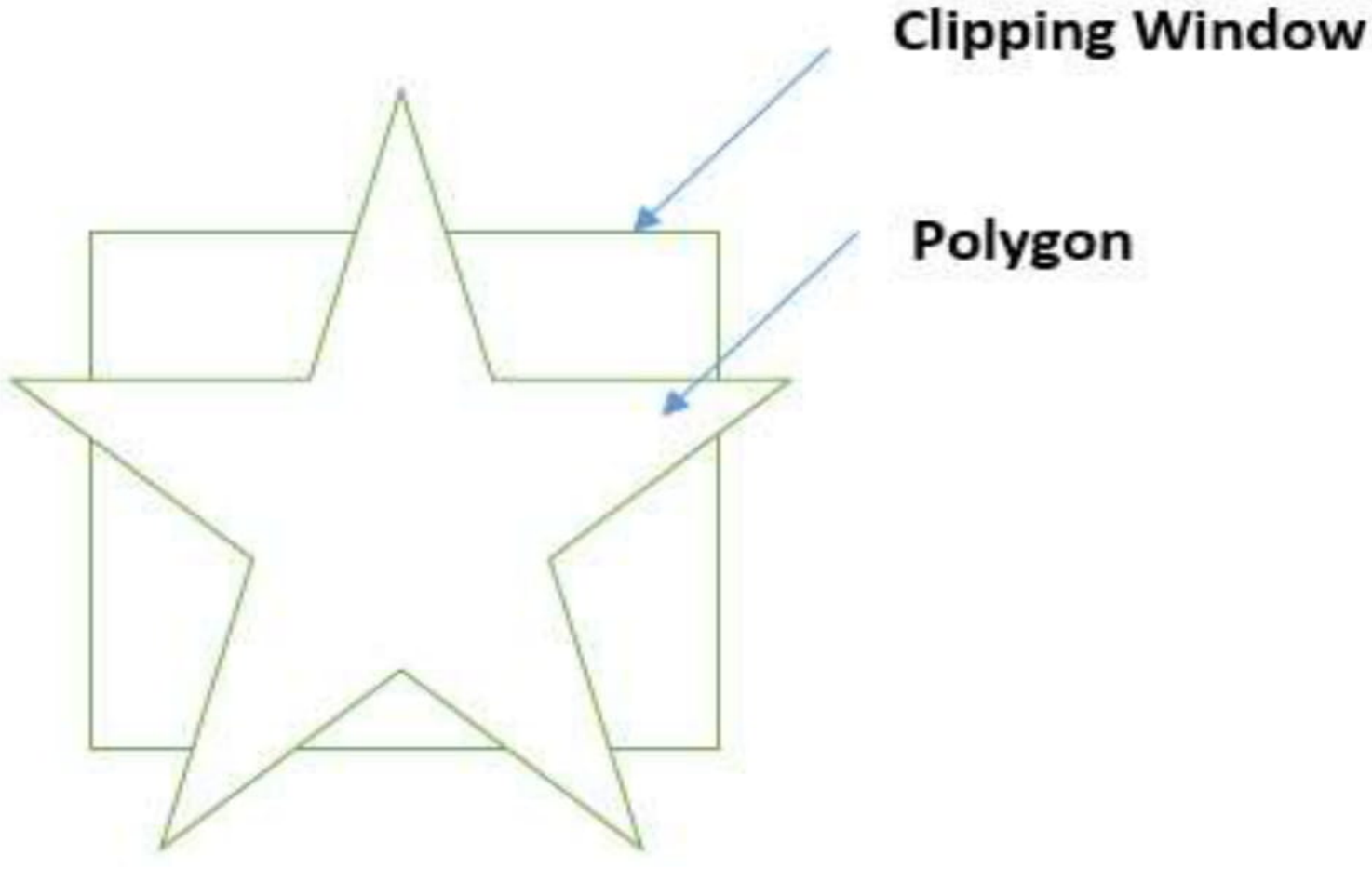


Figure: Polygon before Filling

While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and the a partial edge from the intersection point to the outside edge is clipped. The following figures show left, right, top and bottom edge clippings:

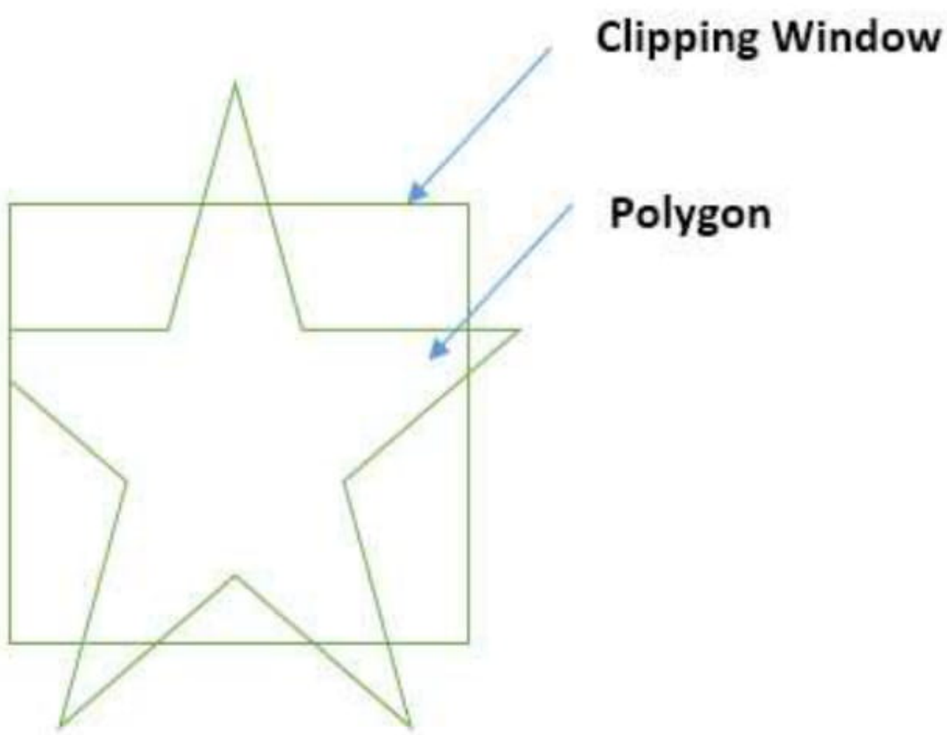


Figure: Clipping Left Edge

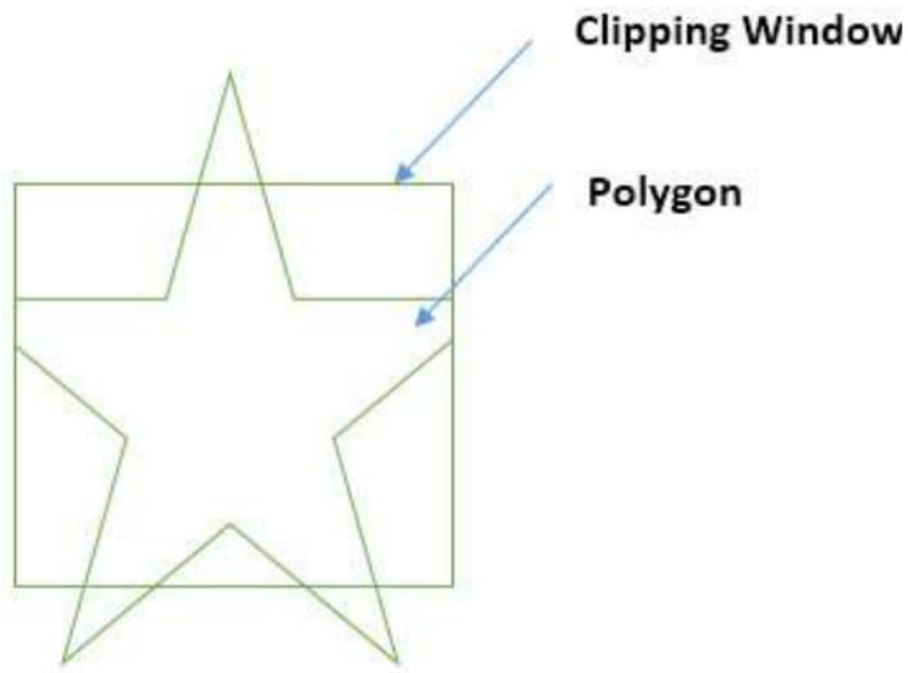


Figure: Clipping Right Edge

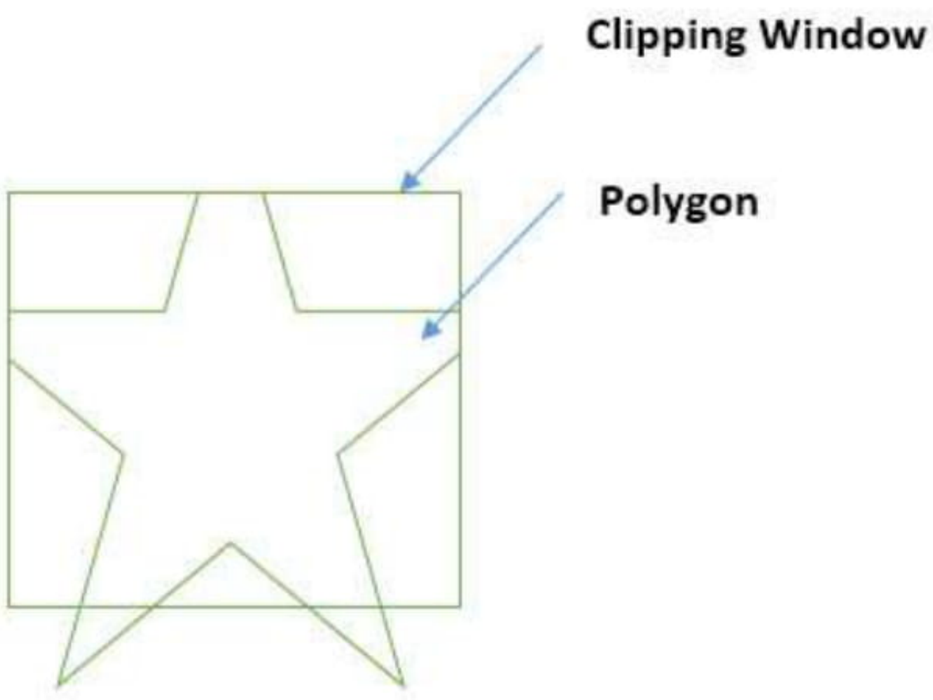


Figure: Clipping Top Edge

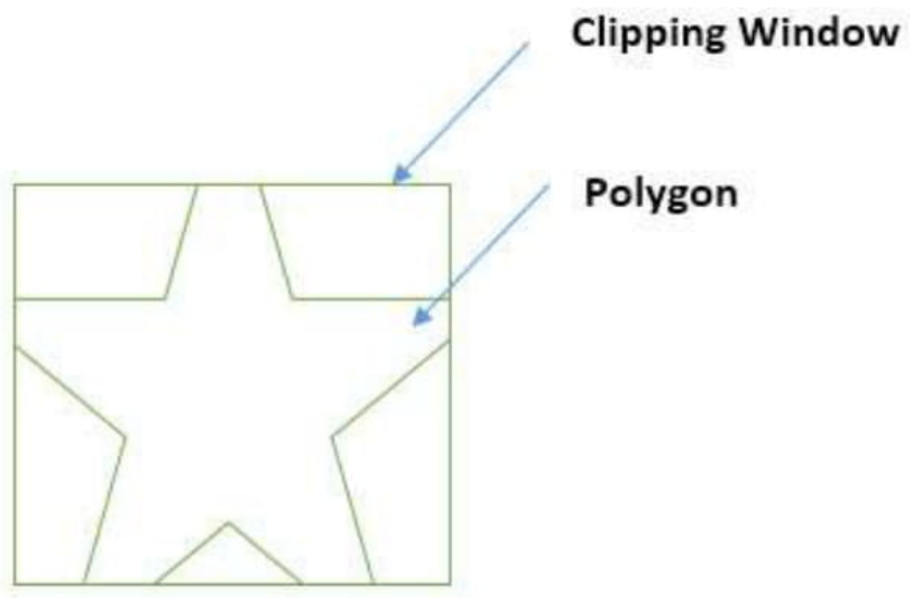


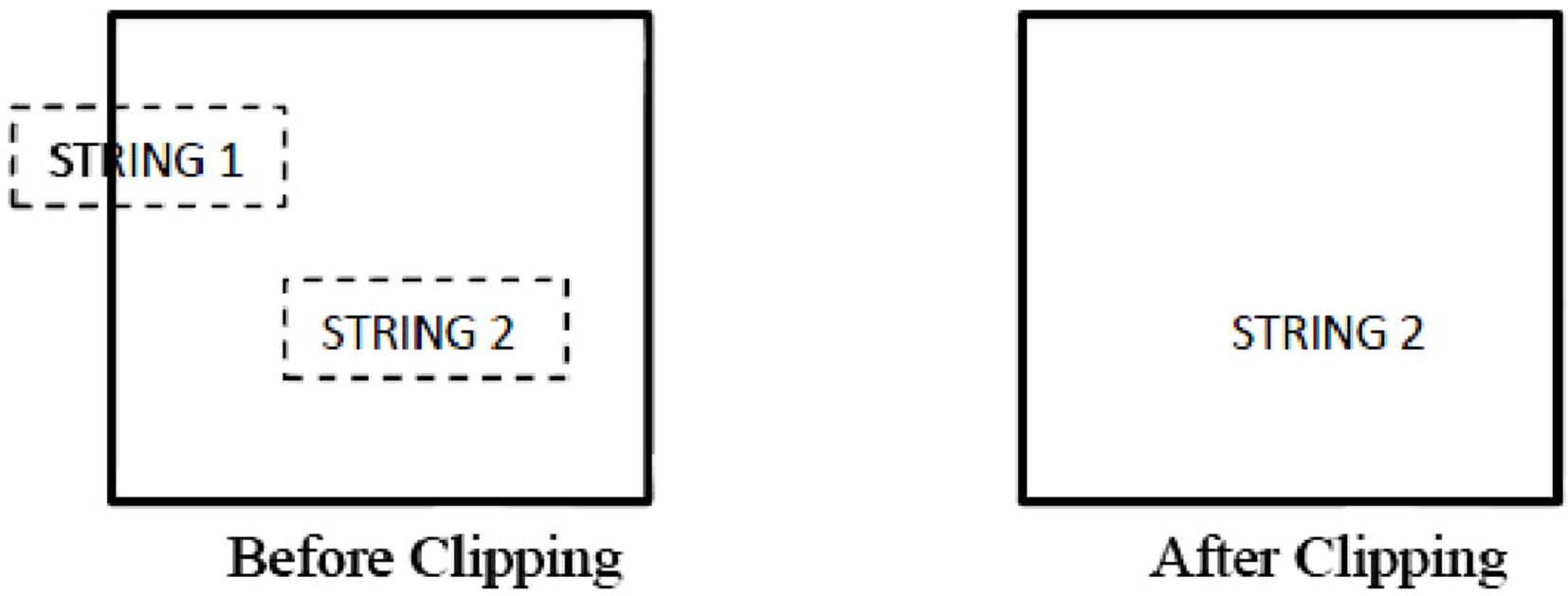
Figure: Clipping Bottom Edge

Text Clipping

Various techniques are used to provide text clipping in a computer graphics. It depends on the methods used to generate characters and the requirements of a particular application. There are three methods for text clipping which are listed below:

1. All or none string clipping
2. All or none character clipping
3. Text clipping

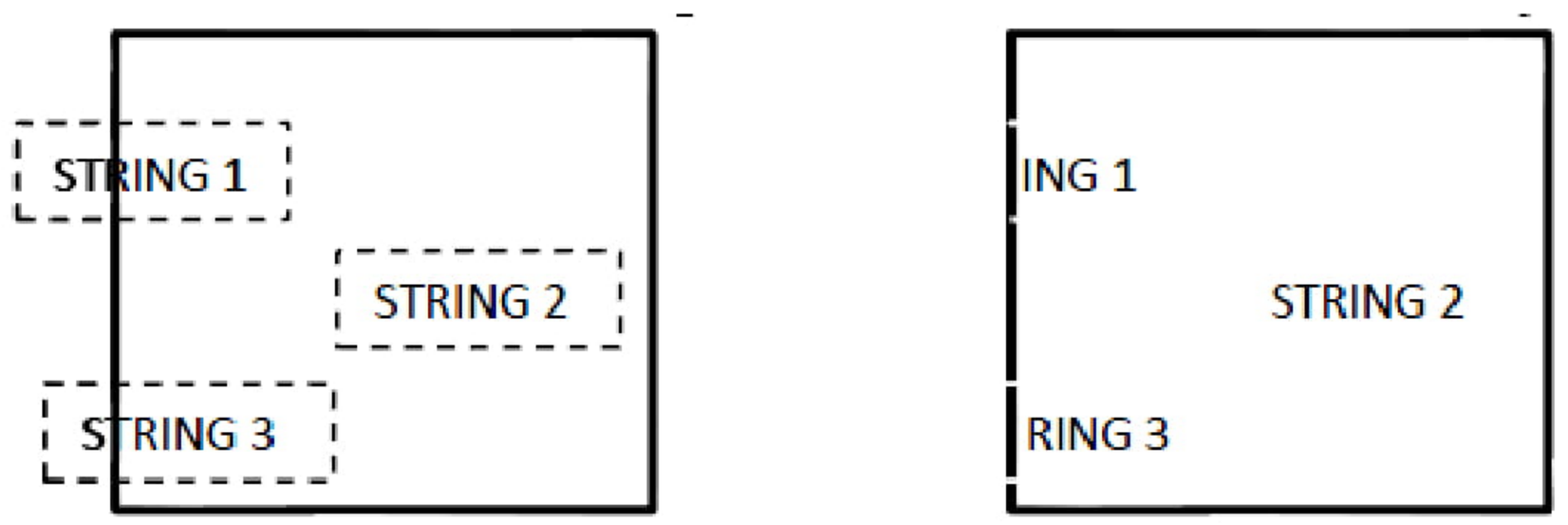
The following figure shows all or none string clipping:



Before Clipping After Clipping

In all or none string clipping method, either we keep the entire string or we reject entire string based on the clipping window. As shown in the above figure, STRING2 is entirely inside the clipping window so we keep it and STRING1 being only partially inside the window, we reject.

The following figure shows all or none character clipping:

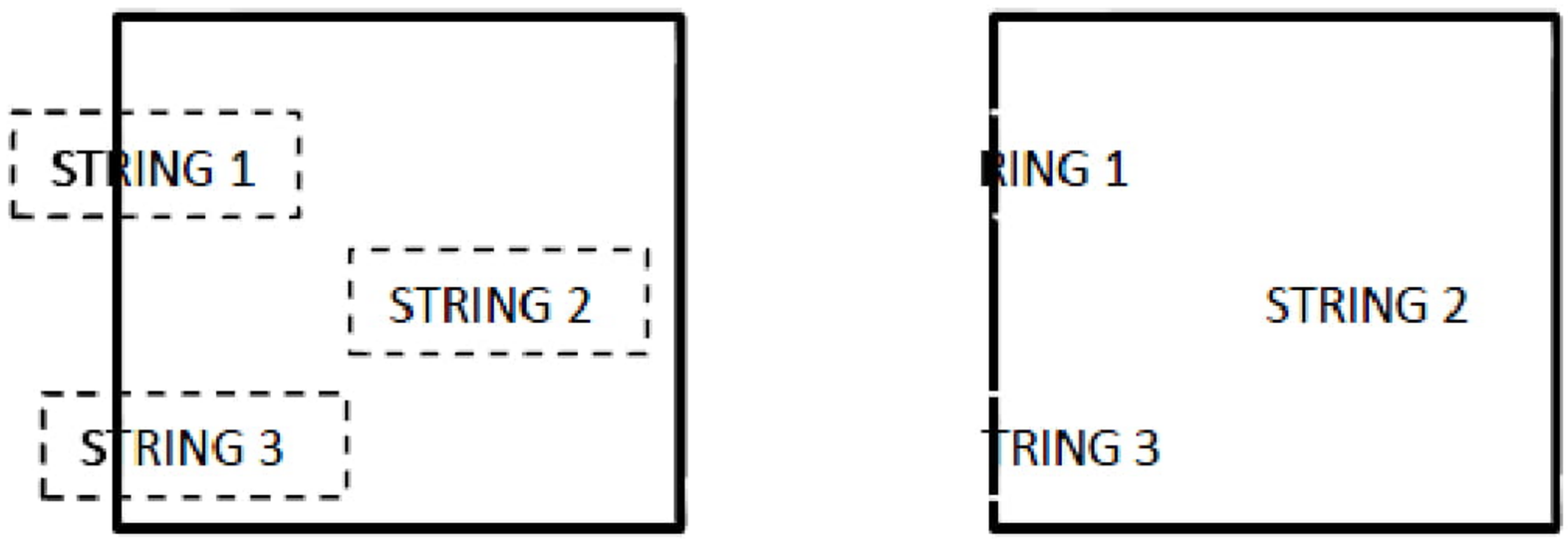


Before Clipping After Clipping

This clipping method is based on characters rather than entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then:

- You reject only the portion of the string being outside
- If the character is on the boundary of the clipping window, then we discard that entire character and keep the rest string.

The following figure shows text clipping:



Before Clipping After Clipping

This clipping method is based on characters rather than the entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then

- You reject only the portion of string being outside.
- If the character is on the boundary of the clipping window, then we discard only that portion of character that is outside of the clipping window.

Bitmap Graphics

A bitmap is a collection of pixels that describes an image. It is a type of computer graphics that the computer uses to store and display pictures. In this type of graphics, images are stored bit by bit and hence it is named Bit-map graphics. For better understanding let us consider the following example where we draw a smiley face using bit-map graphics.

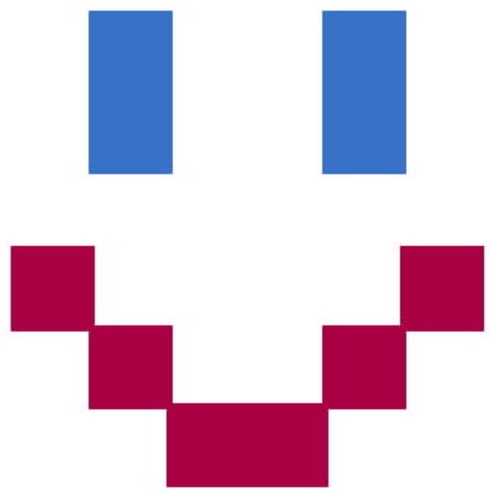


Figure: Original Smiley face

Now we will see how this smiley face is stored bit by bit in computer graphics.

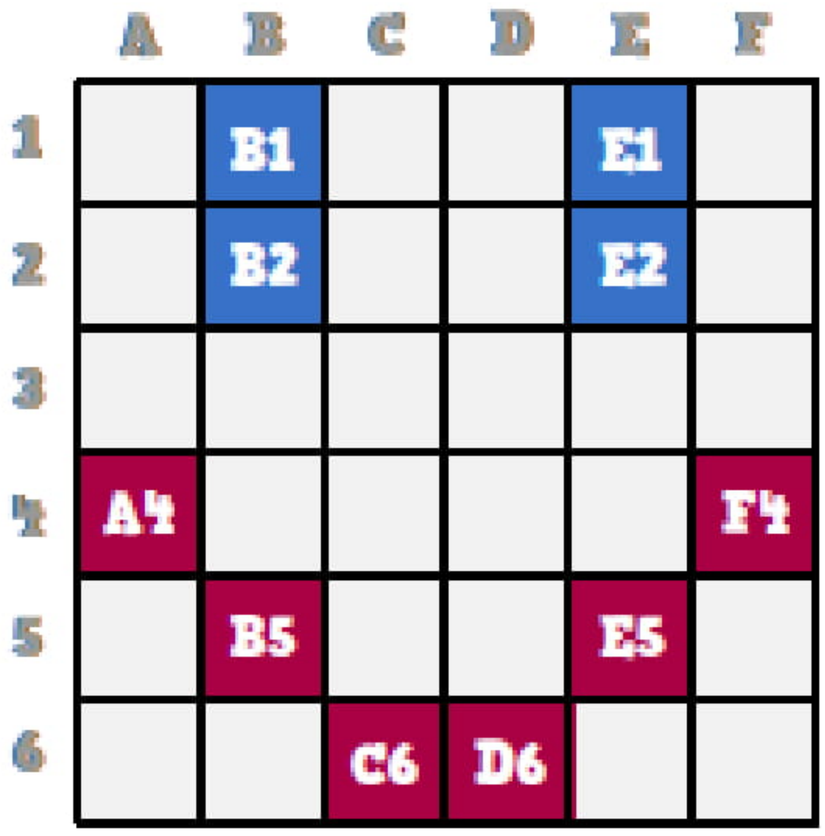


Figure: Bitmap storage of smiley face

By observing the original smiley face closely, we can see that there are two blue lines which are represented as B1, B2 and E1, E2 in the above figure.

In the same way, the smiley is represented using the combination bits of A4, B5, C6, D6, E5, and F4 respectively.

The main disadvantages of bitmap graphics are:

- We cannot resize the bitmap image. If you try to resize, the pixels get blurred.
- Colored bitmaps can be very large.