CHAPTER 6

TWO DIMENSIONAL VIEWING AND CLIPPING

CONTENTS

6.0 INTRODUCTION:

In this chapter we shall discuss the techniques of displaying the views of a picture on the out put device. Any world coordinate system can be referred to as the reference frame and can be used to define a picture. The user can then select a sub-area of the picture and can display the sub-area some where in the output screen. The user can select a single sub-area or can select several areas for displaying in the output screen. The selected areas then mapped into the device coordinates and displayed. Transformation from world to device coordinate system requires translation, rotation and scaling operations. Also the process of

eliminating the parts of the picture should be applied. The mapping of world coordinate to device coordinate and eliminating the non-selected parts are called viewing and clipping respectively. In this chapter we first discuss two-dimensional viewing and the different clipping operations for different objects like points, lines, polygon etc.

6.1 TWO DIMENSIONAL VIEWING:

World Coordinate System: Any convenient Cartesian coordinate system is referred to as world coordinate system.

Window: An area in the world coordinate system that is selected for display is called a window. It says what is to be viewed.

View Port: An area on a display device to which the window is mapped is called a viewport i.e., the viewport says where is to be viewed.

Often window and viewports are rectangular area with the edges parallel to respective coordinate axis. In general mapping the window to viewport is called viewing transformation or window-to-viewport transformation or normalization transformations.

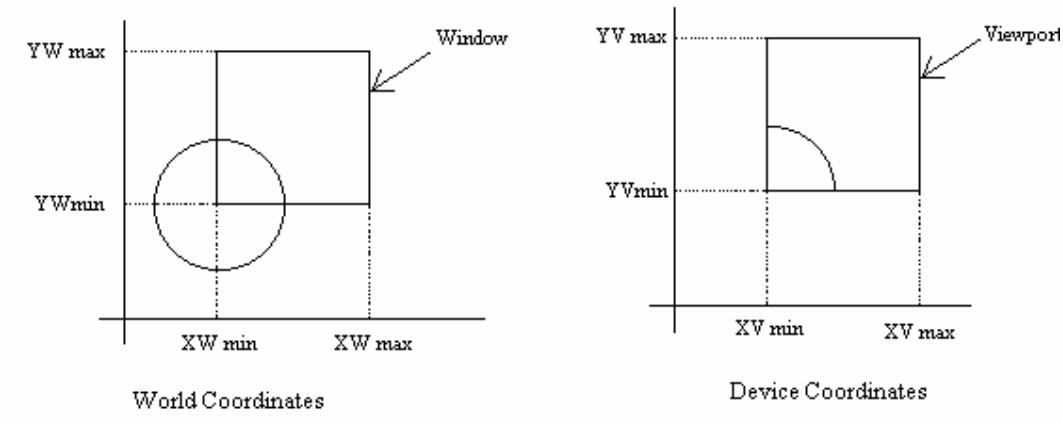

Fig-6.1: World Coordinate to Device Coordinate Transformation

6.1.1. VIEWING PIPELINE:

Sharing of different graphics systems in the overall system architecture is called the graphics pipeline. i.e., before displaying, the object goes through different graphics systems. At the very beginning of the pipeline we have object data stored in specific data-structure. For example, end points for straight-line or vertex pints for a polygon are stored. Whenever we need to display an object, first the modeling transformation is performed to the original data. Then viewing transformation is performed using the current window and viewport settings. Finally the scan conversion is performed with proper pixel and the specified color.

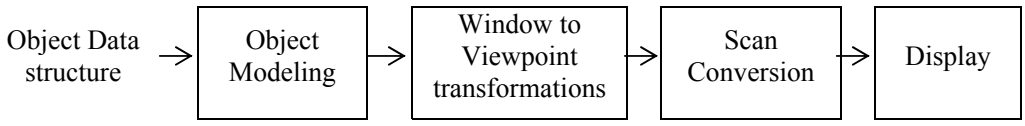

Fig-6.2: Viewing Pipeline

6.1.2 WINDOW-TO-VIEWPORT TRANSFORMATION:

We know that the monitor sizes differ from one system to another. So it will be better to introduce a device independent mechanism to describe the display area. This mechanism is called device independent system or Normalized Device Coordinate System (NDCS). NDCS is a unit square (1x1) whose lower left corner is at the origin of the coordinate system that defines the display area of the device.

A window is specified by the four world coordinates XWmin, XWmax, YWmin YWmax. Similarly a viewport is described by device coordinates XVmin, XVmax, YVmin and YVmax. The ultimate objective of window-to-viewport mapping is to transfer the world coordinates (XW, YW) to its corresponding viewport or normalized device coordinates (XV, YV). For maintaining the relative placement in the viewport we need

$$\frac{XV - XVmin}{XVmax - XVmin} = \frac{XW - XWmin}{XWmax - XWmin} \quad \text{…….. …6.1}$$

and

$$\frac{YV - YVmin}{YVmax - YVmin} = \frac{YW - YWmin}{YWmax - YWmin} \quad \text{……...6.2}$$

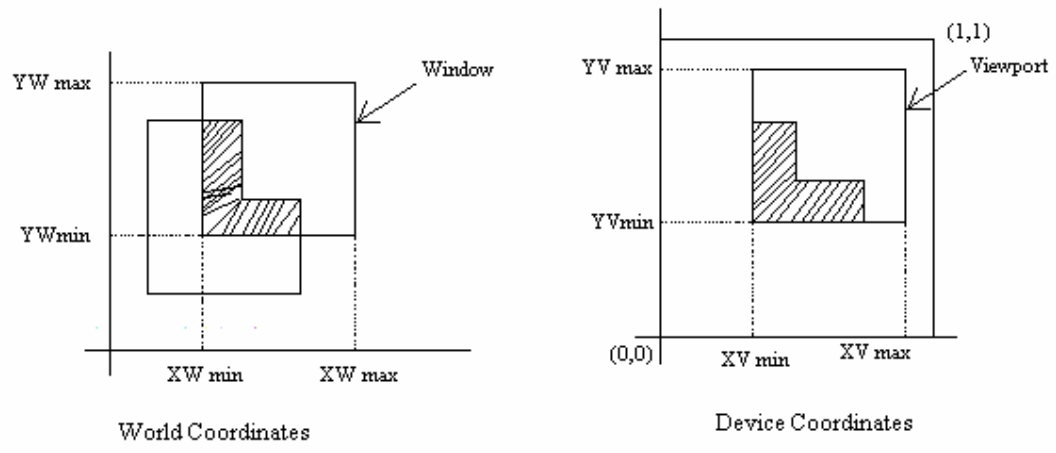

Fig-6.3: Window – to viewport- mapping

Solving the equations 6.1 and 6.2, we get,

$$XV = XVmin + (XW - XWmin)\left(\frac{XVmax - XVmin}{XWmax - XWmin}\right) \quad \text{..... 6.3}$$

$$YV = YVmin + (YW - YWmin)\left(\frac{YVmax - YVmin}{YWmax - YWmin}\right) \quad \text{..... 6.4}$$

Writing $\frac{XVmax - XVmin}{XWmax - XWmin} = Sx$ and

$$\frac{YVmax - YVmin}{YWmax - YWmin} = Sy$$

the equations 6.3 and 6.4 can be rewritten as

$$XV = XVmin + (XW - XWmin)\,Sx \quad \text{..... 6.5}$$

$$YV = Yvmin + (YW - YWmin)\,Sy \quad \text{..... 6.6}$$

Since, coordinate values that defines window and viewport (i.e., XVmax, YVmin etc.) are all constants, we can express these two formulas for computing (XV, YV) from (XW, YW) in terms of translate-scale-translate transformations.

(XV, YV, 1) = (XW, YW, 1). M where

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -XVmin & -YVmin & 1 \end{pmatrix}.\begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ XWmin & YWmin & 1 \end{pmatrix}$$

One interesting point to be noted is that the geometric distortions may occur in this type of transformations i.e., square window may become rectangular viewport if Sx, Sy differs.

6.2 CLIPPING:

Clipping is a mechanism to check which portion of the picture will be inside and which portion will be outside of a specified region. The region against which the clipping is done is called the clip window. For viewing transformation, we want to display only those parts which are inside the window. Everything else will be discarded. There are two ways of applying the clipping algorithms. The first one is to apply the algorithm in the world coordinate system and discard everything which is not within the window. The other method is to map everything in the normalized device coordinates and then clip against the viewport boundary. Both have their advantages and disadvantages. There are several type of clipping but we shall consider only point clipping, line clipping and polygon clipping.

6.3 POINT CLIPPING:

Let the clip window is a rectangle. We shall keep a point (x,y) for display if

$$XWmin \leq x \leq XWmax \text{ and}$$

$$YWmin \leq y \leq YWmax$$

Where XWmax, XWmin, YWmax, YWmin are word-coordinate window boundaries. If any one of the four inequalities is not satisfied, then the point is clipped i.e., the point will not be kept for display.

6.4 LINE CLIPPING:

Line clipping means saving that part of the line which will fall within the window and discarding the other part. We shall take the clipping region as a rectangle. In line clipping we shall execute the following three steps.

(i) Check whether the line is completely inside of the clipping region or not.

(ii) Check whether the line is completely outside of the clipping region or not

(iii) Other wise we have to find out the intersection of the line with one or more boundaries.

We can do the above mentioned steps by inside-outside test.

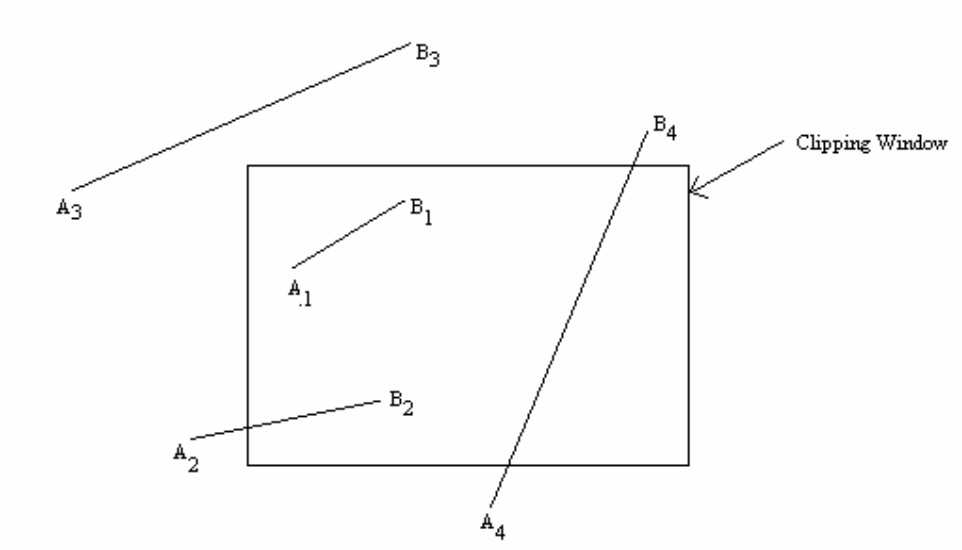

Fig-6.4: Clipping of straight lines by rectangular viewport

For example the line A_1B_1 will be completely kept for display where as line A_3B_3 will completing be discarded. Other lines like A_2B_2 and A_4B_4 requires calculation of one or more intersection points. We shall now develop some efficient line clipping algorithm.

6.4.1 COHEN-SUTHERLAND ALGORITHM:

This is one of the most popular line-clipping algorithms. This algorithm efficiently calculates the number of intersections that must be calculated.

The algorithm works as follows –

(i) Identify those lines which lies completely fall outside the clip window i.e., completely invisible.

(ii) Identify those lines which falls completely inside the clip window i.e., completely visible.

(iii) Identify those lines for which clipping are to be performed.

For doing these efficiently we shall apply the following technique.

Consider the following region code table.

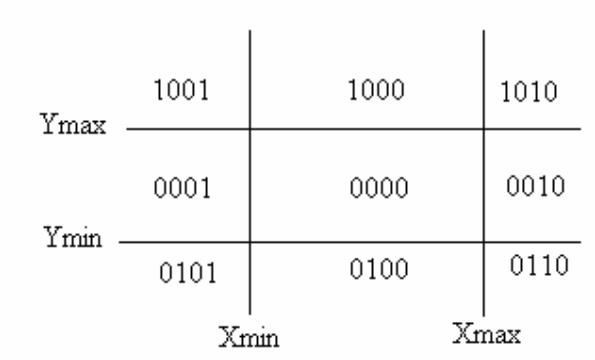

Fig-6.5: Code table for Cohen-Sutherland Algorithm

Every line end point will be assigned a four digit region code and thus the location of that end point will be determined. The location can be determined as follows

We shall consider the least significant bit as bit 1 or the rightmost bit as bit 1

If bit 1 is 1 : left
If bit 2 is 1 : right
If bit 3 is 1 : below
If bit 4 is 1 : above.

A value 1 in any position indicates that the line end point will be at that region. For example a line end point that will lie in region below and right will have the code 0110. An end point that will lie inside the clipping window will have all zeros (0000). Let us now turn to the question, how the bit values are generated. The bit values are generated by comparing the coordinates of the end points with the clip boundaries.

The sign of the difference of the end point and the clip boundary is considered. If sing is positive then corresponding bit value is set to one. If the sign is negative then the bit value is set to zero. A sign 0 will be considered as 0.

The comparison table is given below.

Bit position 1 is 1 if sign (Xmin –X) is positive
Bit position 2 is 1 if sign (X –Xmax) is positive
Bit position 3 is 1 if sign (Ymin –Y) is positive
Bit position 4 is 1 if sign (Y –Ymax) is positive

Once the region codes of the end points are determined the next task is to clip the lines. The line for which both the end pints have region value 0000 is completely inside the clip window i.e., Then is nothing to clip. If the region codes of both the end points contains 1 at the same bit position, then the line is completely outside the clip window i.e. the entire line will be discarded. For example if the region codes for the end points of the line are 1010 and 0010 then the second bit position is giving 1 in both the cases. So the line will be completely discarded. One interesting and efficient method of calculating this is to find out the bitwise and operation of the region codes. If the result is nonzero, the line is completely discarded.

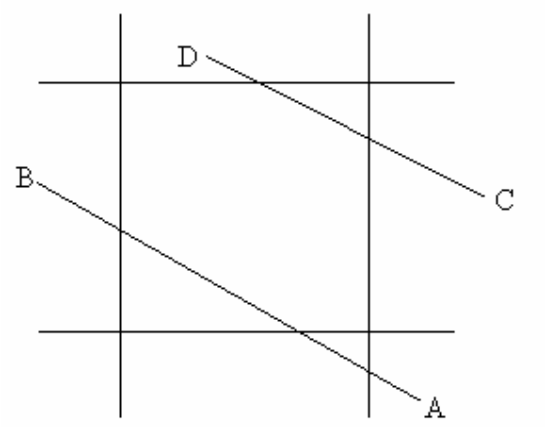

Fig-6.6: Lines considered for Clipping

The lines for which it is not possible to determines whether they are completely inside or outside, are considered for intersection with the clip window boundaries. We can do the same in the following way

If bit 1 is 1 intersect with X = Xmin
If bit 2 is 1 intersect with X = Xmax
If bit 3 is 1 intersect with Y = Ymin
If bit 4 is 1 intersect with Y = Ymax.

Consider the line AB in the above first we shall choose the end point A. Its region code is 0110. As the second bit is 1 choose the line X=Xmax for intersection. If B is chosen then X = Xmin would be chosen because Bit 1 is 1 in this case.

The coordinate of the end point can be given by

$$\begin{cases} X_i = Xmin \text{ or } Xmax \\ Y_i = Y_1 + m(X_i - X_1) \end{cases} \quad \text{Depending on Xmin or Xmax}$$

or

$$\begin{cases} X_i = X_1 + 1/m\,(y_i - y_1) \\ Y_i = Ymin \text{ or } Ymax \end{cases} \quad \text{Depending on Ymin or Ymax}$$

This intersection will divide the line into two parts and the part which lies outside the clip window is rejected and the new line is updated and considered for further clipping. The process terminates when falls in one of the first two categories i.e., completely inside or completely outside.

6.4.2 LIANG -BARSKY ALGORITHM:

Though Cohen-Sutherland algorithm is good for the clipping, there are other methods also. One of these faster methods is Liang-Barsky line clipping algorithm. This line clipping algorithm is based on parametric equation of a line.

Consider the following parametric equation of a straight line segment. Let (X_1, Y_1) and (X_2, Y_2) be the two endpoints of a line segment. The each of the line segment can be given by

$$x = X_1 + \mu\Delta x$$

$$\text{And} \quad y = Y_1 + \mu\Delta y$$

Where $\Delta x = X_2 - X_1$ and $\Delta Y = Y_2 - Y_1$ and $0 \le \mu \le$.

Consider any point on the line (x,y).

If the point falls inside the clipping window, then from the point clipping conditions stated in article 6.3 we get –

$Xmin \le X_1 + \mu\Delta X \le Xmax$ and
$Ymin \le Y_1 + \mu\Delta Y \le Ymax.$

We can rewrite the above four inequalities as

$$\mu P_k \le Q_k \quad \text{for} \quad k = 1,2,3,4.$$

Where $P_1 = \Delta X$ and $Q_1 = X_1 - Xmin$ for left

$P_2 = \Delta X$ and $Q_2 = Xmax - X_1$ for right

$P_3 = \Delta Y$ and $Q_3 = Y_1 - Ymin$ for bottom

$P_4 = \Delta Y$ and $Q_4 = Ymax - Y_1$ for above

The following facts are important

If $P_k = 0$ for k = 1,2,3,4 the the line is parallel to the corresponding clipping window boundary and if $Q_k < 0$ then the line is completely outside the clip window and can therefore be eliminated. If $Q_k \ge 0$ then the line is inside the boundary and further processing is needed.

When $P_k < 0$ then the extension of the line comes from outside to the inside of the corresponding boundary line. When $P_k > 0$, the extension of the line proceeds from inside to the outside. If $P_k \ne 0$, then the value of μ, (where μ is the intersection of the extended line and the extend boundary K) can be calculated by $\mu = P_k / Q_k$

Now, our aim is to calculate values for parameter μ_1 and μ_2 that defines the part of the line that lies within the clip window. For μ_1 we take all the window edges for which $P_k < 0$ and we calculate $r_k = Q_k/P_k$ and μ_1 is the maximum of the set containing 0, and the values of r_k.

Similarly for finding μ_2 we take all the K for which $P_k > 0$ and take $r_k = Q_k/P_k$ and we take the minimum of 1, and values of r_k. If $\mu_1 > \mu_2$ then the line is

completely outside the window. So we shall reject of otherwise the line segment is updated according to μ_1 and μ_2 values.

6.5 POLYGON CLIPPING:

A polygon clipping is basically modified lien clipping. A polygon is a closed area whose edges are straight lines. If we give a polygon to the line clipper, the line clipper will correctly clip the edges of the polygon.

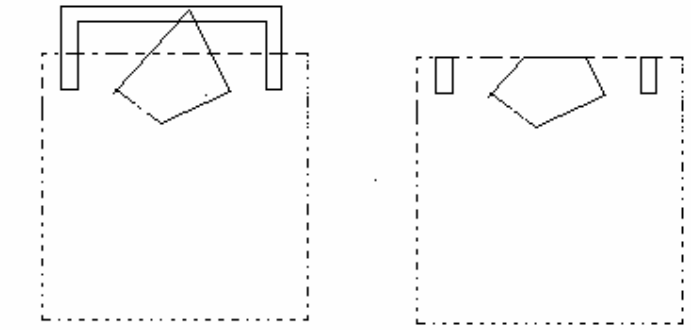

Fig-6.7: Polygon Clipping by Rectangular Clip Window

The only problem is that after clipping, the polygonal region will become open and it will be a collection of some unconnected line segments. What we really want is an algorithm that will generate one or more closed areas or closed polygons after clipping the original polygon. The output of the polygon clippings is basically a sequence of the polygon. Following two algorithms will discuss the techniques of polygon clipping.

6.5.1 SUTHERLAND-HODGEMAN POLYGON CLIPPING:

In this polygon clipping we shall consider the polygon boundary as a whole against each clip boundary. To start with, let us take the left window boundary. We shall calculate the inside and outside vertices against this boundary and shall update the polygon vertices to form a new polygon. The new polygon then considered for the right boundary. Then bottom boundary and then to upper boundary respectively. The figure given below is the illustration of the process.

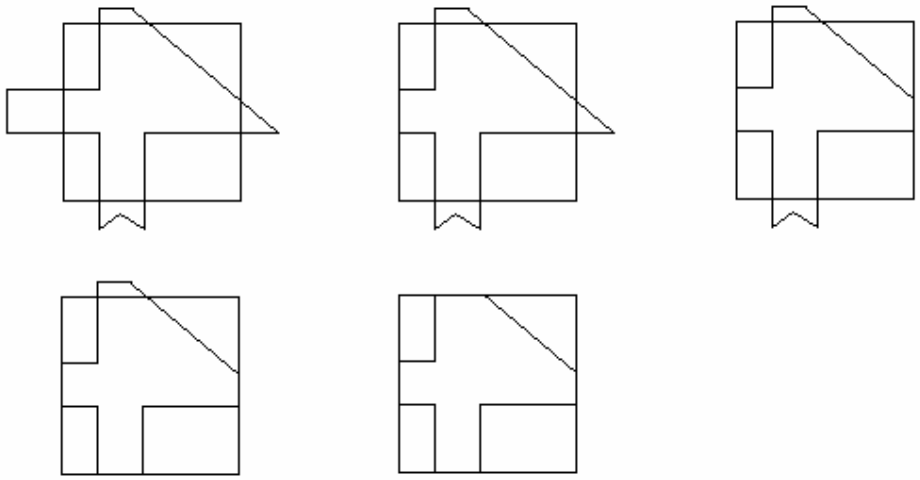

Fig-6.8: Sutherland-Hodgman Polygon Clipping

THE ALGORITHM:

We shall take the vertex of the polygon in a sequence. Let P_1, P_2 …….. P_n to the n vertices of the polygon in specific sequence. Consider the left boundary of the clip window. Let us consider the line $\overline{P_{k-1}\ P_k}$ of the polygon ($\overline{P_{k-1}\ P_k}$ is $\overline{P_1\ P_2}$, $P_2\ P_3$, ……. $\overline{P_{n-1}\ P_n}$ and $\overline{P_n\ P_1}$). There are four possibilities with this line.

Case 1 : If P_{k-1} is outside and P_k is inside i.e., P_{k-1} is in the right hand side of the window boundary, then calculate the intersection of the line $P_{k-1}\ P_k$ with the extended left window boundary. Let this be Q_k include this intersection point Q_k and P_k in the updated polygon vertex list.

Case 2 : If both P_{k-1} and P_k are inside (i.e., the right hand side of the left boundary), include only by P_k in the vertex list.

Case 3 : If P_{k-1} is inside and P_k is outside then the point of intersection of $P_{k-1}\ P_k$ and the window boundary is added to the polygon vertex list.

Case 4 : If both P_{k-1} and P_k falls out side the clip window, nothing is add to the output list. Figure given below illustrates the four cases.

In this way, Clipping right left window boundary is completed and the new resultant polygon then considered for clipping with right boundary. Then bottom and finally top.

In all of the cases the above four cases will occur, but the inside test will be different for right boundary, inside means left of the boundary and outside means right side of the boundary for bottom, inside means above the line and outside means below the line and so on.

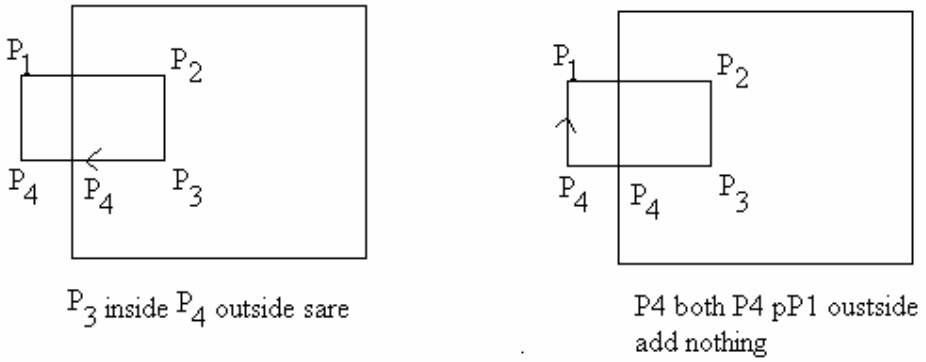

Fig-6.9: The sequence of Inside to outside lines

The Sutherland-Hodgeman algorithm is good but it has some unwanted effects. It generates some extra edges for non-convex polygons. Following is an example of that.

Consider the following figure

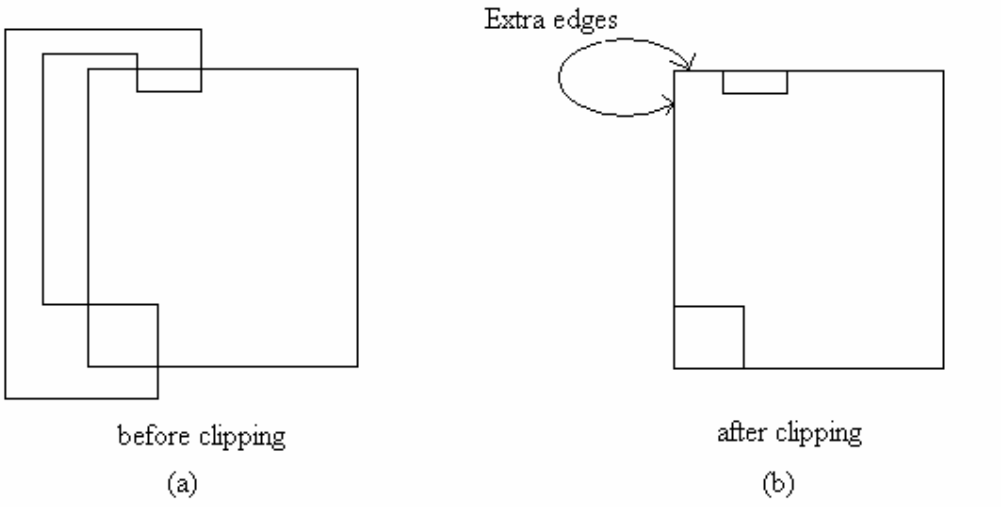

Fig-6.10: Unwanted extra edges in Sutherland-Hodgeman Process

In the figure 6.10 (b) two extra edges has been generated and they are drawn twice. In general, if there are separate parts after clipping, the parts will be connected by extra edges drawn twice in opposite directions. A post-processing is needed to eliminate them which is beyond the scope of this study material.

6.5.2 WEILER-ATHERTON POLYGON CLIPPING:

This polygon clipping algorithm clips the polygon correctly without any extra edges in case of non-convex polygons. In this algorithm the idea is somewhat like this. Instead of processing always the next vertex point, some times we shall follow the clip window boundaries. Which path we shall follow will depend on the direction of the polygon-processing i.e, clockwise or anticlockwise and whether the processing pair of vertices are inside to outside or outside to inside. For clockwise processing, we are the following two cases.

Case 1: For out-side to inside trace, follow the polygon edge.

Case 2: For inside to outside trace follow the window boundary in clockwise direction. For example of we take the following figure, the Weiler-Atherton algorithm will correctly clip the polygon.

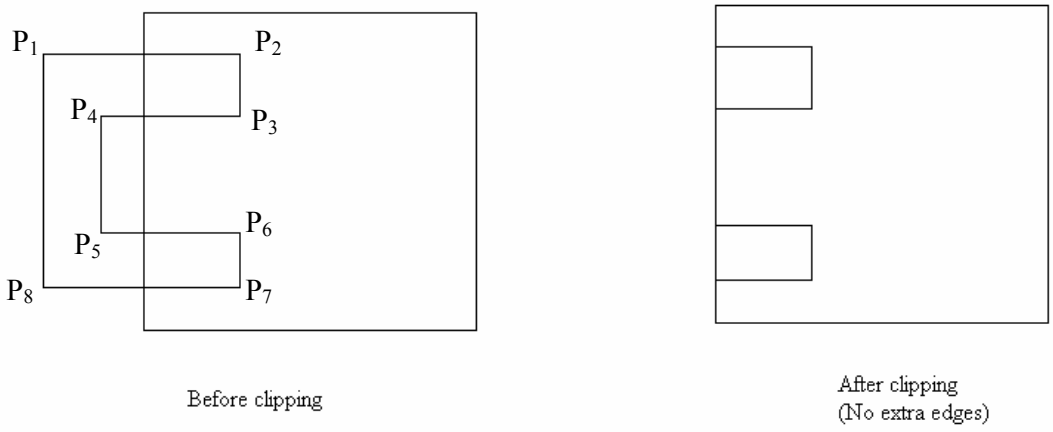

Fig-6.11: No Extra edges in case of Weiler-Atherton process

6.6 PROGRAMS FOR CLIPPING:

Following are few programs for line clippings. These programs are written in C language.

6.6.1 PROGRAM FOR COHEN-SUTHERLAND LINE CLIPPING:

```
/*Cohen-Sutherland Line Clipping*/

#include<graphics.h>
#include<conio.h>
#define LEFT_BOUNDARY 0x1
#define RIGHT_BOUNDARY 0x2
#define BOTTOM_BOUNDARY 0x4
#define TOP_BOUNDARY 0x8
struct pt
 {
  float x;
  float y;
 };
typedef struct pt point;
unsigned char assignCode(point p, point windowMin,point windowMax)
 {
  unsigned char clipCode=0x00; //Assigned hex 00h
  if(p.x<windowMin.x)
   clipCode|=LEFT_BOUNDARY;
  if(p.x>windowMax.x)
   clipCode|=RIGHT_BOUNDARY;
  if(p.y<windowMin.y)
   clipCode|=BOTTOM_BOUNDARY;
  if(p.y>windowMax.y)
   clipCode|=TOP_BOUNDARY;
  return(clipCode);
 }
void Cohen_Sutherland_lineClip(point windowMin,point windowMax,point p1,point p2)
 {
  unsigned char codeX,codeY;
  point tmp;
  int complete=0,draw=0;
  float slope;
  while(complete==0)
   {
```

```
codeX=assignCode(p1,windowMin,windowMax);
codeY=assignCode(p2,windowMin,windowMax);
if((codeX==0x00) && (codeY==0x00))
 {
  complete=1;
  draw=1;
 }
else
  if((codeX&codeY)!=0x00)
   complete=1;
  else
   {
        if(codeX==0x00)
         {
          tmp=p1;p1=p2;p2=tmp;
          codeX^=codeY^=codeX; /*Swaping using Bitwise Operators*/
         }
        if(p2.x!=p1.x)
         slope=(p2.y-p1.y)/(p2.x-p1.x);
        if(codeX&LEFT_BOUNDARY)
         {
          p1.y+=(windowMin.x-p1.x)*slope;
          p1.x=windowMin.x;
         }
        else
         if(codeX&RIGHT_BOUNDARY)
          {
           p1.y+=(windowMax.x-p1.x)*slope;
           p1.x=windowMax.x;
          }
         else
          if(codeX&BOTTOM_BOUNDARY)
           {
                if(p2.x!=p1.x)
                 p1.x+=(windowMin.y-p1.y)/slope;
                p1.y=windowMin.y;
           }
          else
           if(codeX&TOP_BOUNDARY)
                {
```

```
                        if(p2.x!=p1.x)
                          p1.x+=(windowMax.y-p1.y)/slope;
                        p1.y=windowMax.y;
                       }
                  }
                 }

               if(draw)
                 line(p1.x+0.5,p1.y+.5,p2.x+0.5,p2.y+0.5);
  }

int main()
  {
   int gd=DETECT,gm;
   point winUp,winBottom,p1,p2;
   winUp.x=400;winUp.y=300;winBottom.x=200;winBottom.y=100;
   p1.x=150;p1.y=150;p2.x=450;p2.y=400;
   initgraph(&gd,&gm,"f:\\tc\\bgi");
   setcolor(GREEN);
   line(p1.x,p1.y,p2.x,p2.y);
   getch();
   setcolor(YELLOW);
   rectangle(winUp.x,winUp.y,winBottom.x,winBottom.y);
   getch();
   setcolor(BLACK);
   line(p1.x,p1.y,p2.x,p2.y);
   setcolor(YELLOW);
   rectangle(winUp.x,winUp.y,winBottom.x,winBottom.y);
   setcolor(GREEN);
   Cohen_Sutherland_lineClip(winBottom,winUp,p1,p2);
   getch();
   closegraph();
   restorecrtmode();
   return(0);
  }
```

6.7 KEY WORDS

- Device Coordinate
- Line Clipping
- NDCS
- View port
- Viewing
- Window
- Window-to-View port Mapping
- World Coordinate

6.8 SAMPLE QUESTIONS:

6.8.1 Discuss the following terms

i. Window

ii. Viewport

6.8.2 What is viewing pipeline? Explain.

6.8.3 What do you mean by Clipping? Discuss the point clipping technique.

6.8.4 Discuss the Cohen-Sutherland line clipping technique.

6.8.5 Write a program for Cohen-Sutherland line clipping.

6.8.6 Discuss the Liang-Barsky line clipping technique.

6.8.7 What is polygon clipping? Discuss Sutherland-Hodgeman polygon clipping. What is the disadvantage of this polygon clipping?

6.8.8 Discuss the Weiler-Atherton polygon clipping algorithm