CHAPTER FIVE

5. CLIPPING

5.1. Introduction & Terminologies

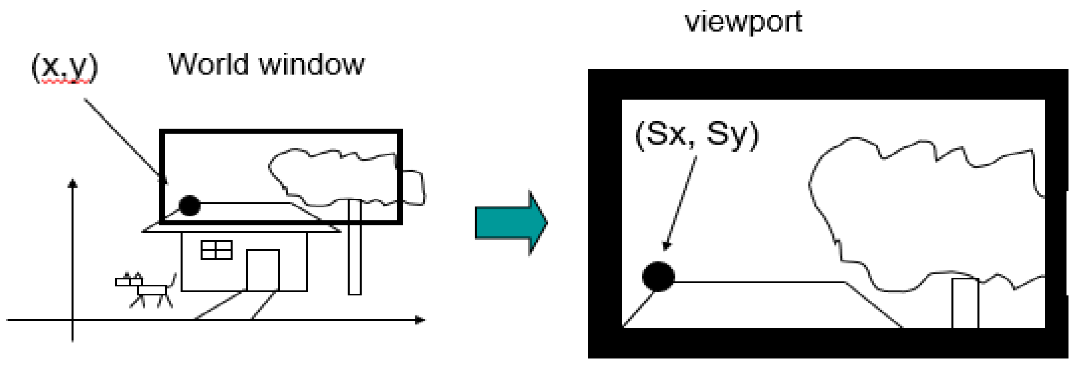
(World) Window: is the rectangle defining the part of the world we wish to display. It is a rectangular region in the world that is to be displayed. A window is a rectangular area specified in world coordinates for displaying images. It is a rectangular area used to select the portion of the scene for which an image is to be generated.

(Screen) Window: is the visual representation of the screen coordinate system for "windowed" displays (coordinate system moves with screen window).

Viewport: is the rectangle within the screen window defining where the image will appear (Default is usually entire interface window). It is the rectangular region defined in the screen coordinate system for displaying the graphical objects defined in the world window.

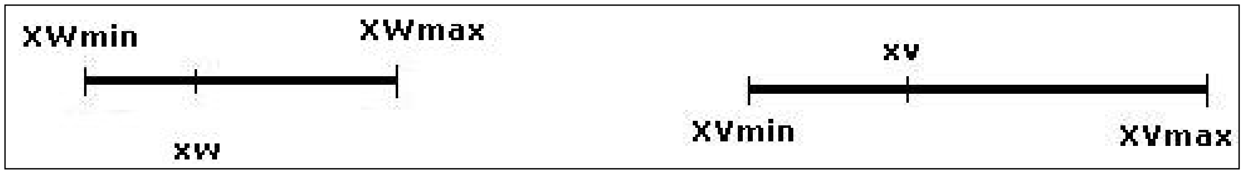
Window-Viewport-Transformation: is the process of mapping from a window in world coordinates to a viewport in screen coordinates. It is also called viewing.

In a window to viewport mapping (viewing), we map a part of the world coordinates scene to device coordinates.

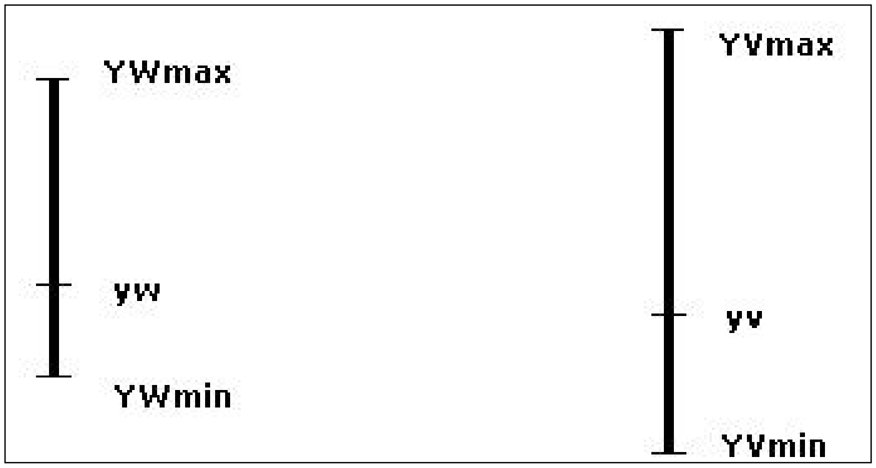


The objective of window-to-viewport mapping is to convert the world coordinates (xw, yw) of an arbitrary point to its corresponding normalized device coordinates (xv, yv). To achieve this objective, we need to preserve proportionalities in X and Y directions.

- ❖ for proportionality in x direction



- ❖ for proportionality in y direction



⇨ In order to maintain the same relative placement of the point in the viewport as in the window, the following ratios must be equal:

- ✓ In the x direction

$$\frac{xw - xw_{min}}{xw_{max} - xw_{min}} = \frac{xv - xv_{min}}{xv_{max} - xv_{min}}$$

- ✓ And, in the y direction

$$\frac{yw - yw_{min}}{yw_{max} - yw_{min}} = \frac{yv - yv_{min}}{yv_{max} - yv_{min}}$$

By solving these equations for the unknown viewport position (xv, yv), the following becomes true:

$$\boxed{\begin{aligned} xv &= \left(\frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}\right)\left(xw - xw_{min}\right) + xv_{min} \\ yv &= \left(\frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}\right)\left(yw - yw_{min}\right) + yv_{min} \end{aligned}}$$

Or, it can be expressed in the following way.

$xv = S_x . xw – S_x . xw_{min} + xv_{min}$

$yv = S_y . yw – S_y . yw_{min} + yv_{min}$ where

$$\boxed{S_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}} \quad \text{and} \quad \boxed{S_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}}$$

❖ We can express the above two formulas for computing (xv, yv) from (xw, yw) interms of a translate-scale-translate transformation: i.e.

$T(-xw_{min}, -yw_{min}) \rightarrow S(S_x, S_y) \rightarrow T(xv_{min}, vy_{min})$

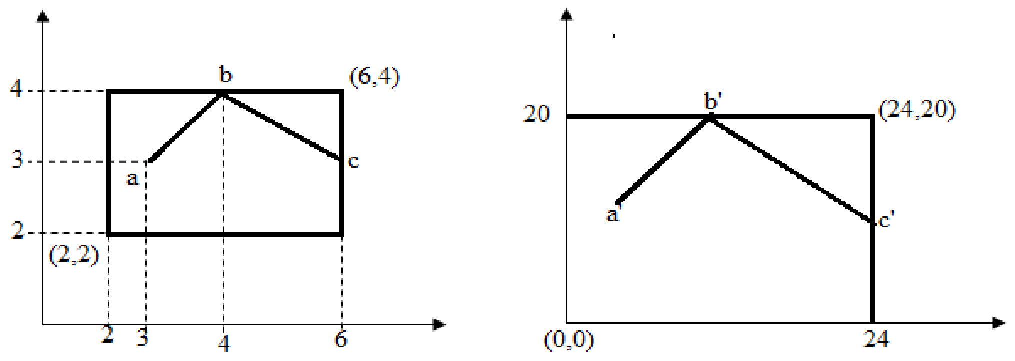
This results the following general matrix representation for mapping.

$$\begin{pmatrix} xv \\ yv \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & -Sx.XWmin+XVmin \\ 0 & Sy & -Sy.YWmin+YVmin \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} xw \\ yw \\ 1 \end{pmatrix}$$

Exercise:

Compute for finding the above viewing transformation matrix.

Example: Consider the following diagram for a certain viewing:



Then, find the coordinates of a’, b’ & c’?

Solution:

First, find S_x and S_y as follows using the previous computations.

$$S_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = \frac{24 - 0}{6 - 2} = \frac{24}{4} = 6$$

$$S_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} = \frac{20 - 0}{4 - 2} = \frac{20}{2} = 10$$

Then, based on the general matrix representation format, the new coordinates are

$$\begin{pmatrix} a' & b' & c' \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & -Sx.XWmin+XVmin \\ 0 & Sy & -Sy.YWmin+YVmin \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 4 & 6 \\ 3 & 4 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 6 & 0 & -6(2)+0 \\ 0 & 10 & -10(2)+0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 4 & 6 \\ 3 & 4 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 6 & 0 & -12 \\ 0 & 10 & -20 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 4 & 6 \\ 3 & 4 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 6 & 12 & 24 \\ 10 & 20 & 10 \\ 1 & 1 & 1 \end{pmatrix}$$

Hence, the required coordinates are a' (6, 10), b'(12, 20) & c'(24, 10).

5.2. Types of Clipping Algorithm

Clipping is a procedure that identifies the portions of a picture that are either inside, outside or partially of a specified region of space. The region against which an object is to be clipped is called clipping window, which is usually a rectangular shape.

The clipping operation eliminates objects or portions of objects that are not visible through the window to ensure the proper construction of the corresponding image. It is the process of removing unwanted part(s) of an object (image).

The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane. The viewing transformation is insensitive to the position of points relative to the viewing volume − especially those points behind the viewer − and it is necessary to remove these points before generating the view.

Depending on when clipping takes place relative to *scan conversion* we can identify 3 different types of clipping algorithm. (Scan conversion here refers to the digitising of picture information for storage in the frame buffer.)

❖ Clipping Before Scan Conversion

The most common approach is to perform clipping before scan conversion. This is known as analytical clipping. Analytical clipping algorithms operate on object primitives such as points, lines and polygons, and this is the most efficient type of clipping for such primitives. For the rest of this chapter we will concentrate on analytical clipping algorithms.

❖ Clipping During Scan Conversion

Clipping during scan conversion is known as scissoring. In this case, scan conversion is performed for each primitive, but before the pixel values are stored in the frame buffer a decision is made as to whether or not the pixel should be clipped. As such, we can view scissoring as a "brute-force" technique, since it scan converts every primitive, regardless of whether it lies inside the view volume or not. Generally this approach is less efficient than analytical clipping. However, because we do not need to perform any processing before scan conversion, scissoring can be more efficient than analytical clipping if the primitive lies mostly inside the clipping region.

❖ Clipping After Scan Conversion

Clipping can occur after scan conversion if we first scan convert all primitives into a temporary canvas and then copy the pixels inside the clipping region from the temporary canvas into the actual frame buffer. Again, this is a "brute force" technique because every primitive is drawn regardless of whether it lies inside the view volume. Therefore generally this is a less efficient technique for primitives such as points, lines and polygons. However, it is a simple and easy approach, and it is sometimes used when clipping text primitives.

5.3. 2-D Clipping

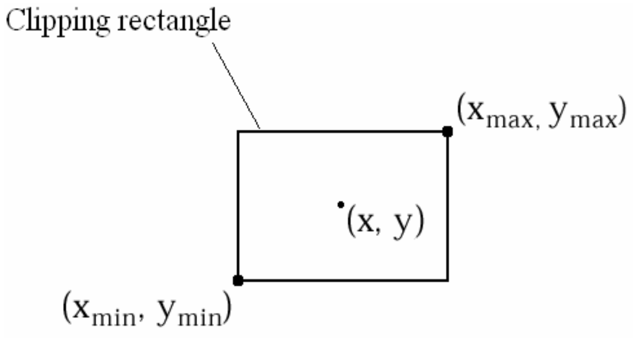
Now we will examine a number of analytical clipping techniques for different types of primitives. Clipping can be done in 2-D or 3-D, depending on whether we are using 2-D or 3-D coordinates. In this section we will examine some techniques for performing 2-D clipping, but many 2-D clipping algorithms are generalisable to 3-D. When performing clipping in 2-D we need to consider four clipping planes: left, right, bottom and top. In the following sections we will examine some algorithms for point clipping, line clipping and polygon clipping.

5.3.1. 2-D Point Clipping

Many graphics packages include point primitives, and points can be used to construct higher-level primitives such as lines or polygons. Therefore it is important to have an efficient technique for clipping points against a clipping rectangle. This is the simplest type of clipping, and it can be performed by testing the following inequalities:

- $x_{min} \le x \le x_{max}$
- $y_{min} \le y \le y_{max}$

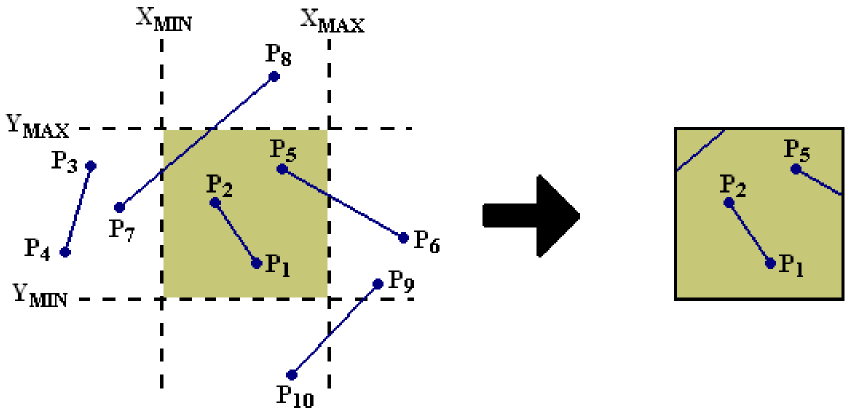
If any of these inequalities is not satisfied, then the point is rejected. If all are satisfied then the point is saved. This is illustrated in the following figure.



5.3.2. 2-D Line Clipping

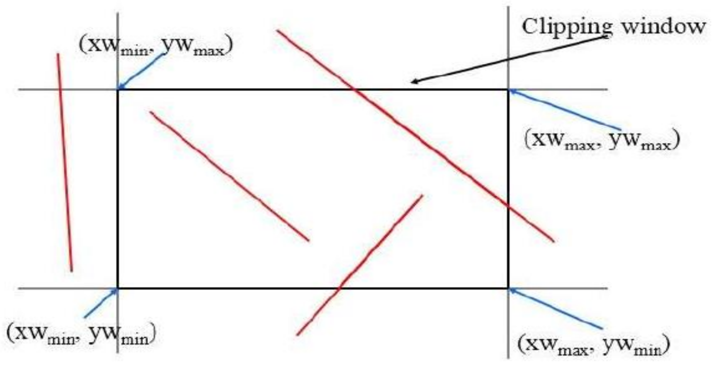
With point primitives there were only two possible outcomes of the clipping process: *accept* or *reject* the point. For other primitives we have three possible outcomes: *accept* the entire primitive, *reject* the entire primitive, or *clip* the primitive so that part of it is accepted and part rejected.

Line primitives can be represented by the coordinates of their two end-points. Depending on the positions of these end-points relative to the four clipping boundaries we may be able to make an immediate decision as to which of the three outcomes is appropriate. Consider the following figure. This shows a number of possible positions for line primitives relative to four clipping boundaries. For some of these lines (e.g. P_1P_2) we should be able to decide straight away that the entire primitive should be accepted. For others (e.g. P_3P_4) we can decide to reject the entire primitive. To make a decision for the other lines we need to do some more work.

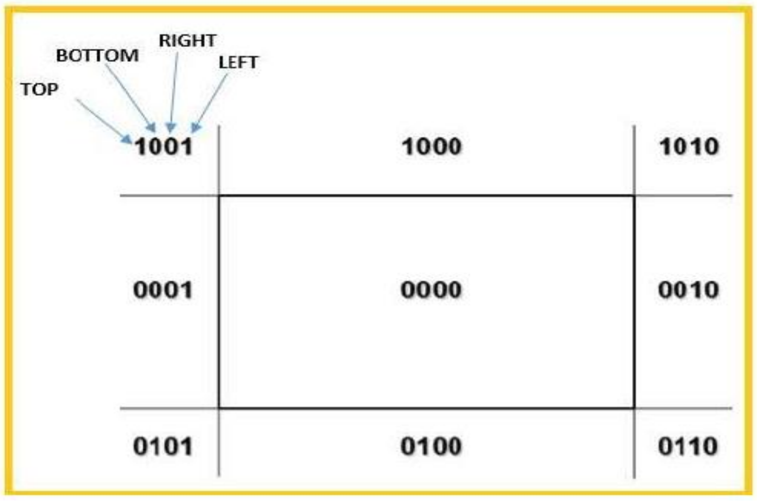


Cohen-Sutherland Line Clipping

This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XW_{min}, YW_{min}) and the maximum coordinate for the clipping region is (XW_{max}, YW_{max}).



The Cohen-Sutherland line clipping algorithm is one of the most popular algorithm for line clipping. This algorithm uses 4-bits to divide the entire region, to indicate which of the nine regions contains the end point of a line. These codes identify the location of the point relative to the boundaries of the clipping window. These 4 bits represent the Top, Bottom, Right, and Left [4321 or TBRL] of the region as shown in the following figure.



- ❖ The left side bit (bit 1, L) is set to 1 if the end point is at the left side of the clipping window, otherwise it is set to 0.
- ❖ Bit 2 (R) is set 1 if the end point lies at the right side of the window, otherwise it is set to 0.
- ❖ Bit 3 (B) is set 1 if the end point lies at the bottom side of window, otherwise it is set to 0.
- ❖ Bit 4 (T) is set 1 if the end point lies at the top side of the clipping window, otherwise it is set to 0

There are 3 possibilities for the line:-

- ➔ Line can be completely inside the window (This line should be accepted).
- ➔ Line can be completely outside of the window (This line will be completely removed from the region).
- ➔ Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

The algorithm for Cohen-Sutherland line clipping is the following.

Step 1: Read the end points of a line as $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$

Step 2: Read the clipping window coordinates as (Wx_{min}, Wy_{min}) and (Wx_{max}, Wy_{max})

Step 3: Take a binary codes (region codes) from where a line end points are located

Step 4: check visibility of the line

a) If binary code for both end points P_1 and P_2 are zero (0000), then the line is completely inside window, and hence it is visible.

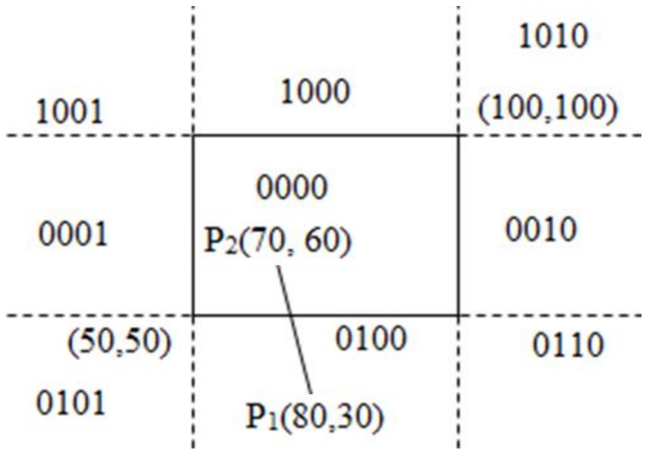
b) Else take the logical AND operation of both binary codes. If the result is not zero, then the line is completely outside window and hence it is invisible.

c) Else determine intersection point(s) with the window sides using the following formula:

- For left boundary $y = y_1 + m (Wx_{min} – x_1)$; and $x = Wx_{min}$
- For right boundary $y = y_1 + m (Wx_{max} – x_1)$; and $x = Wx_{max}$
- For bottom boundary $x = x_1 + \frac{Wy_{min} - y_1}{m}$; and $y = Wy_{min}$
- For top boundary $x = x_1 + \frac{Wy_{max} - y_1}{m}$; and $y = Wy_{max}$

Where m is slope of the line: $m = \frac{y_2 - y_1}{x2 - x1}$

Example: Clip the line segment $P_1(80, 30)$ and $P_2(70, 60)$ against a window $(Wx_{min}, Wy_{min}) = (50,50)$ & $(Wx_{max}, Wy_{max}) = (100,100)$.



<u>Solution</u>:

1. Find the binary code of p_1 and p_2

   $p_1 = 0100$

   $p_2 = 0000$

1. p_1 is different from zero $\Rightarrow$ p_1 is not completely inside clipping window.
2. Making logical **AND** operation (0100 AND 0000), result = zero (0000)

   ➔ The result of the logical AND operation is zero, therefore the line is not completely outside of clipping window.

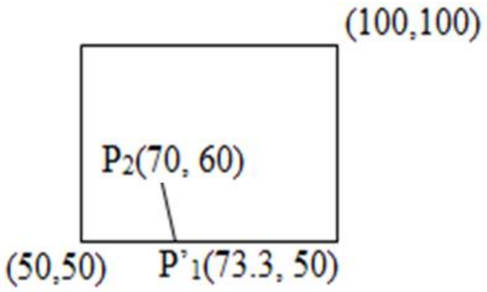
   Slope(m) = $(y_2-y_1)/(x_2-x_1) = (60-30)/(70-80) = 30/-10 = -3$

3. Take the equation for bottom boundary

$$x' = x_1 + (Wy_{min} - y_1)/m$$
$$= 80 + (50 - 30)/-3 = 73.3$$
$$y' = W_{ymin} = 50$$

So, the intersection point between the line and the clipping window is **(73.3, 50)**

After the clipping process is done the line looks like as follow:



5.3.3. 2-D Polygon Clipping

The first stage in polygon clipping is to detect the trivial accept and trivial reject cases. We could do this by performing the Cohen-Sutherland line test for each edge in the polygon individually – if all are accepted or all rejected then we can accept/reject the entire polygon. However, this would be time-consuming, so the normal approach is to use a *bounding box* that encloses the polygon. For example, the following figure shows the detection of trivial accept and trivial reject cases for a polygon. This technique is preferred because calculating a bounding box and testing its coordinate extents against those of the clipping rectangle is fast and easy.

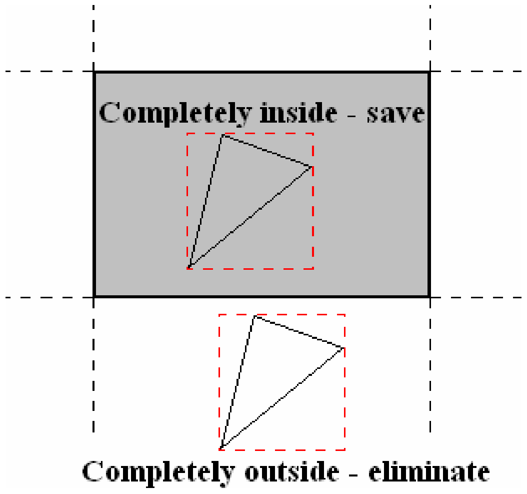
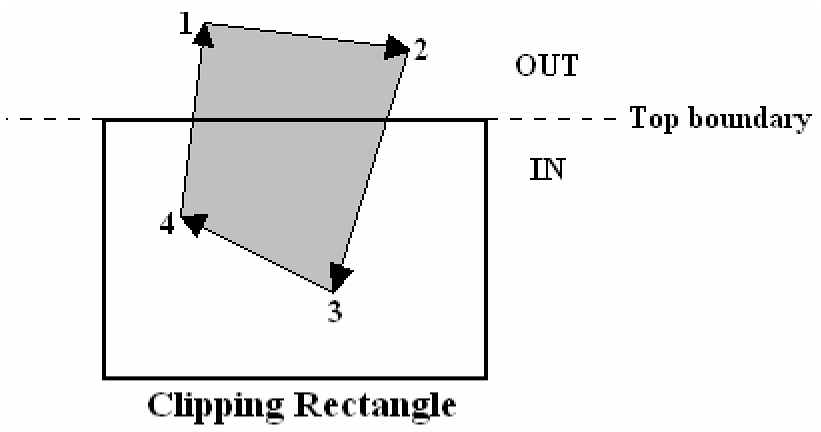


Figure: Determining trivial accept and trivial reject cases for polygon clipping

Sutherland-Hodgman Polygon Clipping

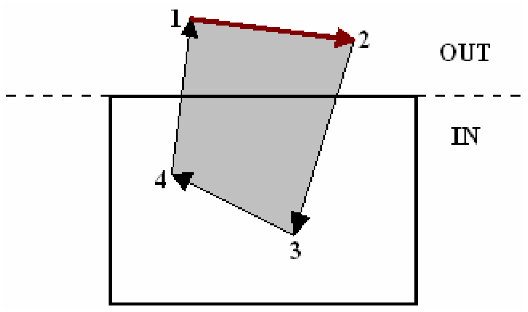
If we cannot accept or reject the entire polygon, we must clip it. A common technique for polygon clipping is the Sutherland-Hodgman algorithm. The first stage in this algorithm is to categorise each of the edges in the polygon according to whether it is entirely inside, entirely outside, entering or leaving the clipping rectangle. This should be done separately for each clipping boundary.



Referring to the above figure, we can see that for the top clipping boundary 4 types of edge can be identified:

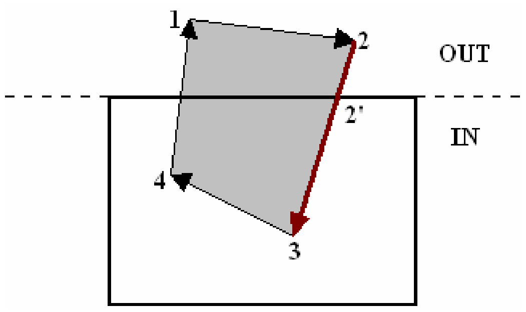
- OUT-OUT: The edge from vertex 1 to 2 is entirely outside the clipping rectangle.
- OUT-IN: The edge from vertex 2 to vertex 3 starts outside the clipping rectangle and finishes inside it.
- IN-IN: The edge from vertex 3 to 4 is entirely inside the clipping rectangle.
- IN-OUT: The edge from vertex 4 to 1 starts inside the clipping rectangle and finishes outside it.

Next, for each clipping boundary, we process each edge in turn. We categorise it according to the 4 categories listed above, and produce a set of output vertices for each edge according to its category. The input for each edge is the two vertices that define the edge. The following figure illustrates the output that is generated for each of the 4 categories of edge. For OUT-OUT edges no output is produced. For OUT-IN edges we compute the intersection of the edge with the clipping boundary and then output the intersection point followed by the second vertex. For IN-IN edges we output the second of the input vertices only. For IN-OUT edges we compute and return only the intersection point. For the example shown in the figure the final output is the sequence of vertices 2', 3, 4, 4', which represents the correct polygon clipped against the top clipping boundary. This process should be repeated for each of the four boundaries.



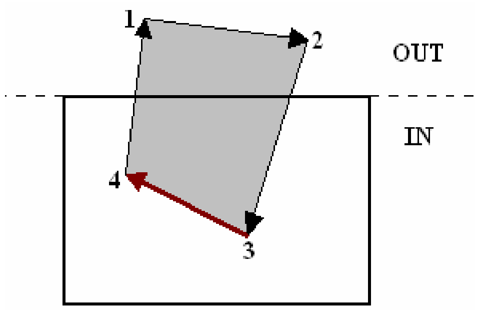
OUT-OUT: Output = none

(a)



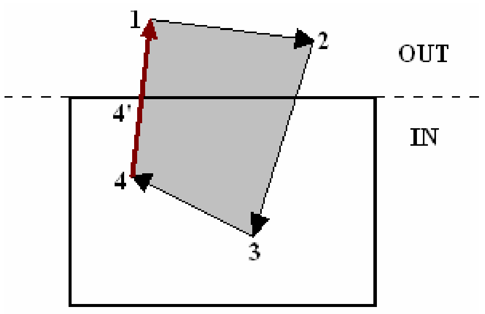
OUT-IN: Output = 2', 3

(b)



IN-IN: Output = 4

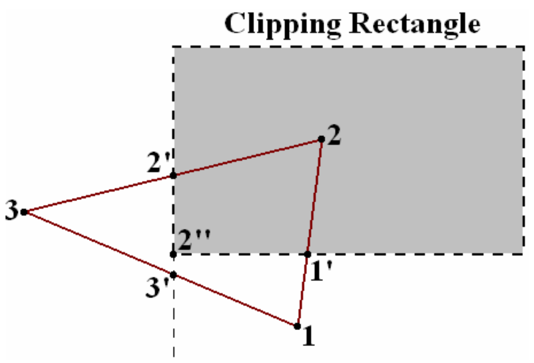
(c)



IN-OUT: Output = 4'

(d)

One appealing factor about the Sutherland-Hodgman algorithm is its potential for parallelism. We can think of the algorithm as consisting of four clipping processes, one for each clipping boundary. As soon as two vertices have been produced as the output from the first clipping process, they can be fed in to the next clipping process, without waiting for the full set of output vertices to be produced. The following figure shows an example application of Sutherland-Hodgman polygon clipping. After the first two edges have been processed by the left clipper, we have a current output of [2, 2']: this forms the first edge fed into the right clipper. When another vertex is available from the left clipper (3'), we form a new edge ([2', 3']) and feed it into the right clipper. As soon as two edges are available from the output of the right clipper, we form a new edge [2', 3'] and feed it into the bottom clipper, and so on. For polygons with many edges this can produce a significant improvement in efficiency, because all four clippers can be operating at the same time.

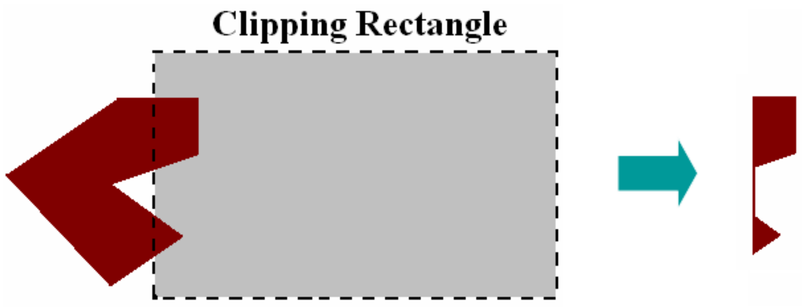


(a)

Left Clipper	Right Clipper	Bottom Clipper	Top Clipper
[1,2] - IN-IN - [2]			
[2,3] - IN-OUT - [2’]	[2,2’] - IN-IN - [2’]		
[3,1] - OUT-IN - [3’,1]	[2’,3’] - IN-IN - [3’]	[2’,3’] - IN-OUT - [2’’]	
	[3’,1] - IN-IN - [1]	[3’,1] - OUT-OUT - [ ]	
	[1,2] - IN-IN - [2]	[1,2] - OUT-IN - [1’,2]	[2’’,1’] - IN-IN - [1’]
		[2,2’] - IN-IN - [2’]	[1’,2] - IN-IN - [2]
			[2,2’] - IN-IN - [2’]
			[2’,2’’] - IN-IN - [2’’]

(b)

One potential limitation with the Sutherland-Hodgman algorithm is that it always produces a single polygon as its output. This can cause problems if we are trying to clip a *concave* polygon. The following figure shows what can happen if concave polygons have two areas of overlap with the clipping rectangle: the output should consist of two separate polygons but the Sutherland-Hodgman algorithm joins them together to form a single linked polygon, which is not the desired result. Such problems do not occur with convex polygons. Although this is a weakness of the technique, we should remember that most graphics packages do not permit concave polygons for reasons related to rendering speed. Therefore, the Sutherland-Hodgman algorithm (or variations of it) is still commonly used by graphics packages.



5.4. 3-D Clipping

All of the techniques described in this chapter are easily generalisable to 3-D. Recall that in 3-D graphics we have an extra two clipping planes to process: the near and far clipping planes. For clipping techniques that use the 4-bit binary region code we simply add an extra 2 bits for the extra clipping planes. The following figure illustrates the 27 regions that are produced using this extension: 9 in front of the near clipping plane, 9 between the near and far clipping planes and 9 beyond the far clipping plane.

5.4.1. 3-D Point Clipping

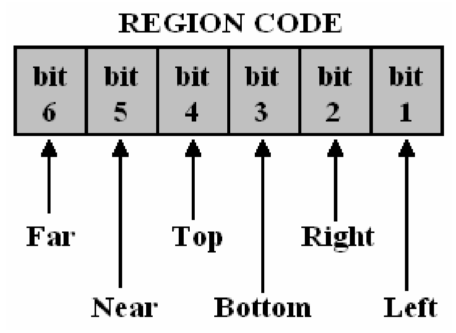
Now 3-D point clipping is just a matter of testing the region code of each point to see if it is equal to 000000. If it is not equal to 000000 then the point should be rejected, otherwise it will be accepted.

5.4.2. 3-D Line Clipping

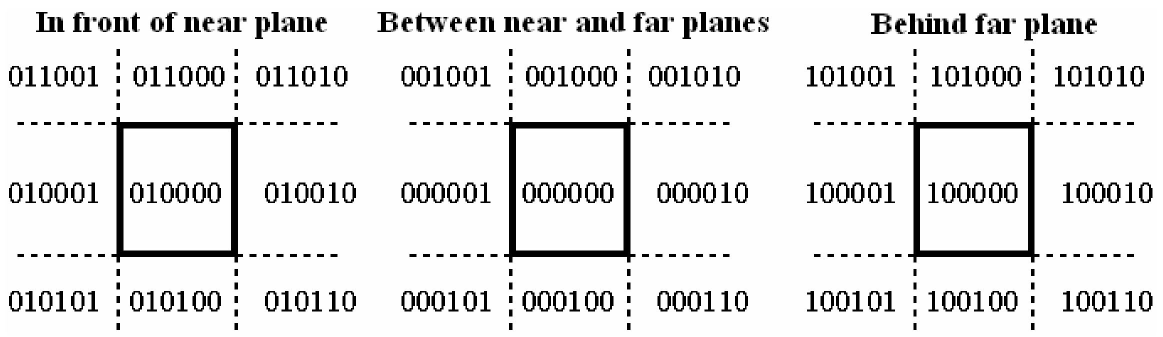
The Cohen-Sutherland line clipping algorithm can be easily extended to 3-D. The trivial accept and trivial reject cases can be detected in a similar way to the 2-D case:

- If C_{start} OR C_{end} = 000000, we perform *trivial accept*
- If C_{start} AND C_{end} ≠ 000000, we perform *trivial reject*

Then the algorithm proceeds with the divide-and-conquer technique, dividing the line at its intersection points with each of six clipping boundaries (instead of four). As in the 2-D case, we only compute intersections for planes where the appropriate bit in the XOR of the regions codes is equal to 1.



(a)



(b)

Figure - Binary Region Codes for 3-D Clipping

5.4.3. 3-D Polyhedron Clipping

Most 3-D graphics objects are *polyhedra* – that is, they are formed from a mesh of planar polygons. To clip such 3-D polyhedra, we can apply the Sutherland-Hodgman algorithm to each polygon in the mesh in turn. The combination of all clipped polygons will form a correct clipped polyhedron. For example, the following figureFigure illustrates a polygonal mesh being clipped against a cuboid view volume. The parts of the polyhedron shown with solid lines are clipped, whereas the portion shown using dashed lines is saved.

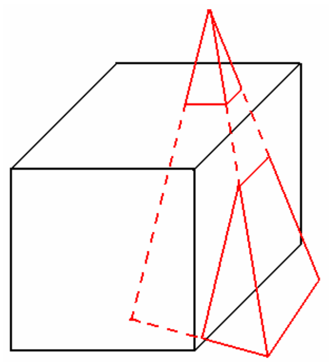


Figure: - 3-D Polyhedron Clipping