Midpoint Subdivision Algorithm :

1. Read two endpoints of the line say $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say (Wx_1, Wy_1 and Wx_2, Wy_2).
3. Assign region codes for two end points using following steps :

   Initialize code with bits 0000
   Set Bit 1 - if ($x < Wx_1$)

   Set Bit 2 - if ($x > Wx_2$)

   Set Bit 3 - if ($y < Wy_1$)

   Set Bit 4 - if ($y > Wy_2$)

4. Check for visibility of line

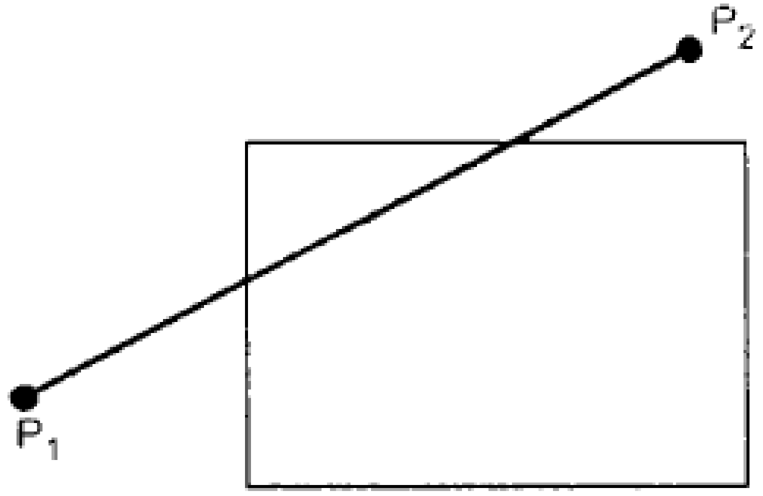
   a) If region codes for both endpoints are zero then the line is completely visible. Hence draw the line and go to step 6.

   b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 6.

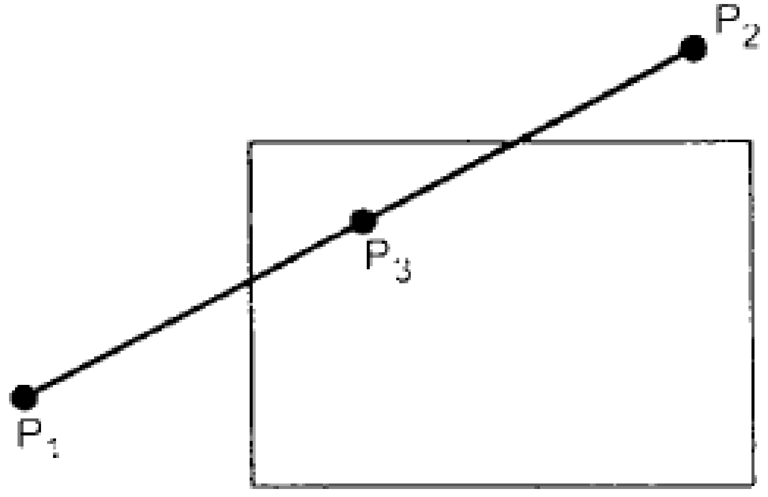
   c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.

5. Divide the partially visible line segment in equal parts and repeat steps 3 through 5 for both subdivided line segments until you get completely visible and completely invisible line segments.

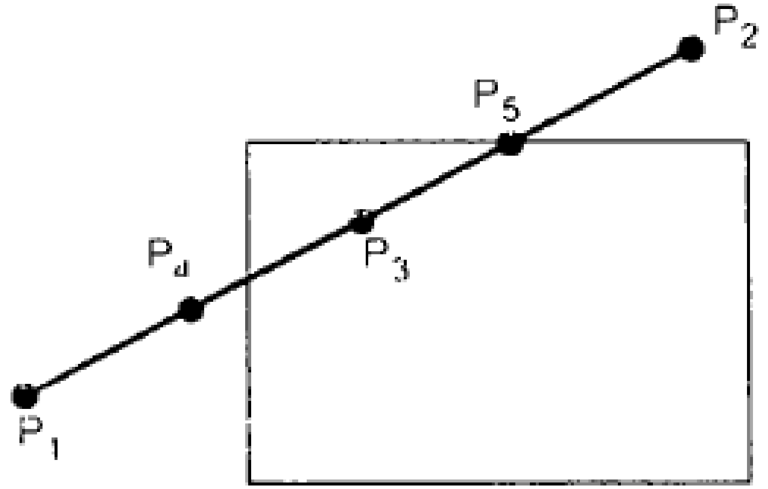
6. Stop.



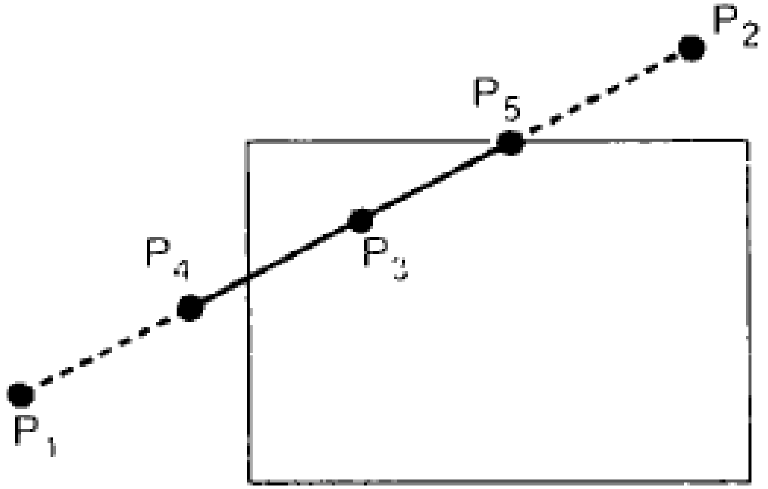
(a)



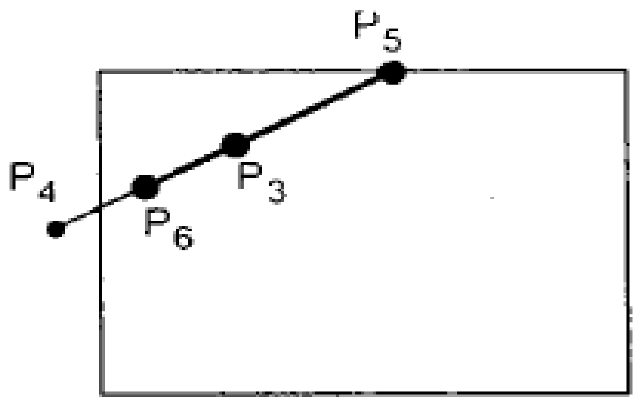
(b)



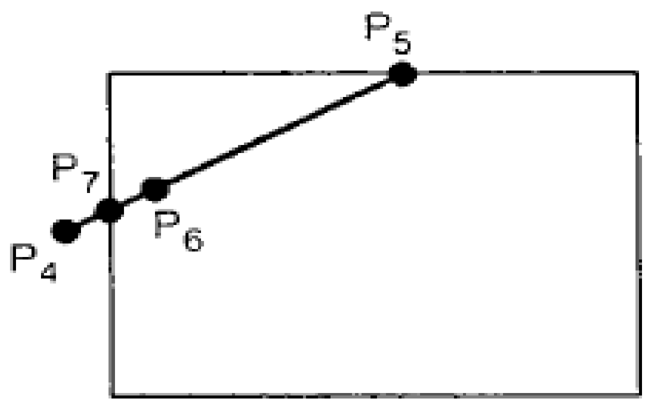
(c)



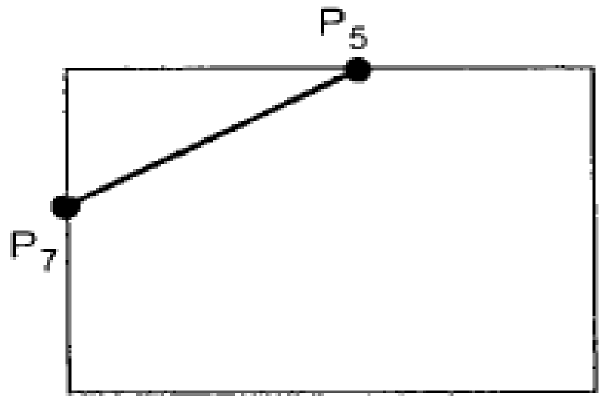
(d)



(e)



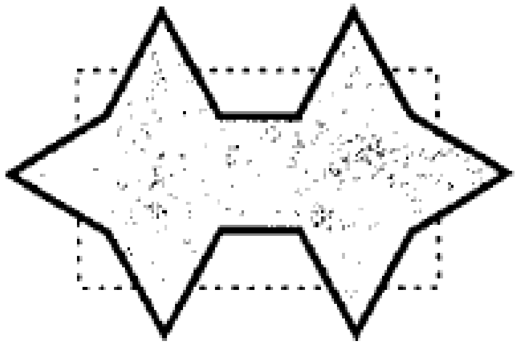
(f)

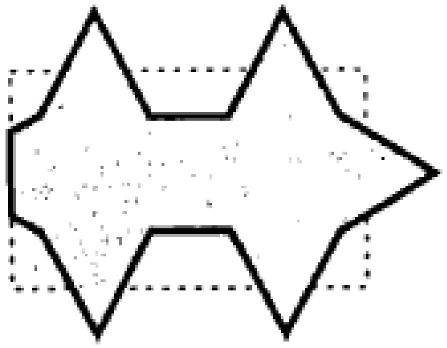


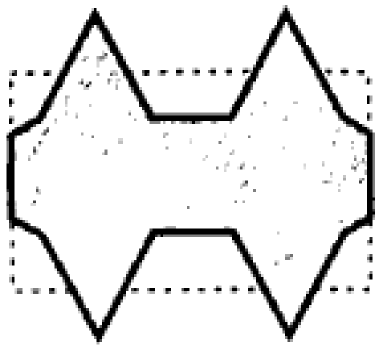
(g)

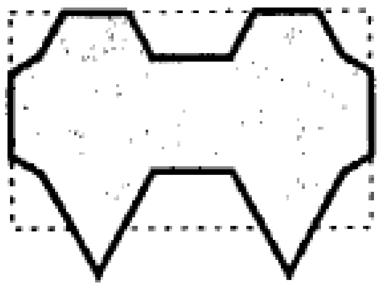
5.5 Sutherland - Hodgeman Polygon Clipping

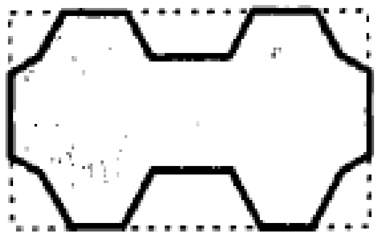
A polygon can be clipped by processing its boundary as a whole against each window edge. This is achieved by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the original set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new set of vertices could then be successively passed to a right boundary clipper, a top boundary clipper and a bottom boundary clipper, as shown in Fig. . At each step a new set of polygon vertices is generated and passed to the next window boundary clipper. This is the fundamental idea used in the Sutherland - Hodgeman algorithm.


Original polygon


Left clipped

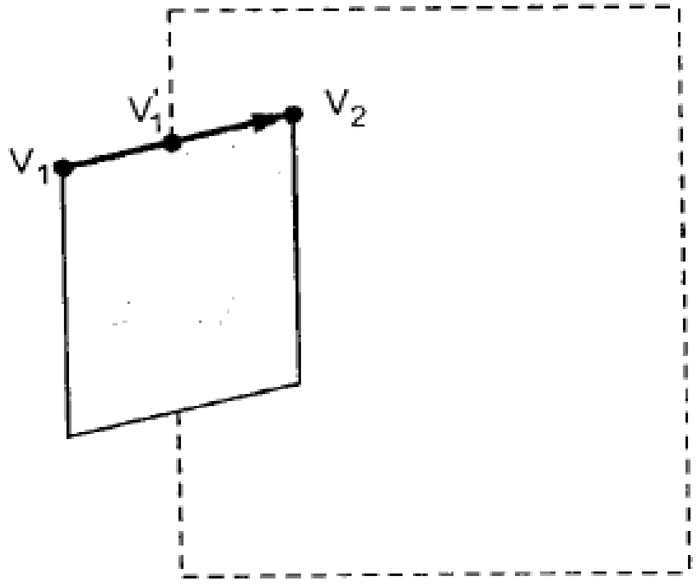

Right clipped


Top clipped


Bottom clipped

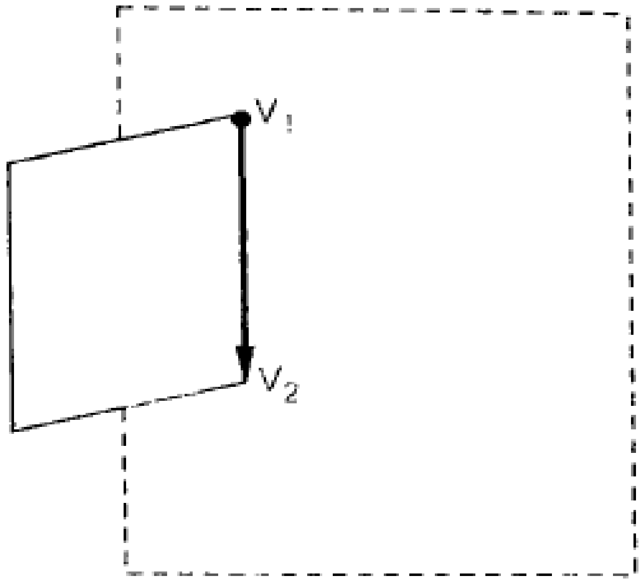
The output of the algorithm is a list of polygon vertices all of which are on the visible side of a clipping plane. Such each edge of the polygon is individually compared with the clipping plane. This is achieved by processing two vertices of each edge of the polygon around the clipping boundary or plane. This results in four possible relationships between the edge and the clipping boundary or plane.

1. If the first vertex of the edge is outside the window boundary and the second vertex of the edge is inside then the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list (See Fig.   a).
2. If both vertices of the edge are inside the window boundary, only the second vertex is added to the output vertex list. (See Fig.   b).



V_1 – Outside
V_2 – Inside
∴ Save V_1' and V_2

(a)

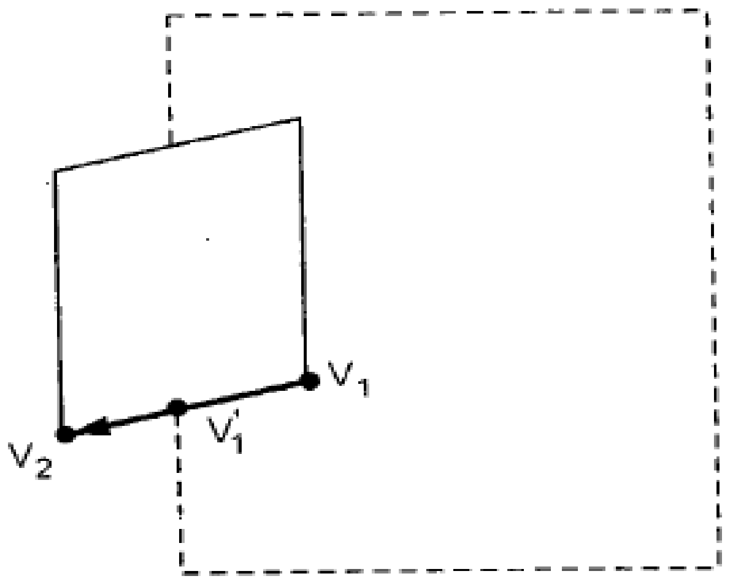


V_1 – Inside
V_2 – Inside
∴ Save V_2

(b)

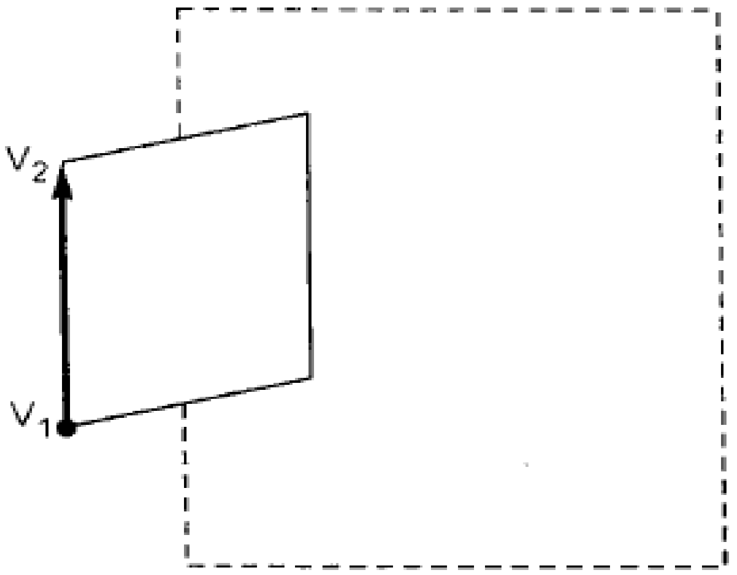
3. If the first vertex of the edge is inside the window boundary and the second vertex of the edge is outside, only the edge intersection with the window boundary is added to the output vertex list. (See Fig. c).

4. If both vertices of the edge are outside the window boundary, nothing is added to the output list. (See Fig. d).



V_1 – Inside
V_2 – Outside
∴ Save V'_1

(c)



V_1 – Outside
V_2 – Outside
∴ Save nothing

(d)

Sutherland-Hodgeman Polygon Clipping Algorithm

1. Read coordinates of all vertices of the polygon.
2. Read coordinates of the clipping window
3. Consider the left edge of the window
4. Compare the vertices of each edge of the polygon, individually with the clipping plane
5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary discussed earlier.
6. Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

The Sutherland-Hodgeman polygon clipping algorithm clips convex polygons correctly, but in case of concave polygons, clipped polygon may be displayed with extraneous lines, as shown in Fig.

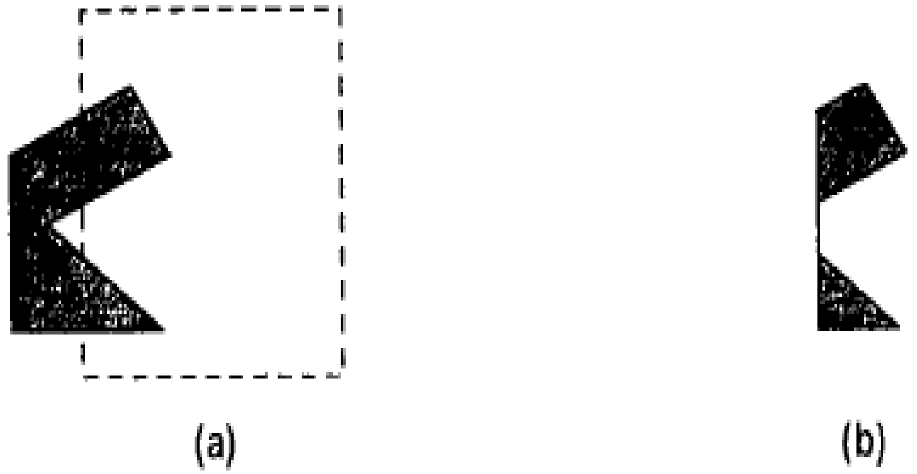
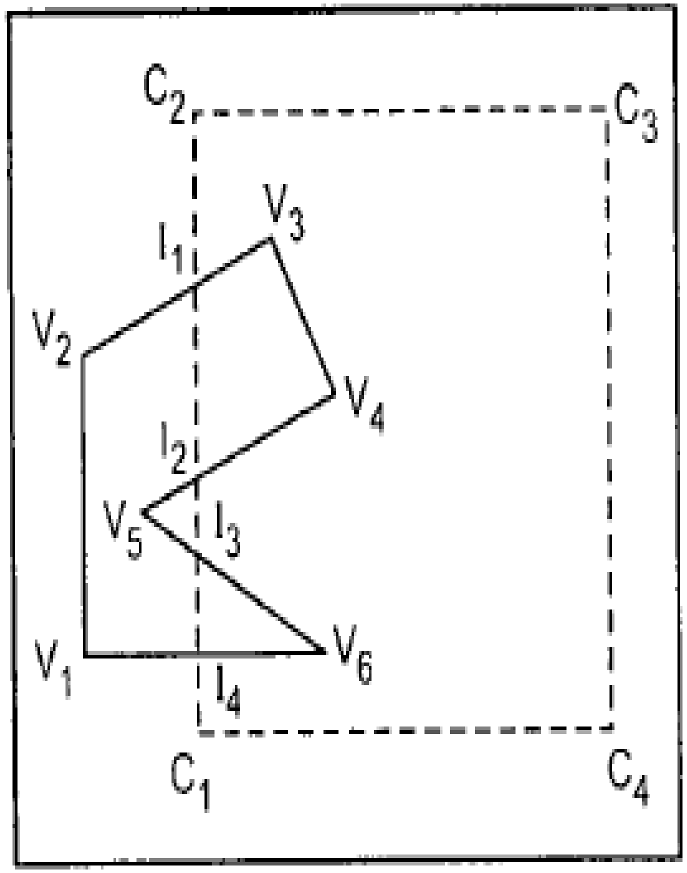


Fig. Clipping the can cave polygon in (a) with the Sutherland-Hodgeman algorithm produces the two connected areas in (b)

The problem of extraneous lines for concave polygons in Sutherland-Hodgeman polygon clipping algorithm can be solved by separating concave polygon into two or more convex polygons and processing each convex polygon separately.

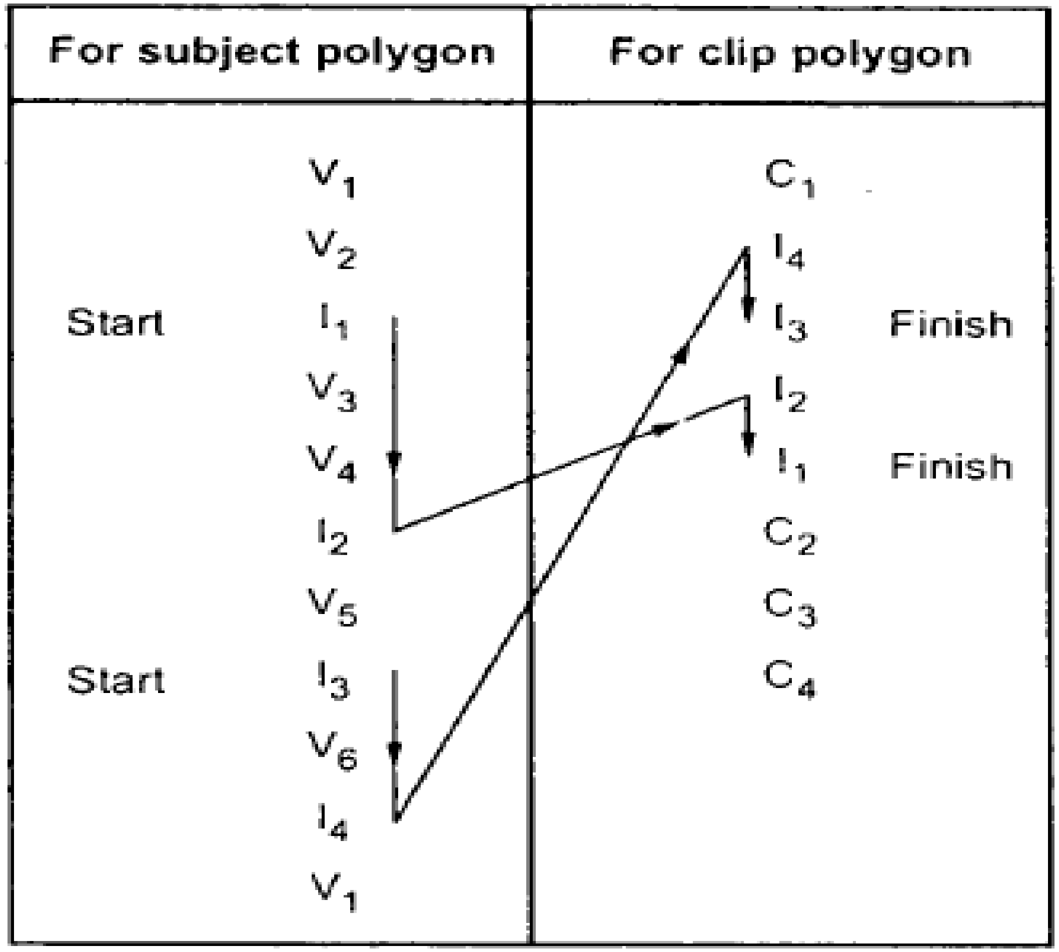
Weiler-Atherton Algorithm

The clipping algorithms previously discussed require a convex polygon. In context of many applications, e.g., hidden surface removal, the ability to clip to concave polygon is required. A powerful but somewhat more complex clipping algorithm developed by Weiler and Atherton meets this requirement. This algorithm defines the polygon to be clipped as a **subject polygon** and the clipping region is the clip polygon.



The algorithm describes both the subject and the clip polygon by a circular list of vertices. The boundaries of the subject polygon and the clip polygon may or may not intersect. If they intersect, then the intersections occur in pairs. One of the intersections occurs when a subject polygon edge enters the inside of the clip polygon and one when it leaves. As shown in the Fig. there are four intersection vertices I_1, I_2 , I_3 and I_4. In these intersections I_1 and I_3 are entering intersections, and I_2 and I_3 are leaving intersections. The clip polygon vertices are marked as C_1, C_2, C_3 and C_4.

In this algorithm two separate vertices lists are made one for clip polygon and one for subject polygon including intersection points. The Table shows these two lists for polygons shown in Fig.



For subject polygon		For clip polygon	
	V_1	C_1	
	V_2	I_4	
Start	I_1	I_3	Finish
	V_3	I_2	
	V_4	I_1	Finish
	I_2	C_2	
	V_5	C_3	
Start	I_3	C_4	
	V_6		
	I_4		
	V_1		

Table List of polygon vertices

The algorithm starts at an entering intersection (I_1) and follows the subject polygon vertex list in the downward direction (i.e. I_1, V_3, V_4, I_2). At the occurrence of leaving intersection the algorithm follows the clip polygon vertex list from the leaving intersection vertex in the downward direction (i.e. I_2, I_1). At the occurrence of the entering intersection the algorithm follows the subject polygon vertex list from the entering intersection vertex. This process is repeated until we get the starting vertex. This process we have to repeat for all remaining entering intersections which are not included in the previous traversing of vertex list. In our example, entering vertex I_3 was not included in the first traversing of vertex list,. Therefore, we have to go for another vertex traversal from vertex I_3.

The above two vertex traversals gives two clipped inside polygons. There are :

I_1, V_3, V_4, I_2, I_1 and I_3, V_6, I_4, I_3

Hidden Surface Elimination Methods

visible surface algorithms were called **hidden line** or **hidden surface algorithms**.

In a given set of 3D objects and viewing specification, we wish to determine which lines or surfaces of the objects are visible, so that we can display only the visible lines or surfaces. This process is known as hidden surfaces or hidden line elimination, or visible surface determination. The hidden line or hidden surface algorithm determines the lines, edges, surfaces or volumes that are visible or invisible to an observer located at a specific point in space.

These algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images. These two approaches are called **object-space methods** or **object precision methods** and **image-space methods**, respectively.

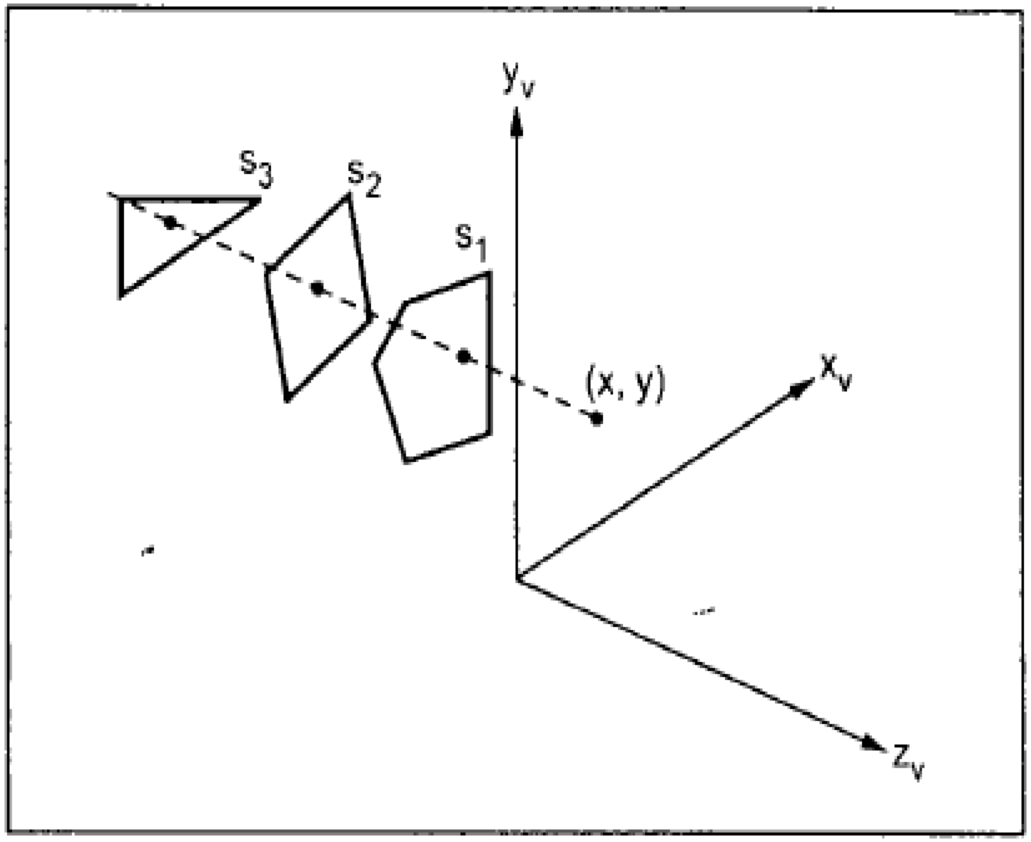
Object-space Method : Object-space method is implemented in the **physical coordinate system** in which objects are described. It compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible. Object-space methods are generally used in **line-display algorithms**.

Image-Space Method : Image space method is implemented in the **screen coordinate system** in which the objects are viewed. In an image-space algorithm, visibility is decided point by point at each pixel position on the view plane. Most **hidden line/surface algorithms** use image-space methods.

8.4.3 Z-Buffer Algorithm

One of the simplest and commonly used image space approach to eliminate hidden surfaces is the **Z-buffer** or **depth buffer** algorithm. It is developed by Catmull. This algorithm compares surface depths at each pixel position on the projection plane. The surface depth is measured from the view plane along the z axis of a viewing system. When object description is converted to projection coordinates (x, y, z), each pixel position on the view plane is specified by x and y coordinate, and z value gives the depth information. Thus object depths can be compared by comparing the z- values.

The Z-buffer algorithm is usually implemented in the normalized coordinates, so that z values range from 0 at the back clipping plane to 1 at the front clipping plane. The implementation requires another buffer memory called Z-buffer along with the frame buffer memory required for raster display devices. A Z-buffer is used to store depth values for each



(x, y) position as surfaces are processed, and the frame buffer stores the intensity values for each position. At the beginning Z-buffer is initialized to zero, representing the z-value at the back clipping plane, and the frame buffer is initialized to the background colour. Each surface listed in the display file is then processed, one scan line at a time, calculating the depth (z-value) at each (x, y) pixel position. The calculated depth value is compared to the value previously stored in the Z-buffer at that position.

If the calculated depth values is greater than the value stored in the Z-buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same xy location in the frame buffer.

Z-buffer Algorithm

1. Initialize the Z-buffer and frame buffer so that for all buffer positions

   Z-buffer (x, y) = 0 and frame-buffer (x, y) = $I_{background}$

2. During scan conversion process, for each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

   Calculate z-value for each (x, y) position on the polygon

   If z > Z-buffer (x, y), then set

   Z-buffer (x, y) = z, frame-buffer (x, y) = $I_{surface}$ (x, y)

3. Stop

Note that, $I_{background}$ is the value for the background intensity, and $I_{surface}$ is the projected intensity value for the surface at pixel position (x, y). After processing of all surfaces, the Z-buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding intensity values for those surfaces.

To calculate z-values, the plane equation

$$Ax + By + Cz + D = 0$$

is used where (x, y, z) is any point on the plane, and the coefficient A, B, C and D are constants describing the spatial properties of the plane.

Therefore, we can write

$$z = \frac{-Ax - By - D}{C}$$

Note, if at (x, y) the above equation evaluates to z_1, then at $(x + \Delta x, y)$ the value of z, is

$$z_1 = \frac{A}{C}(\Delta x)$$

Only one subtraction is needed to calculate z(x + 1, y), given z(x, y), since the quotient A/C is constant and $\Delta x = 1$. A similar incremental calculation can be performed to determine the first value of z on the next scan line, decrementing by B/C for each Δy.

Advantages

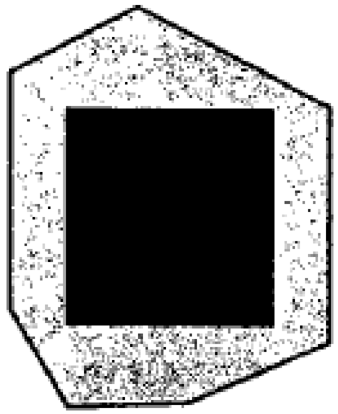
1. It is easy to implement.
2. It can be implemented in hardware to overcome the speed problem.
3. Since the algorithm processes objects one at a time, the total number of polygons in a picture can be arbitrarily large.

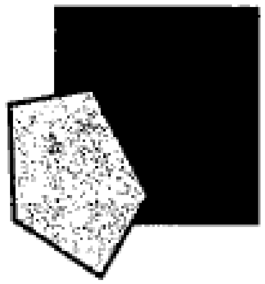
Disadvantages

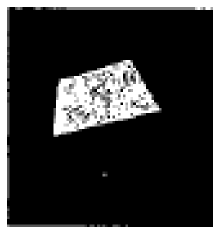
1. It requires an additional buffer and hence the large memory.
2. It is a time consuming process as it requires comparison for each pixel instead of for the entire polygon.

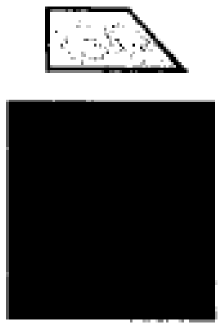
Warnock's Algorithm (Area Subdivision Algorithm)

An interesting approach to the hidden-surface problem was developed by Warnock. He developed area subdivision algorithm which subdivides each area into four equal squares. At each stage in the recursive-subdivision process, the relationship between projection of each polygon and the area of interest is checked for four possible relationships :


(a) Surrounding

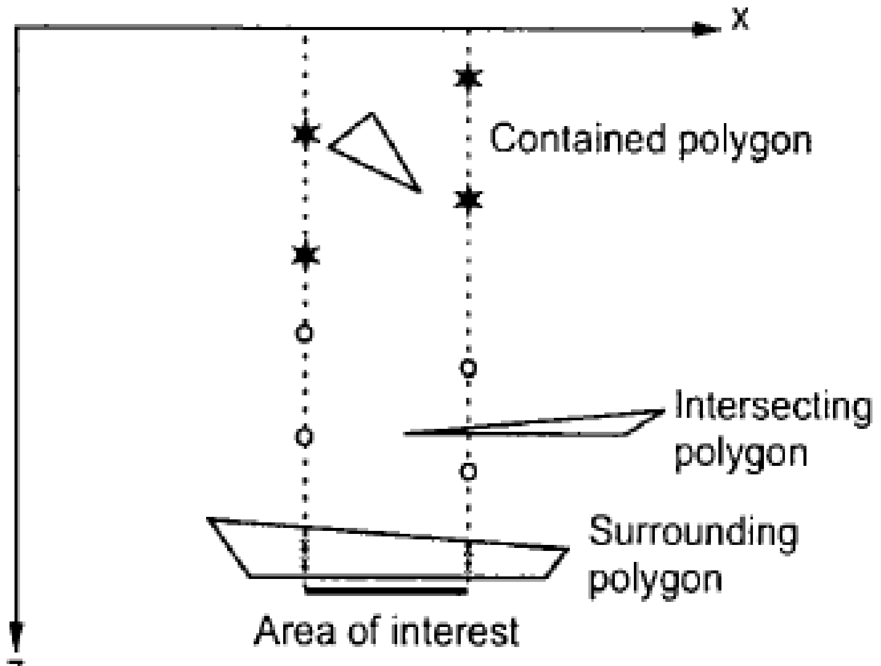

(b) Overlapping


(c) Inside or Contained

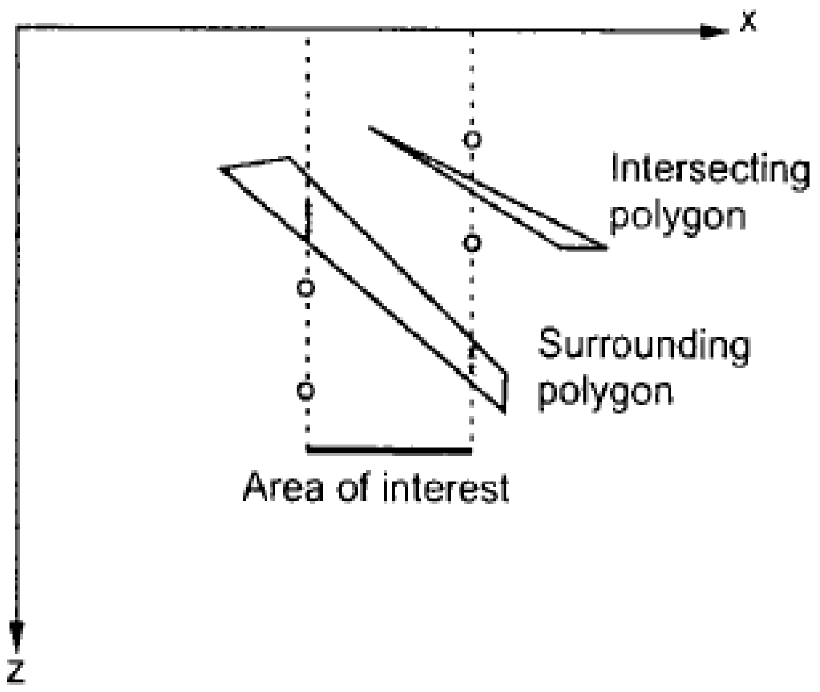

(d) Outside or Disjoint

After checking four relationships we can handle each relationship as follows :

1. If all the polygons are disjoint from the area, then the background colour is displayed in the area.
2. If there is only one intersecting or only one contained polygon, then the area is first filled with the background colour, and then the part of the polygon contained in the area is filled with colour of polygon.
3. If there is a single surrounding polygon, but no intersecting or contained polygons, then the area is filled with the colour of the surrounding polygon.
4. If there are more than one polygon intersecting, contained in, or surrounding the area then we have to do some more processing.



(a)

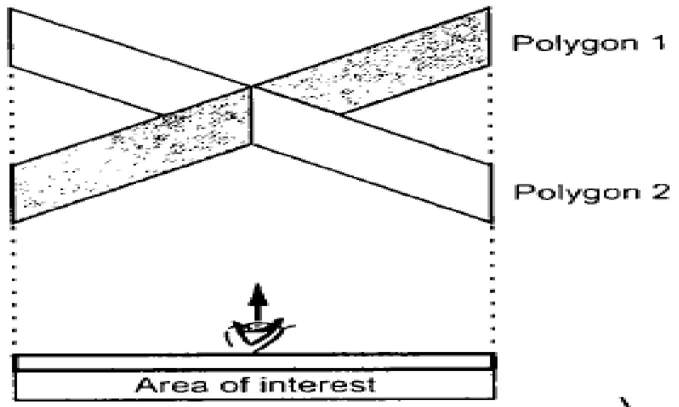


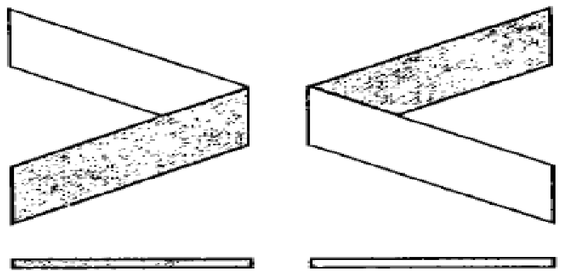
(b)

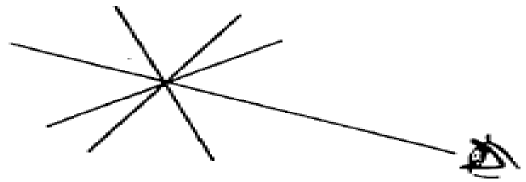
In Fig. (a), the four intersections of surrounding polygon are all closer to the viewpoint than any of the other intersections. Therefore, the entire area is filled with the colour of the surrounding polygon.

However, Fig. (b) shows that surrounding polygon is not completely in front of the intersecting polygon. In such case we cannot make any decision and hence Warnock's algorithm subdivides the area to simplify the problem. This is illustrated in Fig. As shown in the Fig. (a) we can not make any decision about which polygon is in front of the other. But after dividing area of interest polygon 1 is ahead of the polygon 2 in left area and polygon 2 is ahead of polygon 1 in the right area. Now we can fill these two areas with corresponding colours of the polygons.

The Warnock's algorithm stops subdivision of area only when the problem is simplified or when area is only a single pixel.







Algorithm

1. Initialize the area to be the whole screen.
2. Create the list of polygons by sorting them with their z-values of vertices. Don't include disjoint polygons in the list because they are not visible.
3. Find the relationship of each polygon.
4. Perform the visibility decision test
   a) If all the polygons are disjoint from the area, then fill area with background colour.

   b) If there is only one intersecting or only one contained polygon then first fill entire area with background colour and then fill the part of the polygon contained in the area with the colour of polygon.

   c) If there is a single surrounding polygon, but no intersecting or contained polygons, then fill the area with the colour of the surrounding polygon.

   d) If surrounding polygon is closer to the viewpoint than all other polygons, so that all other polygons are hidden by it, fill the area with the colour of the surrounding polygon.

   e) If the **area is the pixel** (x, y), and neither a,b,c, nor d applies, compute the z coordinate at pixel (x, y) of all polygons in the list. The pixel is then set to colour of the polygon which is closer to the viewpoint.
5. If none of the above tests are true then subdivide the area and go to step 2.

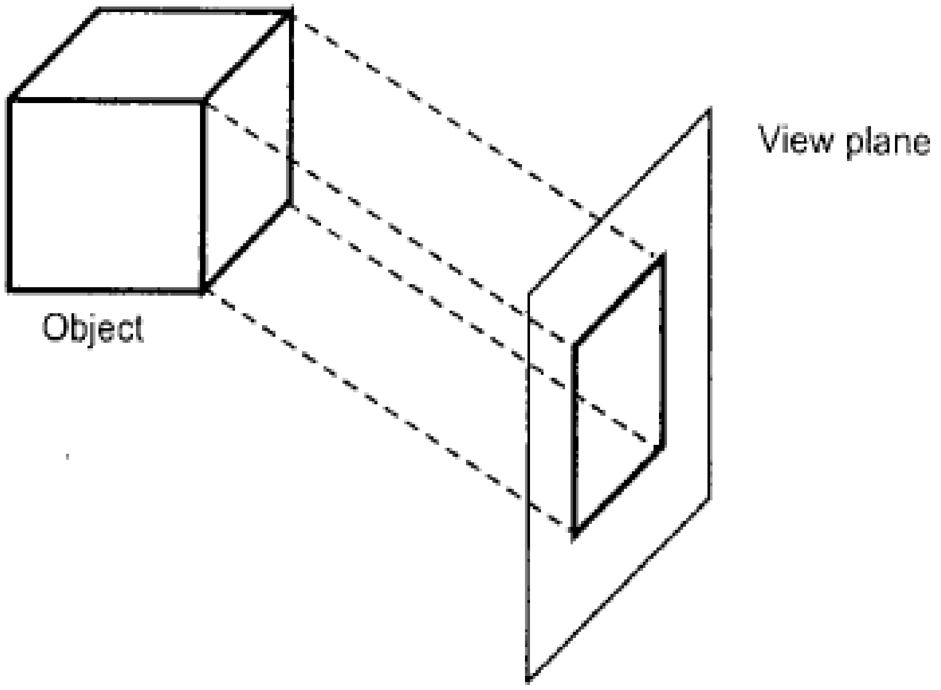
Advantages

1. It follows the divide-and-conquer strategy, therefore, parallel computers can be used to speed up the process.

2. Extra memory buffer is not required.

Parallel Projection

In parallel projection, z - coordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane. The point of intersection is the projection of the vertex. We connect the projected vertices by line segments which correspond to connections on the original object.

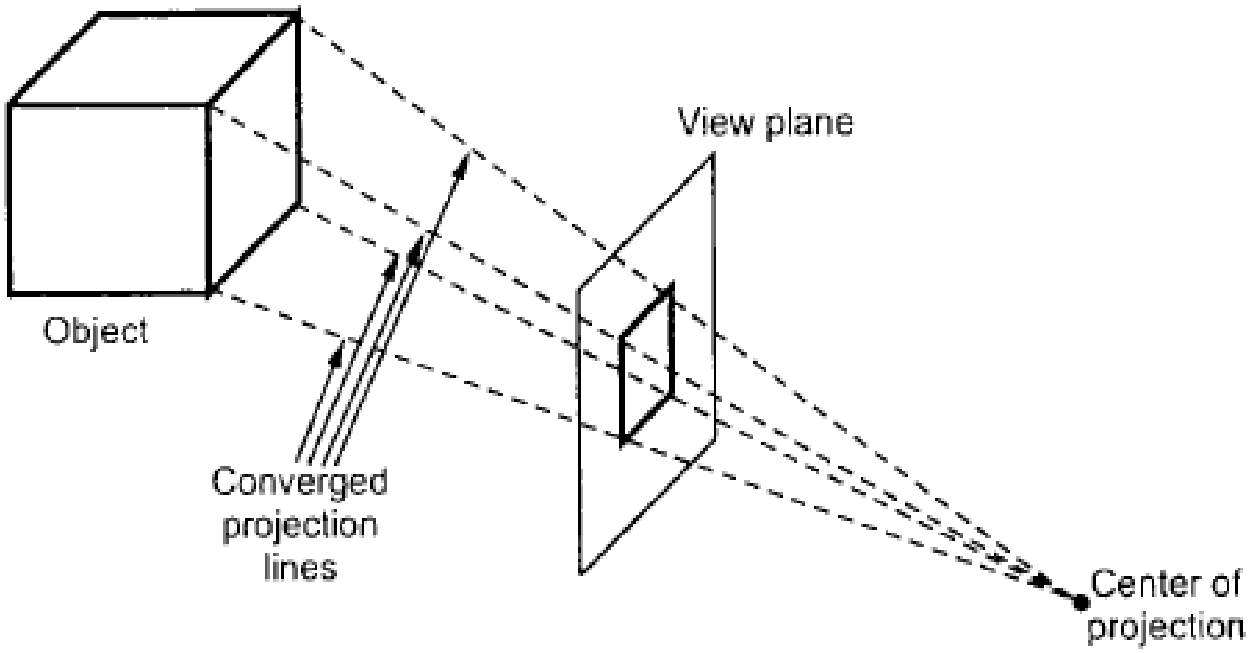


Parallel projection of an object to the view plane

As shown in the Fig.   a parallel projection preserves relative proportions of objects but does not produce the realistic views.

Perspective Projection

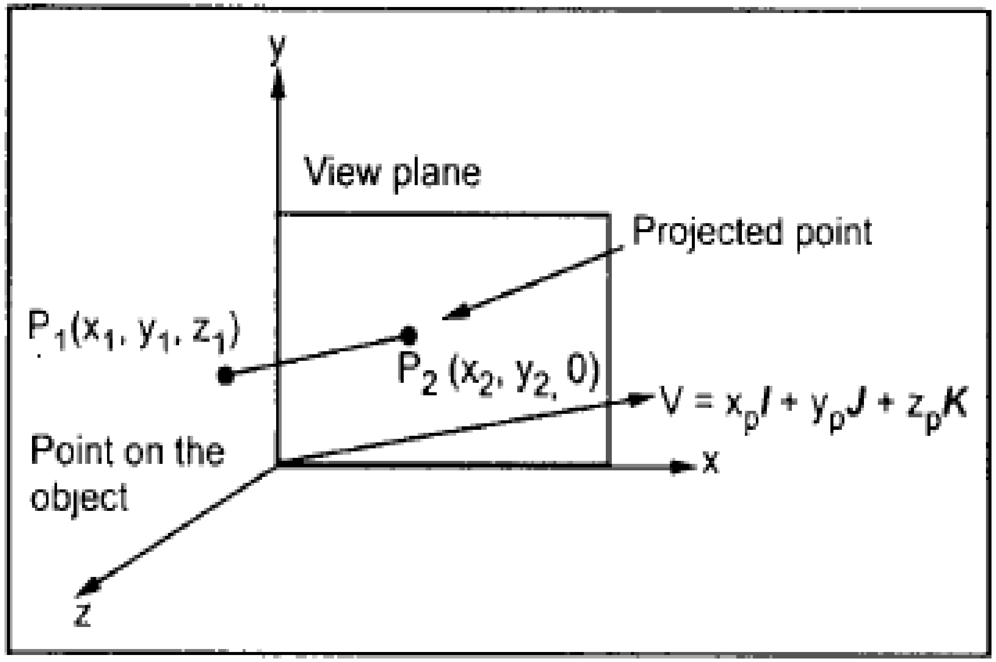
The perspective projection, on the other hand, produces realistic views but does not preserve relative proportions. In perspective projection, the lines of projection are not parallel. Instead, they all converge at a single point called the **center of projection** or **projection reference point**. The object positions are transformed to the view plane along these converged projection lines and the projected view of an object is determined by calculating the intersection of the converged projection lines with the view plane, as shown in the Fig.



Perspective projection of an object to the view plane

Transformation Matrices for General Parallel Projection

On XY Plane



In a general parallel projection, we may select any direction for the lines of projection. Suppose that the direction of projection is given by the vector $[x_p\ y_p\ z_p]$ and that the object is to be projected onto the xy plane. If the point on the object is given as (x_1, y_1, z_1), then we can determine the projected point (x_2, y_2) as given below :

The equations in the parametric form for a line passing through the projected point (x_2, y_2, z_2) and in the direction of projection are given as

$$x_2 = x_1 + x_p u$$
$$y_2 = y_1 + y_p u$$
$$z_2 = z_1 + z_p u$$

For projected point z_2 is 0, therefore, the third equation can be written as,

$$0 = z_1 + z_p u$$

$$\therefore \quad u = \frac{-z_1}{z_p}$$

Substituting the value of u in first two equations we get,

$$x_2 = x_1 + x_p(-z_1/z_p) \quad \text{and}$$

$$y_2 = y_1 + y_p(-z_1/z_p)$$

The above equations can be represented in matrix form as given below :

$$[x_2 \ y_2] = [x_1 \ y_1 \ z_1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -x_p/z_p & -y_p/z_p \end{bmatrix}$$

or in homogeneous coordinates we have,

$$[x_2 \ y_2 \ z_2 \ 1] = [x_1 \ y_1 \ z_1 \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -x_p/z_p & -y_p/z_p & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

i.e. $$P_2 = P_1 \cdot Par_v$$

This is the general equation of parallel projection on xy plane in matrix form.

Here, we ignore the value of z_2 when drawing the projected image.

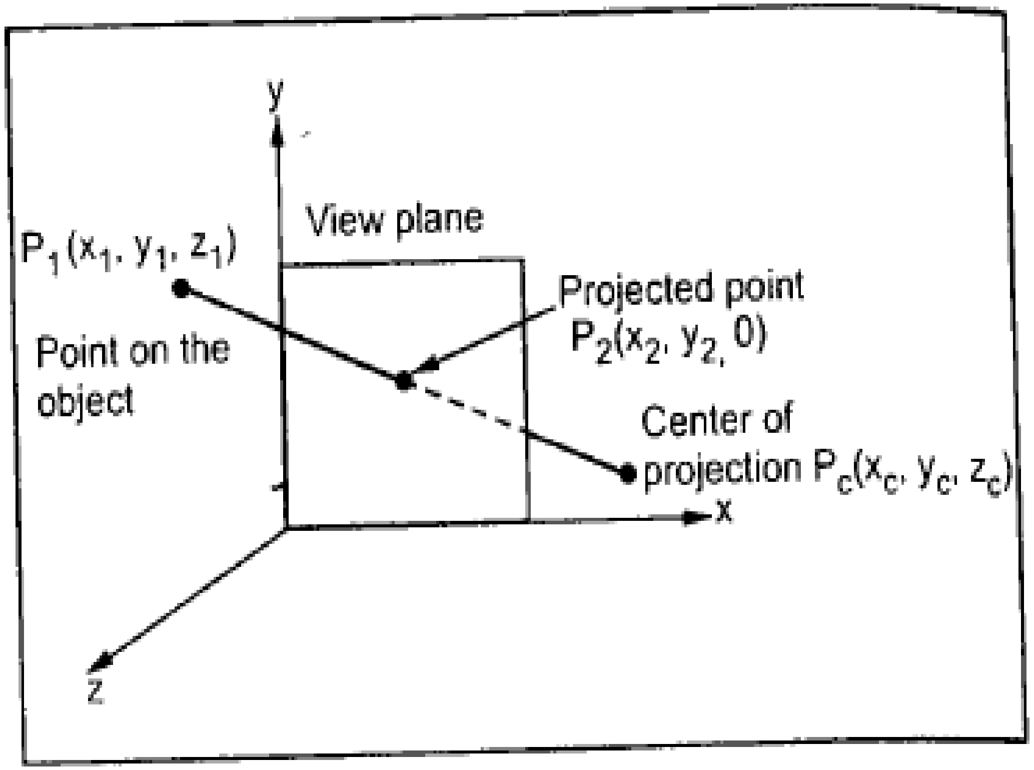
Transformation Matrix for Perspective Projection

Let us consider the center of projection is at $P_c(x_c, y_c, z_c)$ and the point on object is $P_1(x_1, y_1, z_1)$, then the parametric equation for line containing these points can be given as

$$x_2 = x_c + (x_1 - x_c)\,u$$

$$y_2 = y_c + (y_1 - y_c)\,u$$

$$z_2 = z_c + (z_1 - z_c)\,u$$



y
View plane
$P_1(x_1, y_1, z_1)$
Point on the object
Projected point $P_2(x_2, y_2, 0)$
Center of projection $P_c(x_c, y_c, z_c)$
x
z

For projected point z_2 is 0, therefore, the third equation can be written as

$$0 = z_c + (z_1 - z_c)\,u$$

$$\therefore\ u = -\frac{z_c}{z_1 - z_c}$$

Substituting the value of u in first two equations we get,

$$x_2 = x_c - z_c \frac{x_1 - x_c}{z_1 - z_c}$$

$$= \frac{x_c z_1 - x_c z_c - x_1 z_c + x_c z_c}{z_1 - z_c}$$

$$= \frac{x_c z_1 - x_1 z_c}{z_1 - z_c} \quad \text{and}$$

$$y_2 = y_c - z_c \frac{y_1 - y_c}{z_1 - z_c}$$

$$= \frac{y_c z_1 - y_c z_c - y_1 z_c + y_c z_c}{z_1 - z_c}$$

$$= \frac{y_c z_1 - y_1 z_c}{z_1 - z_c}$$

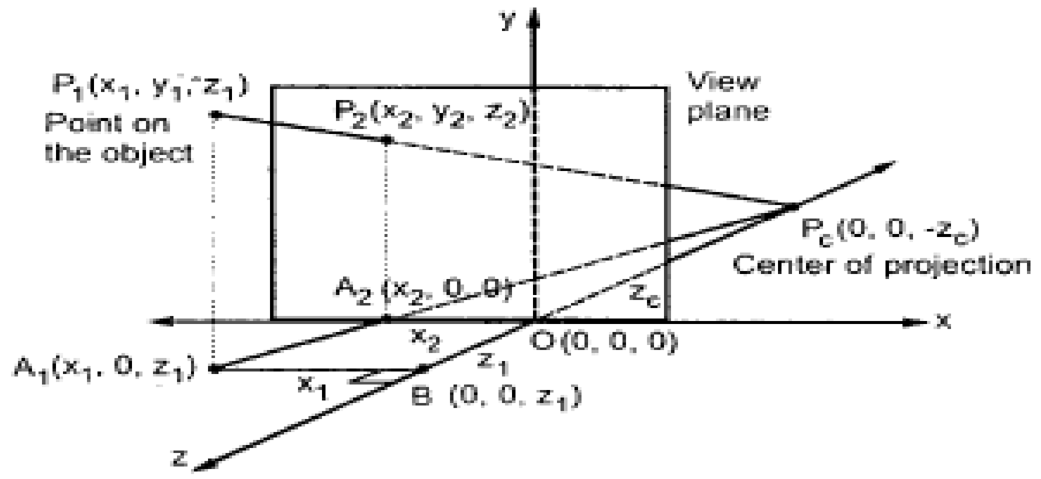
The above equations can be represented in the homogeneous matrix form as given below :

$$[x_2 \quad y_2 \quad z_2 \quad 1] = [x_1 \quad y_1 \quad z_1 \quad 1] \begin{bmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{bmatrix}$$

Here, we have taken the center of projection as $P_c(x_c, y_c, z_c)$. If we take the center of projection on the negative z-axis such that

$$x = 0$$
$$y = 0$$
$$z = -z_c$$

i.e. $P_c(0, 0, -z_c)$ then we have



$$\triangle P_c O A_2 \sim \triangle P_c B A_1$$

$$\therefore\; x_2 = \frac{x_1 z_c}{z_1 + z_c}$$

Similarly,

$$y_2 = \frac{y_1 z_c}{z_1 + z_c}$$

Fig. 7.20

$$x_2 = \frac{z_c x_1}{z_c + z_1}$$

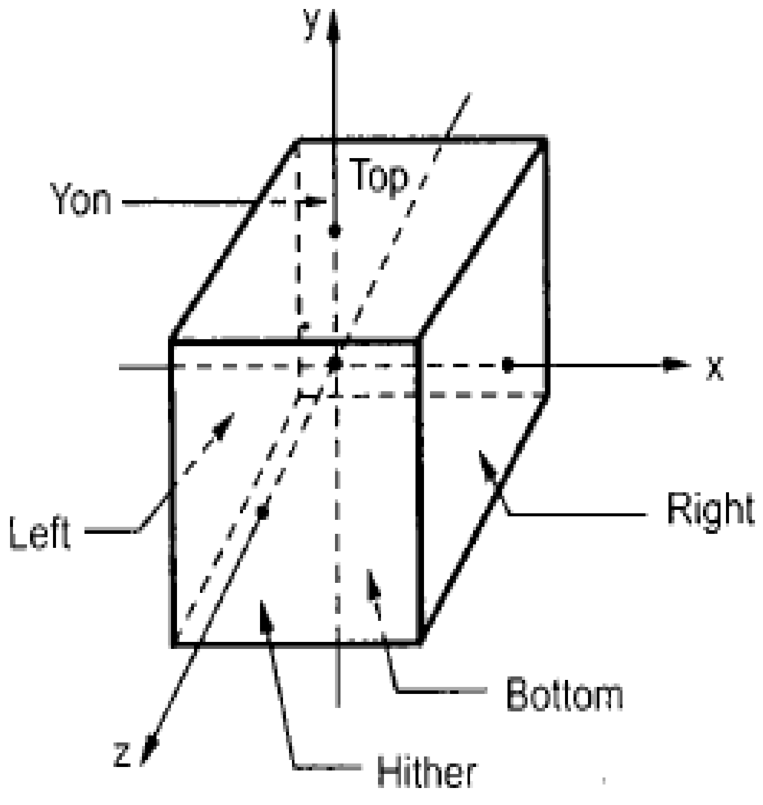
$$y_2 = \frac{z_c y_1}{z_c + z_1}$$

$$z_2 = 0$$

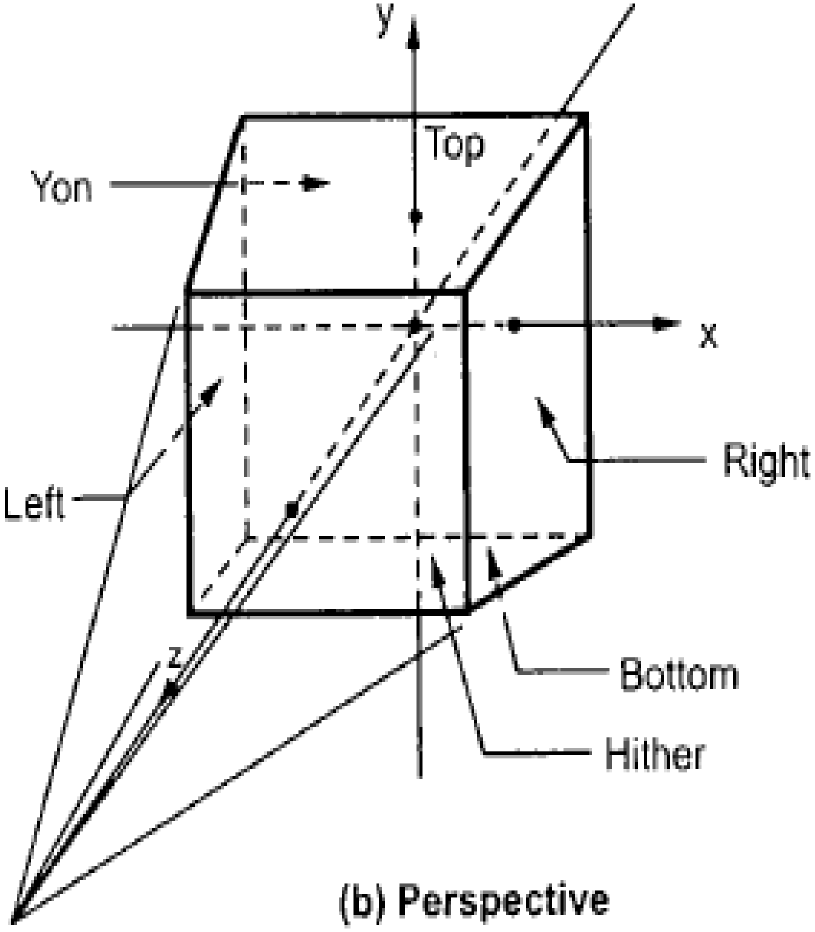
Thus we get the homogeneous perspective transformation matrix as

$$[x_2 \quad y_2 \quad z_2 \quad 1] = [x_1 \quad y_1 \quad z_1 \quad 1] \begin{bmatrix} z_c & 0 & 0 & 0 \\ 0 & z_c & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & z_c \end{bmatrix}$$

Three Dimensional Clipping



(a) Parallel



(b) Perspective

The two-dimensional concept of region codes can be extended to three dimensions by considering six sides and 6-bit code instead of four sides and 4-bit code. Like two-dimension, we assign the bit positions in the region code from right to left as

Bit 1 = 1, if the end point is to the left of the volume

Bit 2 = 1, if the end point is to the right of the volume

Bit 3 = 1, if the end point is the below the volume

Bit 4 = 1, if the end point is above the volume

Bit 5 = 1, if the end point is in front of the volume

Bit 6 = 1, if the end point is behind the volume

Otherwise, the bit is set to zero. As an example, a region code of 101000 identifies a point as above and behind the view volume, and the region code 000000 indicates a point within the view volume.

A line segment can be immediately identified as completely within the view volume if both endpoints have a region code of 000000. If either endpoint of a line segment does not have a region code of 000000, we perform the logical AND operation on the two endpoint codes. If the result of this AND operation is nonzero then both endpoints are outside the view volume and line segment is completely invisible. On the other hand, if the result of AND operation is zero then line segment may be partially visible. In this case, it is necessary to determine the intersection of the line and the clipping volume.

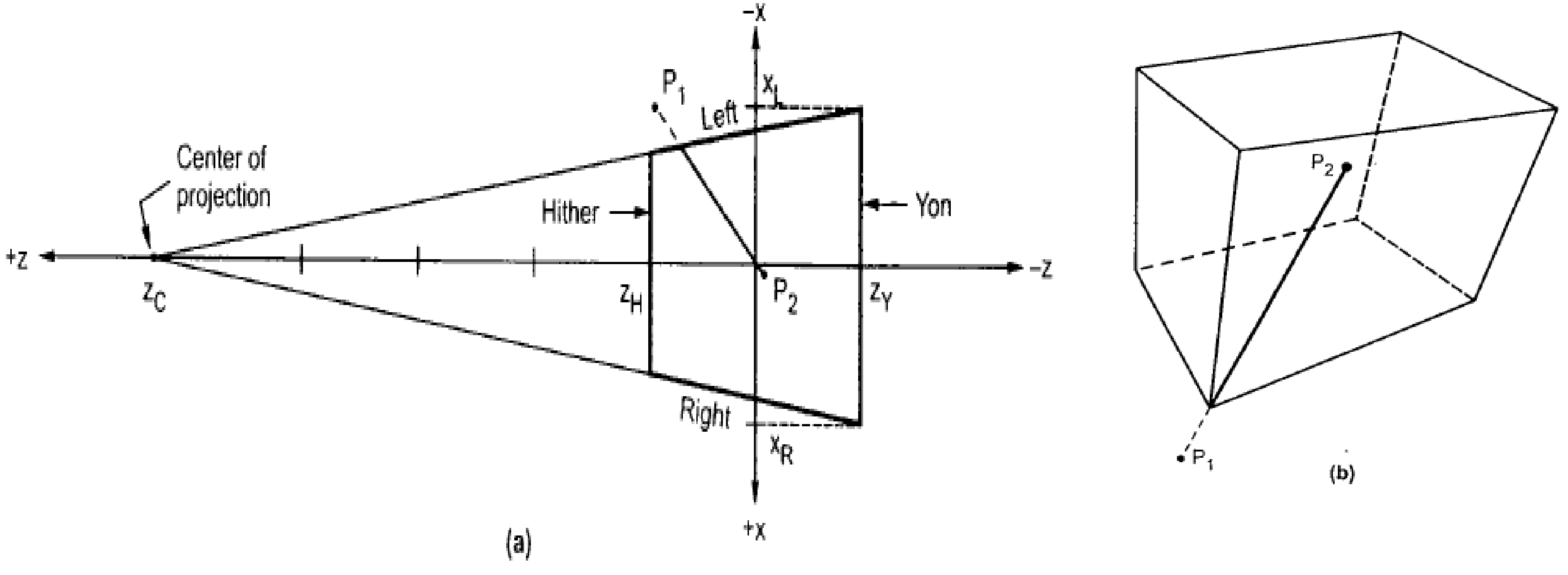


Fig. shows a top view of the perspective clipping volume. The equation of the line which represents the right hand plane in this view can be given as

$$x = \frac{z - z_C}{z_Y - z_C} x_R = z\alpha_1 + \alpha_2$$

where

$$\alpha_1 = \frac{x_R}{z_Y - z_C} \quad \text{and } \alpha_2 = -\alpha_1 z_C$$

This equation of right hand plane can be used to determine whether a point is to the right, on or to the left of the plane, i.e., outside the volume, on the right hand plane, or inside the volume. Substituting the x and y coordinates of a point P into $x - z\alpha_1 - \alpha_2$ gives the following results

$f_R = x - z\alpha_1 - \alpha_2$	> 0	if P is to the right of the right plane
	$= 0$	if P is on the right plane
	< 0	if P is to the left of the right plane

Similarly, we can derive the test functions for the left, top bottom, hither and yon planes. Table shows these test functions.

Plane	Test functions with Results		
Right	$f_R = x - z\alpha_1 - \alpha_2$	> 0	if P is to the right of the right plane
		$= 0$	if P is on the right plane
		< 0	if P is to the left of the right plane
	where	$\alpha_1 = \dfrac{x_R}{z_Y - z_C}$ and	$\alpha_2 = -\alpha_1 z_C$
Left	$f_L = x - z\beta_1 - \beta_2$	< 0	if P is to the left of the left plane
		$= 0$	if P is on the left plane
		> 0	if P is to the right of the left plane
	where	$\beta_1 = \dfrac{x_L}{z_Y - z_C}$	and $\beta_2 = -\beta_1 z_C$

Top	$f_T = y - z\,\gamma_1 - \gamma_2$	> 0	if P is above the top plane
		$= 0$	if P is on the top plane
		< 0	if P is below the top plane
	where	$\gamma_1 = \dfrac{y_T}{z_Y - z_C}$ and $\gamma_2 = -\gamma_1 z_C$	
Bottom	$f_B = y - z\,\delta_1 - \delta_2$	< 0	if P is below the bottom plane
		$= 0$	if P is on the bottom plane
		> 0	if P is above the bottom plane
	where	$\delta_1 = z_C$ and $\delta_2 = -\delta_1 z_C$	
Hither	$f_H = z - z_H$	> 0	if P is in front of the hither plane
		$= 0$	if P is on the hither plane
		< 0	if P is behind the hither plane
Yon	$f_y = z - z_y$	< 0	if P is behind the yon plane
		$= 0$	if P is on the yon plane
		> 0	if P is in front of the yon plane

Table [illegible] Test functions for six planes of clipping volume