COMPUTER GRAPHICS

MODULE - 5

Visible Surface Detection Algorithm (Hidden Surface Removal / Hidden Surface Detection)

- Back face removal.
- Depth Buffer (Z-Buffer) Algorithm.
- A-buffer Algorithm.
- Depth-sorting method.
- Scan-Line Method.

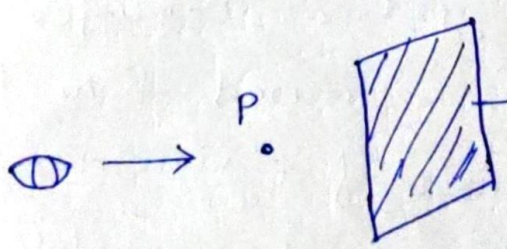
① Back Face Detection & Removal

* An object space method for detecting back faces (hidden faces), eliminate it & then make visible the remaining visible surfaces.

* Here, a point $P(x,y,z)$ is inside a polygon surface with plane parameters A, B, C & D.

  if, $Ax + By + Cz + D < 0$

* When an inside point is along the line of sight to the surface, the polygon must be a back face.



If cannot see, then visible surface.

* This test can be simplified as by considering the normal vector N to a polygon surface, which has cartesian components (A, B, C) & if V is a vector in the viewing direction from the eye (or camera) position, then this polygon is a back face if, $\boxed{V \cdot N > 0}$

* If the viewing direction is parallel to the viewing z-axis, then $V = (0, 0, V_z)$ & so that it only needs to consider the sign of C in the component of the normal vector $N(A, B, C)$

So, $V \cdot N = V_z \cdot C$

This can be further simplified as,

- In right-handed viewing s/m with viewing direction along the negative z-axis ie, $V_z = -1$ then polygon is a backface if $C < 0$

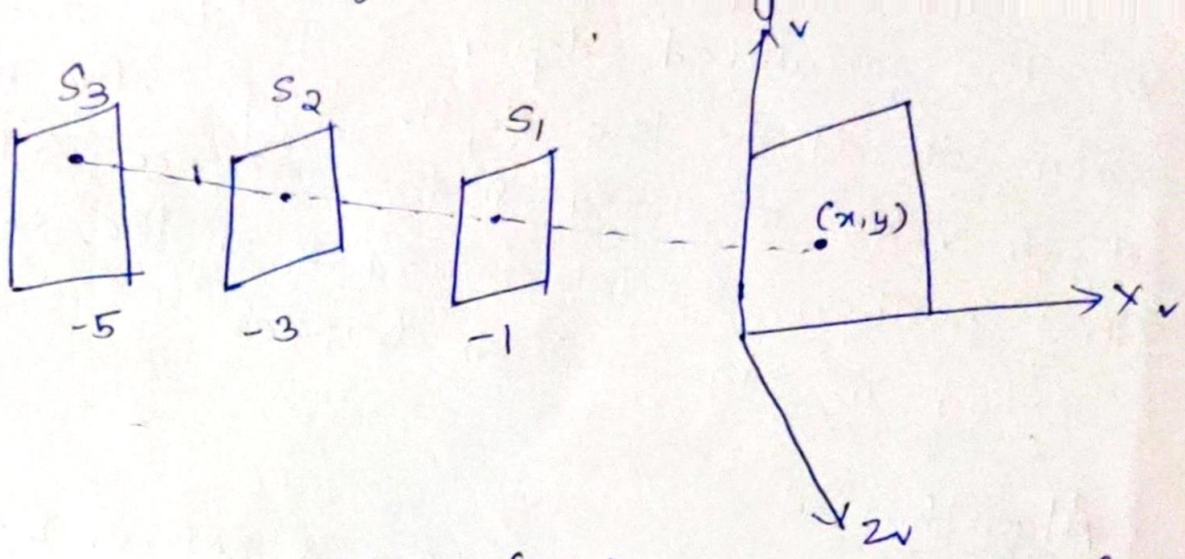
$-1 \cdot -1 > 0$
↳ B.F

- In a left-handed viewing s/m with viewing direction along the positive z-axis ie, $V_z = 1$ then polygon is a backface if $C > 0$

$1 \cdot 1 > 0$
↳ B.F

(2) Depth-buffer (z-buffer) Method.

* A commonly used image-space approach to detecting visible surfaces is the depth-buffer method, which compares surface depths at each pixel position on the projection plane.
* Due to depth calculation it is referred as depth buffer.
* This procedure is also referred to as z-buffer method, since object depth is usually measured from the view plane along the z-axis of a viewing s/m.
* Each surface of a scene is processed separately, one point at a time across the surface.
* The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly & the method is easy to implement.



* For each pixel position (x,y) on the view plane, object depths can be compared by comparing z-values, then surfaces with highest depth (z-val is closer to the viewing position, so it is visibl its surface intensity value at (x,y) is saved for display.
* Two buffer areas are required :
* A <u>depth buffer</u> is used to store depth values for each (x,y) position as surfaces are processed.
* A <u>refresh buffer</u> stores the intensity values for each position.
* Initially, all positions in the depth buffer are set to 0 (minimum depth), & the refresh buffer is initialized to the background intensity.
* Each surface listed in the polygon tables is then processed, one scan line at a time, calculating the depth (z-value) at each (x,y) pixel position. The calculated depth is compared to the value previously stored in the depth buffer at that position.

* If the calculated depth is greater than the value stored in the depth buffer, the new depth value is stored & the surface intensity at that position is determined & in the same xy location in the refresh buffer.

Algorithm

1. Initialize the depth buffer & refresh buffer so that for all buffer positions (x, y),

$$\text{depth}(x, y) = 0$$

$$\text{refresh}(x, y) = I_{backgnd}.$$

2. For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

- Calculate the depth for each (x, y) position on the polygon.
- If $z > \text{depth}(x, y)$ then set,

$$\text{depth}(x, y) = z$$

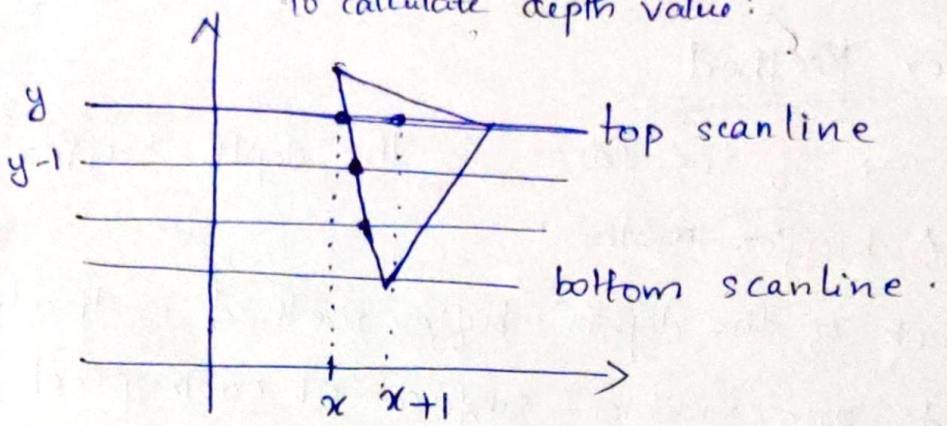
$$\text{refresh}(x, y) = I_{surf}(x, y).$$

where, $I_{backgnd}$ is the value for background intensity

$I_{surf}(x, y)$ is the projected intensity value for the surface at pixel position (x, y).

* After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces & the refresh buffer contains the corresponding intensity values for those surfaces.

To calculate depth value:



* Depth values for a surface position (x, y) are calculated from the plane eqn for each surface.

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

$$z = \frac{-Ax - By - D}{C} \quad \text{——(1)}$$

* If the depth of posn (x, y) has been determined to be z, then the depth z' of the next posn $(x+1, y)$ along the scan line is obtained as,

Sub $(x+1, y)$ in Eq (1)

$$z' = \frac{-A(x+1) - By - D}{C}$$

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

$$\boxed{z' = z - \frac{A}{C}}$$

* The ratio $-A/C$ is constant for each surface, so succeeding depth values across a scan line are obtained from preceeding values with a single addition.

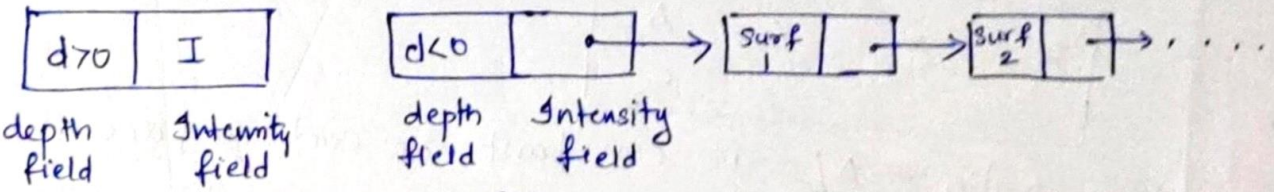
~~Next line,~~

~~$y_2 - y_1$~~

~~$x_2 - x_1$~~

③ A-buffer Method.

* An extension of the ideas in the depth-buffer method is the A-buffer method.
* A drawback of the depth buffer method is that it can only find one visible surface at each pixel position.
* In other words, it deals only with opaque surfaces & cannot accumulate intensity values for more than one surface, as is necessary if transparent surfaces are to be displayed.
* The A-buffer method expands the depth buffer so that each position in the buffer can reference a linked list of surfaces.
* Each position in the A-buffer has two fields :
  - depth field - stores a positive or negative real number
  - intensity field - stores surface-intensity information or a pointer value.



(a) (b)

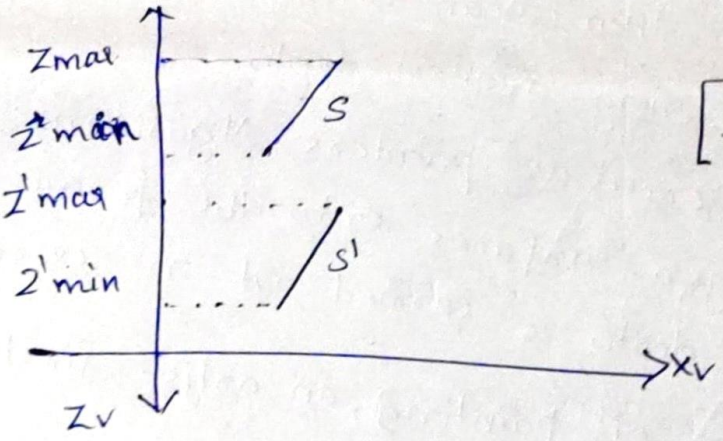
* If the depth field is positive, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area.
The intensity field then stores the RGB component of the surface color at that point & the percent of pixel coverage.
* If the depth field is negative, this indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data.

* Data for each surface in the linked list include:
  - ✓ RGB intensity components
  - ✓ opacity parameter (percent of transparency)
  - ✓ depth.
  - ✓ percent of area coverage
  - ✓ surface identifier.
  - ✓ other surface-rendering parameter
  - ✓ pointer to next surface.

(4) Depth Sorting Method.

* This method uses both image space & object space operation
* The depth-sorting method performs 2 basic functions:
  1. Surfaces are sorted in their decreasing order of depth
  2. Surfaces are then scan converted, starting from surface having highest depth.
* This is also referred as painters Algorithm.
* Painting polygon surfaces onto the frame buffer according to depth is carried out in several steps.
* In creating an oil painting, an artist first paints the background colors.
  Next, the most distant objects are added. Then the nearer objects & so on.
  At the final step, the foreground objects are painted on the canvas over the background & other objects that have been painted on the canvas.
  Each layer of paint covers up the previous layer.
* Using a similar technique, we first sort surfaces according to their distance from the view plane.

* The intensity values for the farthest surface are then entered into the refresh buffer.
* Taking each succeeding surface in turn [in decreasing depth order], we paint the surface intensities onto the frame buffer over the intensities of the previously processed surfaces.
* Assuming we are viewing along -z direction, surfaces are ordered on the first pass according to the smallest z-values on each surfaces. Surface S with greatest depth is then compared to the other surfaces in the list to determine whether there are any overlaps in depth.
* If no depth overlap occurs, S is scan converted.



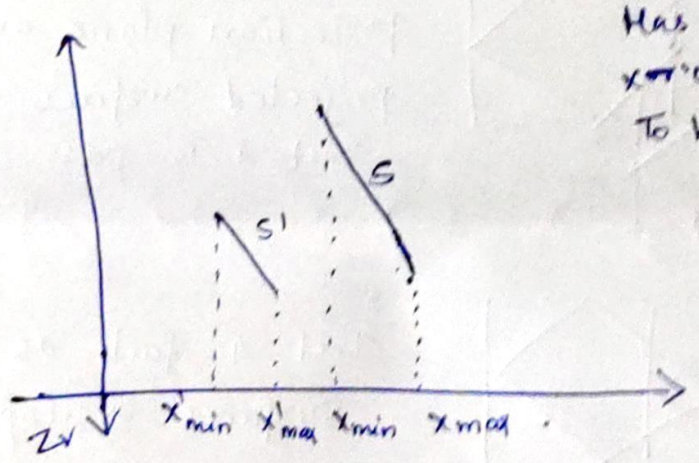
[fig: Two surfaces with no depth overlap]

fig(a) shows two surfaces that overlap in X-Y plane but have no depth overlap.
This process is then repeated for the next surface in the list.
As long as no overlap occurs, each surface is processed in depth order until all have been scan converted.

* If a depth overlap is detected at any point in the list, we need to make some additional comparisions to determine whether any of the surfaces should be reordered.
* We make the following test for each surface that overlaps with the S.
* If any of these test is true, no reordering is necessary for that surface.

The tests are:

(i) The bounding rectangle in the xy plane for the two surfaces donot overlap. [fig (b)



Has depth overlap, but no x overlap.
To have x overlap, it must be
$x_{min} < x'_{max}$

fig (b)

(ii) Surface S is completely behind the overlapping surface relative to the viewing position [fig (c)]

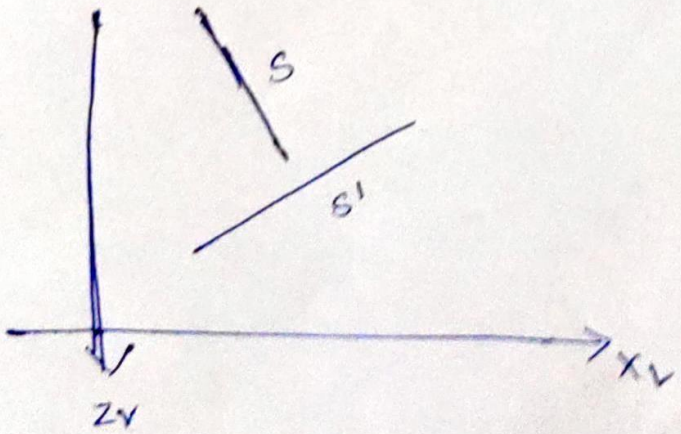
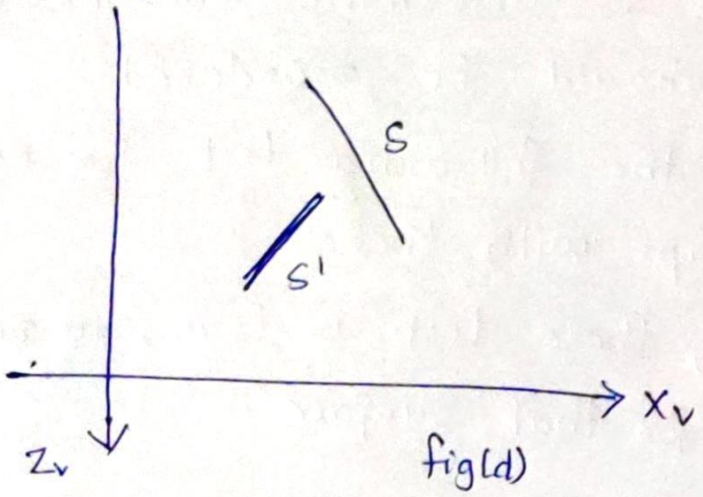


fig (c)

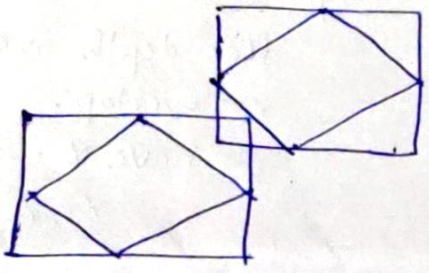
(iii) The overlapping surface is completely infront of S relative to the viewing position [fig (d)]



fig(d)

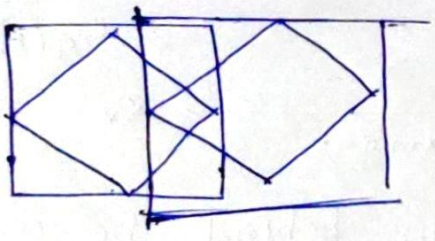
(iv) The projections of the two surfaces onto the viewplane donot overlap

Eg (i)



projection plane overlap but projected surface donot overlap. Test A is pass.

Eg (ii)



Test A fails as the projected surfaces overlap.

* All test fails, then reorder ie, $S \rightarrow S'$ is reordered to $S' \rightarrow S$.

(5) Scan-Line Method.

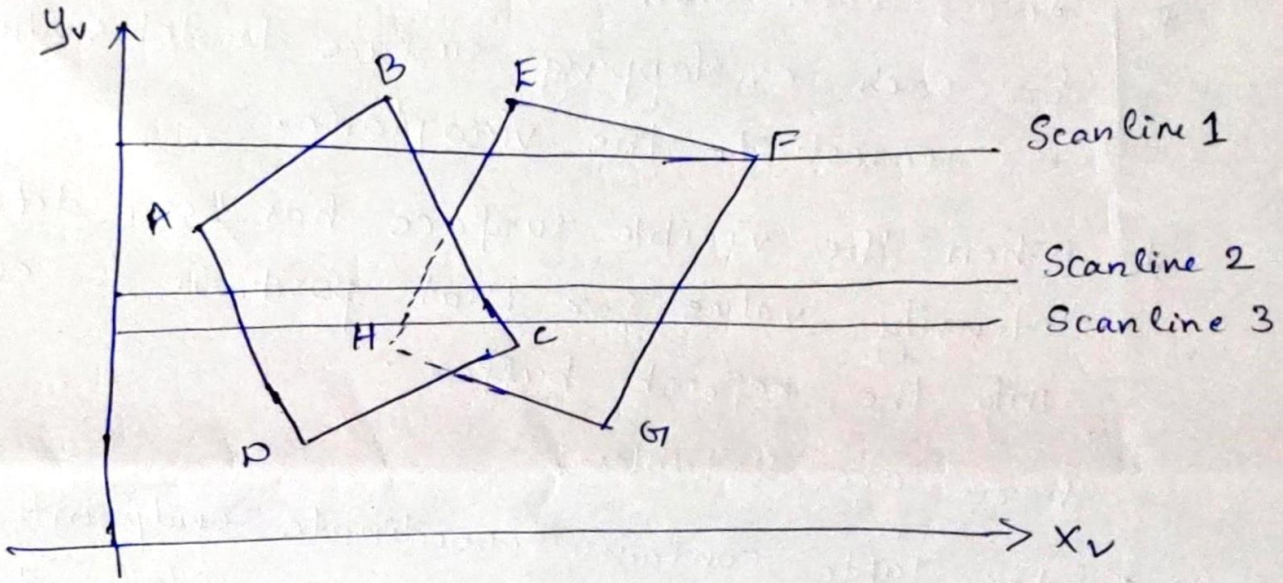
* The Image space method for removing hidden surface is an extension of the scan-line algorithm for filling polygon interiors.
* As each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible.
* Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane.
* When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.

There are 2 tables:

* Edge Table contains coordinate endpoints for each line in the scene, the inverse slope of each line & pointers into the polygon table to identify the surfaces bounded by each line.
* Polygon Table contains coefficients of plane eqn for each surface, intensity information for the surfaces & pointers into the edge table.
* To facilitate the search for surfaces crossing the scan line, set up an active list of edges from information in the edge table.

This active list will contain only edges that cross the current scan line.

* We define a flag for each surface that is set ON or OFF to indicate whether a position along a scanline is inside or outside of the surface.
* Scanlines are processed from left to right.
* At the leftmost boundary of a surface, the surface flag is turned ON & at the rightmost boundary it is turned OFF.



* The active list for scanline 1 contains information from the edge table for edges AB, BC, EH, FG.
* For positions along the scanline b/w AB & BC only the flag for surface S1 is ON.
∴ No depth calculation are necessary & intensity information for surface S1 is entered from polygon table into refresh buffer.
* Similarly, b/w edges EH & FG only flag for surface S_2 is ON.
No other positions along scanline 1 intersect surfaces, so the intensity values in other areas are set to the background intensity.

* For scanline 2 & 3, the active edges contains edges AD, EH, BC, FG.

* Along scanline 2 from edge AD to edge EH, only flag for surface S_1 is ON.

* But b/w edges EH & BC, the flag for both surface are ON.

  In this interval, depth calculations must be made using plane coefficients for the two surfaces.

* The depth of surface S_1 is assumed to be less than that of S_2, so intensities for surface S_1 are loaded into the refresh buffer until boundary BC is encountered.

* Then the flag for surface S_1 goes off & intensities for surface S_2 are stored until edge FG is passed.