Unit 7: Visible Surface detection

Visible surface detection or Hidden surface removal is major concern for realistic graphics for identifying those parts of a scene that are visible from a chosen viewing position. Several algorithms have been developed. Some require more memory, some require more processing time and some apply only to special types of objects.

Visible surface detection methods are broadly classified according to whether they deal with objects or with their projected images.

Types:

1. Object –space methods
2. Image-space methods

Object-space methods:

- Deals with object definitions directly.
- Compare objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.
- It is a continuous method.
- Display is more accurate but computationally more expensive as compared to image space.

Image Space methods:

- Deals with the projected images of the objects and not directly with objects.
- Visibility is determined point by point at each pixel position on the projection plane.
- It is a discrete method.
- A change of display resolution requires recalculation.

A. Back Face Detection Method (Plane Equation method)

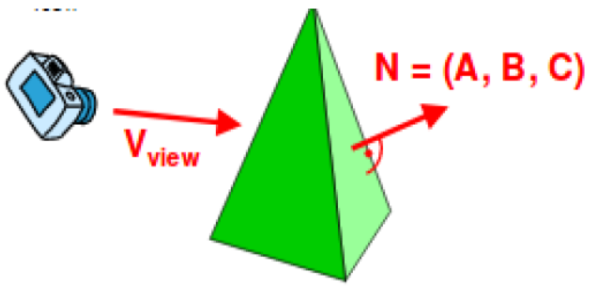
In a solid object, there are surfaces which are facing the viewer (front faces) and there are surfaces which are opposite to the viewer (back faces). These back faces contribute to approximately half of the total number of surfaces. Since we cannot see these surfaces anyway, to save processing time, we can remove them before the clipping process with a simple test.

Each surface has a normal vector. If this vector is pointing in the direction of the center of projection, it is a front face and can be seen by the viewer. If it is pointing away from the center of projection, it is a back face and cannot be seen by the viewer.

The test is very simple, if the z component of the normal vector is positive, then, it is a back face. If the z component of the vector is negative, it is a front face.

Principle:

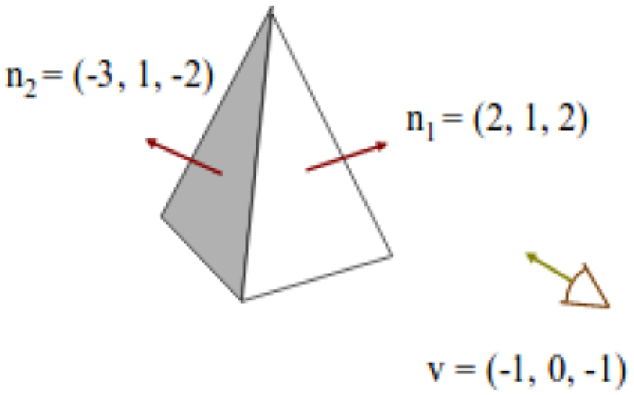
- Remove all surfaces pointing away from the viewer
- Eliminate the surface if it is completely obscured by other surfaces in front of it
- Render only the visible surfaces facing the viewer



Back facing and front facing faces can be identified using the sign of V • N where V is the view vector and N is normal vector.

- If V • N > 0, back face
- if V • N < 0 front face

Q. Find the visibility of n1 and n2 from V.



n1 • v = (2, 1, 2) ·(-1, 0, -1) = -2 – 2 = -4,

i.e n1 • v< 0

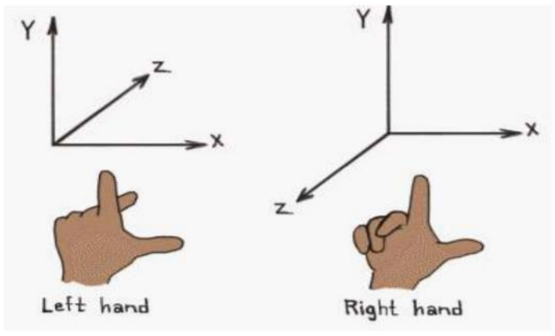
So, n1 front facing polygon

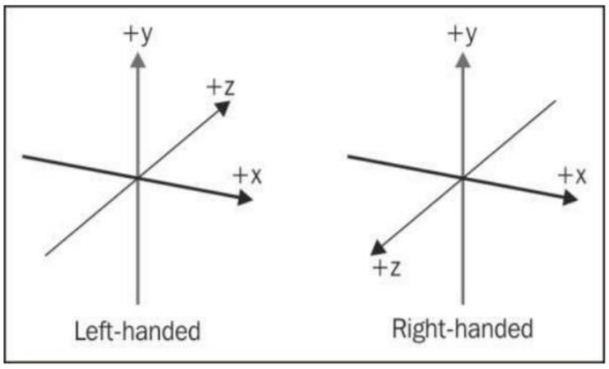
n2 • v = (-3, 1, -2) ·(-1, 0, -1) = 3 + 2 = 5

i.e n2 • v > 0

so n2 back facing polygon

Right hand and Left hand Coordinate System





B. Depth Buffer (Z-Buffer) Method

This method is developed by Edwin Catmull and is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest (visible) surface.

In this method each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared & the closest surface determines the color to be displayed in the frame buffer.

According to the direction of the user we take maximum Z value or minimum Z values to initialize the Z buffer.

No need to compare Z value of those pixels which are not overlapped with each other.

Two buffers are used:

Z (Depth) Buffer: To store the depth values for each (X,Y) position, as surface are processed.

Refresh (frame) Buffer: To store the intensity value at each position (X,Y).

The Z- coordinates are usually normalized to the range [0,1].

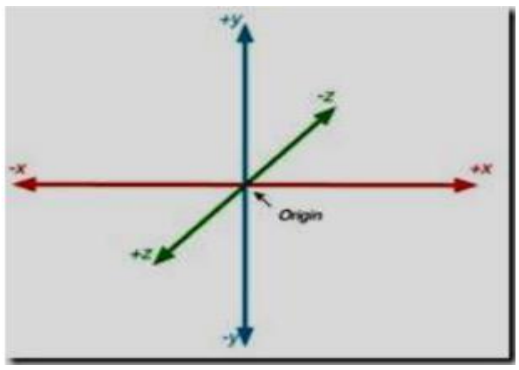
TWO CASE MUST BE CONSIDER IN CASE OF Z-BUFFER ALGORITHM

Case 1: When user sits on positive z axis and looks towards negative z axis:

In first case the object with smaller Z value is far from user and object having more Z value is closer to the user.

Case 2: When user sits on negative z axis and looks towards positive z axis:

In second case the object with larger Z value is far from user and object having smaller z values are closer to the user.



When Viewing direction is from + Z to –Z (case 1)

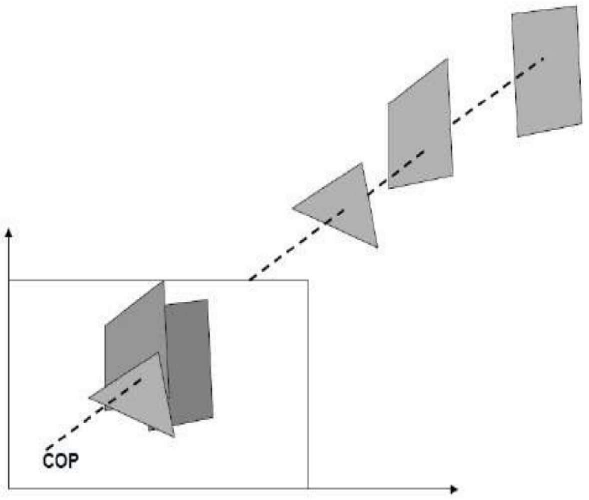
Algorithm

Step-1 − Set the buffer values −

- Depthbuffer (x, y) = 0
- Framebuffer (x, y) = background color

Step-2 − Process each polygon (One at a time)

- For each projected (x, y) pixel position of a polygon, calculate depth z.
- If Z > depthbuffer (x, y)
  - Compute surface color,
  - set depthbuffer (x, y) = z,
  - framebuffer (x, y) = surfacecolor (x, y)



When Viewing Direction is from –Z to +Z.

Algorithm

Step 1 : Set the buffer values:

Depthbuffer (x,y)=1 (maximum value)

Framebuffer (x,y)=background color

Step 2: Process each polygon (one at a time)

For each projected (x,y) pixel position if a polygon, calculate depth z.

If z< depthbuffer (x,y)

Compute surface color

Set depthbuffer(x,y)=z,

Framebuffer (x,y)=surfacecolor(x,y)

After all surface are processed, the depth buffer contains the depth value of the visible surface & the frame buffer contains the corresponding intensity values for the surface. The depth value of the surface position (x,y) is calculated by plane equation of the surface.

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

$$Z=(-Ax-By-D)/C$$

The depth value of the next position (x+1,y) on the scan line can be obtained using

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

Limitation:

This algorithm do not work on transparent surface, this is only for opaque surface.

C. Scan Line Method

Scan – Line Method

This image-space method for removing hidden surfaces is an extension of the scan-line algorithm for filling polygon interiors where, we deal with multiple surfaces rather than one. Each scan line is processed with calculating the depth for nearest view for determining the visible surface of intersecting polygon. When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.

To facilitate the search for surfaces crossing a given scan line, we can set up an ***active list*** of edges from information in the edge table that contain only edges that cross the current scan line, sorted in order of increasing ***x***. In addition, we define a ***flag*** for each surface that is set **on** or **off** to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.

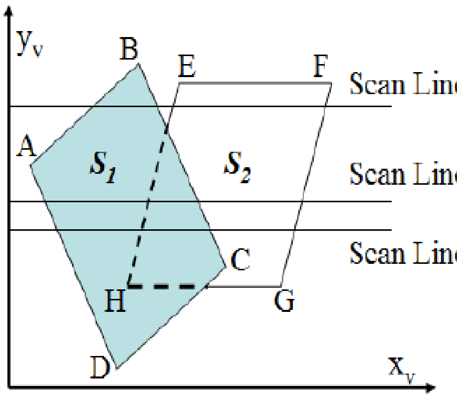


Fig. Scan lines crossing the projection of two surfaces, S_1 and S_2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

The active list for scan line-1 contains information from the edge table for edges AB, BC, EH, and FG. For positions along this scan line between edges AB and BC, only the flag for surface S_1 is on. Therefore, no depth calculations are necessary, and intensity information for surface S_1, is entered from the polygon table into the refresh buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is on. NO other positions along scan line-1 intersect surfaces, so the intensity values in the other areas are set to the background intensity.

For scan lines 2 and 3, the active edge list contains edges AD, EH, BC, and FG. Along scan line 2 from edge AD to edge EH, only the flag for surface S_1, is on. But between edges EH and BC, the flags for both surfaces are on. In this interval, depth calculations must be made using the plane coefficients for the two surfaces. For this example, 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, and intensities for surface S_2 are stored until edge FG is passed.

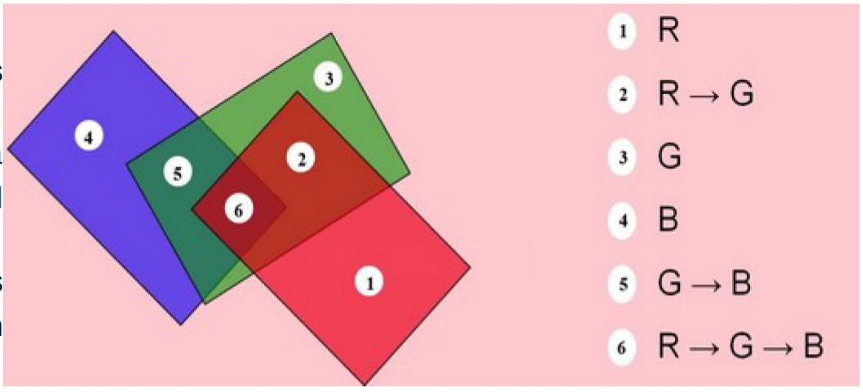
D. A- Buffer Method:

The A-buffer method is an extension of the depth-buffer method. The A-buffer method is a visibility detection method developed at Lucas film Studios for the rendering system Renders Everything You Ever Saw (REYES).

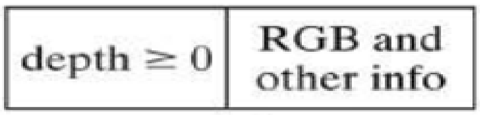
The A-buffer expands on the depth buffer method to allow transparencies. The key data structure in the A-buffer is the accumulation buffer.

Each position in the A-buffer has two fields −

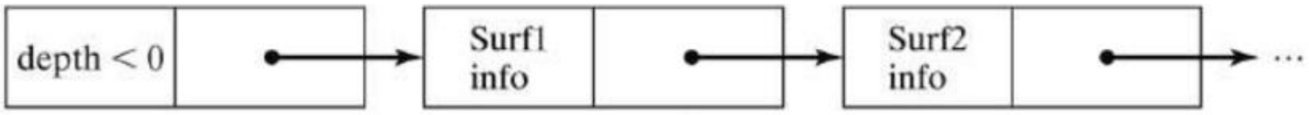
- Depth field − It stores a positive or negative real number
- Intensity field − It stores surface-intensity information or a pointer value



If depth >=0, the number stored at that position is the depth of a single surface.



If depth<0, it indicates multiple surface contributions to the pixel intensity. The intensity filed then stores a pointer to a linked list of surface data.



Eg.

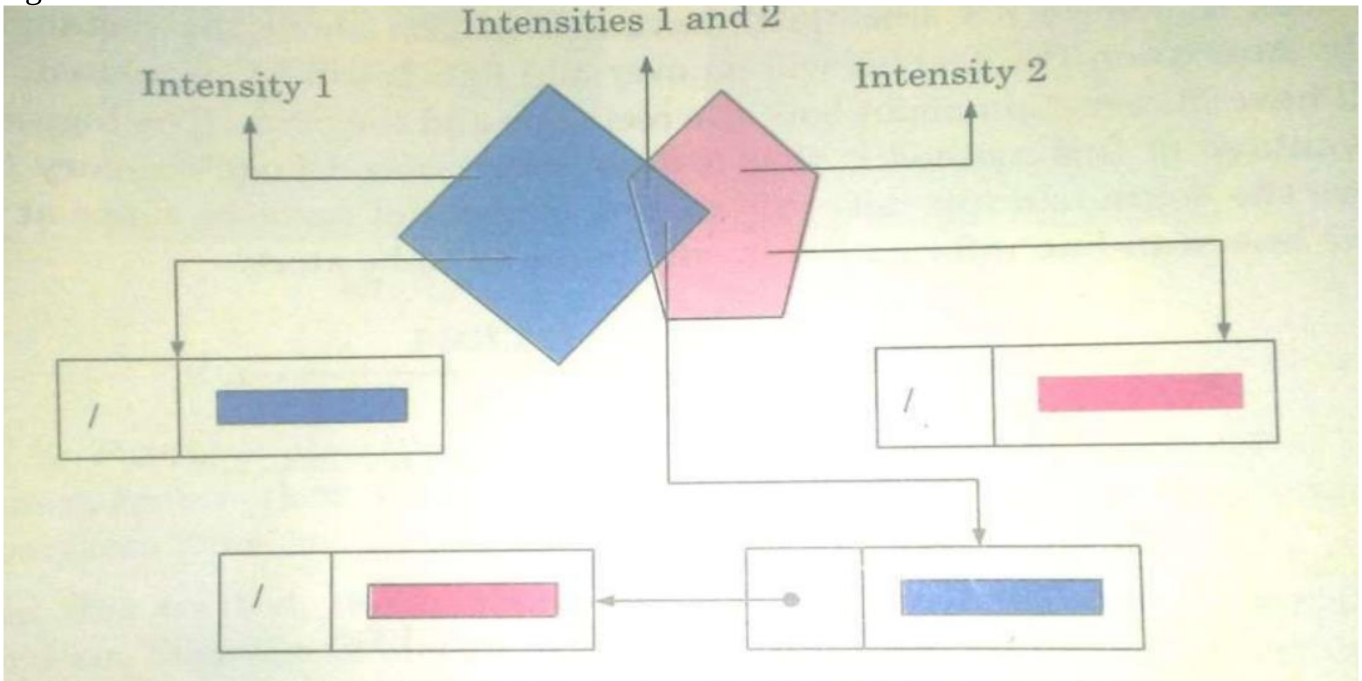


FIGURE 8.7 Link list associated with different parts of a scene.

E. Depth Sorting Algorithm (Painter's Algorithm)

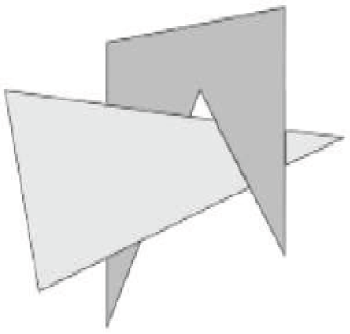
Depth sorting method uses both image space and object-space operations. The depth-sorting method performs two basic functions −

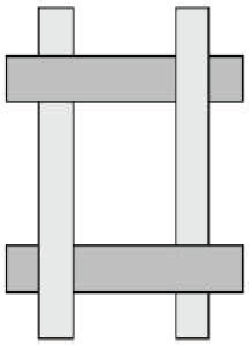
- First, the surfaces are sorted in order of decreasing depth.
- Second, the surfaces are scan-converted in order, starting with the surface of greatest depth.

The scan conversion of the polygon surfaces is performed in image space. This method for solving the hidden-surface problem is often referred to as the **painter's algorithm.**

The algorithm begins by sorting by depth.

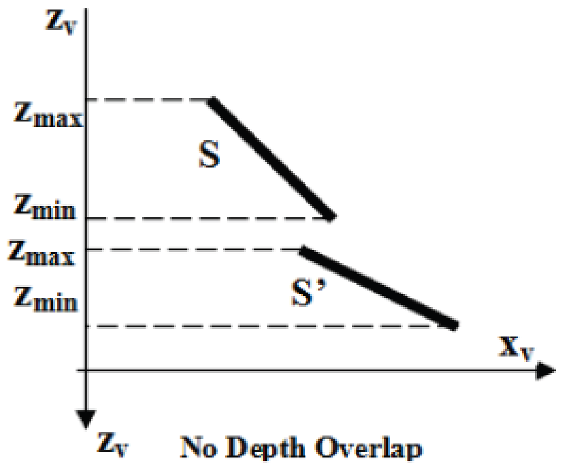
- ➢ Sort all surfaces according to their distances from the view point.
- ➢ Render the surfaces to the image buffer one at a time starting from the farthest surface.
- ➢ Surfaces close to the view point will replace those which are far away.
- ➢ After all surfaces have been processed, the image buffer stores the final image.



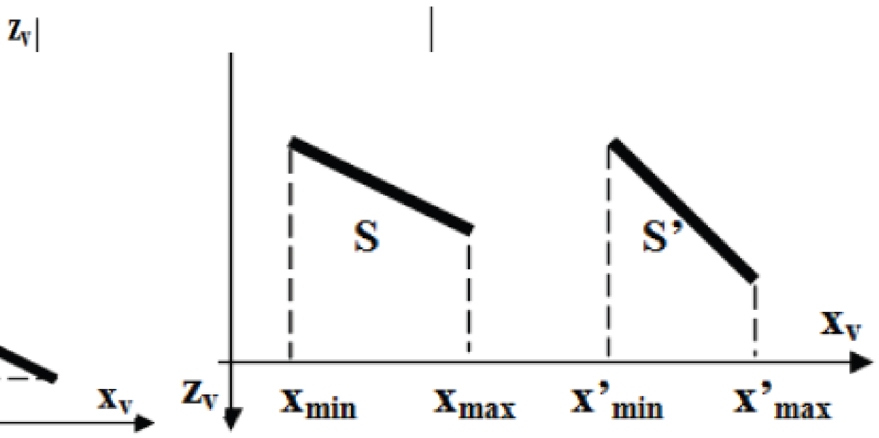


Example:

Assuming we are viewing along the z axis. Surface S with the greatest depth is then compared to other surfaces in the list to determine whether there are any overlaps in depth. If no depth overlaps occur, S can be scan converted. This process is repeated for the next surface in the list. However, if depth overlap is detected, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.



No Depth Overlap

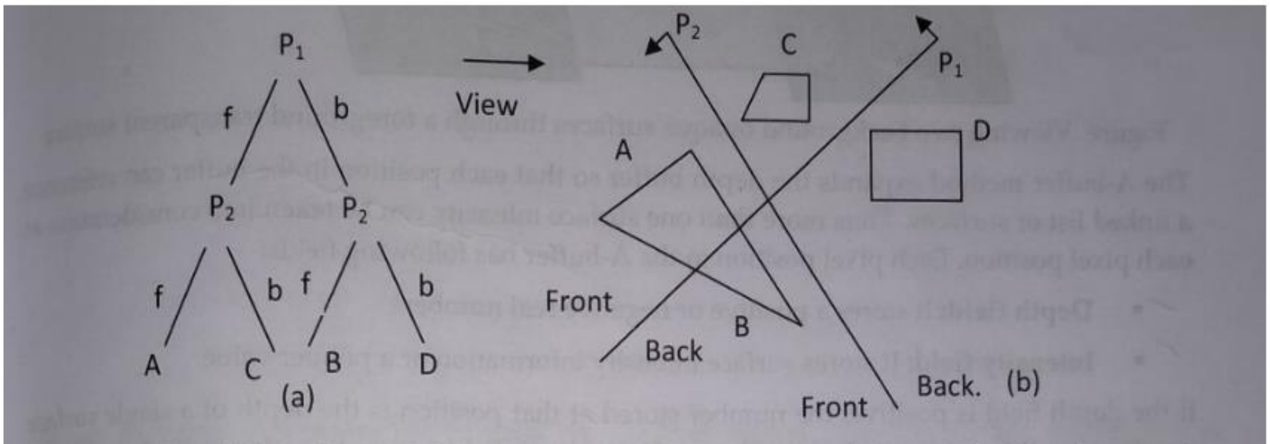


Depth overlap but no overlap in x direction

Binary Space Partitioning (BSP) Tree Method:

- Split the surface in to two, those in front of the selected polygon and those who are in the back of the polygon.

- Split any polygon lying on both sides.
- This process is repetitive until polygon is not sorted.
- The result of this sorting can be visualize in the form of binary tree.
- In the resultant binary tree one subtree shows the polygon who are in front side and other subtree shows the polygon who are in the back.



Here plane P_1 partitions the space into two sets of objects, one set of object is back and one set is in front of partitioning plane relative to viewing direction. Since one object is intersected by plane P_1, we divide that object into two separate objects labeled A and B. Now object A&C are in front of P_1, B and D are4 back of P_1. We next partition the space with plane P_2 and construct the binary free as fig (a). In this tree, the objects are represented as terminal nodes, with front object as left branches and behind object as right branches. When BSP tree is complete, we process the tree by selecting surface for displaying in order back to front. So for ground object are painted over back ground objects.

Octree (Refer Unit 6)

Ray Tracing method (HW)