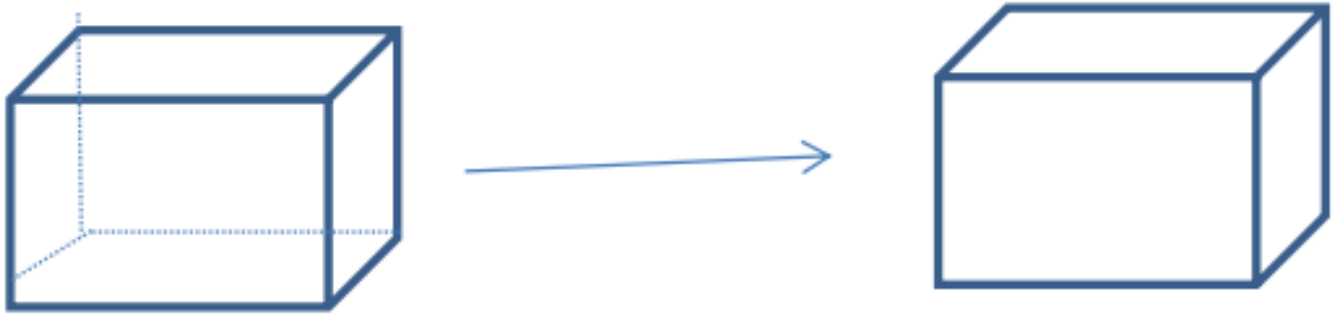
CSC209: Computer Graphics

Unit 7 – Visible Surface Detections

SUSAN SUNUWAR

Visible Surface Detection

- Visible surface detection is the process of identifying those parts of a scene that are visible from a chosen viewing position.
- There are numerous algorithms for efficient identification of visible objects for different types of applications. These various algorithms are referred to as visible-surface detection methods or hidden surface elimination methods.
- When a picture contains non-transparent objects, the surfaces behind the objects can not be viewed. To obtain a realistic screen image, the hidden surfaces need to be removed. This process of identification and removal of these surfaces is known as hidden-surface problem. These processes helps the computer to decrease processing time for rendering image, eliminating extra information of image.



A cube

Hidden Surfaces Removed

Visible Surface Detection

- Hidden surface determination is necessary to render an image correctly, so that one may not view features hidden behind the model itself, allowing only the naturally viewable portion of the graphics to be visible.
- Visible surface detection methods are broadly classified according to whether they deal with objects or with their projected images. These two approaches are:
  - **Object space method**
  - **Image space method**

Object space method

- An object space method compares objects and parts of objects to each other within scene definition to determine which surfaces are visible.
- Line display algorithms generally use object space methods to identify visible lines in wireframe displays.
- This method is easy to implement but is slower.
- This method is implemented in physical coordinate system.

Image space method

- Visibility is decided point by point at each pixel position on the projection plane.
- It is implemented in screen coordinate system.
- Most of the line/surface detection algorithms use image space method.

Back Face Detection (Plane equation method)

- A fast and simple **object space method** used to remove hidden surface from 3D object drawing is known as “Plane equation method”.
- **inside-outside test (CASE I)**
- A point (x, y, z) is inside a surface with plane parameter A, B, C & D if $\boldsymbol{Ax + By + Cz + D < 0}$ **,**i.e., it is in back face.

Back Face Detection (Plane equation method)

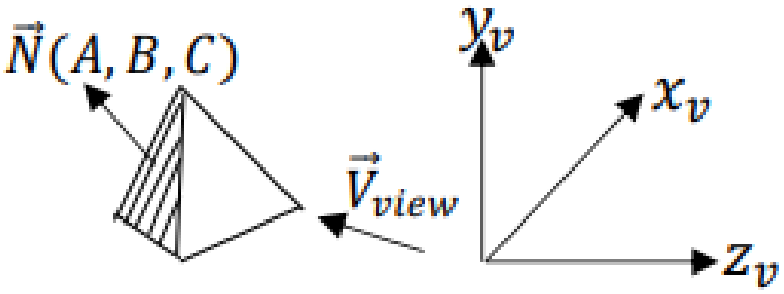
- **(CASE II)** Each surface has a normal vector. If this normal points is in the direction of center of projection, then it is front face and can be seen by viewer. If this normal is pointing away from the direction of center of projection, then it is back face and cannot be seen by viewer.
- Let $\vec{N}$ be the normal vector for a polygon with cartesian components (A, B, C) and $\overrightarrow{V_{view}}$ be the vector in viewing direction, then a polygon surface is back if, $V_{view}.N > 0$

$$V_{view}.N = |V|.|N|cos\theta, \qquad 0 \leq cos\theta \leq 1$$

$$0 \leq \theta \leq \frac{\Pi}{2}$$

Back Face Detection (Plane equation method)

- For the left-handed viewing system if the 'z' component of the normal vector is positive, then it is back face. It the 'z' component is negative then it is a front face.
- For the right-handed viewing system if the 'z' component of the normal vector is negative, then it is back face. It the 'z' component is positive then it is a front face.



Algorithm for right-handed system

(CASE III)

Step 1: Compute N for every face of object.

Step 2: If $(V_z.C < 0)$ then a back face and don't draw

else front face and draw

$$[\boldsymbol{V_{view}} = (\boldsymbol{0, 0, V_z})\ \boldsymbol{and}\ \boldsymbol{N} = (\boldsymbol{A_i, Bj, Ck}), \boldsymbol{where}\ (\boldsymbol{i, j, k})$$

$$= \boldsymbol{unit\ vector}]$$

$$[\boldsymbol{V_{view}.N = V_z.C}]$$

Algorithm for left-handed system

Step 1: Compute N for every face of object.

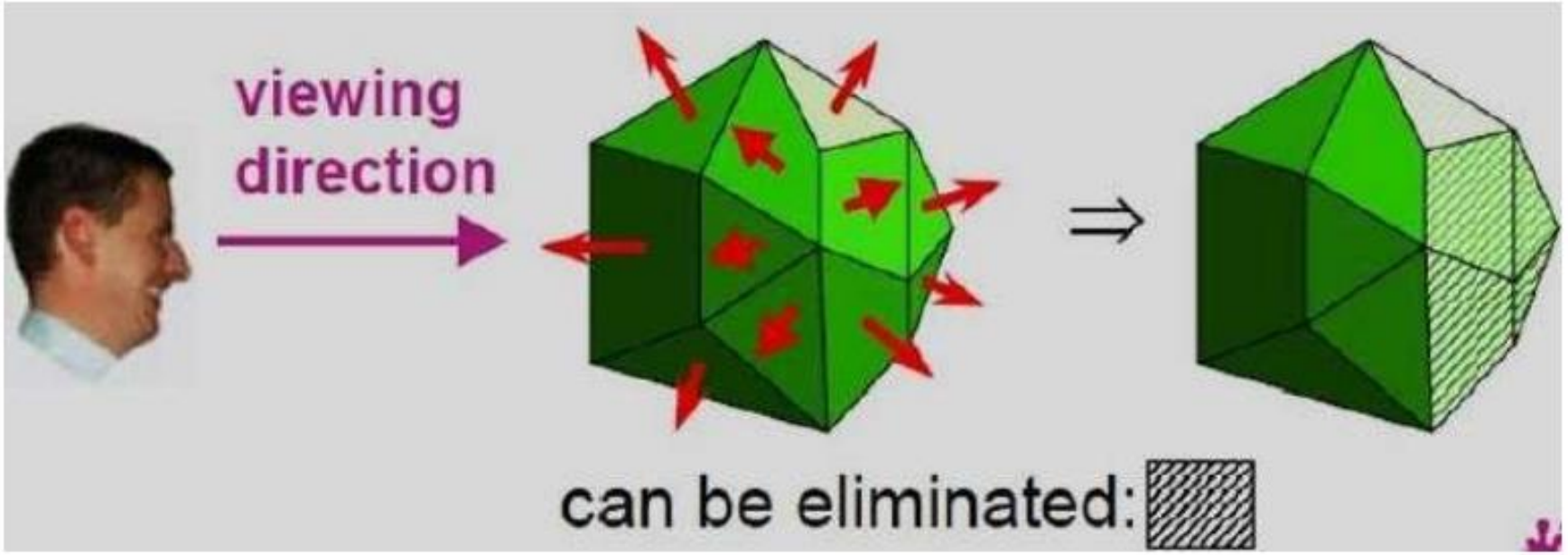
Step 2: If $(V_z.C > 0)$ then a back face and don't draw

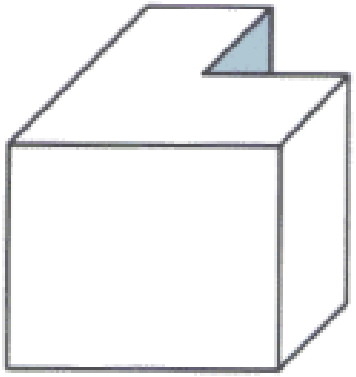
else front face and draw

$$[\boldsymbol{V_{view}} = (\boldsymbol{0, 0, V_z})\ \boldsymbol{and}\ \boldsymbol{N} = (\boldsymbol{A_i, Bj, Ck}), \boldsymbol{where}\ (\boldsymbol{i, j, k})$$

$$= \boldsymbol{unit\ vector}]$$

$$[\boldsymbol{V_{view}.N = V_z.C}\ ]$$





For concave polyhedral or overlapping objects, we still need to apply other methods to further determine where the obscured faces are partially or completely hidden by other objects (e.g., Using Depth-Buffer Method or Depth-sort Method).

Pros and Cons of Back Face Detection

- It is simple and easy to implement.
- No pre-sorting of polygon surfaces is needed.
- It cannot work on partial hidden surfaces.
- It works only with non-overlapping (separate) objects.

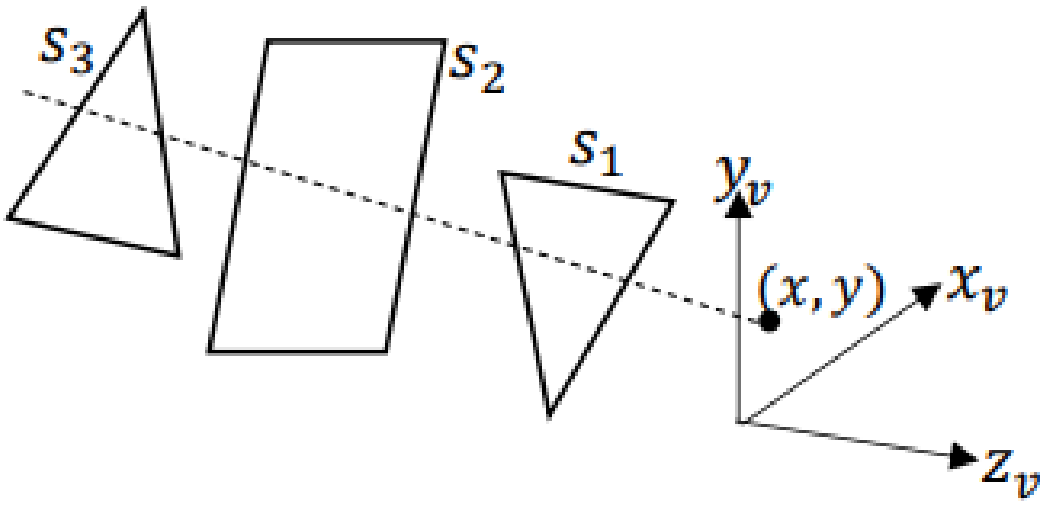
Depth Buffer Method (Z-buffer method)

- It is an **image space method** for detecting visible surface.
- The Z-buffer method compares surface depths of each pixel position on the projection plane. Normally, z-axis is represented as depth.
- In this method, two buffers are used:
  - **Depth buffer**: stores depth values for each (x, y) position

    $(0 \leq depth \leq 1)$
  - **Frame buffer**: stores the intensity values for each position

Depth Buffer Method (Z-buffer method)

- This approach compares surface depths at each pixel position on the projection plane. The depth values for a pixel are compared and the closest surface determines the color to be displayed in the frame buffer.
- Each surface of a scene is processed separately, one point at a time across the surface. And each (x, y, z) position on a polygon surface corresponds to the orthographic projection point (x, y) on the view plane.
- It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers named frame buffer and depth buffer, are used.


s_3
s_2
s_1
y_v
(x, y)
x_v
z_v

Depth Buffer Method (Z-buffer method)

- In figure, three surface S_1, S_2, S_3 at varying distance from view plane $x_v y_v$ the projection along (x, y) is done.
- Surface S_1 is closest to the view-plane so surface intensity of S_1 at (x, y) is saved.
- Initially, all the positions in depth buffer is set to 0 and refresh buffer is initialized to background color. Each surface listed in polygon table is processed one scan line at a time, calculating the depth (z-value) for each position (x, y). The calculated depth is compared to the value previously stored in depth buffer at that position. If calculated depth is greater than stored depth value in depth buffer, new depth value is stored and the surface intensity at that position is determined and placed in refresh buffer.

Algorithm

Step 1: For all buffer positions (x, y), initialize the buffer values;

- depth_buffer $(x, y) = 0$
- frame_buffer (x, y) = background color

Step 2: Process each polygon surface P (One at a time)

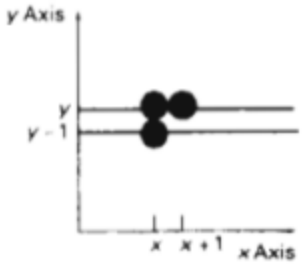
- For each projected (x, y) pixel position of polygon P, calculate depth $(x, y) = z$.
- If z > depth_buffer (x, y) then
  - Compute surface color
  - Set depth_buffer $(x, y) = z$,
  - frame_buffer $(x, y) = surface\_color\ (x, y)$, where $surface\_color\ (x, y)$ is the intensity value for the surface at pixel position (x, y).

Depth Buffer Method (Z-buffer method)

- After all surfaces are processed, the depth value of surface position (x, y) is calculated by plane equation surface.

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

- Let depth z' at position $(x + 1, y)$



$$z' = \frac{-A_{x+1} - By - D}{C}$$

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

- $-\frac{A}{C}$ is constant for each surface so succeeding depth value across a scan line are obtained from preceding values by simple calculation.

Pros and Cons of Depth Buffer Method

- It is simple and easy to implement, no specific hardware is needed.
- No pre-sorting of polygons is needed.
- Can be applied to non-polygonal objects.
- It only deals with opaque surfaces.
- It requires additional buffer and large memory.
- It is time consuming process.

A-buffer Method (Accumulation buffer Method)

- A-buffer method is an extension of depth-buffer method.
- The A-buffer method represents an area-averaged, accumulation-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 and cannot accumulate intensity values for more than one surface, as is necessary if transparent or translucent surfaces are to be displayed.

A-buffer Method (Accumulation buffer Method)

- The A-buffer method expands the depth buffer so that each position in the buffer can reference a linked list of surfaces.
- It maintains a data structure of background surfaces that are behind the foreground transparent surface. This special data structure is called accumulation buffer.
- Each position in the A-buffer has two fields:
  - **Depth field**: Stores a positive or negative depth value.
  - **Intensity Field (Surface data field)**: stores surface-intensity information or a pointer value. It includes: RGB intensity components, Opacity Parameter, Depth, Percent of area coverage, Surface identifier

Pros and Cons of A-buffer Method

- It provides anti-aliasing in addition to what Z-buffer does.
- It is slightly costly than Z-buffer method because it requires more memory in comparison to the Z-buffer method.

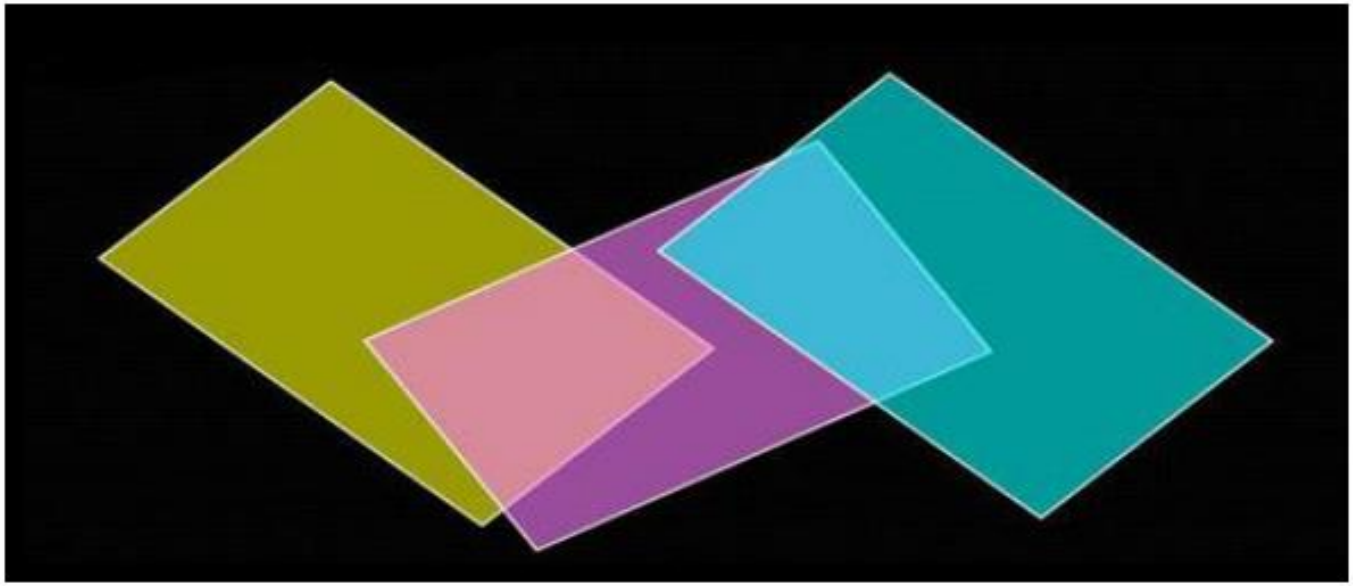


Fig: Viewing two background opaque surfaces through a foreground transparent surface

Fig: Organization of an A-buffer pixel position: (a) single-surface (b) multiple surfaces overlap

Scan Line Algorithm

- It is an image-space method for identifying visible surfaces. It computes and compares depth values along the various scan lines for the scenes. Surfaces are processed using information stored in the polygon table.
- An active list of edges is formed for each scan line which stores only those edges that crosses the scan line in order of increasing $'x'$.
- Also, a flag is set for each surface that is set on or off to indicate whether a position along a scan line is either inside or outside the surface.
- Pixel position across each scan-line are processed from left to right. At the left intersection with a surface the surface flag is turned on and at the right intersection point the flag is turned off.
- Across each scan line, depth calculations are made for each overlapping surface to determine which surface 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.

Scan Line Algorithm

- It requires an edge table, polygon table, active edge list and flag.
  - **Edge table**: It contains coordinate end points for each scan line, pointers into the polygon table to identify the surfaces bounded by each line.
  - **Polygon table**: It contains coefficients of plane equations for each surfaces, pointers into the edge table, and intensity information of the surfaces.
  - **Active edge list**: It contains edges that cross the current scan line, shorted in order of increasing x.
  - **Flag** is defined for each surface that is set ‘ON’ or ‘OFF’ to indicate whether a scan line is inside or outside of the surface. At the left most boundary surface flag is ‘ON’ and at right most boundary flag is ‘OFF’.

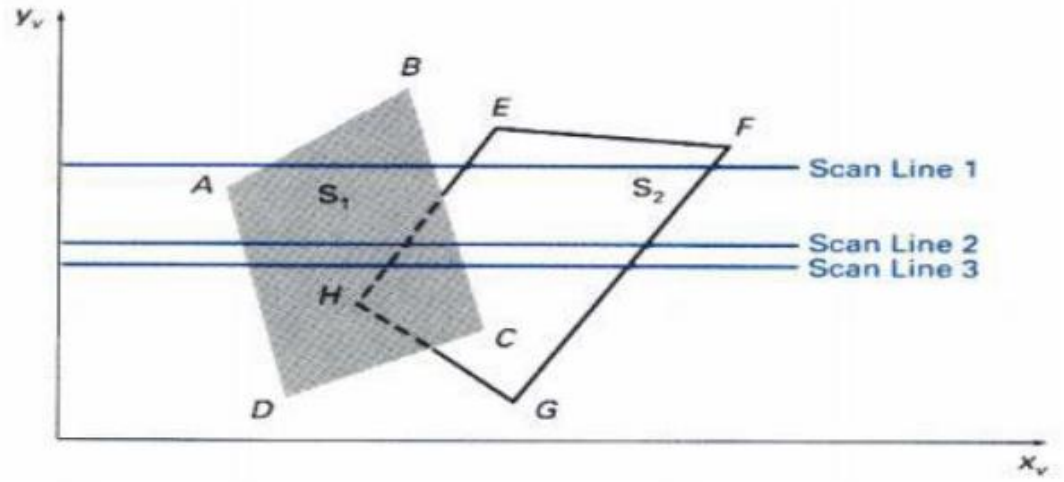


Fig: *Scan lines crossing the projection of two surfaces, S1, and S2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces.*

- **Active Edge Table**

SL1	AB	BC	EH	FG
SL2	AD	EH	BC	FG
SL3	AD	EH	BC	FG

	Edge pairs	Flags for S_1	Flags for S_2
For scan line 1	AB & BC	on	off
	BC & EH	off	off
	EH & FG	off	on
For scan line 2	AD & EH	on	off
	EH & BC	on	on
	BC & FG	off	on

Here, between EH & BC, the flags for both surfaces are ON so depth calculation is necessary. But between other edge pairs no depth calculation is necessary.

Working Procedure

- Above figure illustrate the scan line method for locating visible portion of surfaces for pixel position along the line. The active list for scan line one contains information form the edge table for edges AB, BC, EH and FG.
- For scan line 1, between edges AB and BC, flag (S_1) = ON and flag (S_2) = OFF. 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 edge EH and FG flag (S_1) = OFF and flag (S_2) = ON.
- For scan line 2 and 3, the active edge list contains edges AD, EH, BC and FG. Along the scan line 2, from edge AD to edge EH, flag (S_1) =ON and flag (S_2) = OFF.

Working Procedure

- Between edges EH and BC, Flag (S_1) = ON and flag (S_2) = ON. Therefore, depth calculation must be made using the plane coefficient for the two surfaces.
- For this example, the depth of surface S_1 is assumed to be less than that of S_2, So intensity for surface S_1 are loaded into the refresh buffer until boundary BC is encountered. Then the flag for the surface S_1 goes off. And intensity for surface S_2 is stored until edge FG is passed.

Pros and Cons of Scan-line algorithm

- Any number of polygon surfaces can be processed with this method. Depth calculations are performed only when there are overlapping polygons.
- Deals with transparent, translucent, and opaque surfaces.
- Can be applied to non-polygonal objects.
- Complex.

Depth sorting Method (Painter's Algorithm)

- Also known as list priority algorithm.
- A visible surface detection method that used both image-space and object-space operations.
- This algorithm is also called "Painter's Algorithm" as it simulates how a painter typically produces his/her painting by starting with the background and then progressively adding new (nearer) objects to the canvas.

Basic Procedure [non-overlapping]

- All surfaces in the scene are ordered according to the **smallest** $'z'$ value on each surface.
- The surface $'S'$ at the end of the list is then compared against all other surface to see if there are any **depth overlap**.
- If no overlap occurs, then the surface is scan converted and the process is repeated with the next surface.
- The intensity values for farthest surface are then entered into the refresh buffer. That is **farthest polygon is displayed first**, then the second farthest polygon, so on, and finally, the closest polygon surface.
- After all surfaces have been processed, the refresh buffer stores the intensity values for all visible surfaces.
- When there are only a few objects in the scene, this method can be very fast. However, as the number of objects increases, the sorting process can become very complex and time consuming.

Basic Procedure [overlapping]

When there is depth overlap with 'S' following tests are done:

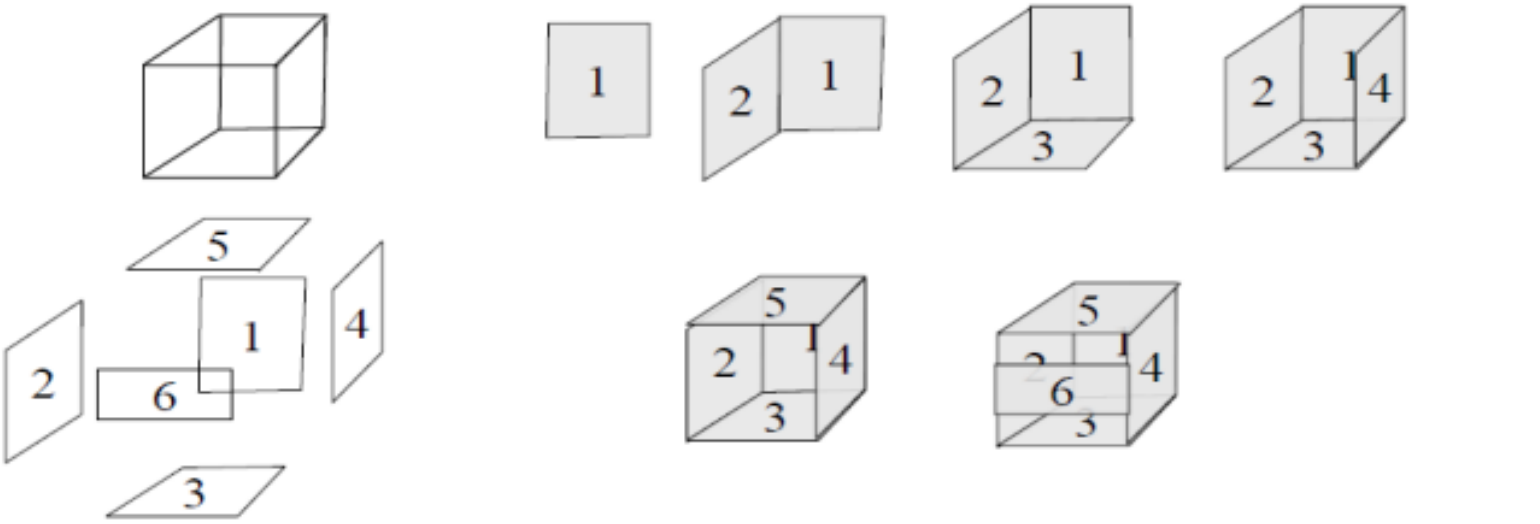
a) The boundary rectangle for the two surfaces do not overlap.

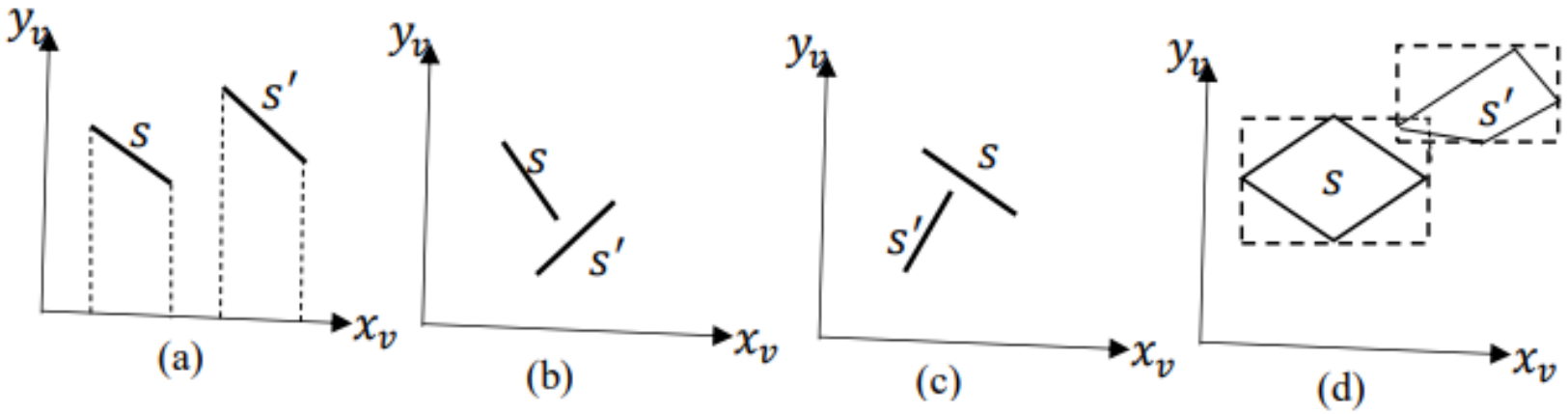
b) Surface 'S' is completely behind the overlapping surface relative to viewing position.

c) The overlapping surface is completely in front of 'S' related to viewing position.

d) The boundary edge projection of the two surfaces do not overlap.

First draw the
distant objects than the closer objects. Pixels of each object overwrites the previous objects.





- If any of this test is true, no reordering is necessary. But if all four tests fail then surface S and S' must be interchanged in sorted list.

Pros and Cons of Depth Sorting Algorithm

- For simple cases (no overlap of surfaces, z-values); very easy to implement.
- Sorting of polygons is required.
- Algorithm can fail in some cases, including intersecting polygons and cyclic overlap. [solution is to split polygons]
- Complex.

Binary Space Partition Tree Method (BSP Tree Method)

- A binary space partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces onto the screen from back to front as in the painter's algorithm.
- A BSP tree is a recursive sub-division of space that treats each line segment (or polygon, in 3D) as a cutting plane which is used to categorize all remaining objects in the space as either being in “front” or in “back” of that plane.
- The BSP tree is particularly useful when the view reference point changes, but object in a scene are at fixed position (static group of 3D objects).
- Basic Idea: Sort the polygons for display in back-to-front order.

BSP Algorithm

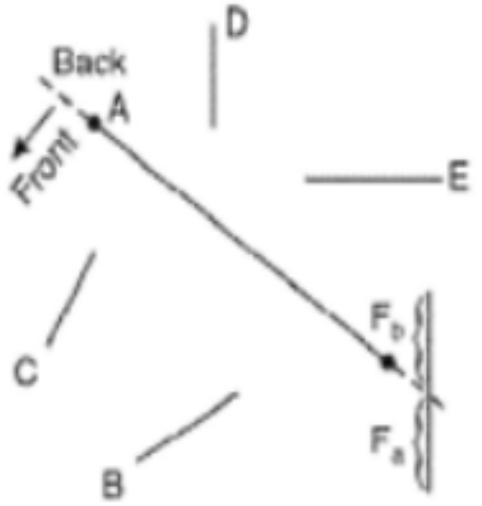
- Construct a BSP Tree
  - Select the partitioning plane (one of the polygon in the picture).
  - Partition the space into two sets of objects; surfaces that are inside/front and outside/back w.r.t. the partitioning plane.
  - In case of intersection, i.e., if object is intersected by partitioning plane, then divide object into two objects.
  - Identify surfaces in each of the half spaces.
  - Each half space is then recursively subdivided using one of the polygon in the half space as partitioning plane. This process is continuing till there is only one polygon in each half space.
  - Then the subdivided space is represented by a binary tree with the original polygon as the root.
  - Objects are represented as terminal nodes with front objects as left branches and back objects as right branches.

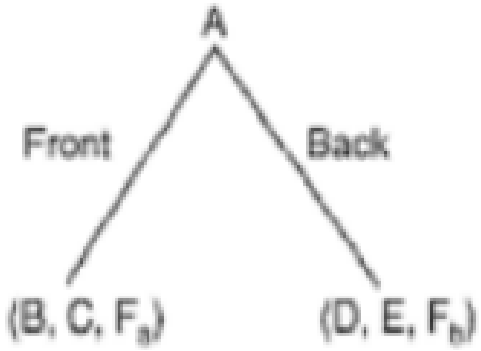
BSP Algorithm

- Display the BSP Tree
  - If the viewpoint is in front of the root polygon, then BSP tree is traversed in the order of back branch, root and then front branch i.e., reverse of in-order traversal.
  - If the viewpoint is behind the root polygon, then BSP tree is traversed in the order of front branch, root and then back branch i.e., normal in-order traversal.

1. Construction of BSP Tree

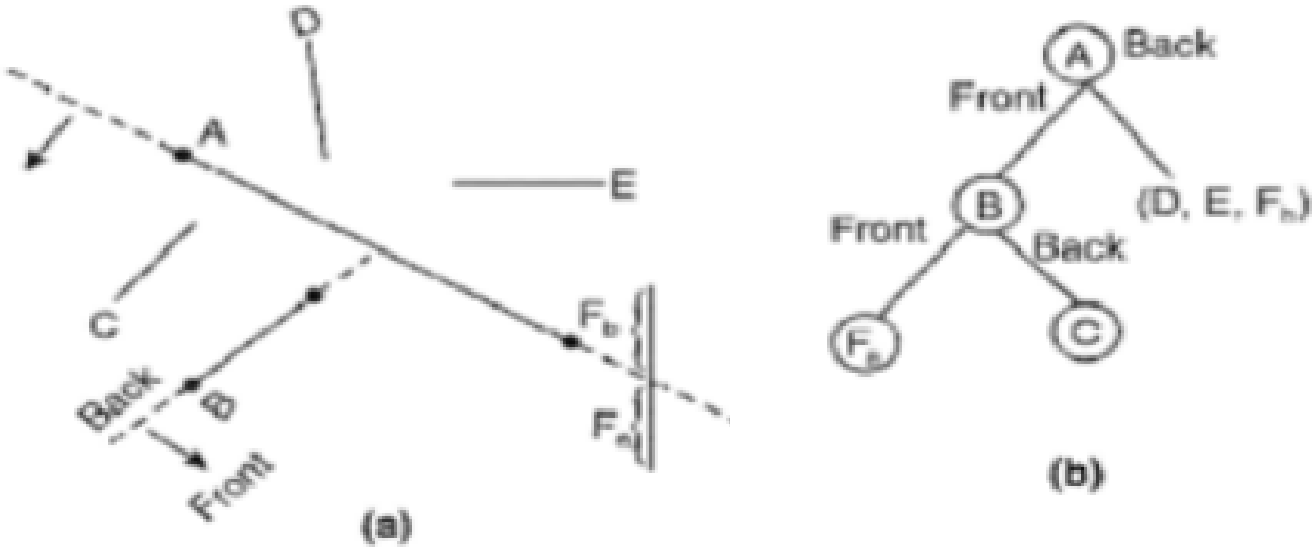
- Consider polygon A as a partitioning plane. Then on front side of A, the polygons will be B, C whereas on back side of A, the polygons will be D and E.
- Each half space is then recursively subdivided using one of the polygon in the half space as separating plane. This process of space subdivision continues till there is only one polygon in each half space, because this is a binary tree. Then the subdivided space is represented by a binary tree with the original polygon as the root.





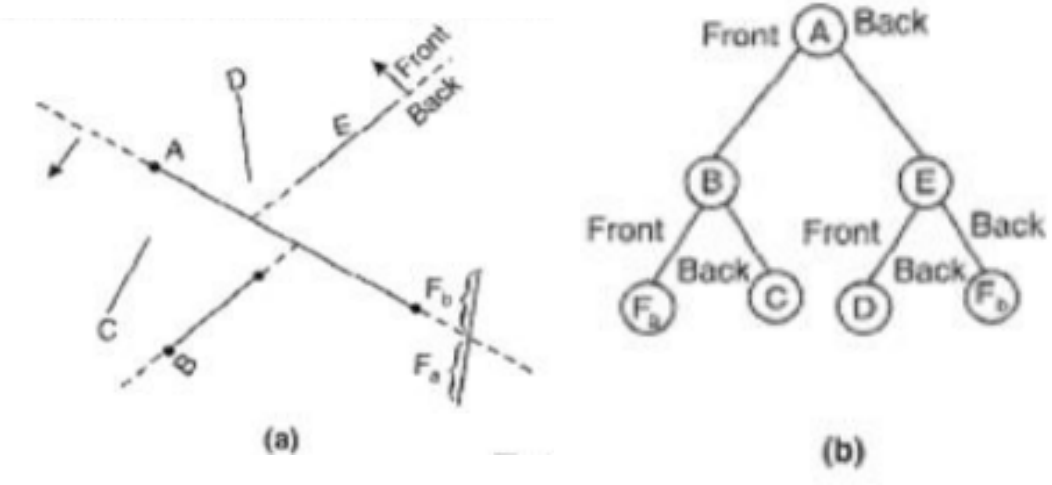
- Consider polygon A as a partitioning plane. Then on front side of A, the polygons will be B, C whereas on back side of A, the polygons will be D and E.

- **Front part**, there are B, C and F_a polygons. Select one polygon out of these as a dividing plane. Let us select polygon B.

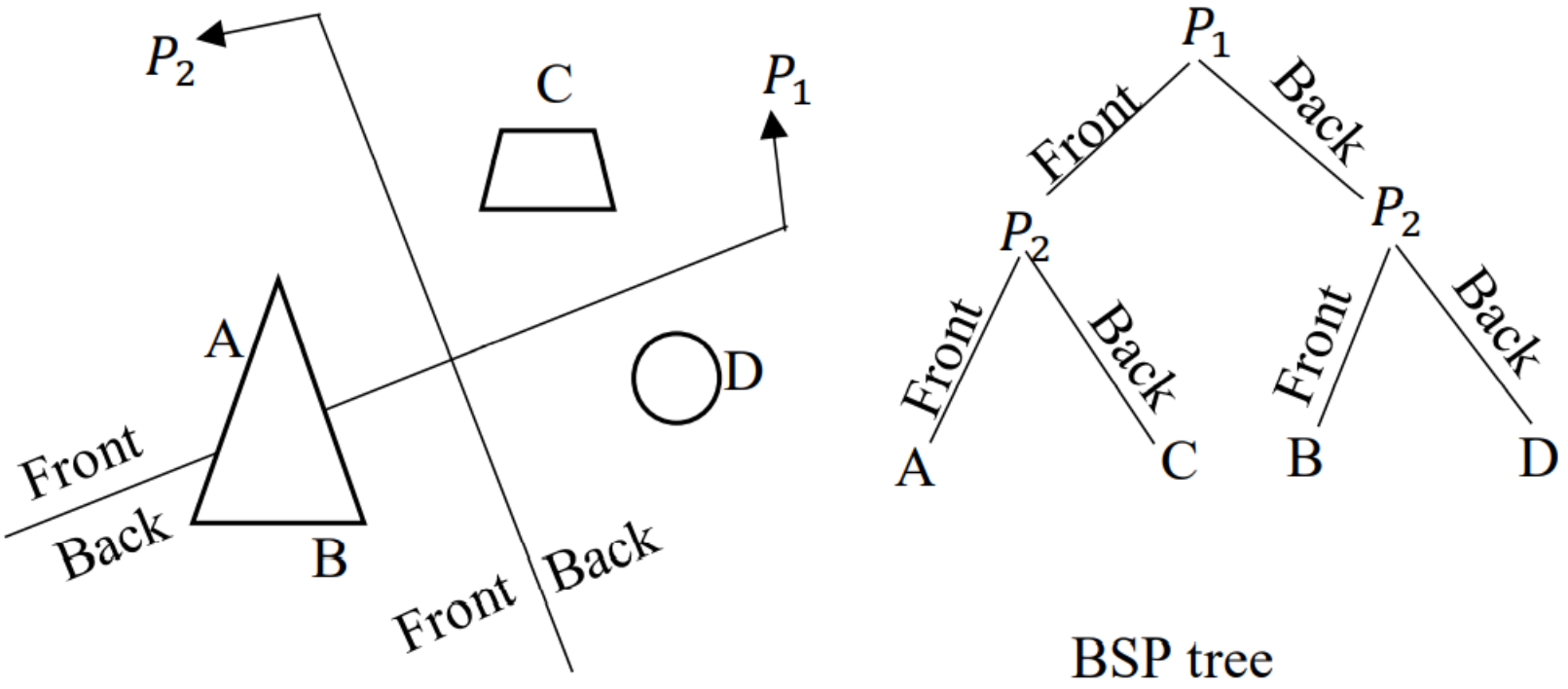


- It places polygon C in back and F_a in front. The BSP tree is shown in Fig. (b).
- F_a or C polygons cannot be further divided.

- **Back part**, there are D, E and F_b polygons. Select one polygon out of these as a dividing plane. Let us select polygon E.



- It places polygon D in front and F_b in back. The BSP tree is shown in Fig. (b).
- F_b or D polygons cannot be further divided. This is a final BSP tree


P_2
C
P_1
A
D
Front
Back
B
Front
Back
P_1
Front
Back
P_2
P_2
Front
Back
Front
Back
A
C
B
D

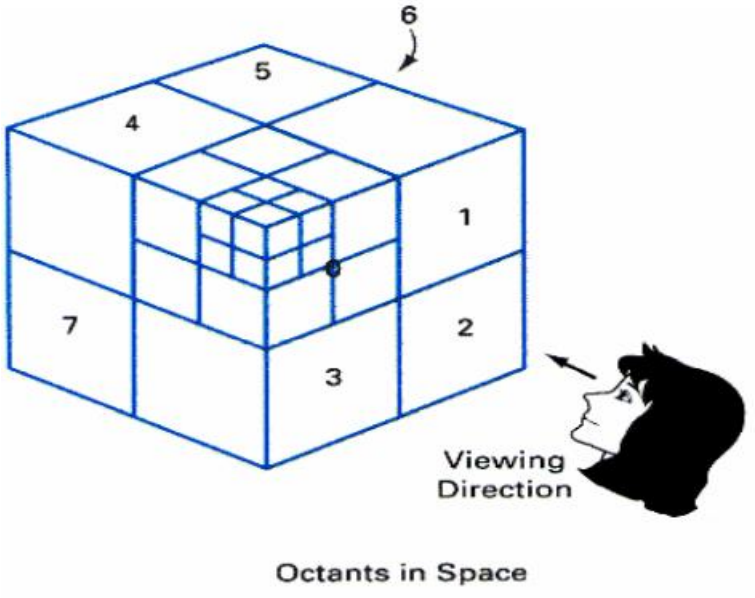
BSP tree

2. Display a BSP Tree

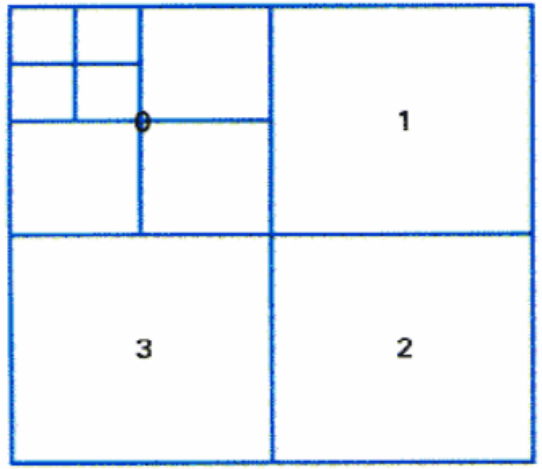
- If the viewpoint is in front of the root polygon, then BSP tree is traversed in the order of back branch, root, front branch, i.e., reverse of in-order traversal.
- For our example, if the viewpoint is in front of the root polygon, then the sequence of polygons for display will be F_b, E, D, A, C, B, F_a.
- If the viewpoint is behind the root polygon, then BSP tree is traversed in the order of front branch, root, back branch, i.e., normal in-order traversal.
- For our example, if the viewpoint is behind the root polygon, as shown then the sequence of polygons for display will be F_a, B, C, A, D, E, F_b.

Octree Method

- In octree method, octree nodes are projected onto the viewing surface in a front-to-back order as shown in figure below.



Octants in Space



Quadrants for
the View Plane

Octree Method

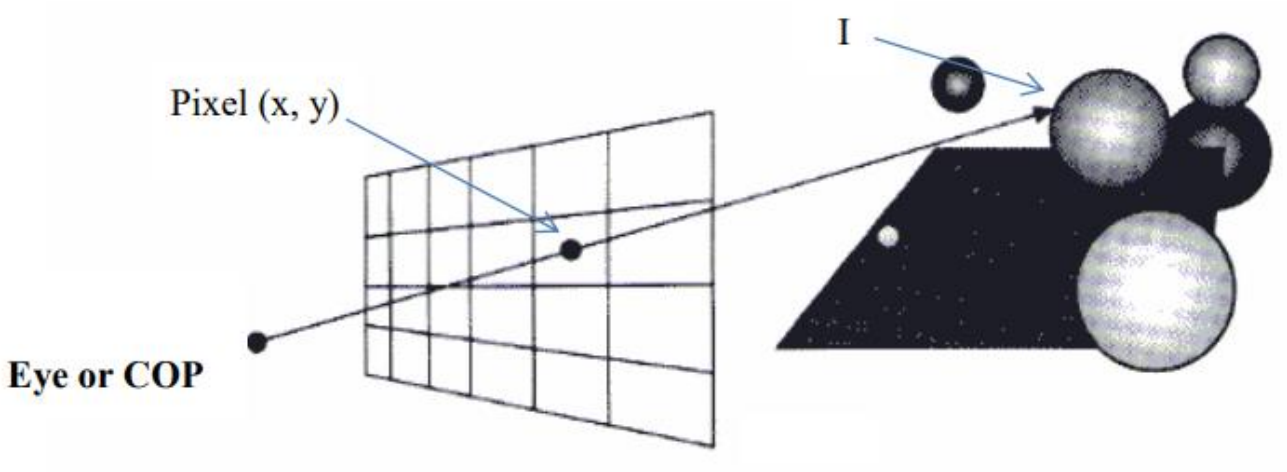
- Any surfaces toward the rear of the front octants (0, 1, 2, 3) or in the back octants (4, 5, 6, 7) may be hidden by the front surfaces.
- With the numbering method (0, 1, 2, 3, 4, 5, 6, 7), nodes representing octants 0, 1, 2, 3 for the entire region are visited before the nodes representing octants 4, 5, 6, 7.
- Similarly, the nodes for the front four sub octants of octant 0 are visited before the nodes for the four back sub octants.
- When a color is encountered in an octree node, the corresponding pixel in the frame buffer is painted only if no previous color has been loaded into the same pixel position.

Algorithm

- If the front octant is homogeneously filled with some color, we do not process the back octant.
- If the front is empty, it is necessary only to process the rear octant.
- If the front octant has heterogeneous regions, it must be subdivided, and the sub-octants are handled recursively.

Ray Tracing Algorithm

- Ray tracing also known as ray casting is efficient image space method for visibility detection in the objects.
- In Ray-tracing algorithm the basic idea is to trace light rays from the viewpoint through pixels until it reaches a surface. Since each ray can intersect several objects, find the intersection point which is closest to the viewpoint. And set the pixel to the color of the surface at the point where the light ray strikes it.
- Ray-tracing algorithm provides the flexibility to handle both flat and curved surfaces. The algorithm is based on the principles of light and optics.
- If resolution of screen is $(x_1 * y_1)$ then there are $x_1 y_1$ pixels and so $x_1 y_1$ light rays are traced.



- A ray is fired from the center of projection through each pixel to which the window maps, to determine the closest object is intersected.

Algorithm

- For each pixel (x, y)
  - Trace a ray from eye point or viewpoint through a pixel (x, y) into scene.
  - Find the intersection point ‘I' with the surface, which is closest to the viewpoint.
  - Set pixel color to the color of surface on which this point 'I' lies.

Pros and Cons

- Suitable for complex curved surfaces.
- Computationally expensive.
- Can be easily combined with lightning algorithms to generate shadow and reflection.