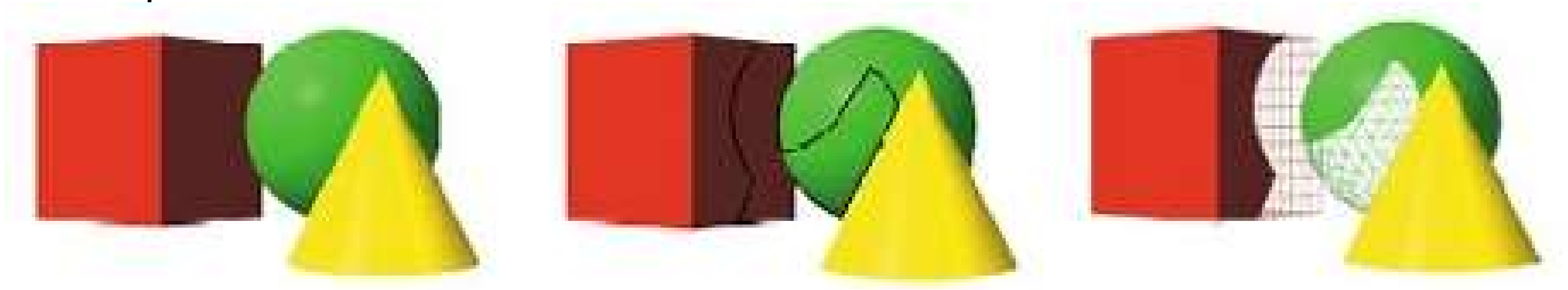
Visible Surface Detection Method

Visible Surface Detection Method

- The process of identifying those parts of a scene that are visible from a chosen viewing position.
- A procedure that removes any surfaces or lines which are not to be displayed in a 3D scene is called **visible surface detection method** or **hidden surface elimination method.**
- For a realistic graphics display, we much identify 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



Classification of Visible Surface Detection Algorithms

- Classified according to they deal with object definitions directly or with their projected images.
- Two approaches:

➢ object-space method

➢ image-space method

Object-space method

- Deals with object definition
- 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.
- Implemented in world-coordinates system and independent of display devices.
- Object space methods work on the objects themselves before they are converted to pixels in the frame buffer.
- Simple but inefficient (requires longer processing time).
- Line display algorithms use this method

<u>Basic Procedure</u>

```
For each object in the scene do
Begin
    – Determine those parts of the objects whose view is unobstructed by other parts
      of it or any other object with respect to the viewing specification.
    – Draw those parts in the object color.
End
```

Image space method

- Deals with projected image
- Visibility is decided point by point at each pixel position on the projection plane.
- Implemented on screen/device coordinate system.
- Most commonly used.
- Faster but requires larger memory.

Basic Procedure

```
For each pixel in the frame buffer
    – Determine the object closest to that pixel.
    – Draw the pixel in that object color
End
```

Algorithms of:

Object Space Method

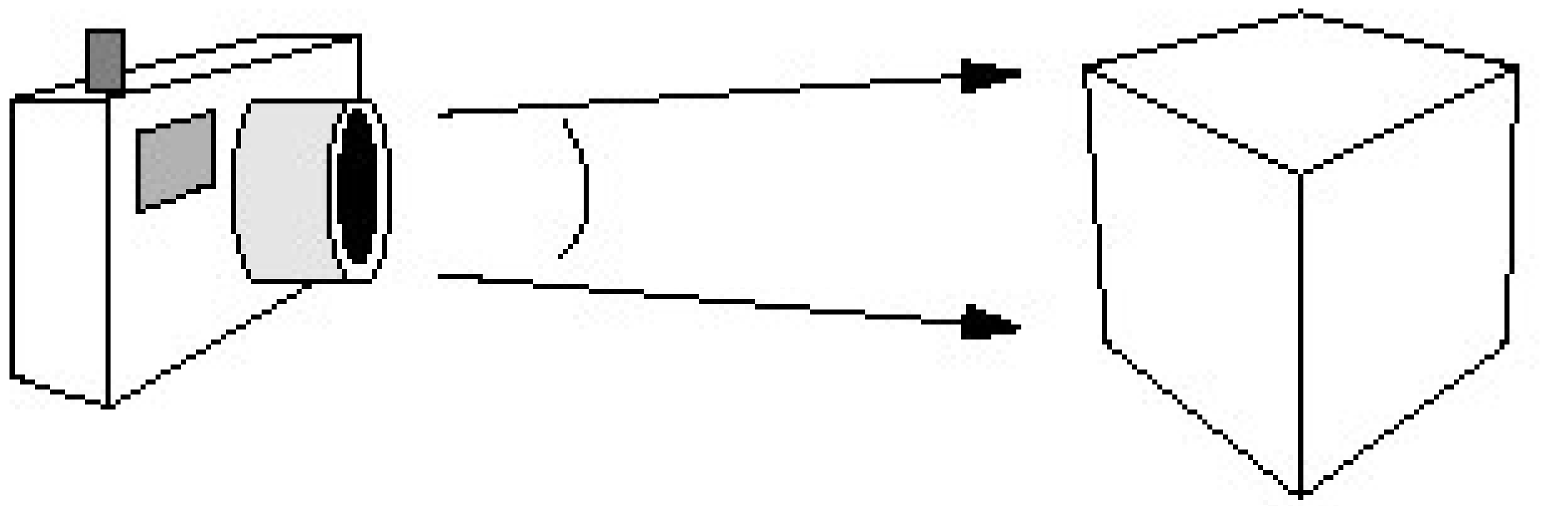
- Back Surface Detection
- Painter’s Algorithm
- BPS Tree

Image Space Method

- Depth (Z) buffer method
- Scan-line method
- Depth-sorting method
- Area-subdivision method
- Ray Casting Method

Back-Face Detection(Plane Equation Method)

- A fast and simple object-space method for identifying the back faces of polyhedron based on the “inside-outside” tests.
- In this approach no faces on the back of the object are displayed.

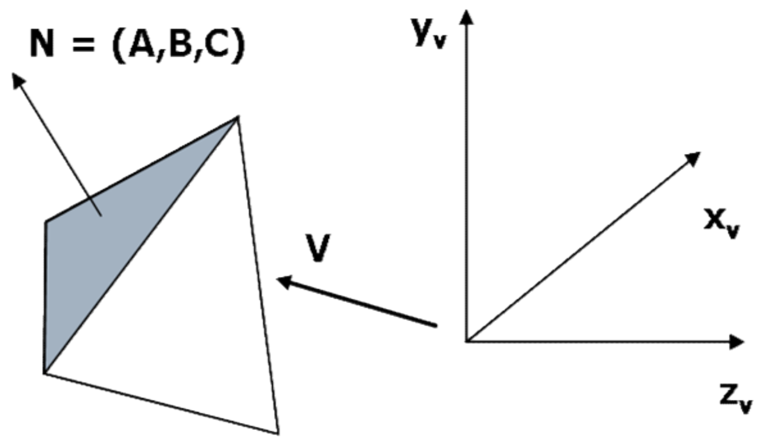


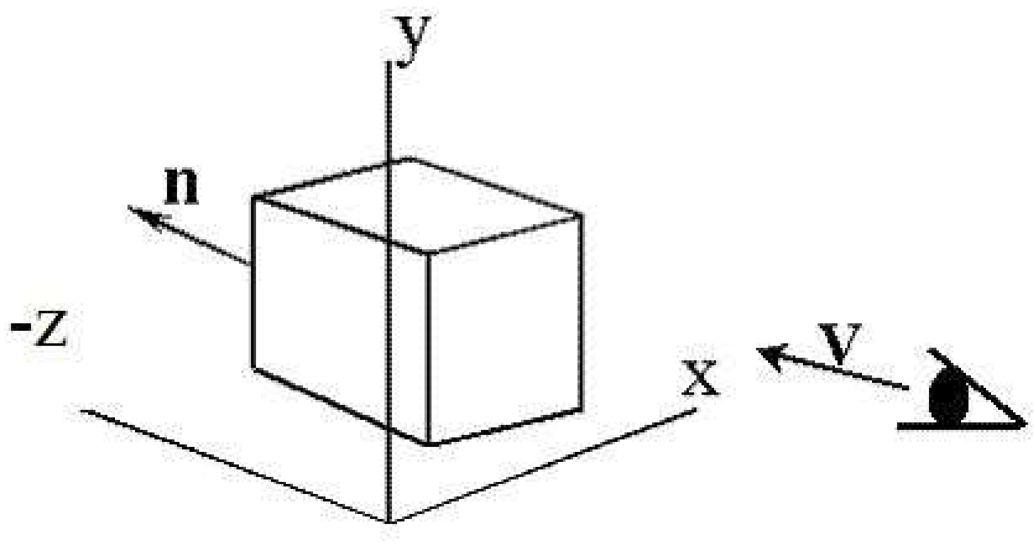
Basic Procedure

- Test whether view point (x, y, z) is inside or outside the polygon surface
  - Tested by the inequality: $Ax + By + Cz + D = 0$
    - If view point is outside the surface → visible → $Ax + By + Cz + D > 0$
    - If inside → back face → $Ax + By + Cz + D < 0$

- The test could be simplified by considering normal vector **N** = (A,B,C) to polygon surface and Vector **V** in viewing direction from the eye(or camera) . Then,

  the polygon is back face if: **V.N**>0





If $\mathbf{n \bullet V > 0}$ then polygon is backface

$\mathbf{n \bullet V = |n|\ |V| \cos \theta}$

$\mathbf{\cos \theta = n \bullet V / (|n|\ |V|)}$

where **|n|** and **|V|** are the magnitudes of the vector

$\mathbf{|V| = sqrt(V_{\mathit{x}}^2 + V_{\mathit{y}}^2 + V_{\mathit{z}}^2)}$

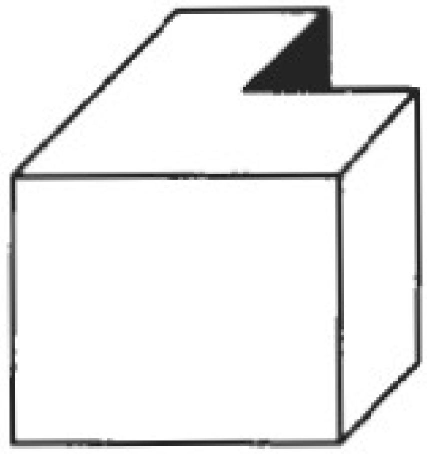
- If the object description have been converted to projection coordinates and our viewing direction is parallel to the viewing Zv axis, then v = (0, 0, Vz) and

$$\mathbf{V.N} = Vz.C$$

- So we need to consider the sign of C and Z component of normal vector.
- In a right-handed viewing system with viewing direction along the negative Zv axis, the polygon is back face
  - If C<0.
  - If C=0, since it is a grazing Viewing i.e. viewing direction is perpendicular to surface normal.
- Thus we label any polygon as back face if its normal vector has z component value:

$$\mathbf{C} \leq 0$$

- For left handed system, when the viewing direction is along the positive z-axis, we label the back face for polygon if $C \geq 0$.



Advantages

- Simple and easy implement
- No pre-sorting surfaces is needed

Limitations

- Used only for solid objects modeled as a polygon mesh.
- Problematic for concave polyhedron.

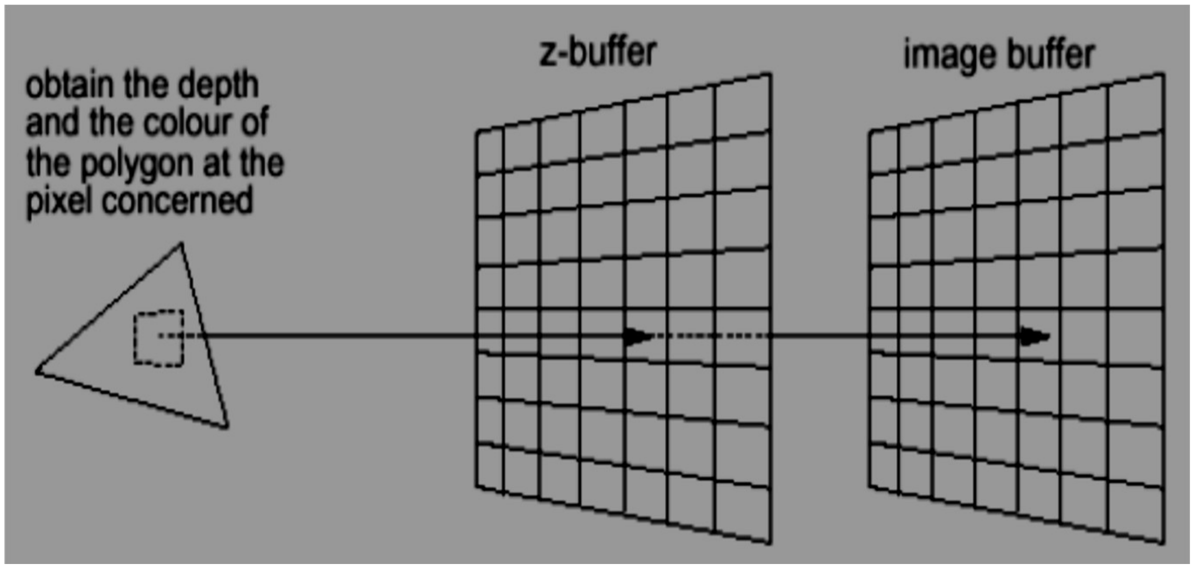
e.g. partially hidden face will not be eliminated by Back-face removal.

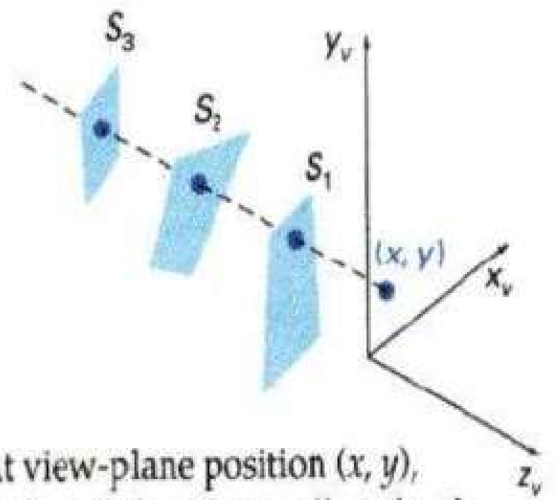
Depth Buffer (z-buffer method)

- Depth Buffer Method is the commonly used **image-space method** for detecting visible surface.
- It *compares surface depths at each pixel position on the projection plane*.
- It is called **z-buffer method** *since object depth is usually measured from the view plane along the z-axis of a viewing system*.

- With object description converted to projection co-ordinates, each (x, y, z) position on polygon surface corresponds to the orthographic projection point (x, y) on the view plane. Therefore for each pixel position on the view plane, object depth is compared by z values.
- Each surface of 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 and method is easy to implement

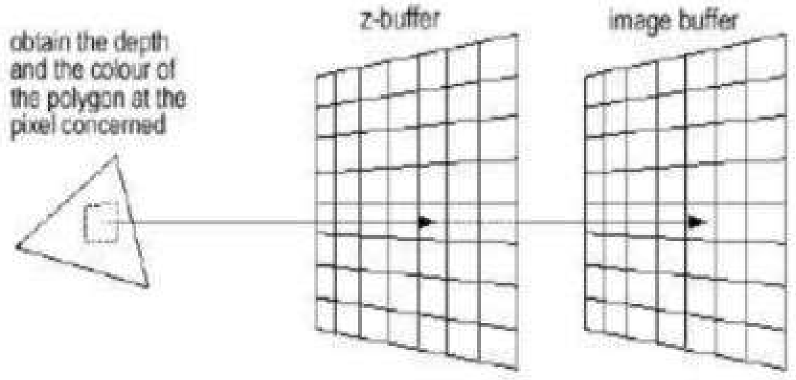
- Two buffers are required
  - Depth buffer: to store depth value for each position i.e. z-values for reach pixel position (x, y)
  - Refresh buffer: to store intensity values or color values of each pixel position
- Drawback: Deals only with opaque surface but not with transparent surface

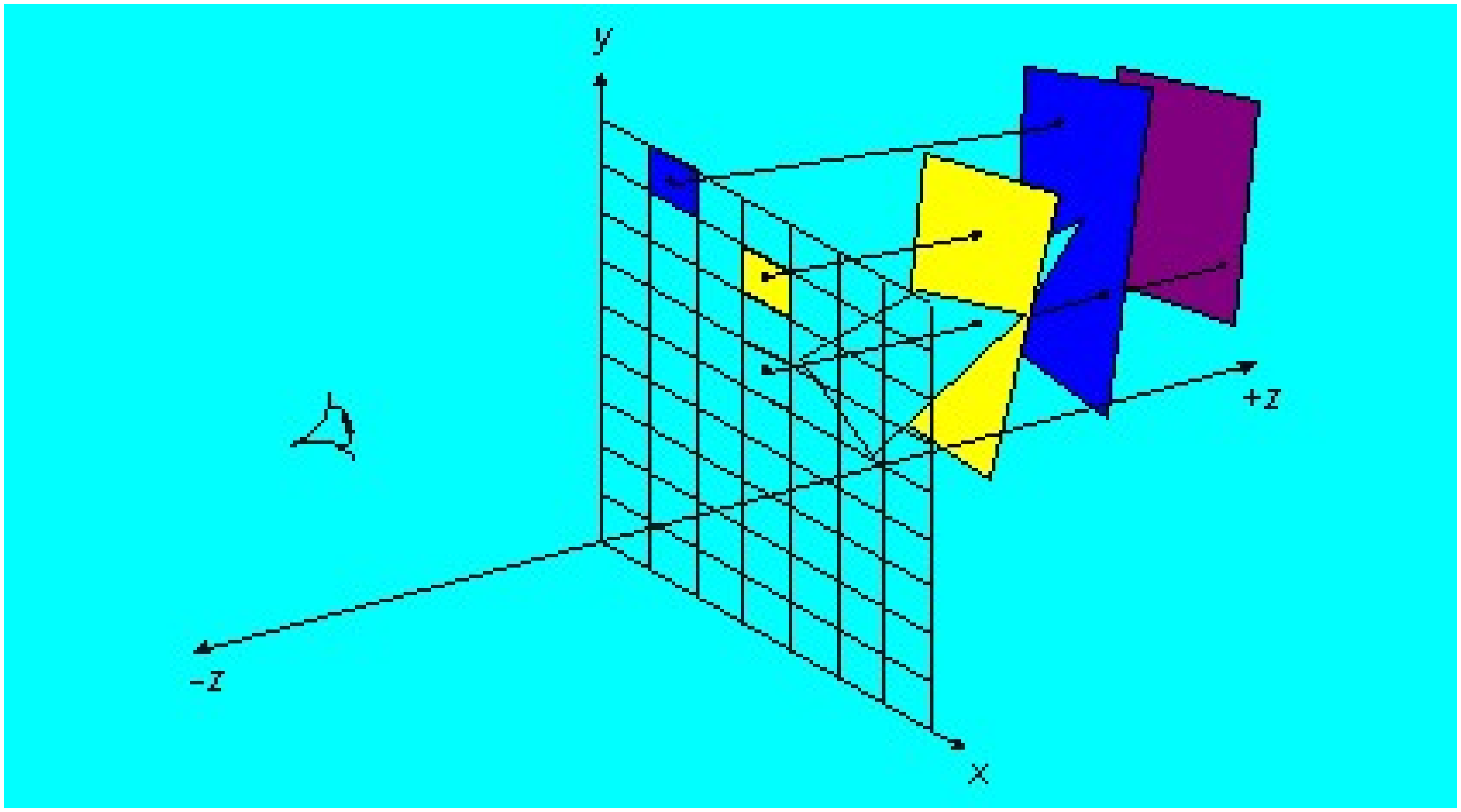

obtain the depth and the colour of the polygon at the pixel concerned
z-buffer
image buffer



At view-plane position (x, y), surface S_1 has the smallest depth from the view plane and so is visible at that position.

- Three surfaces at varying distance from view plane x_vy_v, the projection along (x, y) surface S_1 is closest to the view-plane so surface intensity value of S_1 at (x, y) is saved.


obtain the depth and the colour of the polygon at the pixel concerned
z-buffer
image buffer


y
+z
-z
x

Algorithm

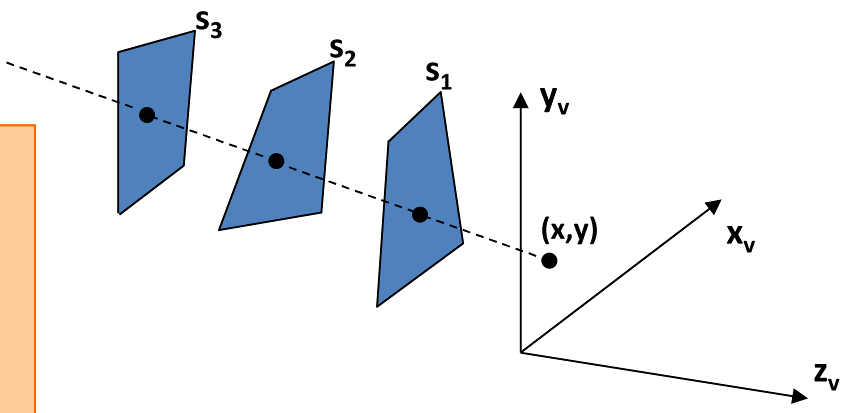
1. Initialize the depth buffer and refresh buffer so that for all buffer positions (x, y),

   depth(x, y) = 0, refresh(x, y) = $I_{background}$

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 z for each (x, y) position on the polygon
   - If $z<depth(x, y)$, then set

     depth(x, y)=z, refresh(x, y)=$I_{surface}(x, y)$

$I_{background}$ = background intensity
$I_{surface}(x,y)$ = projected intensity value for the surface at pixel position (x, y)



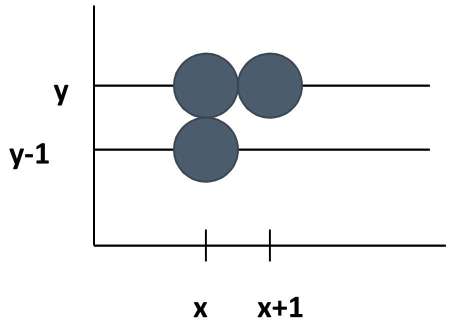
Depth Calculation

- From plane equation, depth is

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

- For the next adjacent pixel in a scan line, depth is

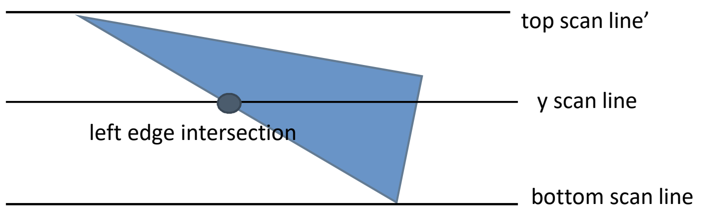
$$z' = \frac{-A(x+1) - By - D}{C} \quad \text{or} \quad z' = z - \frac{A}{C}$$



- Determine minimum and maximum y-coordinates for each polygon
- Start from top scan line to bottom scan line
- Starting from top vertex, calculate x position down the left edge of the polygon recursively as

$x' = x - 1/m$

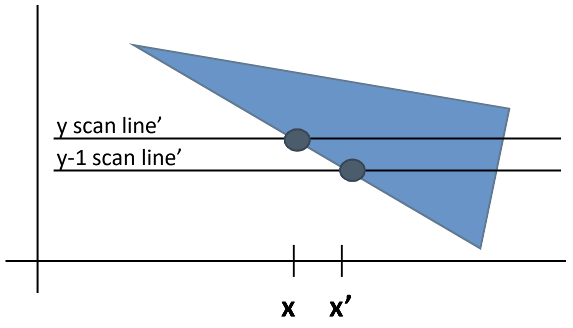
$m(x'-x)=y'-y=-1$



- Thus depth value for the next pixel in left edge is obtained as

$$z' = z + \frac{\frac{A}{m} + B}{C}$$

put x' = x-1/m and y' = y-1



- For vertical edge m = infinity, so:

$$z' = z + \frac{B}{C}$$

Pons and cons

- Simple and easy to implement, no specific hardware is needed.
- No pre-sorting of polygons is needed.
- No object-object comparison is required. Can be applied to non-polygon objects.
- Good for animations.
- Additional memory buffer i.e. z-buffer is required.
- It can only find one visible surface at each pixel position i.e. it deals only with opaque surfaces and cannot accumulate intensity values for more than one surface, as it is necessary if transparent surfaces are to be displayed.

A-Buffer Method

- A –buffer method is an ***extension of idea of z-buffer.*** It represents *anti-aliased, area-averaged, accumulation*-buffer method.

- The depth buffer is expanded so that each position in the buffer can reference a linked list of surfaces thus enabling more than one surface intensity consideration, and object edge can be anti-aliased.

- It maintains a data structure of background surfaces that are behind the foreground transparent surface. This special data structure is called accumulation buffer.

- To accommodate the depth information and link information, A buffer has two field:

1. Depth field: stores a positive or negative real number
   - Positive→ single surface contributes to pixel intensity i.e. surface is opaque
   - Negative → multiple surface contribute to pixel intensity.

2. Intensity field: store surface intensity information or a pointer value
   - Surface intensity if single surface → stores the RGB components of the surface color at that point and percent of pixel coverage
   - Pointer value if multiple surfaces

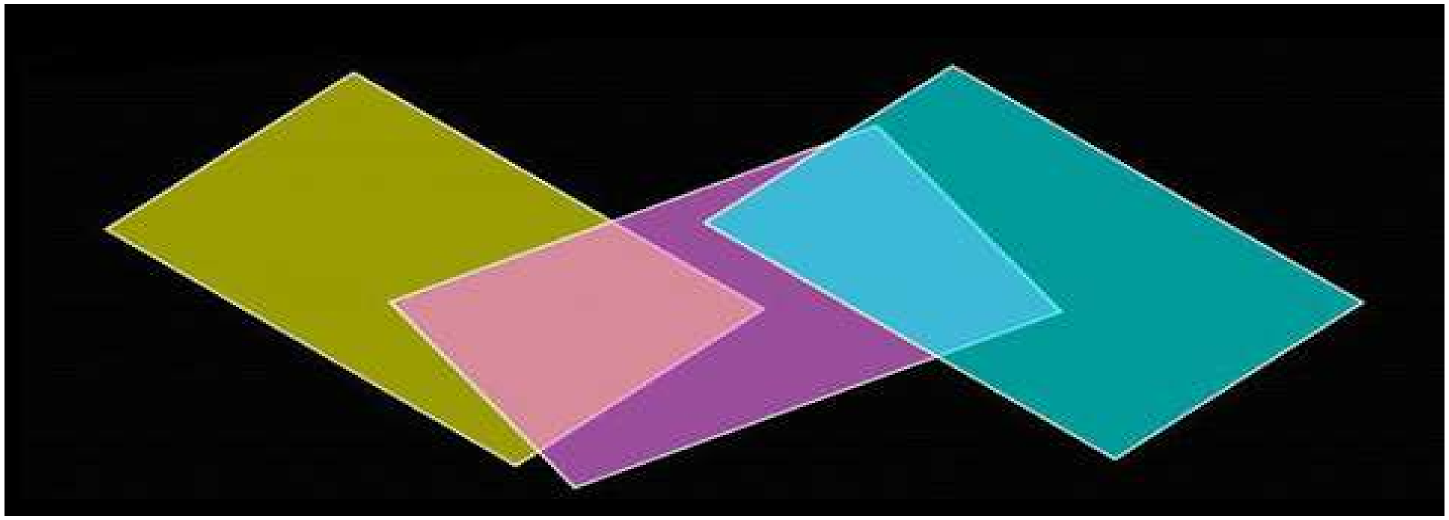


Figure: Viewing two background opaque surfaces through a foreground transparent surface requires multiple surface-intensity contributions for pixel positions

- In case of surface linked list, the data for each surface in the linked list includes:
  - RGB intensity components
  - Opacity parameter (percentage of transparency)
  - Depth
  - Percent of area coverage
  - Surface identifier
  - Other surface rendering parameters
  - Pointer to next surface

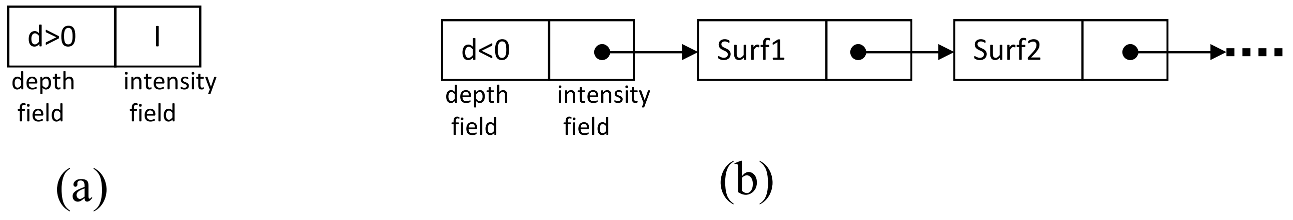


Figure: Organization of an A-buffer position: (a) single-surface overlap of the corresponding pixel area; (b) multiple-surface overlap

- Scanlines are processed to determine surface overlaps of pixels across the individual scanlines.
- Surfaces are subdivided into a polygon mesh and clipped against the pixel boundaries
- The opacity factors and percent of surface overlaps are used to determine the pixel intensity as an average of the contribution from the overlapping surfaces

Scan Line Method

- Image-space method for identifying visible surfaces. It deals with multiple polygon surfaces.
- The scan line method solves the hidden surface problem one scan line at a time. Processing of the scan line start from the top to the bottom of the display.
- This method calculates the depth values for only the overlapping surfaces which are tested by the scan line.
- 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 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.
- It requires an edge table, polygon table, active edge list and flag.

- **Edge table contains:** Coordinate end points for each scan line, pointers into the polygon table to identify the surfaces bounded by each line.

- **Polygon table contains**: Coefficients of plane equations for each surfaces, pointers into the edge table, and intensity information of the surfaces.

- **Active edge list contains**: Edges that cross the current scan line, shorted in order of increasing x.

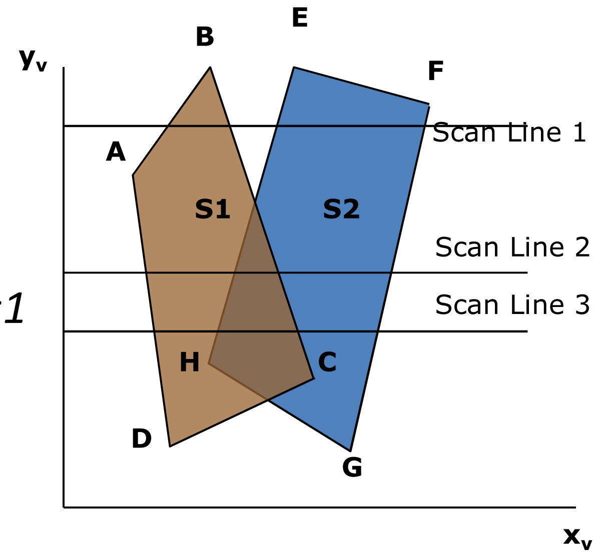
- **Flag** is defined for each surface that is set ‘ON’ or ‘OFF’ to indicate whether a scan line is inside or outside of the surface. At the left most boundary surface flag is ‘ON’ and at right most boundary flag is ‘OFF’.

- To facilitate the search for surfaces crossing a given scan-line an active list of edges is formed for each scan-line as it is processed
- The active list stores only those edges that cross the scan-line in order of increasing x.
- Also a flag is set for each surface to indicate whether a position along a scan-line is either inside or outside the surface.

Procedure

- Pixel positions across each scan-line are processed from left to right
- At the left intersection with a surface the surface flag is turned on
- At the right intersection point the flag is turned off
- We only need to perform depth calculations when more than one surface has its flag turned on at a certain scan-line position

- For scan line 1
  - The active edge list contains edges AB,BC,EH, FG
  - Between edges AB and BC, only *flags for s1 == on* and between edges EH and FG, only *flags for s2==on*
    - no depth calculation needed and corresponding surface intensities are entered in refresh buffer
- For scan line 2
  - The active edge list contains edges AD,EH,BC and FG
  - Between edges AD and EH, only the *flag for surface s1 == on*
  - Between edges EH and BC *flags for both surfaces == on*
    - Depth calculation (using plane coefficients) is needed.
    - In this example depth for s1 is less than for s2, so intensities for surface s1 are loaded into the refresh buffer until boundary BC is encountered
  - Between edges BC and FG *flag for s1 == off* and *flag for s2 == on*
    - Intensities for s2 are loaded on refresh buffer
- For scan line 3
  - Same **coherent** property as scan line 2 as noticed from active list, so no depth calculation needed between edges BC and EH



- **Basic Procedure:**
  - Sort all surfaces of polygon in order of decreasing depth (z-coordinate).
  - 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 final 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.

Algorithm

1. Establish and initialize data structure.
    i. Edge table with line endpoints.
    ii. Polygon table with surface information and pointer to the edge table.
    iii. Initially empty active edge list. i.e., AEL = { }.
    iv. A flag, initially flag is "off" for each surface.
2. Repeat for each scan line.
    a. Update active edge list AEL.
3. For each pixel (x, y) on scan line
    i. Update flag for each surface.
    ii. When flag is ON for only one surface, enter intensity of that surface to refresh buffer.
    iii. When two or more flags are ON, calculate depth and store the intensity of the surface which is nearest to the view plane.

Pros and Cons:

- 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.

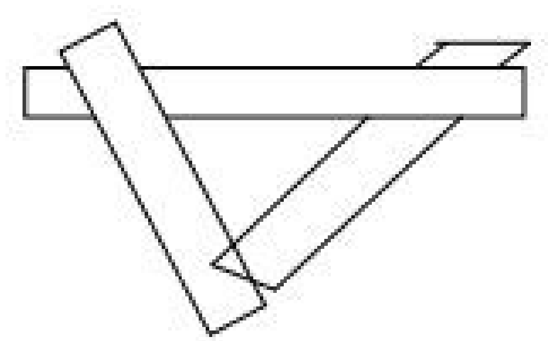
Depth-Sorting Method /Painter’s Algorithm/Priority Algorithms

- Another widely used object space method.
- This method for solving the hidden-surface problem is often referred to as the ***Painter's algorithm or priority fill algorithm.***
- 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.
- This method requires sorting operation of surfaces in both the image and object space.

- This algorithm is very analogous to the oil painting.
- Here, the surfaces are sort first according to the their distance from view plane. Then, the intensity values for the farthest surface are 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.

Problems

- One of the major problem in this algorithm is intersection polygon surfaces. As shown in fig:



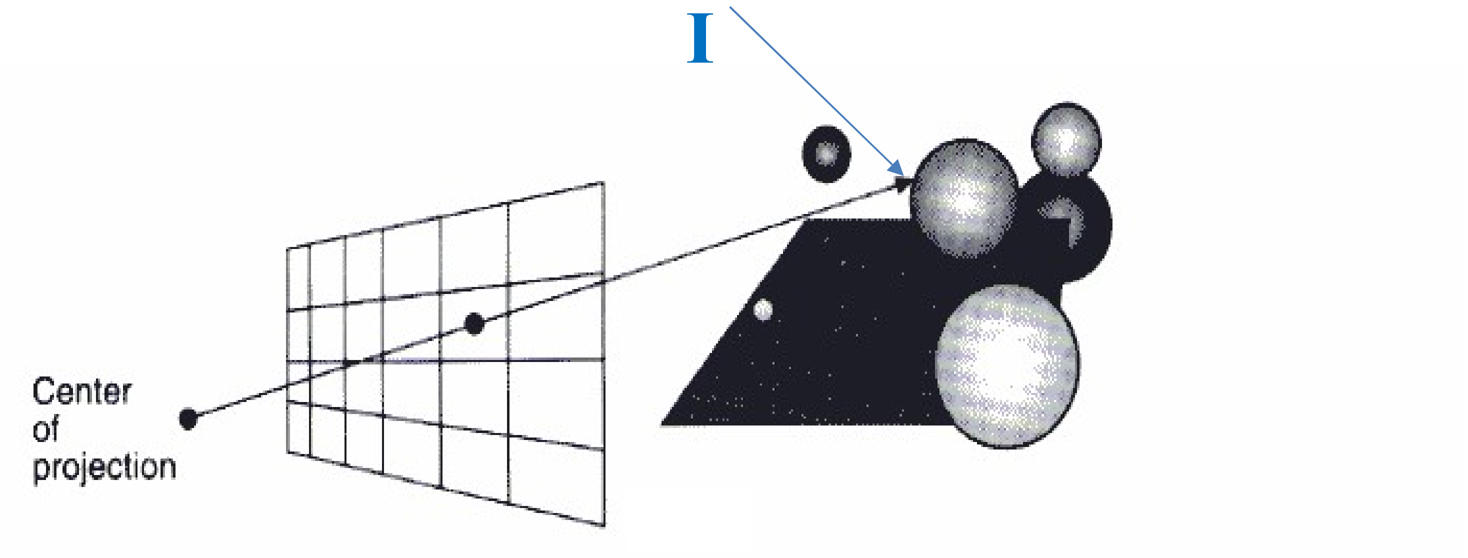
  - Different polygon may have same depth.
  - The nearest polygon could also be farthest

Solution

- For intersection polygons, we can split one polygon into two or more polygons which can then be painted from back to front. This need more times to compute intersection between polygons. So it becomes complex algorithm for such surface existence.

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 $(m * n)$ then there are mn pixels and so mn 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 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.

Area Subdivision Algorithm/Warnock's Algorithm

- Widely used image space visible surface method.
- It uses divide-and-conquer strategy of partitioning in the projection plane or view plane, developed by Warnock.
- The algorithm has two steps:

Step (1) : An area of the projected image is examined, if all polygons are visible in area, they are displayed.

Step (2): Otherwise, the area is subdivided into four equal areas, on each those polygons are further tested to determine which one should therefor be displayed. If a visibility decision cannot be made then this second area is further subdivided either until a visibility decision can be made or until the screen area is single pixel. This method is also known as Quadtree Method.

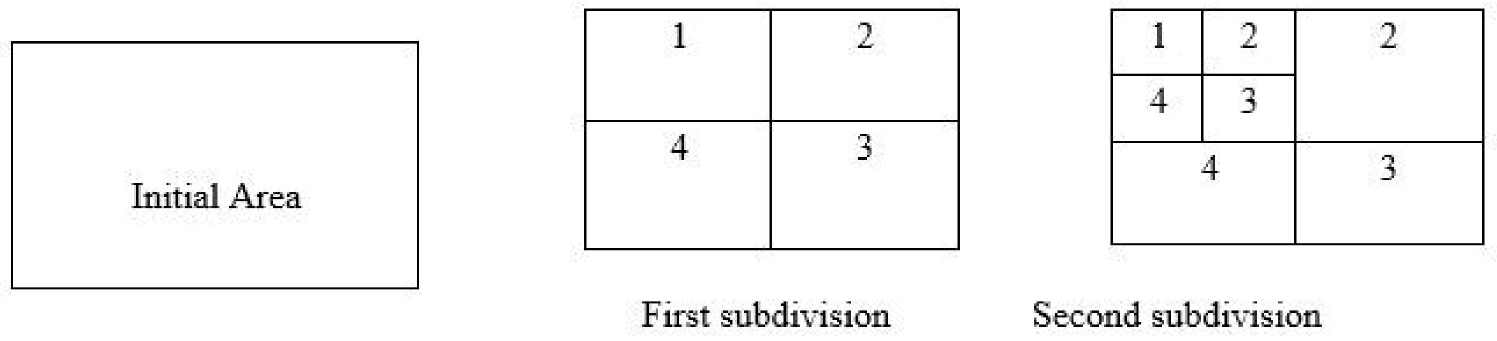
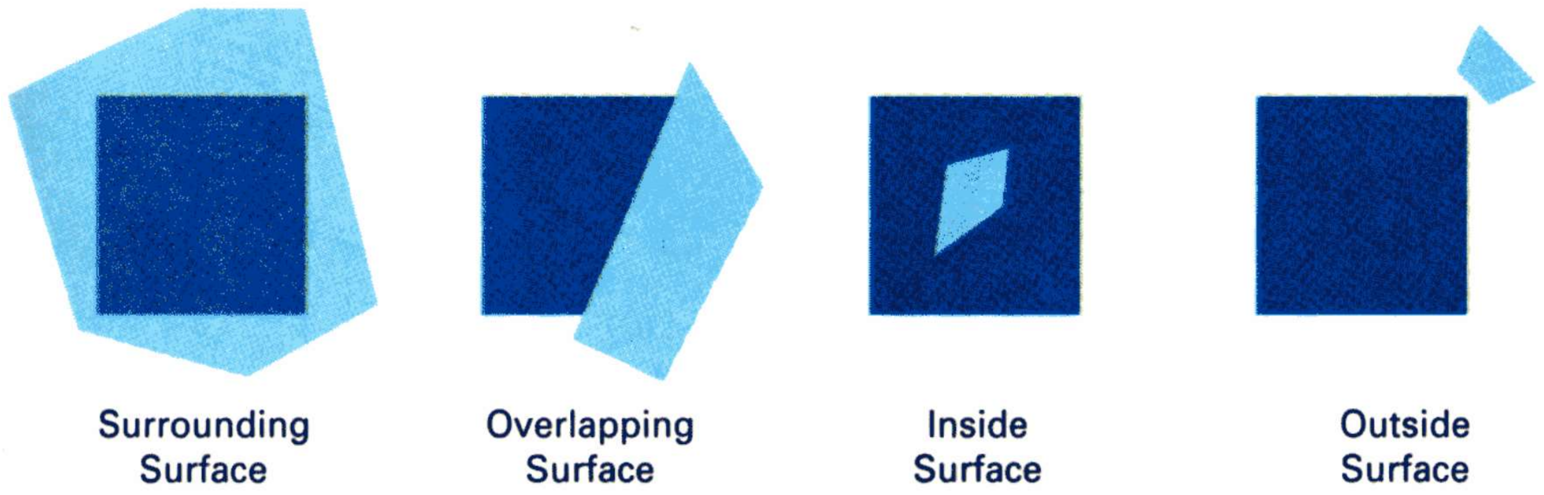


Fig: Area subdivision method

- At each stage in the recursive subdivision process, each polygon surface has one of four relationships to the area of interest (AOI) or the specified area boundary or viewport.
- **Surrounding Surface**
  - A polygon surface surrounds a viewport if it completely encloses or covers the viewport.
- **Intersecting(or Overlapping) Surface**
  - A polygon surface overlaps or intersects the viewport if any of its side cuts the edges of viewport.
- **Contained Surface(Inside Surface)**
  - A polygon is contained in a viewport if no part of it is outside of the edges of the viewport.
- **Disjoint Surface or Outside Surface**
  - A polygon is disjoint from the viewport if the polygon does not intersect the viewport anywhere.



Possible relationships between polygon surfaces and a rectangular area.

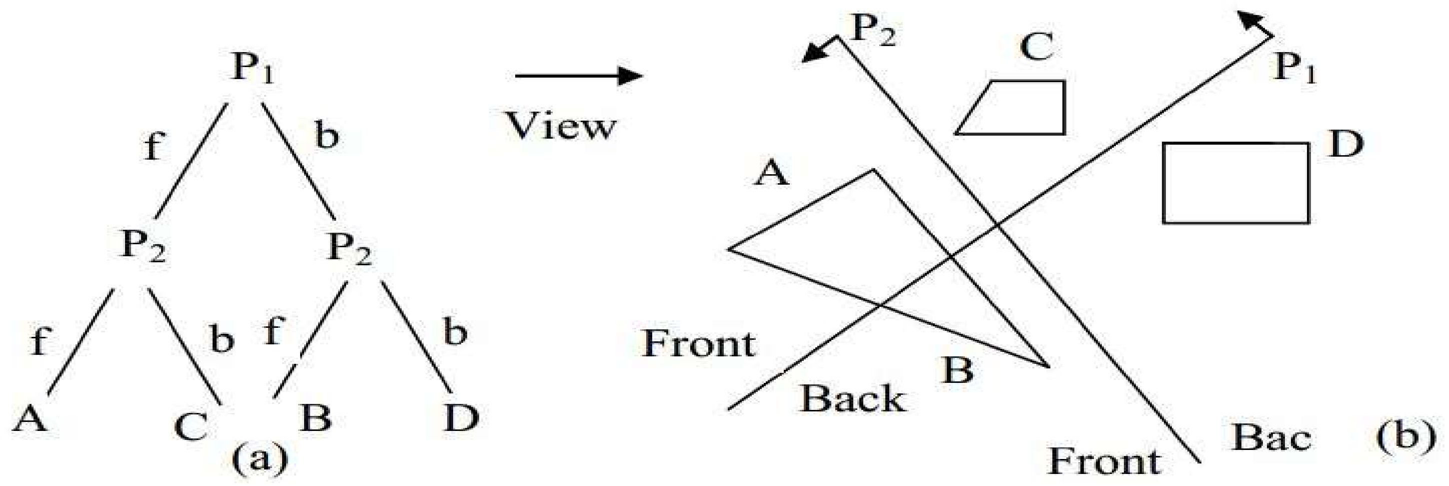
Algorithm:

- For a given area
  - If polygons are disjoint then the background color fills the area i.e., refresh buffer contains background color.
  - If there is only one intersecting or only one contained polygon then the area is first filled with the background color, and then the part of the polygon contained in the area is filled with the color of that polygon.
  - If there is a single surrounding polygon and no intersecting or contained polygons, then the area is filled with the color of the surrounding polygon.
  - If there is a single surrounding polygon in front of surrounding, intersecting or contained polygons, then the area is filled with the color of the front surrounding polygon.
  - Otherwise, break the area into four equal areas and repeat.

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.
- It was created by **Fuchs.**
- 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.*

- Applying a BSP tree to visibility testing involves identifying surfaces that are “inside” or “outside” the partitioning plane at each step of space subdivision relative to viewing direction.
- It is useful and efficient for calculating visibility among a static group of 3D polygons as seen from an arbitrary viewpoint.
- It is a two step process:
  1. Construction of BSP tree.
  2. Display of the tree.



- Here plane P1 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 P1, we divide that object into two objects labeled A and B. Now object A & C are in front of P1, B and D are back of P1.
- We next partition the space with plane P2 and construct the binary tree as fig (a). In this tree, the objects are represented as terminal nodes, with front objects as left branches and behind objects as right branches.
- When BPS tree is complete, we process the tree by selecting surface for displaying in order back to front. So fore ground object are painted over back ground objects.

Pros and cons

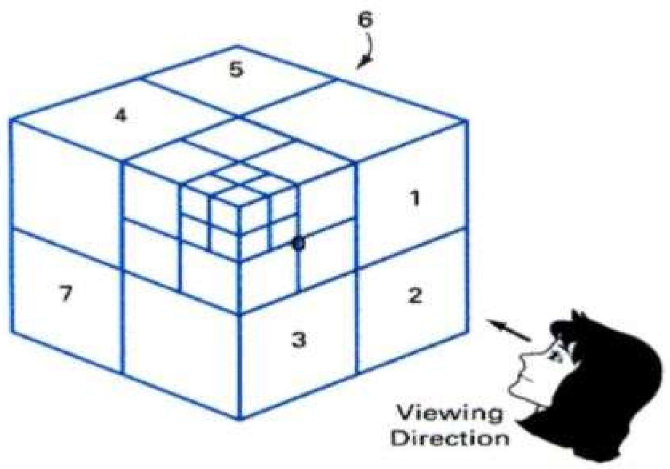
- **The main advantage** of BPS tree is that we are not dealing with z-coordinates at all to decide the priority of the polygons for display.
- **The main disadvantages** of BSP tree is the waste of computational time as we are performing calculations for those objects also, which are hidden.

Summarize of BPS

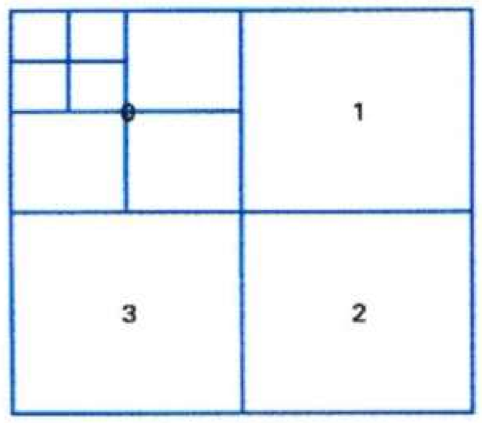
1. Select a polygon as the root of the tree i.e. the first partition plane.
2. Partition the object space into two half spaces (inside and outside of the partition plane determined by the normal to the plane); some object polygon lie in the rear half while the other in the front half with respect to the partition plane.
3. If a polygon is intersected by the partition plane, split it into two polygon so that they belong to different half-spaces.
4. Select one polygon on the root's front as the left node or child and another in the root's back as the right node.
5. Recursively sub-divide the spaces considering the plane of the children's as partition planes until a sub-space contain a single polygon.

Octree Method

- When an octree representation is used for viewing volume, hidden surface elimination is accomplished by projecting octree nodes into viewing surface in a front to back order, as shown in figure below.
- If the figure below, the front face of a region of space (the side towards the viewer) is formed with octants 0,1,2, and 3. surfaces in the front of these octants are visible to the viewer.
- Any surfaces towards the rear of the front octants or in the back octants 4,5,6, and 7 are not visible.
- After octants sub-division and construction of octree, entire region is traversed by depth first traversal.



Octants in Space



Quadrants for the View Plane