Visible Surface Determination

Unit 7

Visible Surface Detection (Hidden Surface Removal) Method

It 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.** Sometimes these methods are also referred to as **hidden-surface elimination methods.**

- To identify those parts of a scene that are visible from a chosen viewing position (**visible-surface detection methods**).
- Surfaces which are obscured by other opaque (solid) surfaces along the line of sight are invisible to the viewer so can be eliminated (**hidden-surface elimination methods**).

Visible Surface Determination..

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

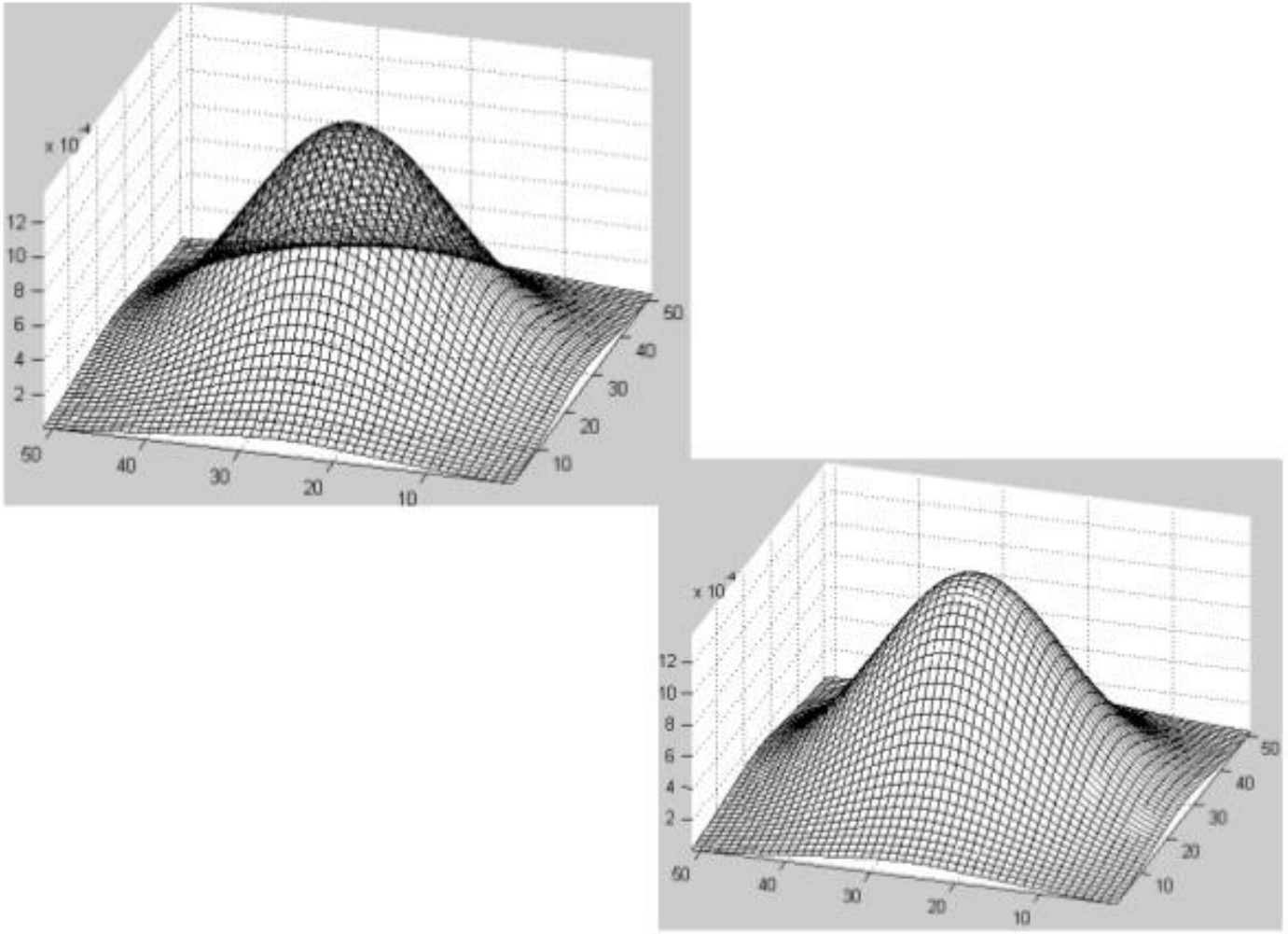
These two approaches are

- ***Object-Space methods(OSM):***
  - Deal with object definition
  - Compares objects and parts of objects to each other within the scene definition to determine which surface as a whole we should label as visible.
  - E.g. *Back-face detection method*
- ***Image-Space methods(ISM):***
  - Deal with projected image
  - Visibility is decided point by point at each pixel position on the projection plane.
  - E.g. *Depth-buffer method, Scan-line method, Area-subdivision method*
- Most visible surface detection algorithm use image-space-method but in some cases object space methods are also used for it.

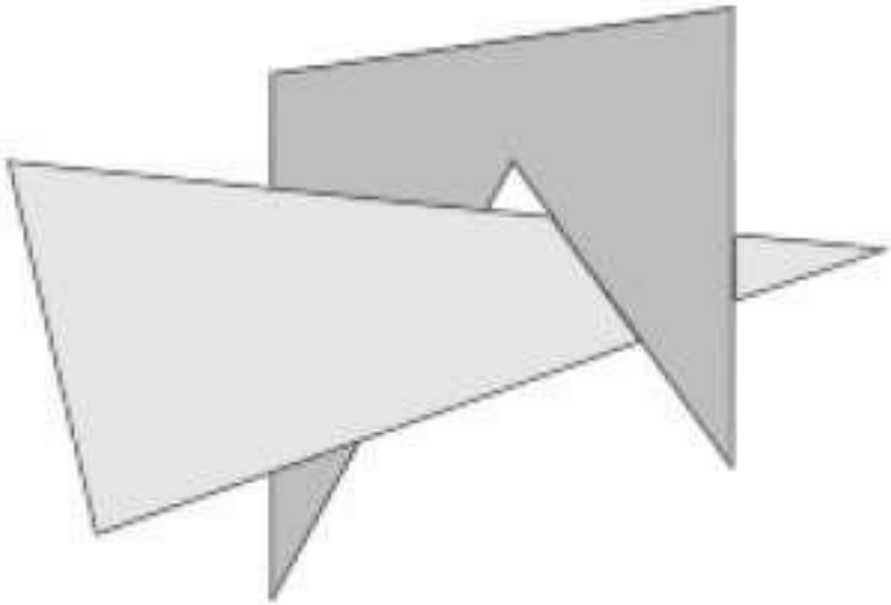
Visible Surface Determination..

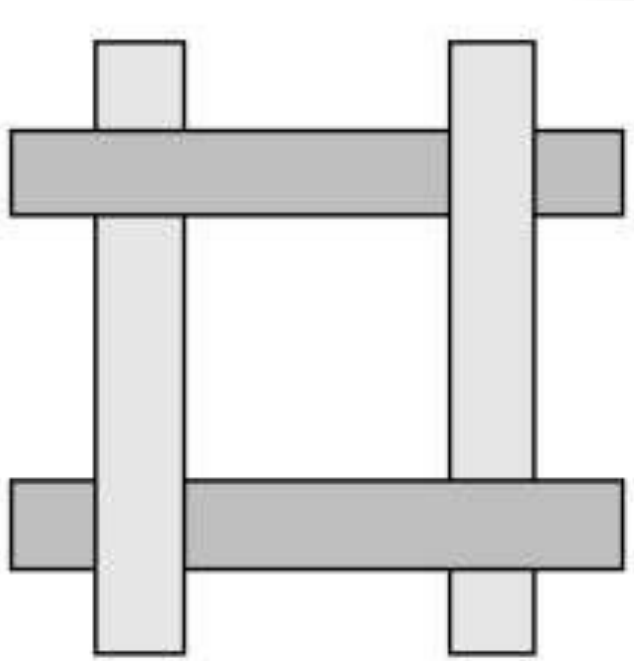
- **List Priority Algorithms**
  - This is a hybrid model that combines both object and image precision operations. Here, depth comparison & object splitting are done with object precision and scan conversion (which relies on ability of graphics device to overwrite pixels of previously drawn objects) is done with image precision.
  - E.g. *Depth-Shorting method, BSP-tree method*

Visible Surface Determination



Visible Surface Determination





Visible Surface Determination

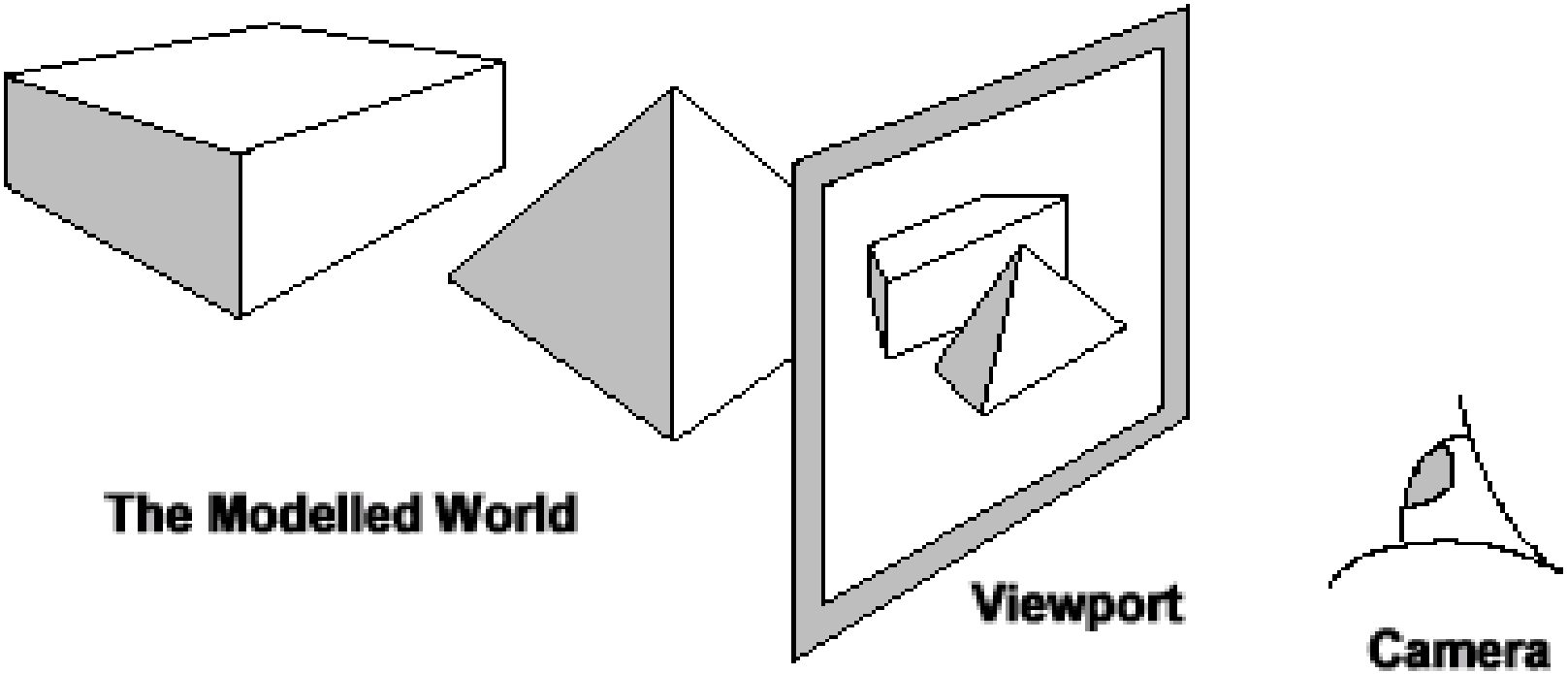


Figure 1. The Basics of 3D Graphical Processing

Object-Space methods

- Algorithms to determine which parts of the shapes are to be rendered in 3D coordinates.
- Methods based on comparison of objects for their 3D positions and dimensions with respect to a viewing position.
- For *N* objects, may require *N***N* comparision operations.
- Efficient for small number of objects but difficult to implement.
- Depth sorting, area subdivision 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.
- Compare each object with all other objects to determine the visibility of the object parts.

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.
- Accuracy of the calculation is bounded by the display resolution.
- A change of display resolution requires re-calculation

Image Space Methods

- Based on the pixels to be drawn on 2D. Try to determine which object should contribute to that pixel.
- Running time complexity is the number of pixels times number of objects.
- Space complexity is two times the number of pixels:
  - One array of pixels for the frame buffer
  - One array of pixels for the depth buffer
- Coherence properties of surfaces can be used.
- Depth-buffer and ray casting methods.

Back – Face Detection Method

- A fast and simple **object-space method** for identifying the back faces of a polyhedron.
- It is based on the performing inside-outside test.

TWO METHODS:

First Method:

- A point *(x,* y, z) is "inside" a polygon surface with plane parameters A, B, C, and D if Ax+By+Cz+D < 0 (from plane equation).
- When an inside point is along the line of sight to the surface, the polygon must be a back face.
- In eq. Ax+By+Cz+D=0

if A,B,C remain constant , then varying value of D result in a whole family of parallel plane

if D>0, plane is behind the origin (Away from observer)

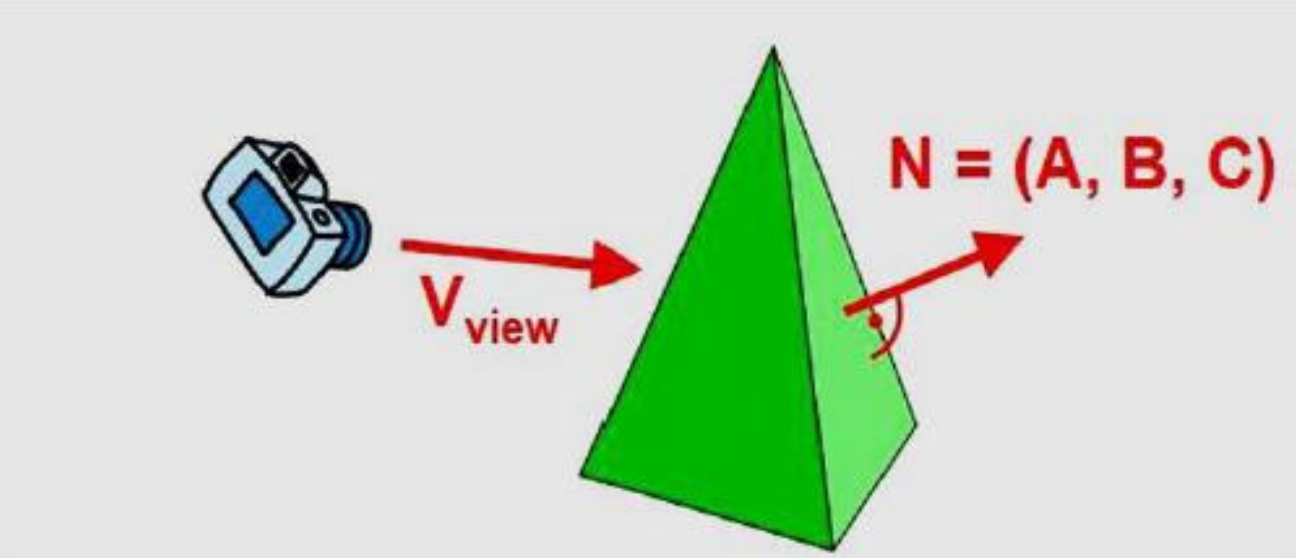
if D<0 , plane is in front of origin (toward the observer)

Back – Face Detection Method

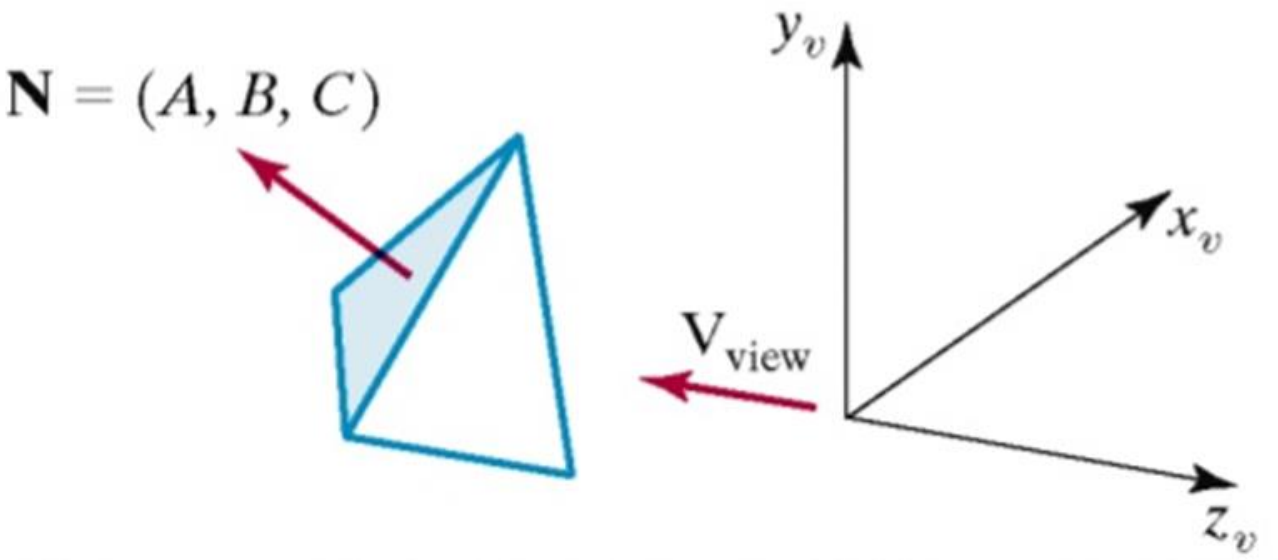
Second Way

- Let N be normal vector to a polygon surface, which has Cartesian components (A, B, C). In general, if V is a vector in the viewing direction from the eye (or "camera") position, then this polygon is a back face

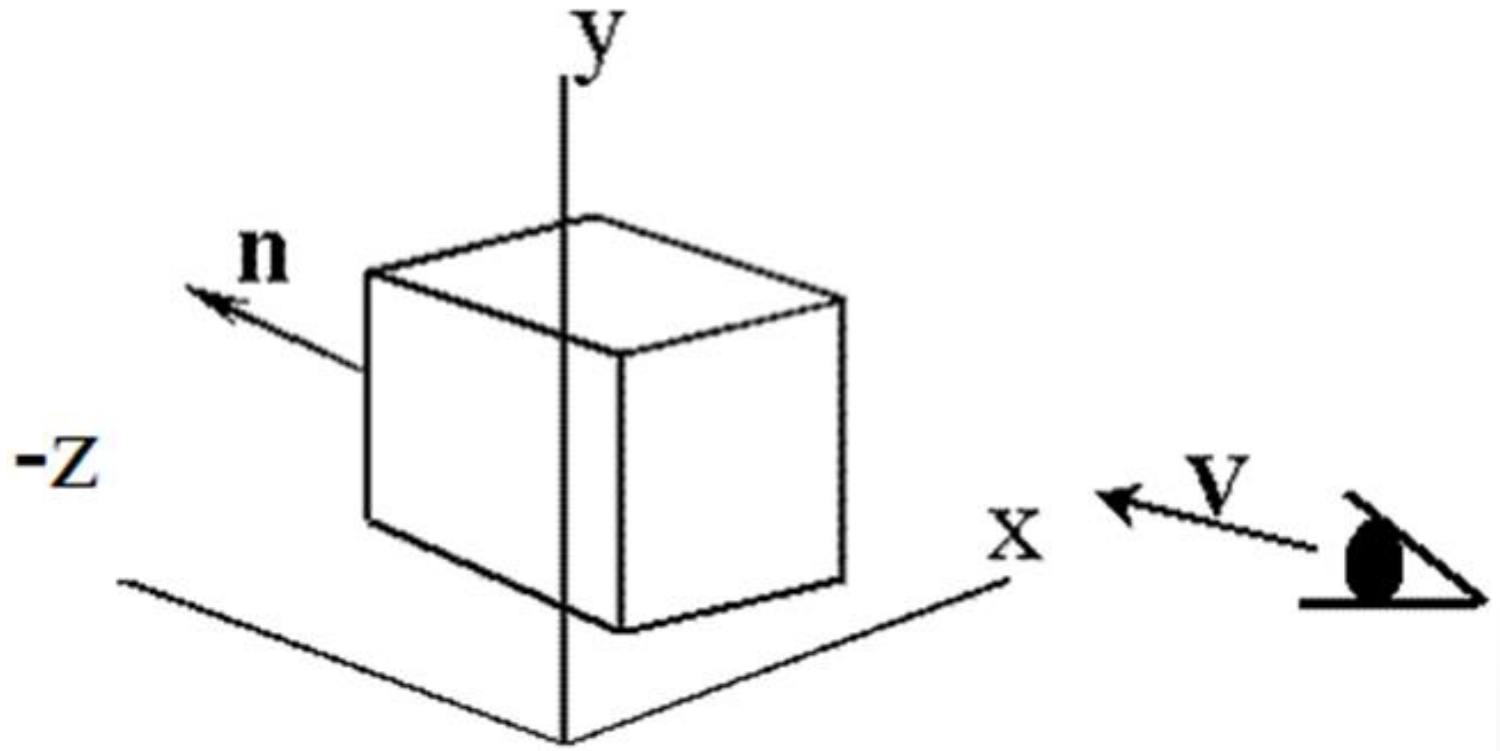
if **V.N>0.**



Back – Face Detection Method



Back – Face Detection Method

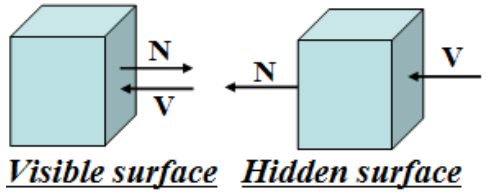


Back – Face Detection Method

A view vector V is constructed from any point on the surface to the viewpoint, the dot product of this vector and the normal N, indicates visible faces as follows:

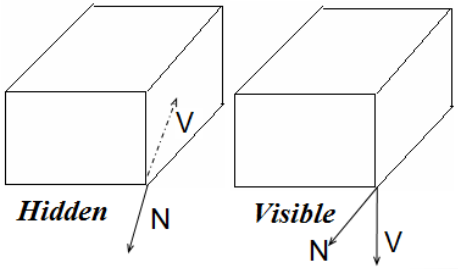
Case-I: (FRONT FACE)

If **V.N < 0** the face is visible ***else*** face is hidden



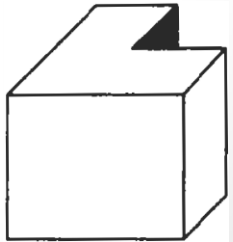
Case-II: (BACK FACE)

If **V.N > 0** the face is visible ***else*** face is hidden



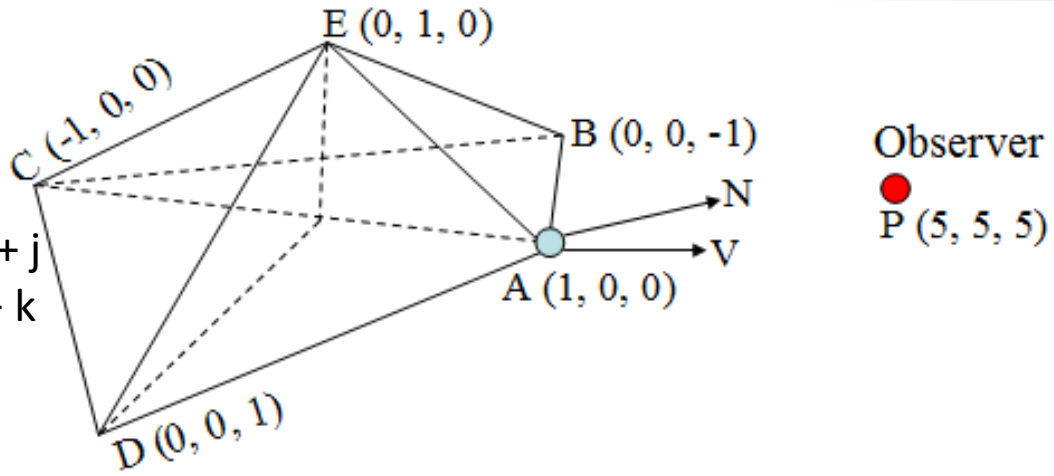
Case-III:

For other objects, such as the concave polyhedron in Fig., more tests need to be carried out to determine whether there are additional faces that are totally or partly obscured by other faces.



Numerical

Find the visibility for the surface AED in rectangular pyramid where an observer is at P (5, 5, 5).



Solution

Here,

AE = (0-1)i + (1-0)j + (0-0)k = -i + j

AD= (0-1)i + (1-0)j + (1-0)k =-i + k

Step-1:

Normal vector N for AED

Thus, $N = AE \times AD = \begin{pmatrix} i & j & k \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} = i(1-0) - j(-1+0) + k(0+1) = i + j + k$

Case-II

Step-2: If observer at P(5, 5, 5) so we can construct the view vector V from surface to view point A(1, 0, 0) as:

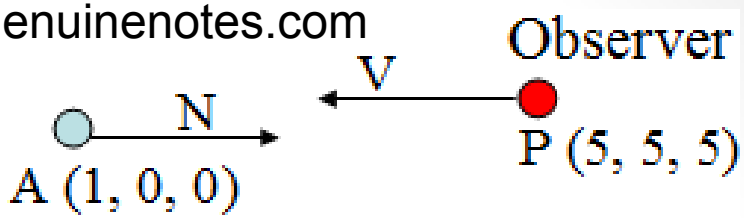
V = AP = (5-1)i + (5-0)j + (5-0)k = 4i + 5j + 5k

Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N as:

V.N = (4i + 5j + 5k).(i + j + k) = 4+5+5 = 14> 0

This shows that the surface is visible for the observer.

Case-I



Step-2: If observer at P(5, 5, 5) so we can construct the view vector

V from surface to view point A(1, 0, 0) as:

V = PA = (1-5)i + (0-5)j + (0-5)k = -4i - 5j - 5k

Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N

as:

V.N = (-4i - 5j - 5k).(i + j + k) = -4-5-5 = -14< 0

This shows that the surface is visible for the observer.

- # Find the visibility for the surface AED in rectangular pyramid where an observer is at P (0,0.5, 0).

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.
- Also called **z-buffer** method since depth usually measured along z-axis. This approach compares surface depths at each pixel position on the projection plane.
- 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 projection point (x, y) on the view plane.

Depth – Buffer (Z – Buffer Method)

This method requires two buffers:

- A **z-buffer** or **depth buffer**: Stores depth values for each pixel position (x, y).
- **Frame buffer (Refresh buffer):** Stores the surface-intensity values or color values for each pixel position.
- As surfaces are processed, the image buffer is used to store the color values of each pixel position and the z-buffer is used to store the depth values for each (x, y) position.

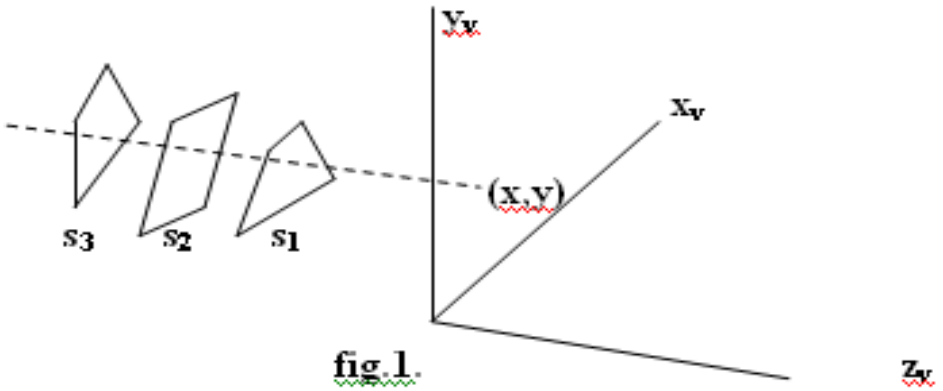
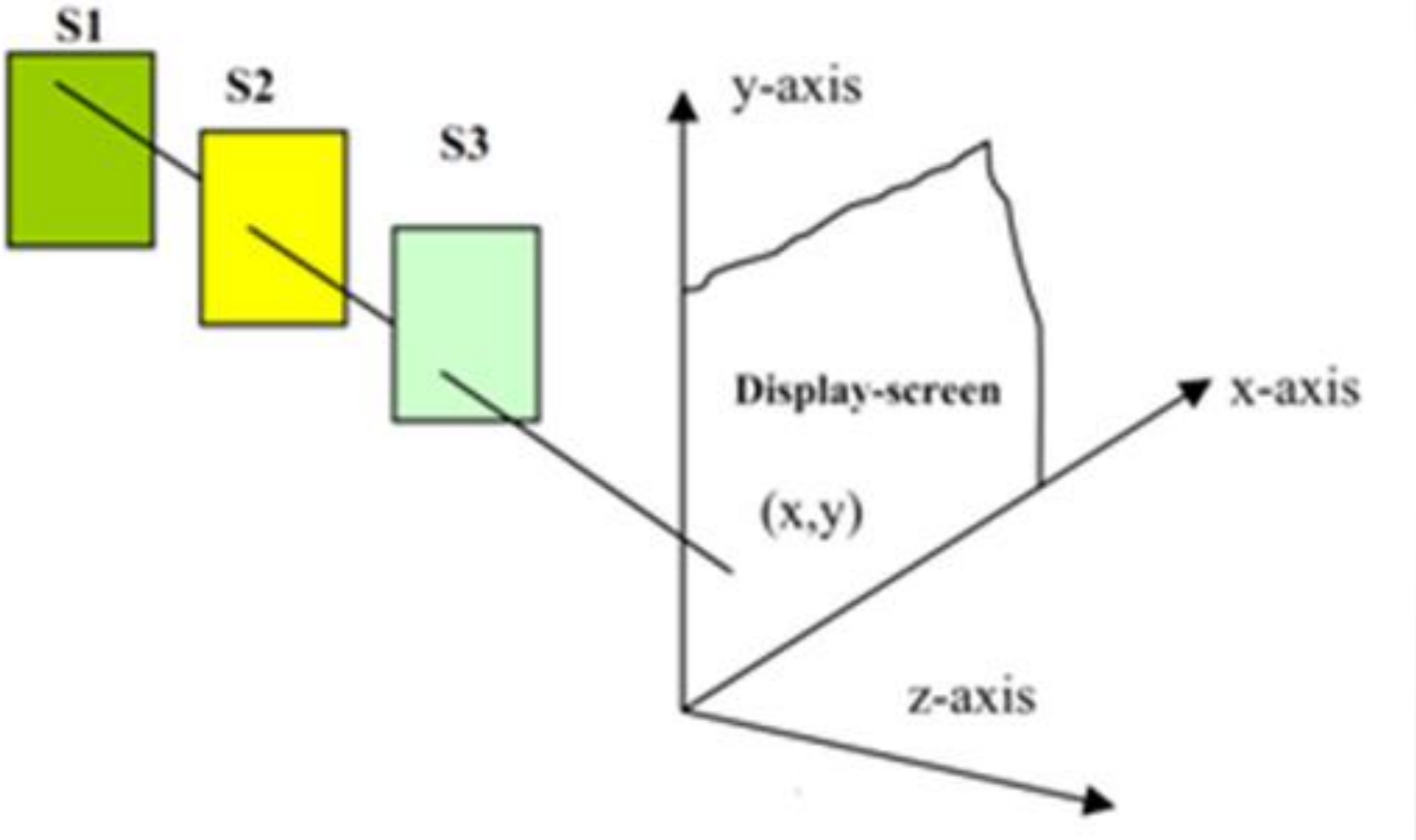


fig.1.

Depth – Buffer (Z – Buffer Method)



Depth – Buffer (Z – Buffer Method)

Initially, all positions in the depth buffer are set to **0** (minimum depth), and 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, and the surface intensity at that position is determined and placed in the same ***xy*** location in the refresh buffer.

A drawback of the depth-buffer method is that it can only find one visible surface for opaque surfaces and cannot accumulate intensity values for transparent surfaces.

Depth – Buffer (Z – Buffer Method)

Algorithm:

1. Initialize both, depth buffer and refresh buffer for all buffer positions (x, y),

   depth(x, y) = 0

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

   (where $I_{background}$ is the value for the background intensity.)

2. Process each polygon surface in a scene one at a time,

   (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.)

   2.1. Calculate the depth z for each (x, y) position on the polygon.

   (The calculated depth is compared to the value previously stored in the depth buffer at that position.)

   2.2. If Z > depth(x, y), then set

   depth(x, y)=z (If the calculated depth is greater than the value stored in the depth buffer, the new depth value is stored,)

   refresh(x, y)= I_{surf}(x, y),

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

3. After all pixels and surfaces are compared, draw object using X,Y,Z from depth and intensity refresh buffer.

Depth – Buffer (Z – Buffer Method)

- After all surfaces have been processed the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces

Depth value for a surface position (x, y) is

$z = (-Ax - By - D)/c$ .....(i)

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

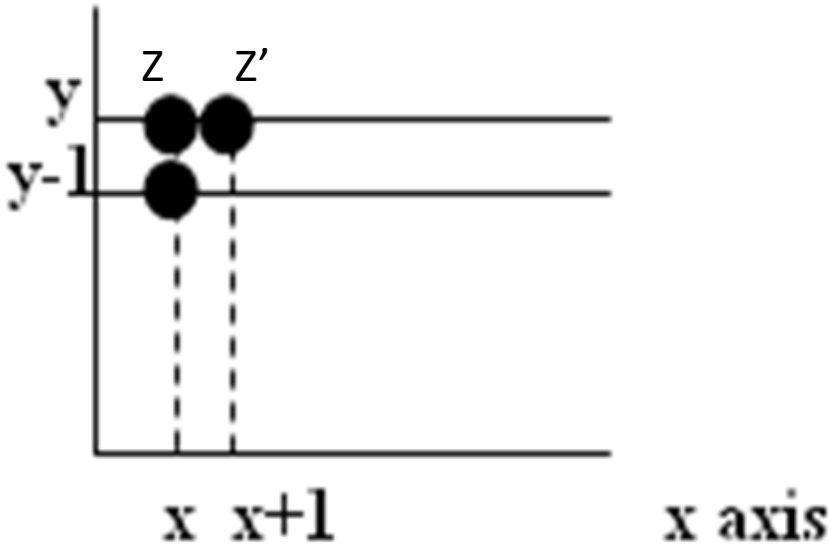
$z' = \{-A(x+1) - By - D\}/c$

$z' = \{-Ax - By - D-A\}/c$

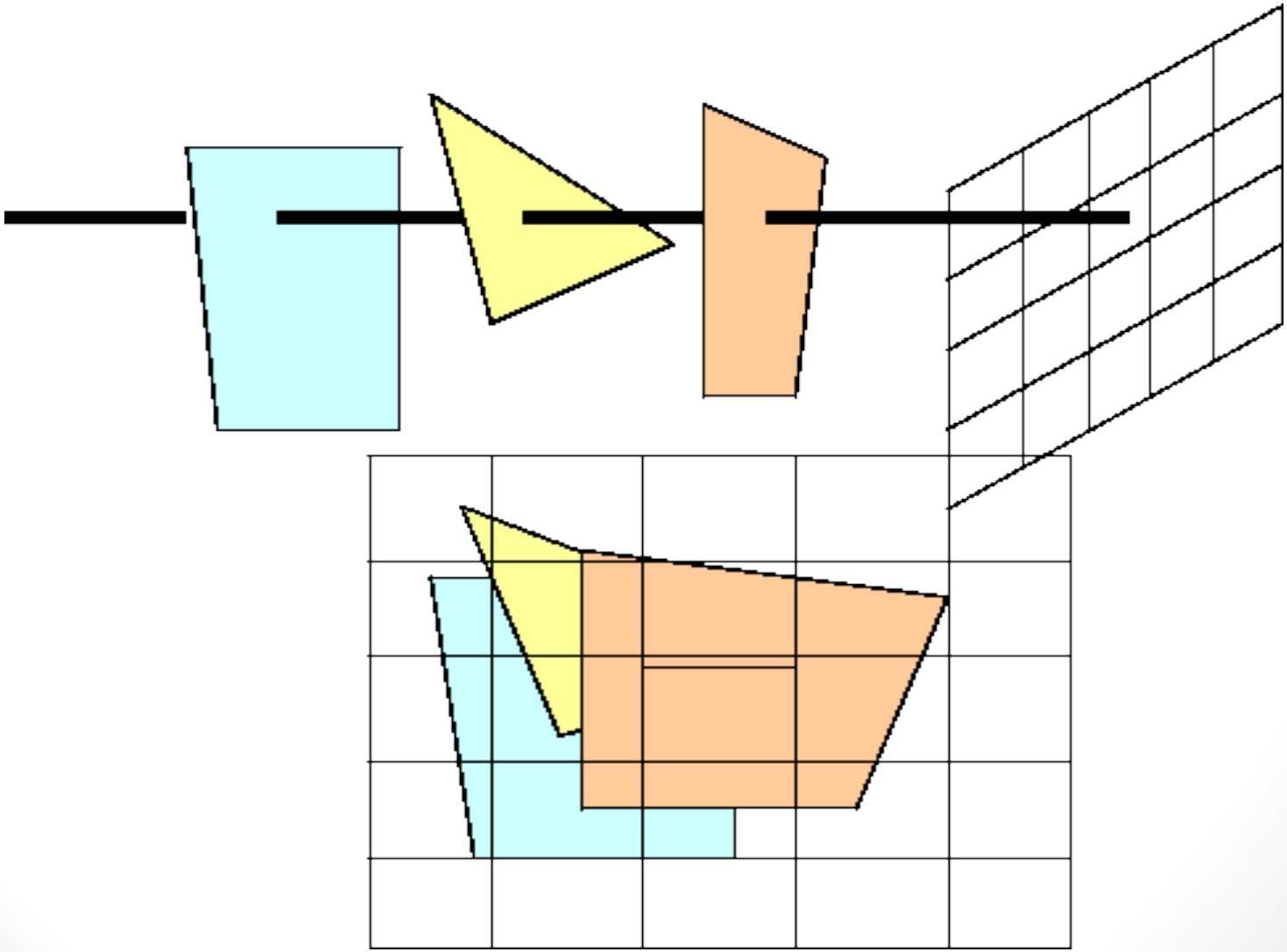
$z' = (-Ax - By - D)/c - A/c$

or

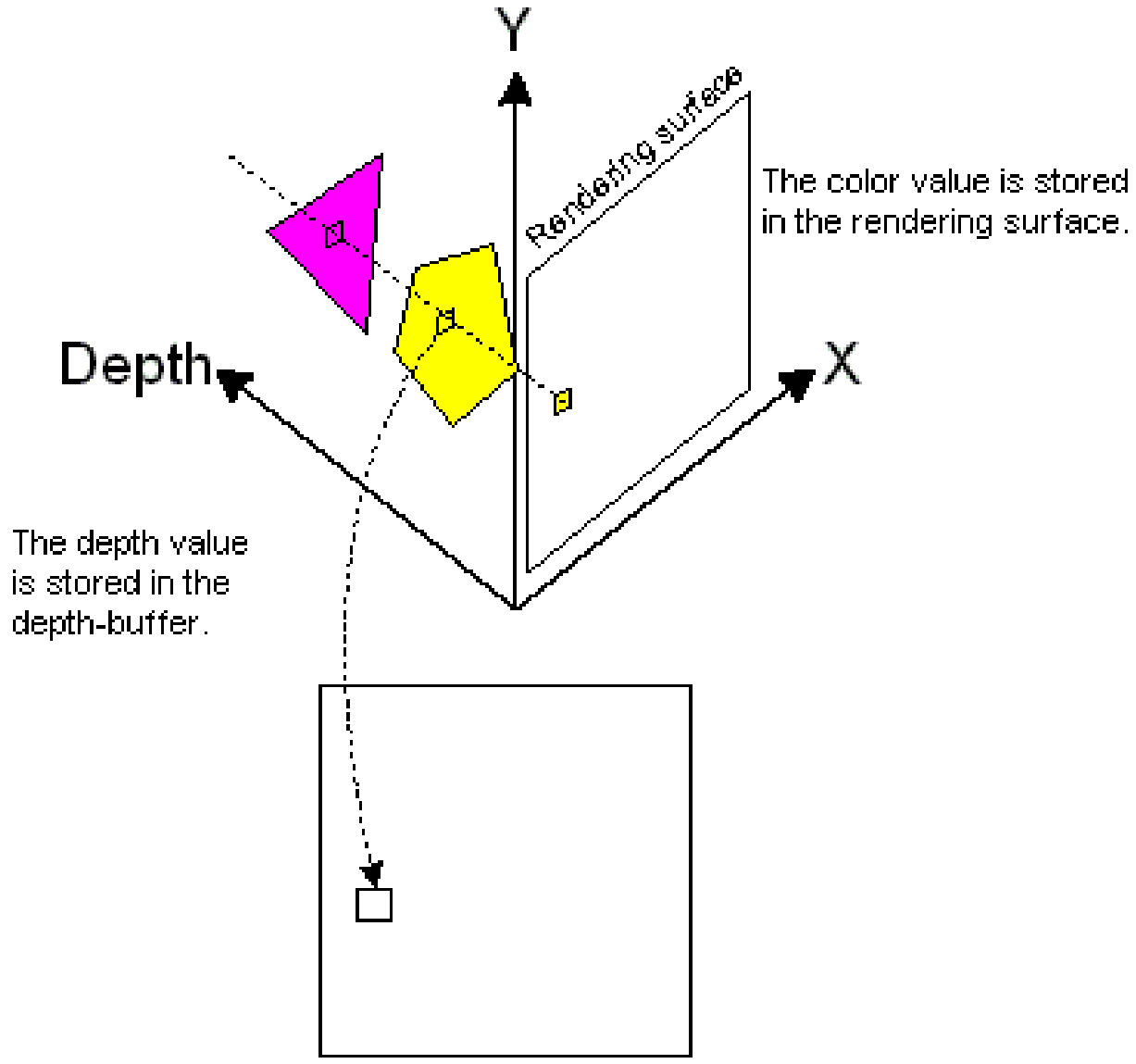
$z' = z - A/c$.....(ii)



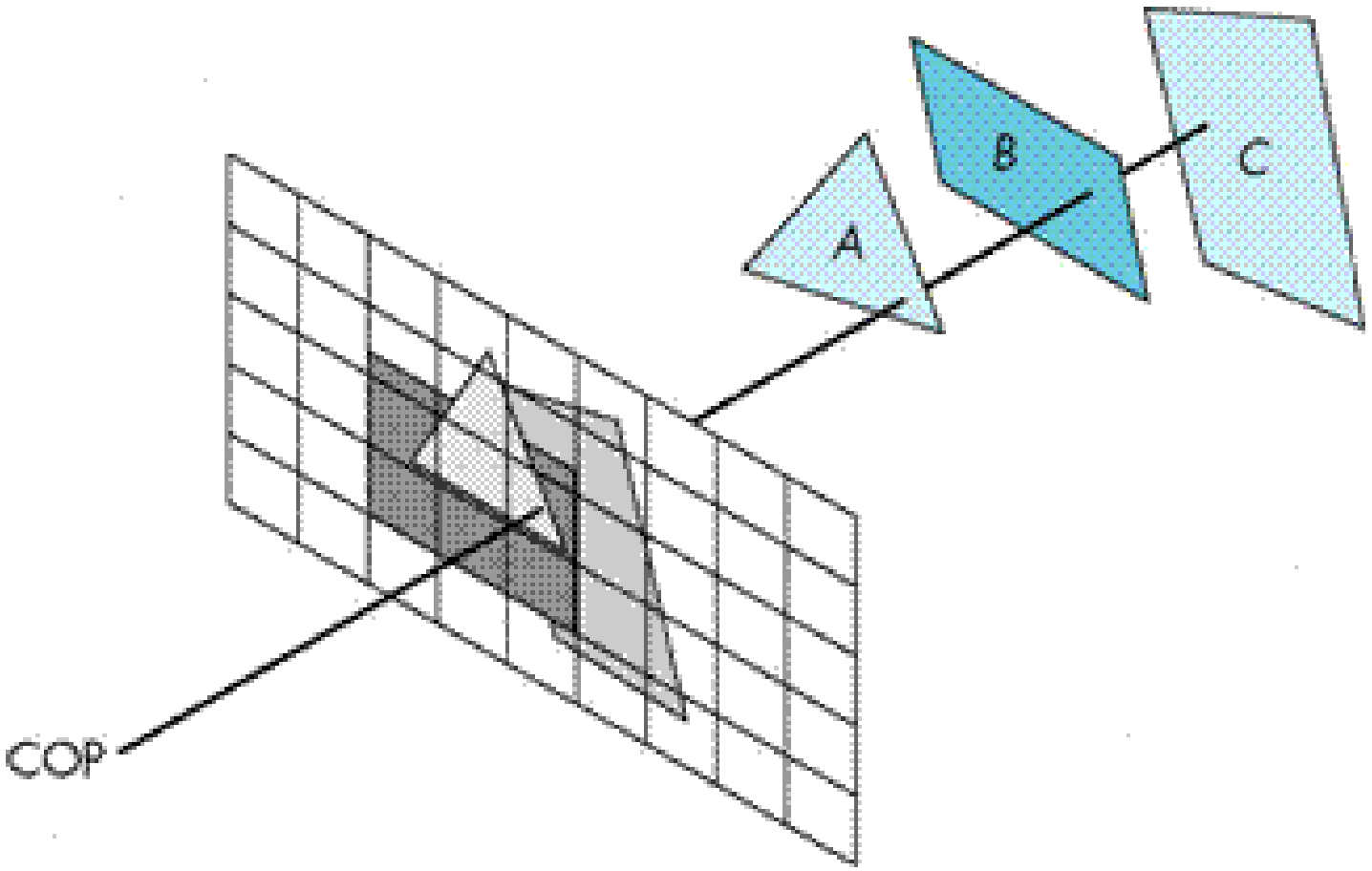
Depth – Buffer (Z – Buffer Method)



Depth – Buffer (Z – Buffer Method)



Depth – Buffer (Z – Buffer Method)



Class Work

Q.N.1> Write a procedure to fill the interior of a given ellipse with a specified pattern. **(2070 TU)**

.

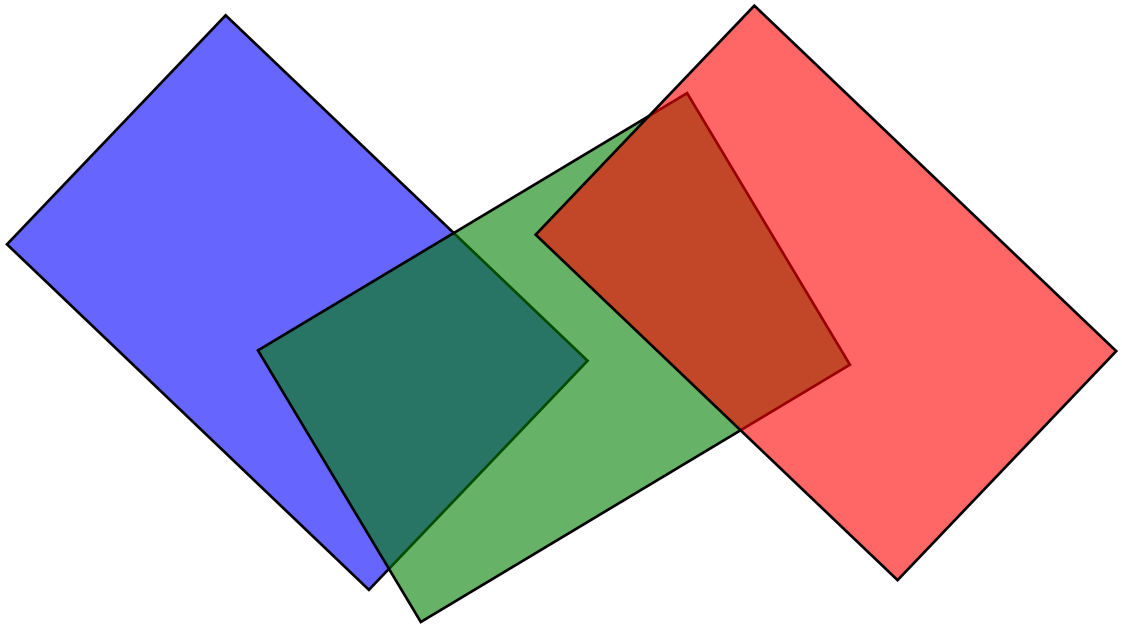
Q.N.2> What do you mean by line clipping? Explain the procedures for line clipping. **(2070 TU)**

A – Buffer Method

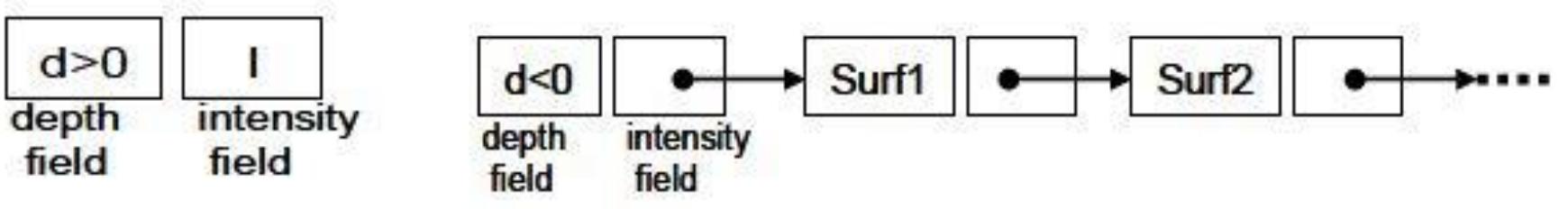
- The **A-buffer** (**anti-aliased**, **area-averaged**, **accumulation** buffer) is an extension of the ideas in the **depth-buffer** method (other end of the alphabet from "**z-buffer**").
- A drawback of the **depth-buffer** method is that it deals only with opaque(Solid) surfaces and cannot accumulate intensity values for more than one transparent surfaces.
- The **A-buffer** method is an extension of the depth-buffer method.
- The **A-buffer** is incorporated into the REYES ("Renders Everything You Ever Saw") 3-D rendering system.
- The **A-buffer** method calculates the surface intensity for multiple surfaces at each pixel position, and object edges can be ant aliased.

A – Buffer Method

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



A – Buffer Method



Each pixel position in the A-Buffer has two fields

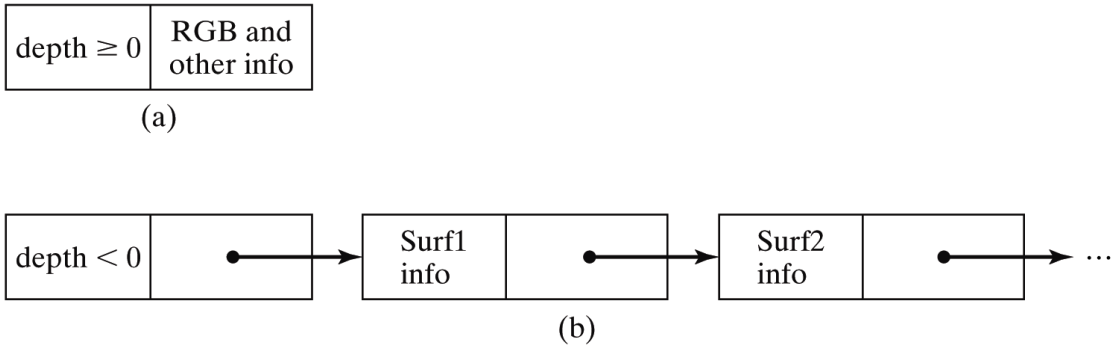
Depth Field : stores a positive or negative real number

- Positive : **single** surface contributes to pixel intensity
- Negative : **multiple** surfaces contribute to pixel intensity

Intensity Field : stores 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
- RGB intensity components
- Opacity parameter(per cent of transparency)
- Per cent of area coverage
- Surface identifier
- Other surface rendering parameters
- Pointer to next surface (Link List)

A – Buffer Method



If depth is >= 0, then the surface data field stores the depth of that pixel position as before (SINGLE SURFACE)

(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 RCB components of the surface color at that point and the percent of pixel coverage, as illustrated first figure.)

If depth < 0 then the data filed stores a pointer to a linked list of surface data (MULTIPLE SURFACE)

(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, as in second figure. Data for each surface in the linked list includes: RGB intensity components, opacity parameter (percent of transparency), depth, percent of area coverage, surface identifier, other surface-rendering parameters, and pointer to next surface)

A – Buffer Method

- The A-buffer can be constructed using methods similar to those in the depth-buffer algorithm. Scan lines are processed to determine surface overlaps of pixels across the individual scan lines. Surfaces are subdivided into a polygon mesh and clipped against the pixel boundaries. Using the opacity factors and percent of surface overlaps, we can calculate the intensity of each pixel as an average of the contributions from the over lapping surfaces.

Class Work

Q.N.1 > Explain with algorithm of generating curves. **(TU 2071)**

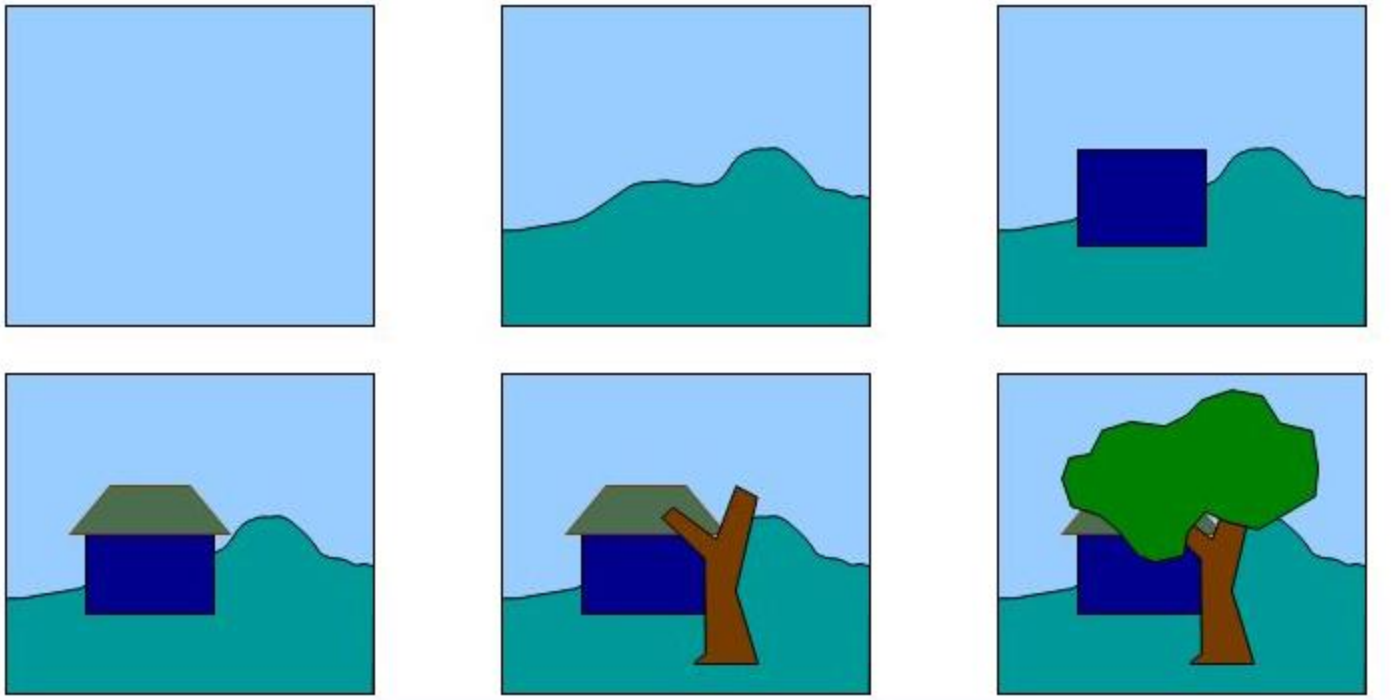
Q.N.2 > Set up a procedure for establishing polygon tables for any input set of data defining an object. **(TU 2071)**

Q.N.3 > Explain the window to view port transformation with its applications. **(TU 2071/2070)**

Q.N.4 > Write a procedure to perform a two-point perspective projection of an object. **(TU 2070)**

DEPTH SORT (Painter Algorithm)

- This method uses both **object space** and **image space** method.
- In this method the surface representation of 3D object are sorted in of decreasing depth from viewer.
- Then sorted surface are scan converted in order starting with surface of greatest depth for the viewer.

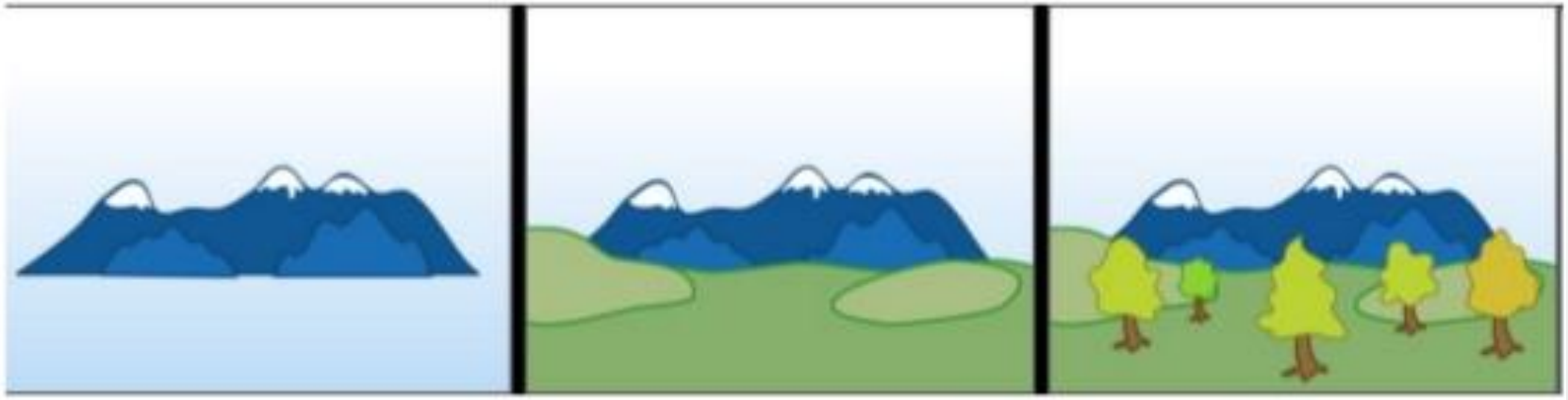


The conceptual steps that performed in depth-sort algorithm are

1. Sort all polygon surface according to the smallest (farthest) Z co-ordinate of each.
2. Resolve any ambiguity(doubt) this may cause when the polygons Z extents overlap, splitting polygons if necessary.
3. Scan convert each polygon in ascending order of smaller Z-co-ordinate i.e. farthest surface first (back to front)

DEPTH SORT (Painter Algorithm)

- In this method, the newly displayed surface is partly or completely obscure the previously displayed surface. Essentially, we are sorting the surface into priority order such that surface with lower priority (lower z, far objects) can be obscured by those with higher priority (high z-value).

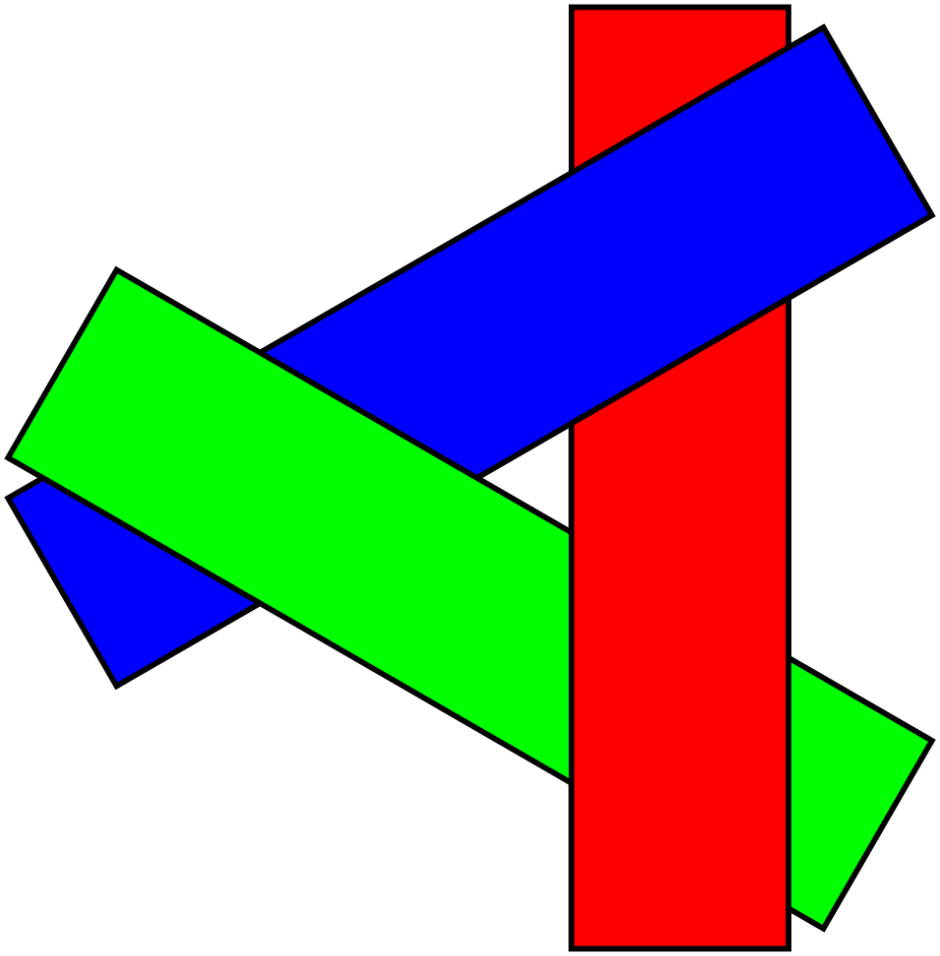


DEPTH SORT (Painter Algorithm)

- This algorithm is also called "**Painter's Algorithm**" as it simulates how a painter typically produces his painting by starting with the background and then progressively adding new (nearer) objects to the canvas.
- Thus, each layer of paint covers up the previous layer.
- Similarly, 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.

DEPTH SORT (Painter Algorithm)

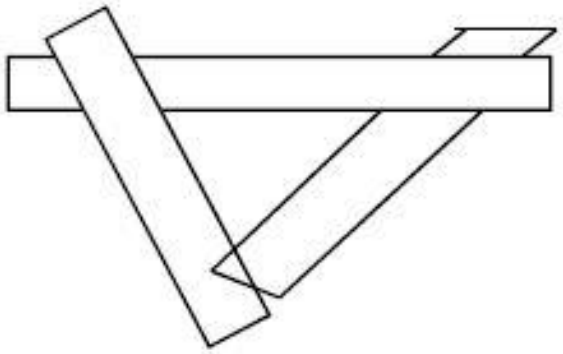
Problem



DEPTH SORT (Painter Algorithm)

Problem

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



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

We cannot use simple depth-sorting to remove the hidden-surfaces in the images.

DEPTH SORT (Painter Algorithm)

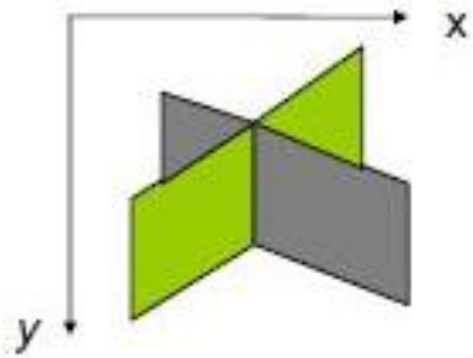
Solution

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

DEPTH SORT (Painter Algorithm)

- Algorithm:
  - Sort the polygons in the scene by their depth
  - Draw them back to front
- **Problem**: Unless all polygons have constant z, a strict depth ordering may not exist

Note: Constant z case is important in VLSI design



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.

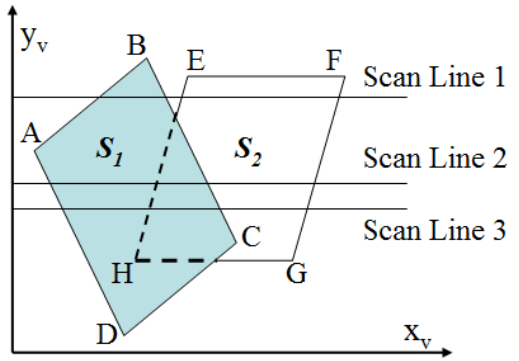


Fig. Scan lines crossing the projection of two surfaces, S_1 and S_2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

Scan-Line Method

- 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.

Scan-Line Method

DATA STRUCTURE

- **A. Edge table** containing
  - Coordinate endpoints for each line in a scene
  - Inverse slope of each line
  - Pointers into polygon table to identify the surfaces bounded by each line
- **B. Surface table** containing
  - Coefficients of the plane equation for each surface
  - Intensity information for each surface
  - Pointers to edge table
- **C. Active Edge List**
  - To keep a trace of which edges are intersected by the given scan line

Note :

- The edges are sorted in order of increasing x
- Define flags for each surface to indicate whether a position is inside or outside the surface

Scan-Line Method

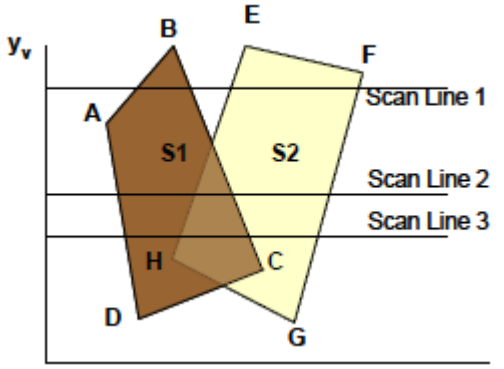
I. Initialize the necessary data structure

1. Edge table containing end point coordinates, inverse slope and polygon pointer.
2. Surface table containing plane coefficients and surface intensity
3. Active Edge List
4. Flag for each surface

II. For each scan line repeat

1. update active edge list
2. determine point of intersection and set surface on or off.
3. If flag is on, store its value in the refresh buffer
4. If more than one surface is on, do depth sorting and store the intensity of surface nearest to view plane in the refresh buffer

Scan-Line Method



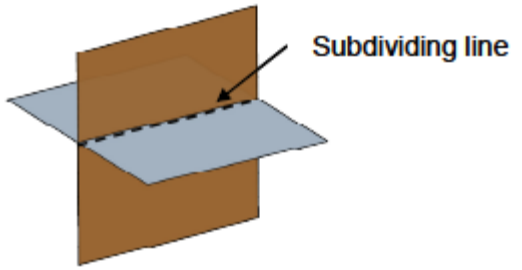
- For scan line 1
  - The active edge list contains edges AB,BC,EH, FG
  - Between edges AB and BC, only *flags for s1 == on* and between edges EH and FG, only *flags for s2==on*
    - no depth calculation needed and corresponding surface intensities are entered in refresh buffer
- For scan line 2
  - 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 ,say s2 is nearer to the view plane than s1, so intensities for surface s2 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

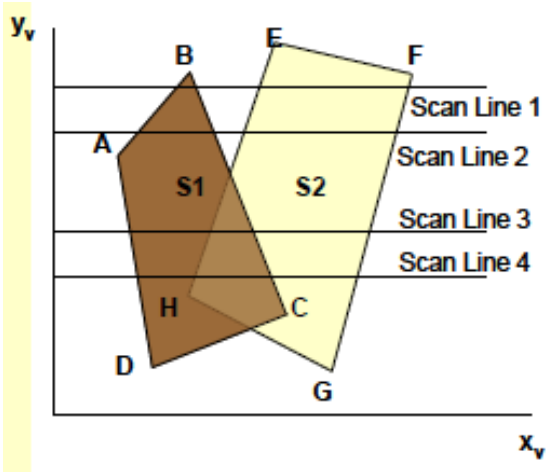
Scan-Line Method

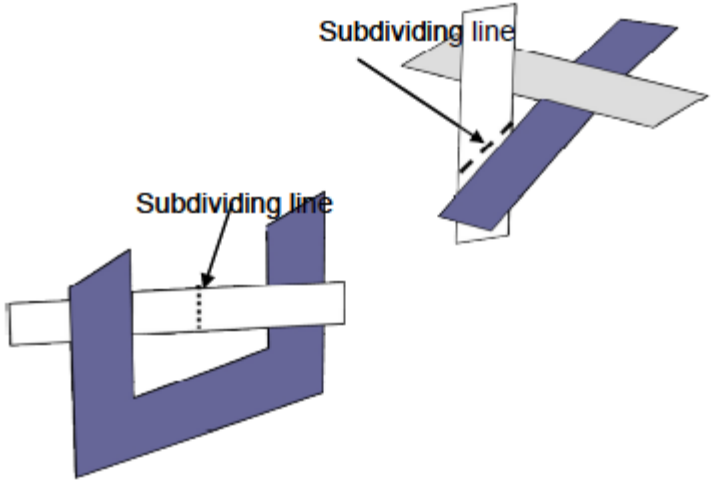
Problem:

Dealing with **cut through** surfaces and **cyclic overlap** is problematic when used coherent properties

- Solution: Divide the surface to eliminate the overlap or cut through





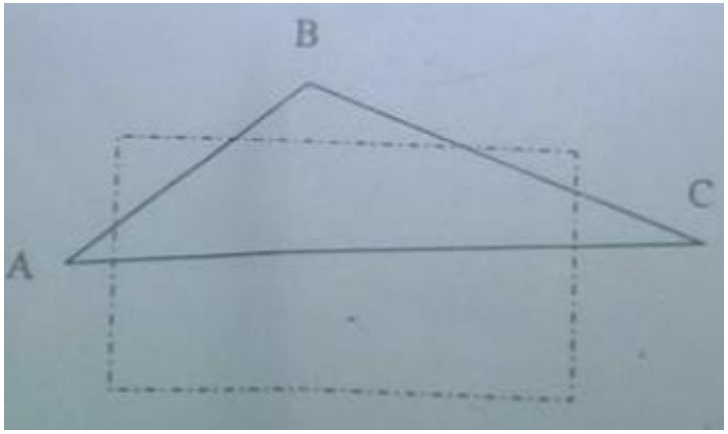


Class Work

Q.N.1> What do you mean by homogeneous coordinates? Rotate a triangle A(5,6), B(6,2) and C(4,1) by 45 degree about an arbitrary pivot point (3,3). **(TU 2072)**

Q.N.2> Given a clipping window P(0,0), Q(30,20), S(0,20) use the Cohen Sutherland algorithm to determine the visible portion of the line A(10,30) and B(40,0). **(TU 2072)**

Q.N.3> Explain polygon clipping in detail. By using the sutherland-Hodgemen Polygon clipping algorithm clip the following polygon. **(TU 2072)**



Binary Space Partitioning (BSP)

- Binary space partitioning is a 3-D graphics programming technique of dividing a scene into two recursively using hyperplanes.
- In other words, a 3-D scene is split in two using a 2-D plane, then that scene is divided in two using a 2-D plane, and so on. The resulting data structure is a binary tree, or a tree where every node has two branches.
- The technique is widely used to speed up rendering of 3-D scenes, especially in games.

Binary Space Partitioning (BSP)

- **binary space partitioning** (**BSP**) is a method for recursively subdividing a space into convex sets by hyperplanes. This subdivision gives rise to a representation of objects within the space by means of a tree data structure known as a **BSP tree**.
- Binary space partitioning was developed in the context of 3D computer graphics, where the structure of a BSP tree allows spatial information about the objects in a scene that is useful in rendering, such as their ordering from front-to-back with respect to a viewer at a given location, to be accessed rapidly. Other applications include performing geometrical operations with shapes (constructive solid geometry) in CAD, collision detection in robotics and 3D video games, ray tracing and other computer applications that involve handling of complex spatial scenes.

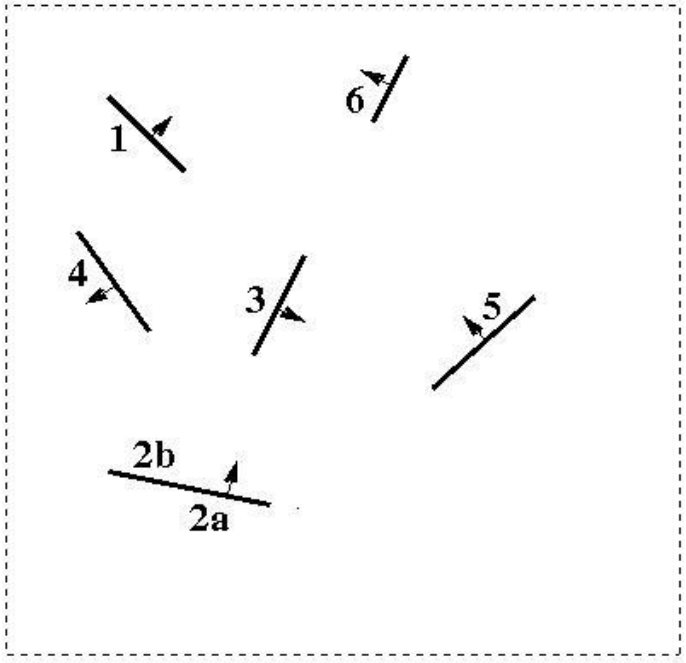
Binary Space Partitioning (BSP)

- Previous list priority algorithms fail in a number of cases non of them is completely general
- BSP tree is a general solution, but with its own problems
  - Tree size
  - Tree accuracy

Binary Space Partitioning Trees (Fuchs, Kedem and Naylor `80)

- More general, can deal with inseparable objects
- Automatic, uses as partitions planes defined by the scene polygons
- Method has two steps:
  - building of the tree independently of viewpoint
  - traversing the tree from a given viewpoint to get visibility ordering

Building a BSP Tree (Recursive)

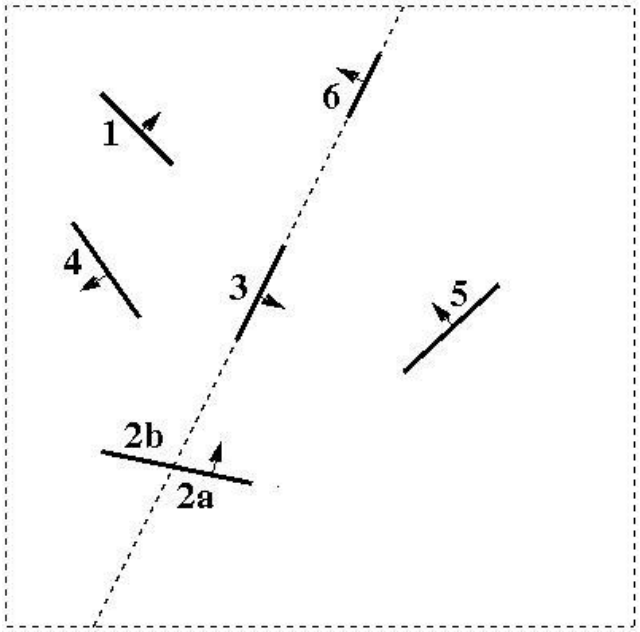


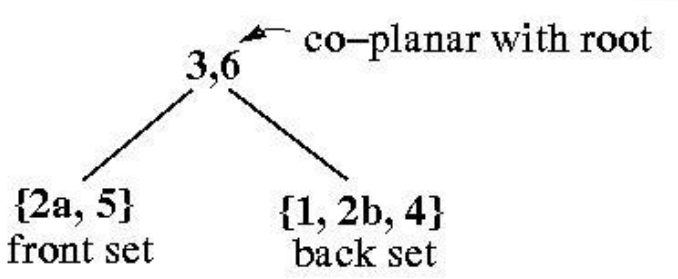
A set of polygons

{1, 2, 3, 4, 5, 6}

The tree

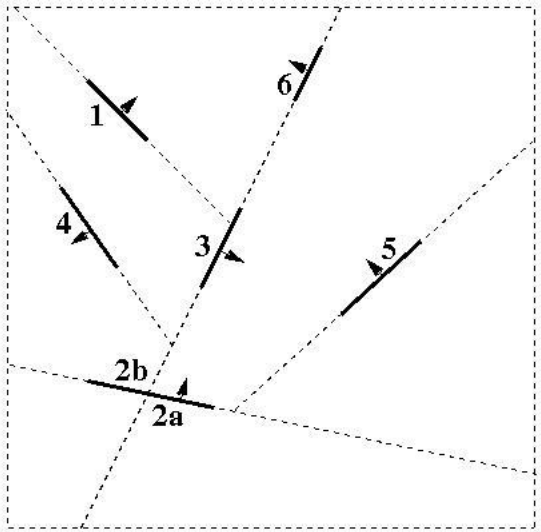
Building a BSP Tree (Recursive)

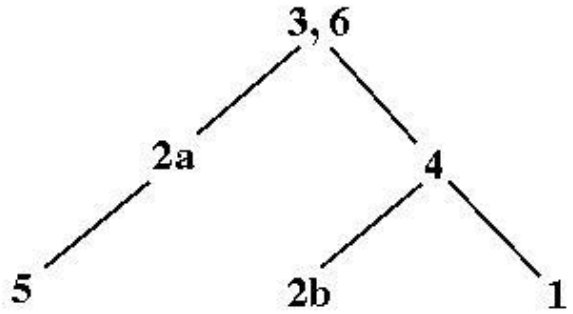




Select one polygon and partition the space and the polygons

Building a BSP Tree (Recursive)





Recursively partition each sub-tree until all polygons are used up

Building a BSP Tree (Recursive)

- Start with a set of polygons and an empty tree
- Select one of them and make it the root of the tree
- Use its plane to divide the rest of the polygons in 3 sets: front, back, coplanar.
  - Any polygon crossing the plane is split
- Repeat the process recursively with the front and back sets, creating the front and back subtrees respectively

Building a BSP Tree (Incremental)

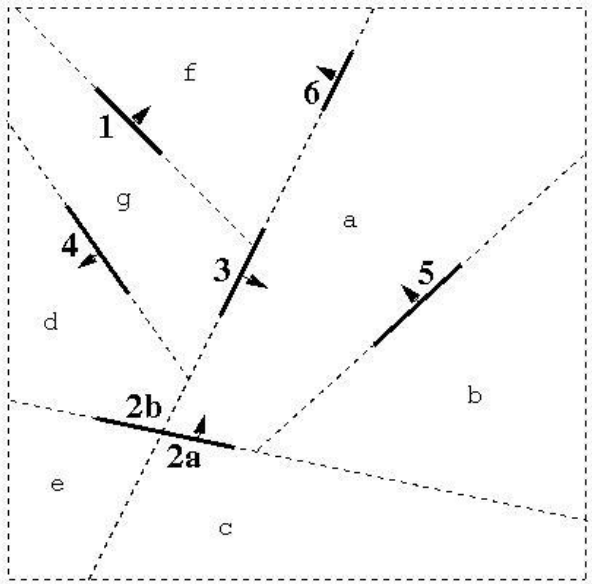
- The tree can also be built incrementally:
  - start with a set of polygons and an empty tree
  - insert the polygons into the tree one at a time
  - insertion of a polygon is done by comparing it against the plane at each node and propagating it to the right side, splitting if necessary
  - when the polygon reaches an empty cell, make a node with its supporting plane

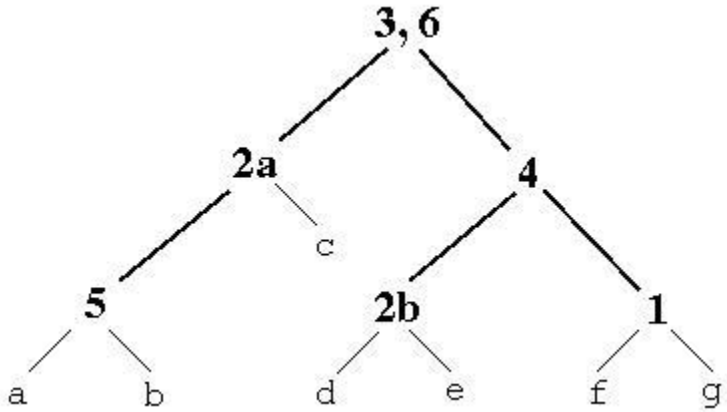
Back-to-Front Traversal

```
void traverse_btf(Tree *t, Point vp)
{
        if (t = NULL) return;
        endif

        if (vp in-front of plane at root of t)
                traverse_btf(t->back, vp);
                draw polygons on node of t;
                traverse_btf;(t->front, vp);
        else
                traverse_btf(t->front, vp);
                draw polygons on node of t;
                traverse_btf(t->back, vp);
        endif
}
```

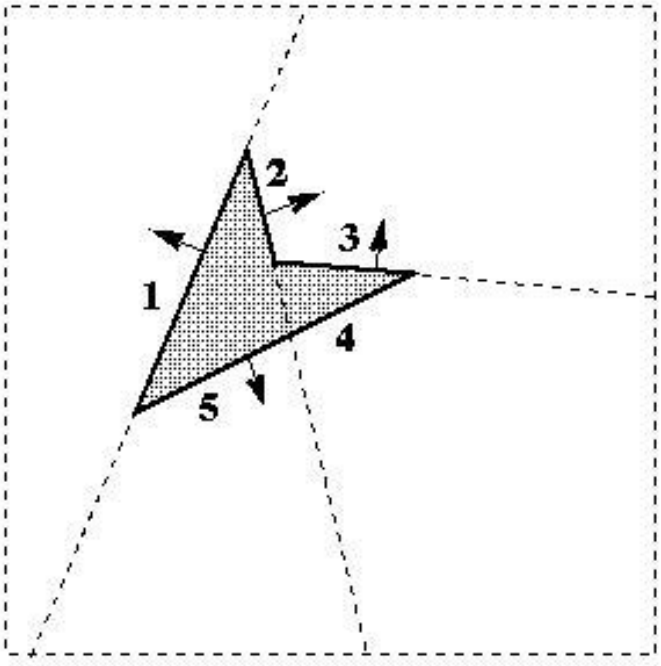
BSP as a Hierarchy of Spaces

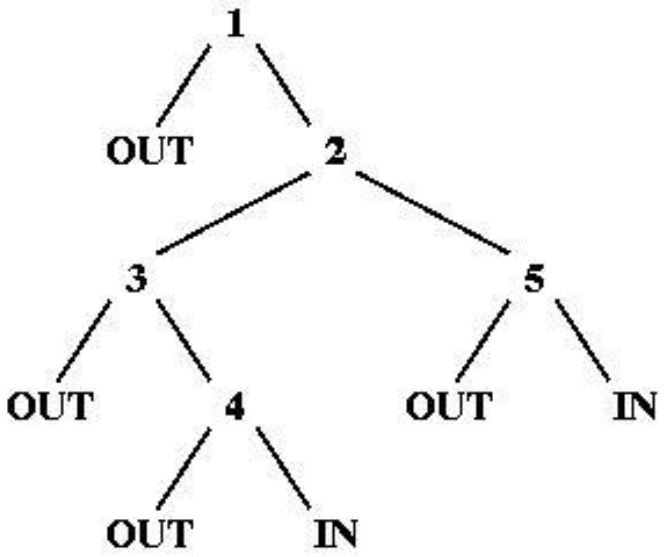




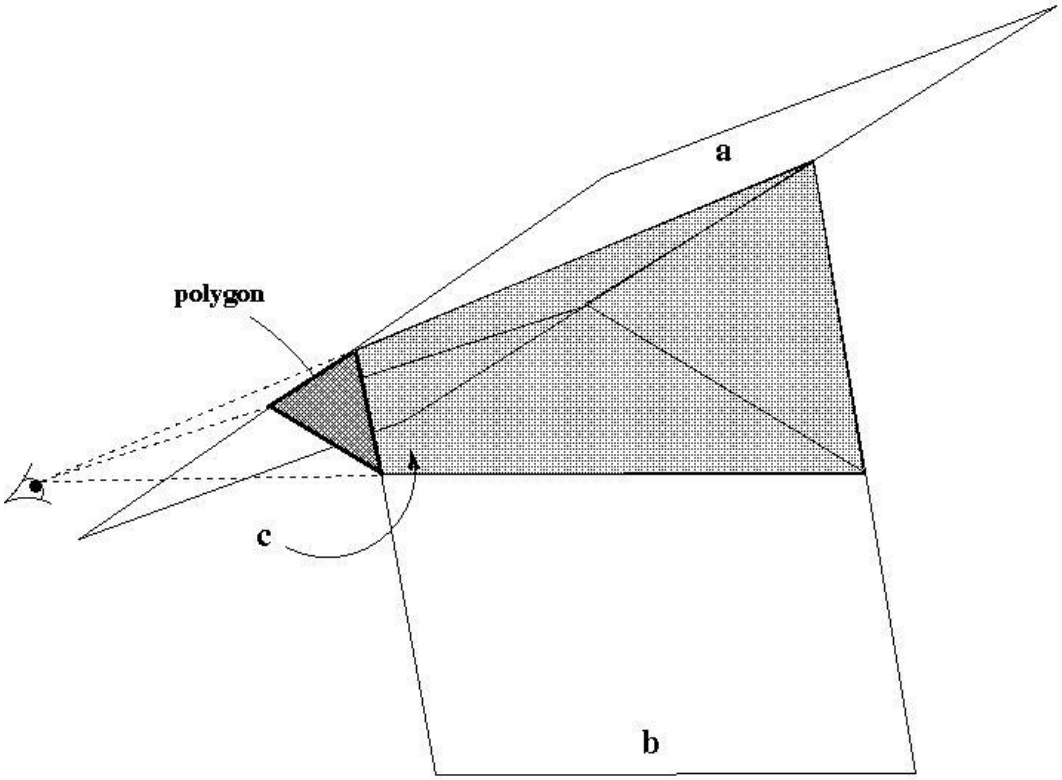
- Each node corresponds to a region of space
  - the root is the whole of R^n
  - the leaves are homogeneous regions

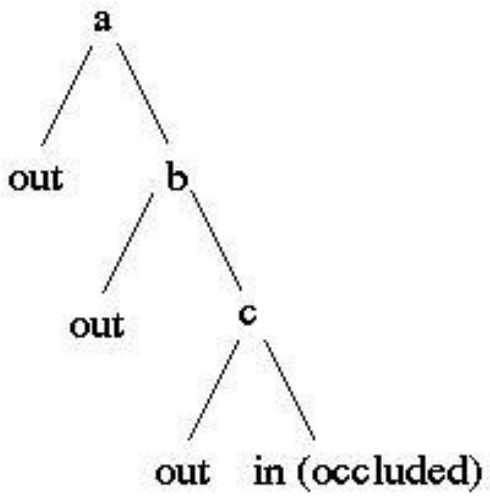
Representation of Polygons





Representation of Polyhedra





BSP Trees for Dynamic Scenes

- When an object moves the planes that represent it must be removed and re-inserted
- Some systems only insert static geometry into the BSP tree
- Otherwise must deal with merging and fixing the BSP cells (see the book!)

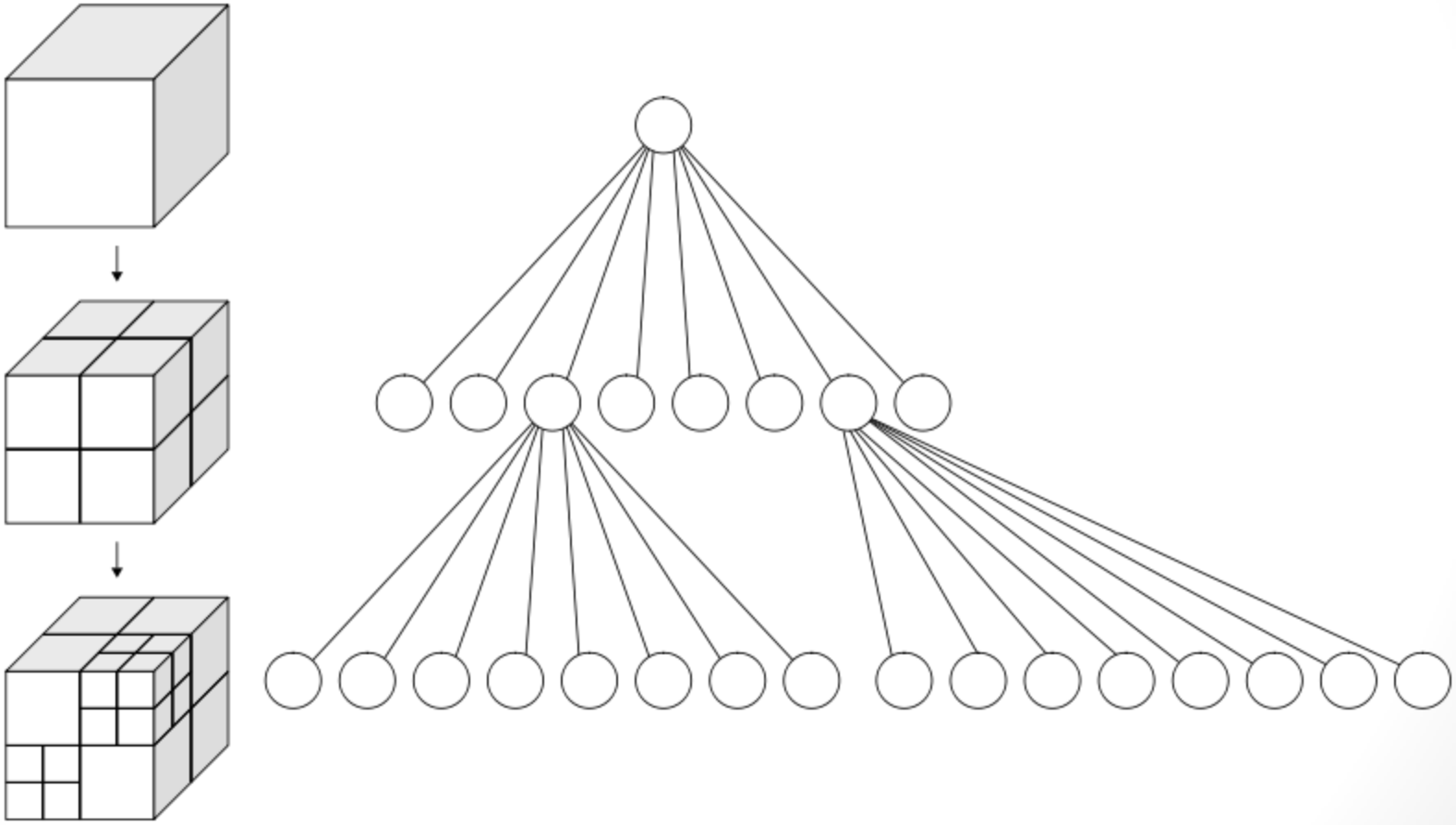
Recap

- A BSP is a sequence of binary partitions of space
- Can be built recursively or incrementally
- Choice of plane used to split is critical
- BSP trees are hard to maintain for dynamic scenes

octree

- **octree** is a tree data structure in which each internal node has exactly eight children. Octrees are most often used to partition a three-dimensional spaceby recursively subdividing it into eight octants.
- Octrees are the three-dimensional analog of quadtrees. The name is formed from *oct* + *tree*, but note that it is normally written "*octree*" with only one "t". Octrees are often used in 3D graphics and 3D game engines.

octree



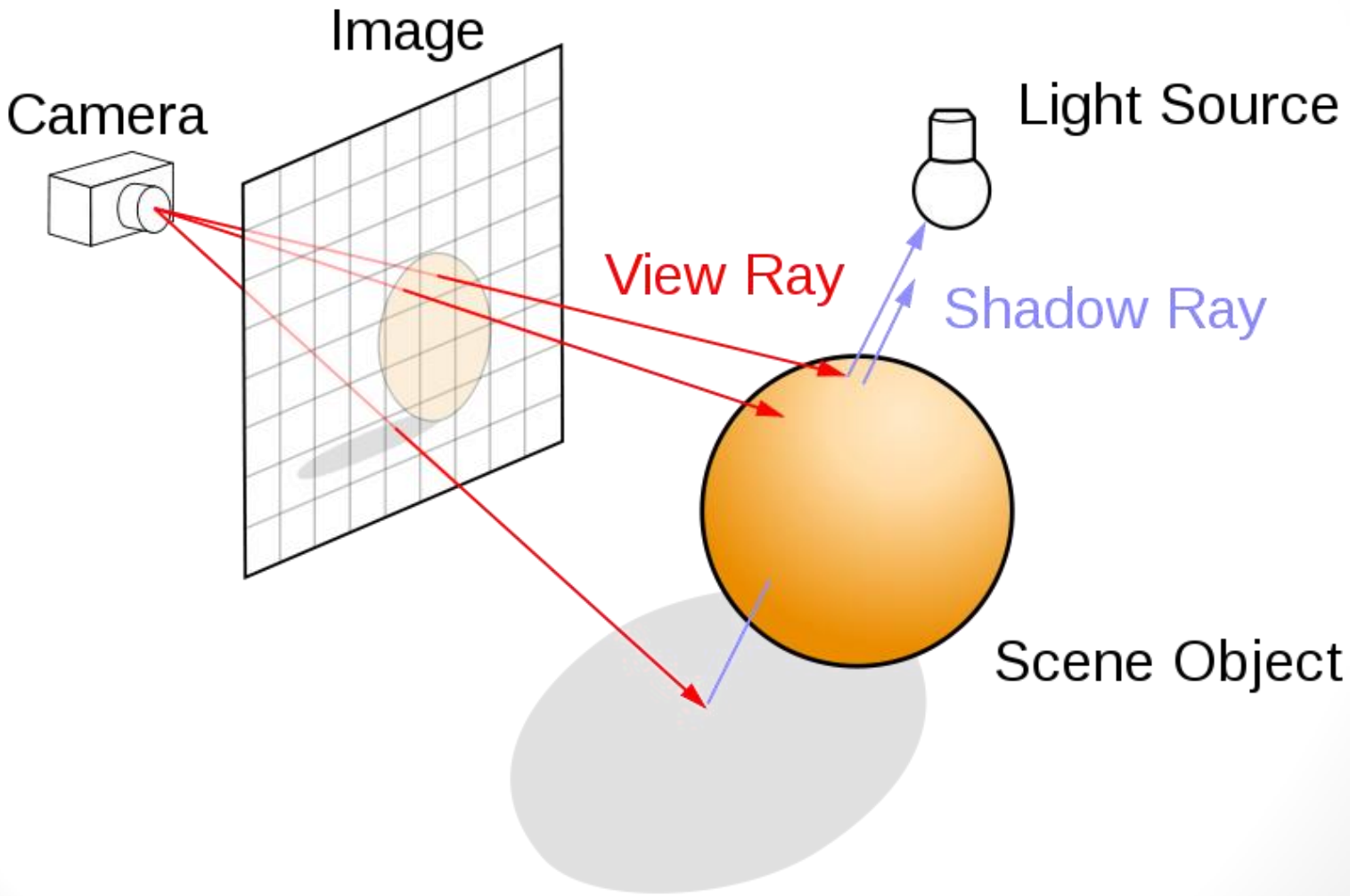
Ray tracing

- **Ray tracing** is a rendering technique for generating an image by tracing the path of light as pixels in an image plane and simulating the effects of its encounters with virtual objects.
- The technique is capable of producing a very high degree of visual realism, usually higher than that of typical scanline renderingmethods, but at a greater computational cost.

Ray tracing

- This makes ray tracing best suited for applications where taking a relatively long time to render a frame can be tolerated, such as in still images and film and television visual effects, and more poorly suited for real-time applications such as video games where speed is critical. Ray tracing is capable of simulating a wide variety of optical effects, such as reflection and refraction, scattering,and dispersion phenom ena (such as chromatic aberration).

Ray tracing



Ray tracing

- Optical ray tracing describes a method for producing visual images constructed in 3D computer graphic senvironments, with more photorealism than either ray casting or scanline rendering techniques.
- It works by tracing a path from an imaginary eye through each pixel in a virtual screen, and calculating the color of the object visible through it.

Chapter 6 Finished