Visible surface Detection

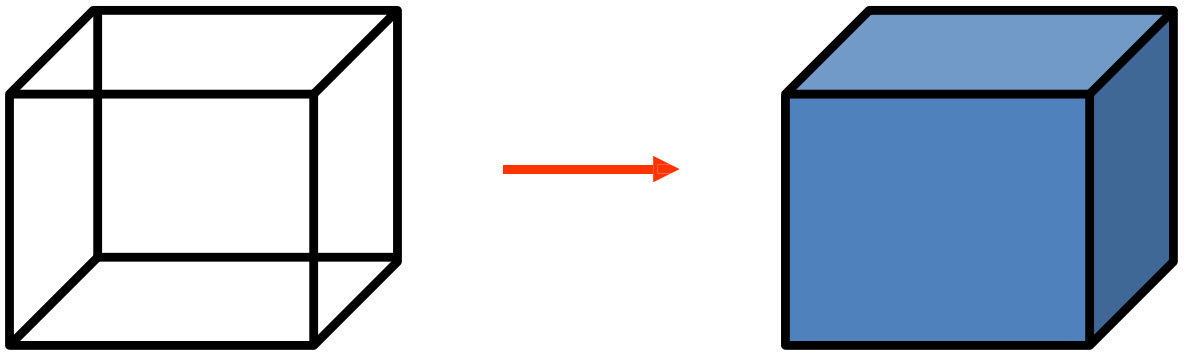
What is Visible surface Detection

- For creating realistic view of any object we must draw those line which are visible to the user, not those lines which are not viewed by the user.
- The identification and removal of these surface is called “Visible surface detection” and the algorithm used for this purpose are called “visible surface detection algorithm”

Visible-Surface Detection

Problem:

Given a scene and a projection, what can we see?



Visible Surface Detection Algorithms

- Z-Buffer or Depth Buffer Algorithm
- BSP Algorithm (Binary Space Partition Algorithm)
- Back-face detection algorithm
- Area Sub-Division
- Scan-line Algorithm
- Painter’s or Depth sorting Algorithm
- Octree Method

Right hand and Left hand Coordinate System

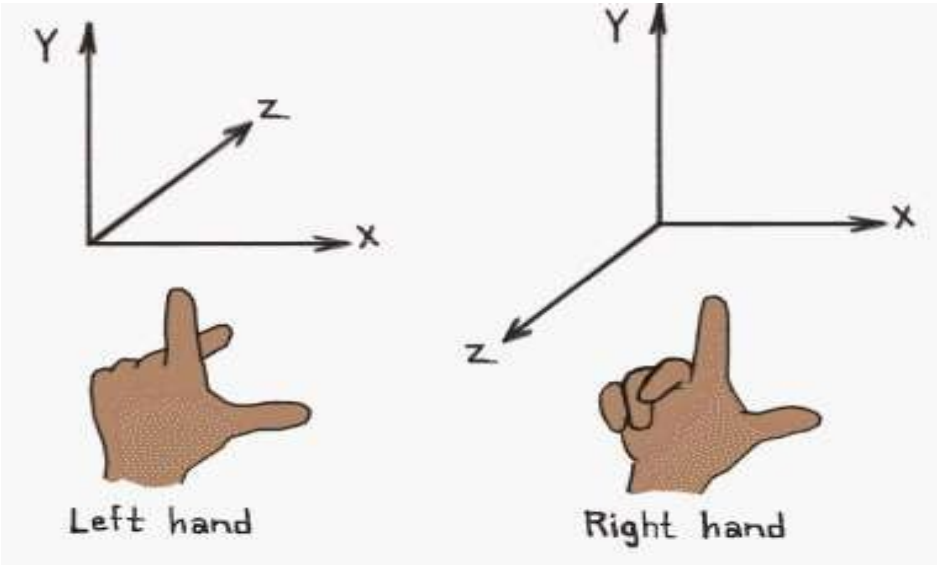


Figure 1

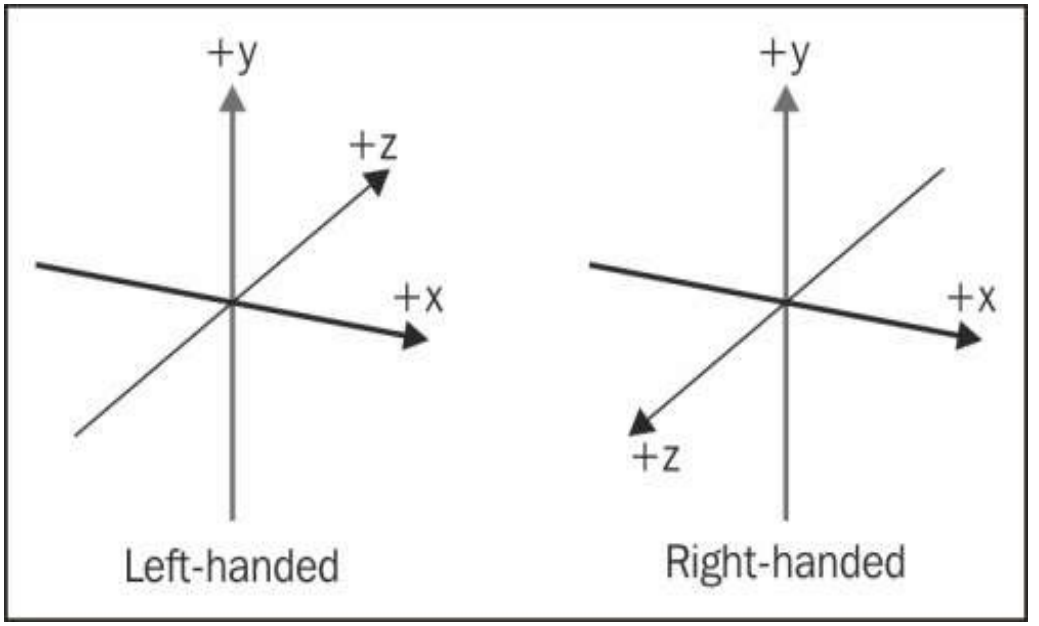


Figure 2

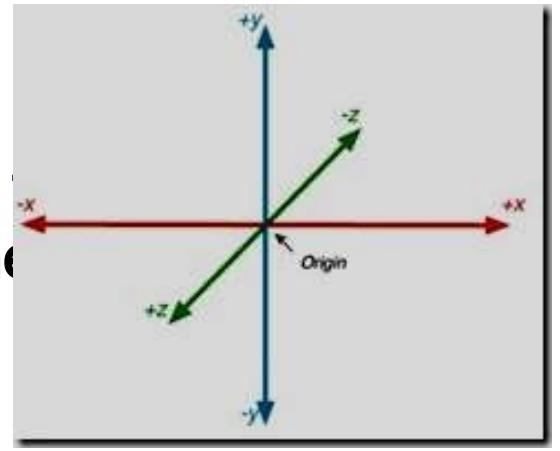
General Z-buffer Algorithm

- In Z buffer algorithm we compare the Z values of the pixels which are overlapped with each other.
- According to the direction of the user we take maximum Z value or minimum Z values to initialize the Z buffer.
- No need to compare Z value of those pixels which are not overlapped with each other.

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

1. When user sits on positive z axis and looks towards negative z axis
2. When user sits on negative z axis and looks towards positive z axis

Case-1:



In first case the object with smaller Z value is
and object having more Z value is closer to the

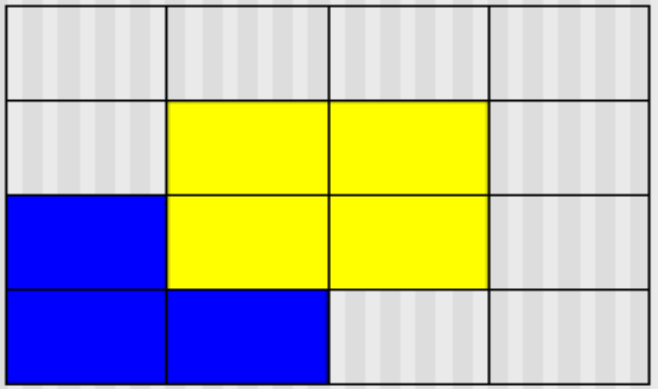
Case 2:

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

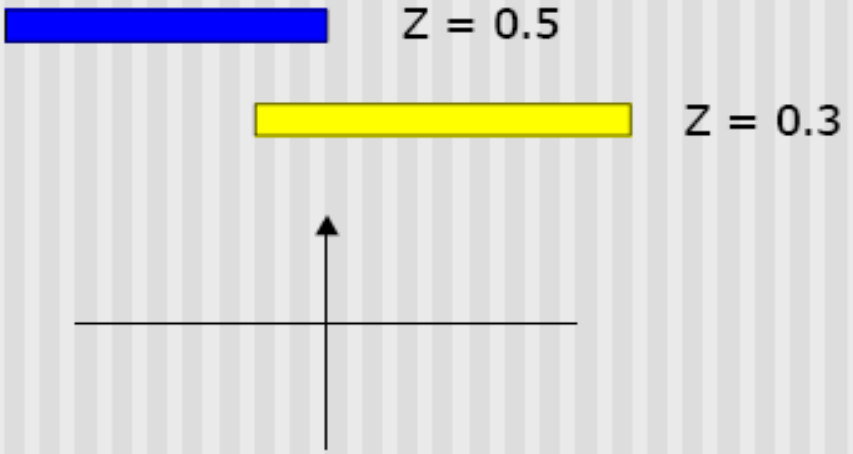
Hence Z buffer algorithm is depend upon the viewing direction of the user.

Z buffer example

When user sits on -z direction and looks towards +z direction

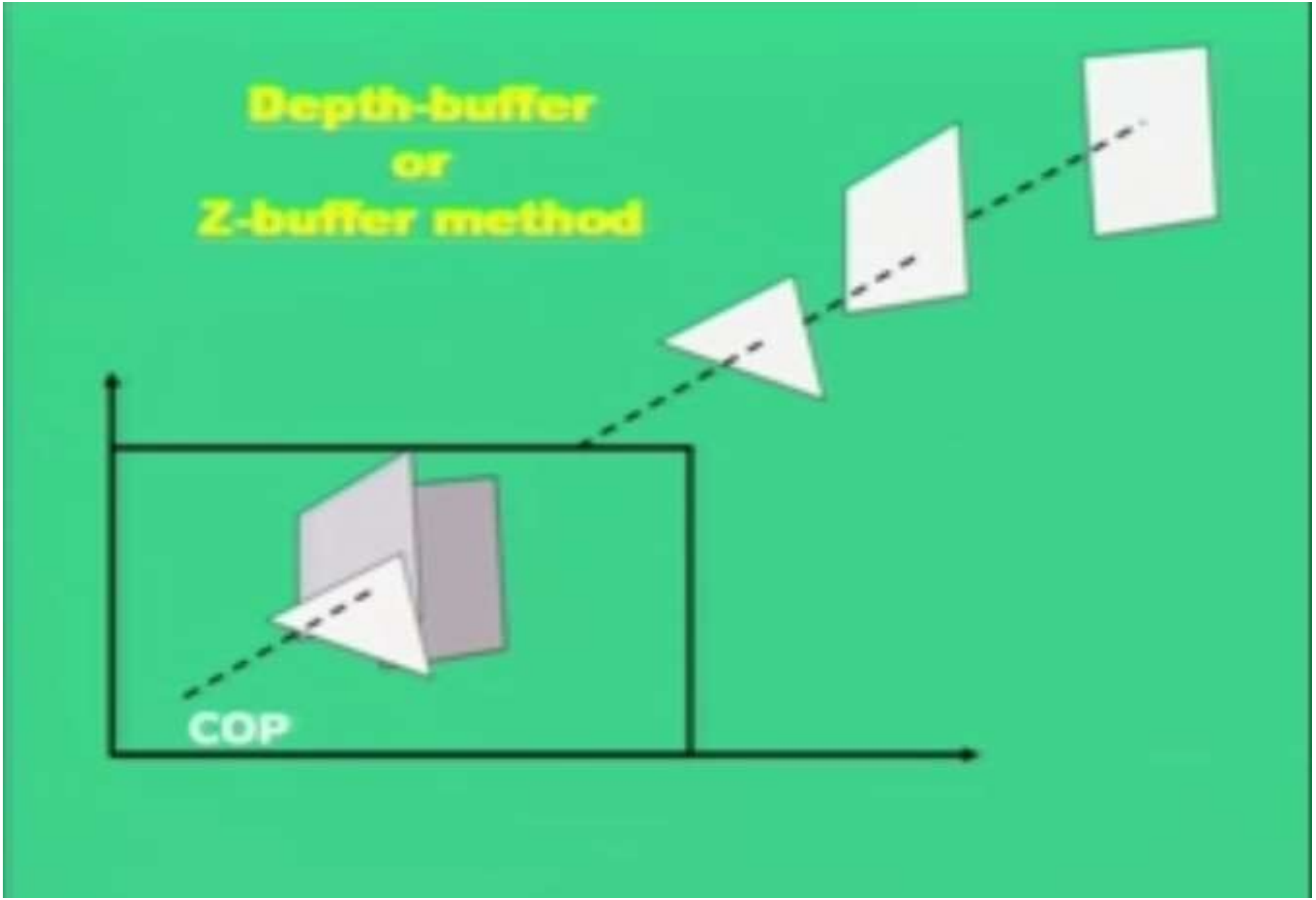


Correct Final image



Top View

- Here user's viewing direction is towards positive Z-direction
- Hence object having minimum Z values are closer to user and objects having larger Z-values are away from user.



In this figure we see three polygons in window, which are occluding each others. It means they are not separated out..

Important Concept of Z-Buffer Algorithm

Each (X,Y,Z) point on a polygon surface, corresponds to the orthographi projection (X,Y) on the view point plane.

At each point (X,Y) on the PP, object depths are compared by using the depth (Z) values.

Buffers used in Z-Buffer Algorithm

Two buffers are used :

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

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

Two Case In Z-Buffer Algorithm

1. When Viewing direction is from + Z to –Z
2. When Viewing Direction is from –Z to +Z.

In First Case Polygon having less Z value are on back side and other are on front side.

In second Case polygon having more Z value are on backside and other are on front side.

When View Direction is from –Z to +Z direction

Steps for Processing:

(a) Initialize

$$\forall (X,Y) \quad depth(X,Y) = Z_{max}, \quad refresh(X,Y) = I_s$$

I_s = background intensity

(b) For each position on each polygon surface:

(i) Calculate depth Z for each position (X, Y) on the polygon.

(ii) If Z < depth(X, Y) then

depth (X, Y) = Z;

refresh (X, Y) = I_s (X, Y)

When View Direction is from +Z to -Z direction

Z- Buffer Algorithm:

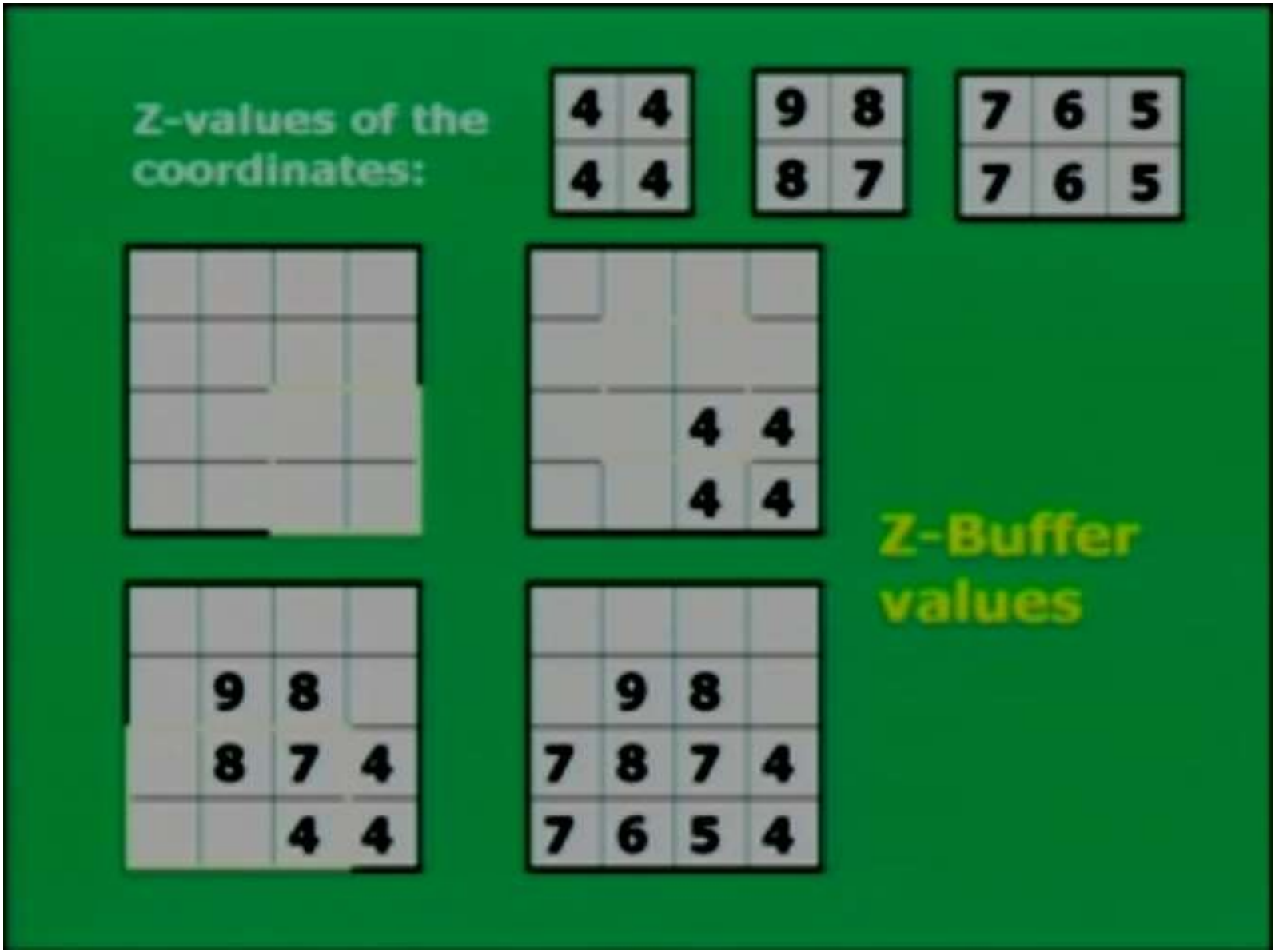
for all (x,y)
  depth(x,y) = -∞ */* watch this change */*
  refresh(x,y) = I_B

for each polygon P
  for each position (x,y) on polygon P
  calculate depth z

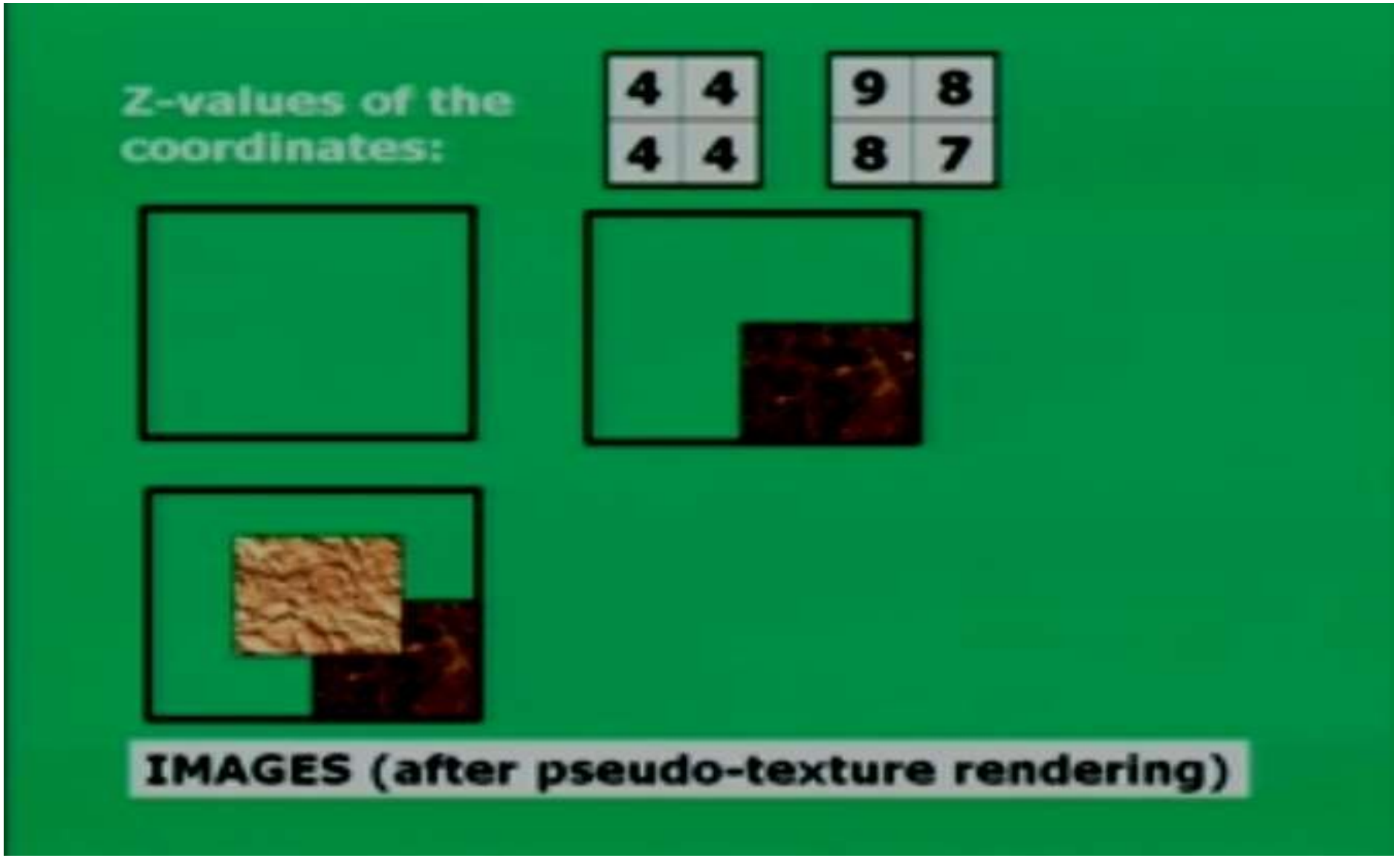
  if z > depth(x,y) then
    1. depth(x,y) = z
    2. refresh(x,y) = I_p(x,y)

Where
I_B = is the value for background intensity
I_p(x,y) = projected intensity value for the surface at pixel position (x,y)

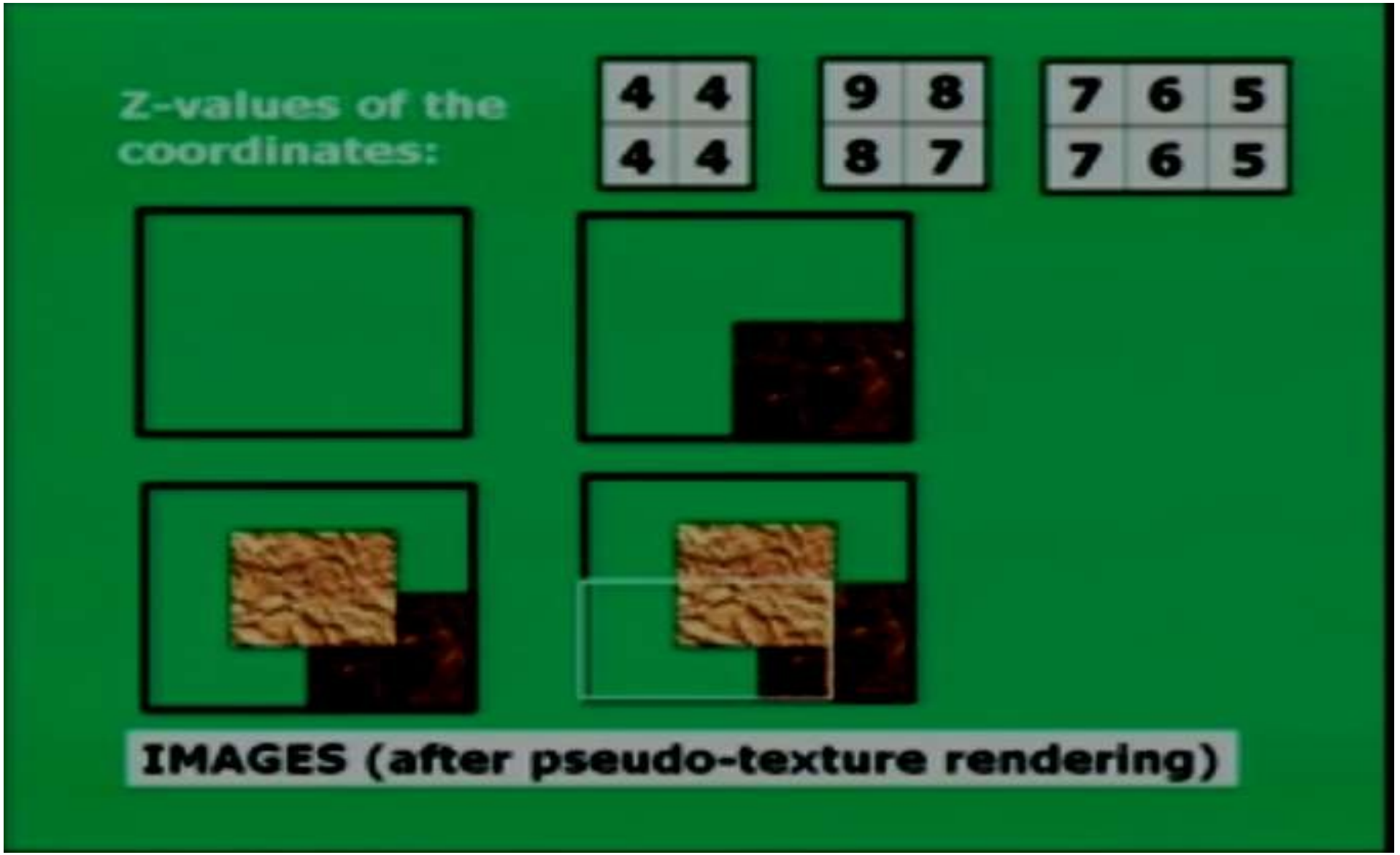
Example on Z-Buffer Algorithm (Viewing Direction from positive to Negative)



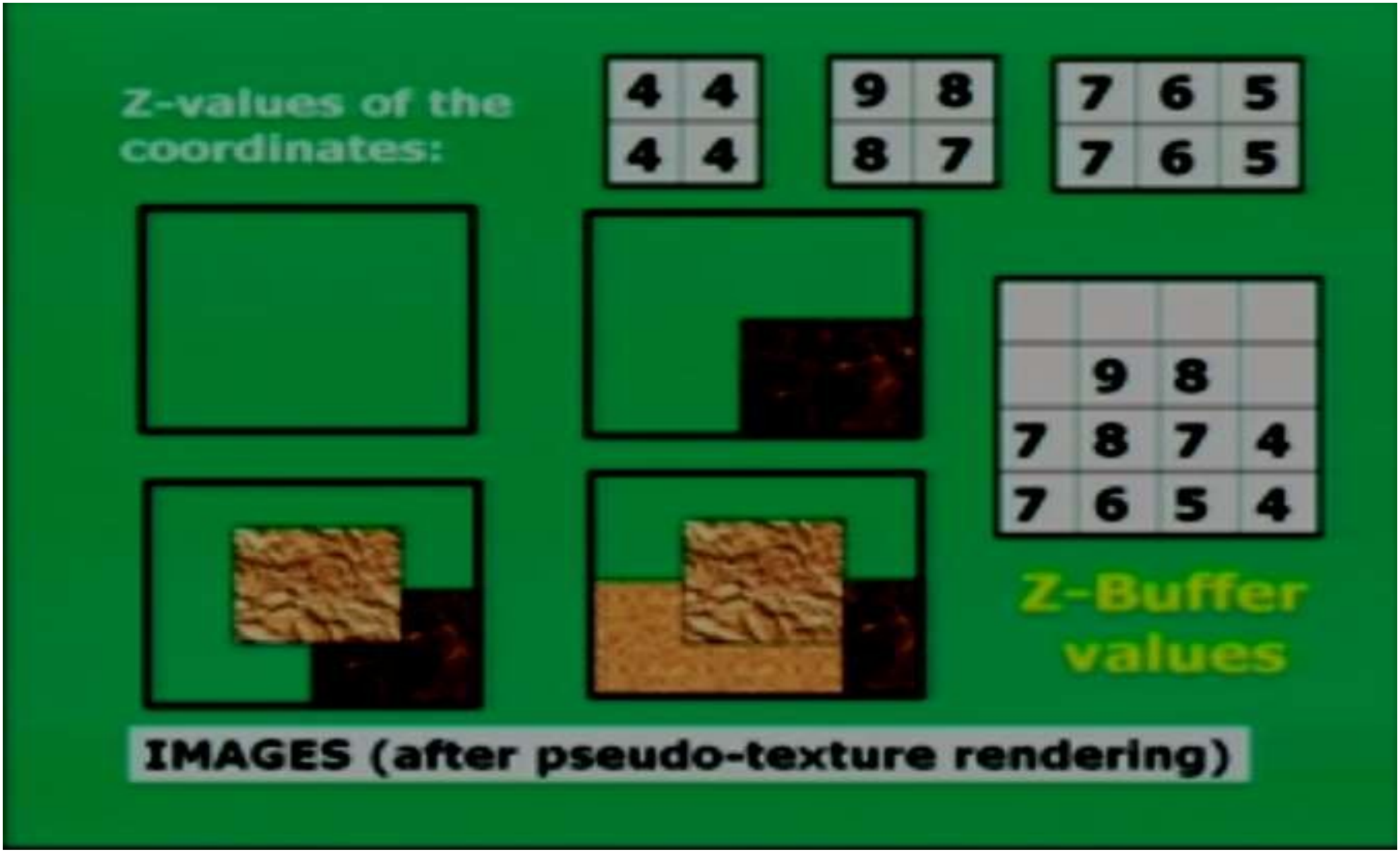
Z-buffer Algorithm



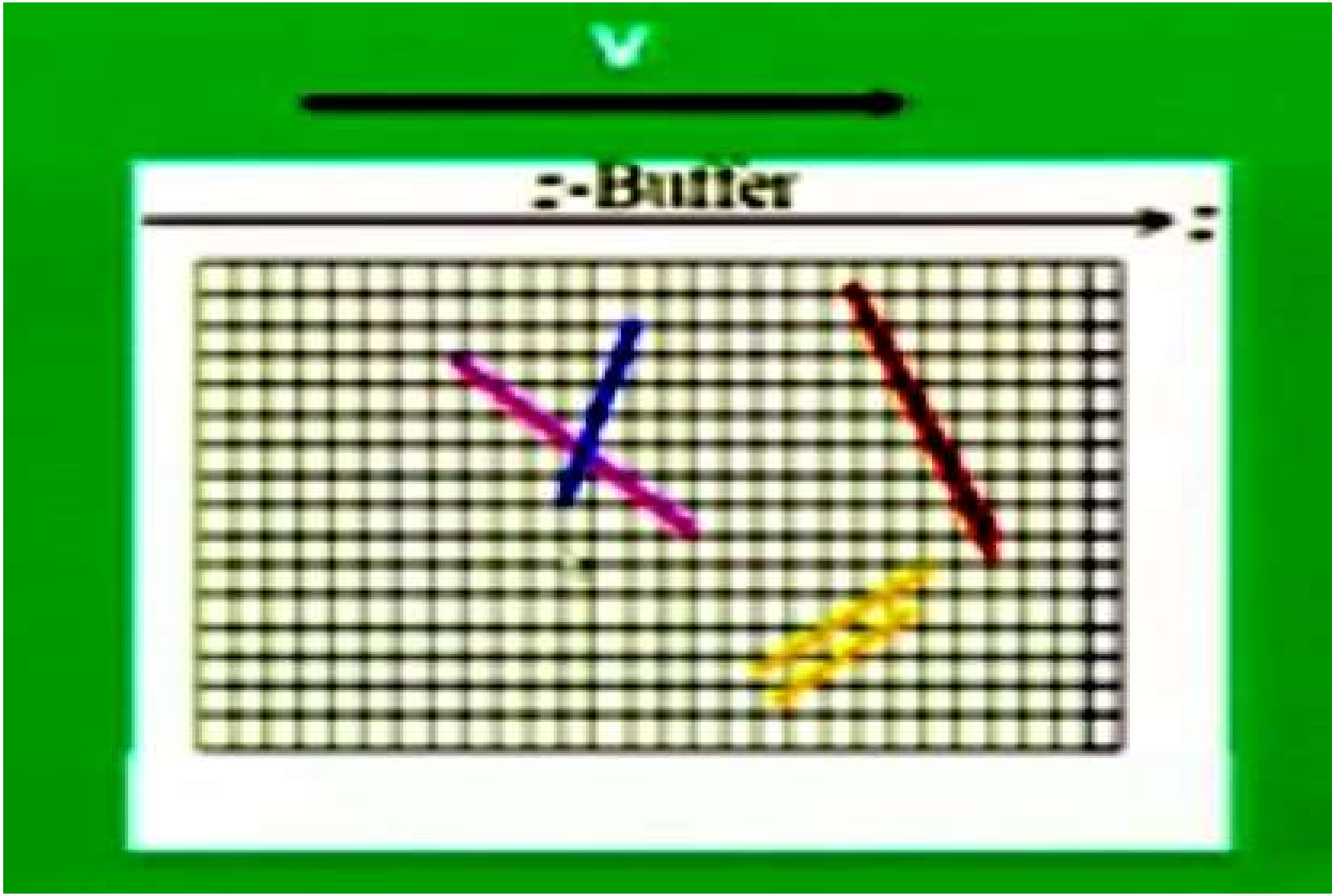
Z-buffer Algorithm

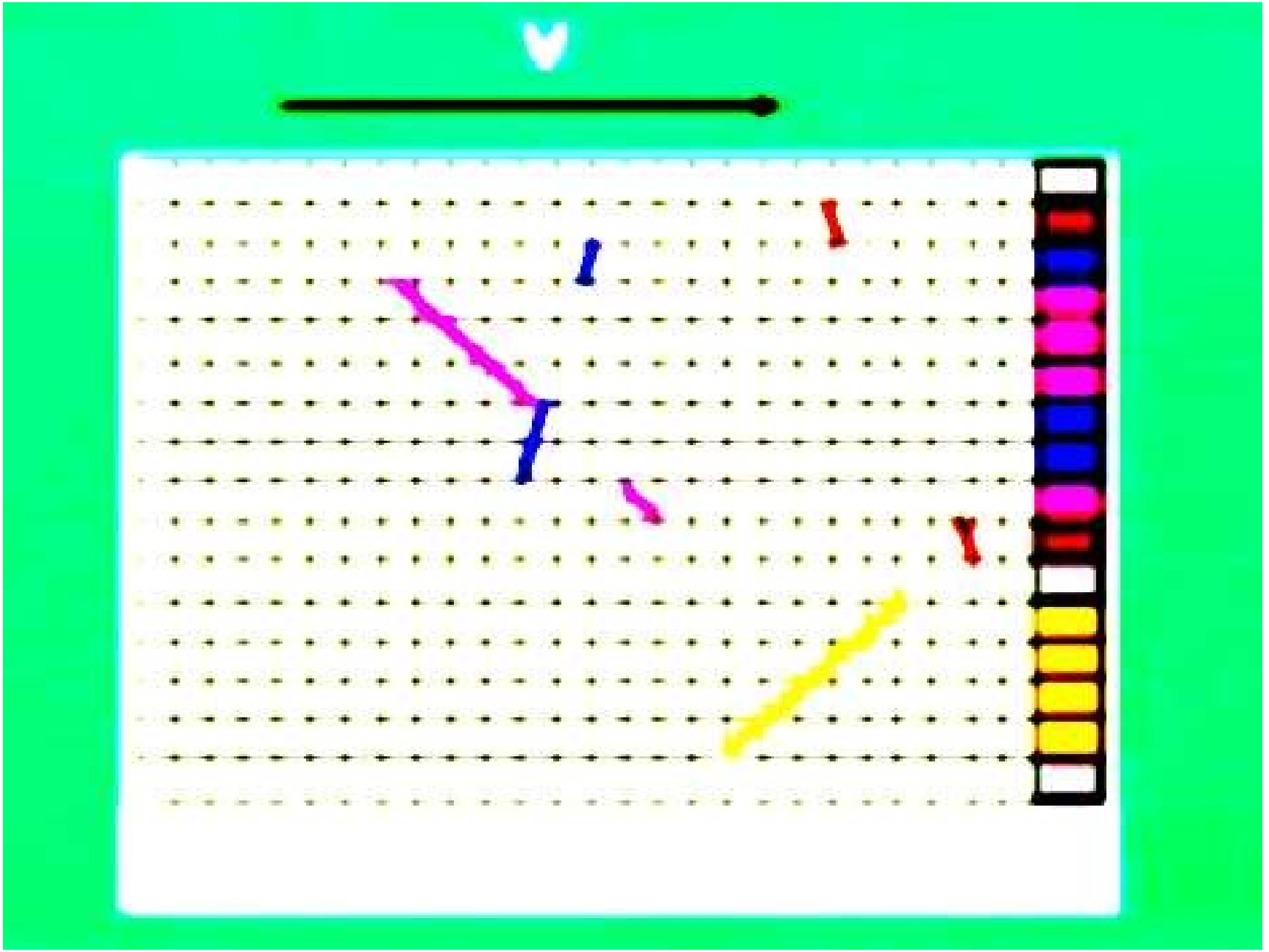


Z-buffer Algorithm



Z-Buffer Algorithm (Refresh buffer and Depth buffer Together)





Limitation of Z-buffer Algorithm

- This algorithm do not work on transparent surface, this is only for opaque surface.
- Hence here no method is available to compare the Z values of the surfaces which are transparent.
- For working on transparent surface another method is available which is called A-buffer Algorithm.
- Extra memory is required to store depth value.

BSP (Binary Space Partitioning) Tree.

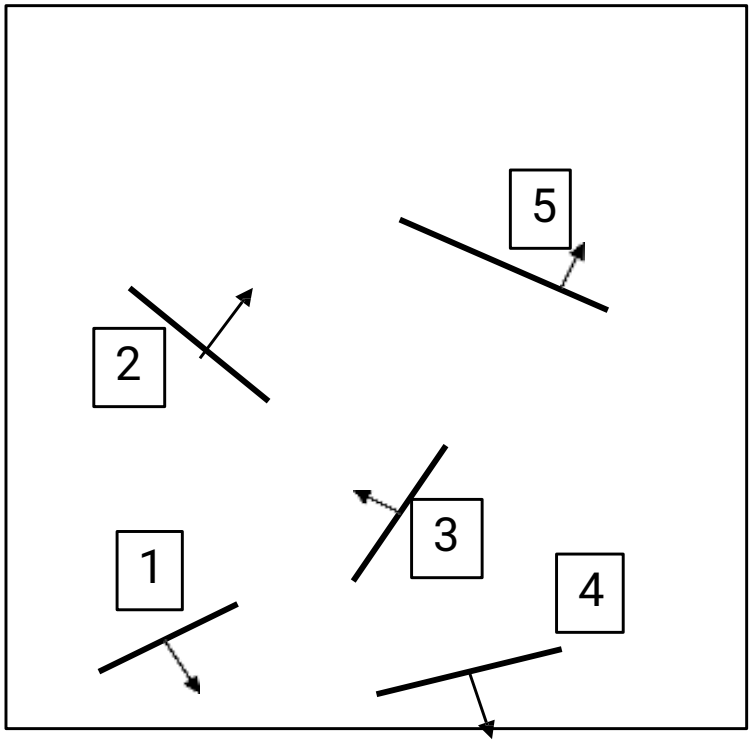
The idea of this algorithm is to sort the polygon to display in back to front order or from front to back order.

Method of BSP tree algorithm

- Choose any polygon and compare all the others to it.
- Split the surface in to two, those in front of the selected polygon and those who are in the back of the polygon.
- Split any polygon lying on both sides.
- This process is repetitive until polygon is not sorted.
- The result of this sorting can be visualize in the form of binary tree.

- In the resultant binary tree one subtree shows the polygon who are in front side and other subtree shows the polygon who are in the back.

- To get the final result traverse final tree by inorder traversal method.



View of scene from above

BSP Tree.

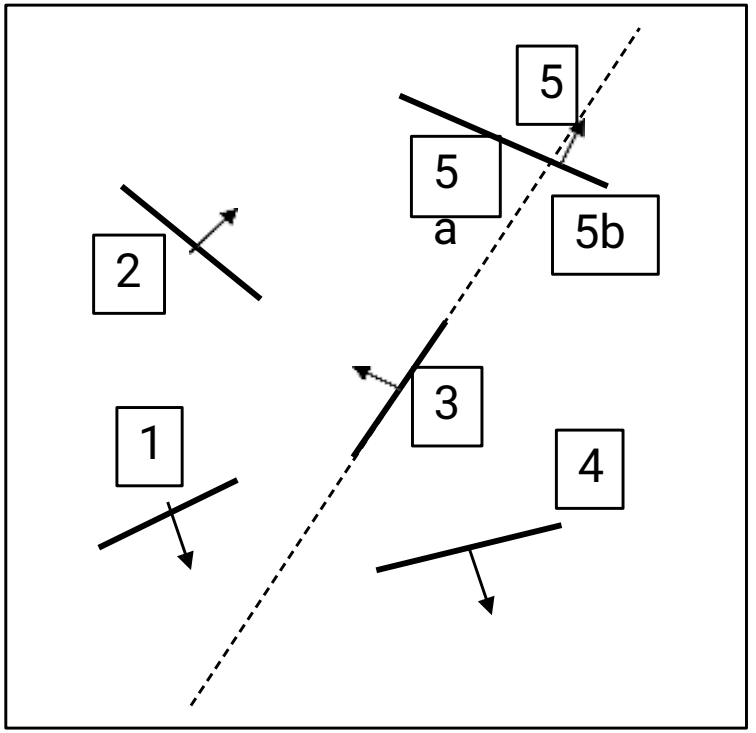
- Choose polygon arbitrarily

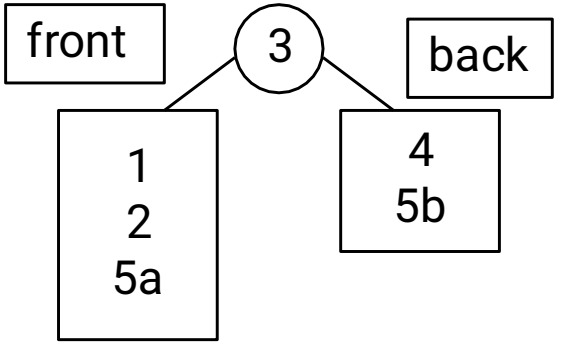
- Divide scene into front (relative to normal) and back half-spaces.

- Split any polygon lying on both sides.

- Choose a polygon from each side – split scene again.

- Recursively divide each side until each node contains only 1 polygon.





BSP Tree.

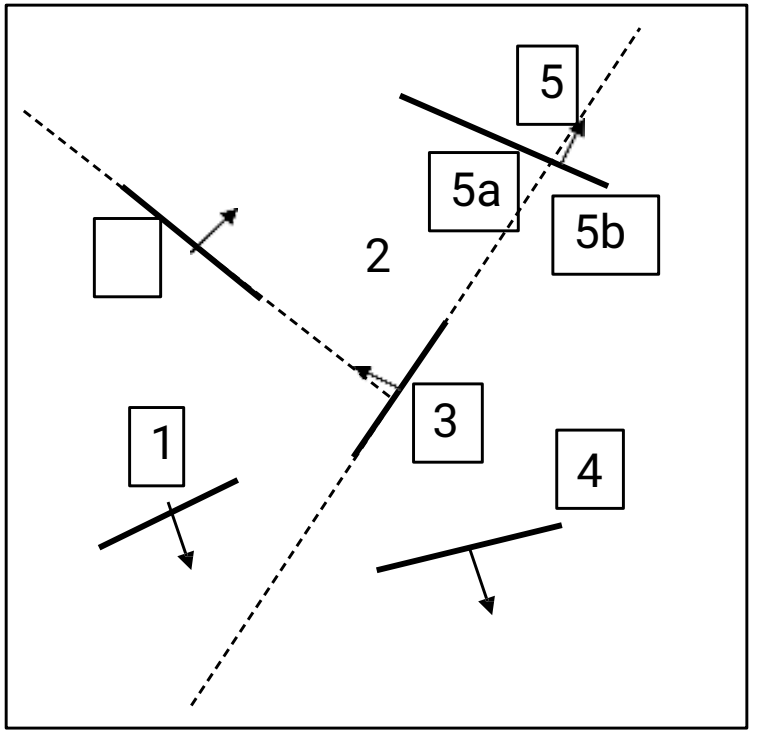
• Choose polygon arbitrarily

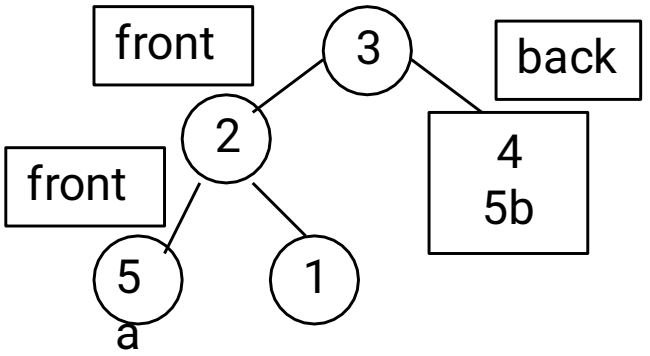
•Divide scene into front (relative to normal) and back half-spaces.

• Split any polygon lying on both sides.

•Choose a polygon from each side – split scene again.

•Recursively divide each side until each node contains only 1 polygon.





BSP Tree.

- Choose polygon arbitrarily

- Divide scene into front (relative to normal) and back half-spaces.

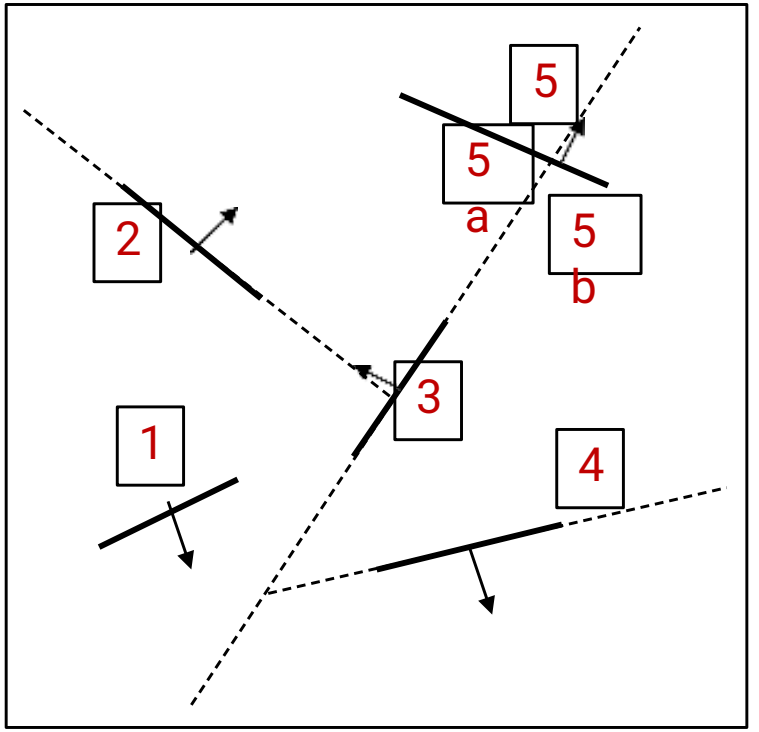
- Split any polygon lying on both sides.

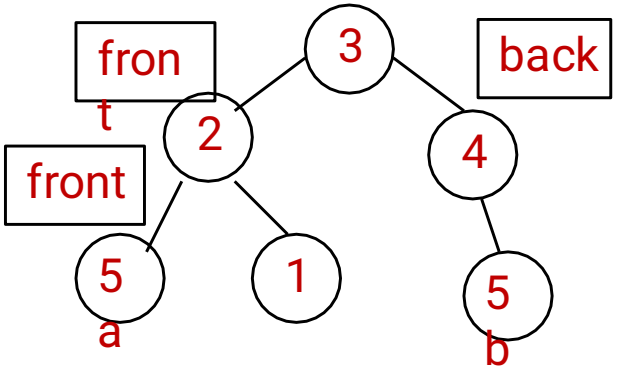
- Choose a polygon from each side – split scene again.

- Recursively divide each side until each node contains only 1 polygon.

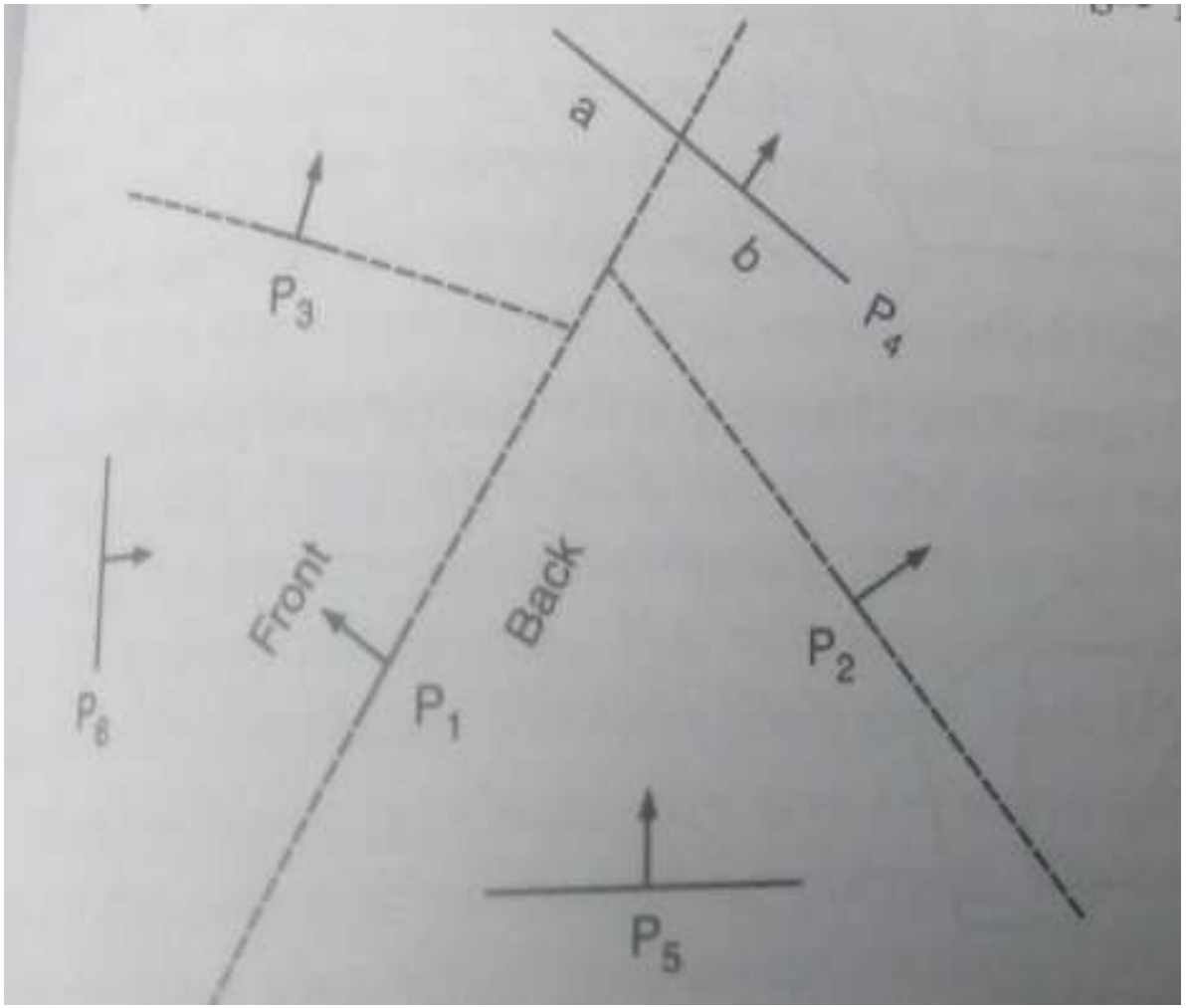
- **Result: In-order traversal of this tree is:**

  5a , 2 , 1 , 3 ,4, 5b





Question?



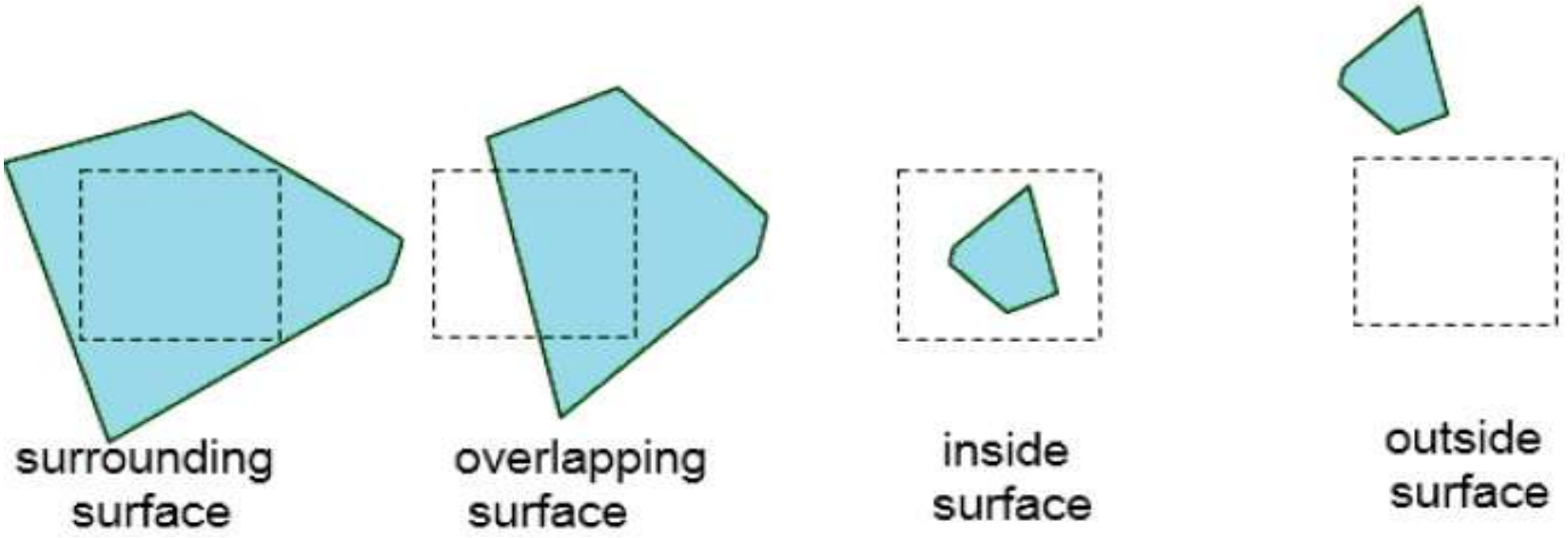
Find out front polygon and back polygon for polygon: P1

Area-Subdivision Method

- The area-subdivision method takes advantage by locating those view areas that represent part of a single surface. Divide the total viewing area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.
- Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step. There are four possible relationships that a surface can have with a specified area boundary.

1. **Surrounding surface** − One that completely encloses the area.
2. **Overlapping surface** − One that is partly inside and partly outside the area.
3. **Inside surface** − One that is completely inside the area.
4. **Outside surface** − One that is completely outside the area.

Area-Subdivision Method



The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true −

- All surfaces are outside surfaces with respect to the area.
- Only one inside, overlapping or surrounding surface is in the area.
- A surrounding surface obscures all other surfaces within the area boundaries.

Back-Face Detection

A fast and simple object-space method for identifying the back faces of a polyhedron is based on the "inside-outside" tests. A point x,y,z is " inside" a polygon surface with plane parameters A, B, C, and D, if When an inside point is along the line of sight to the surface, the polygon must be a back-face

We are inside that face and cannot see the front of it from our viewing position.

We can simplify this test by considering the normal vector N 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

$$V.N > 0$$

Furthermore, if object descriptions are converted to projection coordinates and your viewing direction is parallel to the viewing z-axis, then −

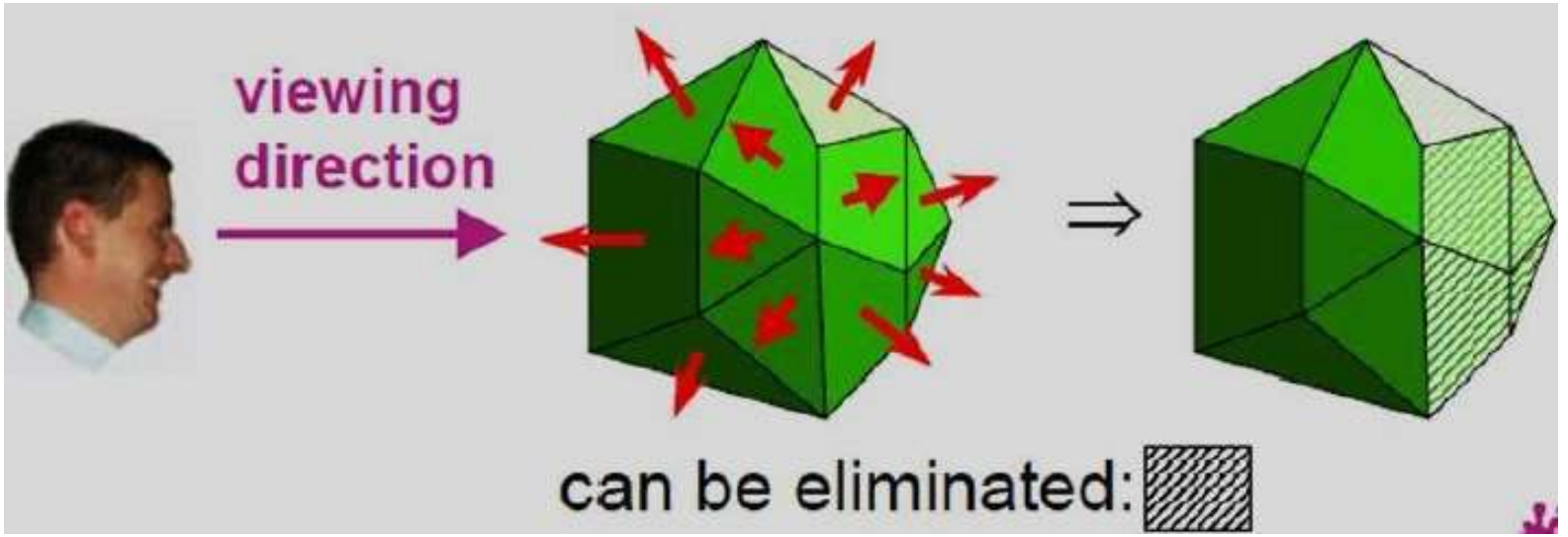
$$V = (0, 0, V_z) \text{ and } V.N = V_Z C$$

So that we only need to consider the sign of C the component of the normal vector **N**.

Back-Face Detection

Similar methods can be used in packages that employ a left-handed viewing system. In these packages, plane parameters A, B, C and D can be calculated from polygon vertex coordinates specified in a clockwise direction unlike the counter clockwise direction used in a right−handed system.

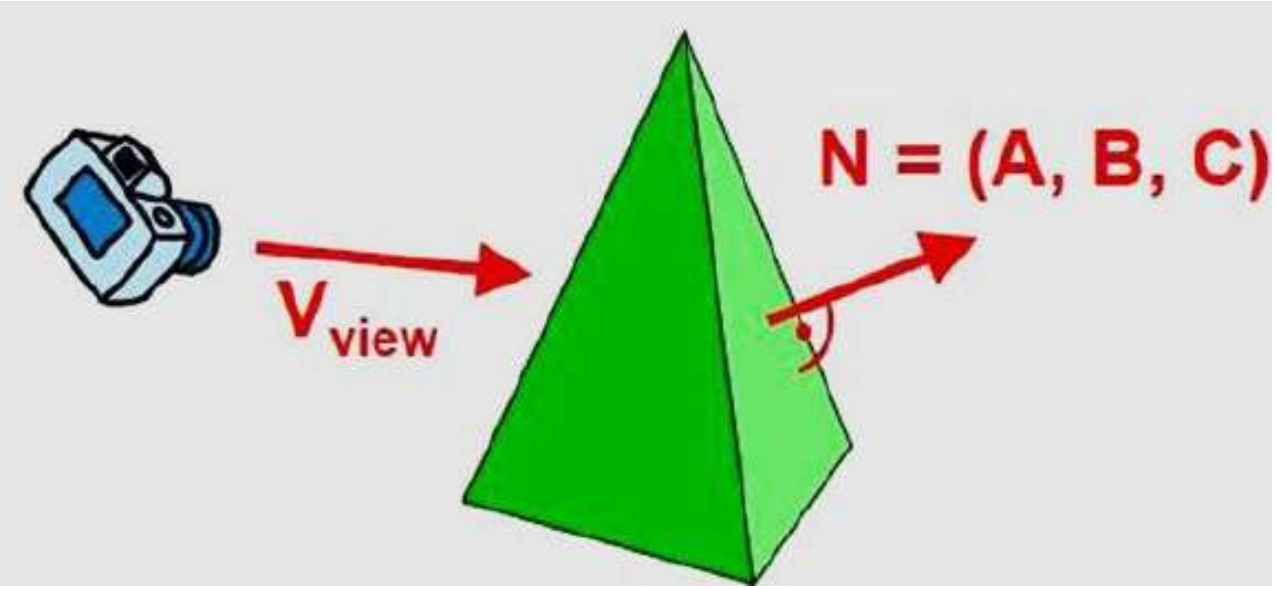
Also, back faces have normal vectors that point away from the viewing position and are identified by C >= 0 when the viewing direction is along the positive Zv axis. By examining parameter C for the different planes defining an object, we can immediately identify all the back faces.



Back-Face Detection

In a right-handed viewing system with viewing direction along the negative ZV axis, the polygon is a back face if C < 0. Also, we cannot see any face whose normal has Z component C = 0, since your viewing direction is towards that polygon. Thus, in general, we can label any polygon as a back face if its normal vector has a z component value −

C <= 0



Scan-Line Method

- Unlike Z-buffer, scan-line method is also used to detect visible surface.
- In this algorithm we maintain depth information for each scan line.
- We do not compare the Z value where portion of surfaces are not overlapped but in case of overlapping surface we need to compare the Z value same as Z buffer algorithm.

In this algorithm we have to maintain following things:

- Build table of edges of all polygons in scene.
- Maintain active-edge-table as we visit each scan-line in scene. AET now contains edges for all polygons at that scan line. Just Like Scan line algorithm (Polygon Fill).
- Must maintain flag for each surface to determine whether pixel on scan-line is inside that surface.

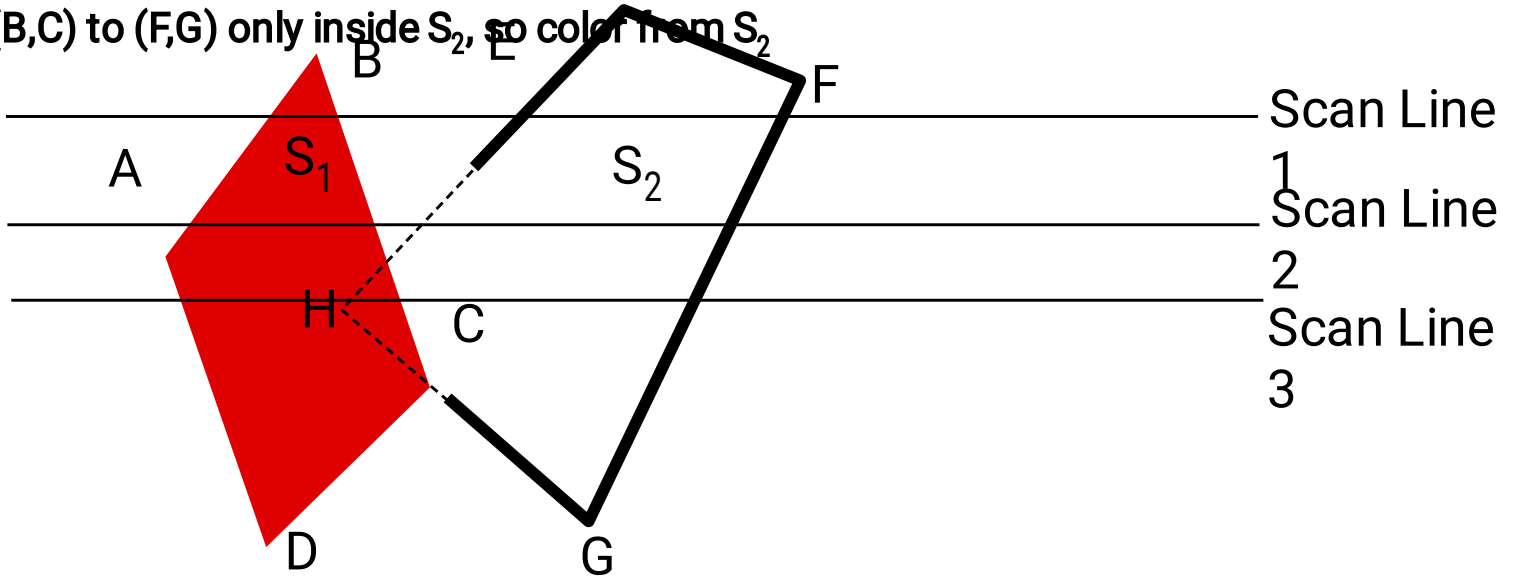
Scan-Line Method

The Edge Table − It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

The Polygon Table − It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.

Scan-Line Method Basic Example

- Scan Line 1:
  - (A,B) to (B,C) only inside S_1, so color from S_1 (Flag1 will on and 2 will off)
  - (E,H) to (F,G) only inside S_2, so color from S_2 (Flag 2 will on 1 will off)
- Scan Line 3:
  - **(A,D) to (E,H) only inside S_1, so color from S_1**
  - **(E,H) to (B,C) inside S_1 and S_2 , so compute & test depth. In this example we color from S_1**
  - **(B,C) to (F,G) only inside S_2, so color from S_2**



Scan-Line Method Basic Example

To facilitate the search for surfaces crossing a given scan-line, an active list of edges is formed. 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.

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 and at the right, the flag is turned off. You only need to perform depth calculations when multiple surfaces have their flags turned on at a certain scan-line position.

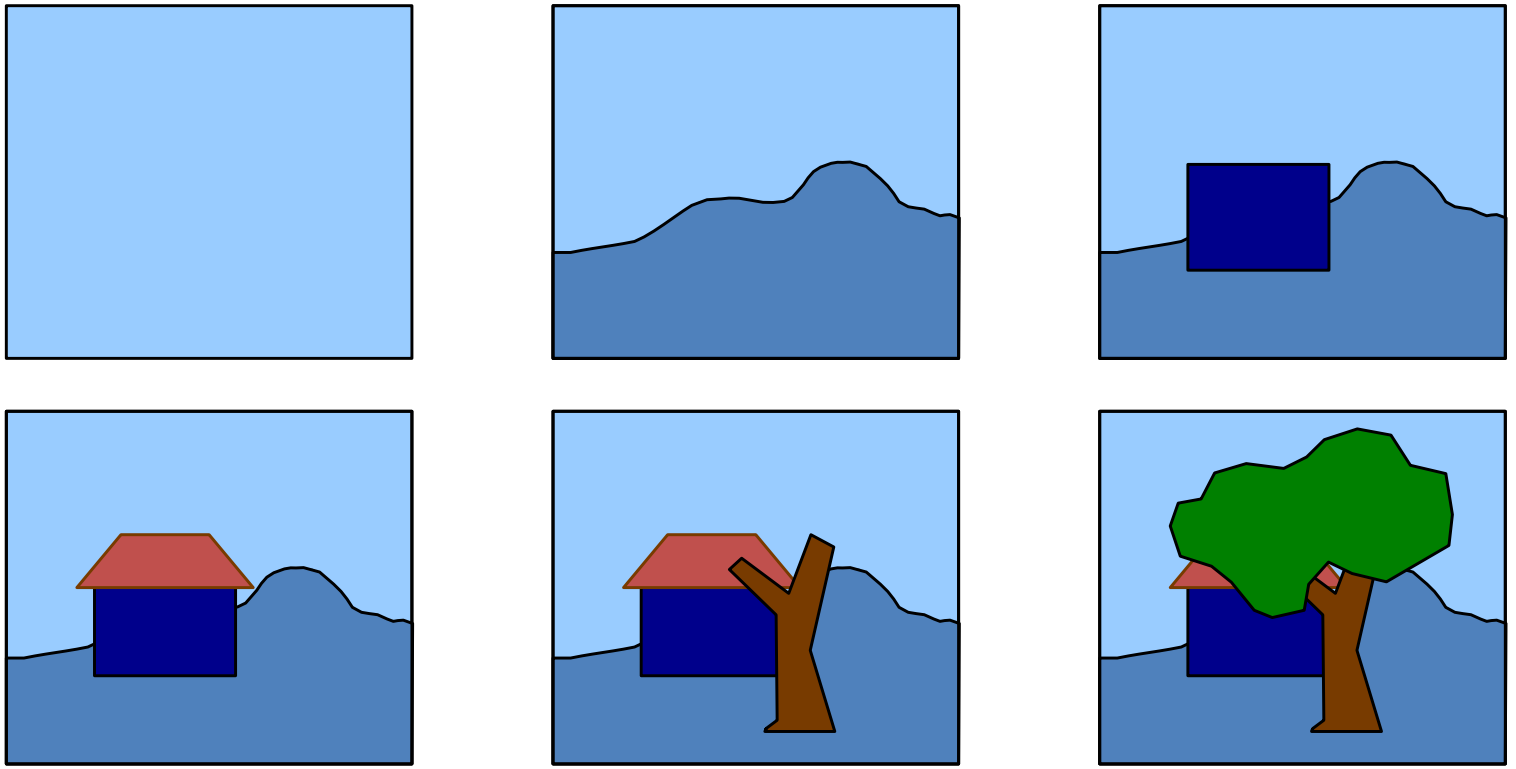
Painter's or Depth sorting method

- In this algorithm polygons are sorting according in ascending order.
- In this algorithm we maintain "behind counter". This behind counter is used to record number of polygons behind any polygon.
- Polygon which has "behind counter" Zero are those polygon that have no other polygon behind them.
- After painting the polygon we decrease behind counter of that polygon by one.
- This algorithm terminates when "Behind counter" of all the polygon becomes -1.

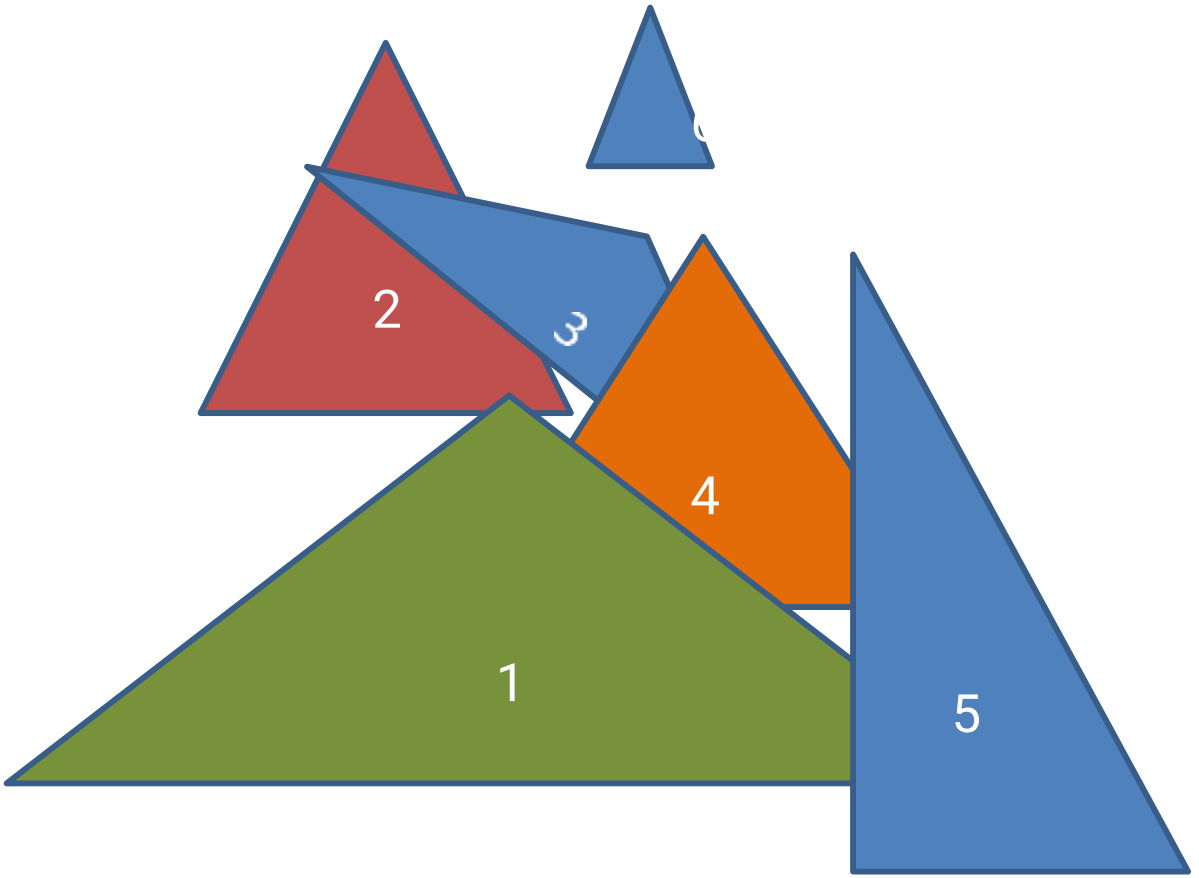
Painter's Algorithm

Draw surfaces from back (farthest away) to front (closest):

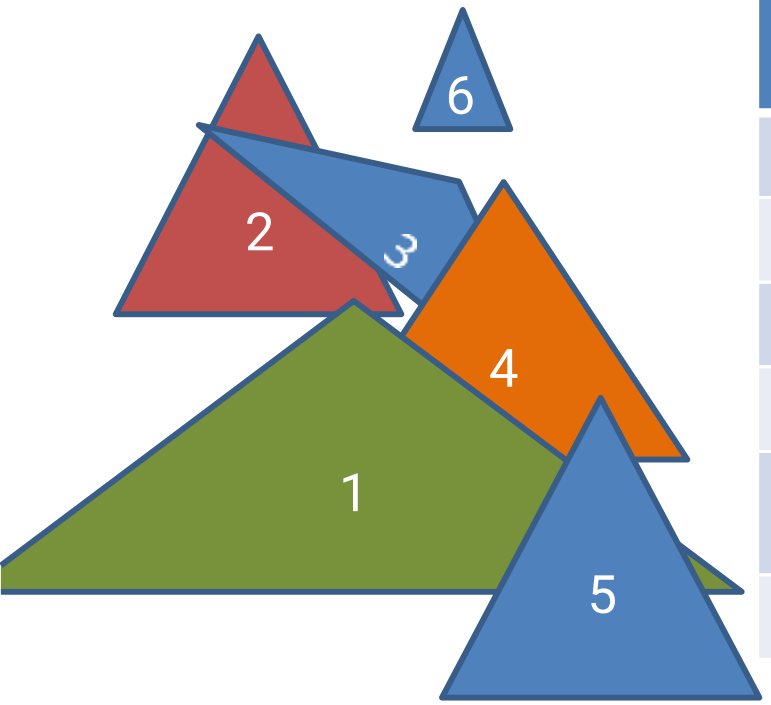
- Sort surfaces/polygons by their depth (*z* value)
- Draw objects in order (farthest to closest)
- Closer objects paint over the top of farther away objects



Painter's Algorithm or Depth sorting Method



Depth Order Table



Triangle	Behind counter	Triangle name Which is overlapped	List of triangle in front
1	2	T2 , T4	5
2	0	NULL	1,3
3	1	T2	4
4	1	T3	1,5
5	2	T4,T1	0
6	0	NULL	0

Flow of Algorithm: Step -1

Triangle	Behind counter	Triangle overlapped	List of triangle in front
1	2	T2 , T4	5
2	0	NULL	1,3
3	1	T2	4
4	1	T3	1,5
5	2	T4,T1	0
6	0	NULL	0

Triangle	Behind counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted triangle
1	2	T2 , T4	5	1	
2	0	NULL	1,3	-1	
3	1	T2	4	0	T2, T6
4	1	T3	1,5	1	
5	2	T4,T1	0	2	
6	0	NULL	0	-1	

Triangle	Behind counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted triangle
1	2	T2 , T4	5	1	
2	0	NULL	1,3	-1	
3	1	T2	4	0	T2, T6
4	1	T3	1,5	1	
5	2	T4,T1	0	2	
6	0	NULL	0	-1	

Triangle	List of new Behind Counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted Triangle
1	1	T2 , T4	5	1	
2	-1	NULL	1,3	-1	
3	0	T2	4	-1	T2,T6,T3
4	1	T3	1,5	0	
5	2	T4,T1	0	2	
6	-1	NULL	0	-1	

Triangle	List of new Behind Counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted Triangle
1	1	T2 , T4	5	1	
2	-1	NULL	1,3	-1	
3	0	T2	4	-1	T2,T6,T3
4	1	T3	1,5	0	
5	2	T4,T1	0	2	
6	-1	NULL	0	-1	

Triangle	List of new Behind Counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted Triangle
1	1	T2 , T4	5	0	
2	-1	NULL	1,3	-1	
3	-1	T2	4	-1	T2,T6,T3, T4
4	0	T3	1,5	-1	
5	2	T4,T1	0	1	
6	-1	NULL	0	-1	

Triangle	List of new Behind Counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted Triangle
1	1	T2 , T4	5	0	
2	-1	NULL	1,3	-1	
3	-1	T2	4	-1	T2,T6,T3,T4
4	0	T3	1,5	-1	
5	2	T4,T1	0	1	
6	-1	NULL	0	-1	

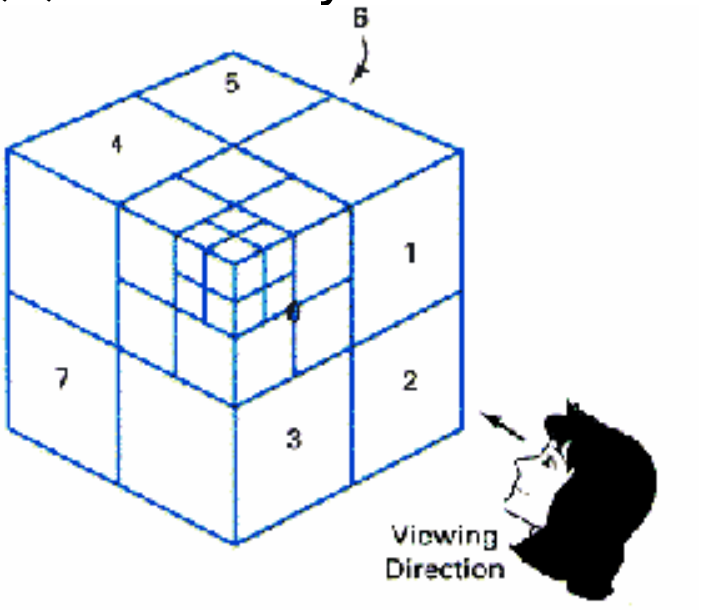
Triangle	List of new Behind Counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted Triangle
1	0	T2 , T4	5	-1	
2	-1	NULL	1,3	-1	
3	-1	T2	4	-1	T2,T6,T3,T4, T1
4	-1	T3	1,5	-1	
5	1	T4,T1	0	0	
6	-1	NULL	0	-1	

Triangle	List of new Behind Counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted Triangle
1	1	T2 , T4	5	0	
2	-1	NULL	1,3	-1	
3	-1	T2	4	-1	T2,T6,T3, T4
4	0	T3	1,5	-1	
5	2	T4,T1	0	1	
6	-1	NULL	0	-1	

Triangle	List of new Behind Counter	Triangle overlapped	List of triangle in front	List of new Behind Counter	Painted Triangle
1	-1	T2 , T4	5	-1	
2	-1	NULL	1,3	-1	
3	-1	T2	4	-1	T2,T6,T3, T4, T1, T5
4	-1	T3	1,5	-1	
5	0	T4,T1	0	-1	
6	-1	NULL	0	-1	

Octree Methods

In these methods, octree nodes are projected onto the viewing surface
in a front-to-back order. 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 suboctants of octant 0 a for the four back suboctants.



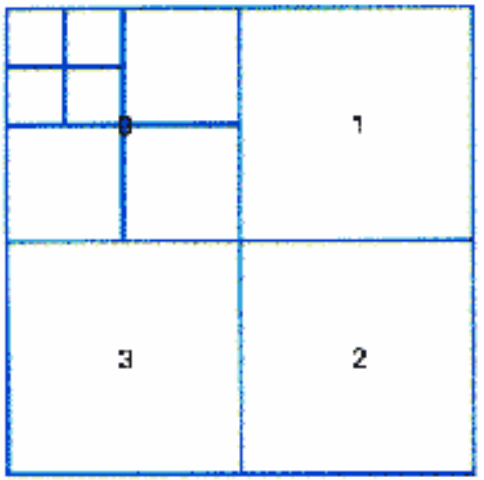
Octants in Space

Octree Methods

When a colour 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.

In most cases, both a front and a back octant must be considered in
determining the correct color values for a quadrant.
But

- 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 has to be subdivided and the sub-octants are handled recursively.



Quadrants for the View Plane

References

1. Schaum's Outline of Computer Graphics

2. Computer Graphics by Donald Hearn and M. Pauline Baker