Visible surface Detection.

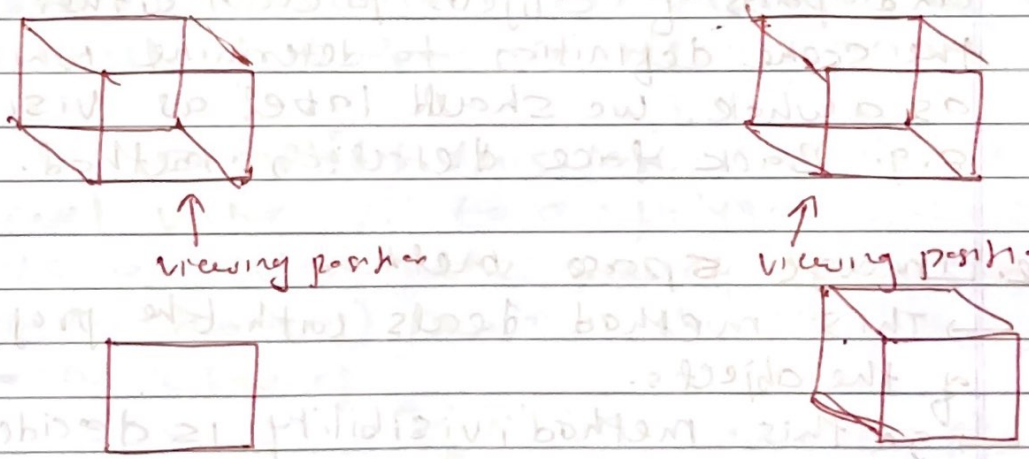
Chapter - 6

Date
Page 1

* Whats visible surface detection?

→ visible surface Detection and Hidden surface removal both are the same terminology.

→ visible surface detection is to identify those parts of the scene that are visible from a choosen viewing position.



→ In above both scenarios, viewing position are different. So in both cases ~~where~~ only visible portions ~~are~~ of original image needs to be taken. And rest of the image parts which is not visible needs to be discarded.

→ There are many approaches or algorithms for efficient identification of visible objects but all are differing according as memory requirement, processing time, special application etc.

→ There are two approaches/Algorithm for visible surface detection.

1. Object space method (OSM)
   → It is implemented on physical coordinate system.
   → It deals with object definition.
   → An object space method 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.
   E.g. Back face detection method.

2. Image space method
   → This method deals with the projected images of the objects.
   → In this method, visibility is decided point by point at each pixel position on the projection plane.
   → Most visible surface ~~meth~~ algorithms use image space methods.
   → Eg. Depth-buffer method, Scan line method etc.

* Back-face Detection method
  → It is a fast and simple object-space method for identifying the back faces of a polyhedron (a solid in 3D with flat polygonal faces, straight edges and sharp corners or vertices) and it is based on the inside-outside tests.

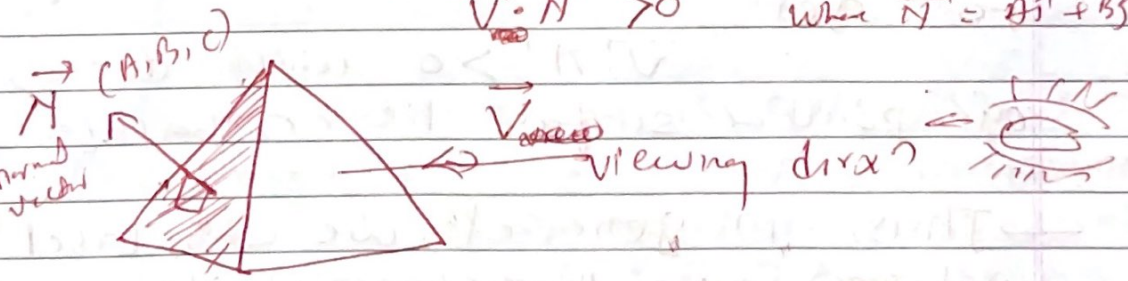
→ A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C & D if

$$Ax + By + Cz + D < 0$$

→ 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 & can't see the front of it from our viewing position).

→ To simplify this test, consider the normal vector $\vec{N}$ to a polygon surface. If $\vec{V}$ is a vector in the viewing direction from the eye (or camera) position, then this polygon is a back face if

$$\vec{V} \cdot \vec{N} > 0 \qquad \text{where } \vec{N} = A\vec{i} + B\vec{j} + C\vec{k}$$



$\vec{V} \cdot \vec{N} > 0$ (since both are in same direction)

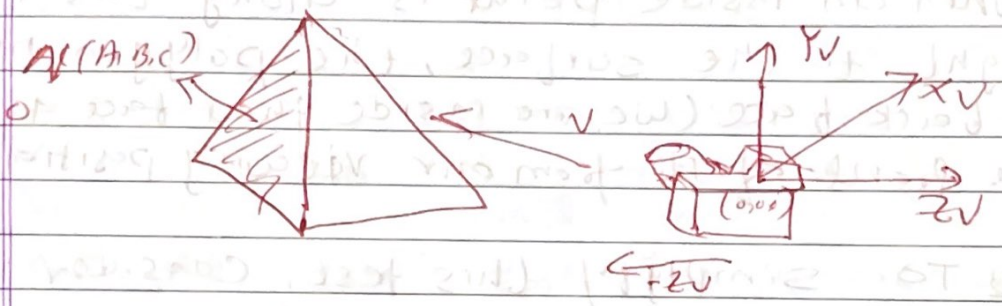
→ If object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing z_v axis, then

$$\vec{V} = (0, 0, z_v) \text{ and}$$

$$\vec{V} \cdot \vec{N} = (0, 0, V_z) \cdot (A, B, C)$$
$$= V_z C$$

→ So that we only need to consider the sign of C, the z component of normal vector $\vec{N}$.

→ Consider Right hand viewing system



→ Here $Z_v = -ve$

→ V moves from +ve to -ve. $C = -ve$

So $Z_v = -ve$

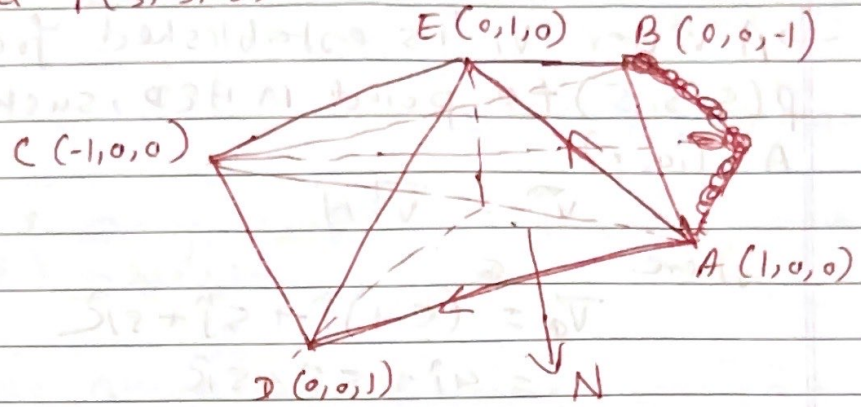
→ C must be -ve to have -ve value to get

$\vec{V}.\vec{N} > 0$ which decides the our pointed surface lie on backface or not.

→ Thus, in general, we can label any polygon as a backface if its normal vector has a z component value

$C \leq 0$.

Q. Find the visibility for the surface AED where observer at P(5,5,5).



→ Here, given a plane and it is defined by three points on the plane A, B & C in 3D space and trying to find a normal vector to that plane.

→ How to find vector normal / perpendicular to the plane?

→ If you define two vectors on the plane, the result of their cross product is always another vector which is perpendicular to the plane which first two vectors lies.

→ find the normal vector N for AED surface (always take anti-clockwise direction convention).

i.e $\overrightarrow{AE} \times \overrightarrow{AD}$ (not $\overrightarrow{AD} \times \overrightarrow{AE}$).

Where

$\overrightarrow{AE} = E(0,1,0) - A(1,0,0) = (-1,1,0) = -\hat{i} + \hat{j}$

$\overrightarrow{AD} = D(0,0,1) - A(1,0,0) = (-1,0,1) = -\hat{i} + \hat{k}$

$\therefore N = \overrightarrow{AE} \times \overrightarrow{AD} = (-\hat{i} + \hat{j}) \times (-\hat{i} + \hat{k})$

$= (-\hat{i} \times -\hat{i}) + (-\hat{i} \times \hat{k}) + (\hat{j} \times -\hat{i}) + (\hat{j} \times \hat{k})$

$\therefore \vec{N} = \hat{i}+\hat{j}+\hat{k}$

→ A vector V_i is established from the viewpoint $p(5,5,5)$ to point in AED, such as point $A(1,0,0)$

$$\vec{V_1} = \vec{V} \cdot \vec{N}$$

where

$$\vec{V_0} = (5-1)\hat{i} + 5\hat{j} + 5\hat{k}$$
$$= 4\hat{i} + 5\hat{j} + 5\hat{k}$$

→ The dot product

$$V_1 = \vec{V} \cdot \vec{N}$$
$$= (4\hat{i}+5\hat{j}+5\hat{k}) \cdot (\hat{i}+\hat{j}+\hat{k})$$
$$= 14 > 0$$ This shows that the surface is visible to the observer.

→ The polygon is backface if $C < 0$. Here $V_z C > 0$

→ Similarly, visibility operation are performed for the remaining faces with the following results.

Face	Visibility
AED	visible
BEA	visible
DCE	visible
CEB	invisible
ABCD	invisible.

* Z-Buffer Algorithm OR Depth Buffer Algorithm

→ The drawback of Backface Detection algorithm is

① It can't work on partial hidden surfaces

② It works only with non-overlapping (separate) objects.

→ Z-Buffer Algorithm removes the drawback of Back face Detection Algorithm.

→ It is an Image space method to eliminate hidden surfaces.

→ It compares surface depth at each-pixel position.

→ Z-value means distance bet^n surface and view plane.

→ It uses 2 Buffer

① Z-Buffer
→ It → It saves Z-value.
→ Z-value is saved under Z-buffer.

② Refresh Buffer
→ It saves pixel's intensity value.
→ pixel's intensity value is saved under 'Refresh Buffer'.

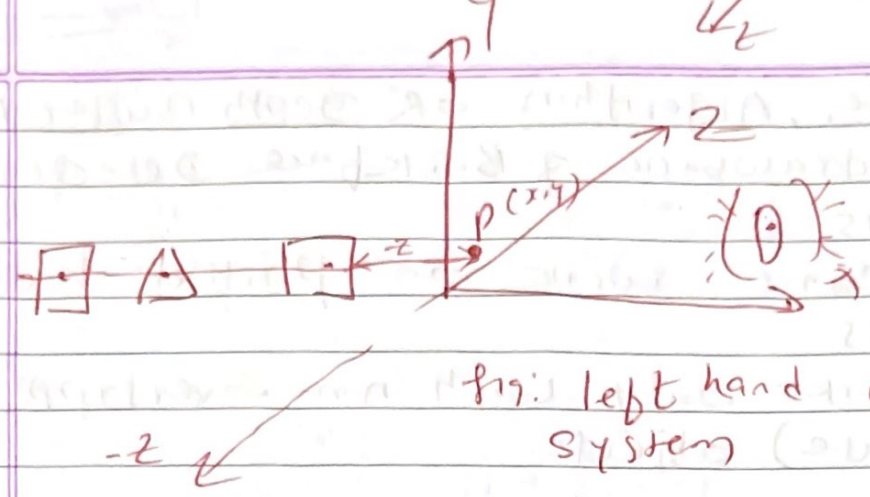
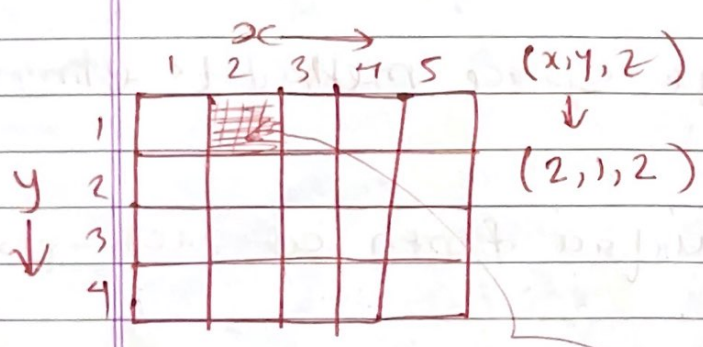


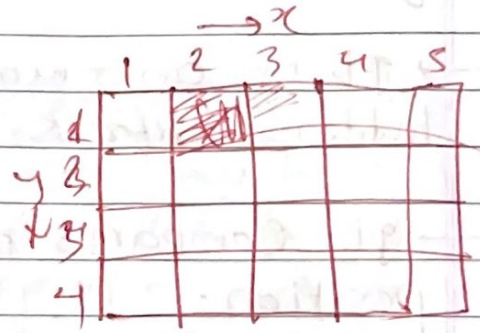
fig: left hand coordinate system

→ Z-value is depth. Depth is the distance betn Camera viewplane to object



→ Z value = 2 is stored in this buffer location

fig: Depth Buffer.



→ pixel intensity value will be stored in this buffer location

fig: Refresh buffer

→ There are two cases these depends on where we are sitting → looking is which direction.

1. Case I

→ sitting @ +Z, looking origin −Z
→ Larger Z-value = close
→ smaller Z-value = farther
→ Initialize depth & Refresh buffer

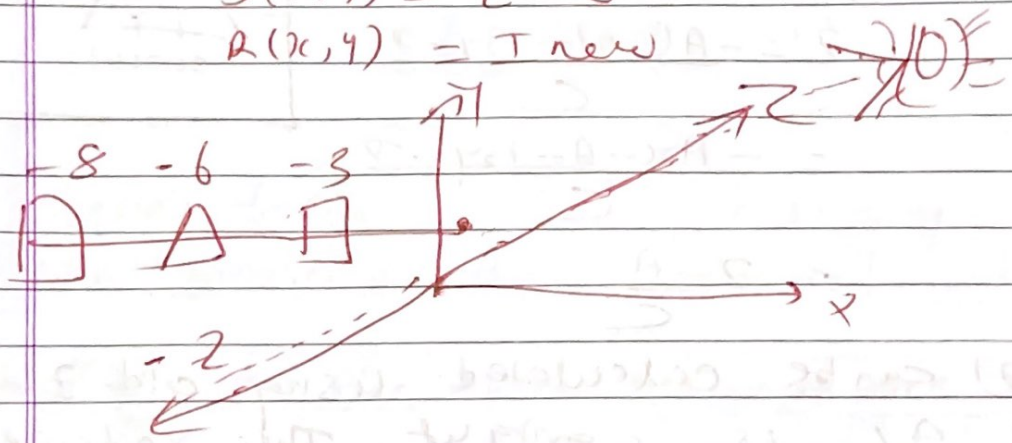
depth buffer (i) $D(x,y) = 0$

Refresh buffer $R(x,y) = I_{background}$

(ii) $Z_{new} > D(x,y)$

$D(x,y) = Z_{new}$

$R(x,y) = I_{new}$



2. Case II

→ sitting @ −Z, looking origin +Z
→ smaller Z-value - closer
→ larger Z-value - farther

(i) $D(x,y) = Z_{max}(\infty)$

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

(ii) $Z_{new} < D(x,y)$

$D(x,y) = Z_{new}$

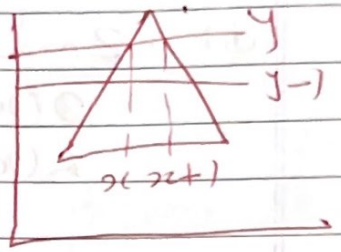
$R(x,y) = I_{new}$

* Surface Equation

→ Depth values for a surface position (x, y) can be calculated from the plane equation for each surface as --

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

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



→ for $(x+1)$

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

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

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

→ z' can be calculated using old z-value as A/C is constant. This reduces the calculation time.

* Drawbacks

→ This algorithm is also only used when there are opaque surfaces.

A -Buffer Method

* A-Buffer Algorithm

→ A-Buffer Algorithm ~~is~~ is also called Antialized, Area, Average, Accumulation Buffer Algorithm.

→ Z-Buffer Algorithm drawback is that work only with opaque surface.

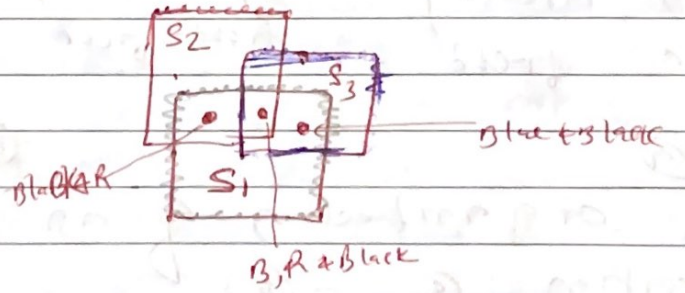
→ A-Buffer Algorithm removes the drawback of Z-Buffer Algorithm.

→ A-Buffer Algorithm maintains two fields.

2 - fields

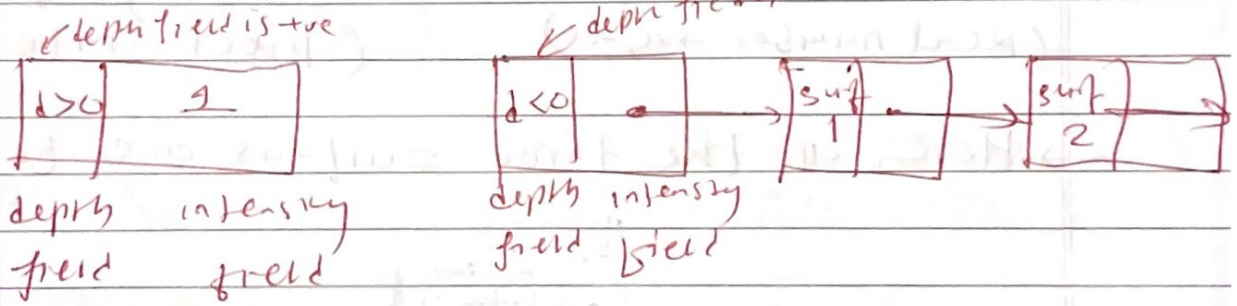
Depth field	Intensity field
(Real number +ve, -ve)	(pixel intensity)

→ Here, all the three surfaces are transport.



→ 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 RGB components of the surface color at that point.

→ If the depth field is -ve, this indicates multiple surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data. Data for each surface in the linked list includes
- RGB intensity components
- opacity parameter (% of transparency)
- Depth
- % of area coverage
- surface identifier
- other surface-rendering parameters
- pointer to next surface



(a) (b)

fig: organization of an A-buffer pixel position (a) & (b)

* Scan Line method
→ It is an image space method to identify visible surface.
→ It is an extension of the scan line algorithm for filling polygon interiors.
→ Here, the algorithm deals with more than one surfaces.
→ As each scan line is passed, it examines all polygon surfaces intersecting that line to determine which line are visible.
→ It then does the depth calculation and finds which polygon is nearest to the view plane.
→ Finally, it enters the intensity value of the nearest polygon at that position into the frame buffer/refresh buffer.

→ Different data structures or tables are set up for various surfaces which includes both an edge table and a polygon table.

① Edge table
→ It contains coordinate endpoints of each line in the scene, inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

② Surface table
→ It contains coefficient of the plane eqn. for each surface, surface intensity information, and pointers to the edge table.

③ Active edge list

→ It contains only edges that cross the current scan line, sorted in order of increasing x.

④ Surface flag

→ It indicates whether a position along a scan line is inside or outside of the surface. At the leftmost boundary of a surface, the surface flag is turned on and at the rightmost boundary it is turned off.

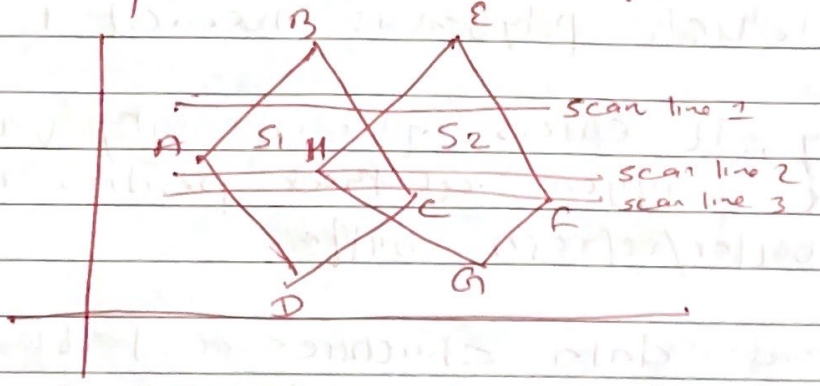


fig: Scan line crossing the projection of two surfaces S1 & S2, in the view plane.

→ Active list for scan line 1 contains information from the edge table for edges AB, BC, EH & EF.

→ For positions along this scan line between edges AB & BC only the flag for surface S1 is on.

→ So, no depth calculations are necessary and the intensity information for surface S_1 is entered

from the polygon table into the refresh buffer.
→ Both edges EH & Ef, only the flag for surface S_2 is entered into the refresh buffer, while for all other positions the intensity values are set to the background intensity.
→ Similarly, active list for scan line 2 contains the edges AB, EH, BC & Ef.
→ Bcm the edges AB & EH, only the flag for surface S_1 is set on & intensity value for S_1 is stored into refresh buffer.

→ Bcm edges EH & BC, the flags for both surfaces S_1 & S_2 are set on. for this interval depth calculation must be done and depending upon which surface is closer to the viewplane, its intensity value is stored into the refresh buffer.
→ Bcm. edges BC & Ef, the flag for surface S_1 goes off & the intensity value for surface S_2 is stored into the refresh buffer.

→ for scan line 3, it has same active edge list. so scan line 3 can take advantage of coherence. No need to ~~calculate~~ do the depth calculate due to coherence property.
↳ it takes advantages of regularity in the scene.

* Advantages:
① Take advantages of coherence principle.
② Any number of overlapping surfaces can be processed

* Area subdivision (Warnock) Algorithm

→ This is a divide & conquers approach. It is also known as area subdivision method.

→ Image-space method (comparing pixel of projected image).

→ Uses area coherence property (to decrease its calculation).

→ Strategy :- Divide & conquer

→ This method is applied to successively divide the total viewing area smaller & smaller rectangles and each small area is the projection of part of a single visible surface or no surface at all.

→ To implement this method, we need to establish test

- we apply the tests to determine whether we should subdivide the total area into smaller rectangles.

- Before apply subdivision, we apply test
- After performing test, if the test indicates the view is sufficiently complete, we subdivide it.
- Next, we apply the test to each of the smaller areas, subdividing these if the test indicates the visibility of single surface is still unsated.

→ The tests for determining surface visibility within an area can be stated in terms of these four classifications.

Date
Page 5

→ We continue this process until they are reduced to the size of a single pixel!

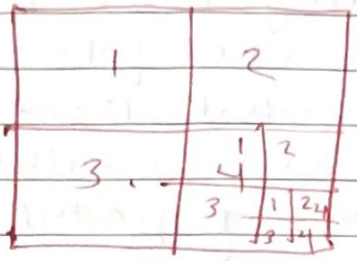
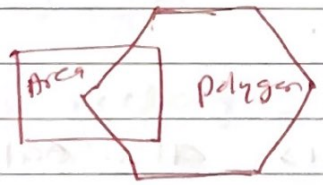


fig: Dividing a square area into equal sized quadrants at each step.

→ The procedure to classify each of the surfaces is based on the relations with the area which may be of following type

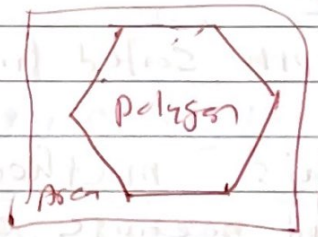
Surrounding surface

→ Area will be filled by color of surrounding polygon.



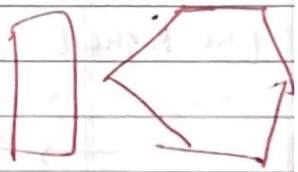
overlapping surface

→ The area covered by/intersected by polygon has the color of polygon & rest of the area has its own background color.



inside surface

→ The area covered by polygon has color of polygon & part not covered by polygon has its own background color.



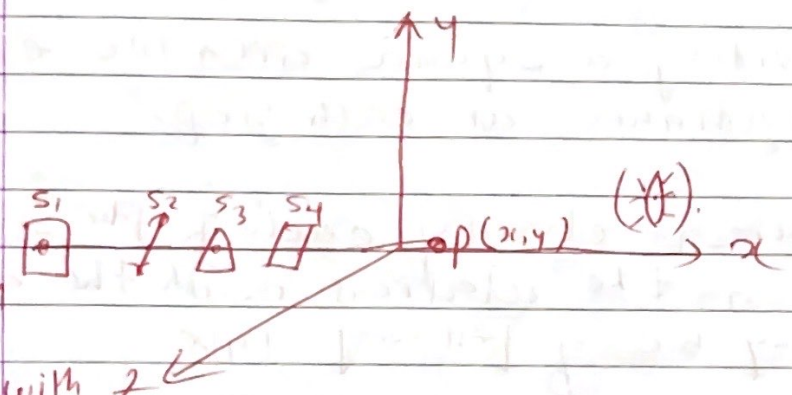
outside surface.

→ since area is outside the polygon so area has its own background color.

* Depth sorting method

→ This method for solving the hidden-surface problem by using both image-space & object-space operations in following manner.

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



- Object with the highest depth sorted first.

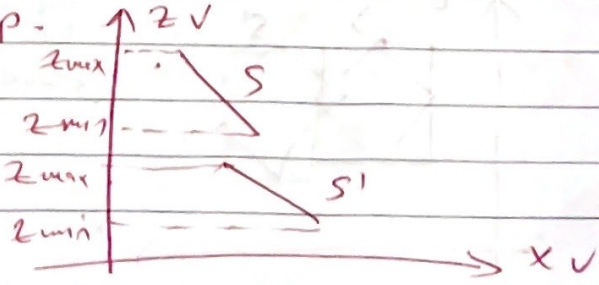
→ This method is also called painter's method because an artist first paints the background colors, then the most distant objects are added, then the nearer objects, and so forth. Each added layer of paint covers up the previous layer's i.e distant layer.

→ 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.

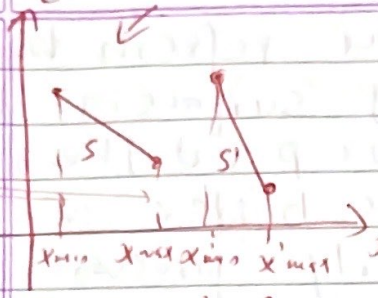
→ As long as no overlaps occurs, each surface is processed in depth order until all have been scan converted.
→ If the depth overlap is detected at any point in the list, the surfaces should be recorded.
→ Here are some tests for each surface that overlaps with S.
→ If any one of these tests is true, no reordering is necessary for that surfaces.

① Bounding rectangle in xy plane for two surfaces don't overlap.
② Surface S is completely behind overlapping surface relative to viewing position.
③ overlapping surface S' is completely in-front of S relative to viewing position.
④ projection of 2 surfaces onto view plane doesn't overlap.



no depth overlap

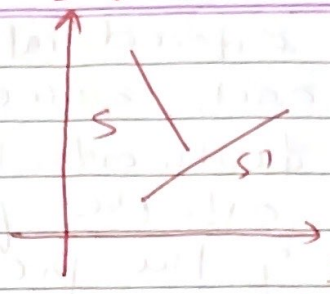
overlapping on z-plane.



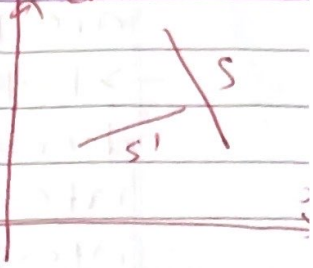
Depth overlap but no overlap in x-direction.

↓

no reordering needed.



S is completely behind overlapping surface S'



overlapping surface S' is completely infront of S.

Test 1

↓

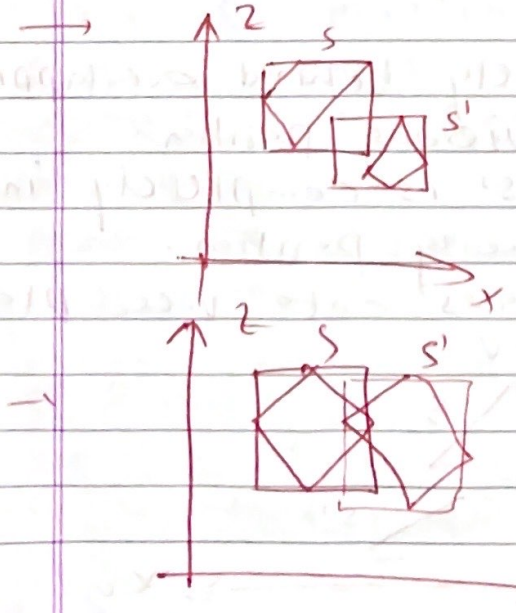
point check inside/outside

$Ax+By+Cz+D=0$

- substitute $S(x,y,z)$ in this eq^n

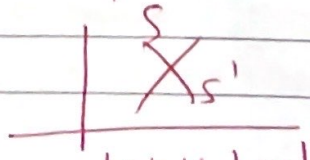
if $(Ax+By+Cz+D)_S < 0$, completely behind.

- Test 4



No overlapping projection

overlapping projection.



test 4 fail - need reordering.