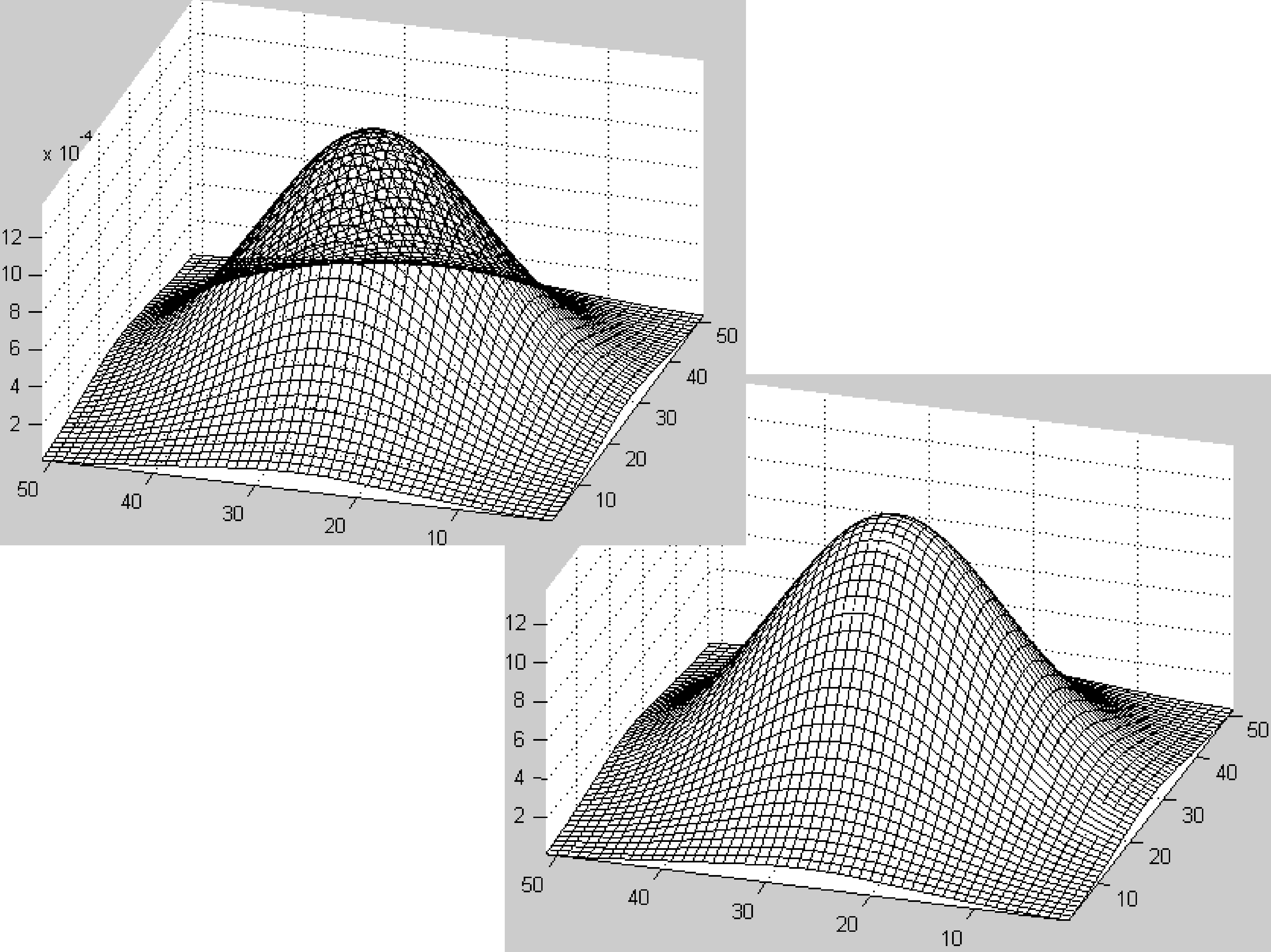
VISIBLE SURFACE DETECTION


x 10^-4
12
10
8
6
4
2
50
40
30
20
10
50
40
30
20
10
12
10
8
6
4
2
50
40
30
20
10
50
40
30
20
10

The problem of Visibility – Occlusion.

Problem Definition:

Given a set of 3-D surfaces to be projected onto a 2-D screen, obtain the nearest surface corresponding to any point on the screen.

Two types of methods used:

- **Object-space methods (Continuous):**

  Compares parts of objects to each other to determine which surfaces should be labeled as visible (use of bounding boxes, and check limits along each direction). Order the surfaces being drawn, such that it provides the correct impression of depth variations and positions.

- **Image Space methods (discrete):**

  Visibility is decided point by point at each pixel position on the projection plane. Screen resolution can be a limitation.

Hidden Surface – (a) Surface for rendering or
(b) Line drawing

Coherence properties:

- Object Coherence – If one object is entirely separate from another, do not compare.

- Face Coherence – smooth variations across a face; incrementally modify.

- Edge Coherence – Visibility changes if a edge crosses behind a visible face.

- Implied edge coherence – Line of intersection of a planar face penetrating another, can be obtained from two points on the intersection.

- Scanline coherence – Successive lines have similar spans.

- Area Coherence – Span of adjacent group of pixels is often covered by the same visible face.

- Depth Coherence – Use difference equation to estimate depths of nearby points on the same surface.

- Frame Coherence – Pictures of two successive frames of an animations sequence are quite similar (small changes in object and viewpoint).

Different Visible Surface Detection Methods:

- **Back-face Detection**
- **Depth (Z) buffer method**
- **Scan-line method**
- **Depth-sorting method**
- **Area-subdivision method**
- **Octree methods**
- **BSP Trees**
- **Ray casting method**

Visible surface techniques are 3D versions of sorting algorithms –

***basically compare depth*.**

Back face Culling or removal

A Polygon (in 3D) is a ***back face*** if:

V.N > 0.

Let V = (0, 0, V_z) and
N = Ai + Bj + Ck.

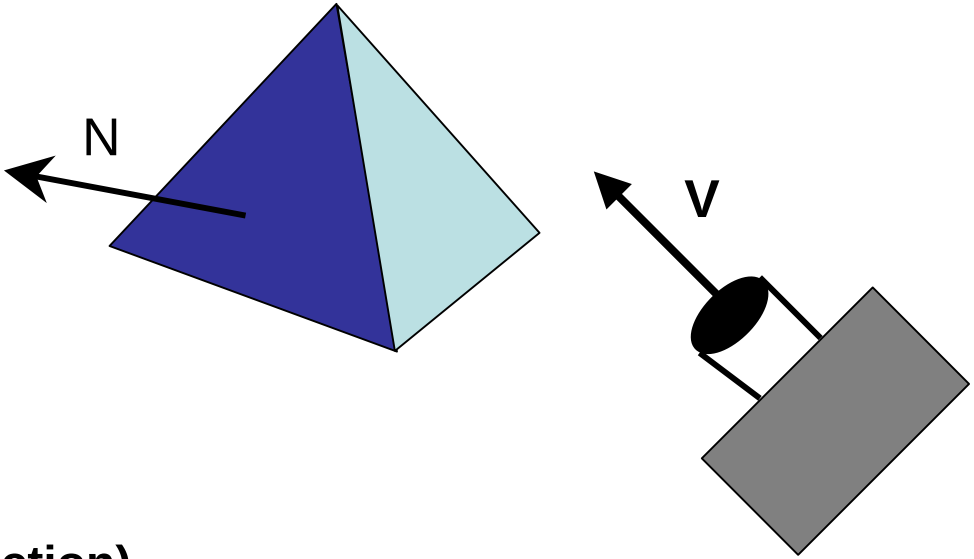
Then V.N = V_z.C;

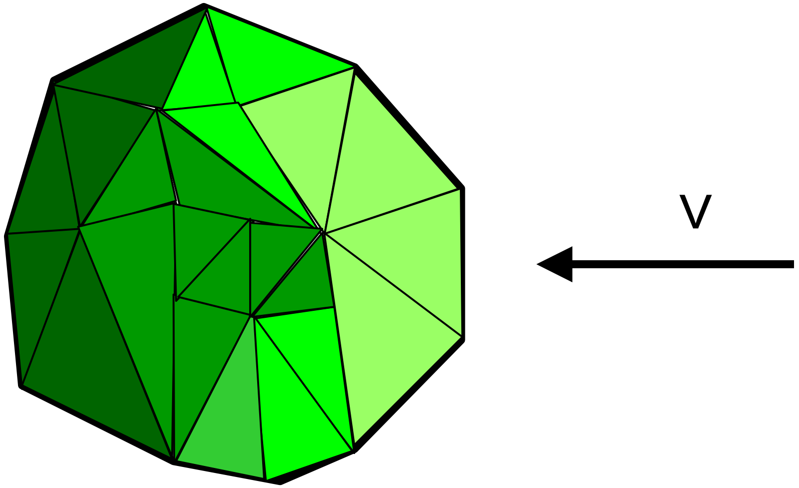
Let V_z be +ve (view along +ve Z-direction),

Check the sign of C.

Condition of back face is thus:

sgn(C) = 0.

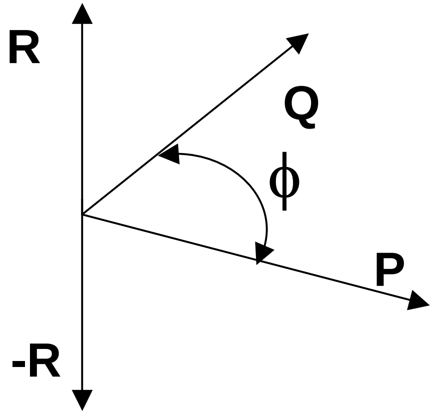




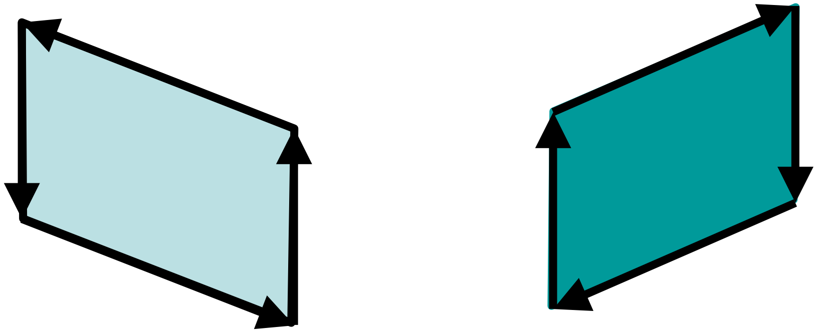
What happens if
V. N = 0 ??

How to get normal vector (N) for a 3D surface, polygon ?

$R = P \times Q$



Order of vertices for calculation of N – front side of the surface

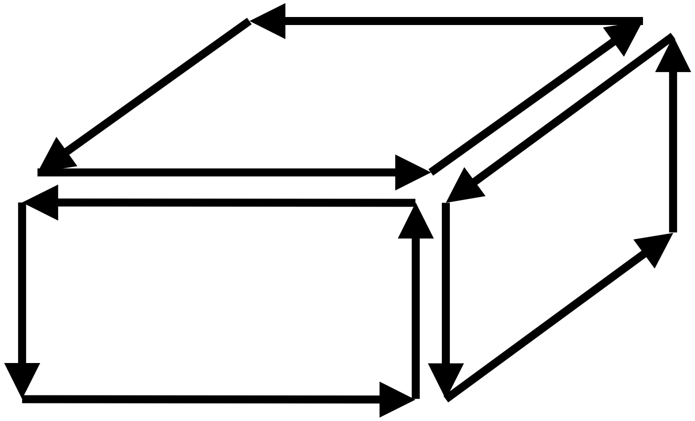


Brighter Side

Darker side

Take the order of vertices to be counter-clockwise for the brighter side. Take the case of a cube:

Exterior faces of a cube:



You can choose any two vectors (edges) to obtain the normal to the surface. Any risks in such a case, if we randomly choose any two vectors?

Better Solution to obtain the direction cosines of N:

Assume n vertices of the polygon (X_i, Y_i, Z_i). Compute:

$$a = \sum_{i=1}^{n} (Y_i - Y_j)(Z_i + Z_j)$$

$$b = \sum_{i=1}^{n} (Z_i - Z_j)(X_i + X_j) \quad \Bigg| \quad where\ j = i + 1$$

$$c = \sum_{i=1}^{n} (X_i - X_j)(Y_i + Y_j)$$

$$If \quad i = n, \quad j = 1$$

Take the expression of C.

Can you relate that to one property of the 2-D polygon in X-Y plane?

a, b **and** ***c*** **also describe the projection surface of the polygon on the YZ, Z-X and X-Y planes (or along the X, Y and Z axis) respectively.**

If A is the total area of the polygon, the projected areas are:

A_{YZ} = a/2; A_{XZ} = b/2; A_{XY} = c/2;

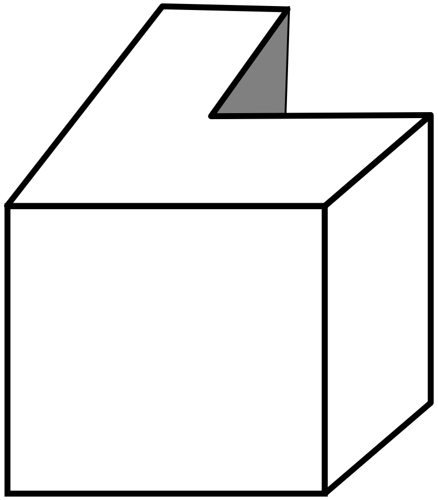
The surface normal appears to be pointing outwards. Facing towards the viewer, the edges of the polygon appears to be drawn counter-clockwise.

The polygon is a back face, if $c \leq 0$

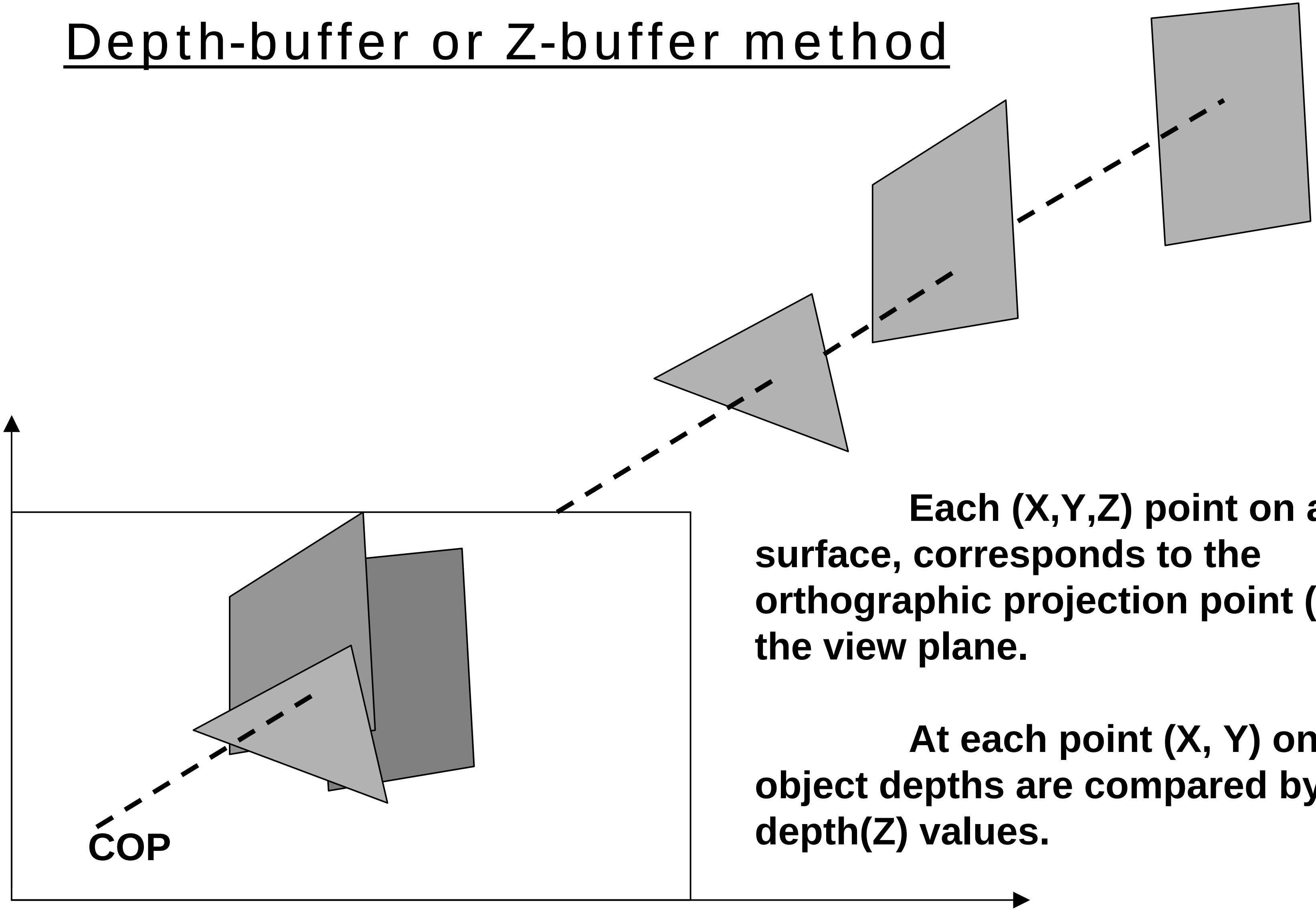
Drawbacks of back face culling:

- **Partially hidden faces cannot be determined by this method**
- **Not useful for ray tracing, or photometry/radiosity.**

However, this is still useful as a pre-processing step, as almost 50% of the surfaces are eliminated.



Depth-buffer or Z-buffer method



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

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

Assume normalized coordinates: (FCP) $Z_{max} > Z > 0$ (BCP), where $Z_{max} = 1$.

Two buffer areas are used:

(i) Depth (Z) buffer: To store the _*depth*_ values for each (X, Y) position, as surfaces are processed.

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

Steps for Processing:

(a) Initialize

$$\forall(X,Y) \left| \begin{array}{l} depth(X,Y) = Z_{\max} \\ refresh(X,Y) = I_B \end{array} \right.$$

I_B = 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)

I_s is the projected intensity value of the surface at position (X, Y), which has the minimum value of Z, at the current stage of iteration

Calculation of Z:

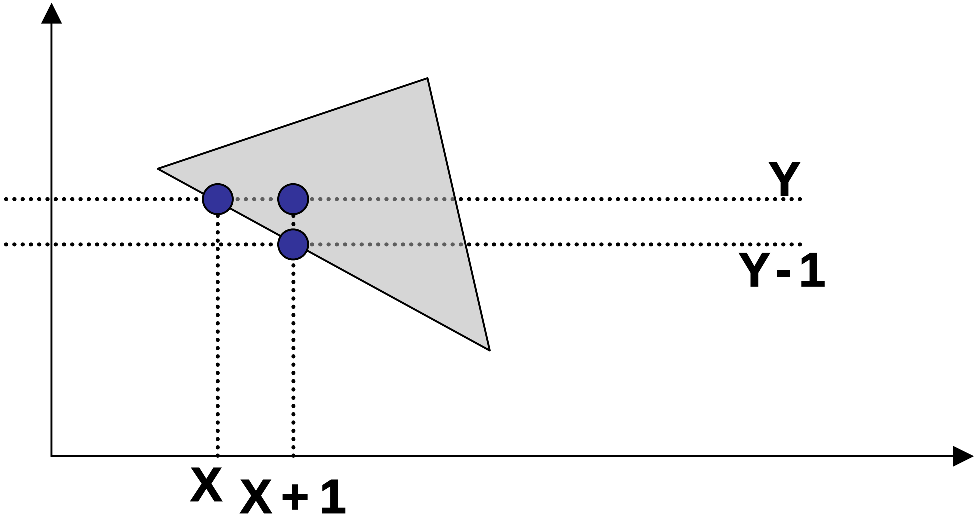
$$\text{Equation of the surface:}$$

$$AX + BY + CZ + D = 0;$$

$$Z = \frac{-AX - BY - D}{C}$$

Equation for Z at the next scan line:

Remember scanline/Polyfill algorithm?
Using edge coherence, we get for the next scanline (Y+1):
X’ = X -1/m; Thus:



$$Z_{X+1} = Z_X - \frac{A}{C};$$

$$Z_{Y+1} = Z_Y + \frac{\frac{A}{m} + B}{C};$$

For a vertical edge: Z’ = Z + B/C;

So to implement the algorithm, three constants are required for each surface:

$$-\frac{A}{C},\ \frac{(\frac{A}{m}+B)}{C},\ \frac{B}{C}$$

What is the special condition, C = 0?

Using the above three constants, we can keep calculating the successive depth values along and for successive scanlines. Similar approaches can be used for curved surfaces: Z = f(X, Y).

Z-Buffer Algo:

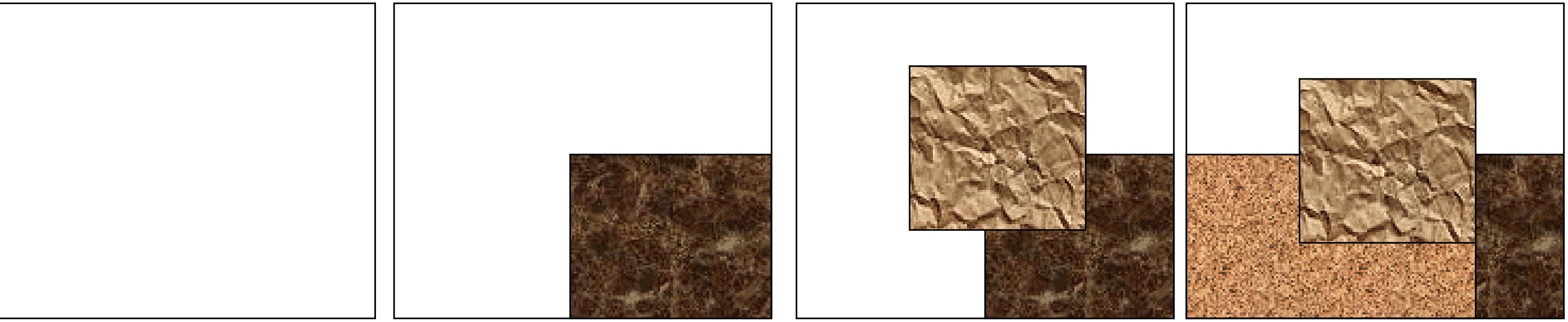
```
for all (x,y)
        depth(x,y) = $-\infty$      /* Watch this change */
        refresh(x,y) = $I_B$
for each polygon P
        for each position (x,y) on polygon P
        calculate depth z
        if z > depth(x,y) then
                1. depth(x,y) = z
                2. refresh(x,y) = $I_P(x,y)$
```

Z-values of the coordinates:

4	4
4	4

9	8
8	7

7	6	5
7	6	5



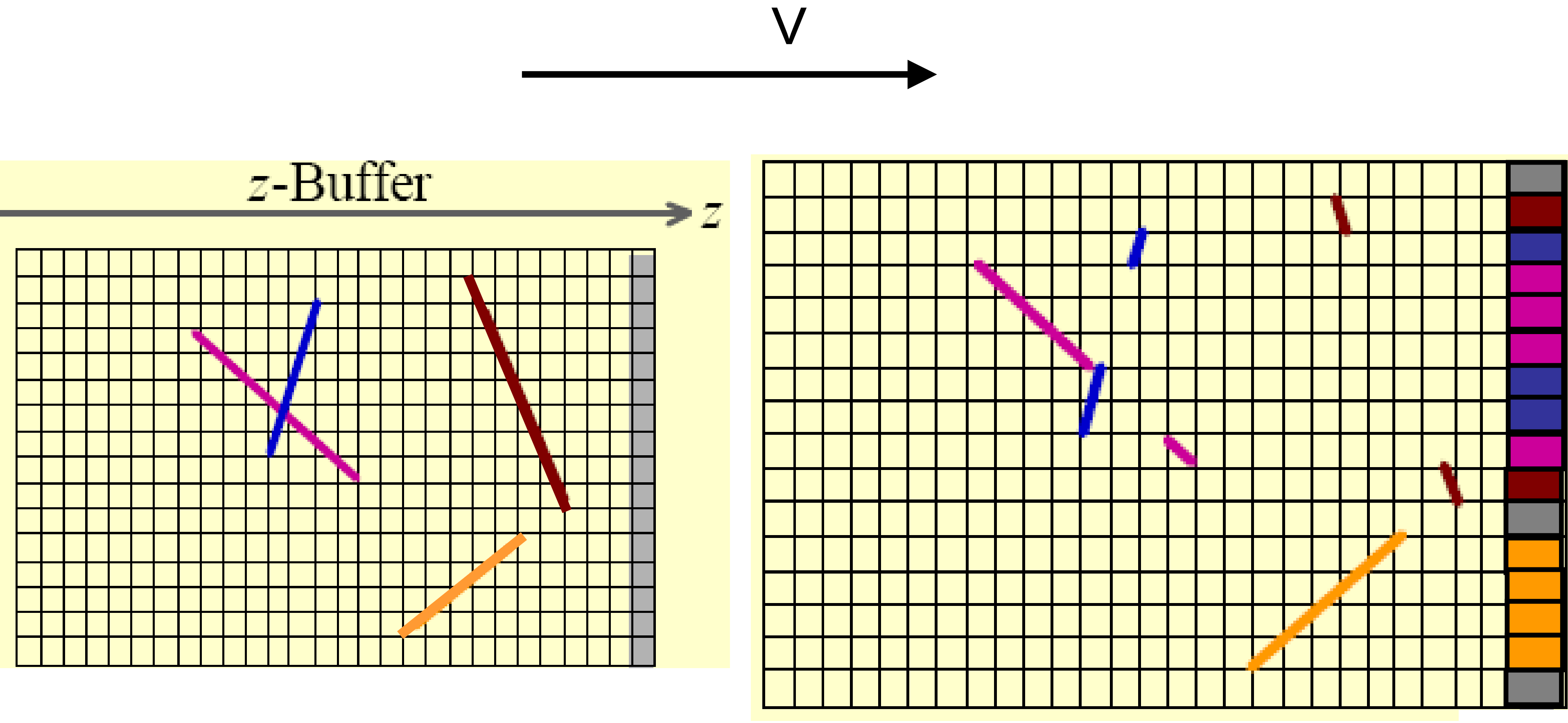
IMAGES (after pseudo-texture rendering)

		4	4
		4	4

	9	8	
	8	7	4
		4	4

	9	8	
7	8	7	4
7	6	5	4

Z-Buffer values


V
z-Buffer
z

SCAN-LINE Algorithms (for VSD)

Extension of 2-D Scanline (polyfill) algorithm. Here we deal with a set of polygons.

Data structure used: ET (Edge Table), AET (Active Edge Table) and PT (Polygon Table).

Edge Table entries contain information about edges of the polygon bucket sorted based on each edge's smaller Y coordinate. Entries within a bucket ordered by increasing X-coordinate of their endpoint.

Structure of each entry in ET:

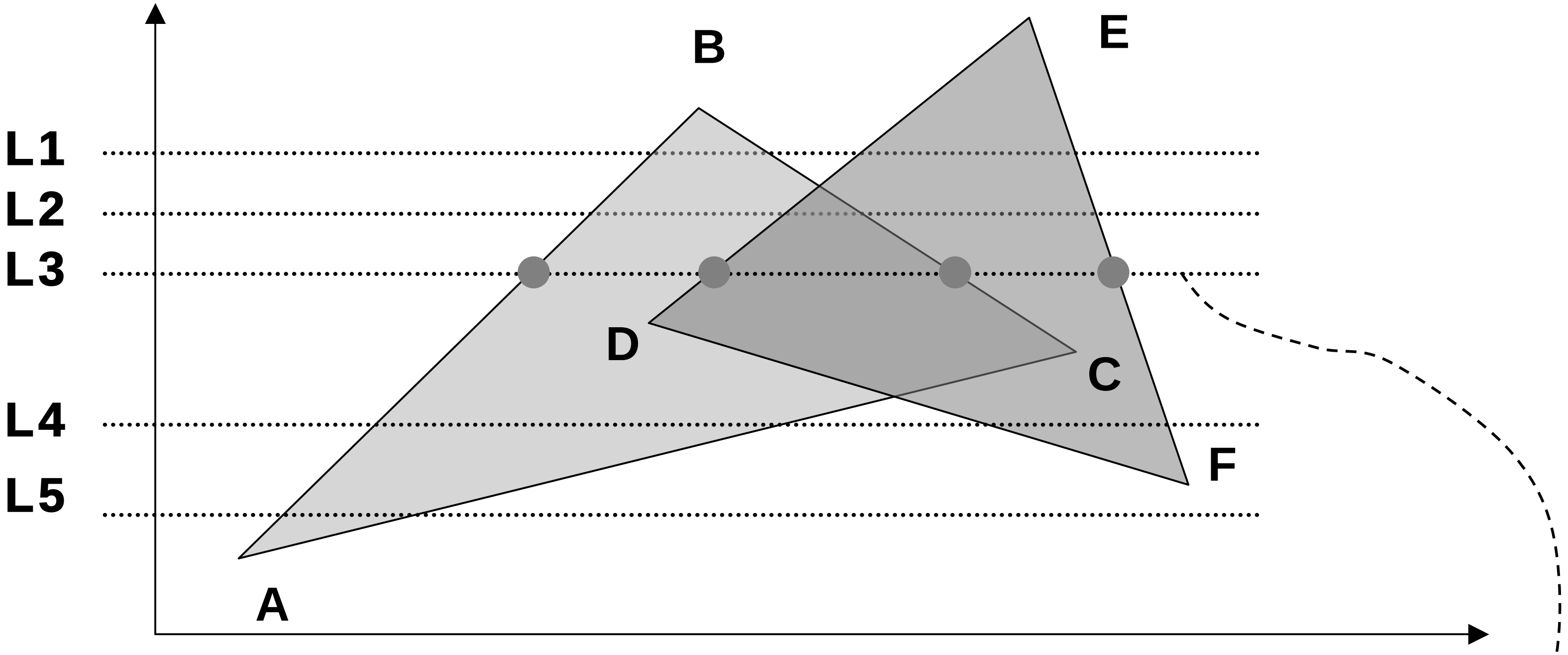
- **X-Coordn. of the end with the smaller Y-Coordn.**
- **Y-Coordn. of the edge's other end point**
- **ΔX = 1/m**
- **Polygon ID**

X	Y_{max}	ΔX	ID	● →

Structure of each entry in PT:

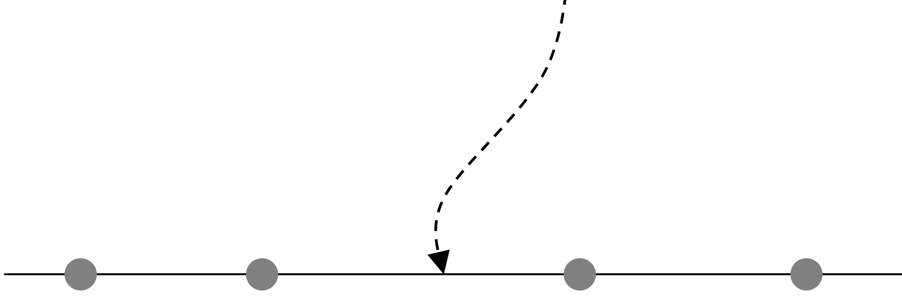
- **Coefficients of the plane equations**
- **Shading or color information of the polygon**
- **Flag (IN/OUT), initialized to '*false*'**

ID	Plane Coeffs.	Shading Info.	IN/OUT



AET Contents

Scan Line	Entries			
L5	AB	AC		
L4	AB	AC	FD	FE
L3, L2	AB	DE	CB	FE
L1	AB	CB	DE	FE



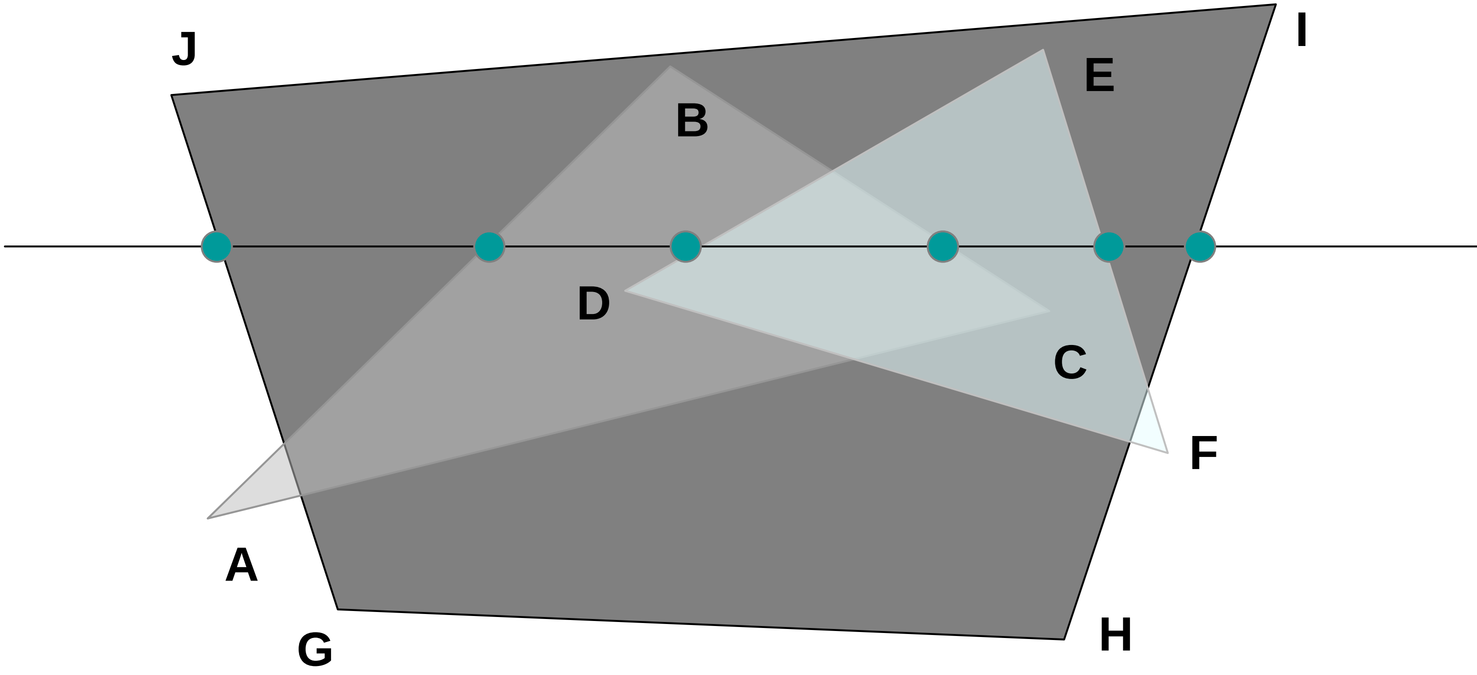
ABC IN;
DEF OUT

ABC IN;
DEF IN.

ABC OUT;
DEF IN.

Compare Z values
in this region to find
the visible Z value.

Three non-intersecting polygons

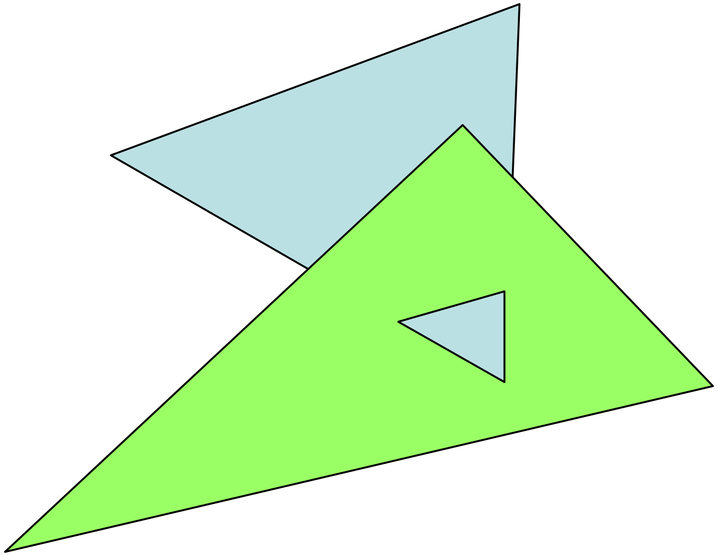


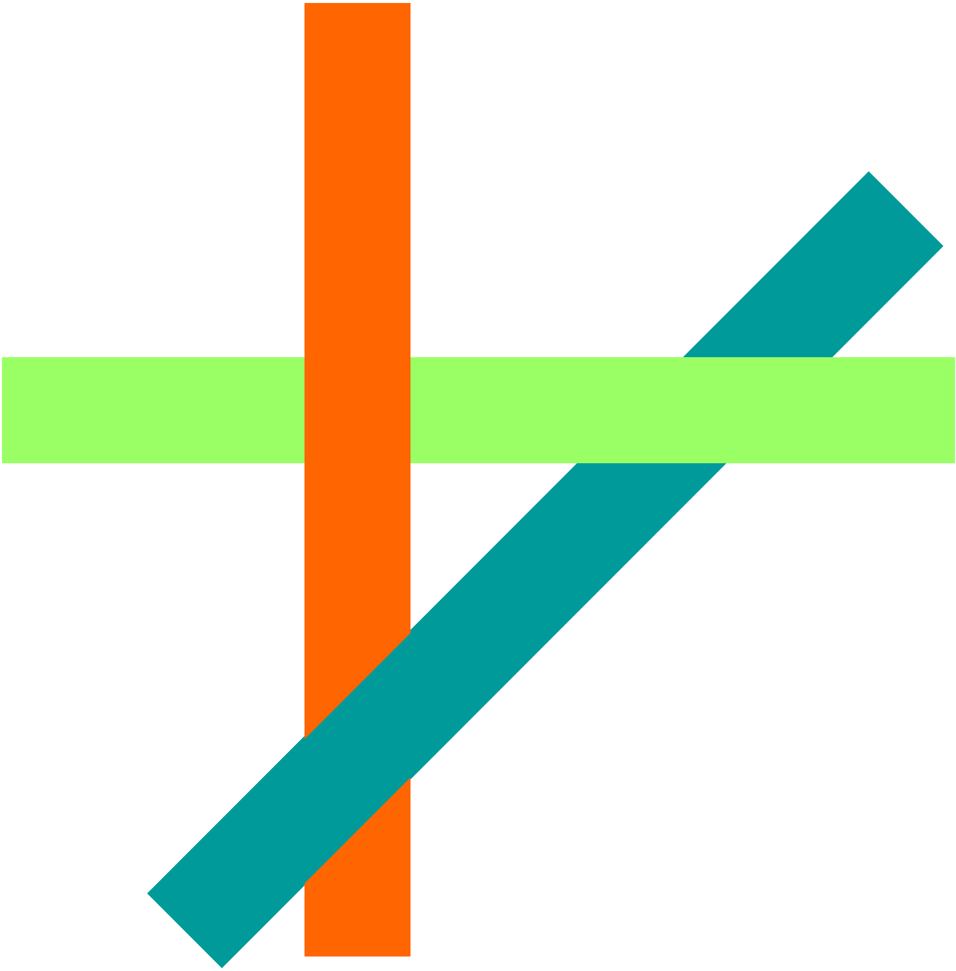
GHIJ is behind ABC and DEF.

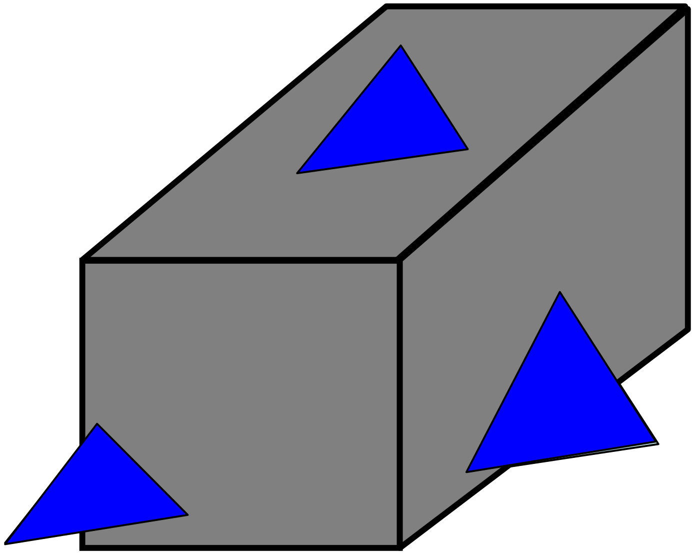
So, when scanline leaves edge BC, it is still inside polygons DEF and GHIJ. If polygons do not intersect, depth calculations and comparisons between GHIJ and DEF can be avoided.

Thus depth computations are unnecessary when the scanline leaves an obscured polygon. It is required only when it leaves an obscuring polygon.

Additional treatment is necessary for intersecting polygons.







Depth-sorting or Painter's Algorithm

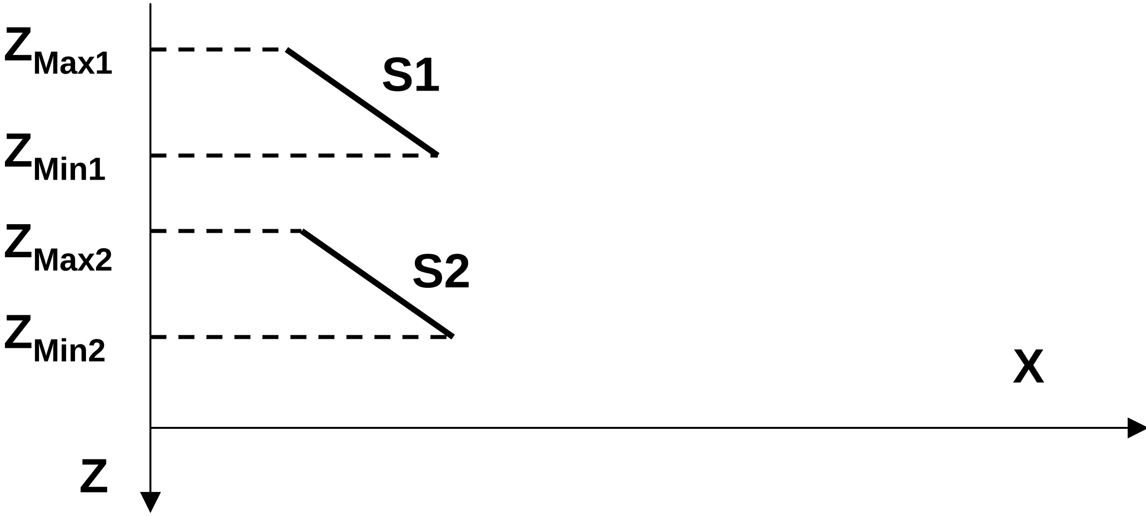
Paint the polygons in the frame buffer in order of decreasing distance from the viewpoint.

Broad Steps:

- **Surfaces are sorted in increasing order of DEPTH.**
- **Resolve ambiguities when polygons overlap (in DEPTH), splitting polygons if necessary.**
- **Surfaces are scan converted in order, starting with the surface of greatest DEPTH.**

Principle:

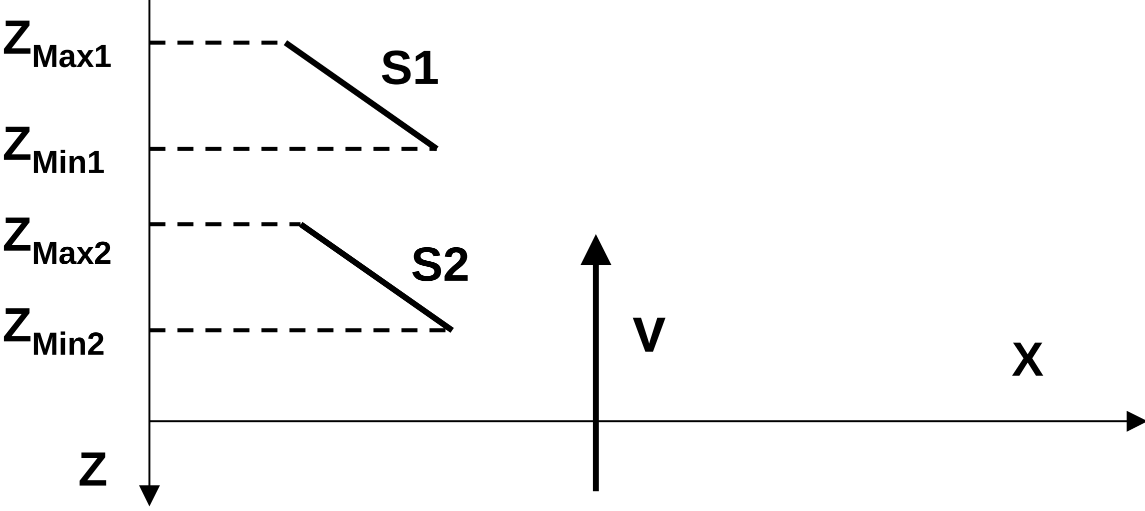
Each layer of paint (polygon surface of an object) covers up the previous layers while drawing.



Since,
$Z_{Min1} > Z_{Max2}$
no overlap occurs.

S1 is first scan converted, and then S2.

This goes on, as long as no overlap occurs.



If depth overlap occurs, additional comparisons are necessary to reorder the surfaces. The following set of *tests* are used to ensure that no re-ordering of surfaces is necessary:

1. **The boundary rectangles in the X-Y plane for the two surfaces do not overlap.**

2. **Surface S is completely behind the overlapping surface relative to the viewing position.**

3. **The overlapping surface is completely in front of S relative to the viewing position.**

4. **The projections of the two surfaces onto the view plane do not overlap.**

Tests must be performed in order, as specified.

Condition to RE-ORDER the surfaces:

If any one of the 4 tests is TRUE, we proceed to the next overlapping surface. i.e. If all overlapping surfaces pass at least one of the tests, none of them is behind S. No re-ordering is required, and then S is scan converted.

RE-ORDERing is required is all the four tests fail.

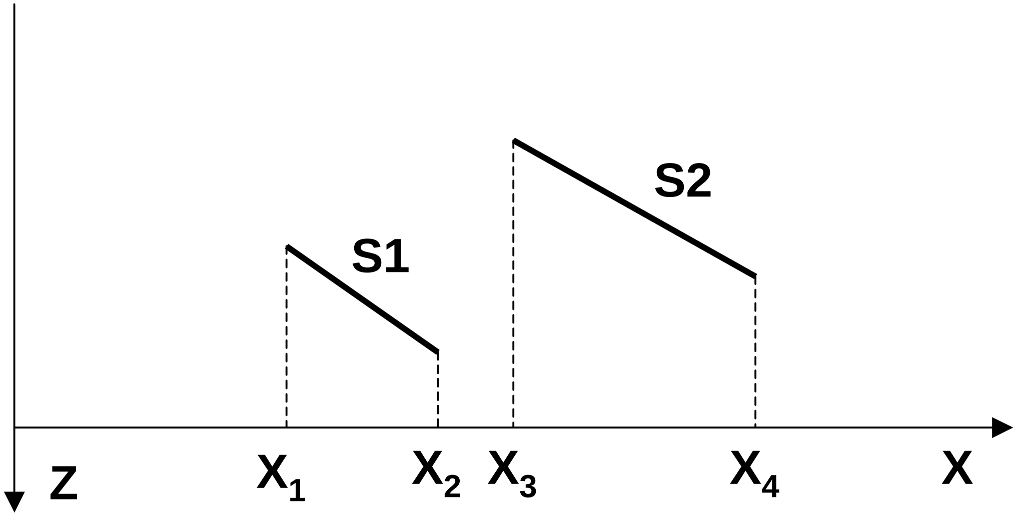
TEST #1:

The boundary rectangles in the X-Y plane for the two surfaces do not overlap.

We have depth overlap, but no overlap in X-direction.

Hence, Test #1 is passed, scan convert S2 and then S1.

If we have X overlap, check for the rest.



TEST #2:

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

TEST #3:

The overlapping surface is completely in front of S relative to the viewing position.

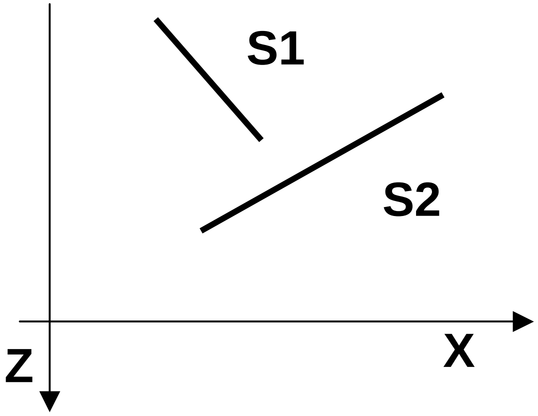


Fig. 1. S1 is completely behind/inside the overlapping surface S2

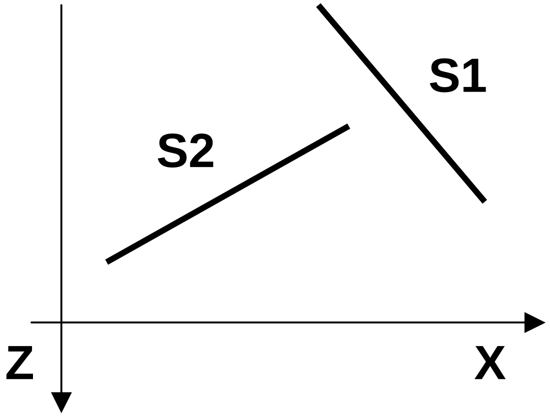


Fig. 2. Overlapping surface S2 is completely front/outside S1. But S1 is not completely behind S2.

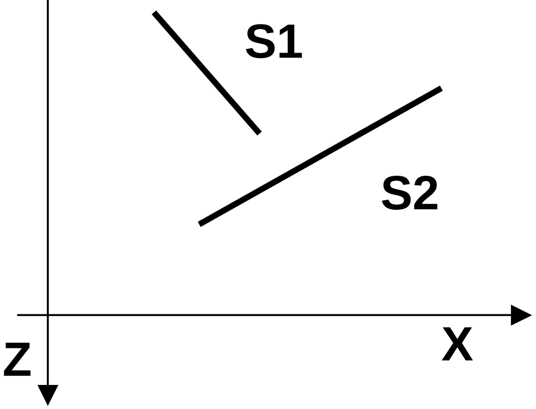


Fig. 1. S1 is completely behind/inside the overlapping surface S2

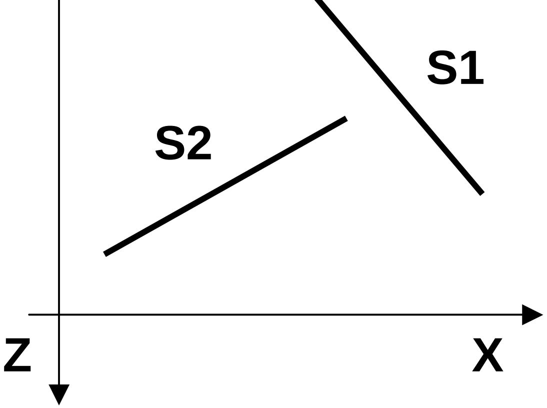


Fig. 2. Overlapping surface S2 is completely front/outside S1. But S1 is not completely behind S2.

In Fig. 2, S2 is in front of S1, but S1 is not completely inside S2 – Test 2 it not TRUE or FAILS, although Test 3 is TRUE.

How to check the conditions?

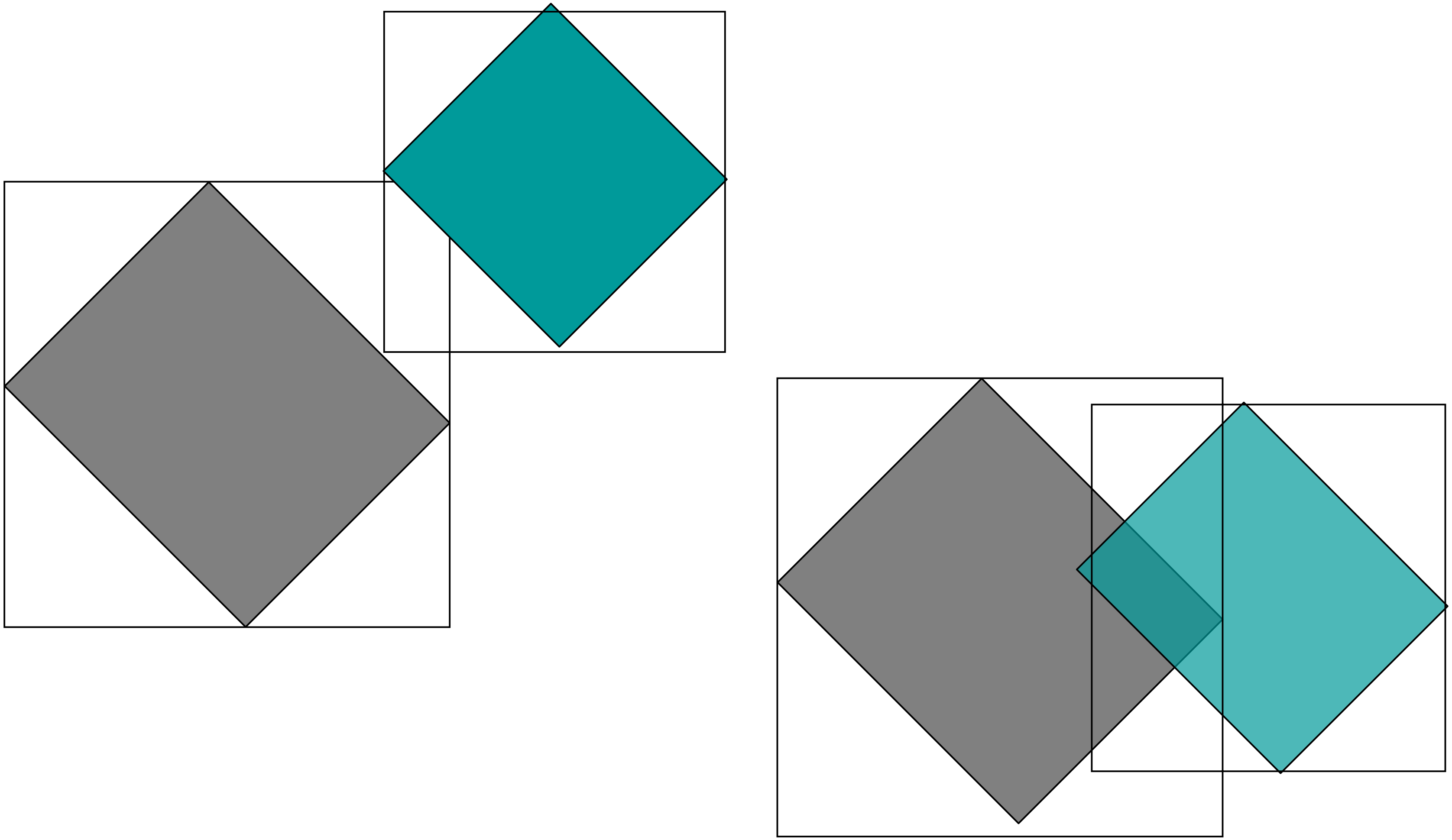
i) Set the plane equation of S2, such that the surface S2 is towards the viewing position.

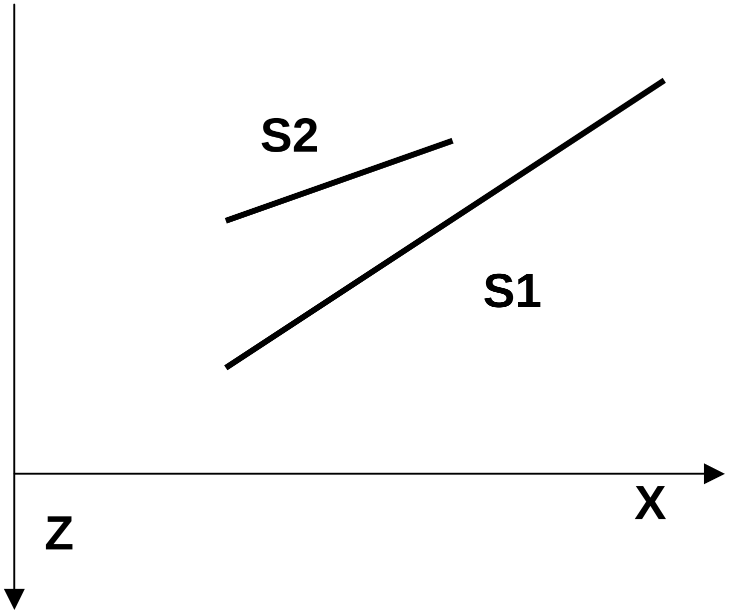
ii) Substitute the coordinates of all vertices of S1 into the plane equation of S2 and check for the sign.

iii) If all vertices of S1 are *inside* S2, then S1 is behind S2. (Fig. 1). If all vertices of S1 are outside S2, S1 is in front of S2.

TESt #4:

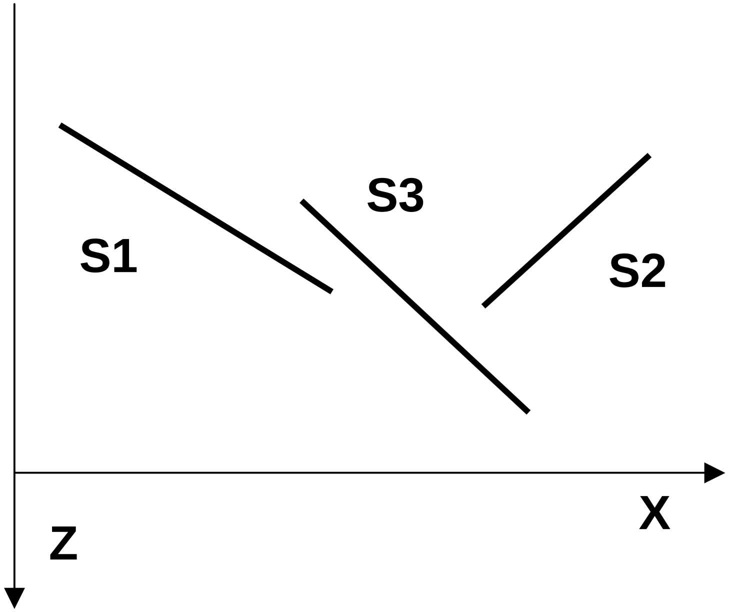
The projections of the two surfaces onto the view plane do not overlap.





Initial order:
S1 -> S2

Change the order:
S2 -> S1

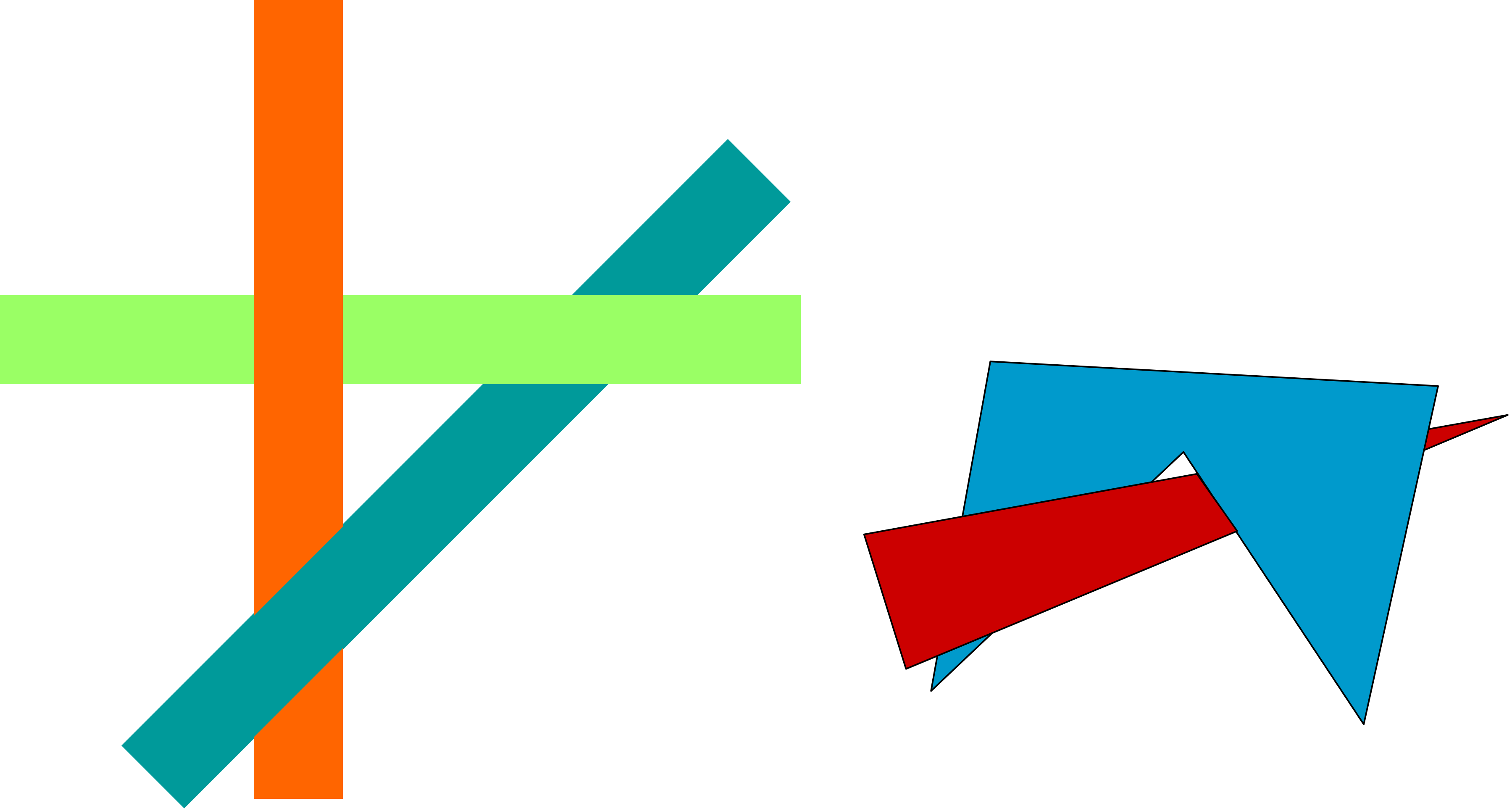


Initial Order:
S1 -> S2 -> S3.

S1 -> S2, Test 1 passed.

Check S1, S3. All tests fail. Interchange.
S3 -> S2 -> S1.

Check S2 and S3. All tests fail again.
Correct Order: S2 -> S3 -> S1.



Process of checking the order, results in an infinite loop.

Avoided by setting a FLAG for a surface that has been re-ordered or shuffled. If it has to be altered again, split it into two parts.

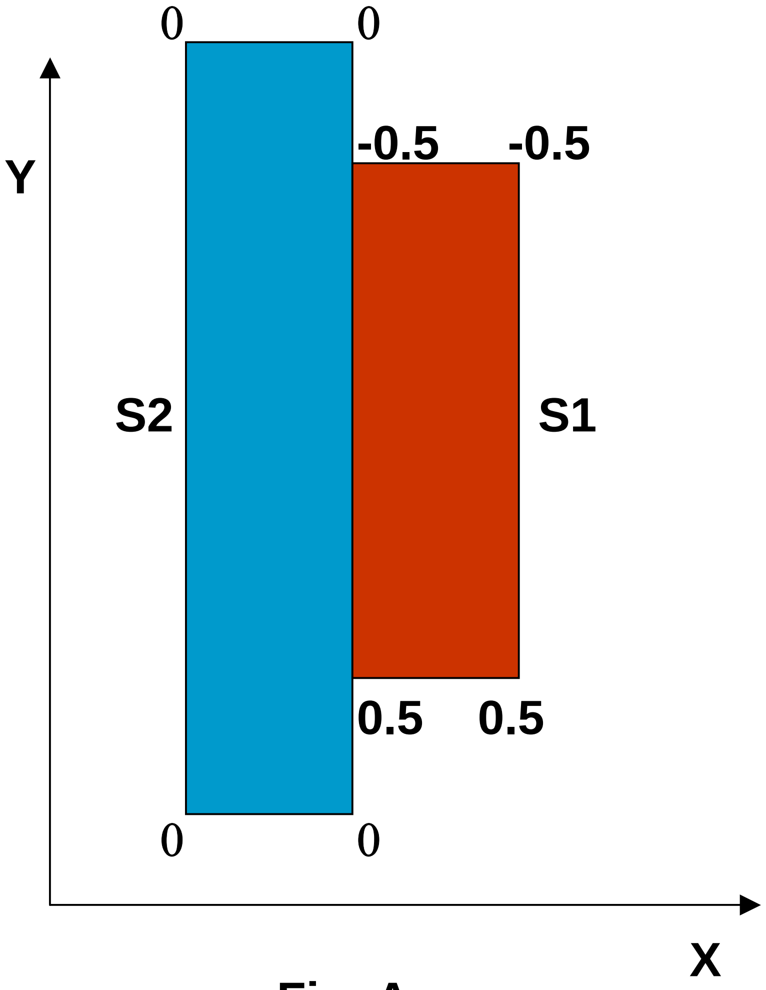


Fig. A

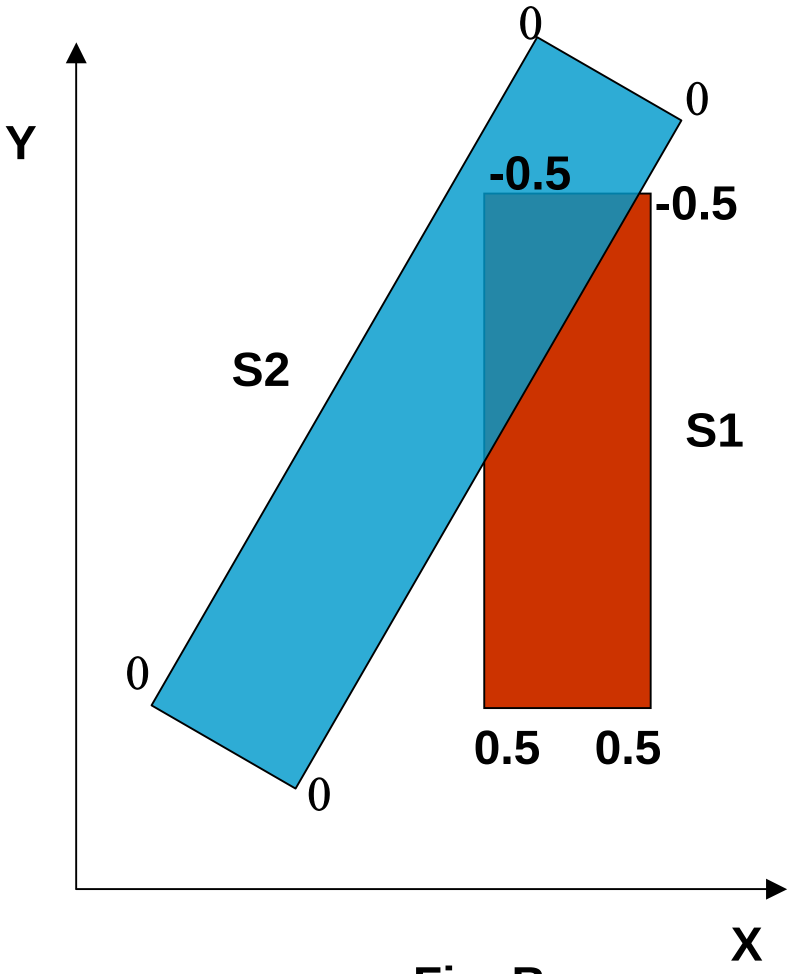


Fig. B

What happens in these two cases ?

Area sub-division method

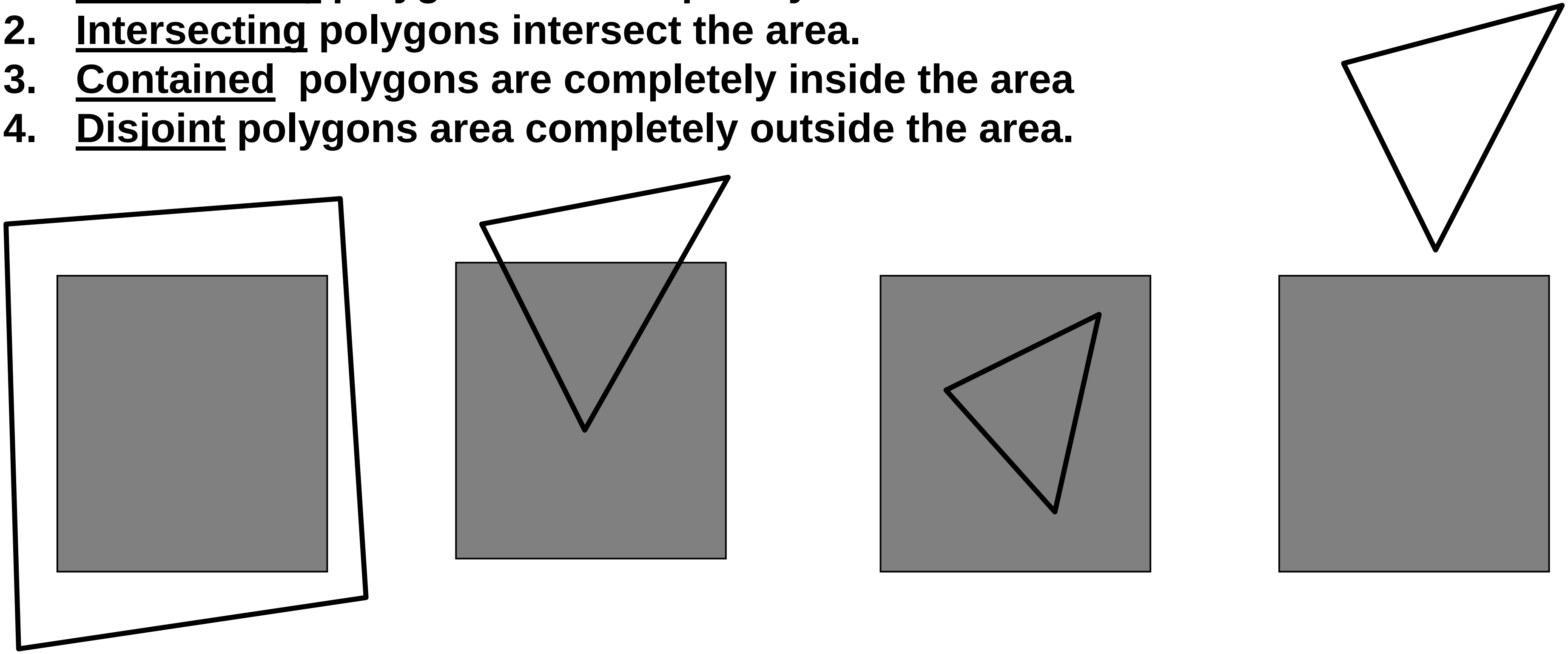
- Works in image-space
- Area Coherence is exploited
- Divide-and-conquer strategy.

Examine and divide if necessary

WARNOCK’s Algorithm

Possible relationships of the area of interest (rectangular) and the projection of the polygon:

1. **Surrounding polygons are completely inside the area of interest.**
2. **Intersecting polygons intersect the area.**
3. **Contained polygons are completely inside the area**
4. **Disjoint polygons area completely outside the area.**



Surrounding **Intersecting** **Contained** **Disjoint**

Treat interior part of the intersection and contained as equivalent. Disjoint is irrelevant. The decisions about division of an area is based on:

1. **All polygons are disjoint from the area. Display background color.**

2. **Only one Intersecting part or interior (contained): Fill the area with background color and then scan-convert the polygon.**

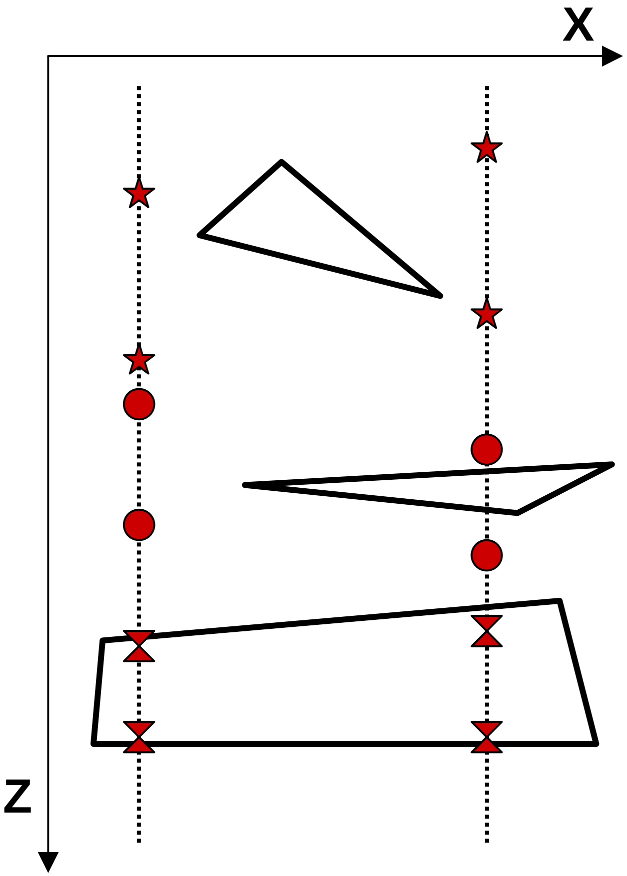
3. **Single surrounding polygon, and no intersecting or contained polygon. Use color of surrounding polygon to shade the area.**

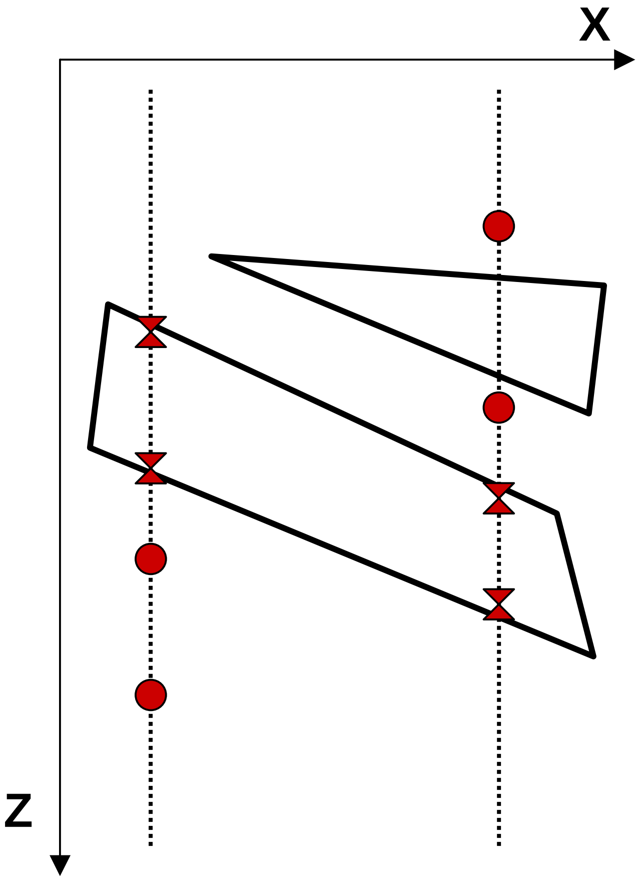
4. **More than one polygon intersect, contained in and surrounding the area. But the surrounding polygon is in front of all other polygons. Then fill the area with the color of the surrounding polygon.**

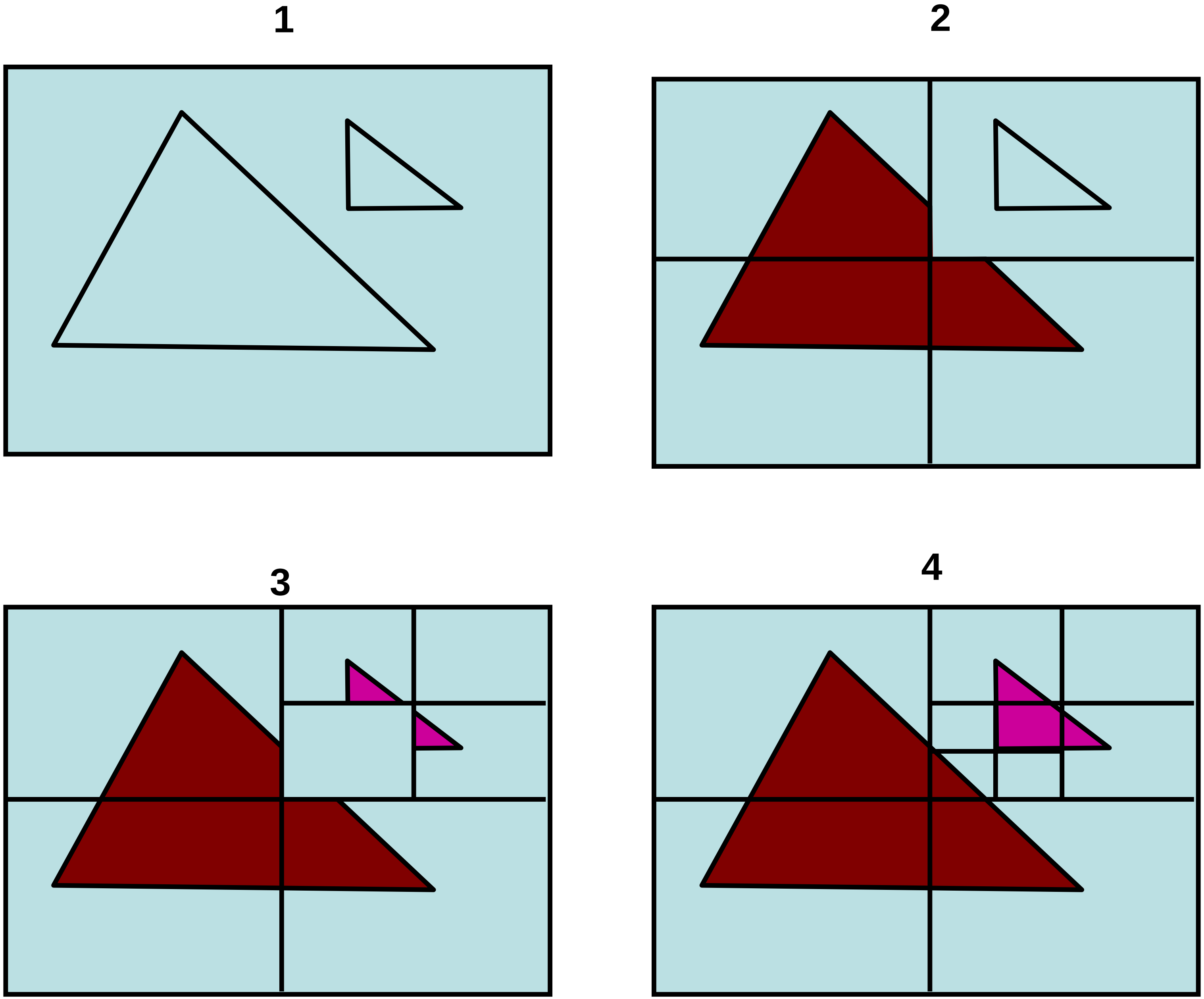
Step 4 is implemented by comparing the Z-coordinates of the planes of all polygons (involved) at the four corners of the area

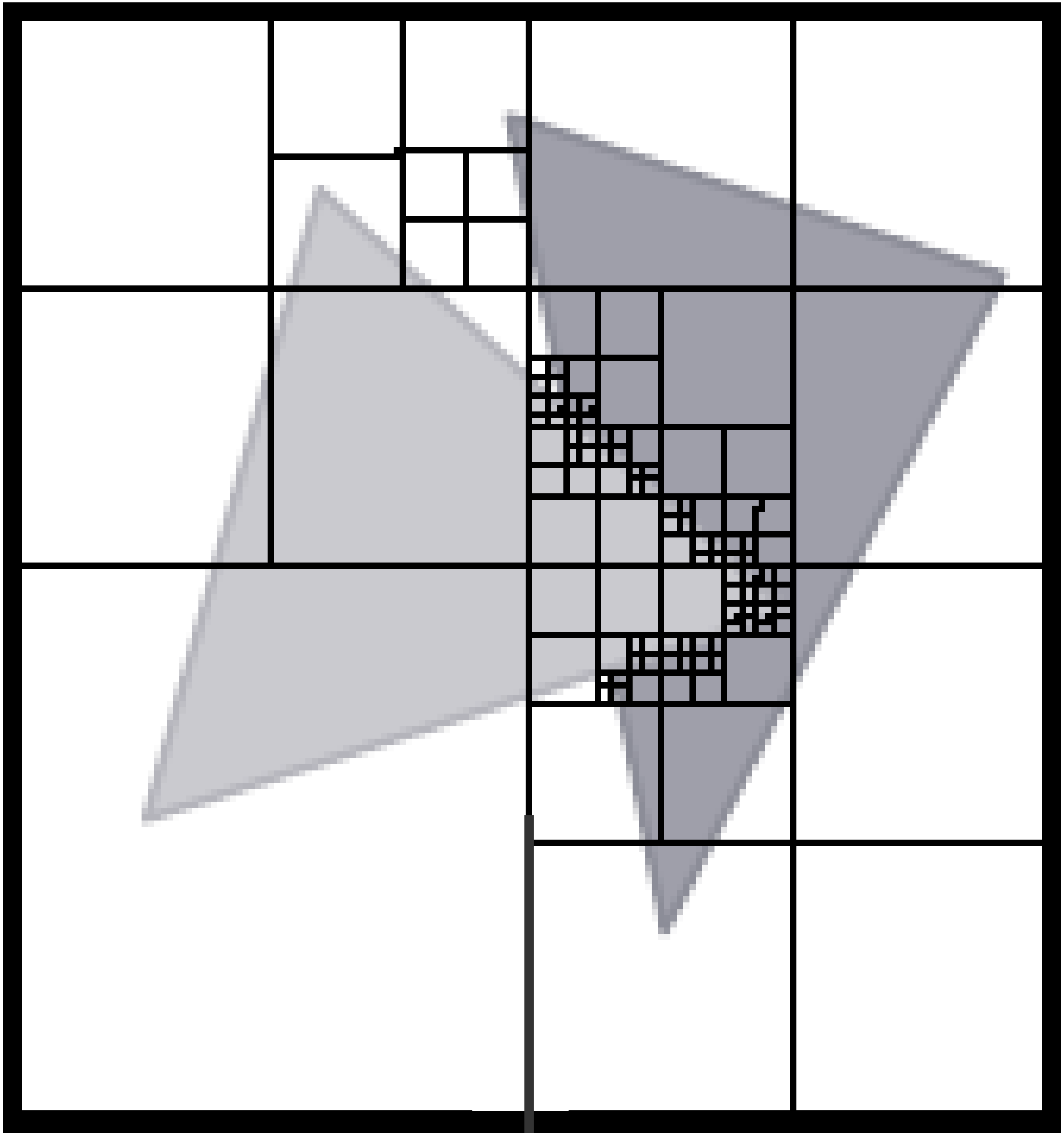
If all four tests fail, divide the rectangular area into four equal parts. Recursively apply this logic, till you reach the lowest minimum area or maximum resolution – pixel size. Use nearest surface in that case to paint/shade.

Decision – Step 4;

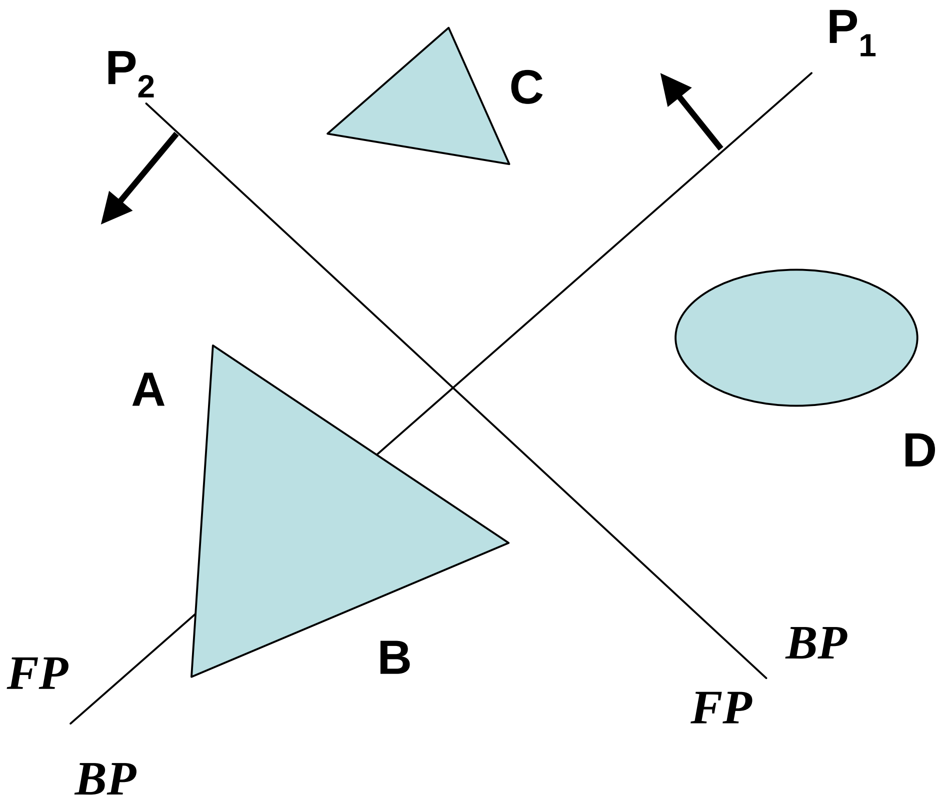


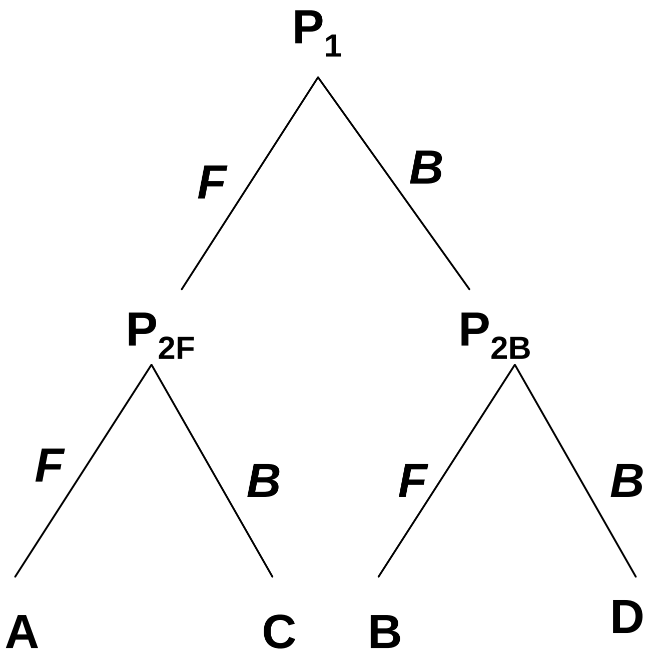



1
2
3
4



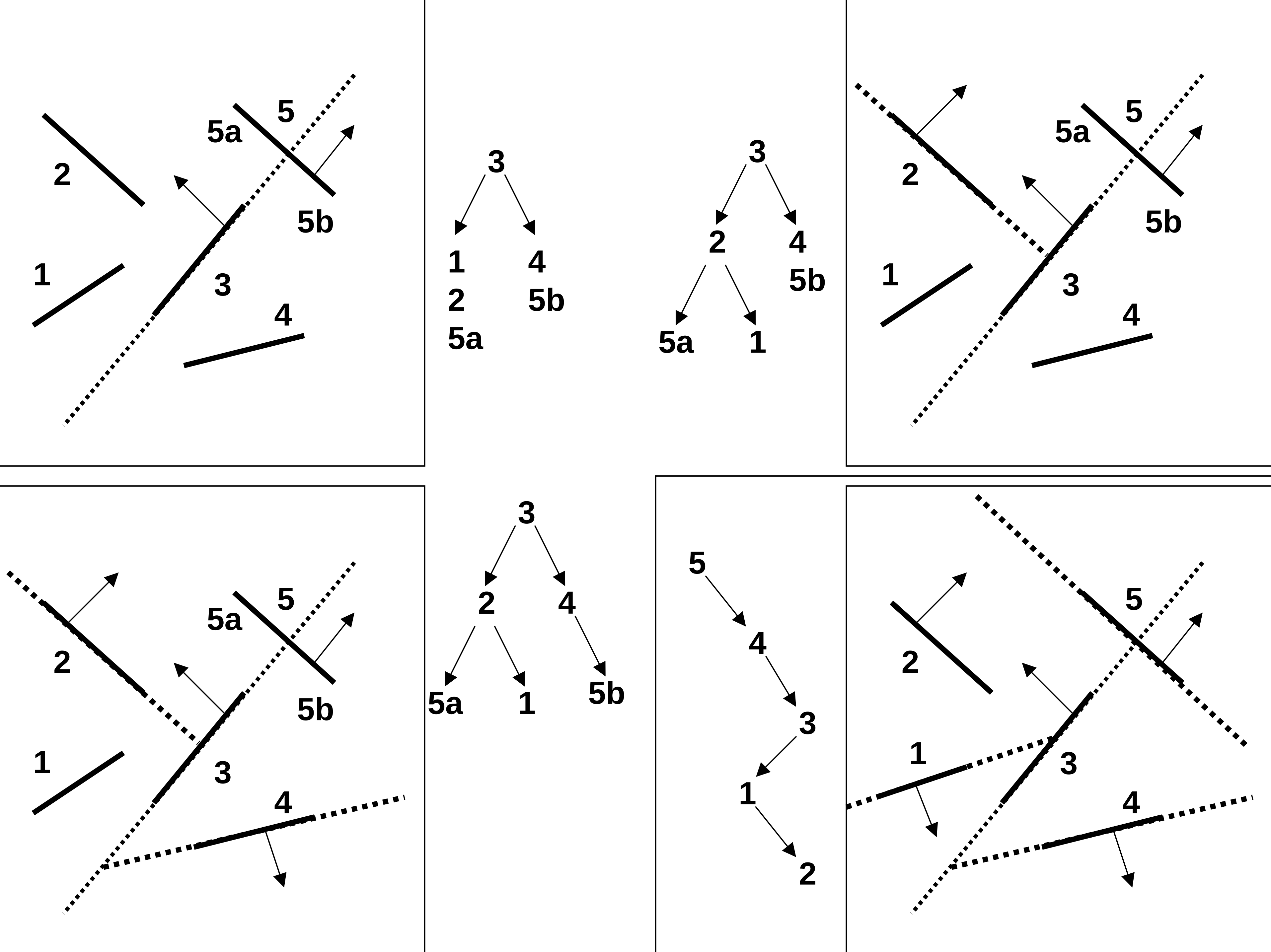
BSP (Binary Space Partition) trees





Salient features:

- **Identify surfaces that are inside/front and outside/back the partitioning plane at each step of the space division, relative to the viewing direction.**
- **Start with any plane and find one set of objects behind and the rest in the front.**
- **In the tree, the objects are represented as terminal nodes with front objects as left branches and back objects as right branches.**
- **BSP tree’s root is a polygon selected from those to be displayed.**
- **One polygon each from the root polygon’s front and back half plane, becomes its front and back children**
- **The algorithm terminates when each node contains only a single polygon.**
- **Intersection and sorting at object space/precision**


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

Display list of a BSP tree

- BSP tree may be traversed in a modified in-order tree walk to yield a correctly priority-ordered polygon list for an arbitrary viewpoint.
- If the viewer is in the root polygon’s front half space, the algorithm must first display:

  1. all polygons in the root’s rear half-space
  2. then the root
  3. and finally all polygons in the front half-space.

- Use back face culling
- Each of the root’s children is recursively processed.

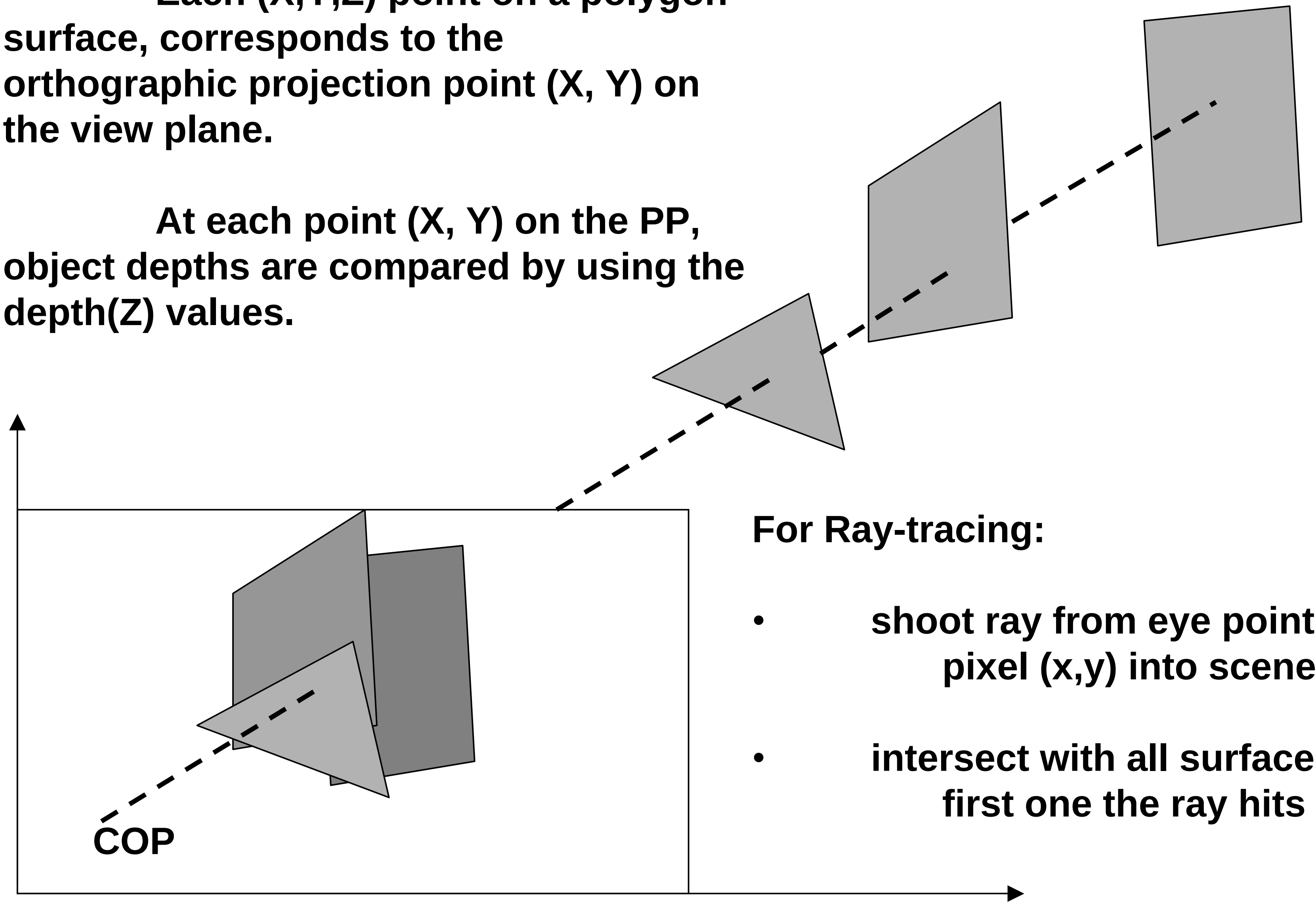
- **Disadvantages**
  - **more polygon splitting may occur than in Painter's algorithm**
  - **appropriate partitioning hyperplane selection is quite complicated and difficult**

VSD Ray Tracing

Remember? For Z-buffer:

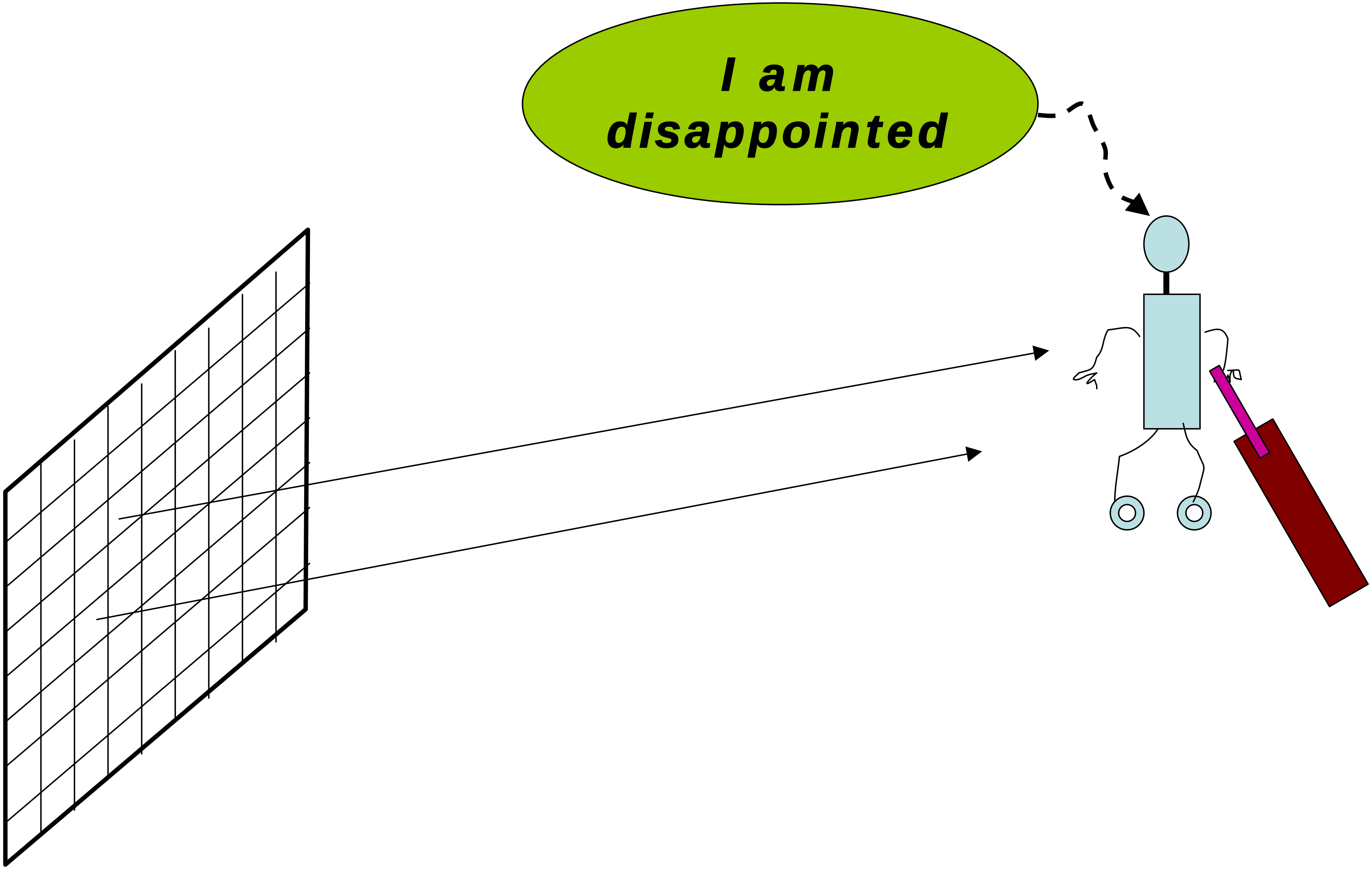
Each (X,Y,Z) point on a polygon surface, corresponds to the orthographic projection point (X, Y) on the view plane.

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



For Ray-tracing:

- **shoot ray from eye point through pixel (x,y) into scene**
- **intersect with all surfaces, find first one the ray hits**
- **shade that point to compute the color of pixel (x,y)**



I am disappointed

- **Line of Sight of each pixel is intersected with all surfaces**
- **Consider only the closest surface for shading**
- **Based on optics of image formation, paths of light rays are traced**
- **Light rays are traced backward**
- **Suitable for complex curved surfaces**
- **Need of an efficient ray-surface intersection technique**
- **Computationally expensive**
- **Almost all visual effects can be generated**
- **Can be parallelized**
- **Has aliasing problems**

Mathematically, the intersection problem is of finding the roots of an equation:

Surface – RAY = 0; **Ray Equation:** **r(t) = t (P – C)**

Eqns. for

LINE:

$$x = x_0 + t\Delta x\,;\ \ y = y_0 + t\Delta y\,; z = z_0 + t\Delta z\,;$$

CIRCLE:

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$$

Substitution gives us:

$$(x_0 + t\Delta x)^2 - 2a(x_0 + t\Delta x) + a^2$$
$$+ (y_0 + t\Delta y)^2 - 2b(y_0 + t\Delta y) + b^2$$
$$+ (z_0 + t\Delta z)^2 - 2c(z_0 + t\Delta z) + c^2 = r^2$$

Collecting terms gives us:

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 +$$
$$2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)]$$
$$+ (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$$

Equation is Quadratic, with coefficients from the constants of sphere and ray equations.

Cases:

- No real roots - Surface and ray do not intersect
- One real root - Surface tangentially grazes the surface
- Two real roots - From both intersections, get the one with the smallest value of t.

What is the shade, at that point of intersection?

Sphere has center (a, b, c). The surface normal at any point of intersection is:

$$[\frac{x_p - a}{r}, \frac{y_p - b}{r}, \frac{z_p - c}{r}]$$

Eqns. for

LINE : $x = x_0 + t\Delta x\,;\;\; y = y_0 + t\Delta y\,; z = z_0 + t\Delta z\,;$

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

Substitution gives us:

$$t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{(A\Delta x + B\Delta y + C\Delta z)}$$

- Denominator should not be zero;

- Find if intersection is within the polygon, by projecting onto a suitable coordinate plane;

- Overall processing is ray-wise, not polygon-wise.

Comparison of VSD (HSR) techniques

Algorithms/ Methods	Memory	Speed	Issues in Implementation	Remarks
Z-Buffer	Two arrays	Depth complexity	Scan conversion, Hardware	Commonly used
Painter's	One array	Apriori sorting helps speed-up	Scan conversion	Splitting and sorting the major bottleneck
Ray casting	Object database	O(#pixels, #surfaces or objects)	Spatial data structures help speedup	Excellent for CSG, shadows, transparency
Scanline, Area sub-division			Hard	Cannot be generalized for non-polygonal models.