Visible Surface Detection

Dr. V Masilamani

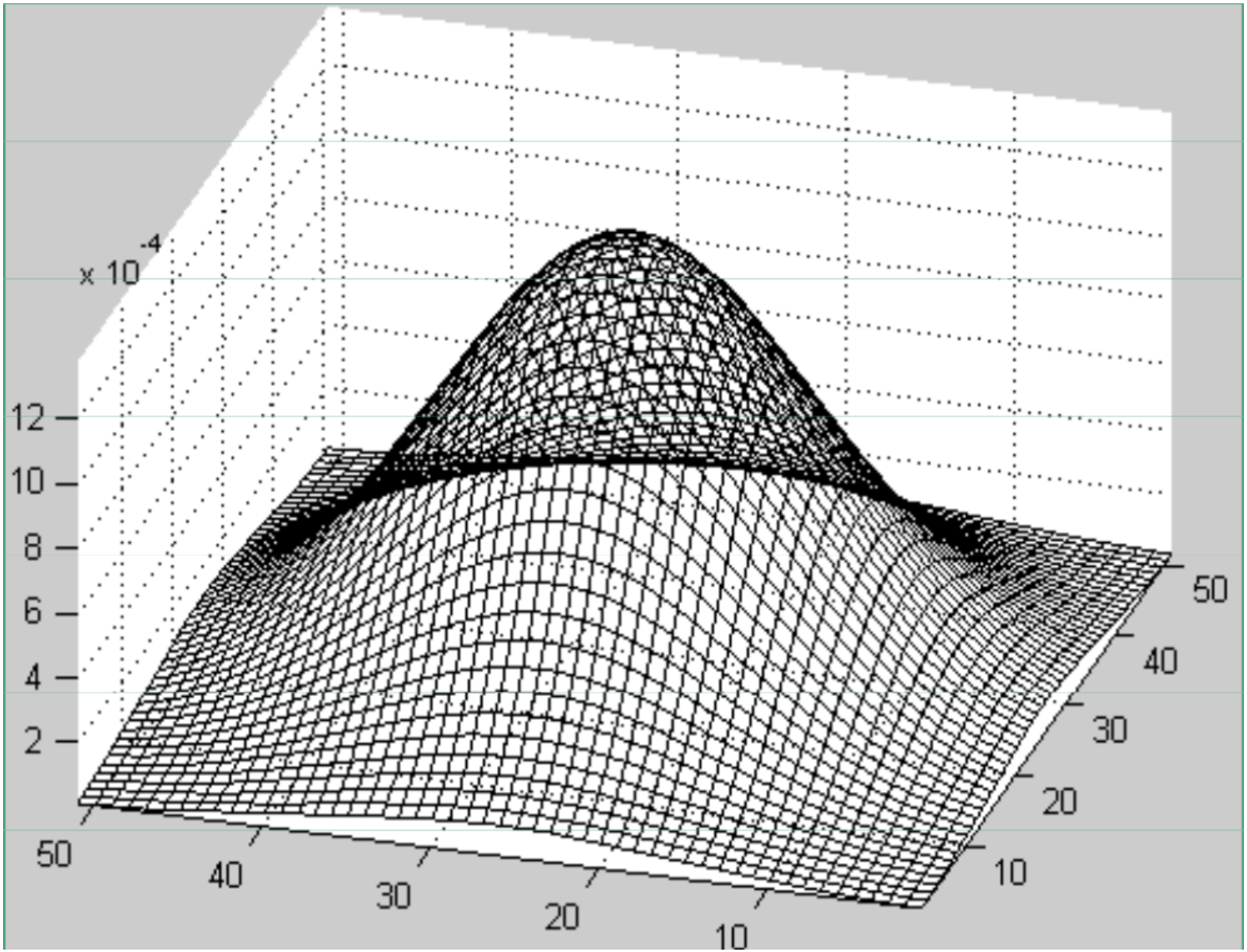
masila@iiitdm.ac.in

Department of Computer Science and Engineering
IIITDM Kancheepuram
Chennai-127

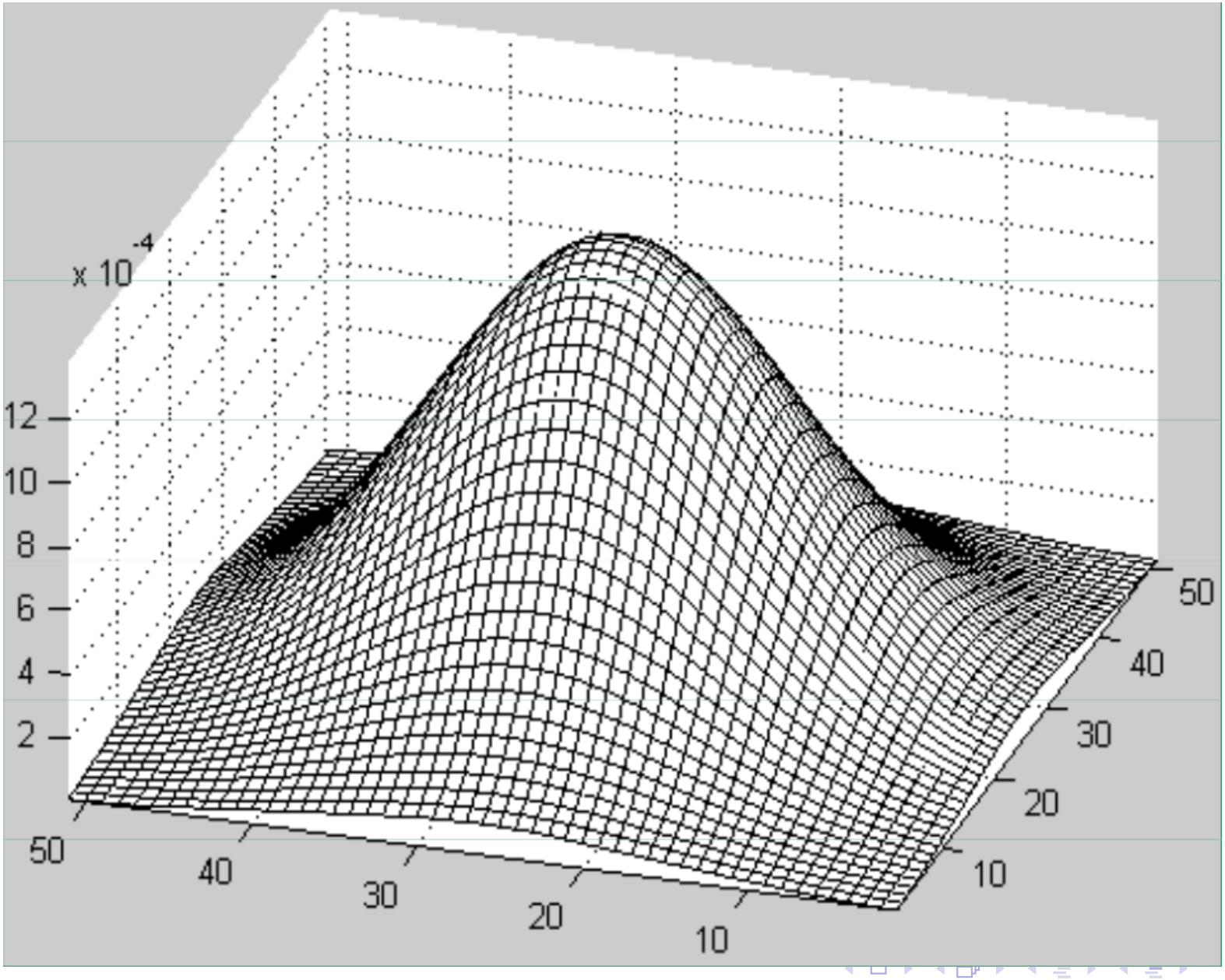
Overview

1. Need for Visible Surface Detection
2. Back face Culling or removal
3. Depth-buffer/Z-buffer Method
4. SCAN-LINE Algorithms for VSD
5. Depth-Sorting or Painter's Algorithm
6. Ray Tracing

Visible Vs Hidden surfaces of 3D Object

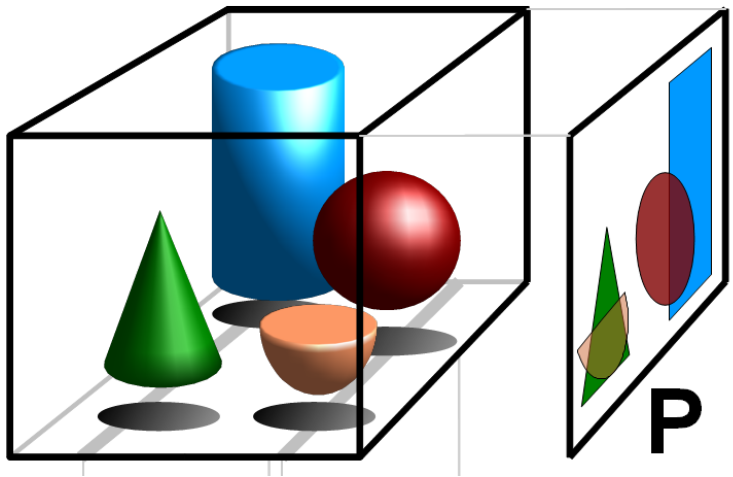


Visible Vs Hidden surfaces of 3D Object



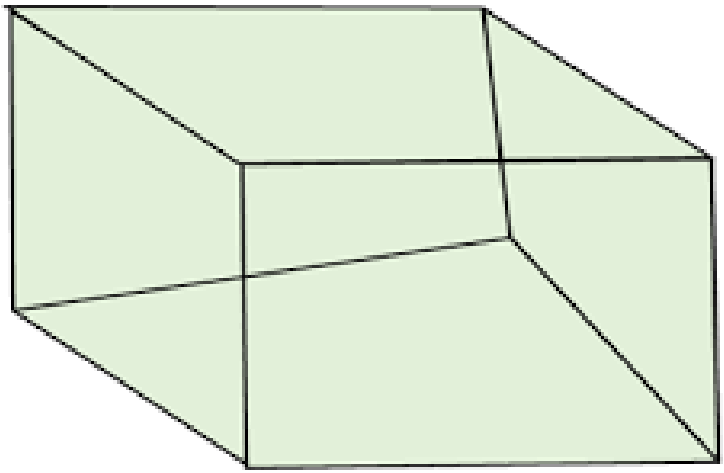
Need for Visible Surface Detection

- Projection along X direction, without removing the hidden surface
- **Hidden Surface:** For a person standing on X, and looks towards the -ve X, surface of only one object is visible(not the overlapping of two surfaces)

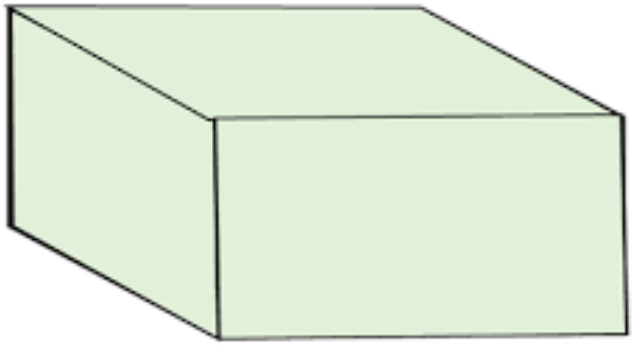


Need for Visible Surface Detection (cont.)

3D object vs visible portion of the 3D object



Object with hidden line



Object when hidden lines removed

Visible Surface Detection Problem

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:
  - Surface for rendering
  - Line drawing

Coherence Properties - 1/2

- 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.

Coherence Properties - 2/2

- 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 animation sequence are quite similar (small changes in object j and viewpoint).

Methods for Visible Surface Detection

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

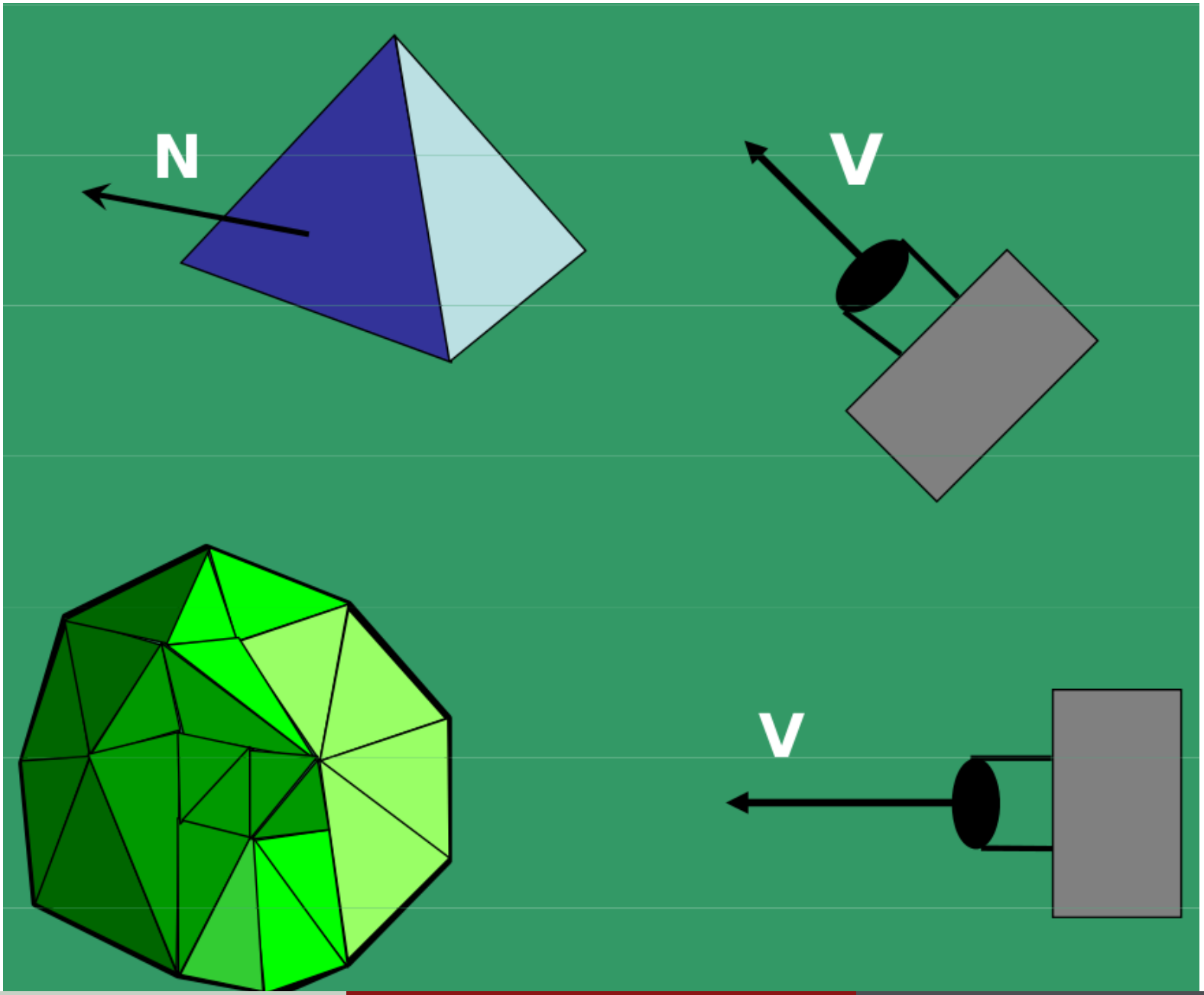
Visible surface detection techniques are 3D versions of sorting algorithms – basically compare depth.

Back face Culling or removal

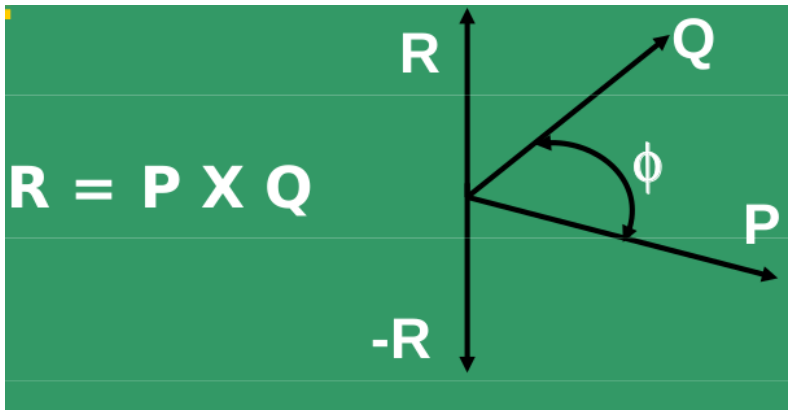
A Polygon (in 3D) is a back face if $\vec{V} \cdot \vec{N} > 0$
Let $V = (0, 0, V_z)$ and $N = A\hat{i} + B\hat{j} + C\hat{K}$
Then $\vec{V} \cdot \vec{N} = \vec{V_z} \cdot \vec{C}$
Let $\vec{V_z}$ be +ve (view along +ve Z-direction), the sign of C.
Condition of back face is thus:

$$sgn(C) >= 0$$

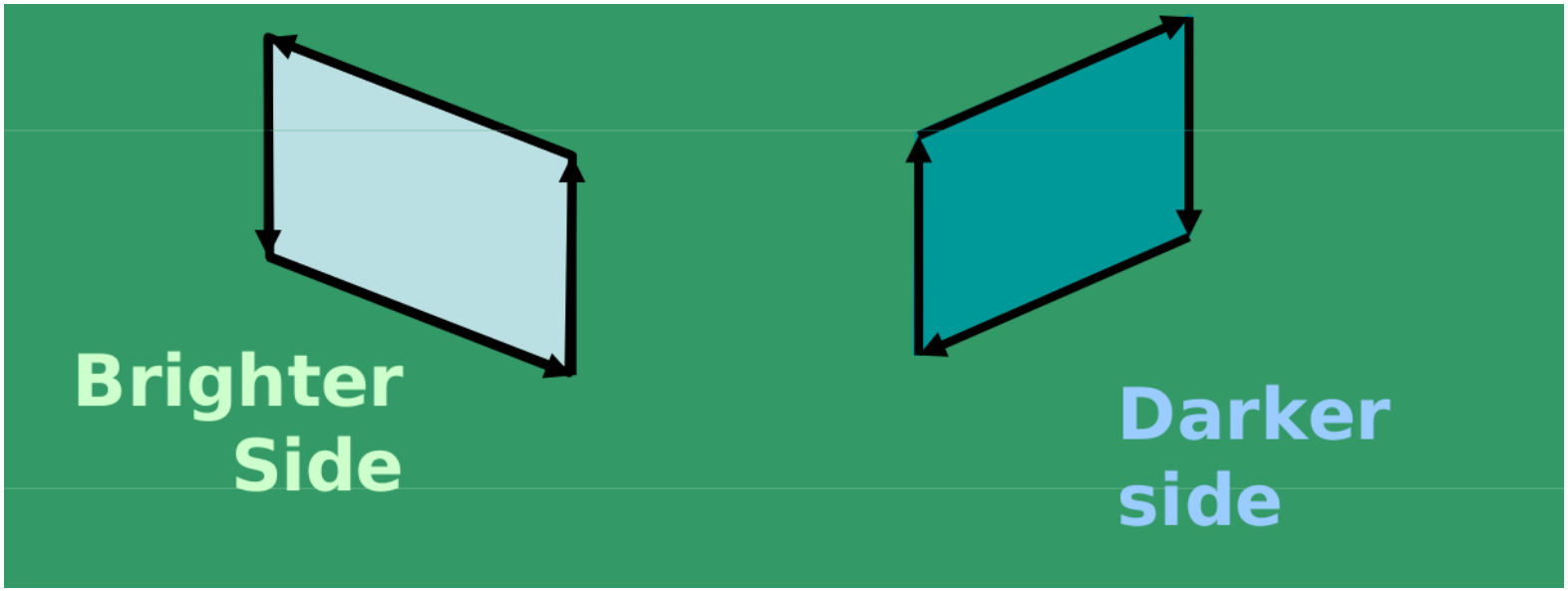
What happens if $\vec{V} \cdot \vec{N} > 0$?


N
V
V

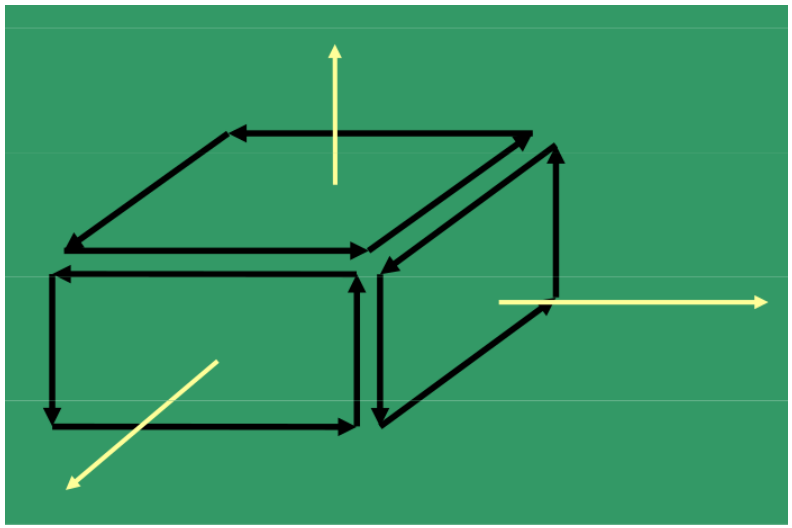
How to get normal vector ($\vec{N}$) for a 3D surface, polygo ?



Order of vertices for calculation of $\vec{N}$ – w.r.t the front side of the surface.



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



How to find sgn(C), where $N = A\hat{i} + B\hat{j} + C\hat{K}$

- Choose any two vectors (edges) V_1 and V_2, to obtain the normal to the surface, using $V_1 \times V_2 = |V_1||V_2|sin(\theta)\hat{N}$
- The direction of the normal to the surface of the polygon is the same as that of $V_1 \times V_2$
- Any risks in such a case, if we randomly choose any two vectors?
- Better Solution to obtain the direction rations of Normal to the surface : Use Newell formula

- The direction ratios (a, b, c) of surface of the polygon with n vertices $(X_i, Y_i, Z_i); i = 1, 2, ..., n$.
  Compute:

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

$$b = \sum_{i=1}^{n} (Z_i - Z_j)(X_i + X_j)$$

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

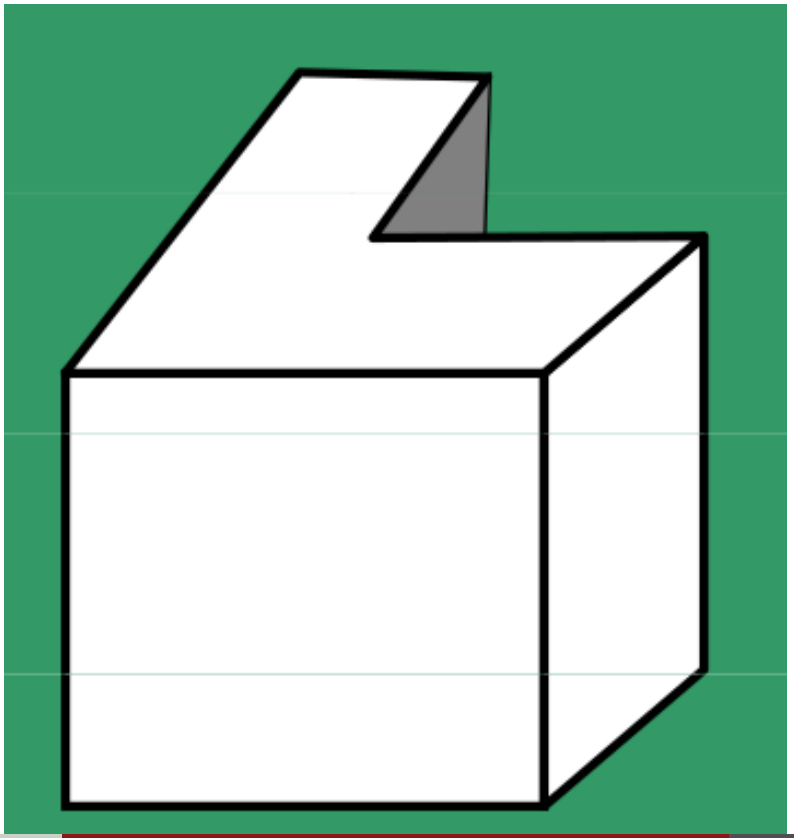
  where $j = i + 1$. If $i = n, j = 1$.

- The polygon is a back face if $c \geq 0$.
- If the view direction is along the –ve Z-direction, the above condition becomes: $c < 0$.

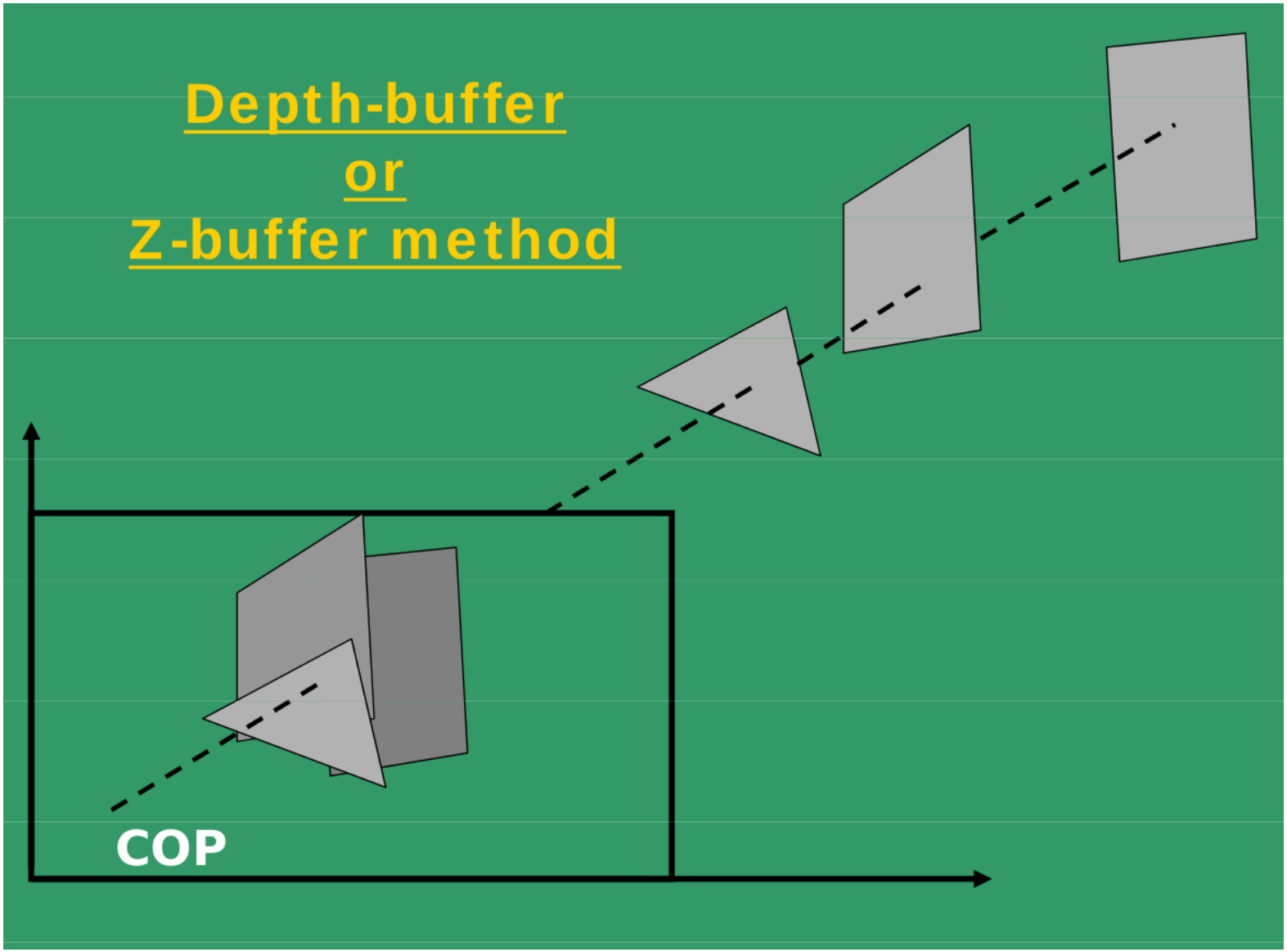
Drawbacks of back face culling:

- 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 surfaces are eliminated.



Depth-buffer/Z-buffer Method



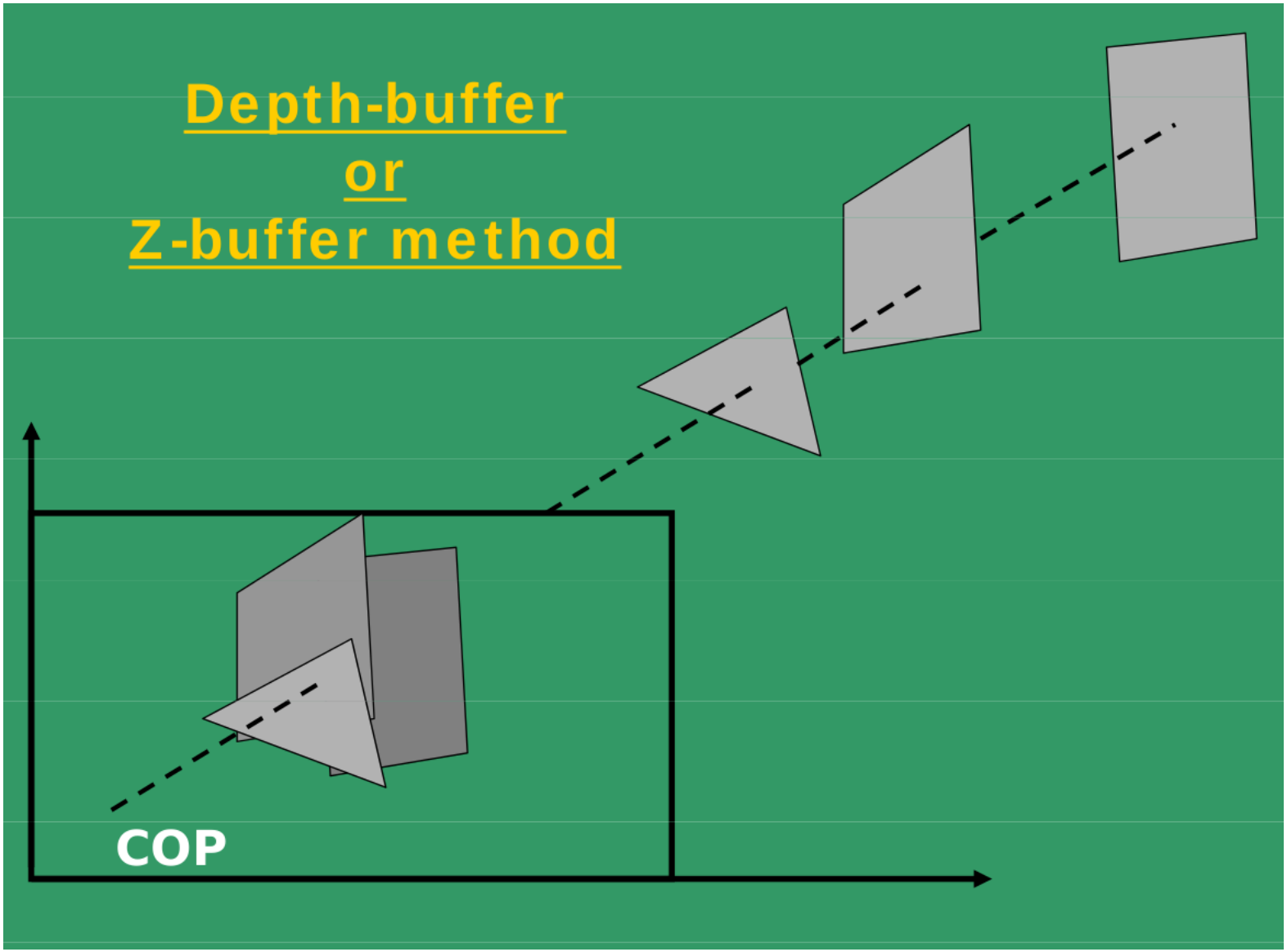
Each (X, Y, Z) point on a polygon surface, corresponds to the orthographic projection point (X, Y) on the view plane.

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

Two buffers are used in Z-buffer method:

- Depth (Z) buffer: To store the depth values for each (X, Y) position, as surfaces are processed.
- Refresh Buffer: To store the intensity value at each position (X, Y).

Depth-buffer/Z-buffer Method



Z-buffer Algorithm when Looking towards +ve Z-axis

1. Initialize:
   $depth(X, Y) = Z_{max}$
   $refresh(X, Y) = I_B$
   $\forall(X, Y)\ I_B =$ background intensity
2. For each position on each polygon surface:
   2.1 Calculate depth Z for each position (X, Y) on the polygon.
   2.2 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:

Equation of the surface:

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

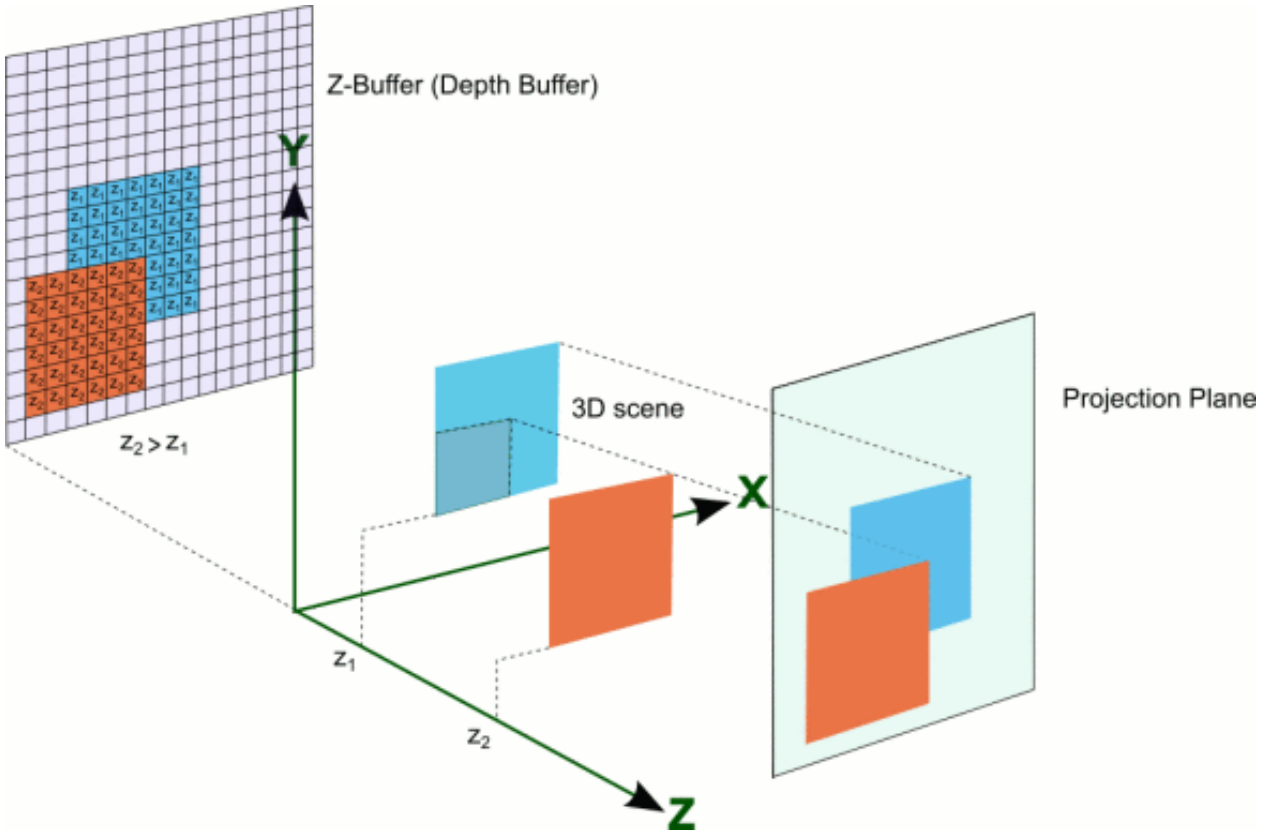
$$Z = \frac{-AX - BY - D}{C}$$

Equation for Z at the next scan line:

Remember scanline/Polyfill algorithm? Using edge coherence, we get for the next scanline $(Y + 1)$:

$$X' = X - 1/m$$

Z-Buffer Algorithm when looking towards -ve Z-axis

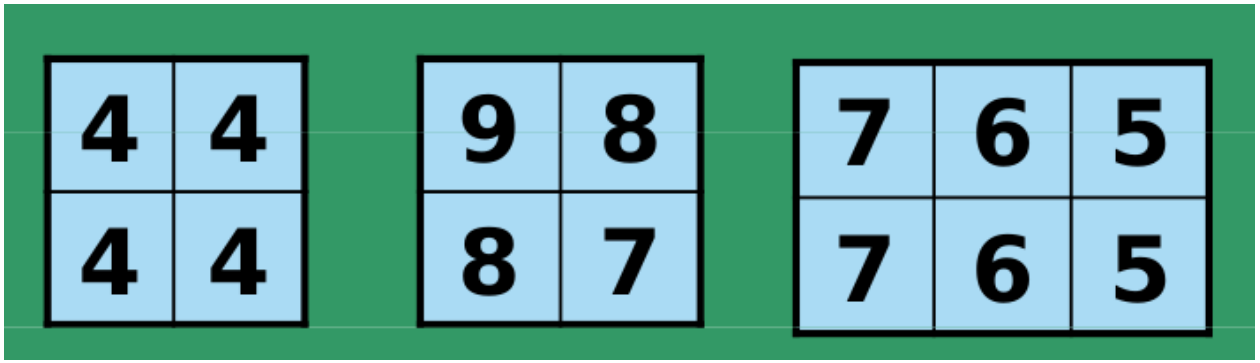


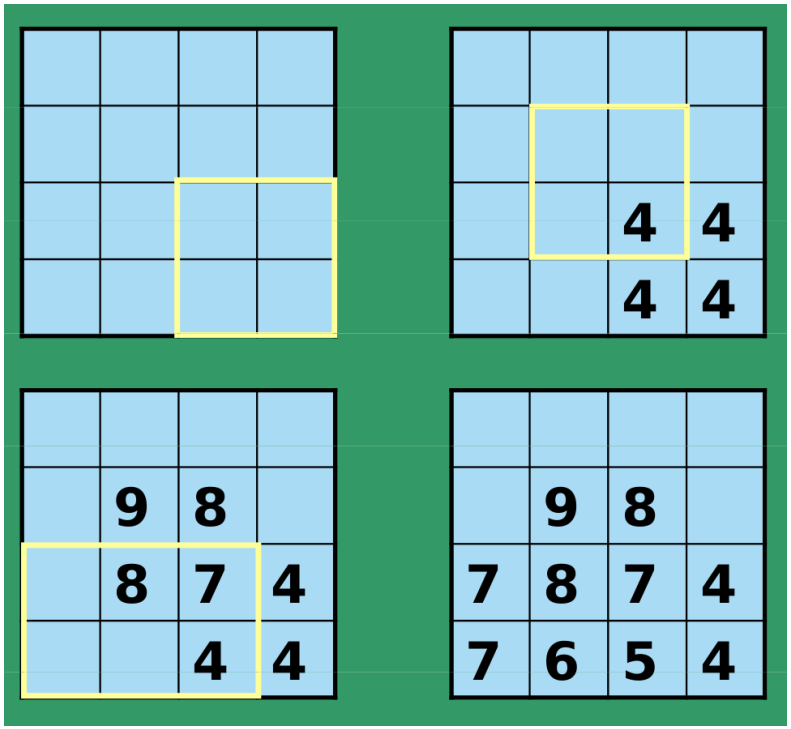
Z-Buffer Algorithm when looking towards -ve Z-axis

- For all (x, y):
  $depth(x, y) = -\infty$
  $refresh(x, y) = I_B$
- For each polygon P,
  for each position (x, y) on polygon P
  calculate depth z
- if $z > depth(x, y)$, then
  $depth(x, y) = z$
  $refresh(x, y) = I_p(x, y)$

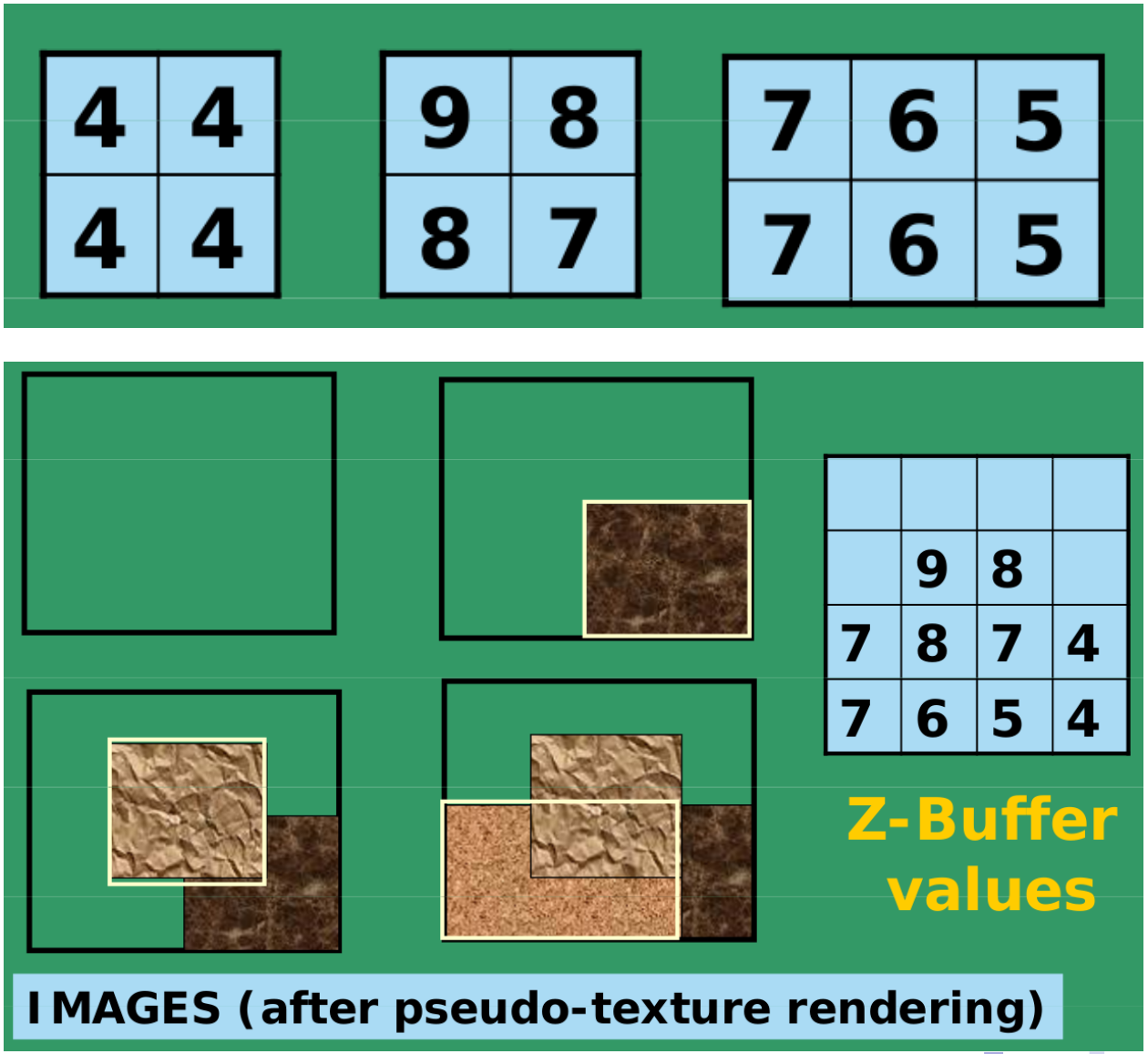
An Example

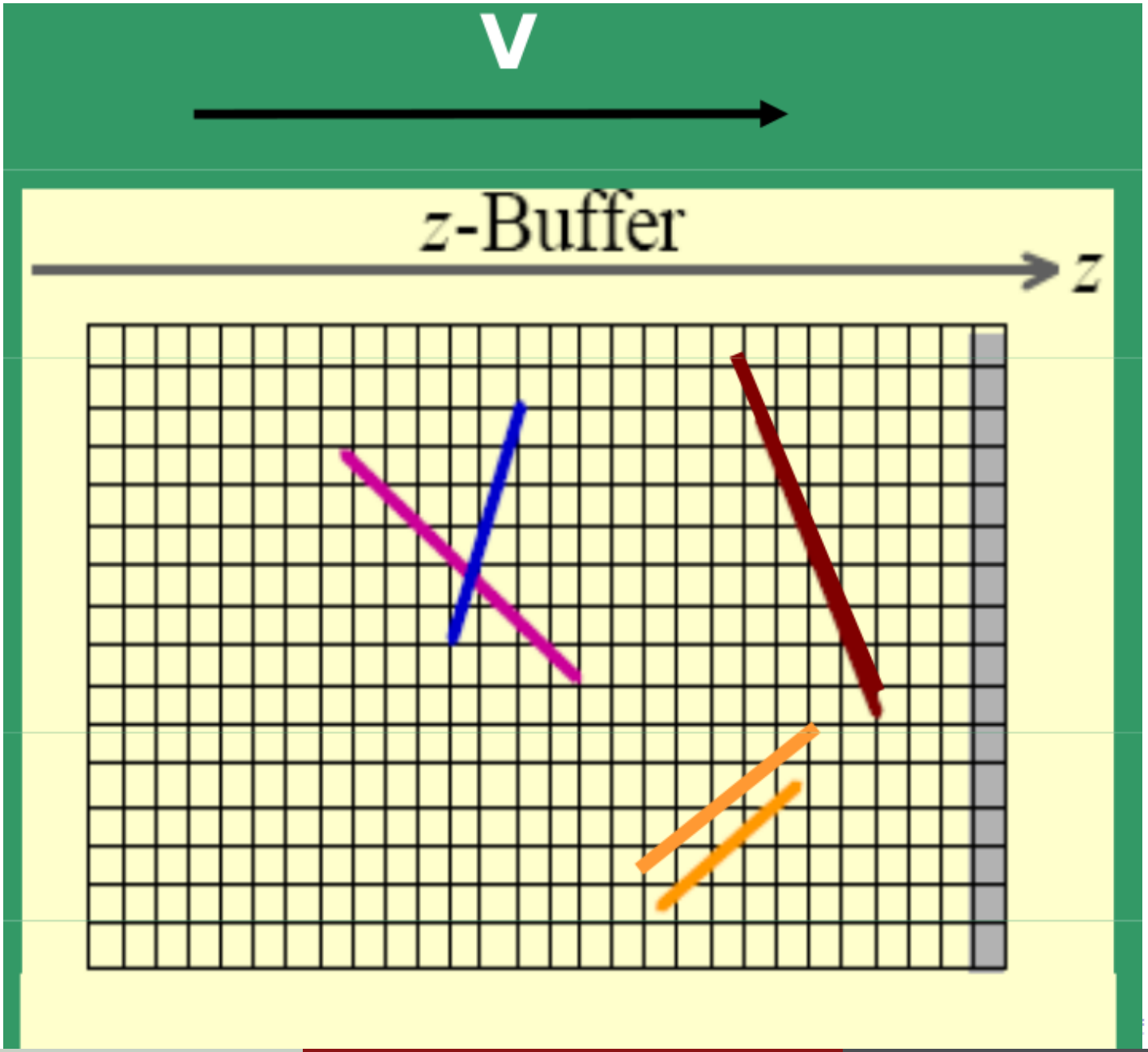
Z-values of the co-ordinates:

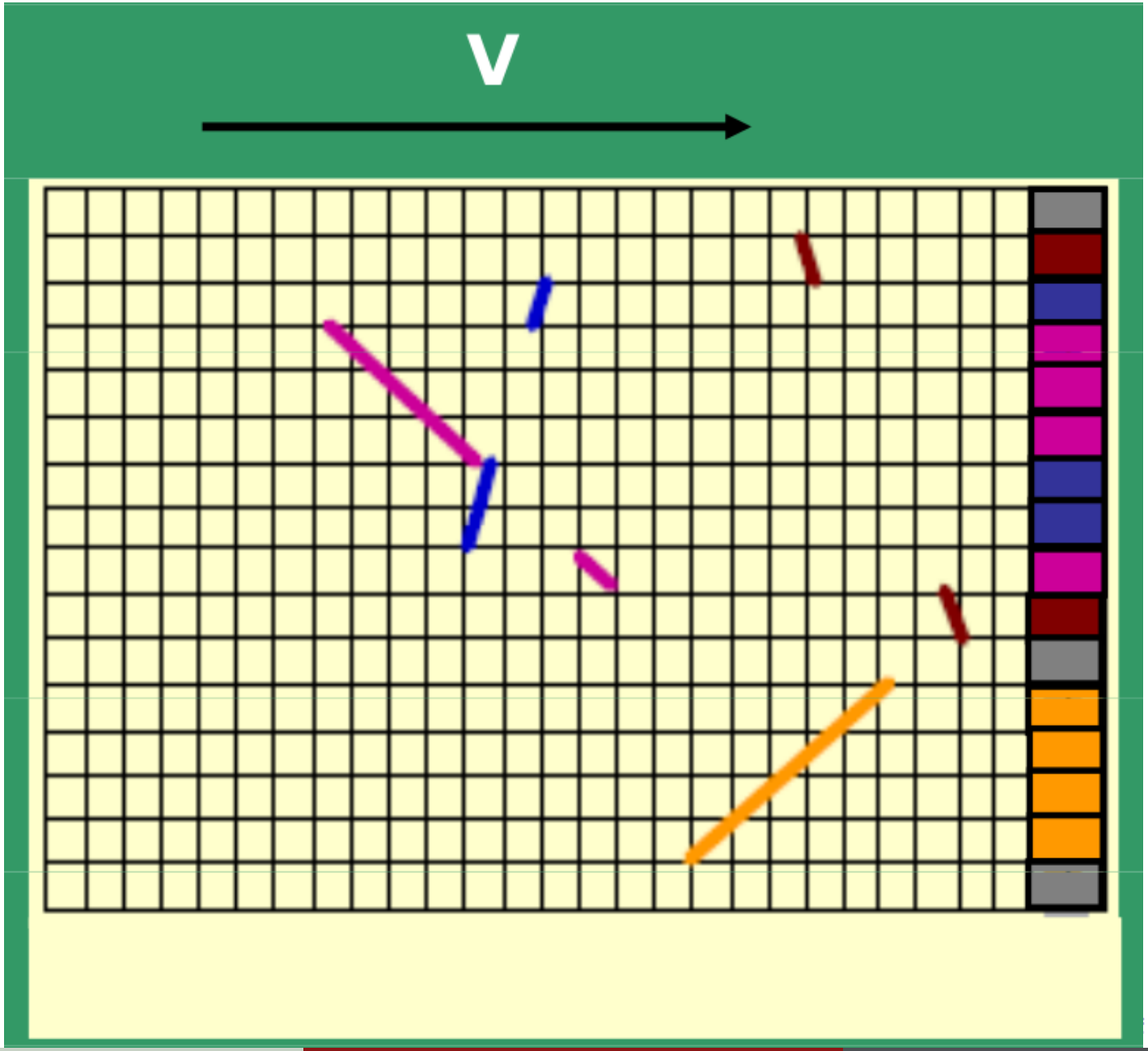




Z-values of the co-ordinates:




V
z-Buffer
z


V

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)
- 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 are ordered by increasing X-coordinate of their endpoint.

Structure of each entry in ET:

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

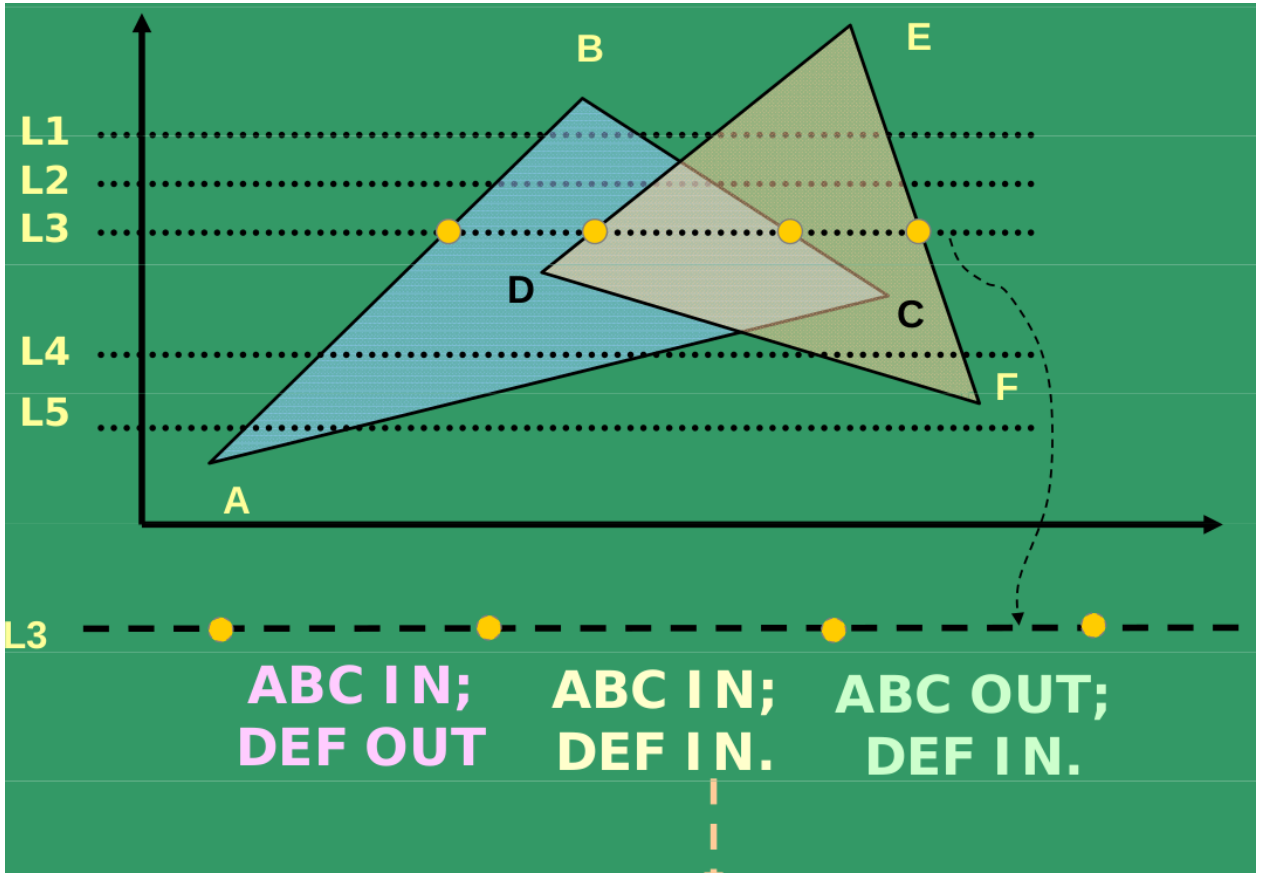
X	Y_{max}	ΔX	ID	(next pointer)

Structure of each entry in PT:

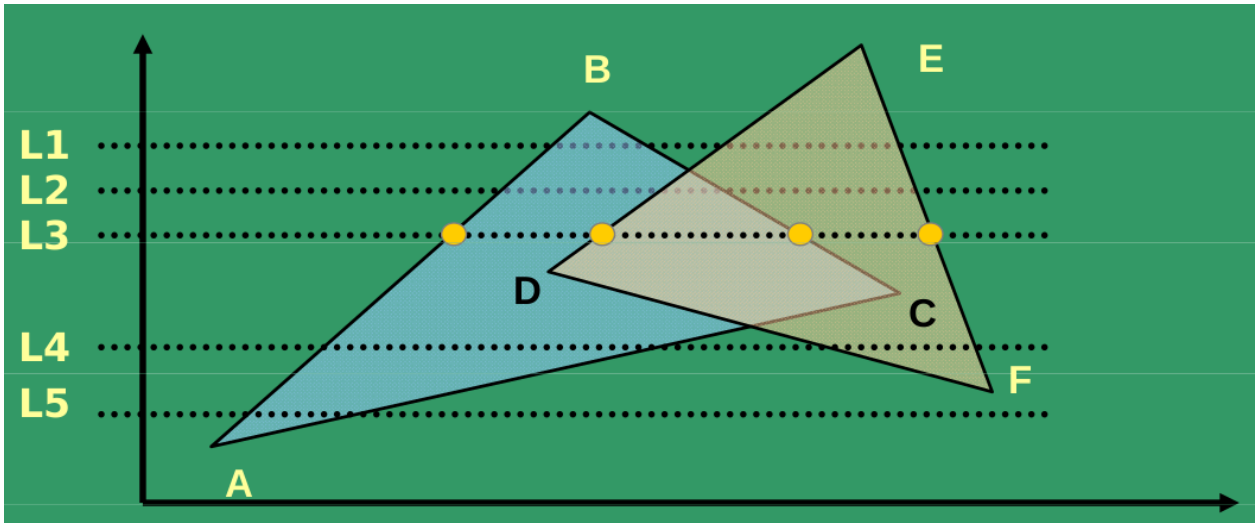
- Coefficients of the plane equations
- Shading or color information of the polygon
- Flag (IN/OUT), initialized to 'false'

ID	Plane Coeffs.	Shading Info.	IN/OUT

Take Adjacent (not successive) pairs of intersections to fill; FILL (within pair) if POLY_FLAG is set to IN.



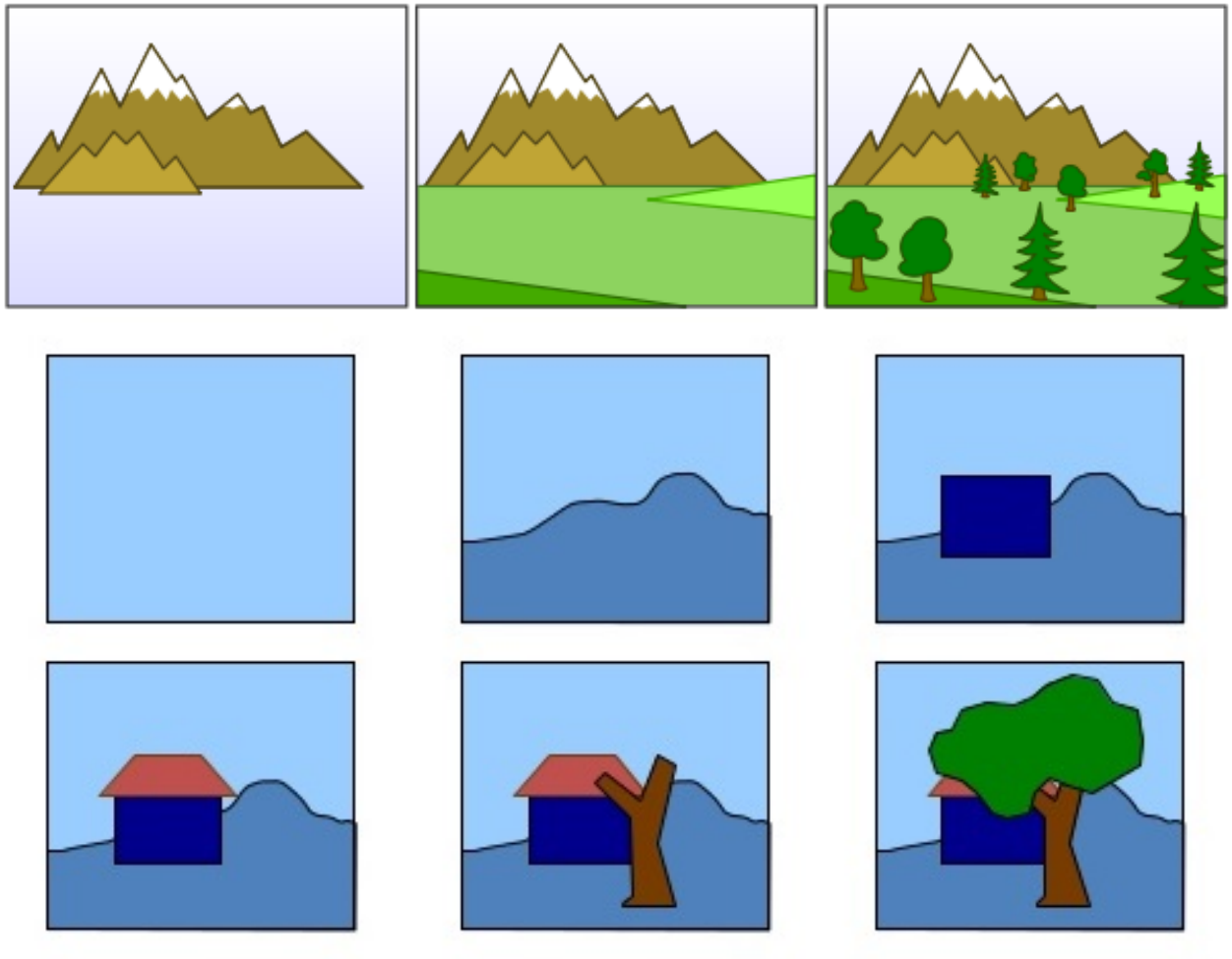
Compare Z values in this region to find the visible Z value.



AET Contents				
Scanline	Entries			
L5	AB	CA		
L4	AB	CA	FD	EF
L3, L2	AB	DE	BC	EF
L1	AB	BC	DE	EF

Depth-Sorting or Painter's Algorithm

How does painter paint -Mimic



Depth-Sorting or Painter's Algorithm

How does painter paint -Mimic

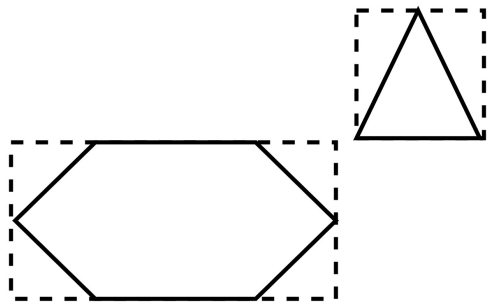
Distant Background is painted first

Objects closer than the background are painted

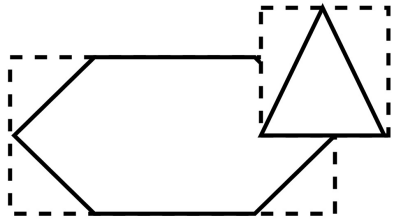
Finally, Nearest Objects are painted

Depth-Sorting or Painter's Algorithm

How does painter paint -Mimic



Rectangles are not overlapping

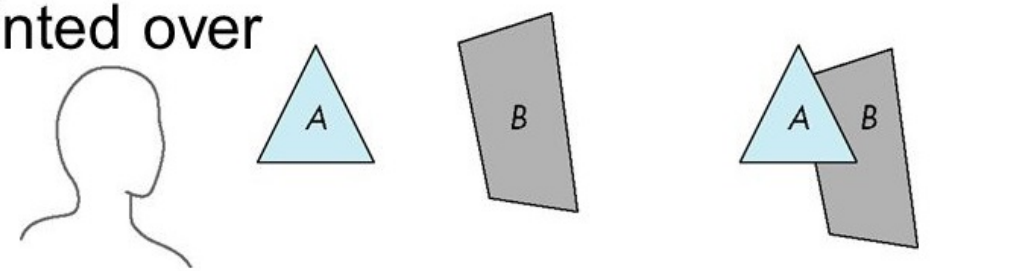


Rectangles are overlapping

Depth-Sorting or Painter's Algorithm

How does painter paint -Mimic

- Render polygons a back to front order so that polygons behind others are simply painted over



B behind A as seen by viewer

Fill B then A

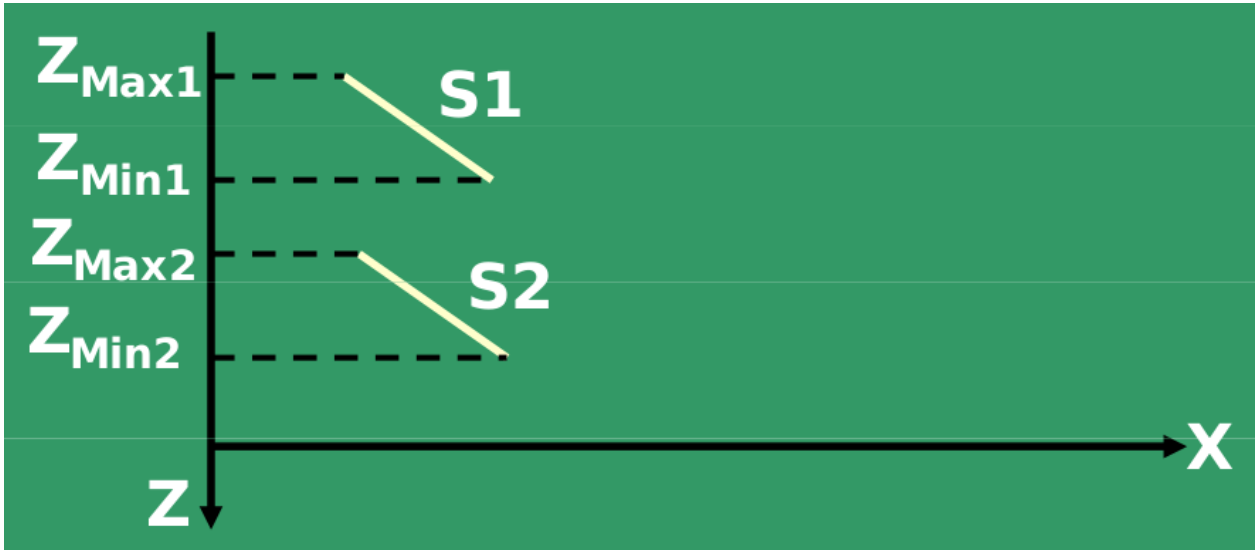
Depth-Sorting or Painter's Algorithm

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

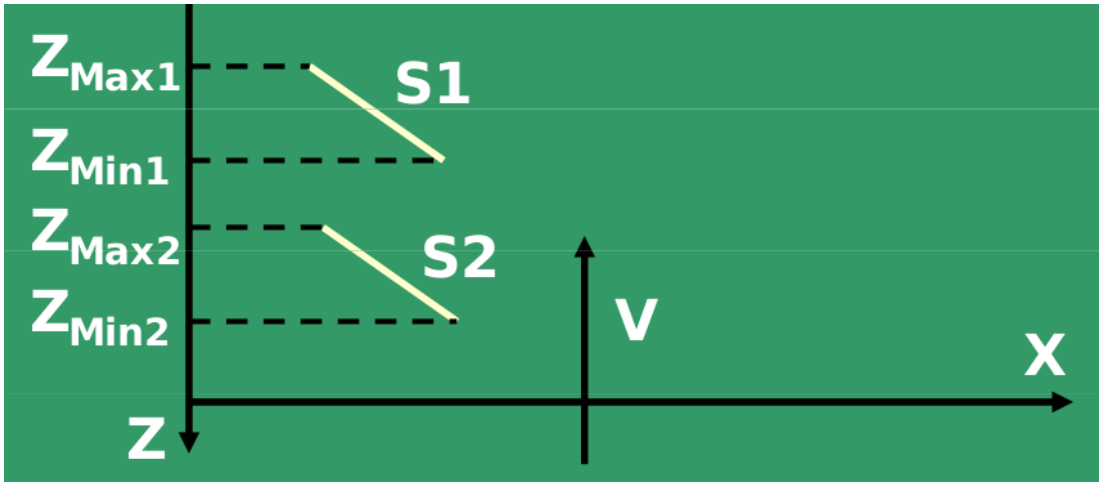
Broad Steps:

- 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.

The four tests in the Painter's Algorithm

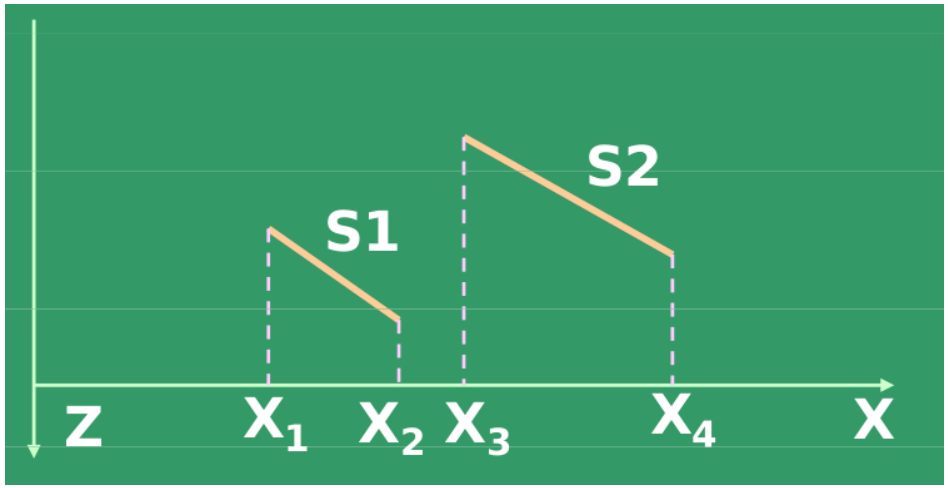
1. The boundary rectangles in the X-Y plane for the two surfaces do not overlap.
2. Surface S is completely behind the overlapping surface relative to the viewing position.
3. The overlapping surface is completely in front of S relative to the viewing position.
4. The projections of the two surfaces onto the view plane do not overlap.

The 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 if 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.

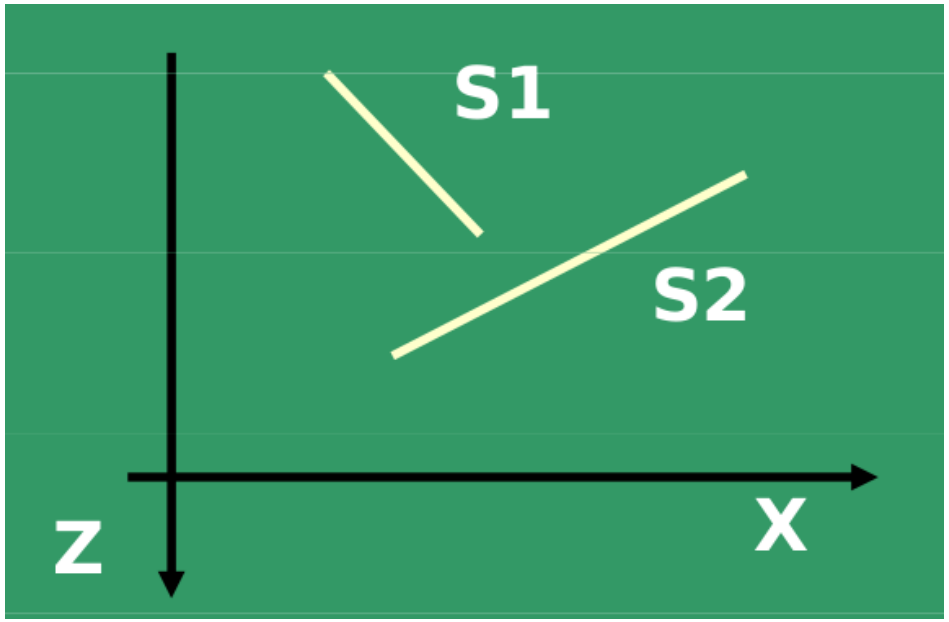


Figure 2: S1 is completely behind/inside the overlapping surface S2

Test 3

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

S1 is not completely behind S2. So, Test #2 fails.

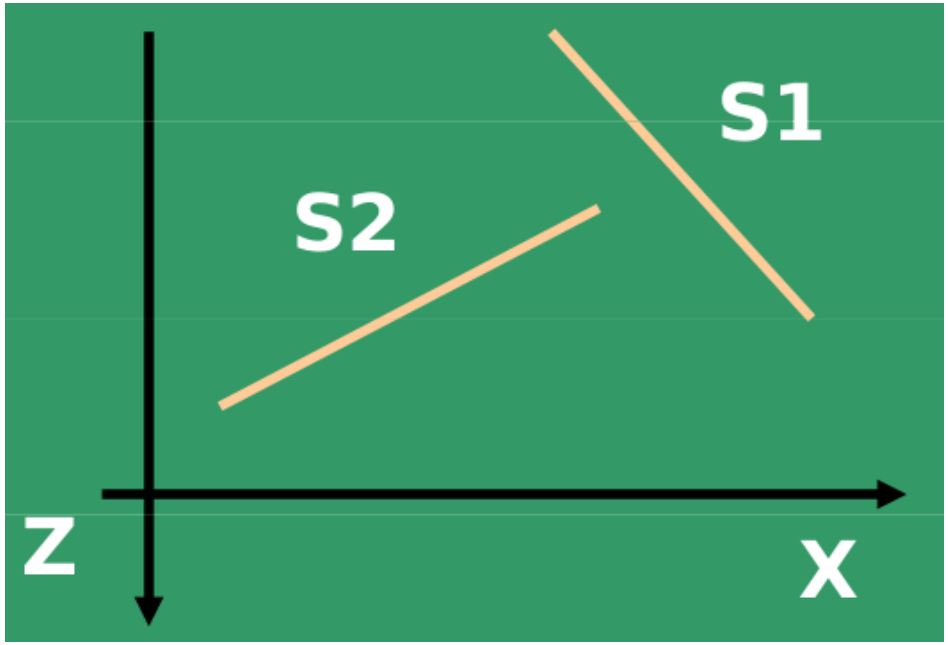


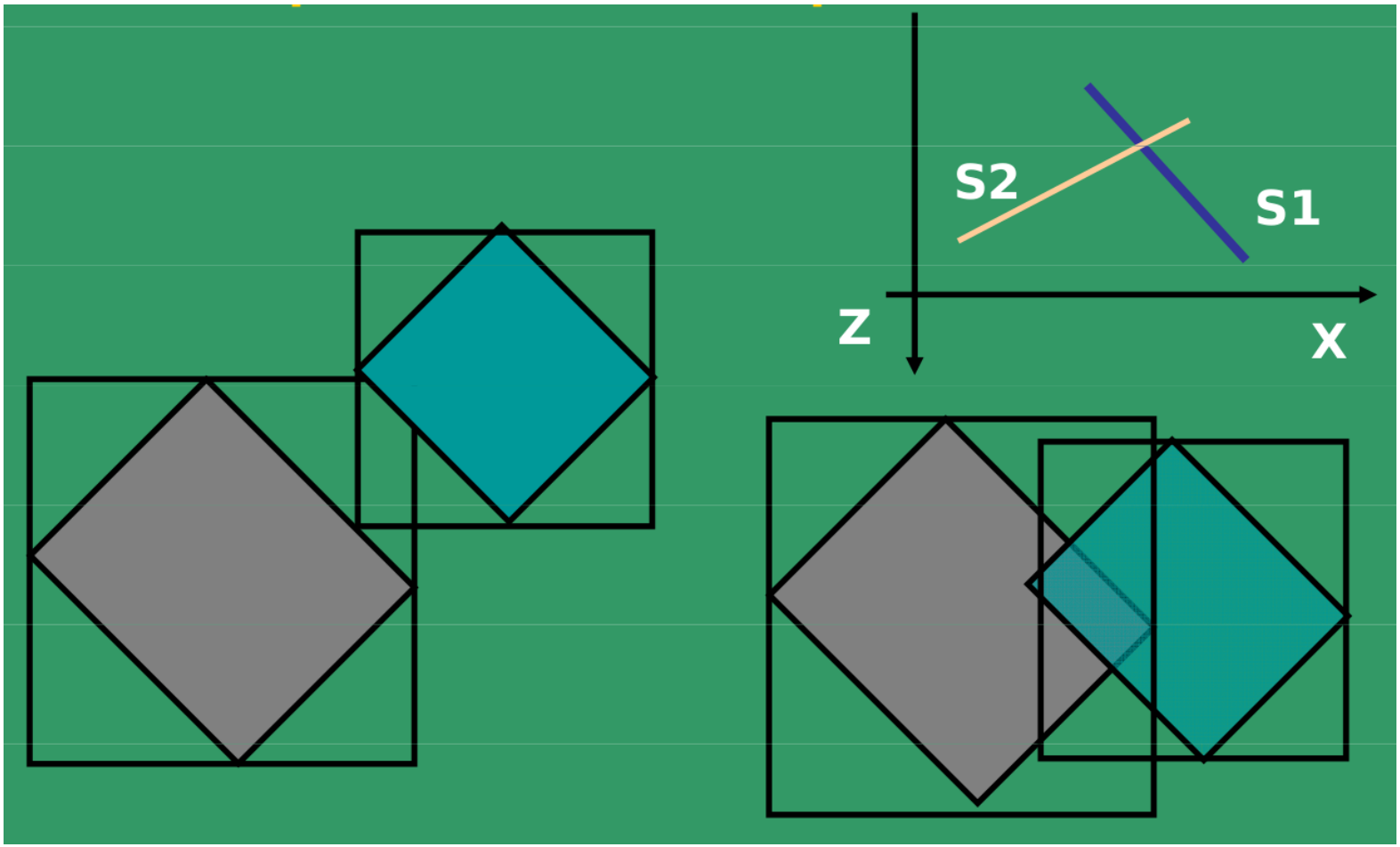
Figure 3: S1 is completely behind/inside the overlapping surface S2

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 these conditions?

1. Set the plane equation of S2, such that the surface S2 is towards the viewing position.
2. Substitute the coordinates of all vertices of S1 into the plane equation of S2 and check for the sign.
3. If all vertices of S1 are inside S2, then S1 is behind S2. (Fig. 1).
4. 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.



The four tests in the Painter's Algorithm - Revisited

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.

The tests must be performed in order, as specified.

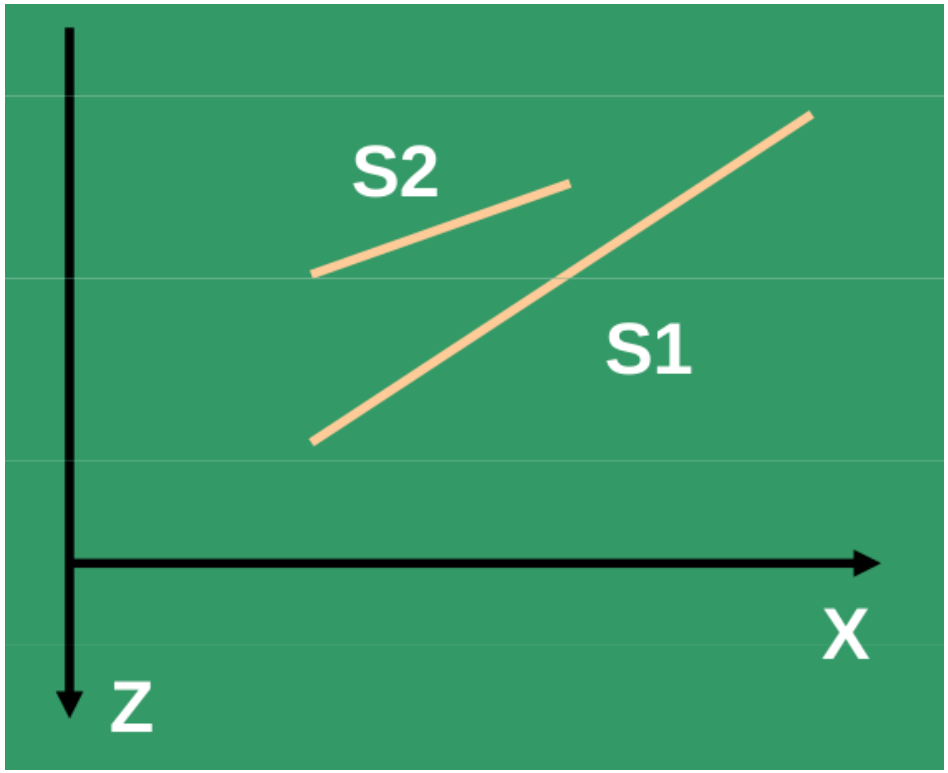
Condition to RE-ORDER the surfaces:
If any one of the 4 tests is TRUE, we proceed to the next overlapping surface. i.e. If all overlapping surfaces pass at least one of the tests, none of them is behind S.

No re-ordering is required, and then S is scan converted.

RE ORDERing is required if all the four tests fail.

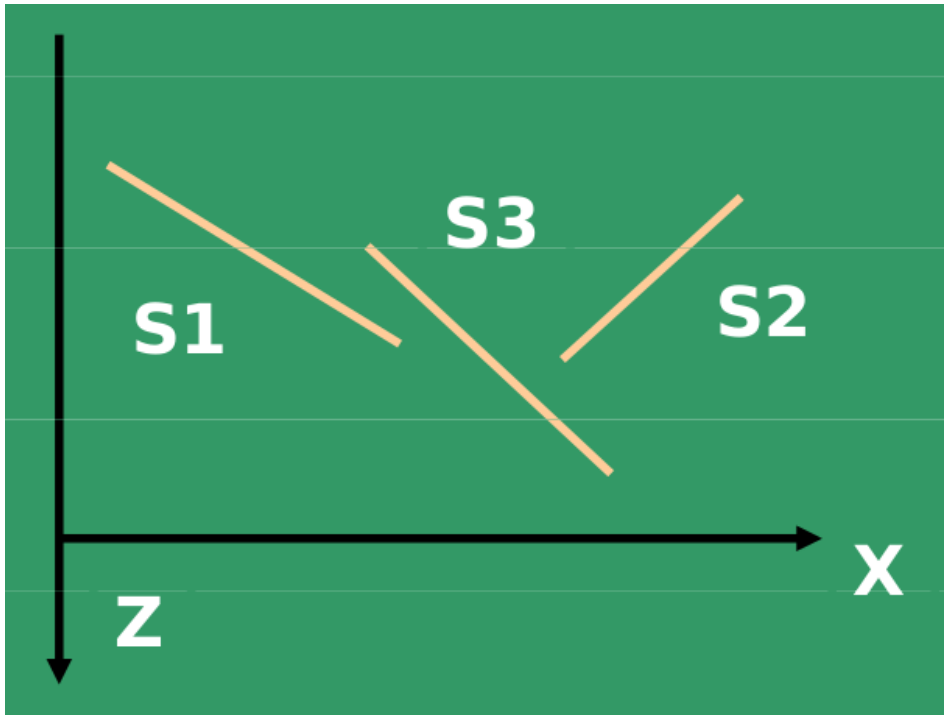
Case Studies: Examples of Painter's Algorithm

Case Study 1:

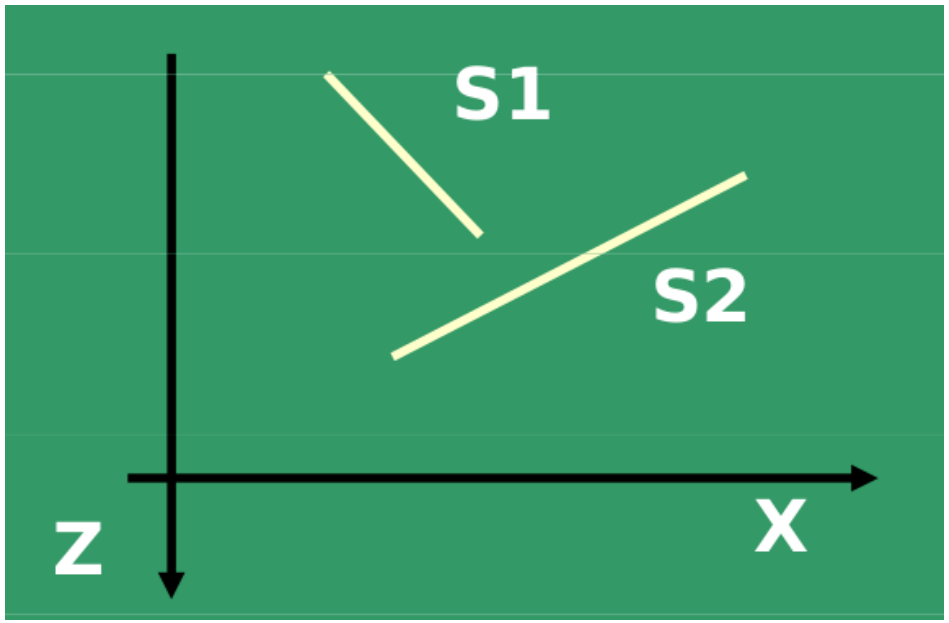


Initial Order: $S1 \rightarrow S2$
Change the Order: $S2 \rightarrow S1$

Case Study 2: Initial Order: $S1 \rightarrow S2 \rightarrow S3$
$S1 \rightarrow S2$. Test 1 passed.

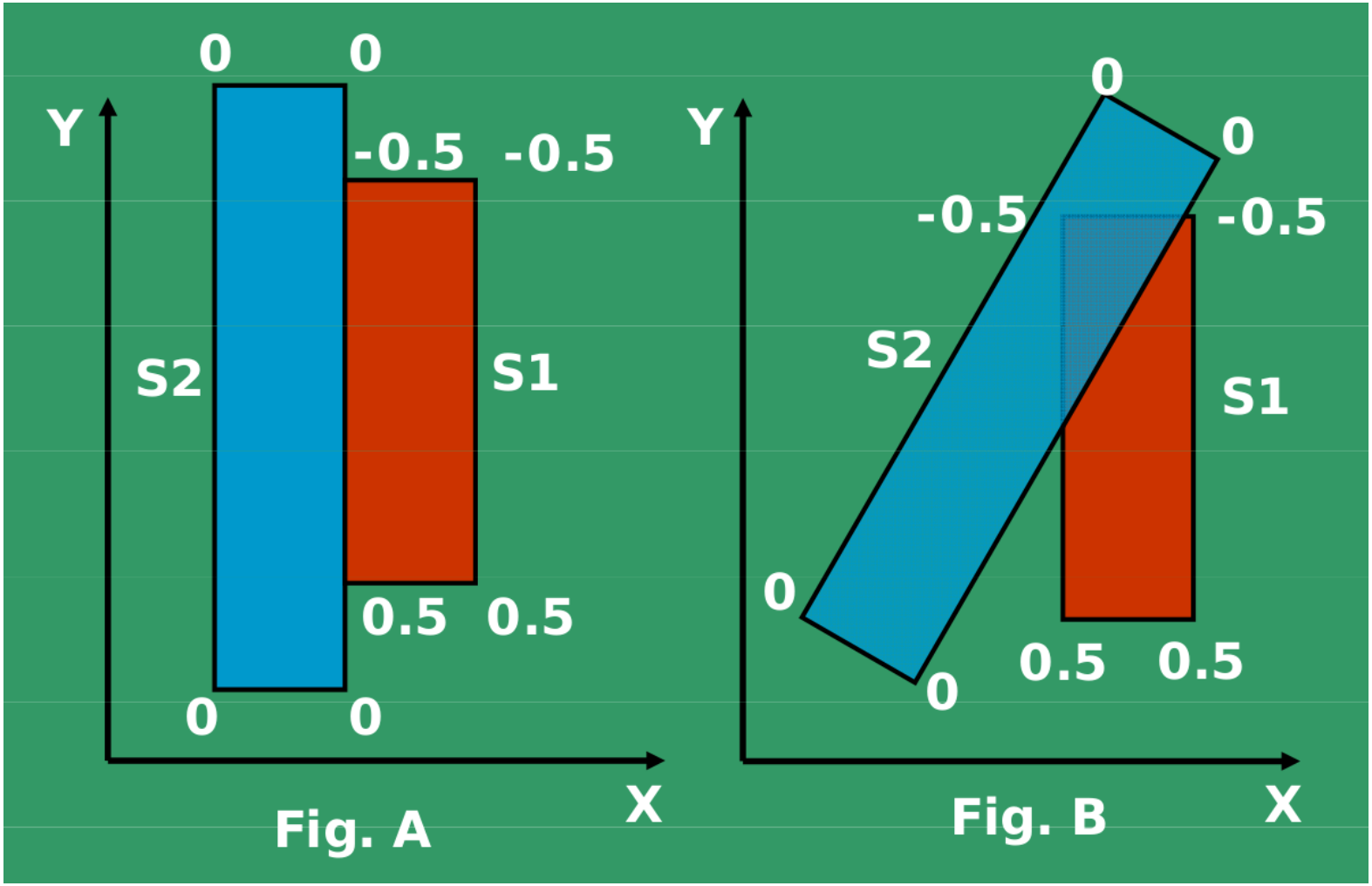


Check $S1$, $S3$. All tests fail.
Interchange. $S3 \rightarrow S2 \rightarrow S1$. Check $S2$ and $S3$. All tests fail again.
Correct Order: $S2 \rightarrow S3 \rightarrow S1$.



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.



What happens in these cases?

VSD-Tracing

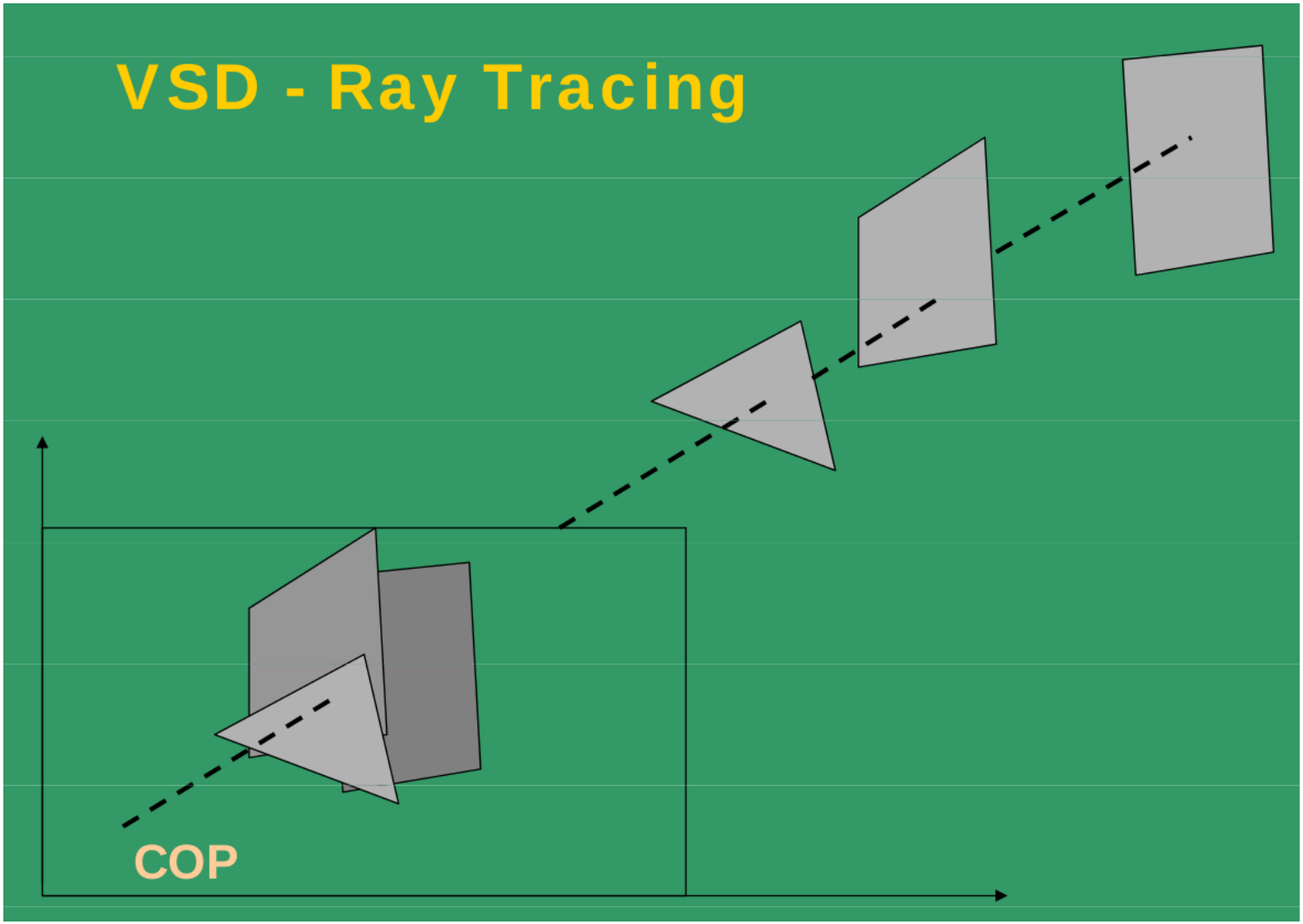
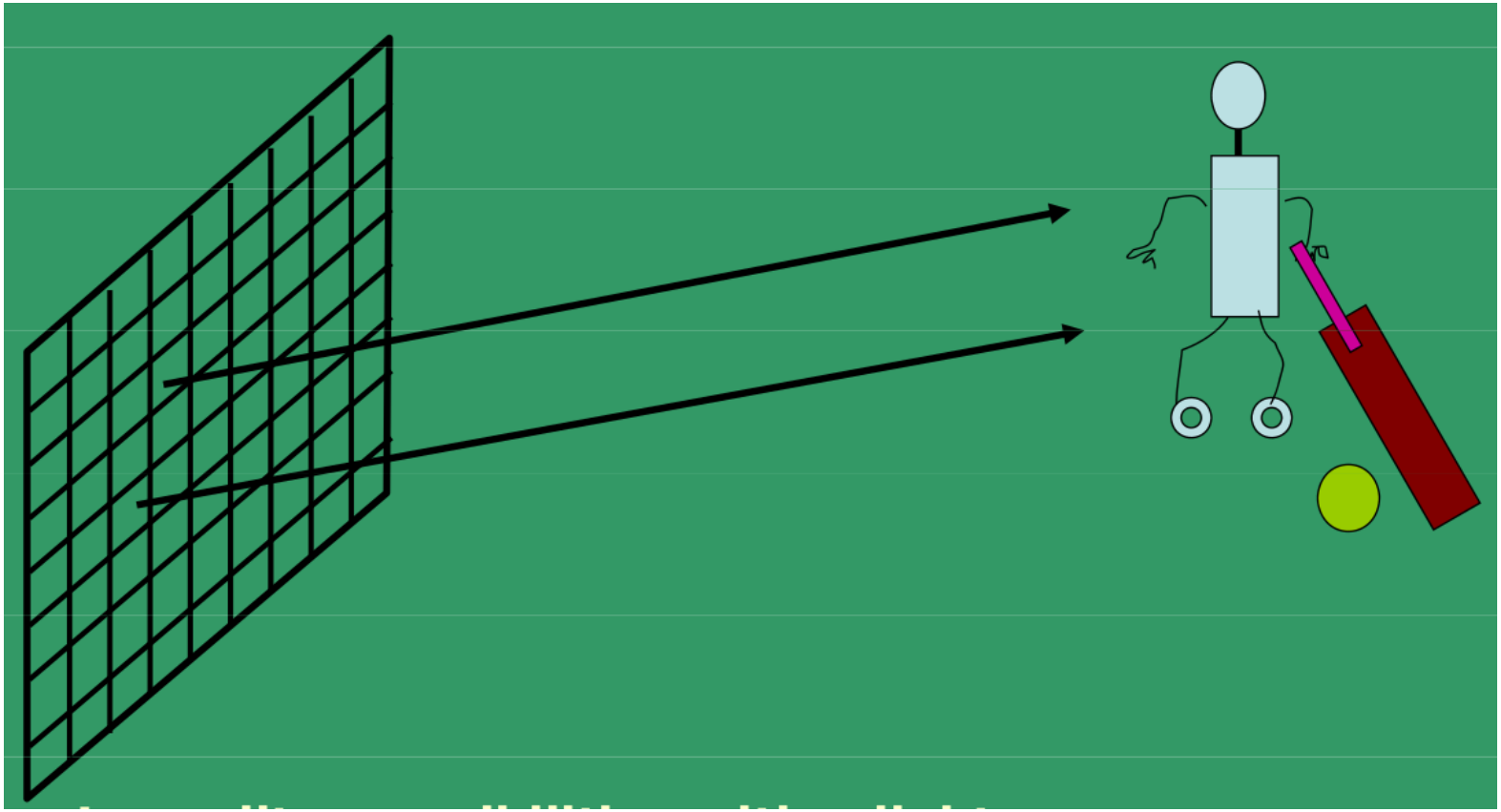


Figure for Illustrating VSD-Ray Tracing



In reality, possibilities with a light ray: absorption, reflection, refraction, scattering, fluorescence, chromatic aberration (error in lens).

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:

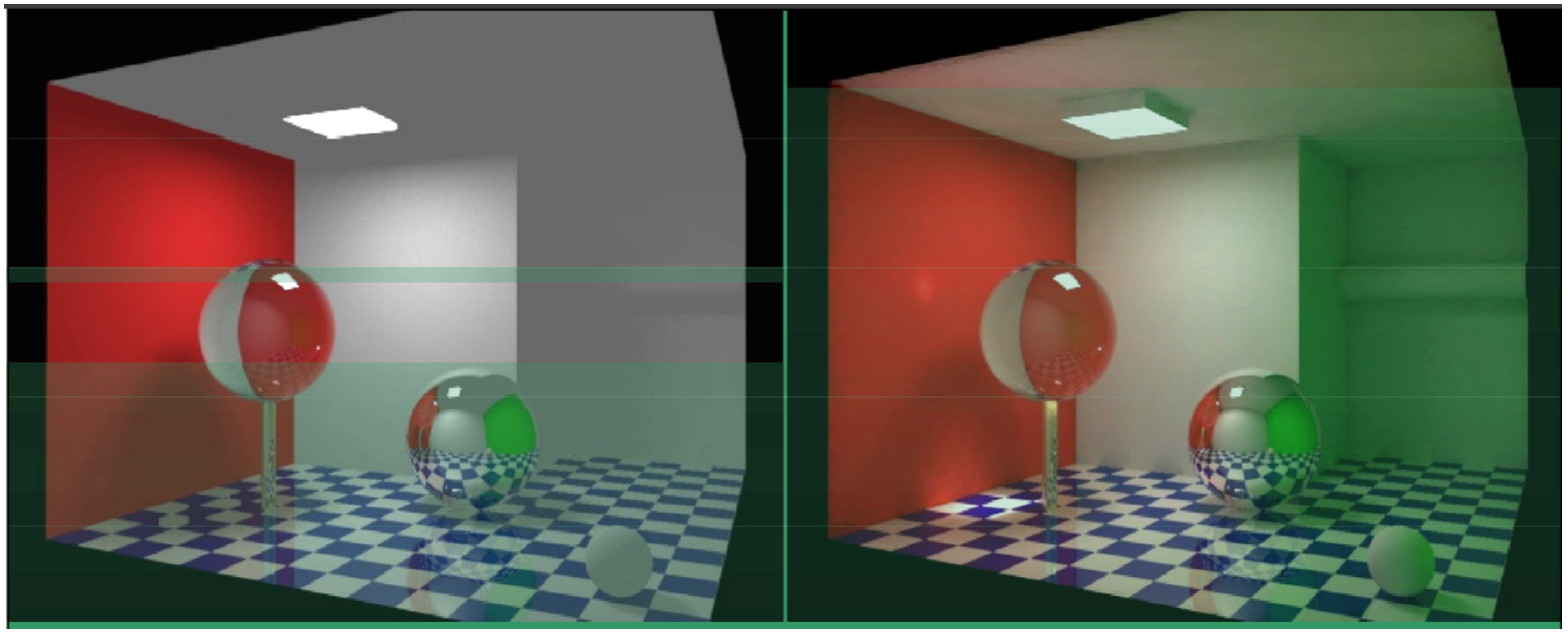
- Shoot ray from eye point through pixel (x,y) into scene
- Intersect with all surfaces, find first one the ray hits

For Ray-tracing (cont'd.:

- Line of Sight of each pixel is intersected with all surfaces
- Shade that point to compute the color of pixel (x,y)
- Consider only the closest surface for shading
- Based on optics of image formation, paths of light rays are traced
- Light rays are traced backward

For Ray-tracing (cont'd.:

- Suitable for complex curved surfaces
- Computationally expensive
- Need of an efficient ray-surface intersection technique
- Almost all visual effects can be generated
- Can be parallelized
- Has aliasing problems



Rendering with global illumination: Light is reflected by surfaces, and colored light transfers from one surface to another Color from the red wall and green wall (latter not visible) reflects onto other surfaces in the scene. Also notable is the caustic projected onto the red wall from light passing through the glass sphere.


Direct Illumination
Radiosity

Acknowledgements

- The slides have been adopted from NPTEL Lectures by Prof. Sukhendu Das. The due credits are acknowledged.

Thank You! :)