FLAME UNIVERSITY

EVERLASTING *learning*

FUNDAMENTALS OF COMPUTER GRAPHICS (CSIT304)

VISIBLE SURFACE DETECTION

CHIRANJOY CHATTOPADHYAY

Associate Professor,

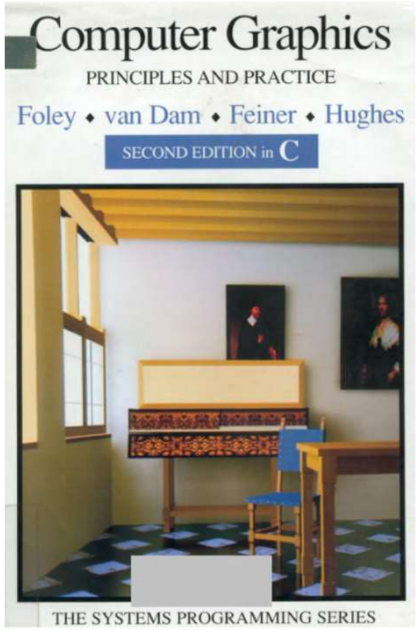
FLAME School of Computation and Data Science

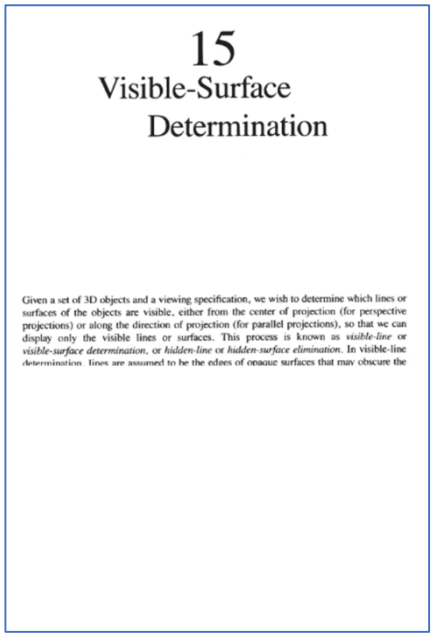
RECAP

- Solid Modeling
  - How to represent solid
  - Various approaches
  - Quad Tree. Octree
  - …

OBJECTIVE

- After completing this lecture, students will be able to
  - Explain different ways of detecting visible surfaces/ remove hidden surfaces
  - Solve numerical/ algorithmic problems




15
Visible-Surface Determination

Given a set of 3D objects and a viewing specification, we wish to determine which lines or surfaces of the objects are visible, either from the center of projection (for perspective projections) or along the direction of projection (for parallel projections), so that we can display only the visible lines or surfaces. This process is known as *visible-line* or *visible-surface determination*, or *hidden-line* or *hidden-surface elimination*. In visible-line determination, lines are assumed to be the edges of opaque surfaces that may obscure the

INTRODUCTION

- Visible surface detection (VSD) is a crucial step in computer graphics that involves identifying which parts of a 3D object are visible to the viewer and which are hidden.

- This process is necessary because computer graphics applications need to display only the visible parts of an object to create an accurate and realistic image.

- There are several algorithms that can be used for visible surface detection, including Z-buffering, scanline rendering, and ray tracing.

- The choice of algorithm depends on factors such as the complexity of the scene, the hardware available, and the desired level of accuracy and speed.

CLASSIFICATION OF APPROACHES OF VSD

Object-space

- Work directly with the 3D model of the objects in the scene.
- They **involve computing the visibility of each polygon in the scene**
- Compares visibility with the other polygons in the same object.
- Examples: **back-face culling and occlusion culling.**

Image-space

- Operate on the 2D image produced by projecting the 3D scene onto the screen.
- Computes the **visibility of each pixel in the image**
- Compares visibility with the other pixels in the same image.
- Examples: **depth-buffering and scanline rendering**.

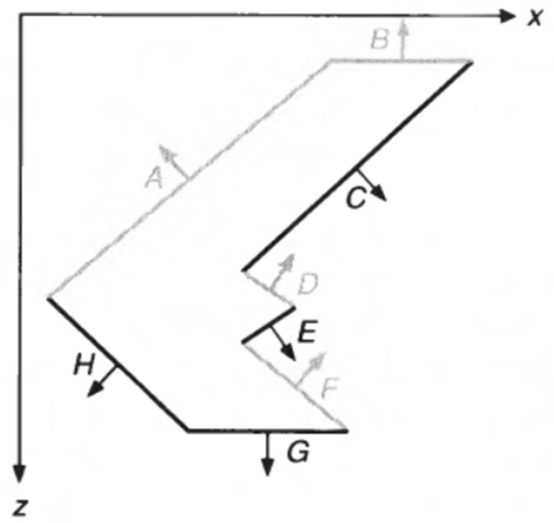
Hybrid

- Combine elements of both object-space and image-space methods.
- Preprocessing the 3D model to create a hierarchical data structure
- Used to quickly determine the visibility of objects and pixels in the scene.
- Examples: **BSP trees and octrees**.

OBJECT SPACE APPROACH

BACK FACE CULLING: INTRODUCTION (1)

- **Back-face culling** is a technique to improve rendering performance
- Eliminates the rendering of polygons that face away from the viewer
- "**culling**" refers to the process of removing or discarding elements of a scene that are not necessary for rendering.
- "**culling**" specifically refers to the removal of polygons that are facing away from the viewer and therefore not visible.



BACK FACE CULLING: INTRODUCTION (2)

- Based on the concept of the normal vector of a polygon (the direction that the polygon is facing).
- Compares the angle between the normal vector of a polygon and the direction vector of the viewer's line of sight.
  - If the angle is greater than 90 degrees, the polygon is facing away from the viewer and can be culled.
- Advantage:
  - A simple and fast technique that can improve rendering performance, especially for complex scenes with many polygons.
- Disadvantage:
  - Not suitable for all types of objects and scenes.
  - Thin wall or hollow objects may have polygons facing both inwards and outwards, back-face culling ineffective.
  - Scenes with complex lighting or transparency effects may require more sophisticated visibility determination techniques.

NORMAL VECTOR (1)

- Normal vectors are essential for understanding the orientation of polygons (such as triangles or quadrilaterals) in 3D space.

- Each polygon has a normal vector that points outward from its surface.

- Think of it as an arrow perpendicular to the polygon's surface, indicating which way the polygon "faces."

NORMAL VECTOR COMPONENTS

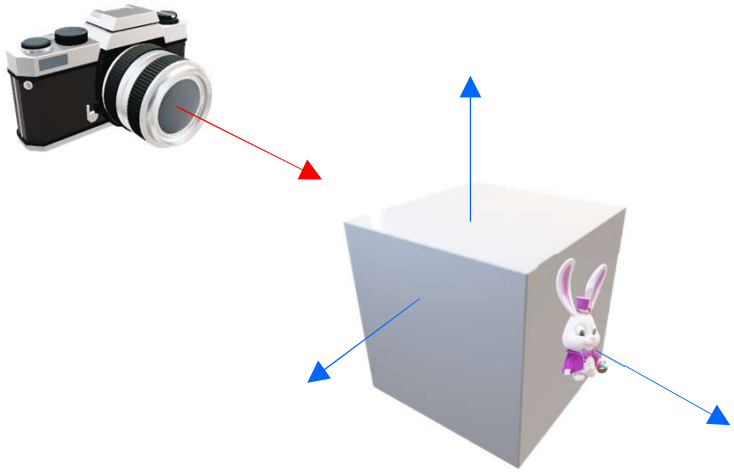
- The normal vector is typically represented by three components: (A, B, C).
- These components define the direction of the normal in the x, y, and z axes, respectively.
- For example:
  - If a polygon lies flat in the xy-plane (parallel to the ground), its normal vector would be (0, 0, 1) (since it points directly upward along the positive z-axis).

VIEWING DIRECTION

- When rendering a 3D scene, we need to consider the viewer's perspective.
- The viewing direction is the line of sight from the viewer (or camera) to the scene.
- In our case:
- We assume the viewer looks along the negative Z-axis (i.e., looking away from the viewer).

BACK-FACE DETECTION

- Locating **back faces** of a polyhedron is based on front-back tests.
- A point (x, y, z) is behind a polygon surface if
  - $Ax + By + Cz + D < 0$; A, B,C, and D are the plane parameters
- Considering the direction of the normal vector N for a polygon surface.
  - If V_{view} is a vector in the viewing direction from our camera position



Then a polygon is a back face if

$$V_{view} \cdot N > 0$$

BACK FACE IDENTIFICATION (1)

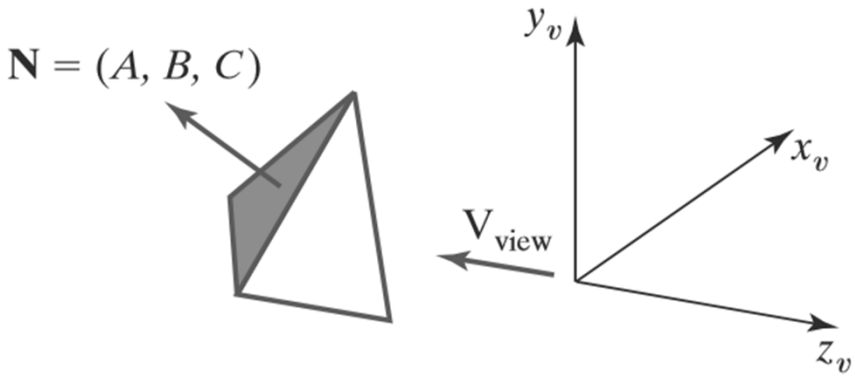
- Surface equation $ax + by + cz + d = 0$
  - a, b, and c are the coefficients of the x, y, and z variables, respectively.
  - d is a constant term.
  - x,y,z is on the plane
- The normal vector of the plane is given by (a, b, c).
  - It points perpendicular to the plane's surface.
  - The magnitude of the normal vector represents the inclination of the plane.
- Viewing Direction
  - let's consider the viewer looking along the negative Z-axis (away from the viewer).

BACK FACE IDENTIFICATION (2)

- Calculate the dot product of the normal vector (a, b, c) with the viewing direction vector (which is (0, 0, -1) for our case).
  - $a.b = a_x b_x + a_y b_y + a_z b_z = 0+) + c.(-1) = -c$
- If the dot product is negative, the polygon is a back face.
- This means the normal vector points away from the viewer (opposite to the viewing direction).
- If the z-component of the normal vector is less than zero (C < 0), the polygon is identified as a back face.

VISIBLE SURFACE DETECTION

- Viewing direction along the negative z axis
- A polygon surface with plane **parameter C< 0 in a right-handed viewing coordinate system is identified as a back face** when the viewing direction is along the negative

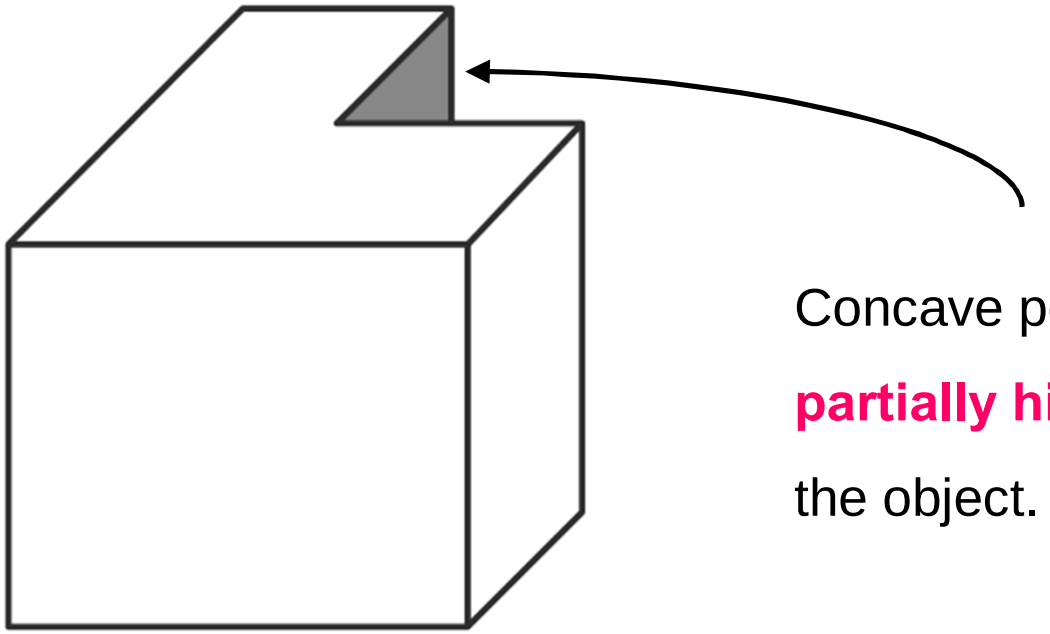


ALGORITHMIC STEPS

1. For each polygon in the scene
    1. Calculate its normal vector.
2. Determine the position of the viewer's eye in relation to the polygon.
    1. This can be done by taking the dot product of the polygon's normal vector and the vector from the polygon to the viewer's eye.
3. If the dot product is negative
    1. the polygon is facing towards the viewer and should be rendered.
4. If the dot product is positive or zero
    1. the polygon is facing away from the viewer and can be culled.
5. Repeat this process for all polygons in the scene.

LIMITATION OF BACK-FACE CULLING

- It's worth noting that this algorithm assumes that all polygons are oriented consistently with respect to the viewer.
- If some polygons in the scene are oriented in the opposite direction, the algorithm may incorrectly cull visible polygons or render invisible ones.
- In these cases, additional checks and techniques may be necessary to ensure correct visibility determination.



Concave polyhedron with **one face partially hidden** by other faces of the object.

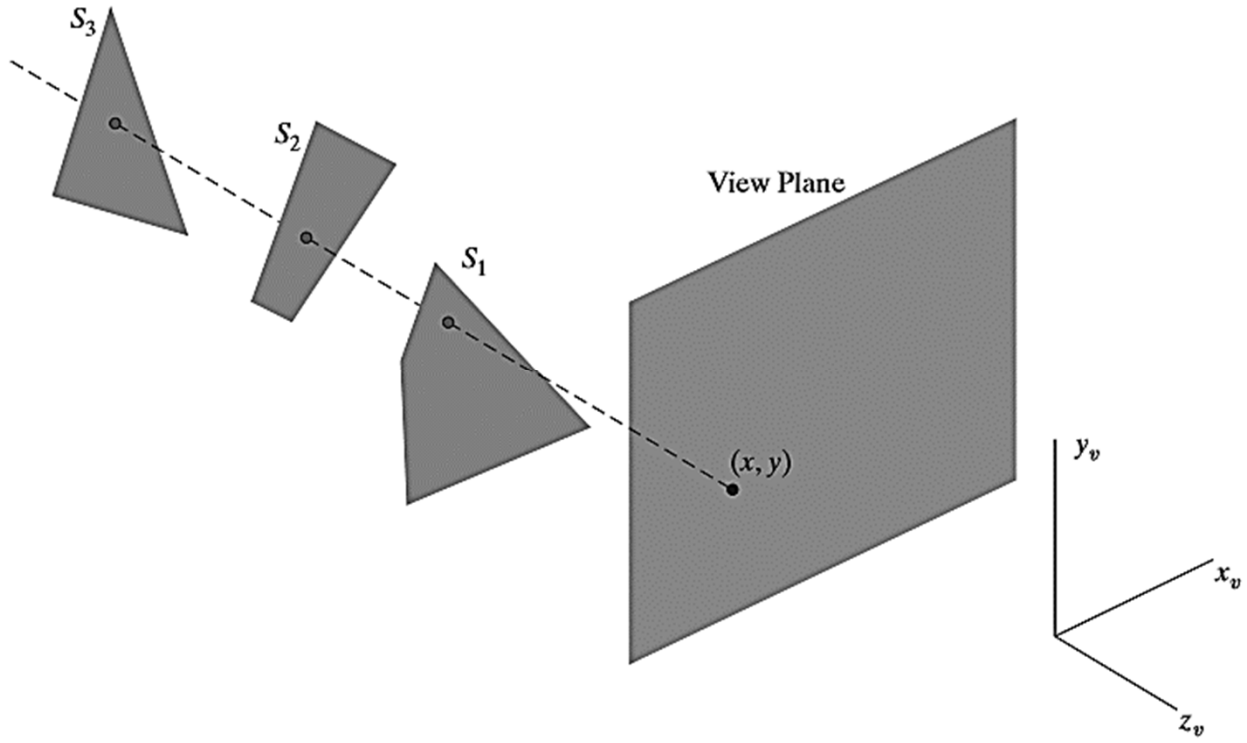
IMAGE SPACE APPROACH

DEPTH-BUFFER METHOD

- Image-space approach for detecting visible surfaces
- Compares **surface depth values** throughout a scene **for each pixel position** on the **projection plane**
- **Each surface of a scene is processed separately**
- The algorithm is usually applied to **scenes containing only polygon surfaces**
- Also frequently alluded to as the **z-buffer method**

ILLUSTRATION

- Three surfaces overlapping pixel position (x, y) on the view plane.
- The visible surface, S_1, has the smallest depth value.



DEPTH-BUFFER ALGORITHM

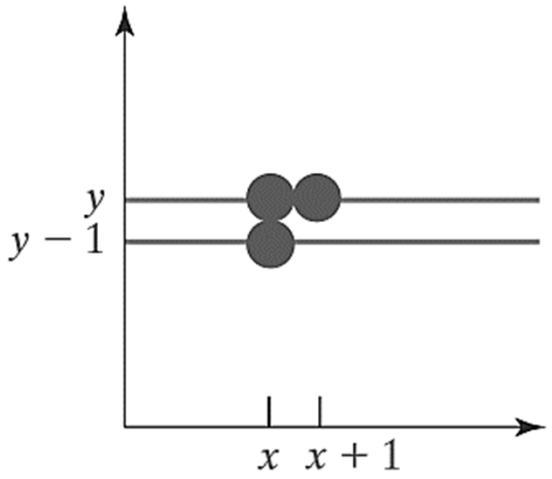
1. Initialize the **depth buffer** and **frame buffer** so that for all buffer positions (x, y),
    1. depthBuff (x, y) = 1.0, frameBuff (x, y) = backgndColor
2. Process each polygon in a scene, one at a time, as follows:
    1. For each projected (x, y) pixel position of a polygon, calculate the depth z (if not already known).
    2. If z < depthBuff (x, y), compute the surface color at that position and set
        1. **depthBuff (x, y) = z, frameBuff (x, y) = surfColor (x, y)**

After all surfaces have been processed, the **depth buffer contains depth values for the visible surfaces** and the **frame buffer contains the corresponding color values for those surfaces**

CALCULATION

- Given the depth values for the vertex positions of any polygon in a scene

- At surface position (x, y), the depth is calculated from the plane equation as:



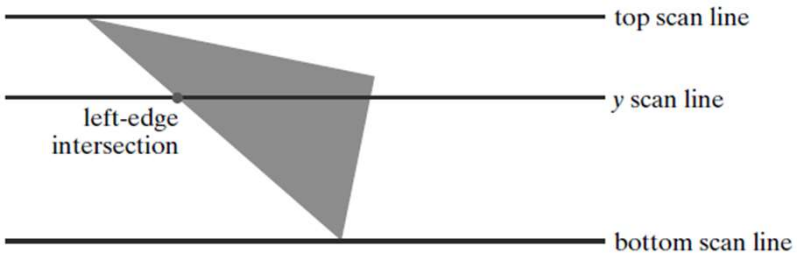
$$z = \frac{-Ax - By - D}{C}$$

The depth z' of the next position (x+1, y) along the scan line is obtained

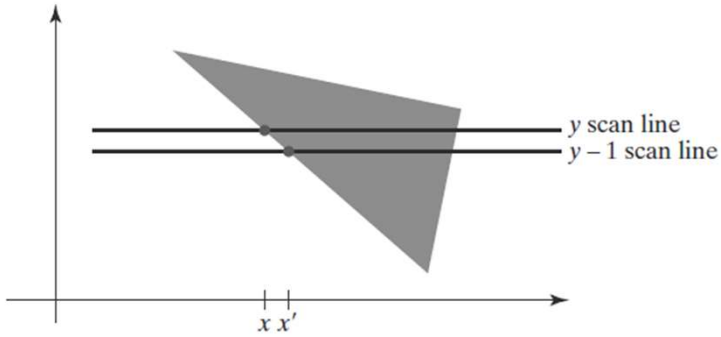
$$z = \frac{-A(x+1) - By - D}{C}$$

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

ILLUSTRATION



Scan lines intersecting a polygon surface.



Intersection positions on successive scan lines along a left polygon edge.

PROS AND CONS OF Z-BUFFER METHOD

Advantages

- Simplicity: The Z-buffer algorithm is straightforward to implement.
- Handling Opaque Surfaces: It can handle any kind of opaque surface.
- Complex Surface Intersections: It effectively displays complex surface intersections.
- Linear Computational Complexity: No depth storing of objects is needed, resulting in linear computational complexity.

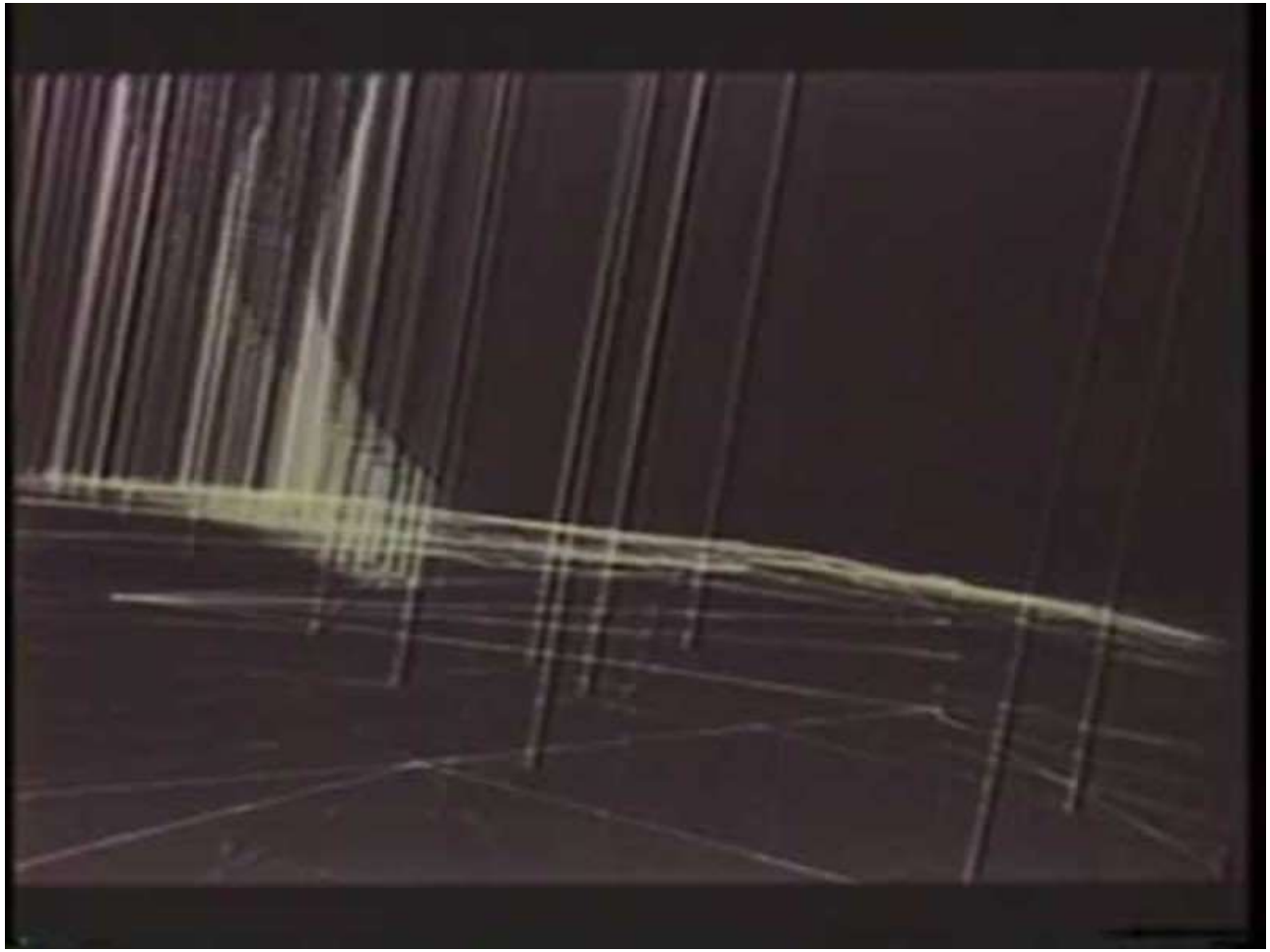
Disadvantages

- High Storage Requirements: requires two buffers
- Dynamic Z-Buffer Updates: frequent updates based on the number of surfaces representing the scene.
- Time-Consuming Process: The algorithm needs to scan and convert every polygon in the scene.
- Large Space Requirement: The buffers (depth and frame) involve significant memory space.
- Transparent Objects can not be rendered.

A-BUFFER METHOD

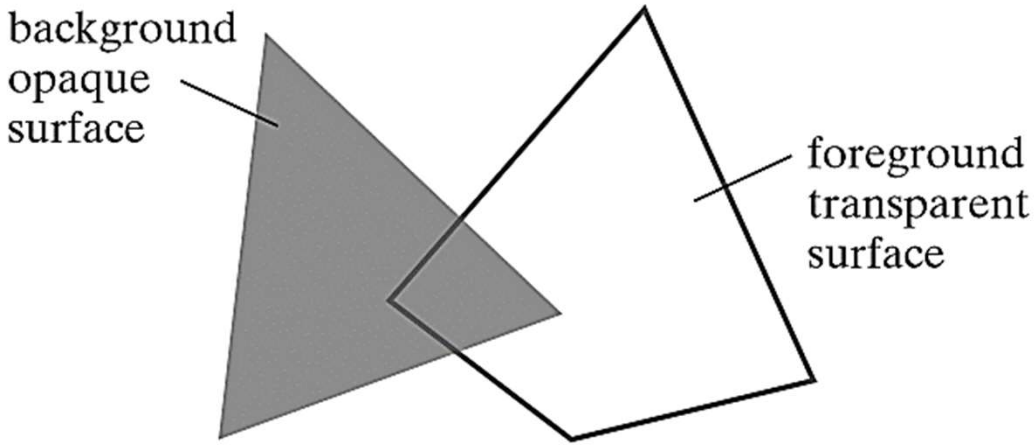
- An extension of the depth-buffer ideas
- An antialiasing, area-averaging, visibility-detection method
- REYES ("Renders Everything You Ever Saw") 3D rendering system
- The buffer region - **accumulation buffer**
- Stores a variety of surface data, in addition to depth values

THE GENESIS DEMO

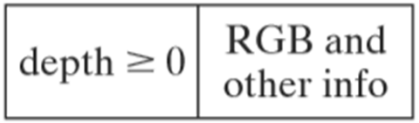


ADVANTAGE OF A-BUFFER

Drawback of Z-Buffer:
It deals only with opaque surfaces and cannot accumulate color values for more than one surface



Single surface overlaps a pixel



(a)

Each position in the A-buffer has two fields

Depth field: A real-number value (+ve, -ve, 0).	**Surface data field:** Surface data or a pointer.

More than one surface overlaps a pixel

(b)

A BUFFER: SURFACE INFORMATION

- RGB intensity components
- Opacity parameter (percent of transparency)
- Depth
- Percent of area coverage
- Surface identifier
- Other surface-rendering parameters

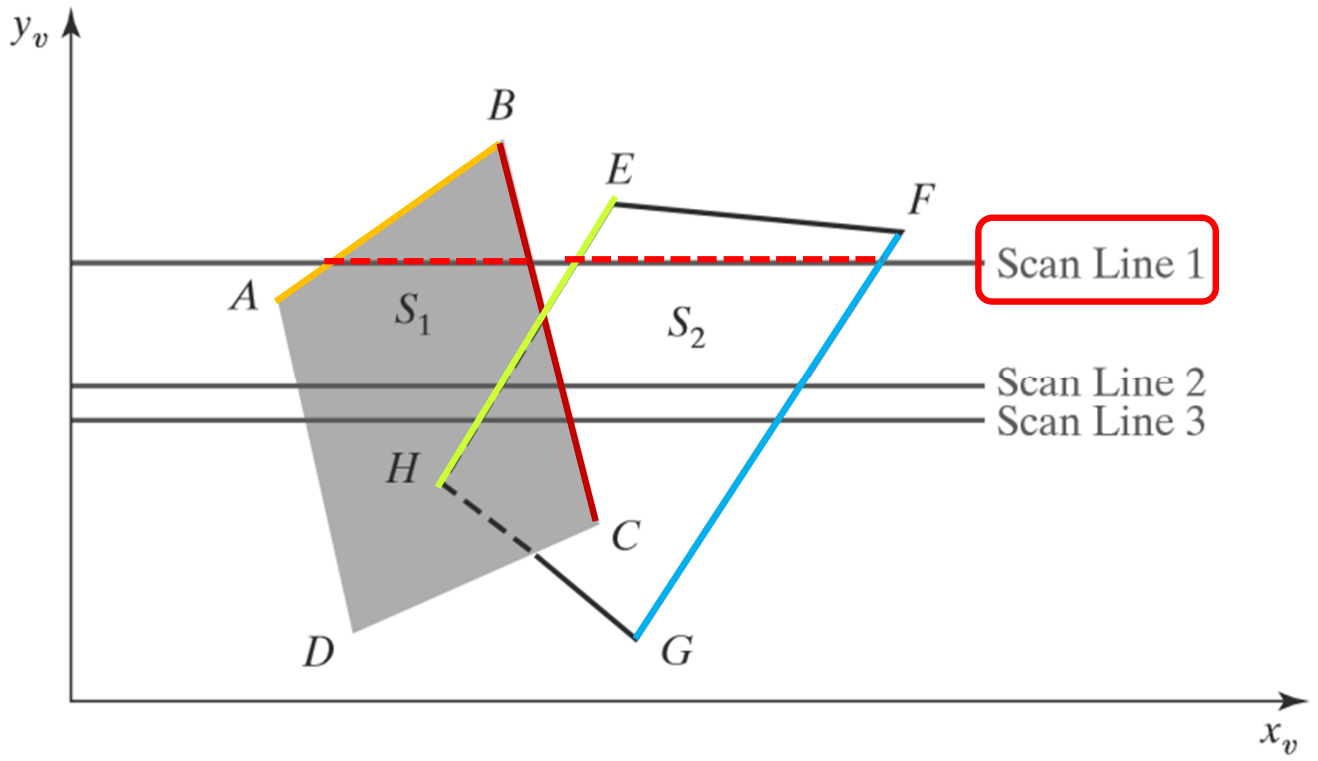
SCAN-LINE METHOD

- **Image-space method** for identifying visible surface
- Compares depth values along the various scan lines for a scene
- Different tables used for VSD
  - Edge table
  - Surface-facet table
  - Active list table
- Flag for each surface

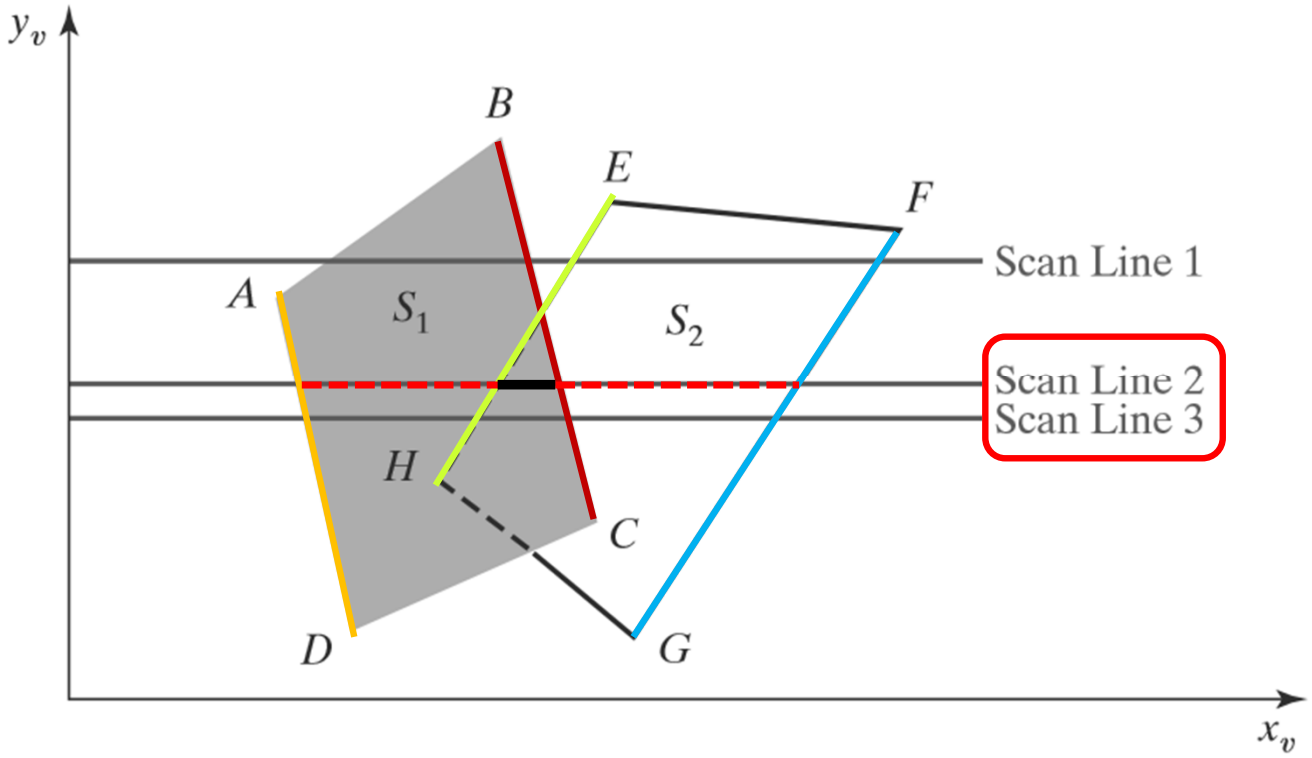
DATA STRUCTURES USED IN SCAN LINE APPROACH

- Edge table
  - Contains the coordinate of two endpoints.
  - Inverse slope of each line
- Active Edge Table
  - Contain edges a given scan line intersects during its sweep.
  - Sorted in increasing order of x.
- Surface Facet Table
  - Plane coefficients, surface material properties, other surface data, and possibly pointers into the edge table.

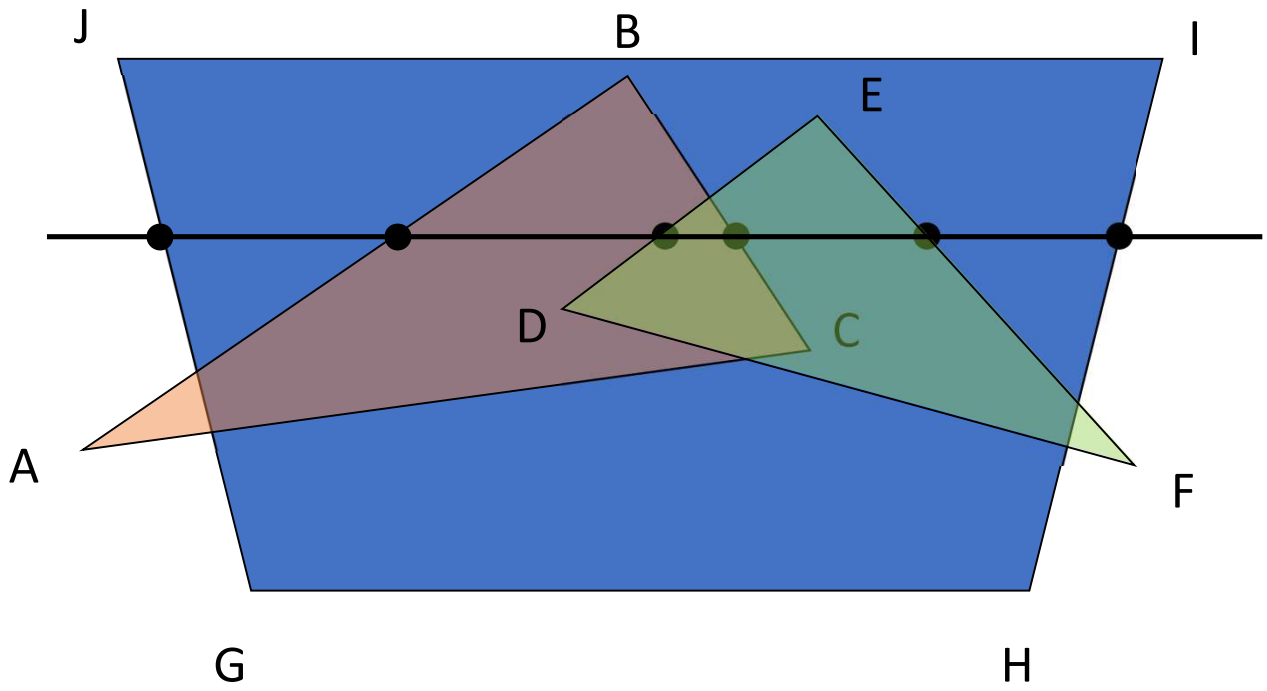
ILLUSTRATION



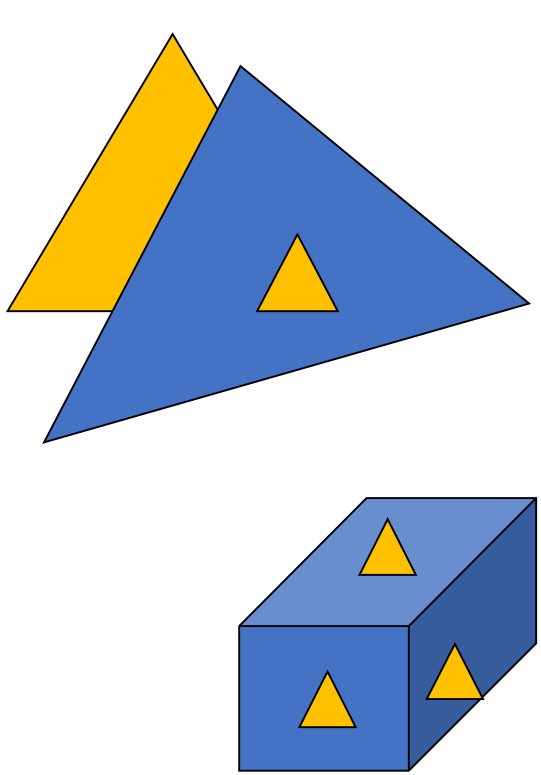
ILLUSTRATION

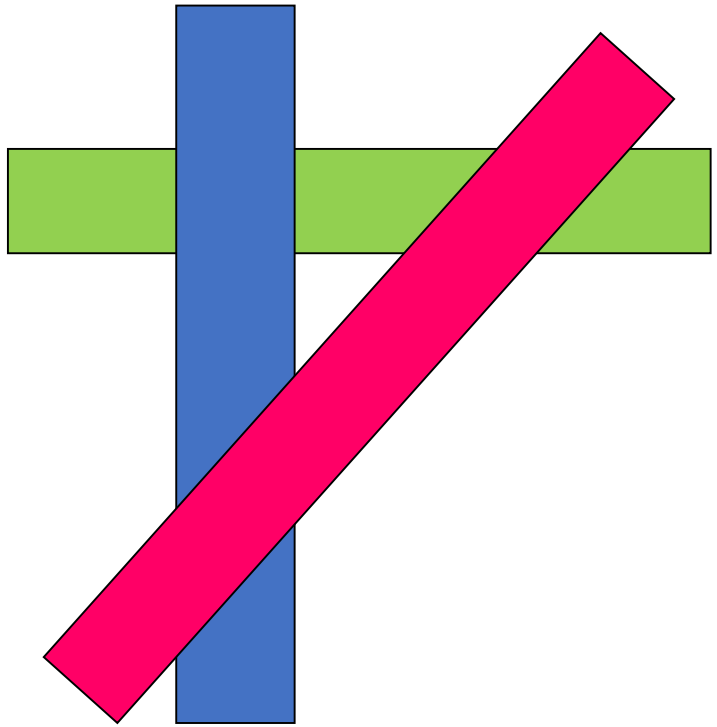


MULTIPLE NON-INTERSECTING POLYGON



SPECIAL CASES OF POLYGON INTERSECTION





HYBRID APPROACH

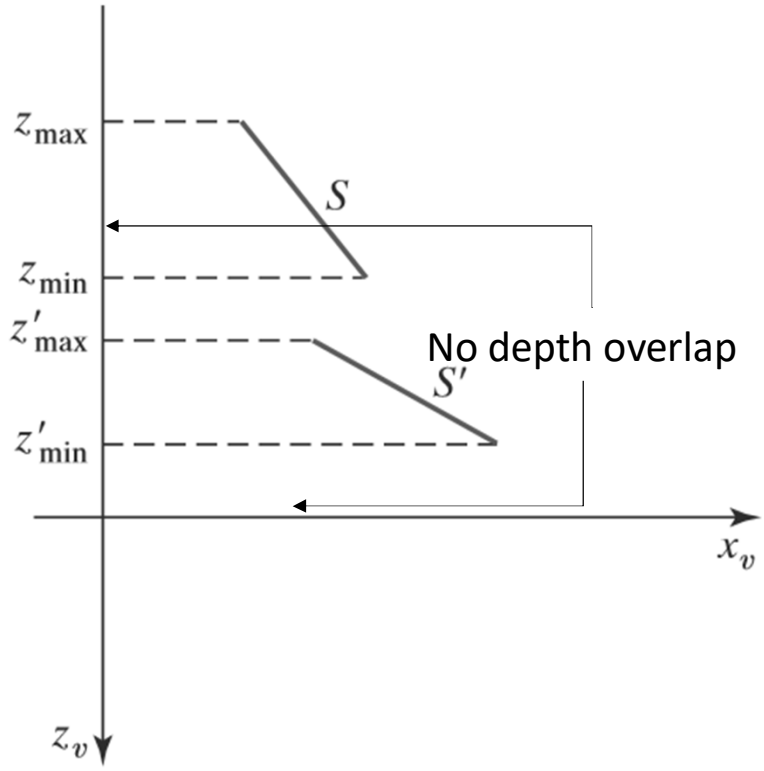
DEPTH-SORTING METHOD (PAINTER'S ALGORITHM)

- Uses both image-space and object-space operations
- Surfaces are sorted in order of decreasing depth
- Surfaces are scan converted in order, starting with the surface of greatest depth.
- Sorting operations are carried out in both image and object space
- The scan conversion of the polygon surfaces is performed in image space.

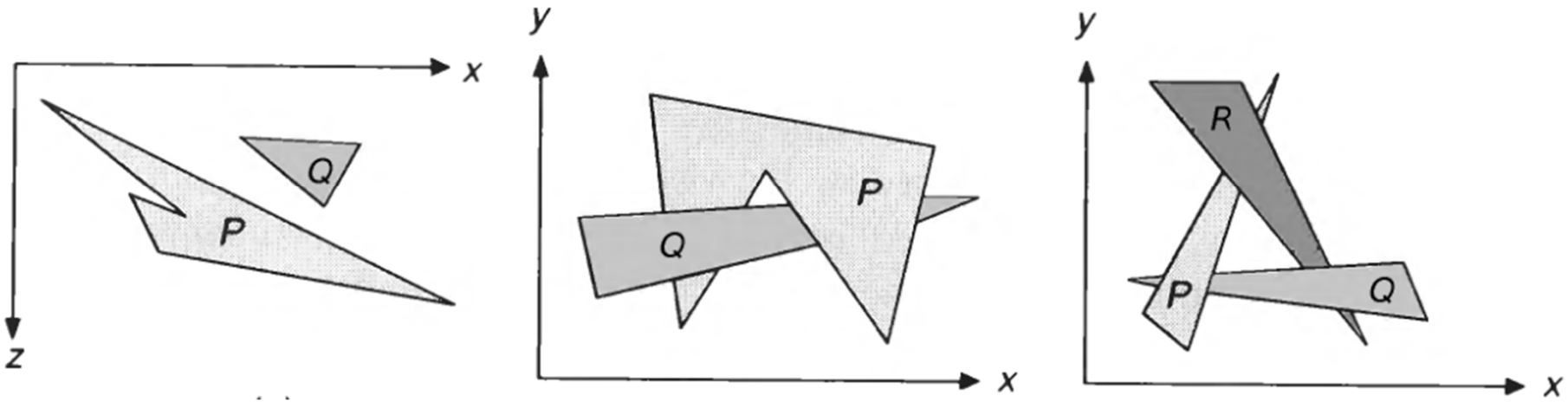
ILLUSTRATION

- Surfaces are ordered on the first pass according to the smallest z value on each surface.
- The surface S at the end of the list (with the greatest depth) is then compared to the other surfaces in the list to determine whether there are any depth overlaps.
- If no depth overlaps occur
  - S is the most distant surface and it is scan-converted

Assuming we are viewing along the z direction



AMBIGUITY



Depth Test

1. Do the polygons' x extents not overlap?
2. Do the polygons' y extents not overlap?
3. Is P entirely on the opposite side of Q's plane from the viewpoint?
4. Is Q entirely on the same side of P's plane as the viewpoint?
5. Do the projections of the polygons onto the (x,y) plane not overlap?

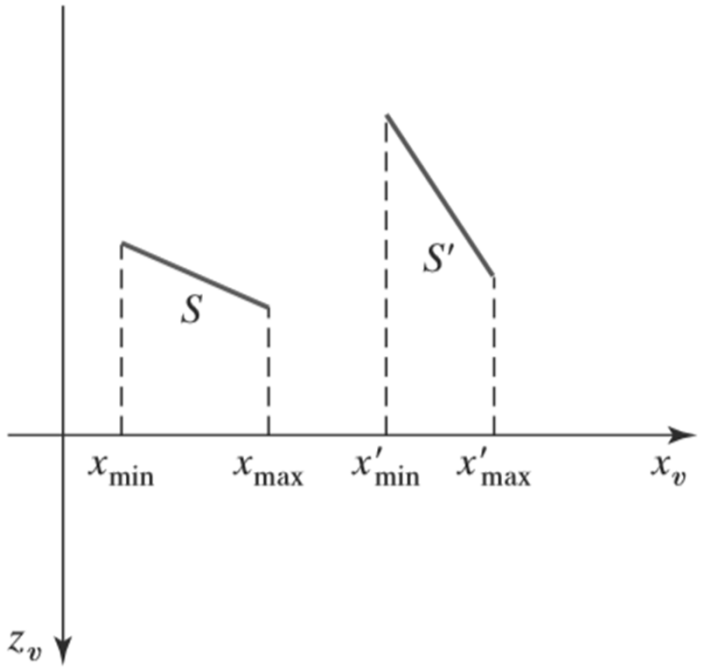
DEPTH TEST (IN THE ORDER OF DIFFICULTY)

1. The bounding rectangles (coordinate extents) in the xy directions 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 boundary-edge projections of the two surfaces onto the view plane do not overlap.

TEST 1

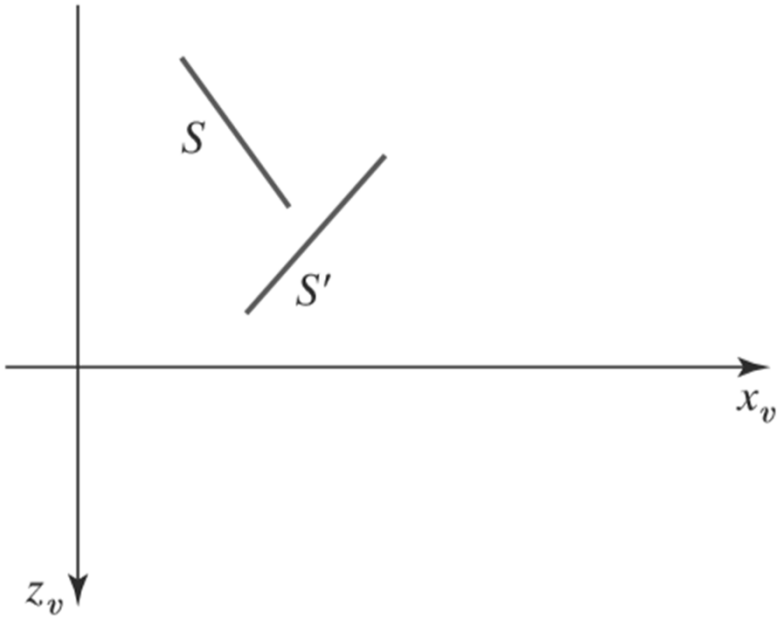
- Test 1 is performed in two parts.
- We check for overlap first in the x direction, then in the y direction.
- If there is no surface overlap in either of these directions
  - The two planes cannot obscure one other

Two surfaces with depth overlap but **no overlap in the x direction.**



TEST 2 AND 3

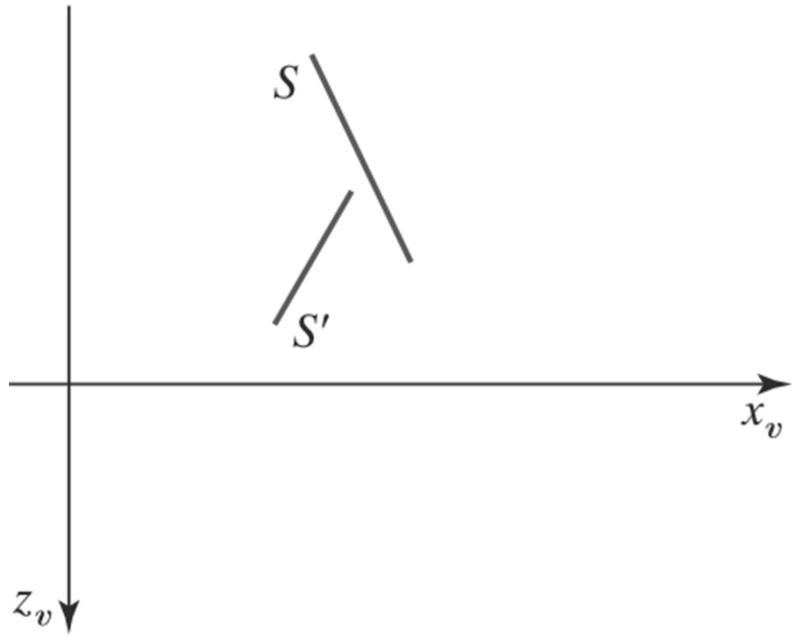
Surface S is completely behind the overlapping surface S'



- Perform tests 2 and 3 using back-front polygon tests.
- Substitute the coordinates for all vertices of S into the plane equation for the overlapping surface and check the sign of the result
- If the plane equations are set up so that the front of the surface is toward the viewing position
  - Then S is behind S′ if all vertices of S are in back of S′

TEST 2 AND 3

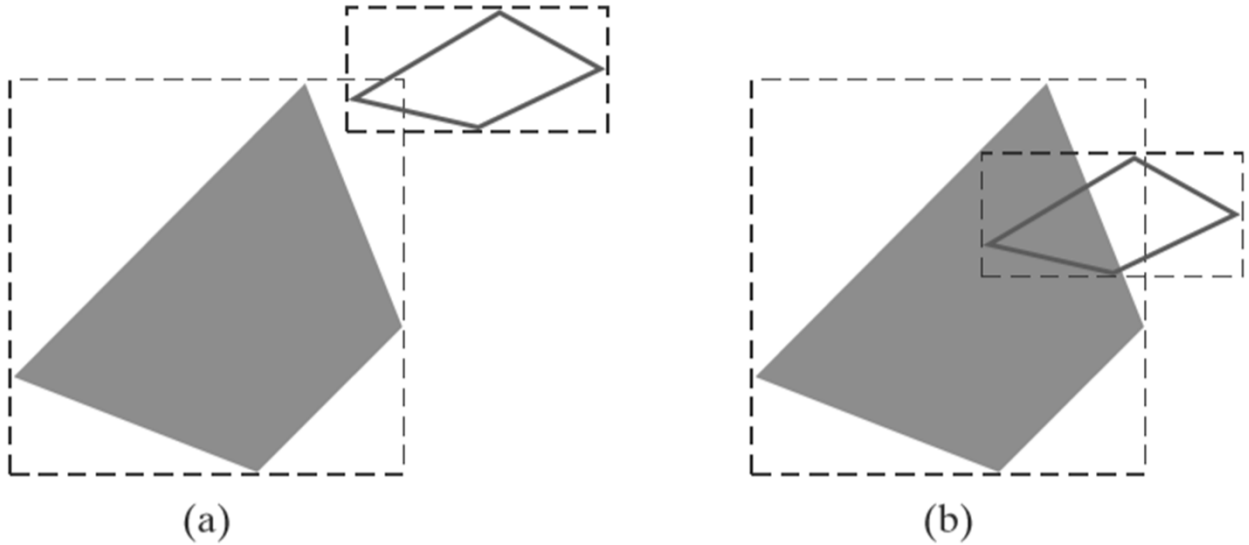
- S′ is completely ahead of S if all vertices of S are in front of S



Overlapping surface S' is completely in front of surface S , but S is not completely behind S'

TEST 4

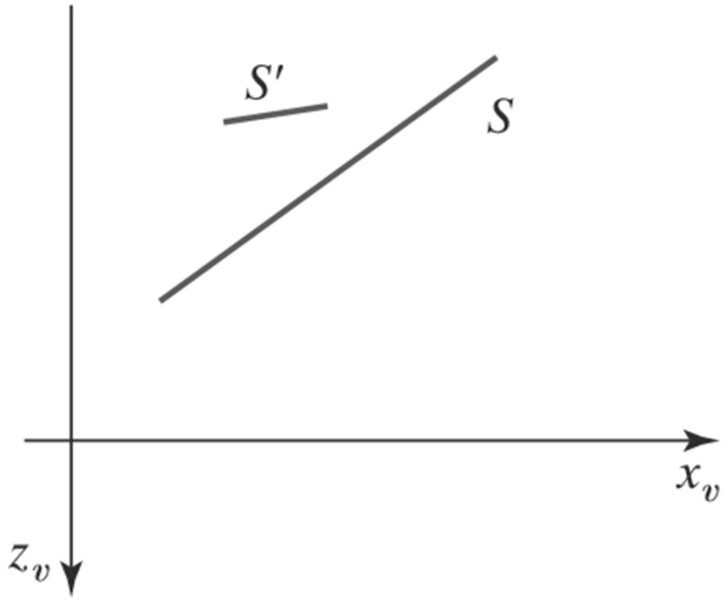
- If tests 1 through 3 have all failed, we perform test 4 to determine whether the two surface projections overlap.



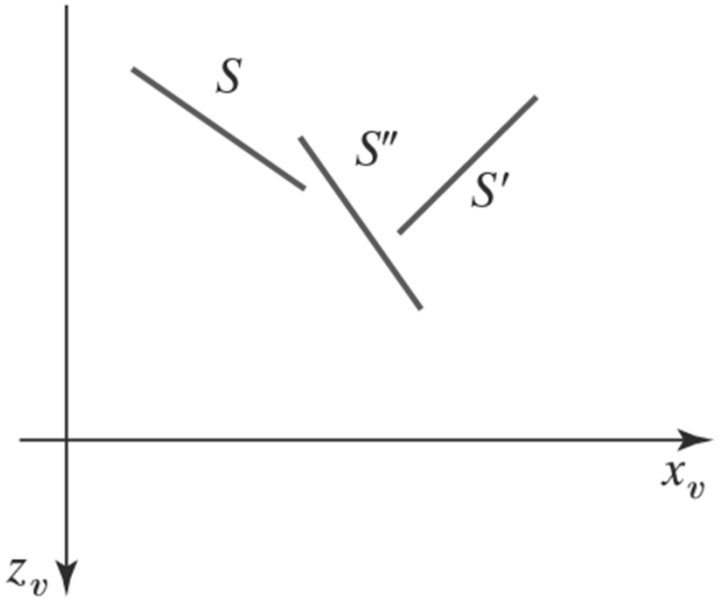
Two polygon surfaces with overlapping bounding rectangles in the xy plane.

USE OF SORTED LIST

- Should all four tests fail for an overlapping surface S, we **interchange** surfaces S and S' in the sorted list.



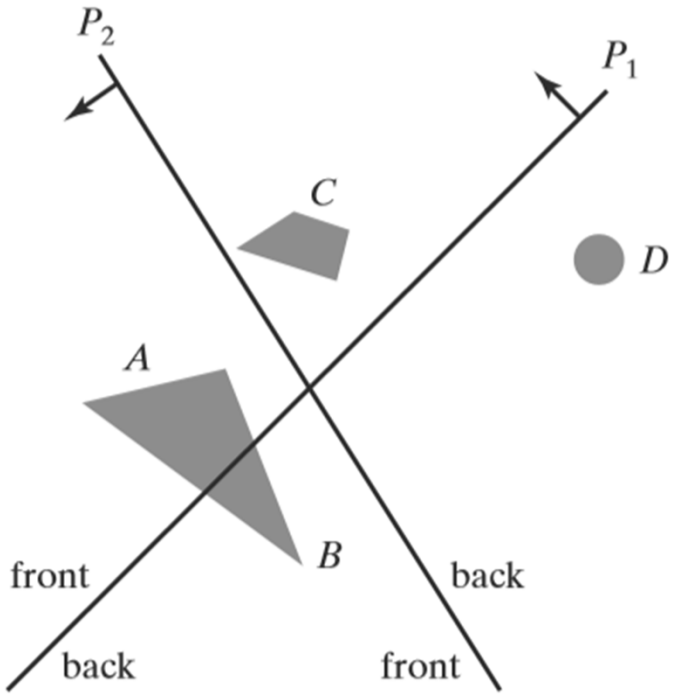
Surface S extends to a greater depth, but it obscures surface S'

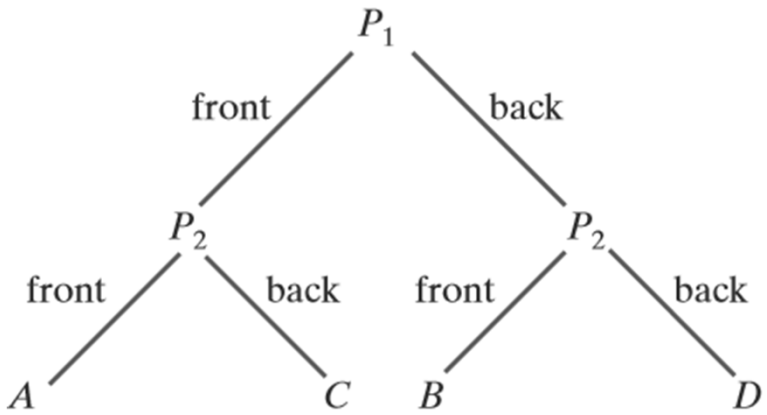


Three surfaces that have been entered into the sorted surface list in the order S, S', S'' should be reordered as S',S'', S.

BSP-TREE METHOD

- An efficient method
- Determine object visibility by painting surfaces into the frame buffer from back to front
- Particularly useful when the view reference point changes, but the objects in a scene are at fixed positions.

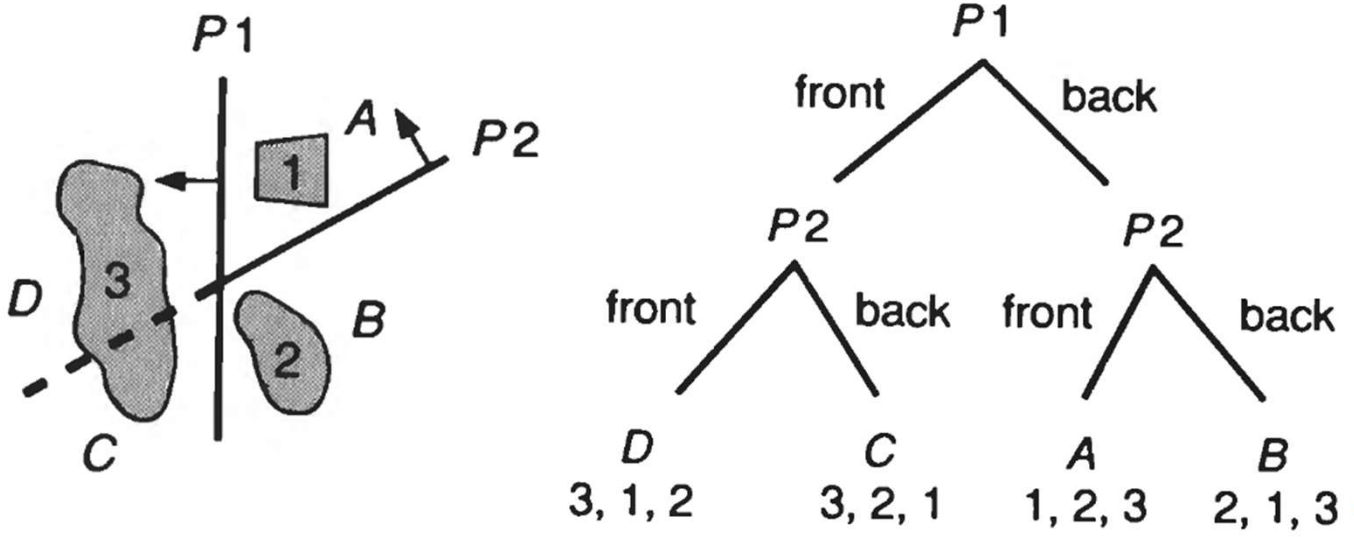




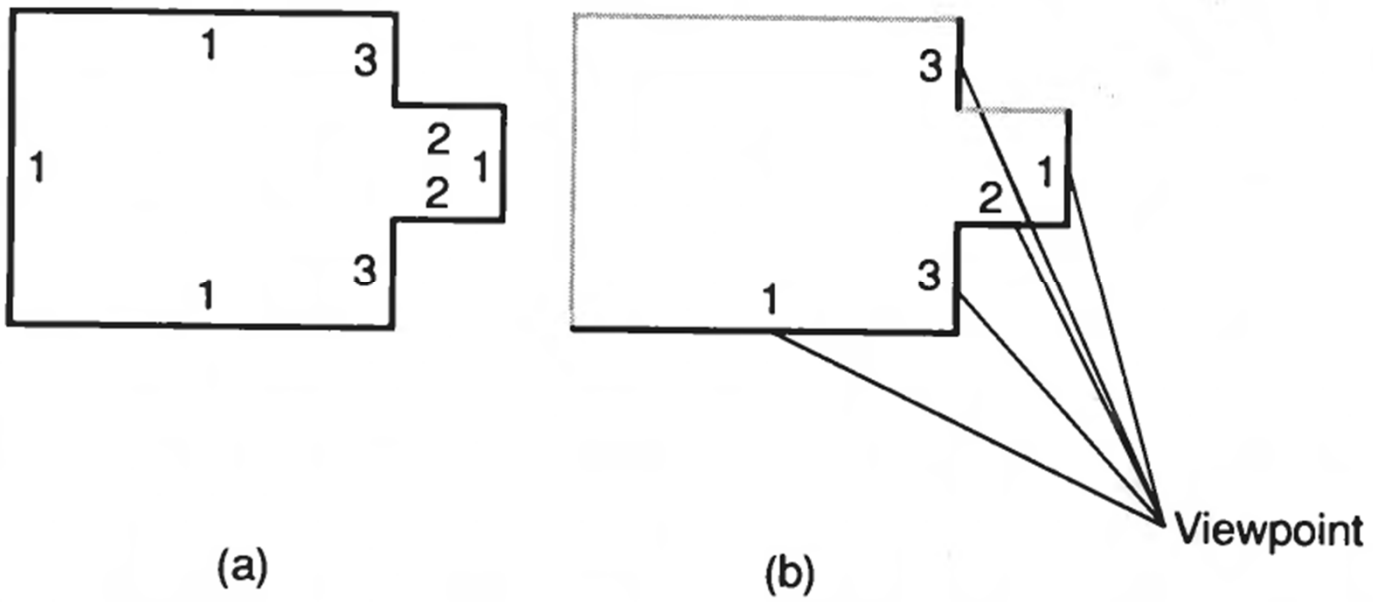
BSP tree representation

BSP TREE METHOD

- Extremely efficient method
- Calculates the visibility relationships among a static group of 3D polygons
- The viewpoint can be arbitrary
- Environments can be viewed as being composed of clusters



FACE PRIORITY



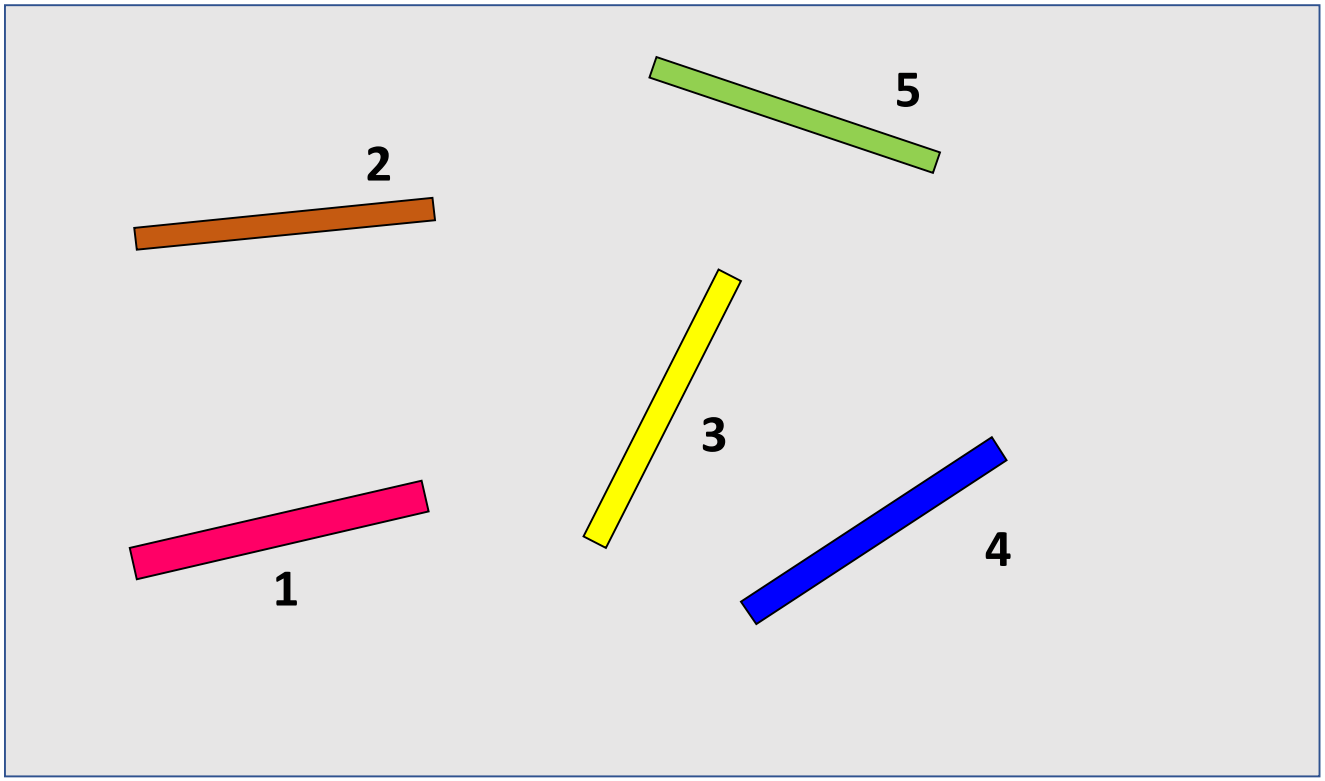
BSP TREE

- The BSP tree's root is a polygon selected from those to be displayed
- The root polygon is used to partition the environment into **two half-spaces.**
- **Front-face and Back Face**
- Recursively partition the space till a single polygon is remaining

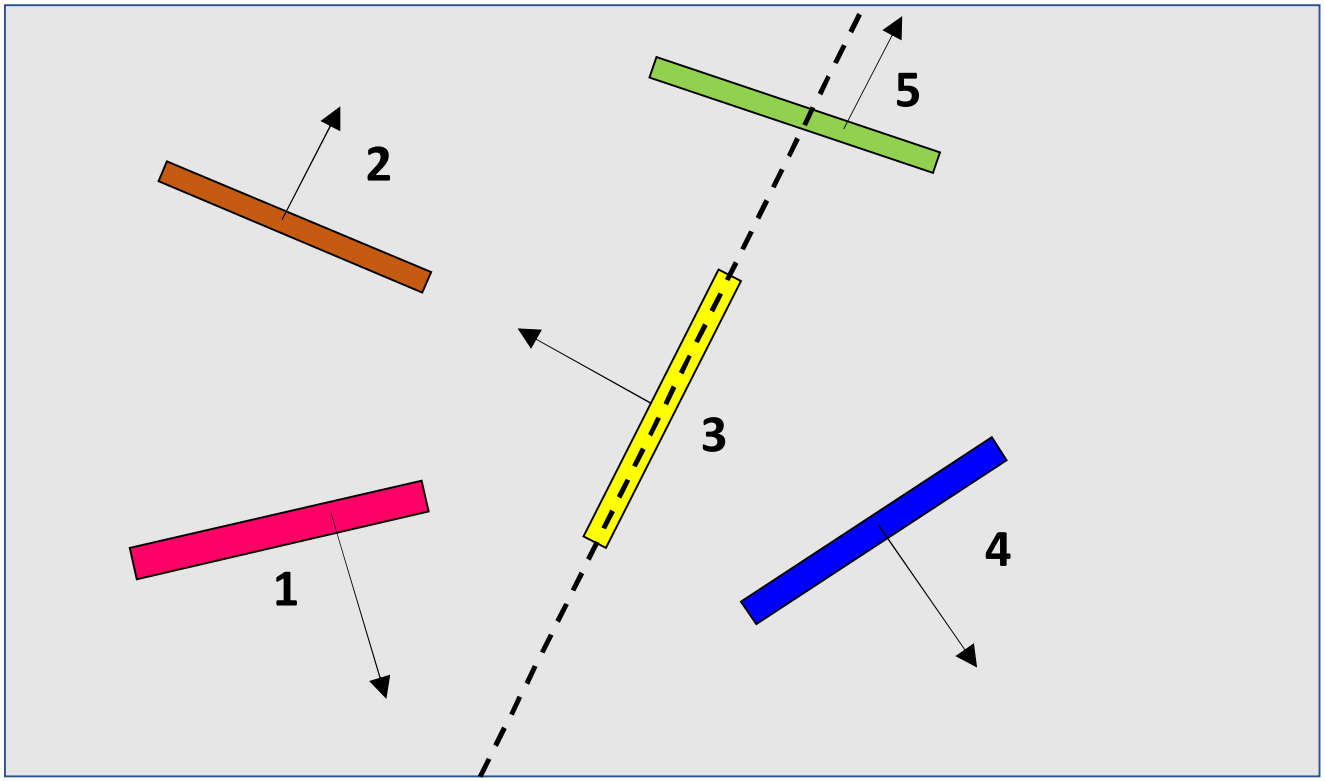
PSEUDOCODE

```
BSP_ tree *BSP_makeTree (polygon *polyList) {
     polygon root,
     polygon *backList, *frontList;
     polygon p, backPart, frontPart;
     if {polyList == NULL)
           return NULL;
     else {
           root = BSP.selectAndRemovePoly (&polyList);
           backList = NULL;
           frontList = NULL;
           for (each remaining polygon p in polyList) {
                if (polygon p in front of root)
                     BSP.addToList (p, &frontList);
                else if (polygon p in back of root)
                     BSP.addToList (p, &backList);
                else { /* Polygon p must be split. */
                     BSP.splitPoly (p, root, &frontPart, &backPart);
                     BSP.addToList {frontPart, &frontList);
                     BSP.addToList (backPart, &backList);
                     }
           }
           return BSP.combineTree (BSP.makeTree (frontList), root, BSP. makeTree (backList));
     }
}
```

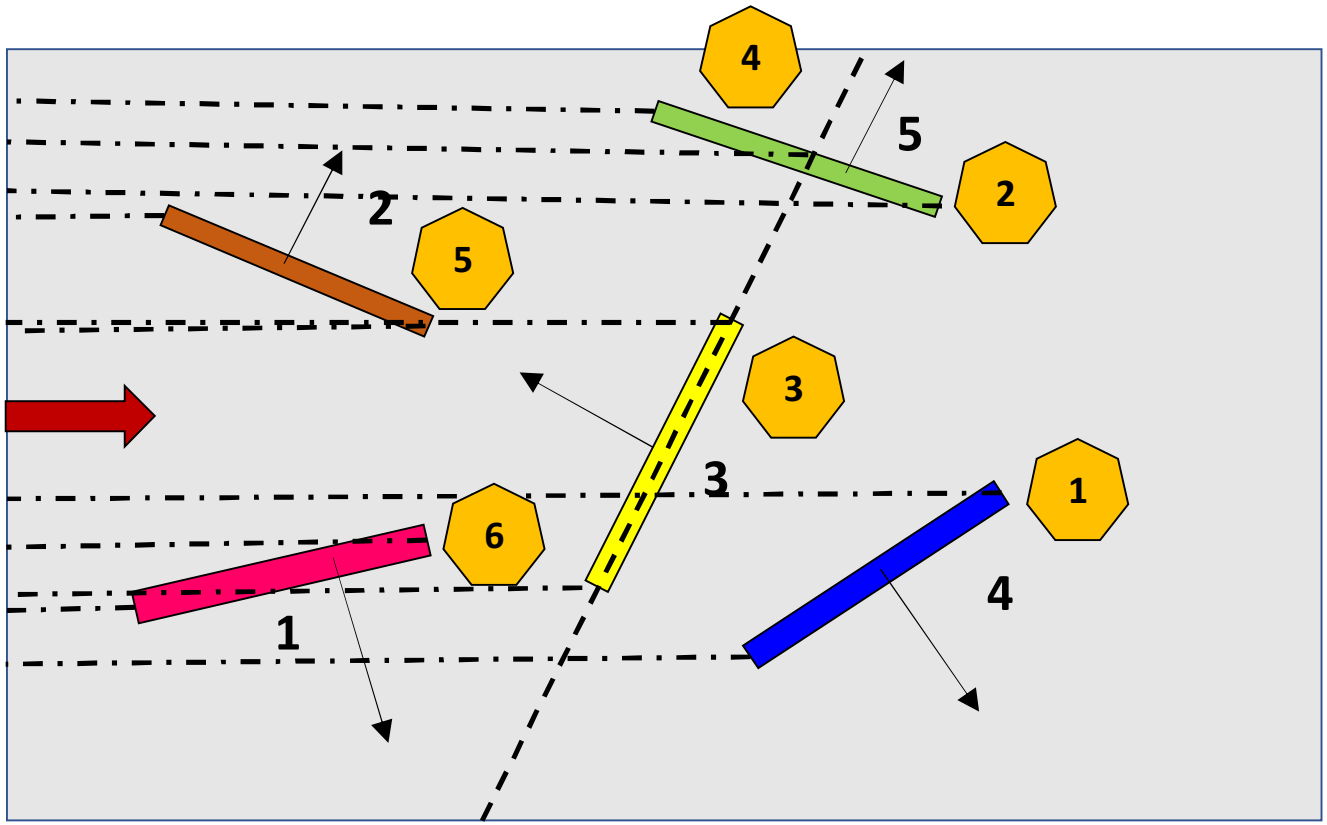
ILLUSTRATION



ILLUSTRATION



ILLUSTRATION



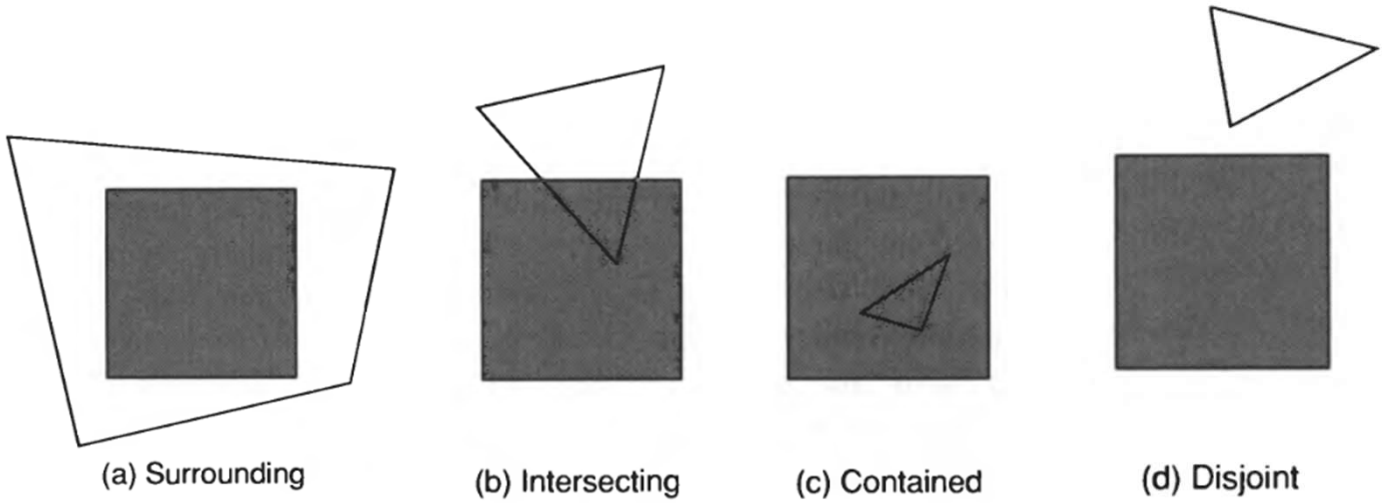
AREA-SUBDIVISION

- Follow the divide-and-conquer strategy
- Spatial partitioning in the projection plane.
- An area of the projected image is examined.
- This approach exploits area coherence,

WARNOCK'S ALGORITHM

- Subdivides each area into four equal squares.

- The projection of each polygon has one of four relationships to the area of interest



STEPS

1. All the polygons are disjoint from the area.
    1. The background color can be displayed in the area.
2. There is only one intersecting or only one contained polygon.
    1. The area is first filled with the background color, and
    2. then the part of the polygon contained in the area is scan-converted.
3. There is a single surrounding polygon, but no intersecting or contained polygons.
    1. The area is filled with the color of the surrounding
4. More than one polygon is intersecting, contained in, or surrounding the area, but one is a surrounding polygon that is in front of all the other polygons.

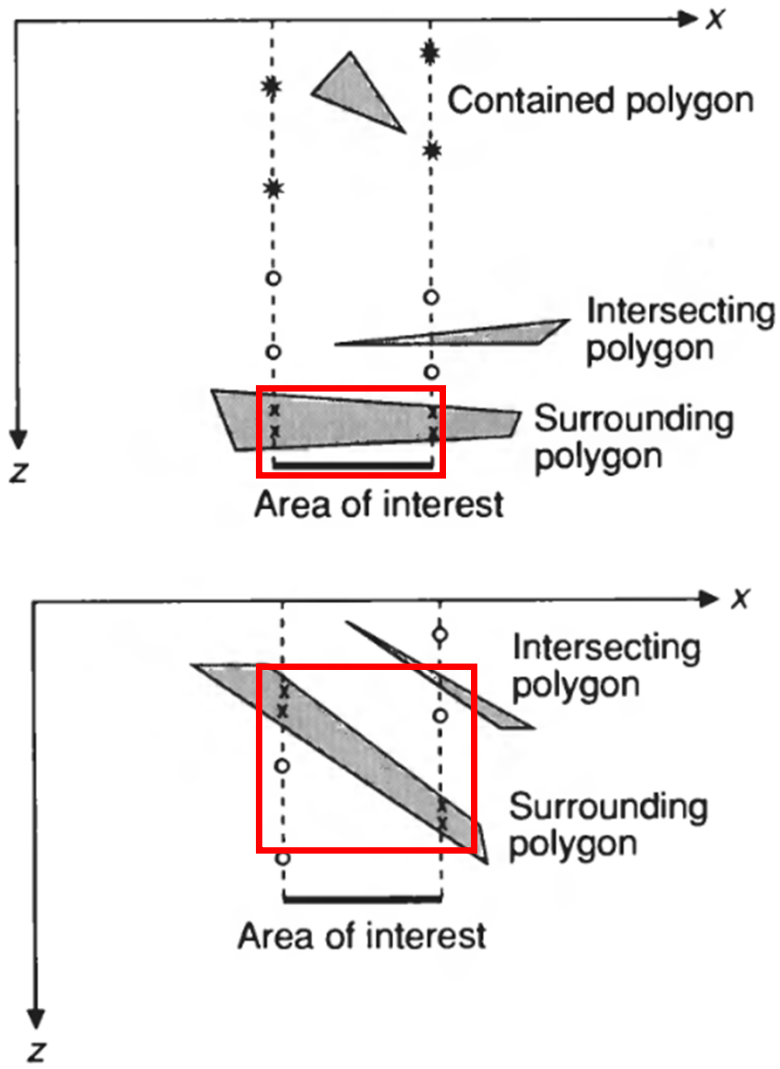
ILLUSTRATION

- Two examples of case 4 in recursive subdivision
- Top:
  - Surrounding polygon is closest at all corners of area of interest,
- Bottom:
  - Intersecting polygon plane is closest at left side of area of interest

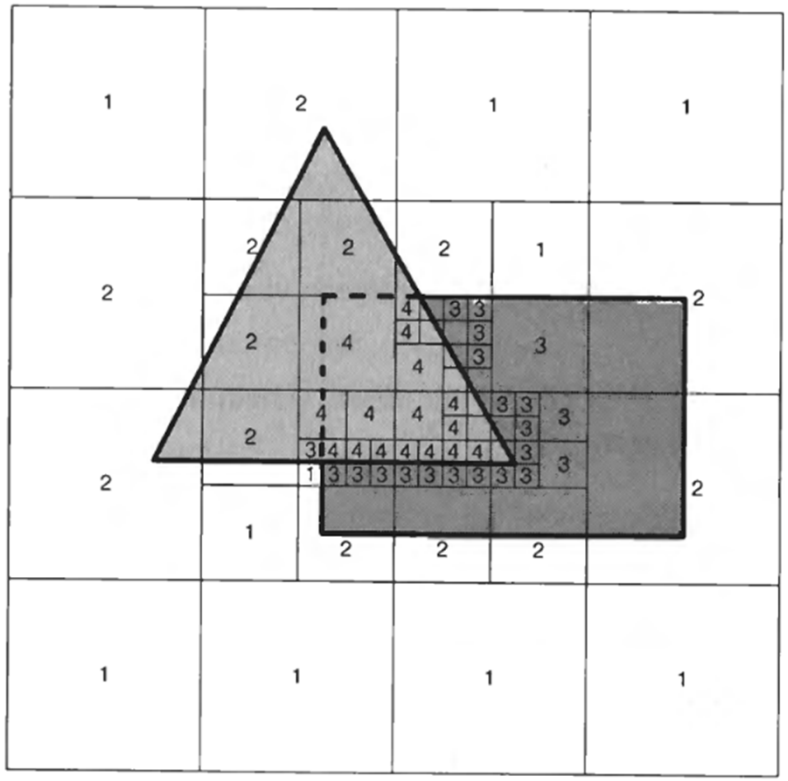
- x marks the intersection of surrounding polygon plane;

- o marks the intersection of intersecting polygon plane;

- * marks the intersection of contained polygon plane.



EXAMPLE



EXAMPLE

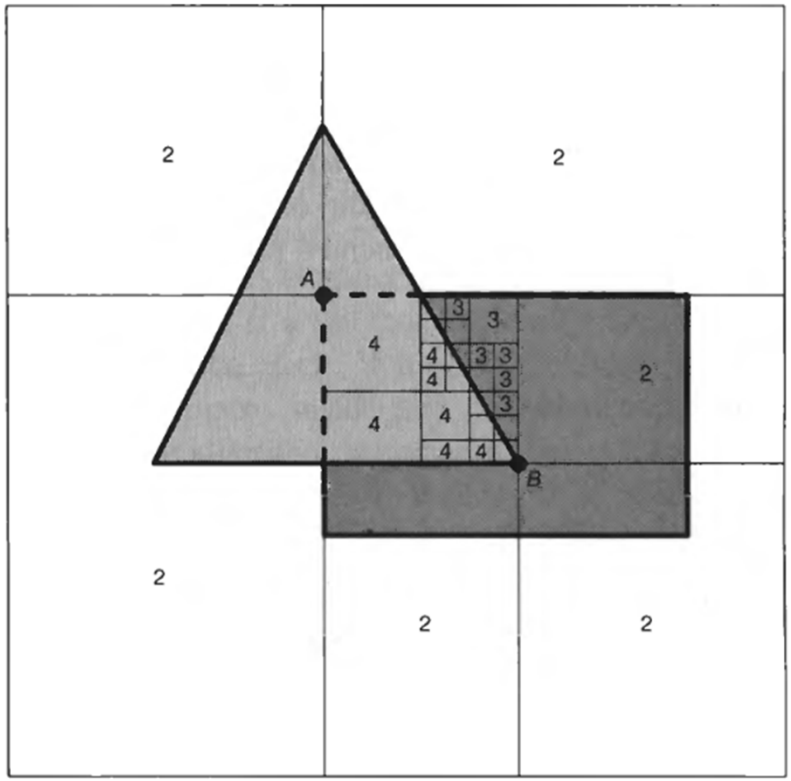


ILLUSTRATION TO EXPLAIN THE 3D RENDERING PROCESS

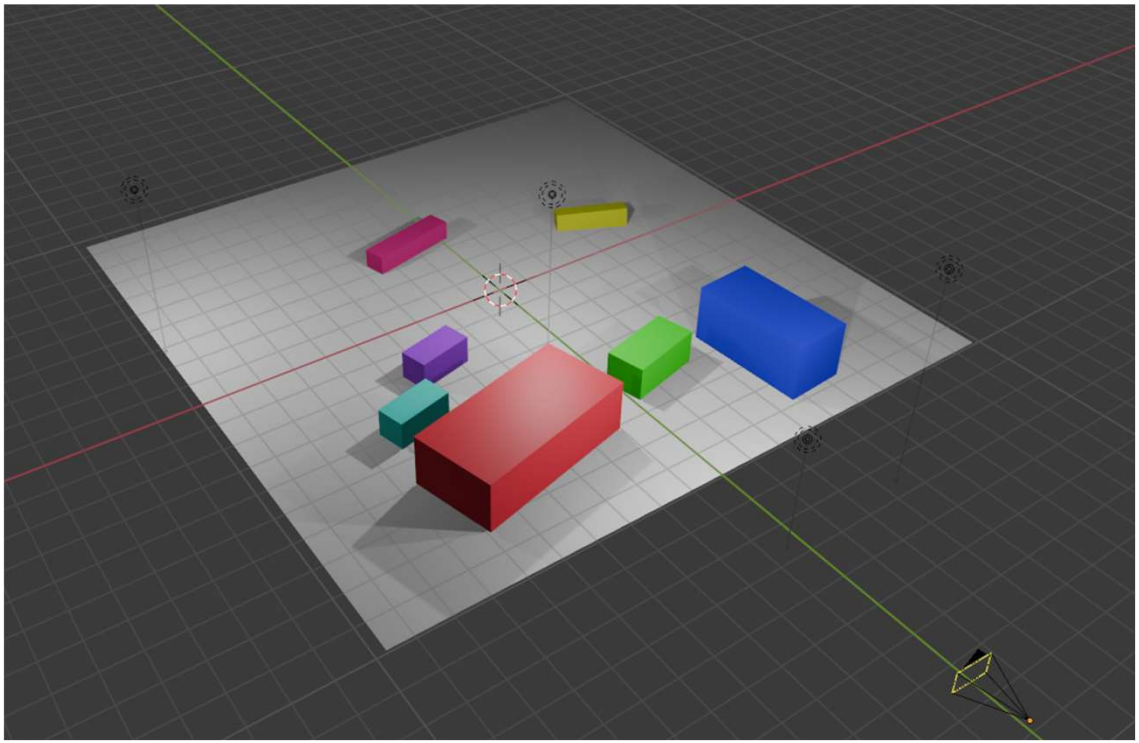
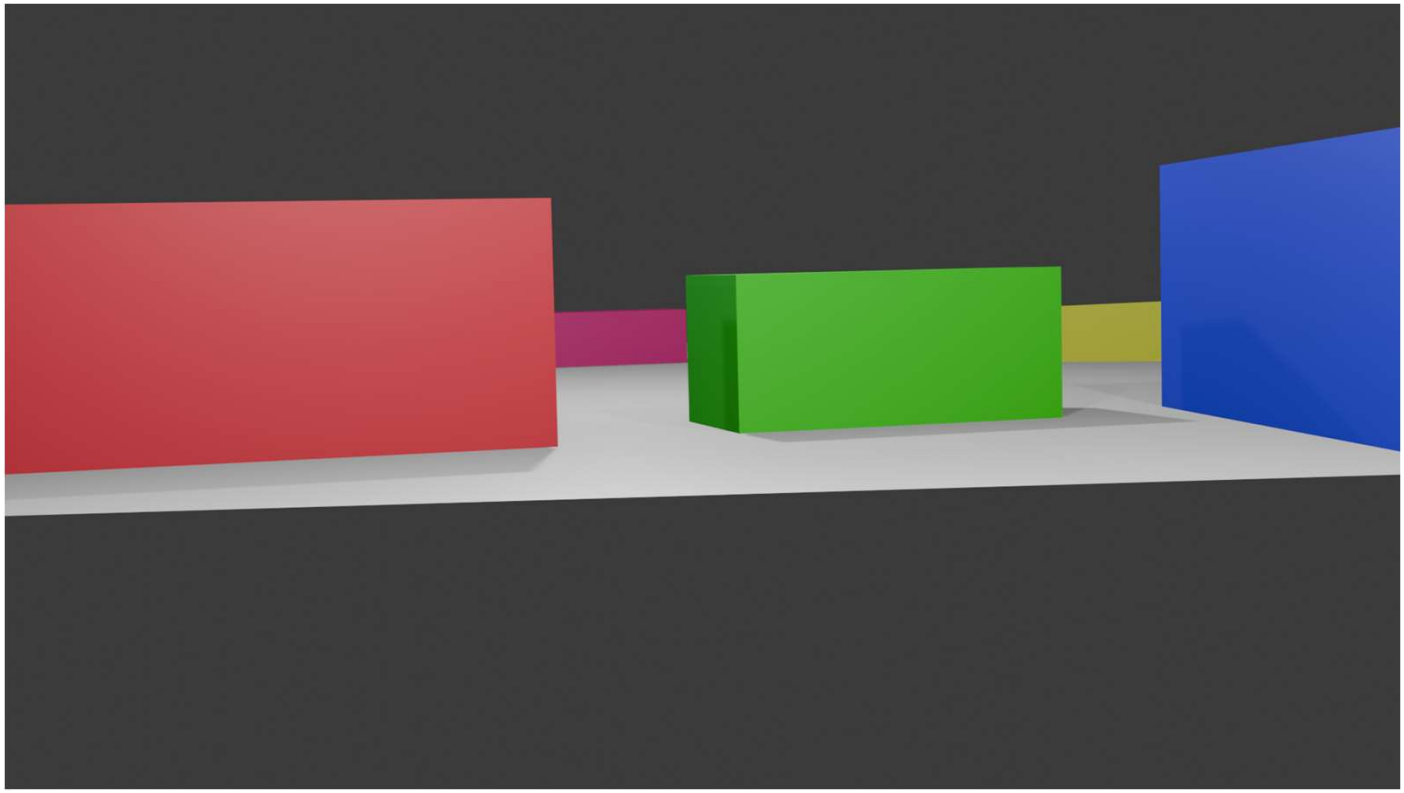
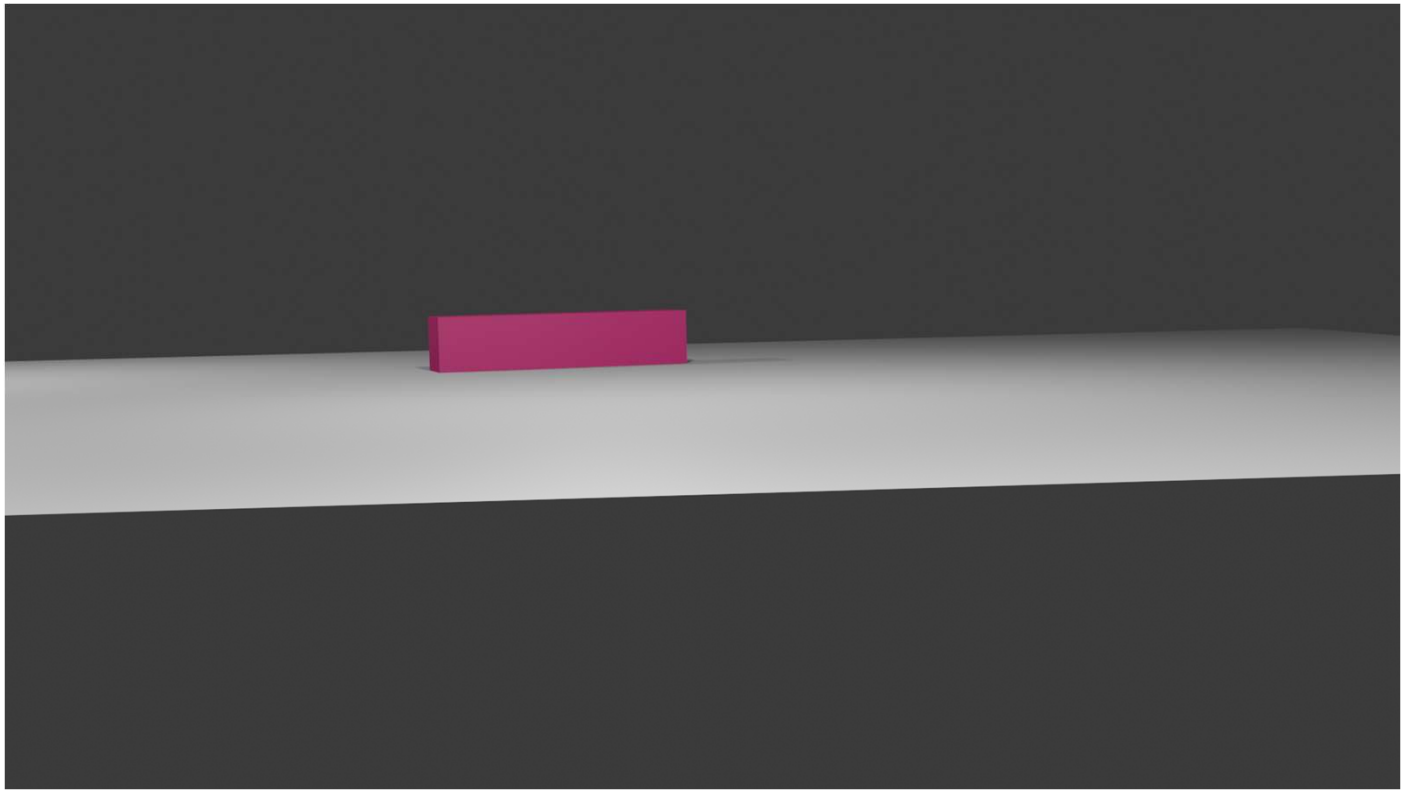


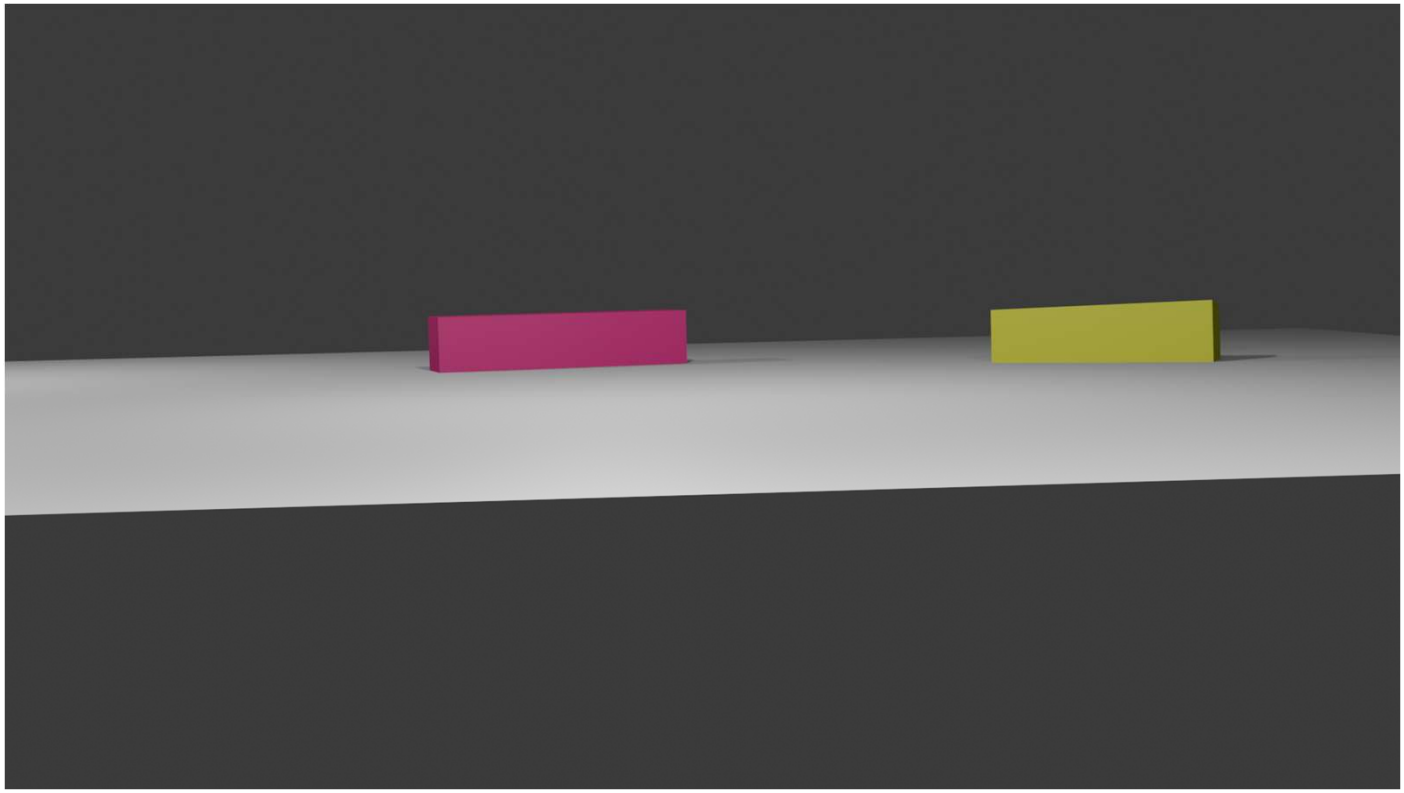
ILLUSTRATION TO EXPLAIN THE 3D RENDERING PROCESS



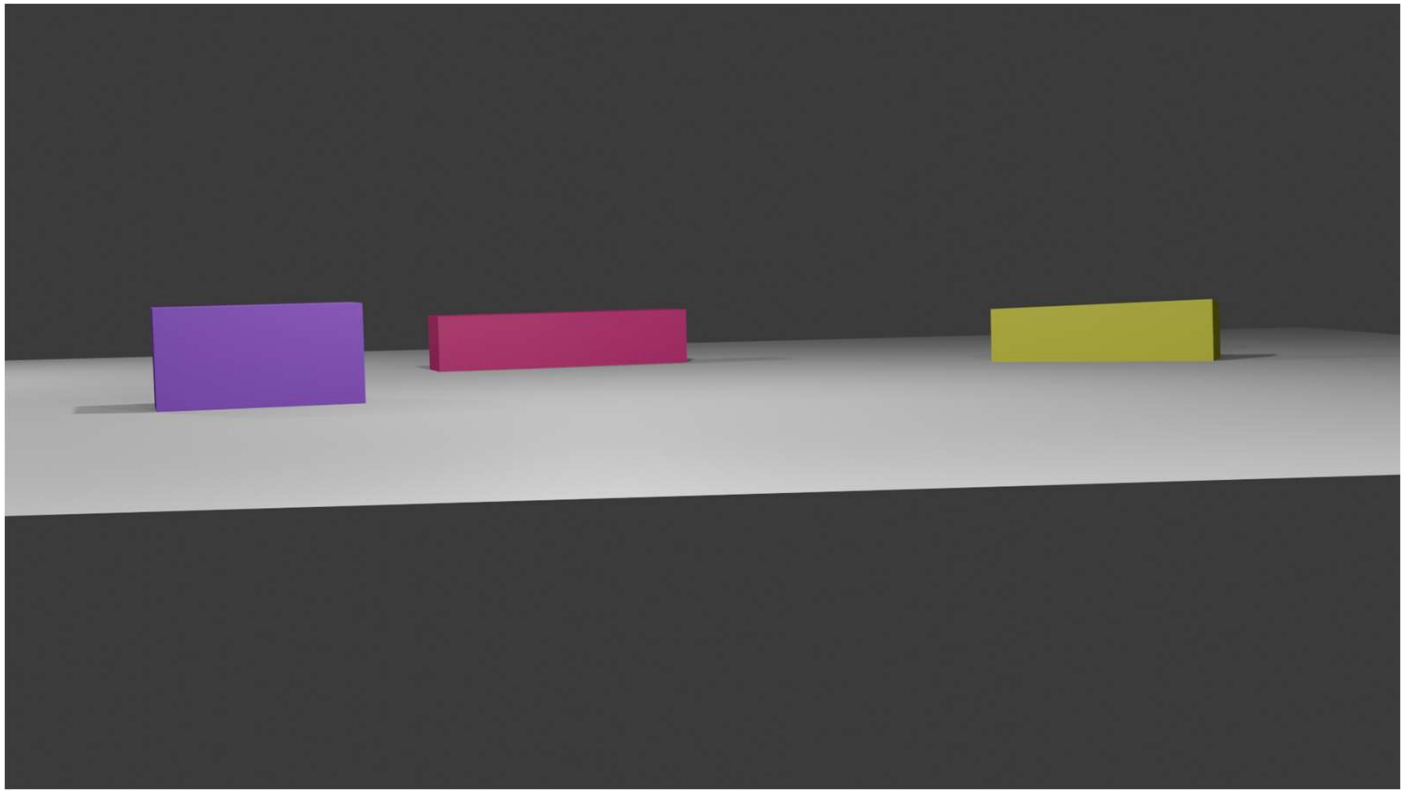
PAINTER'S ALGORITHM



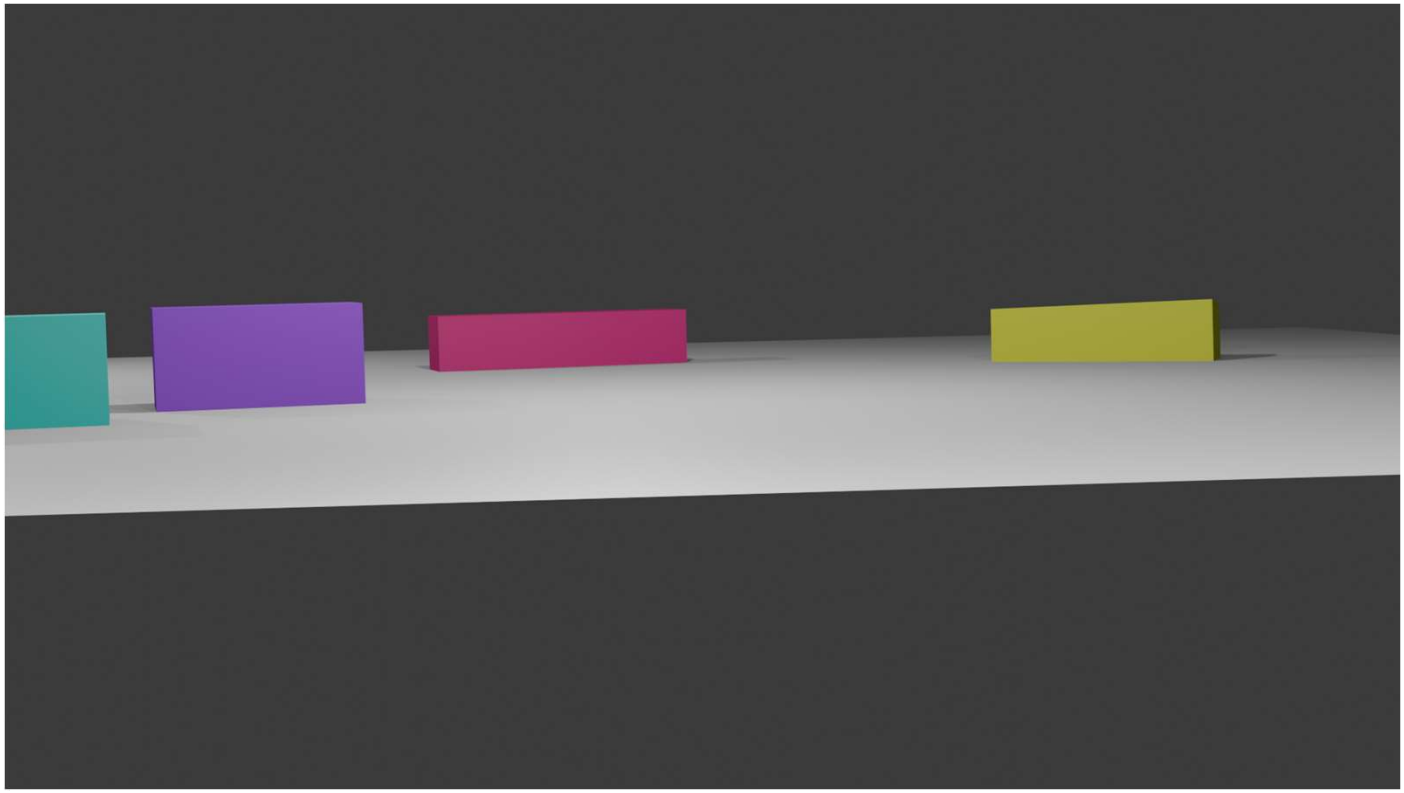
PAINTER'S ALGORITHM



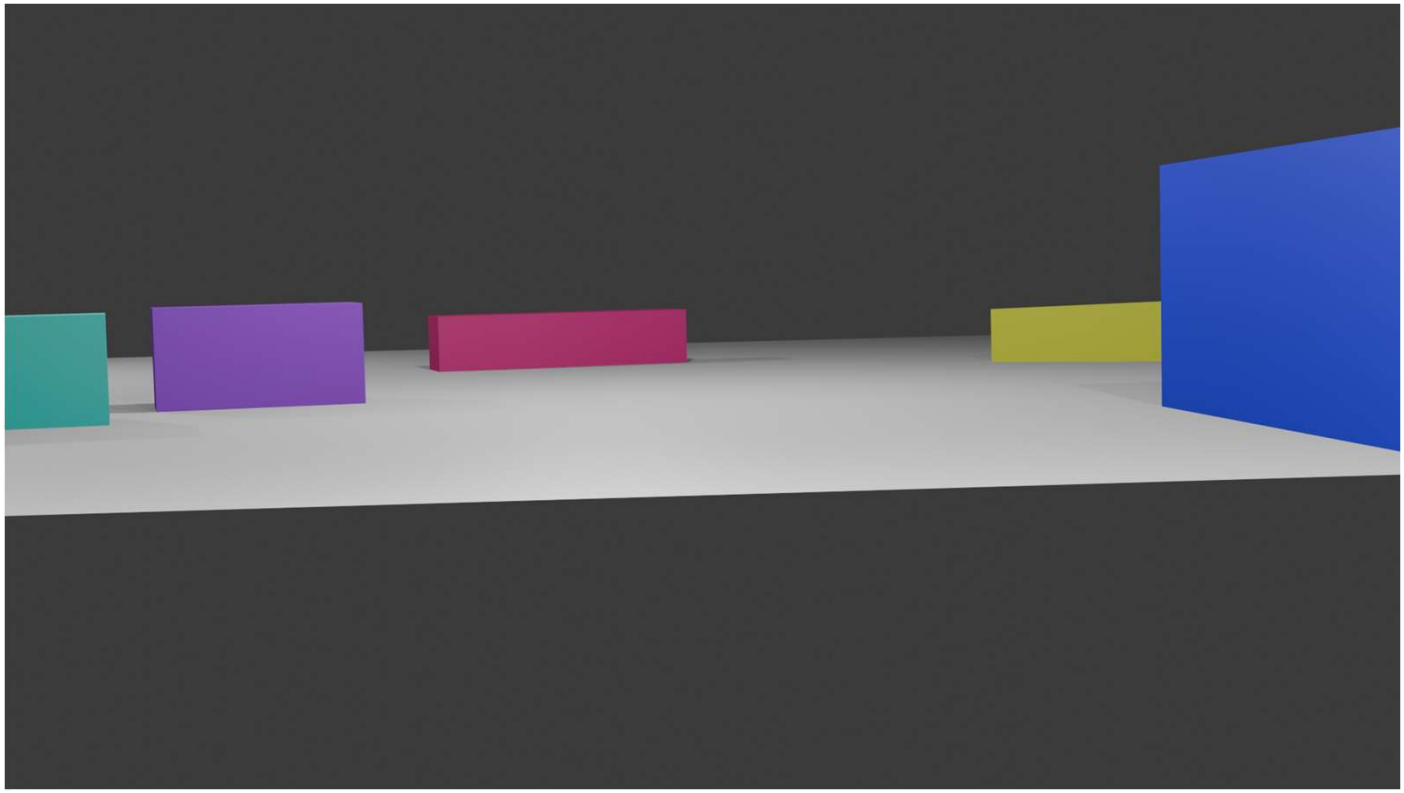
PAINTER'S ALGORITHM



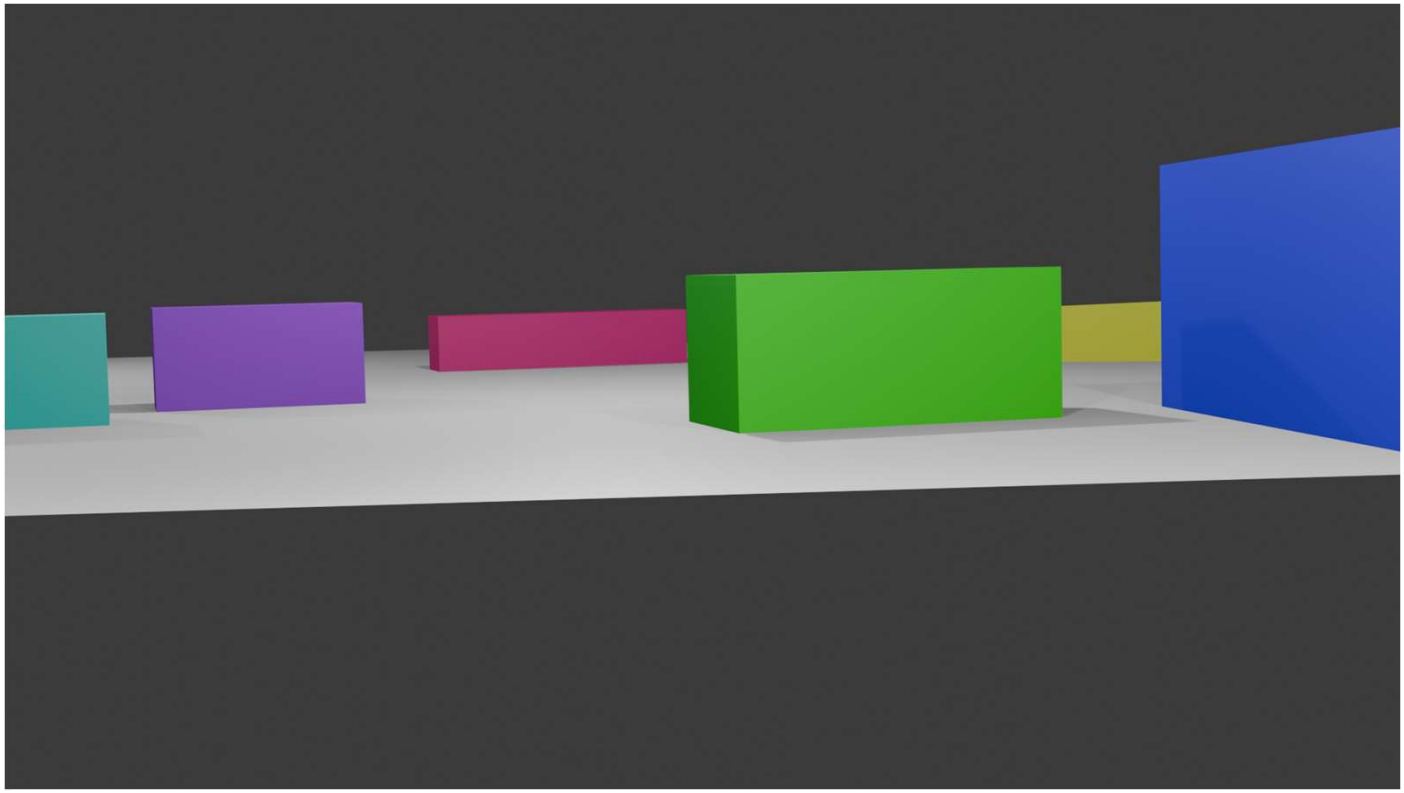
PAINTER'S ALGORITHM



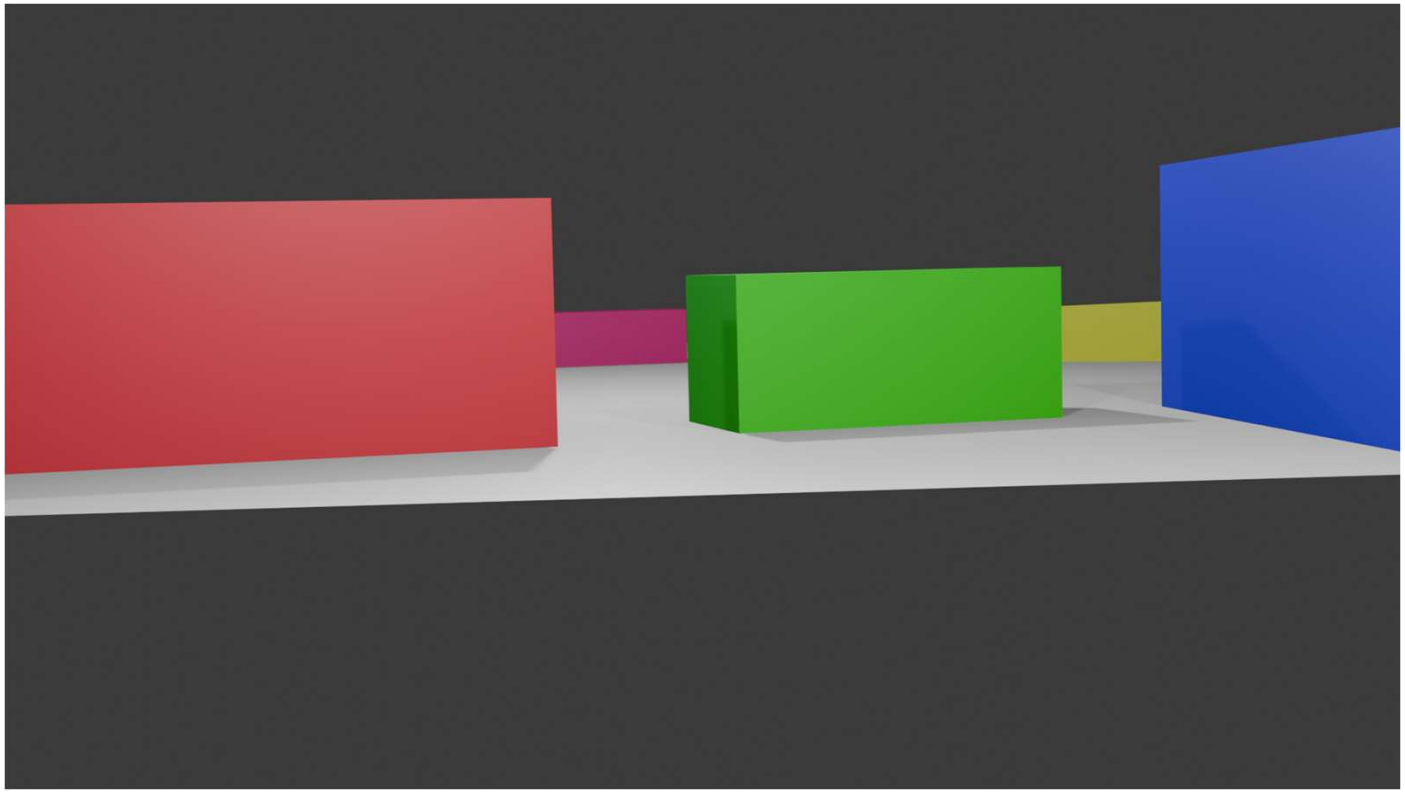
PAINTER'S ALGORITHM



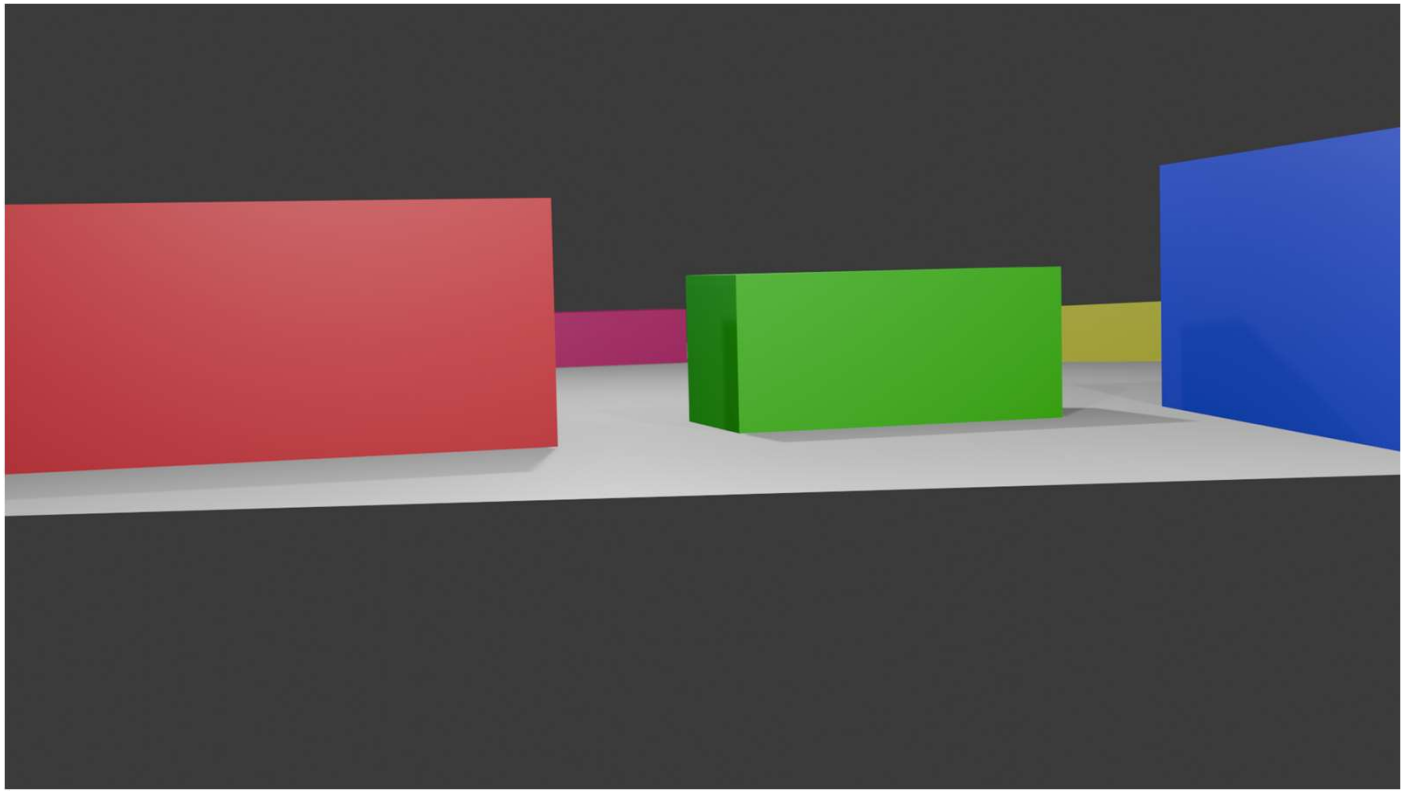
PAINTER'S ALGORITHM



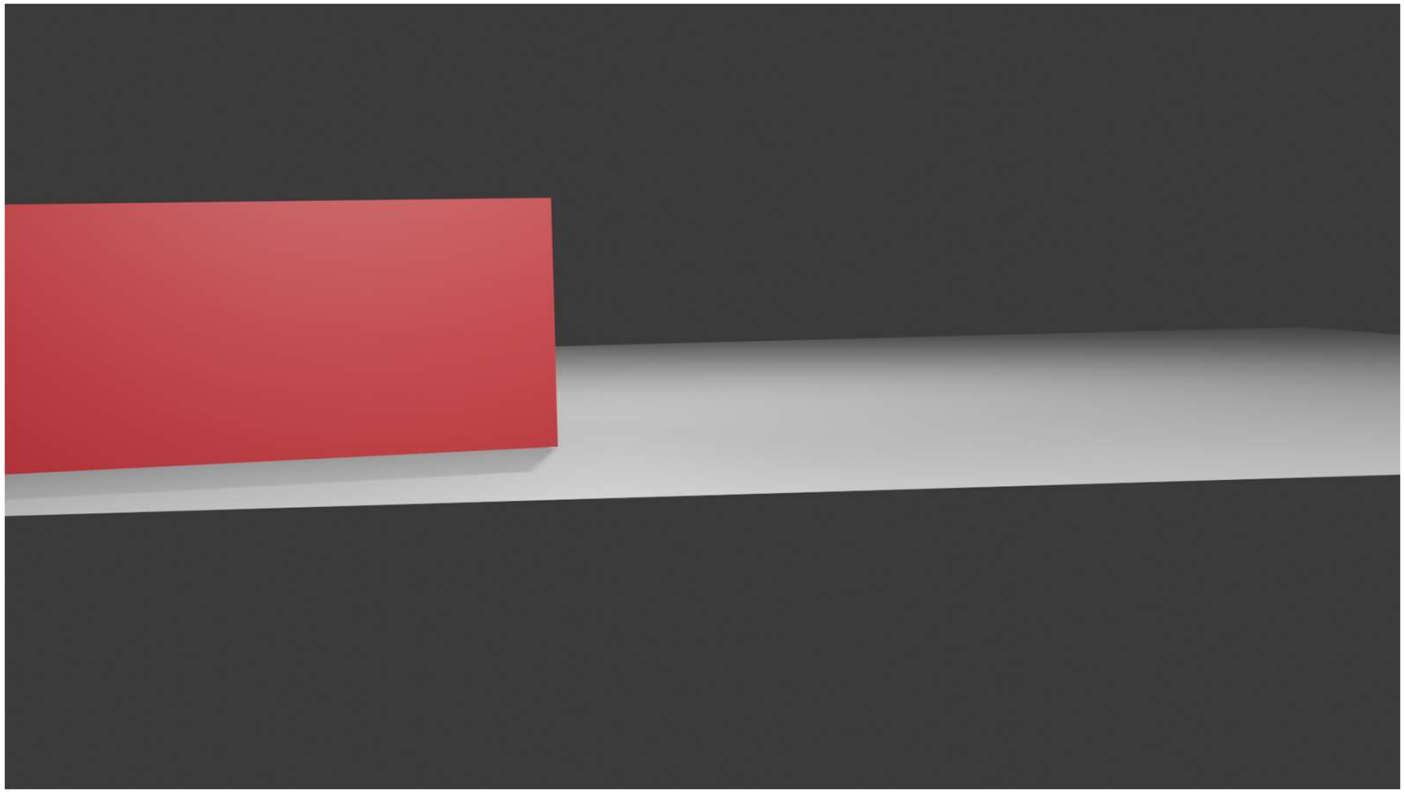
PAINTER'S ALGORITHM



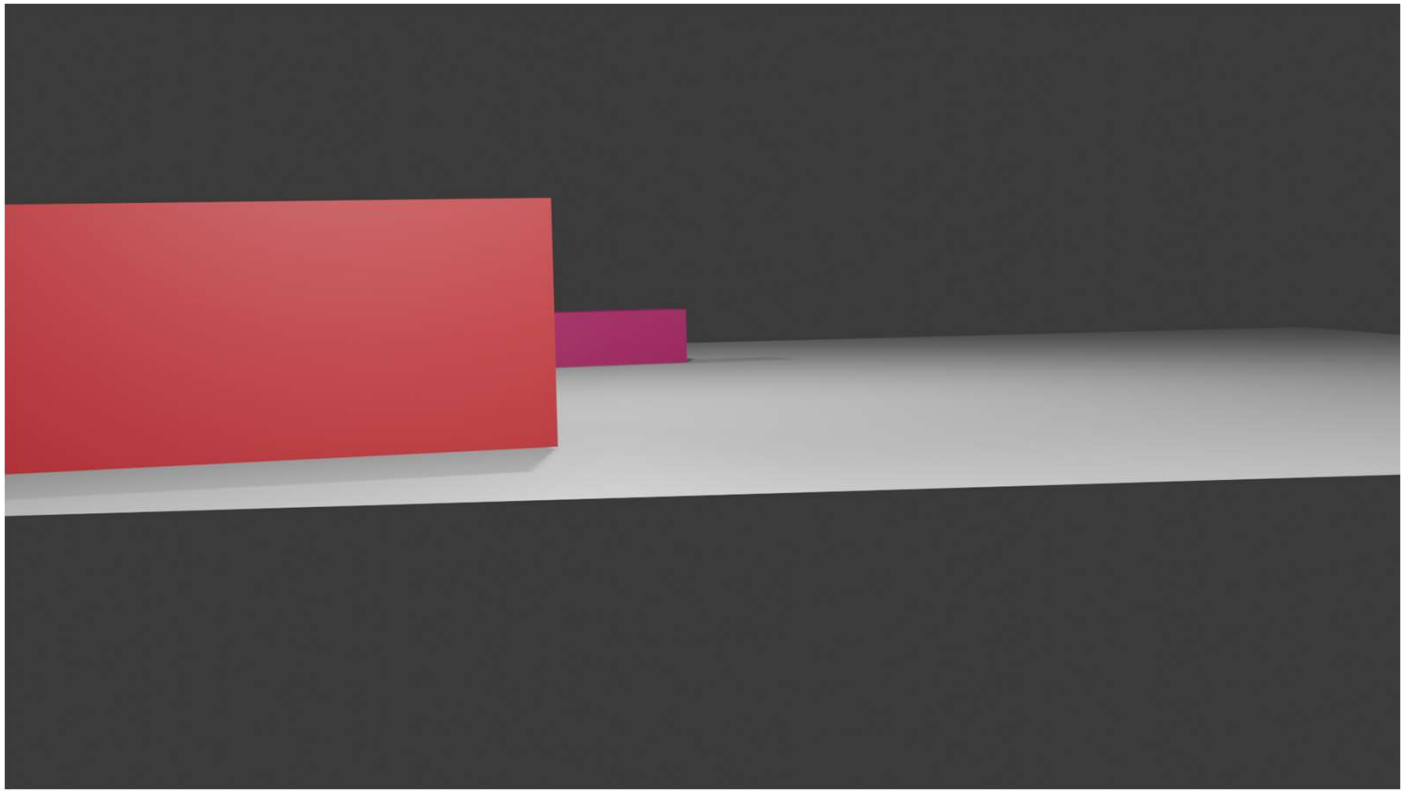
PAINTER'S ALGORITHM



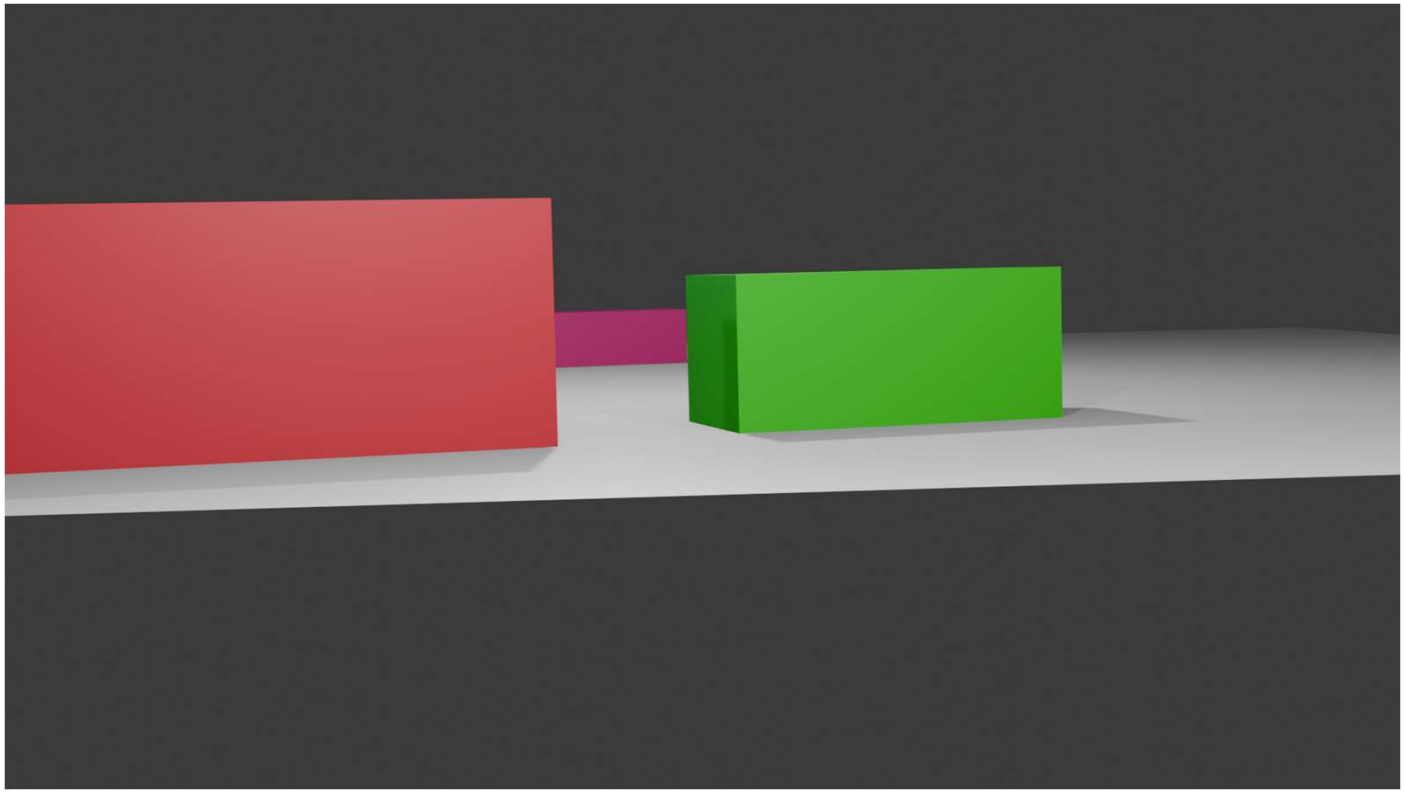
RAY CASTING



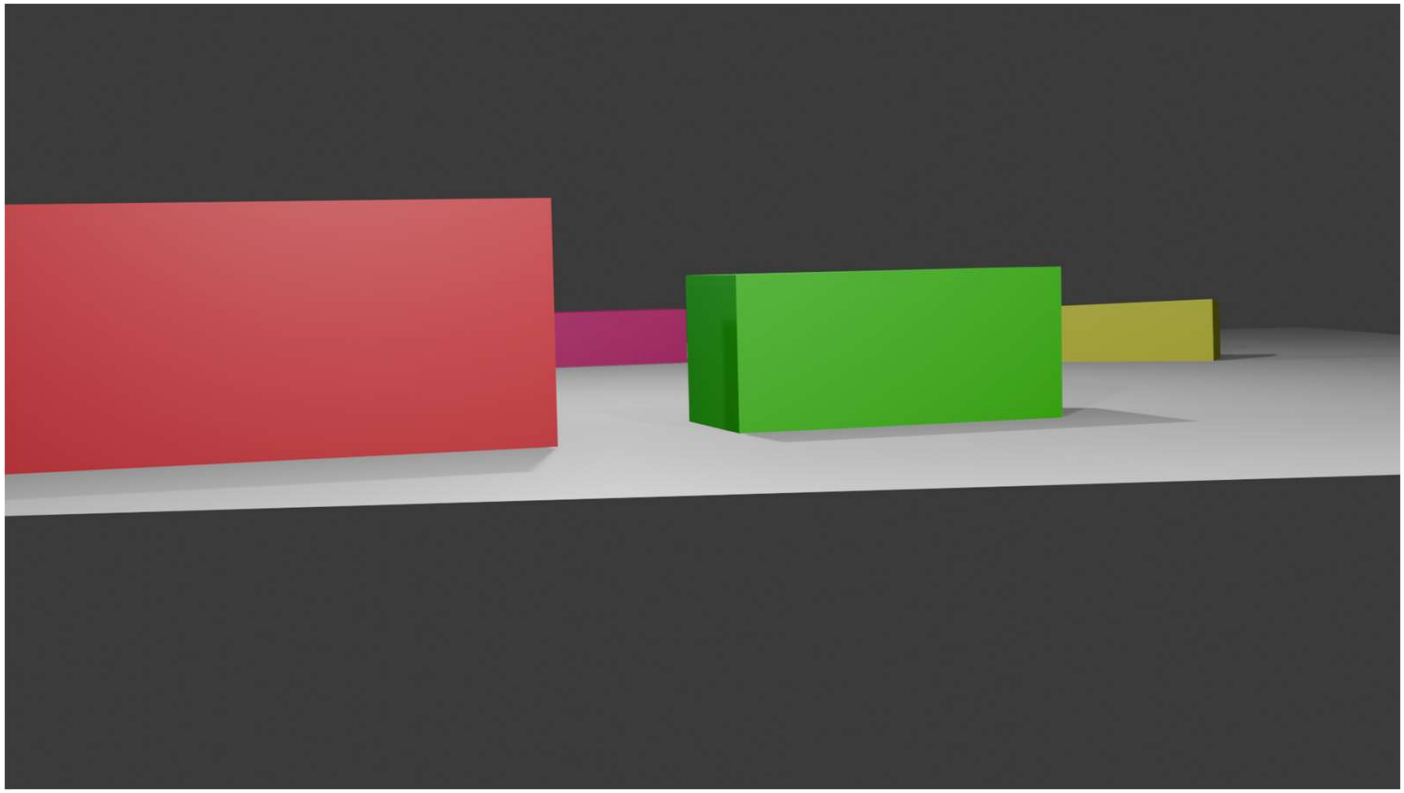
RAY CASTING



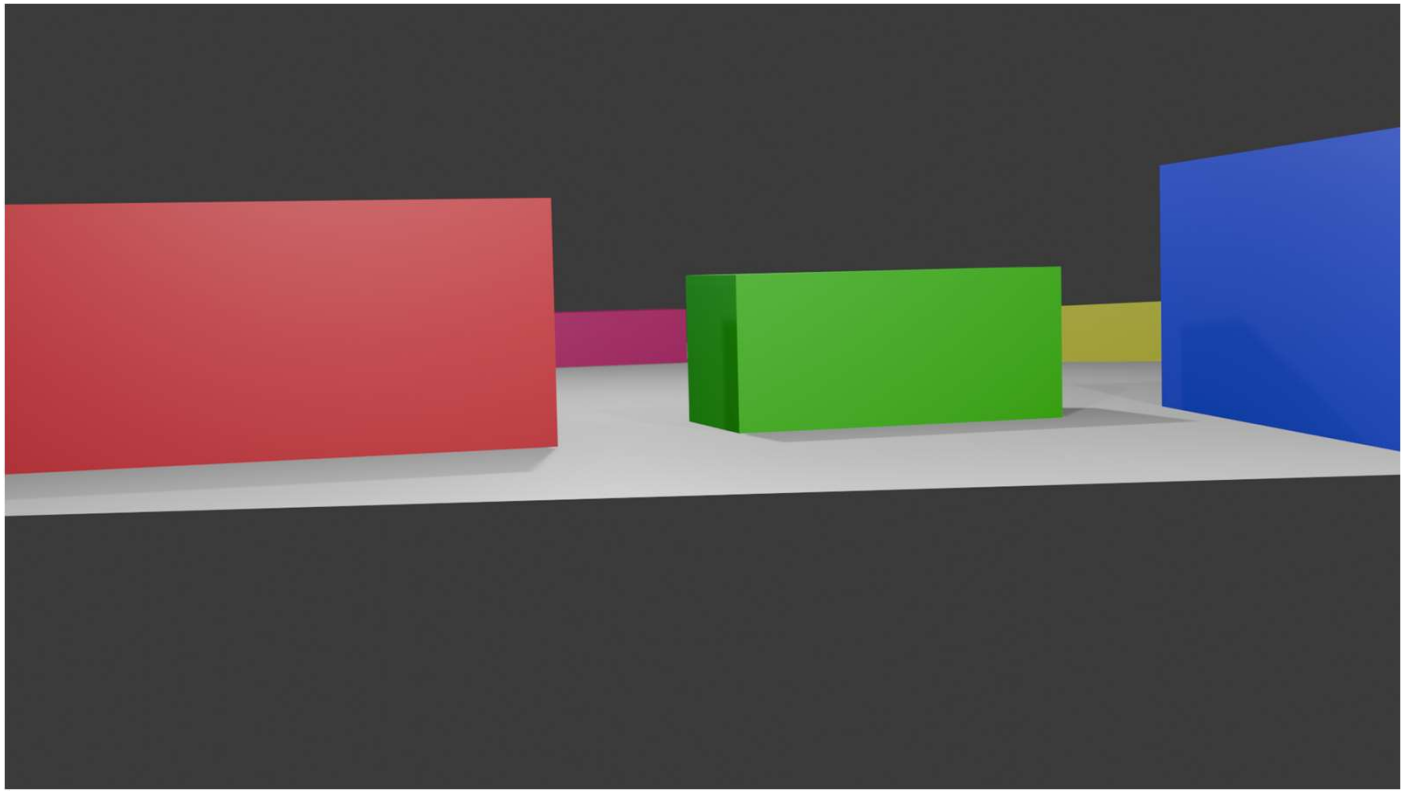
RAY CASTING



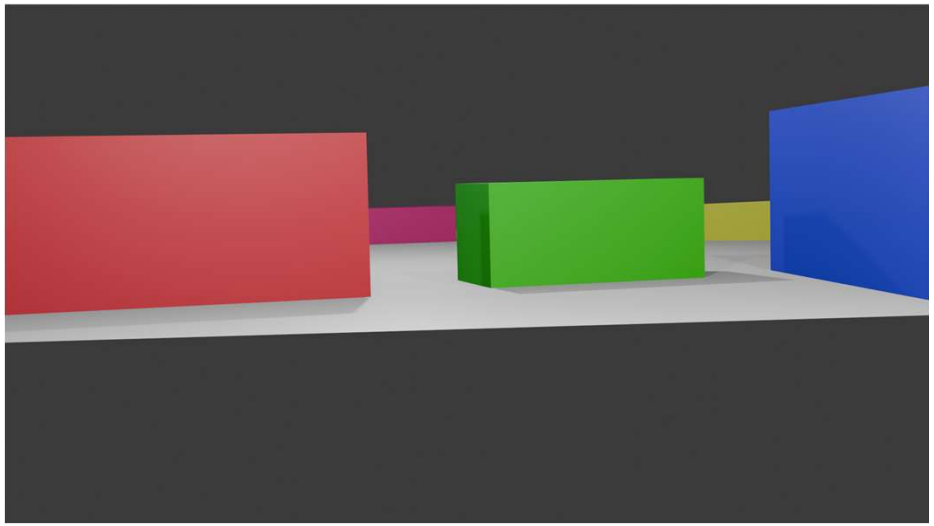
RAY CASTING

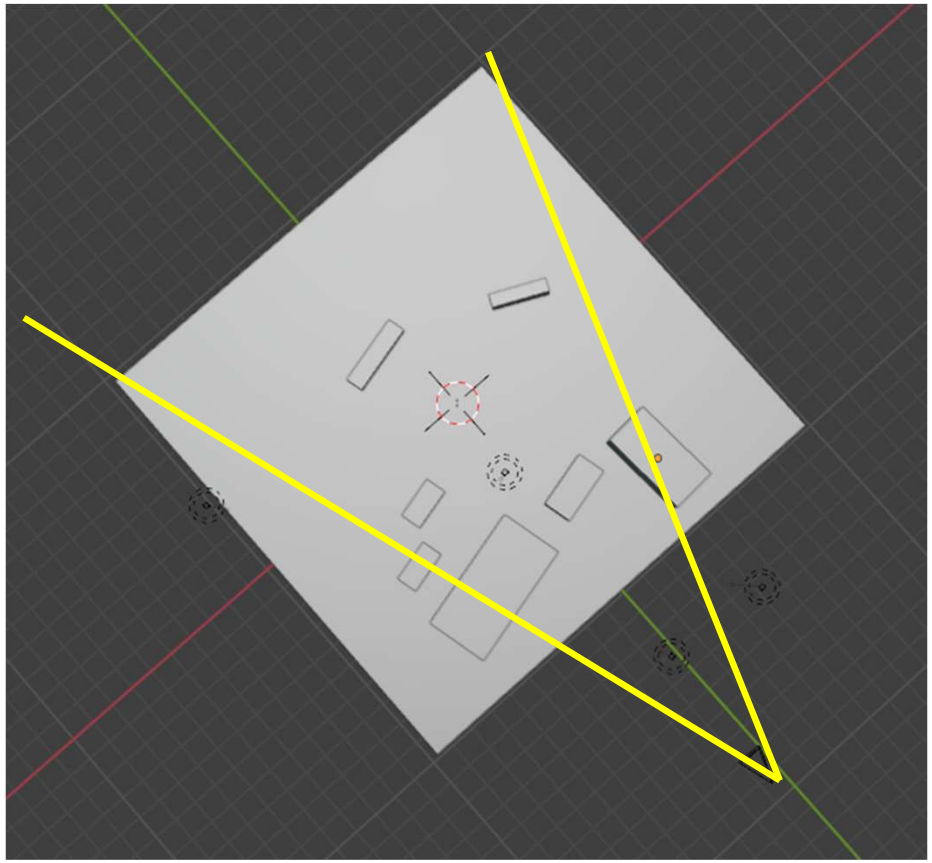


RAY CASTING

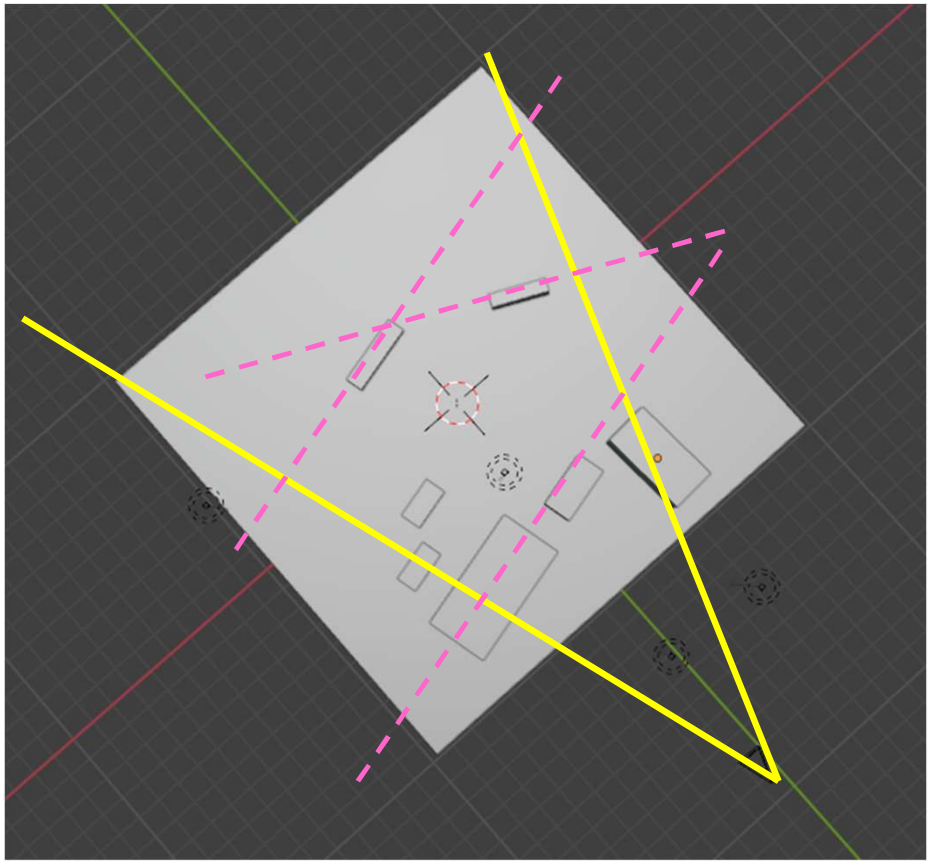


WHAT HAPPENS IN CASE OF BSP TREE





SPACE PARTITIONING



FLAME UNIVERSITY

EVERLASTING *learning*

THANK YOU