Computer Graphics

by Ruen-Rone Lee
ICL/ITRI

Wrap up from last week

- Lighting
- Illumination Model
  - Ambient
  - Diffuse
  - Specular

Introduction to Computer Graphics

Part I: Basic Hidden Surface Removal

Back-Face Culling

Z-buffering

BSP Tree

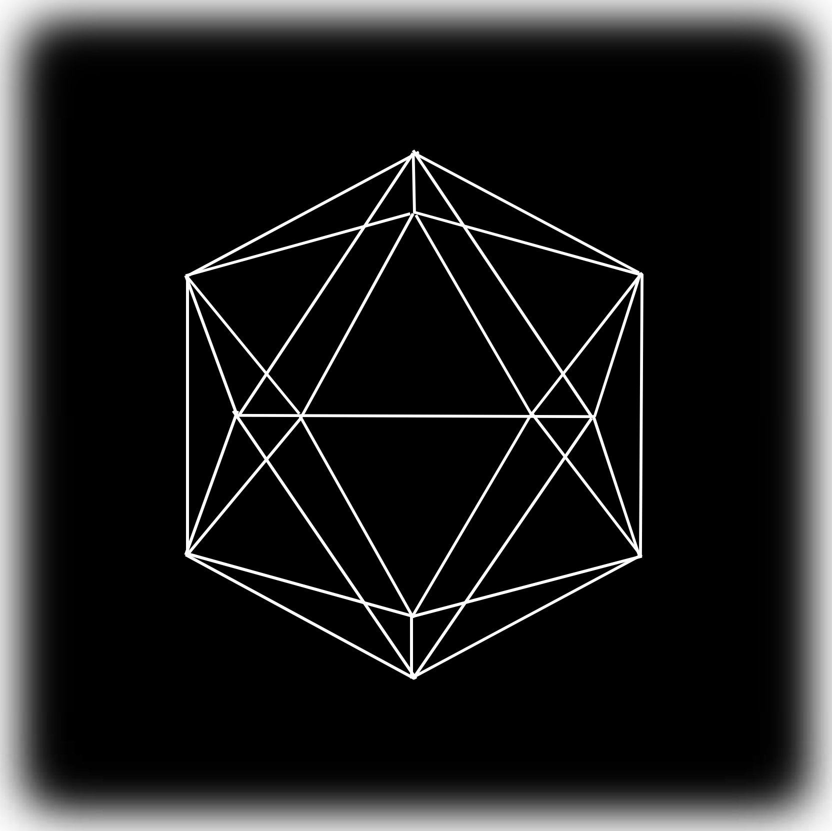
Portal Culling

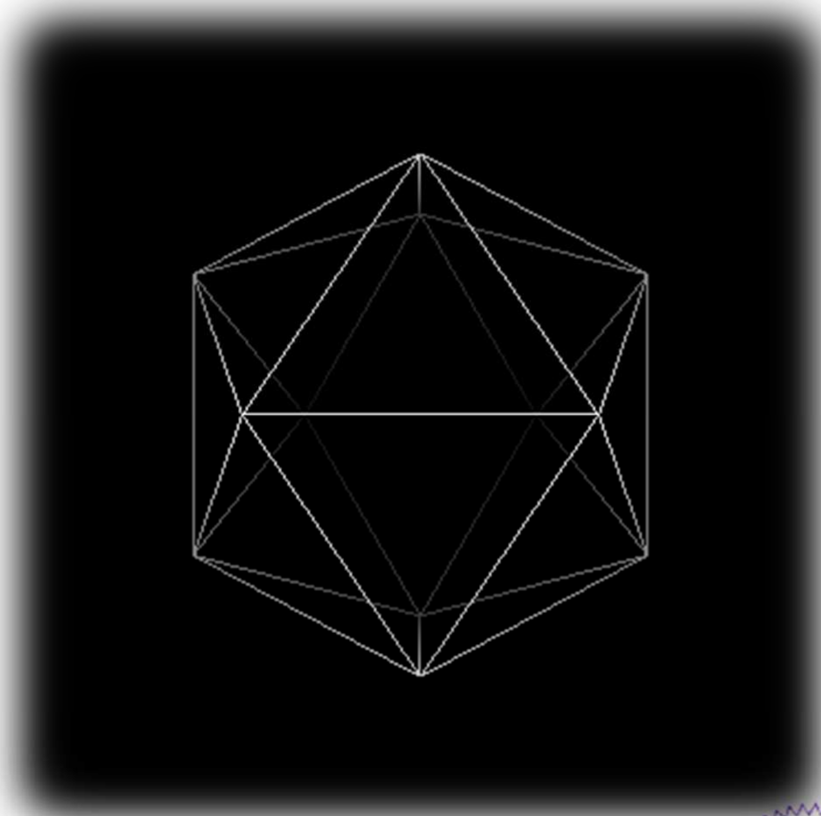
Objectives of Hidden Surface Removal

- Remove invisible surfaces, polygons, or lines
- Reduce unnecessary computation for invisible surfaces
- Obtain better perception of depth information

Depth Cue

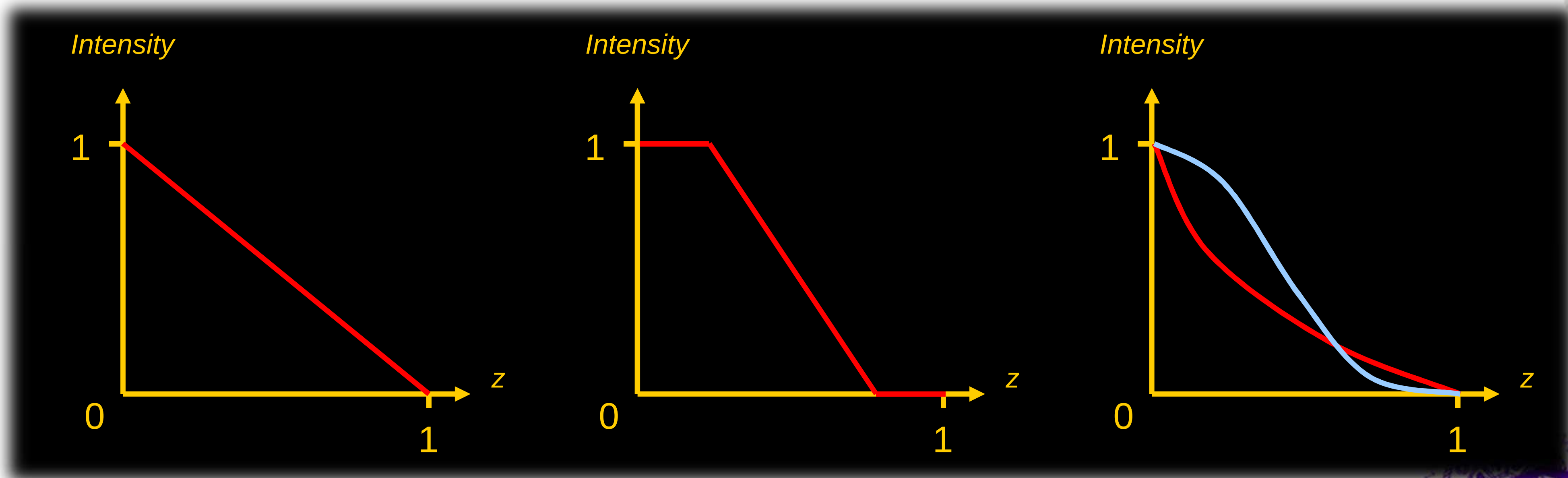
- Wireframe Display





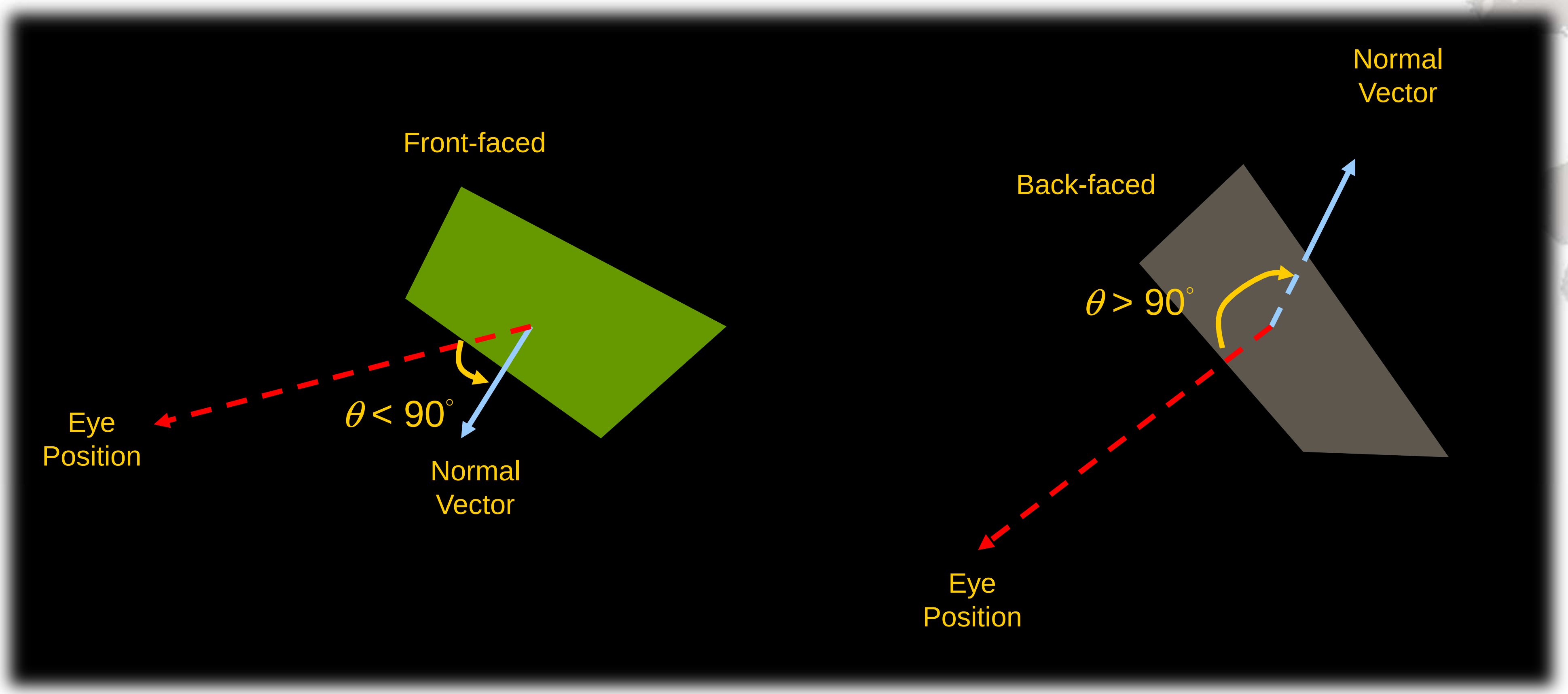
Depth Cue

- Implementation
  - e.g., $Intensity = f(z) = 1/z,\ 0 \le z \le 1$
  - Map the intensity values within the range of [0, 255]



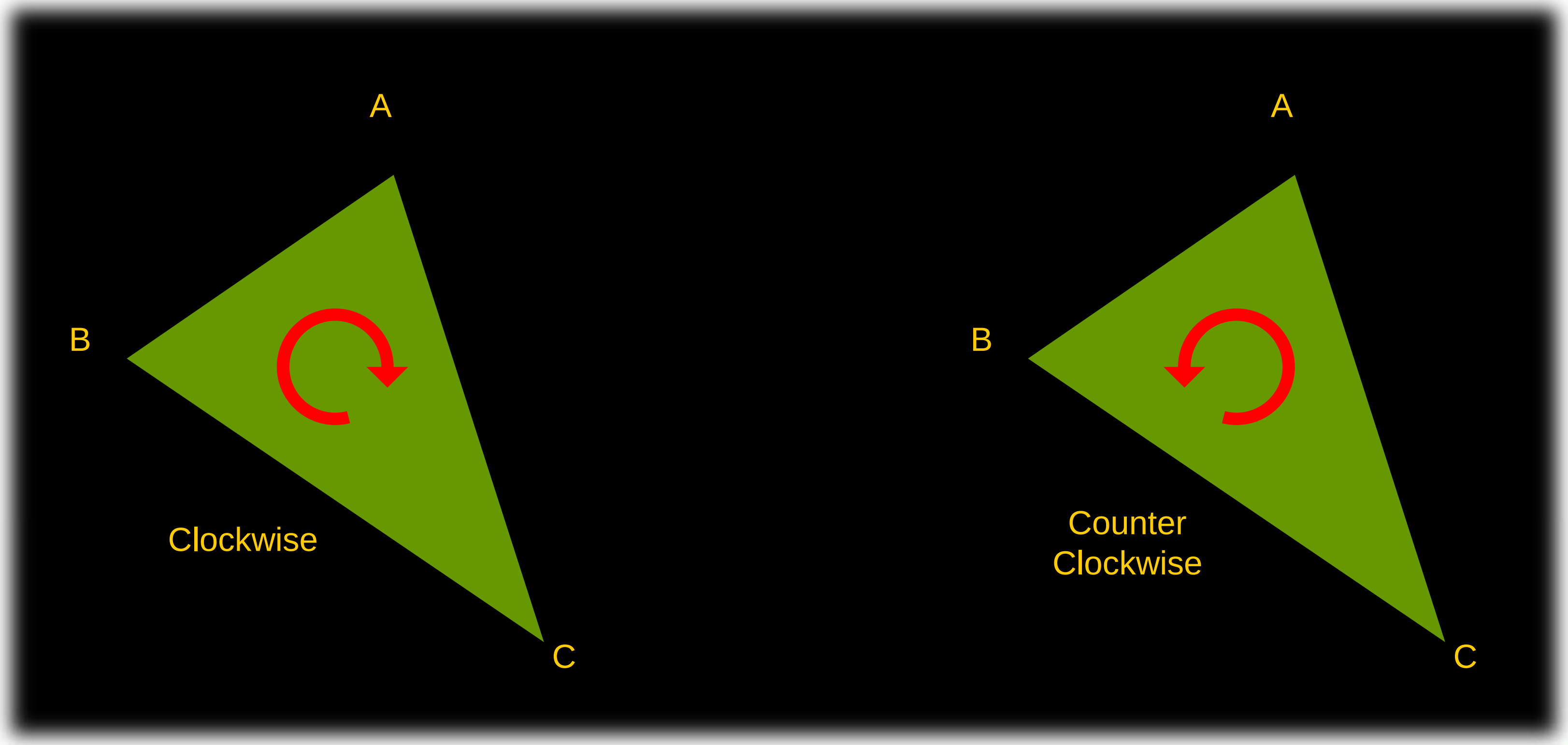
Back-Face Culling

- Back-face determination



Back-Face Culling

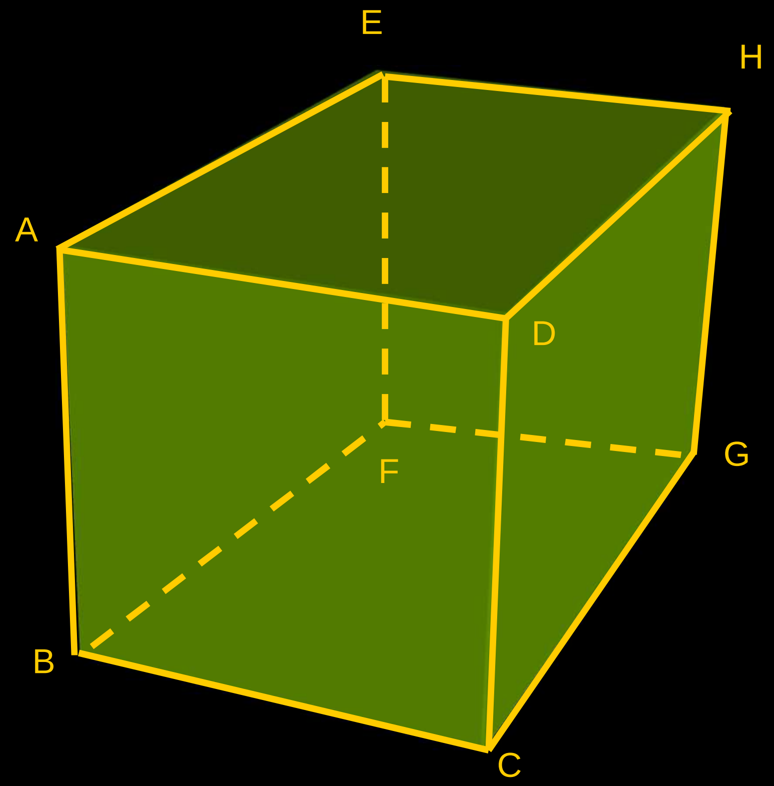
- Define clockwise or counter clockwise for front-faced polygon



Back-Face Culling

- **Assume the counter clockwise polygon is the front-faced polygon**

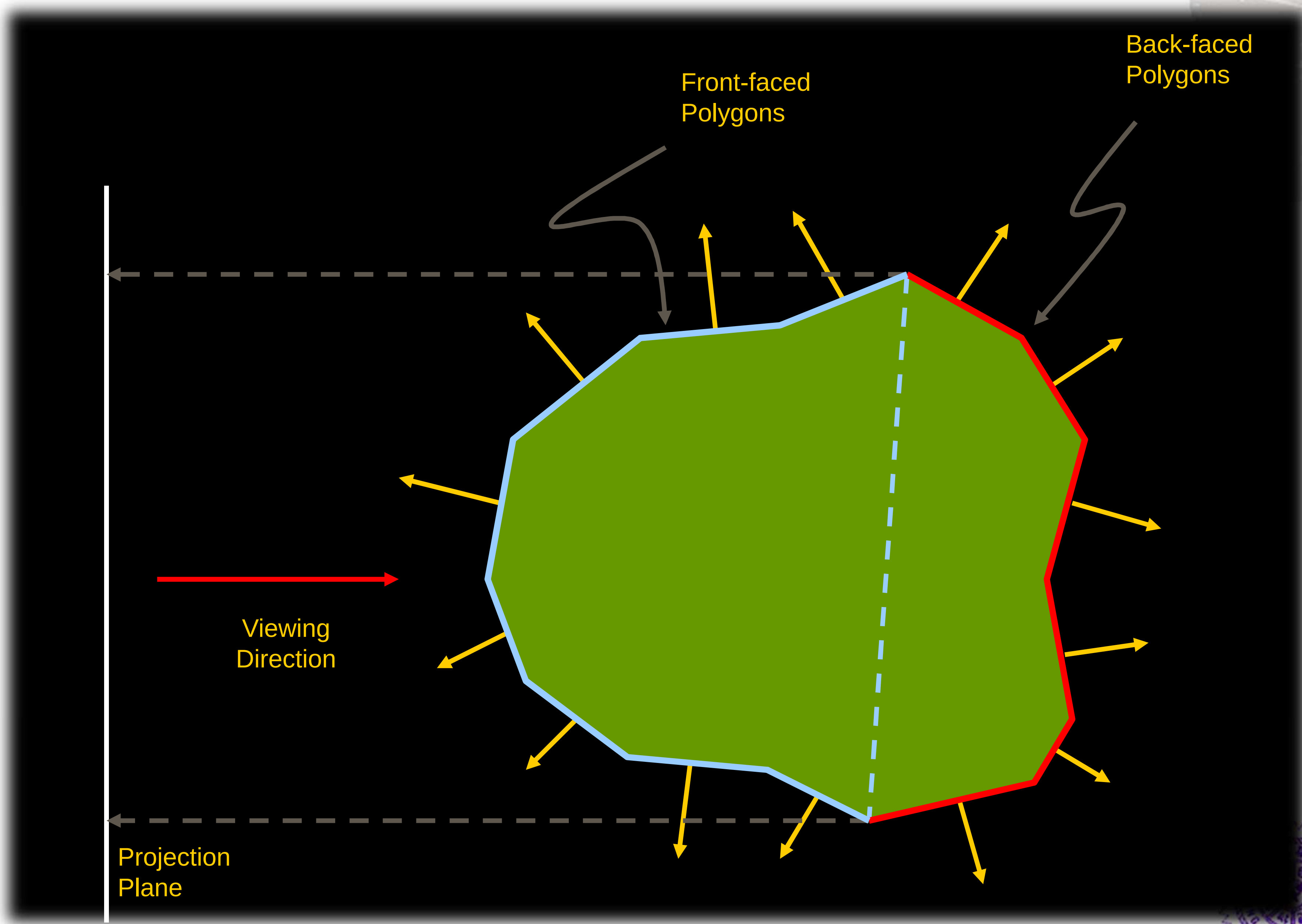
Polygon(A, B, C, D)	Front-faced
Polygon(D, C, G, H)	Front-faced
Polygon(H, G, F, E)	Back-faced
Polygon(E, F, B, A)	Back-faced
Polygon(A, D, H, E)	Front-faced
Polygon(F, G, C, B)	Back-faced



Back-Face Culling

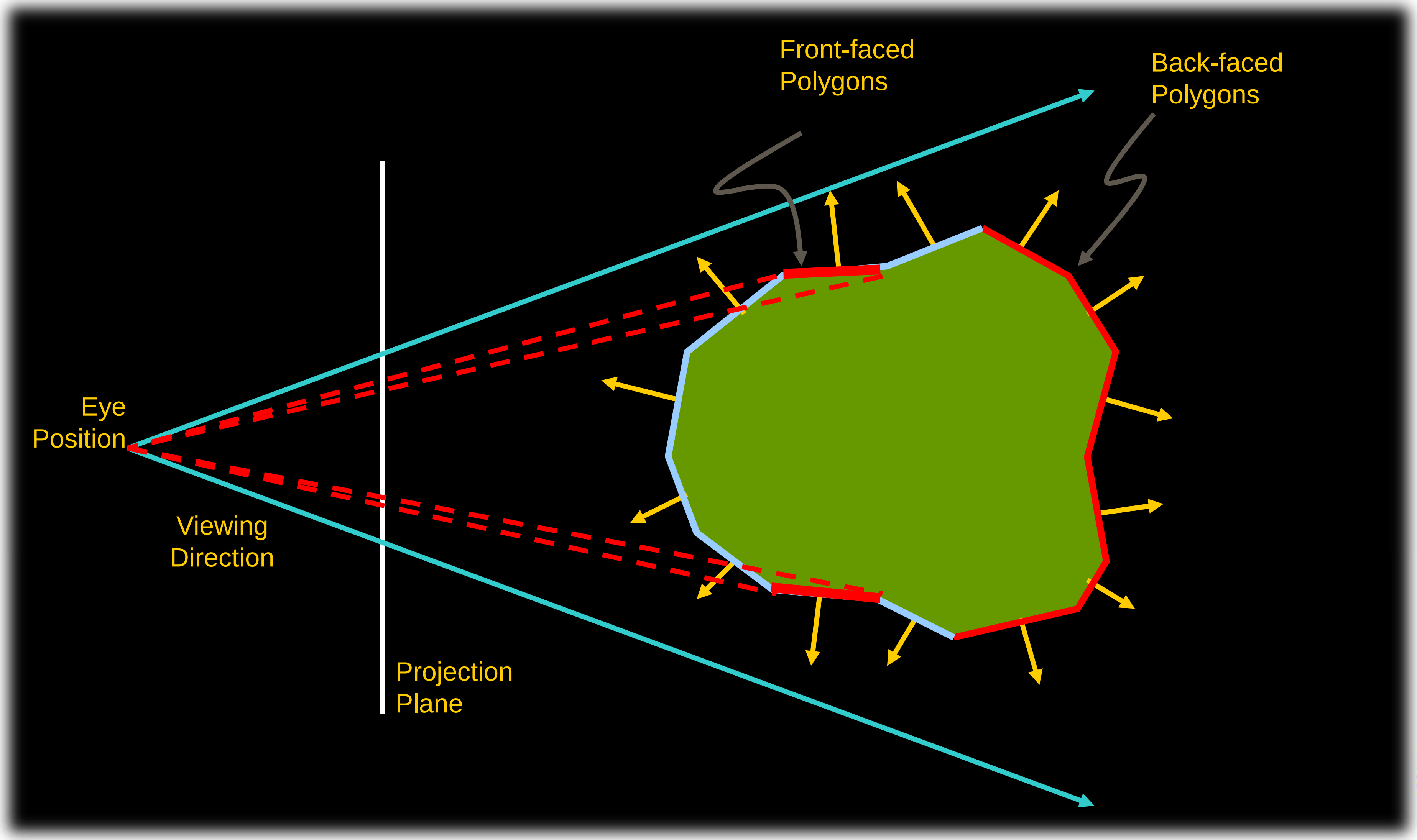
- Apply cross product to determine the normal vector of a polygon
- A dot product of the eye vector and the normal vector determines the facing attribute
  - $N \bullet E > 0$ implies front-facing
  - $N \bullet E < 0$ implies back-facing
- ps. if clockwise is defined to be the front-faced then $N = -N$ before applying dot product

Back-Face Culling



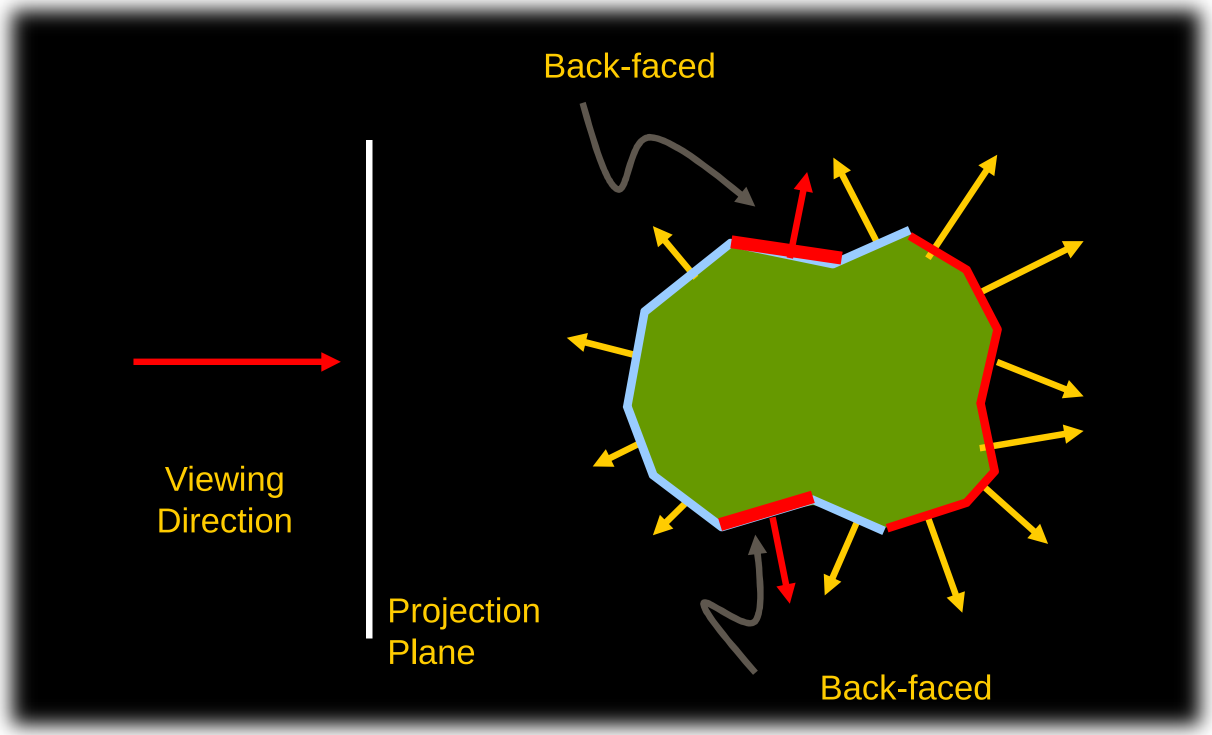
Back-Face Culling

- Why consider parallel projection only?



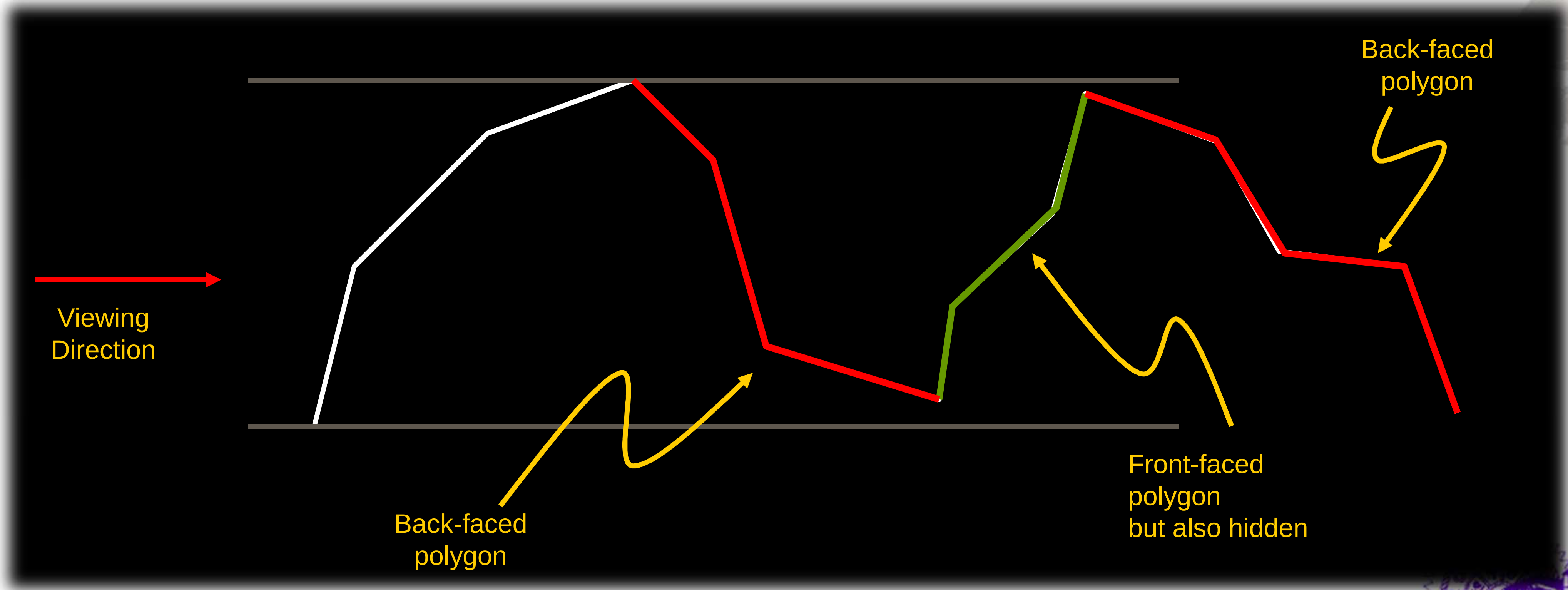
Back-Face Culling

- Why consider parallel projection only?
  - After perspective normalization, the polygon normals will toward the right directions to distinguish front-faced or back-faced polygons



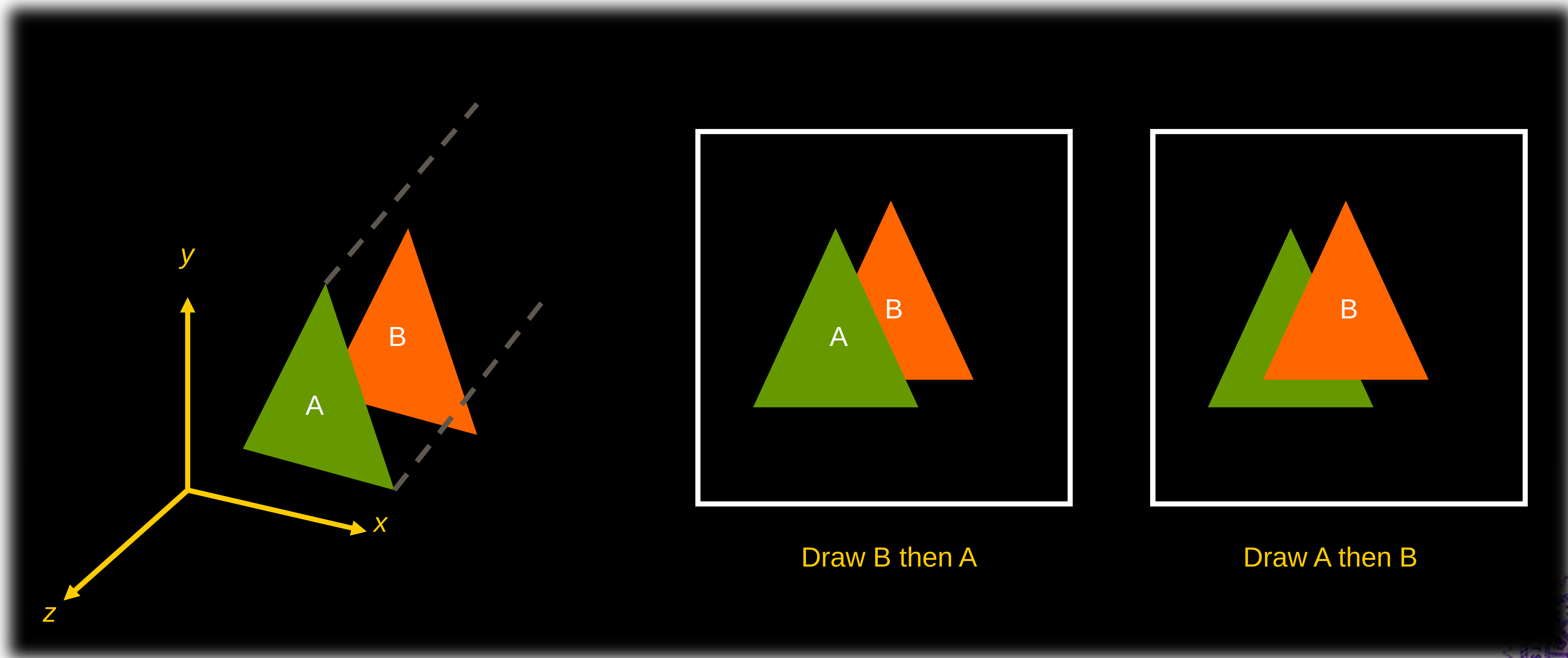
Back-Face Culling

- Back-face culling cannot guarantee right hidden surface removal. But, it can eliminate invisible surfaces easily.



Display Order

- **The display order might derive in different results if using only back-face culling**



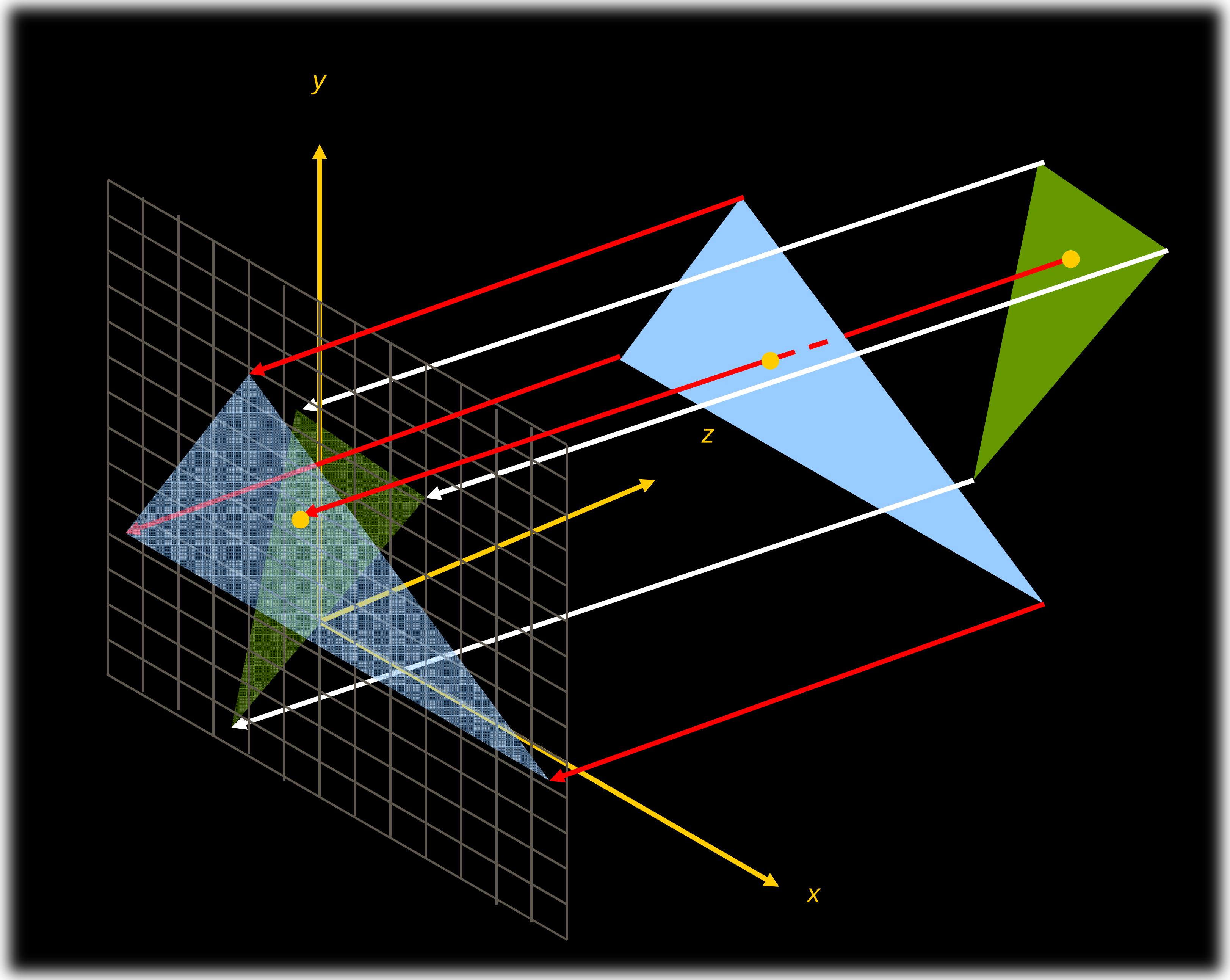
Z-Buffer Algorithm

- Eliminate the limitation of display order
- Require a memory space, equal to the size of display buffer, to store the depth / Z values
- Comparison is required to determine which pixel is closer to the viewer

Z-Buffer Algorithm

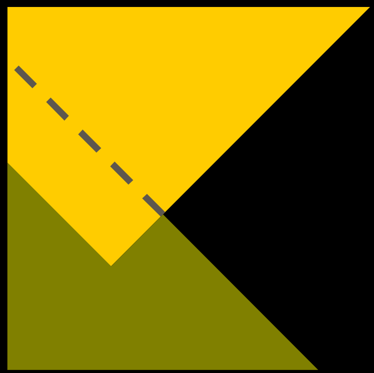
- Basic concept
  - Store the depth value of the closest object at each pixel found so far
- Render each polygon, pixel by pixel
- Compare the depth of each pixel of a polygon with the depth of the corresponding depth value in Z-buffer
- If the comparison pass, then render the color to color buffer and replace the depth value by current pixel depth

Z-Buffer Algorithm



Z-Buffer Algorithm

Example: (larger z value is closer to viewer)



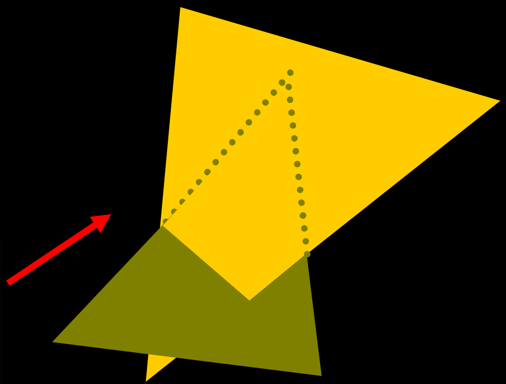
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

+

5	5	5	5	5	5	5	
5	5	5	5	5	5		
5	5	5	5	5			
5	5	5	5				
5	5	5					
5	5						
5							

=

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
5	5	5	0	0	0	0	0
5	5	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
5	5	5	0	0	0	0	0
5	5	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

+

3							
4	3						
5	4	3					
6	5	4	3				
7	6	5	4	3			
8	7	6	5	4	3		

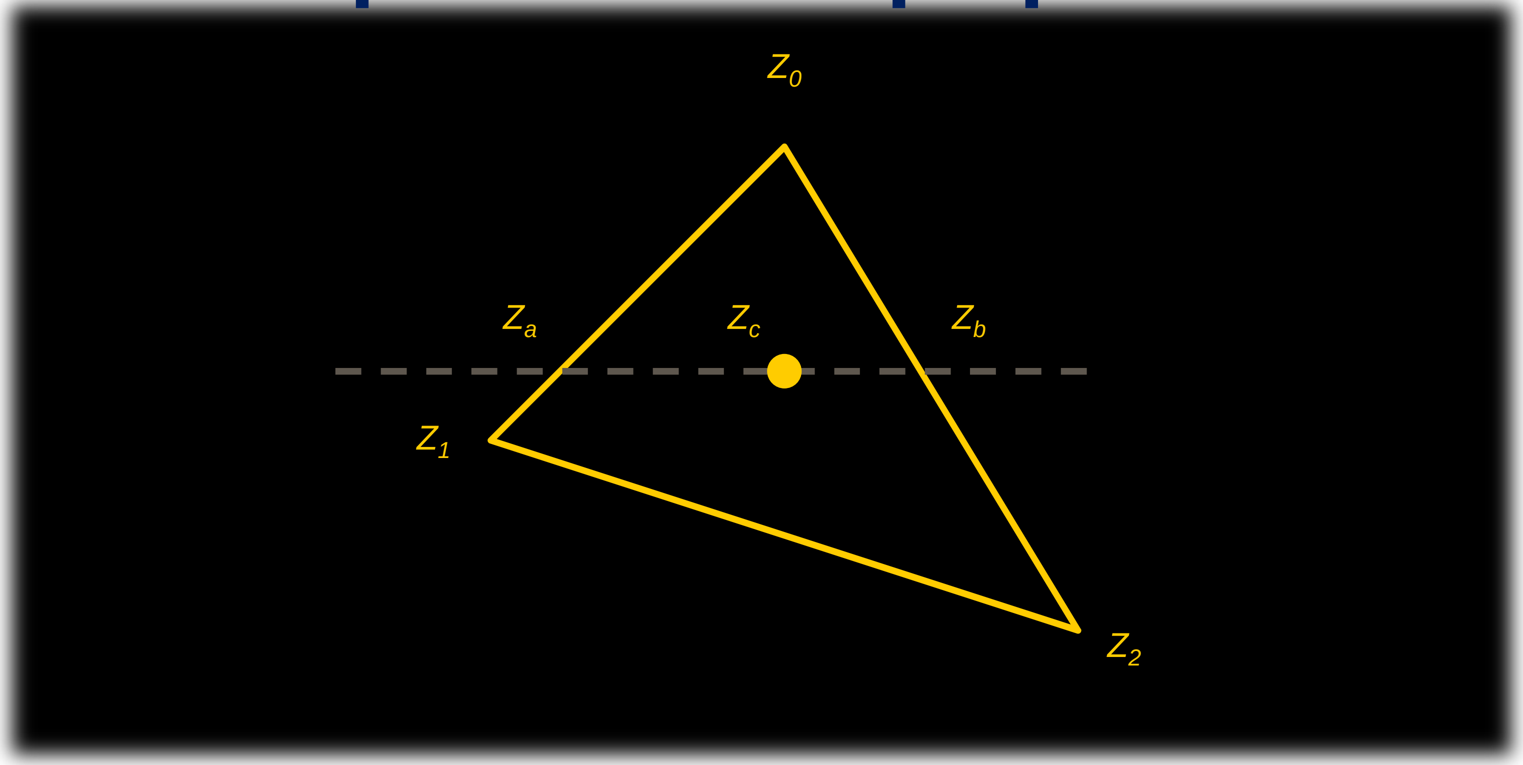
=

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
6	5	5	3	0	0	0	0
7	6	5	4	3	0	0	0
8	7	6	5	4	3	0	0
0	0	0	0	0	0	0	0

Choose of Depth Values

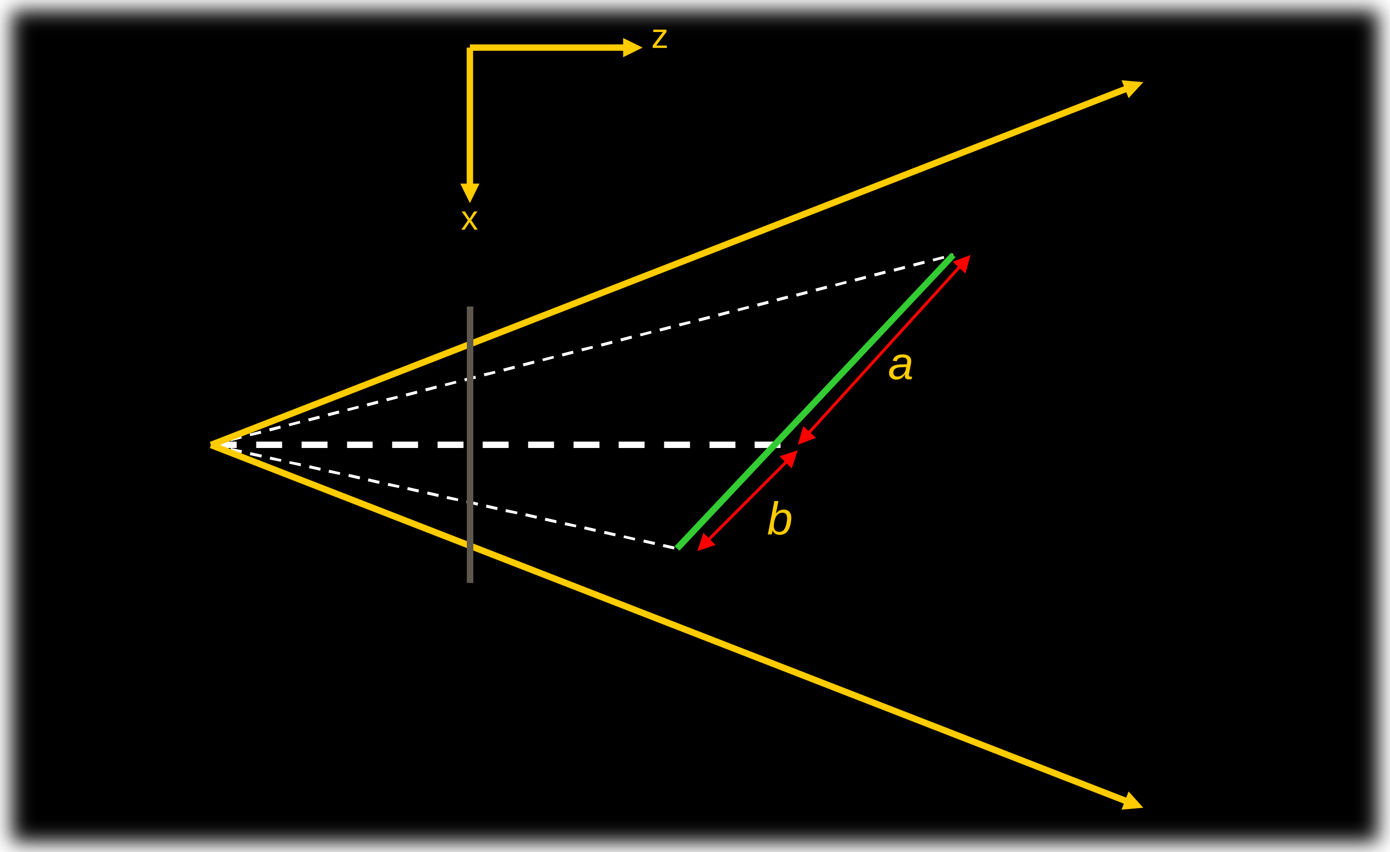
- When Interpolation Z in screen space, which Z values should be used?

World space Z or clip space Z?



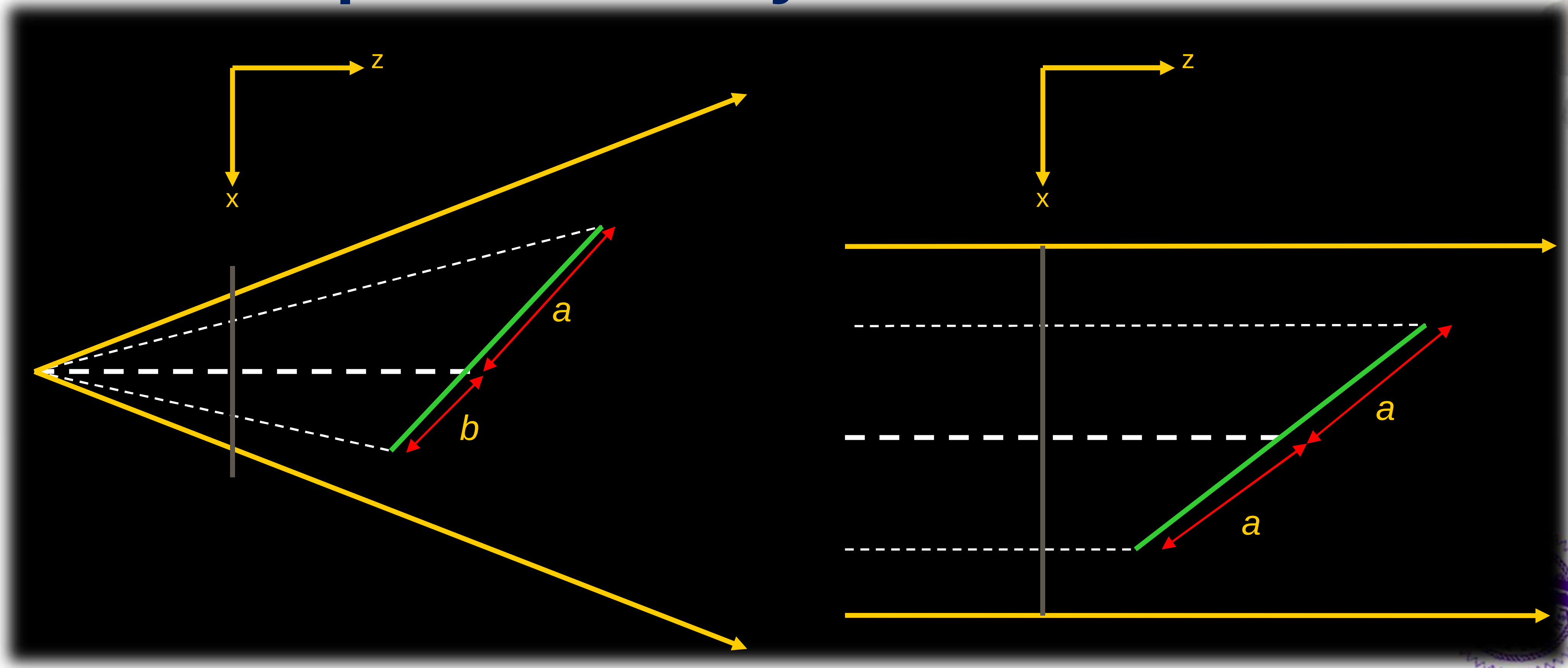
Choose of Depth Values

- **Linear interpolation for depth in screen space does not hold for depth interpolation in world space under perspective view**



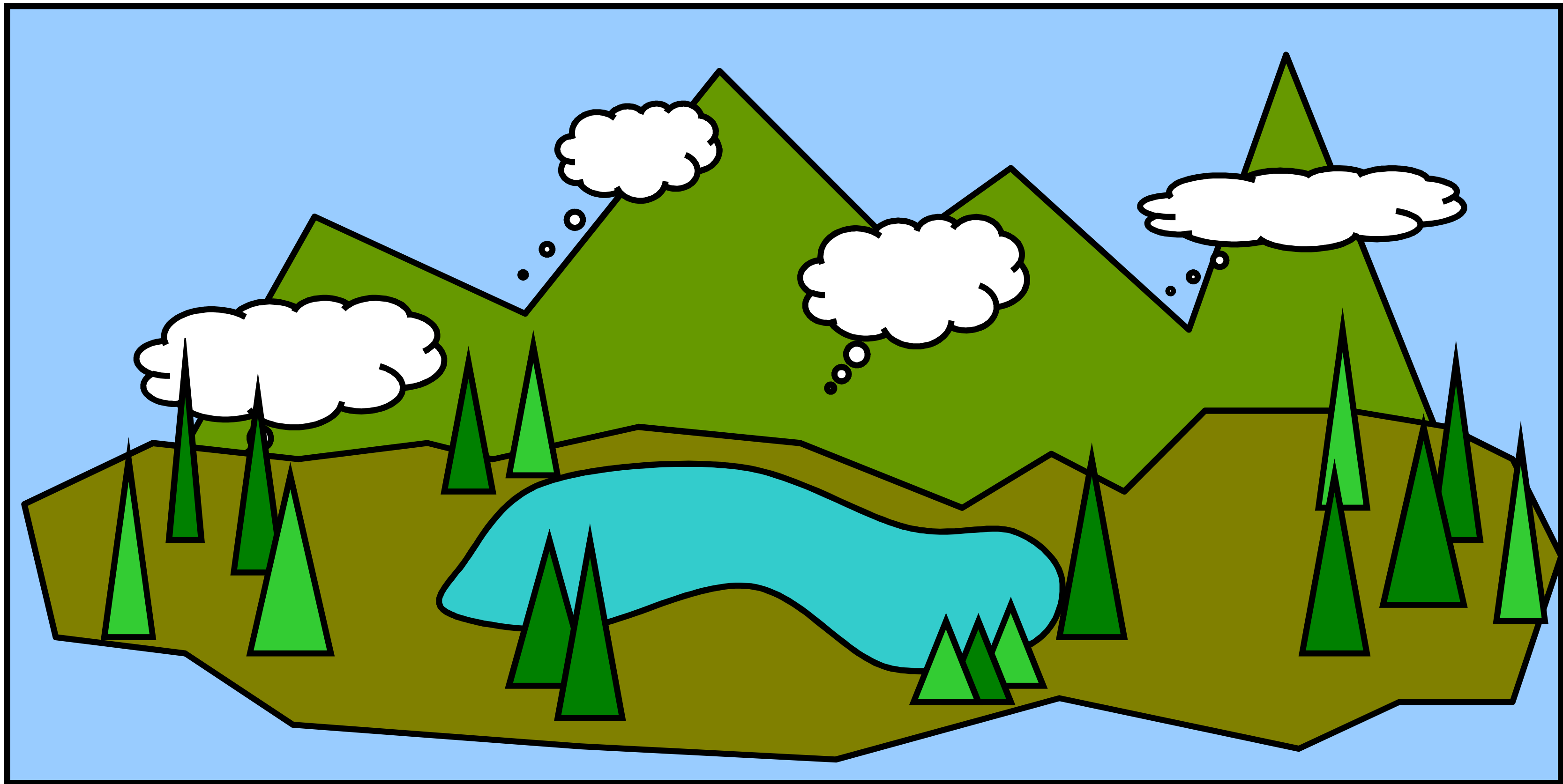
Choose of Depth Values

- After perspective transform and normalized into clip space, the depth value can be used to interpolate linearly



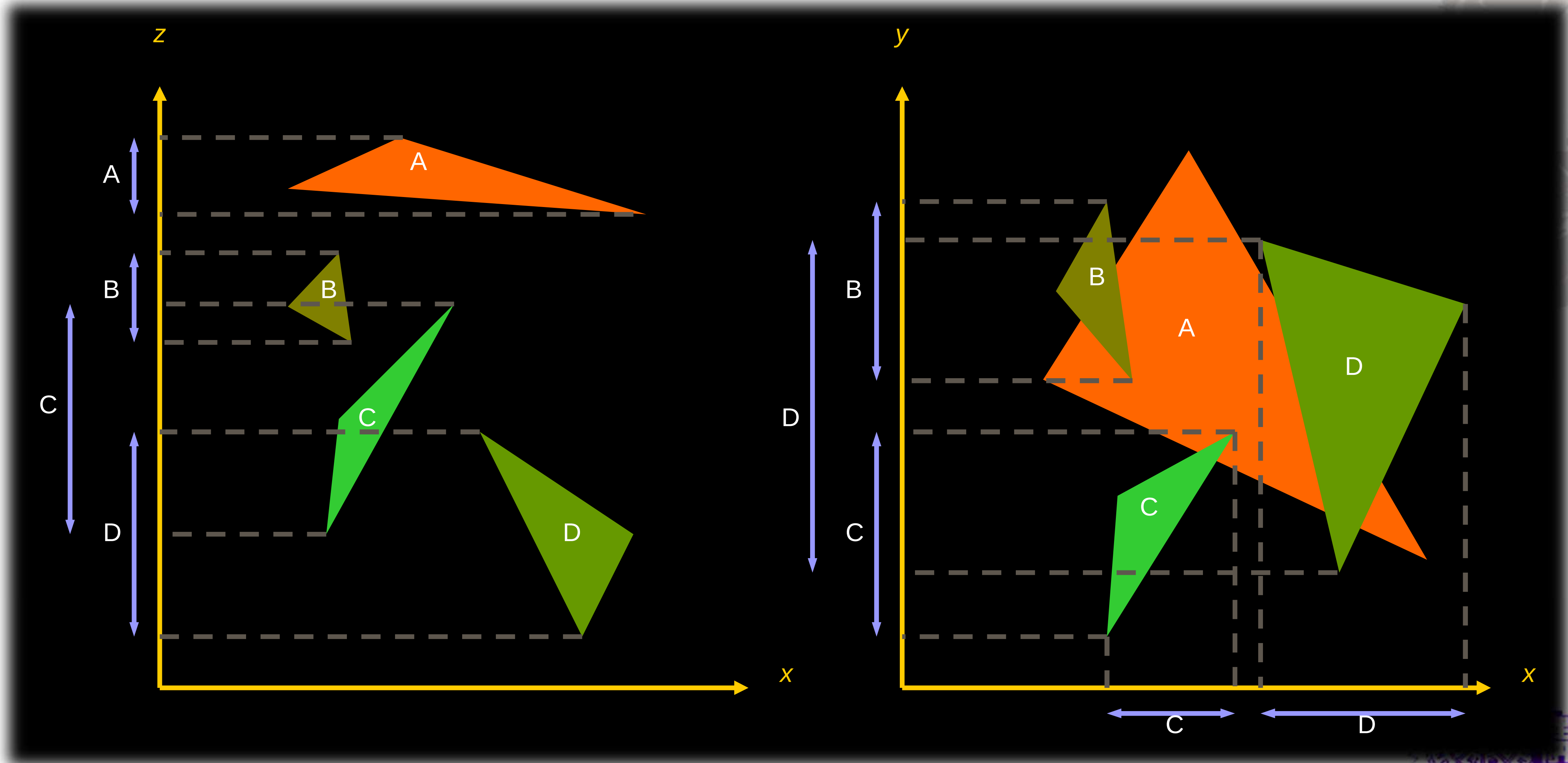
Painter's Algorithm

- **Back-to-Front Rendering**



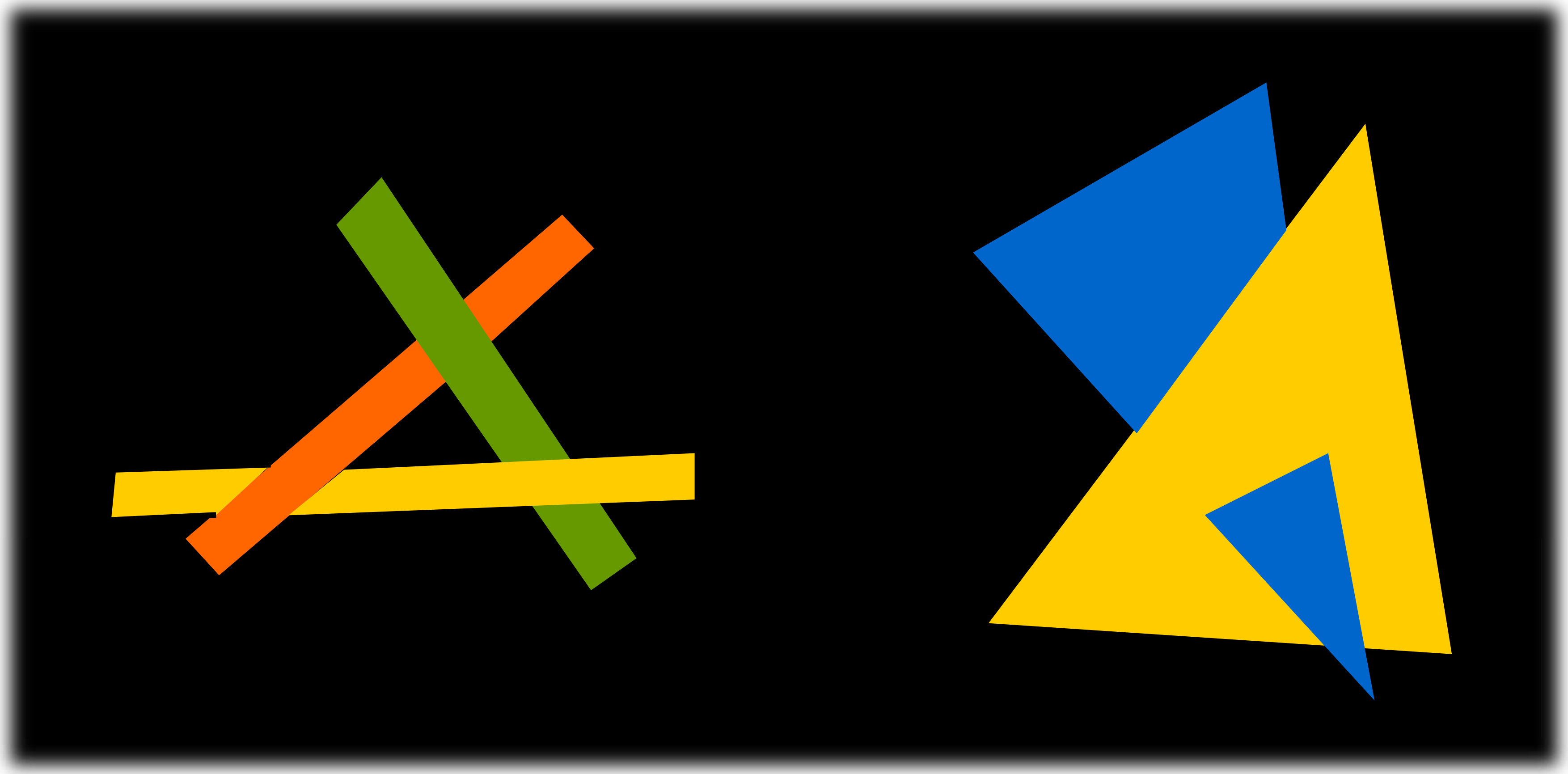
Depth Sort

◆ Sort the polygons in depth order



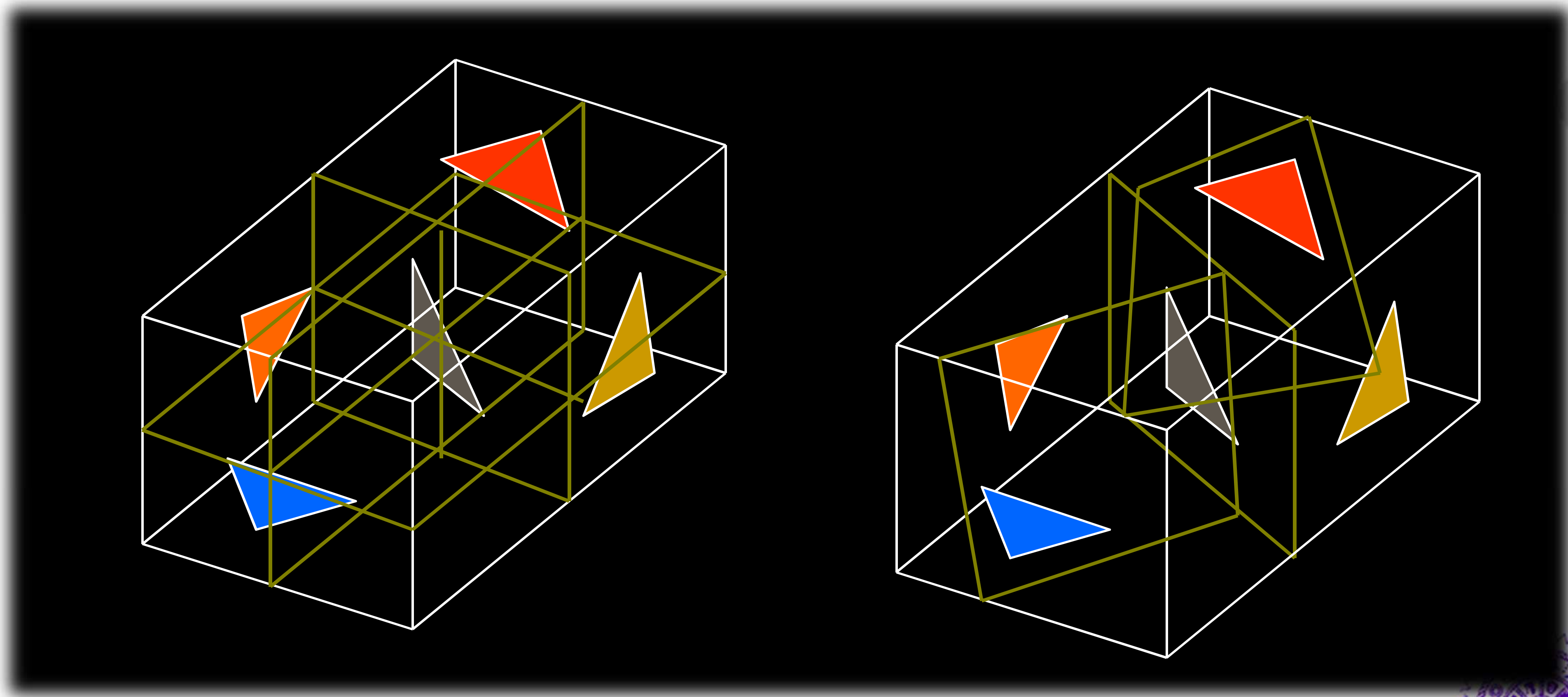
Difficult Cases

- Might need further subdivision to make it easy to identify the hidden surface



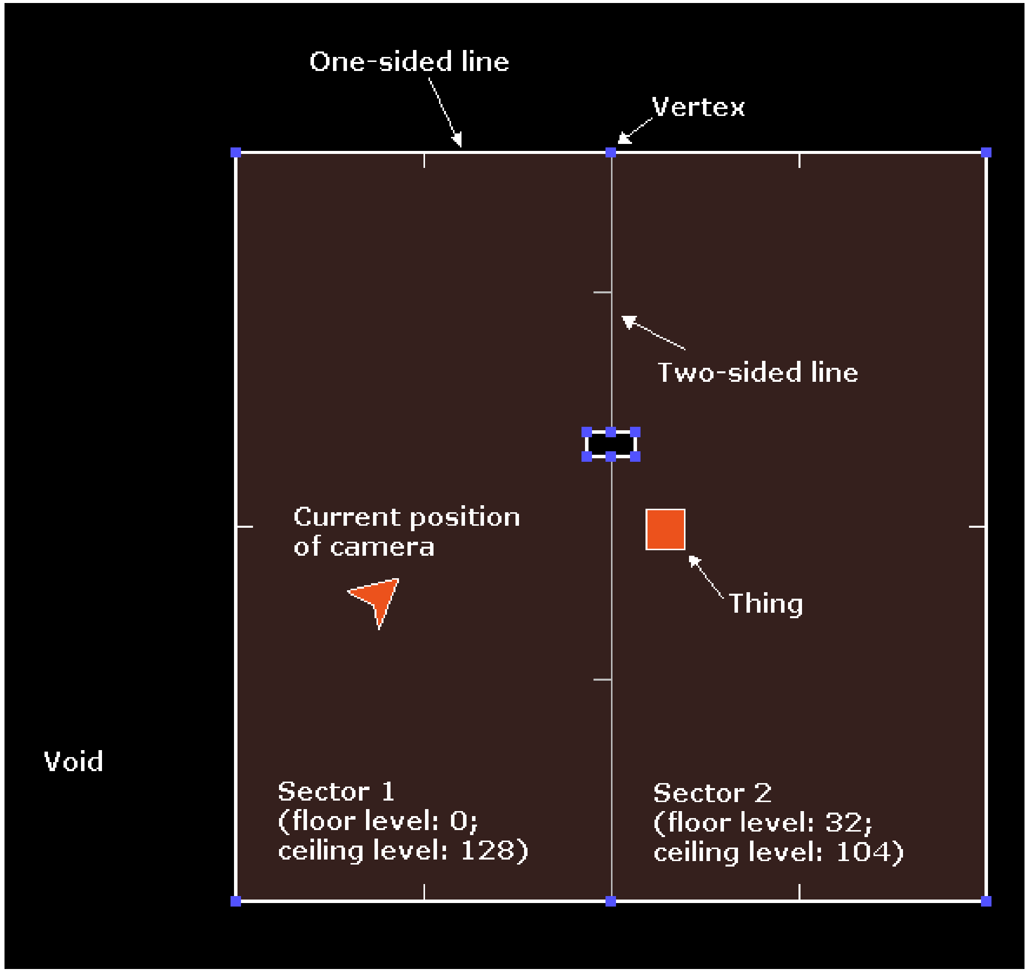
Binary Space Partition Trees

- **Axis Aligned Partition vs. Arbitrary Plane Partition**



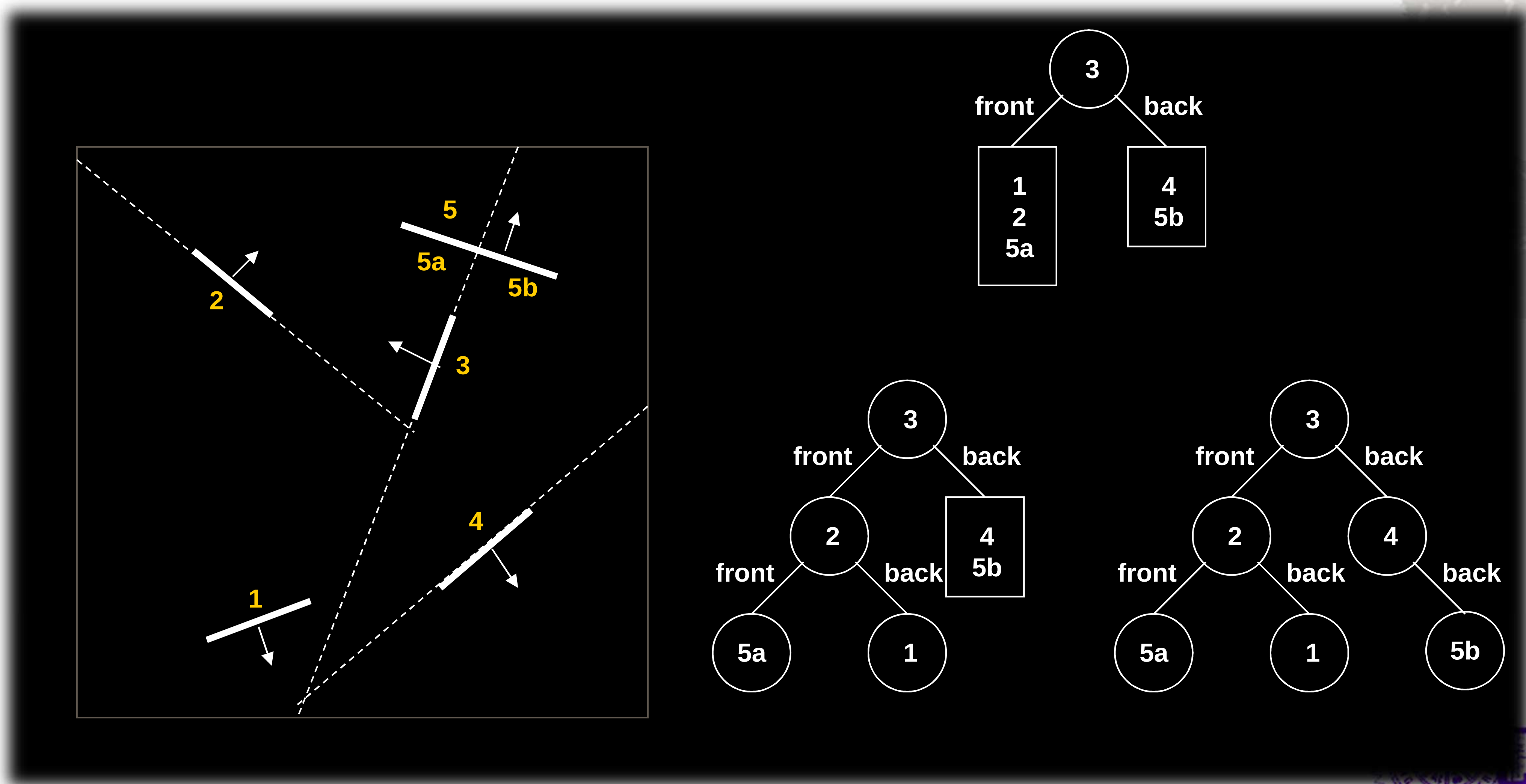
Games using BSP Tree

- Doom-like or Quake-like Games



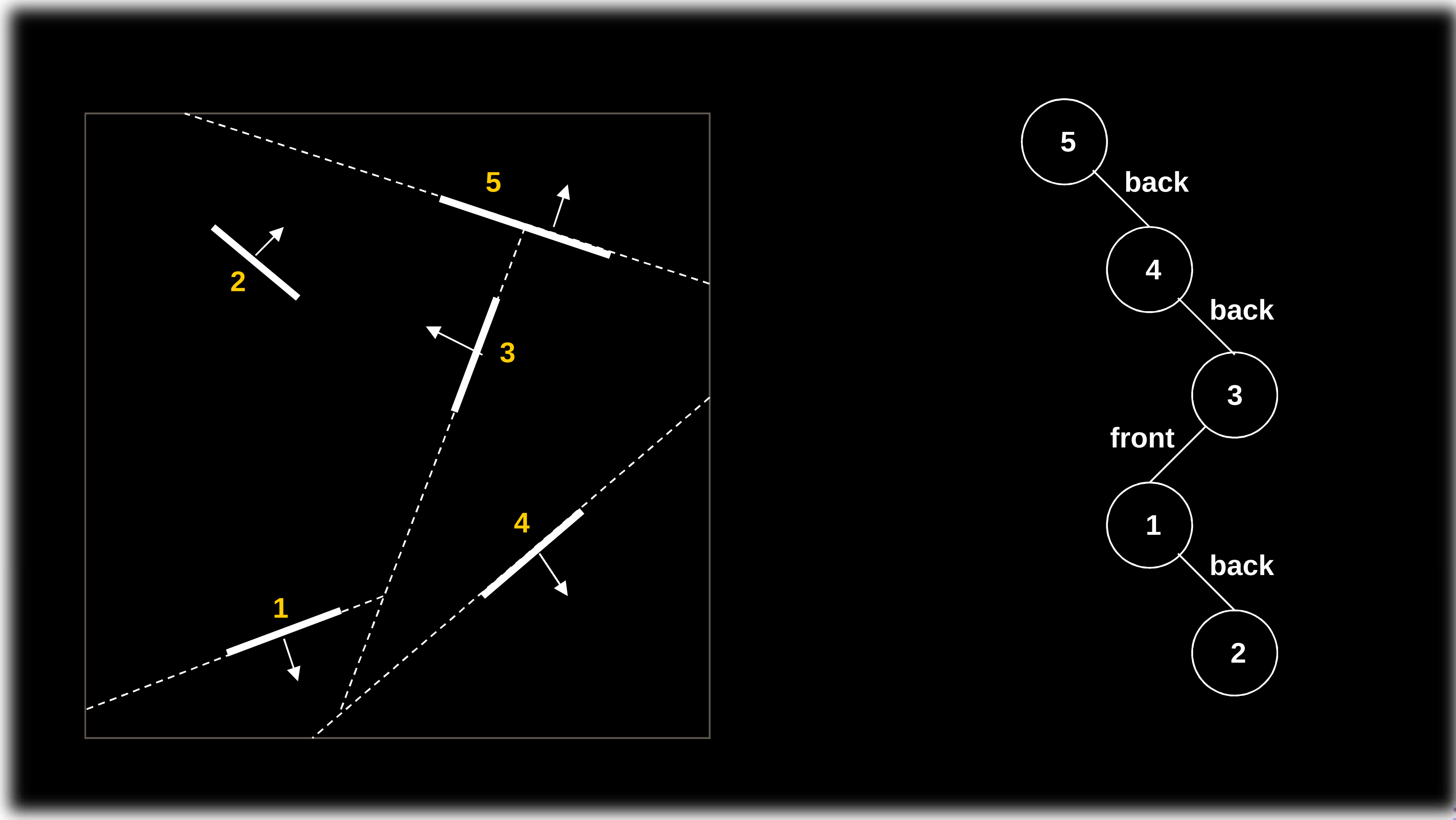
Building a BSP Tree

- Balanced Tree



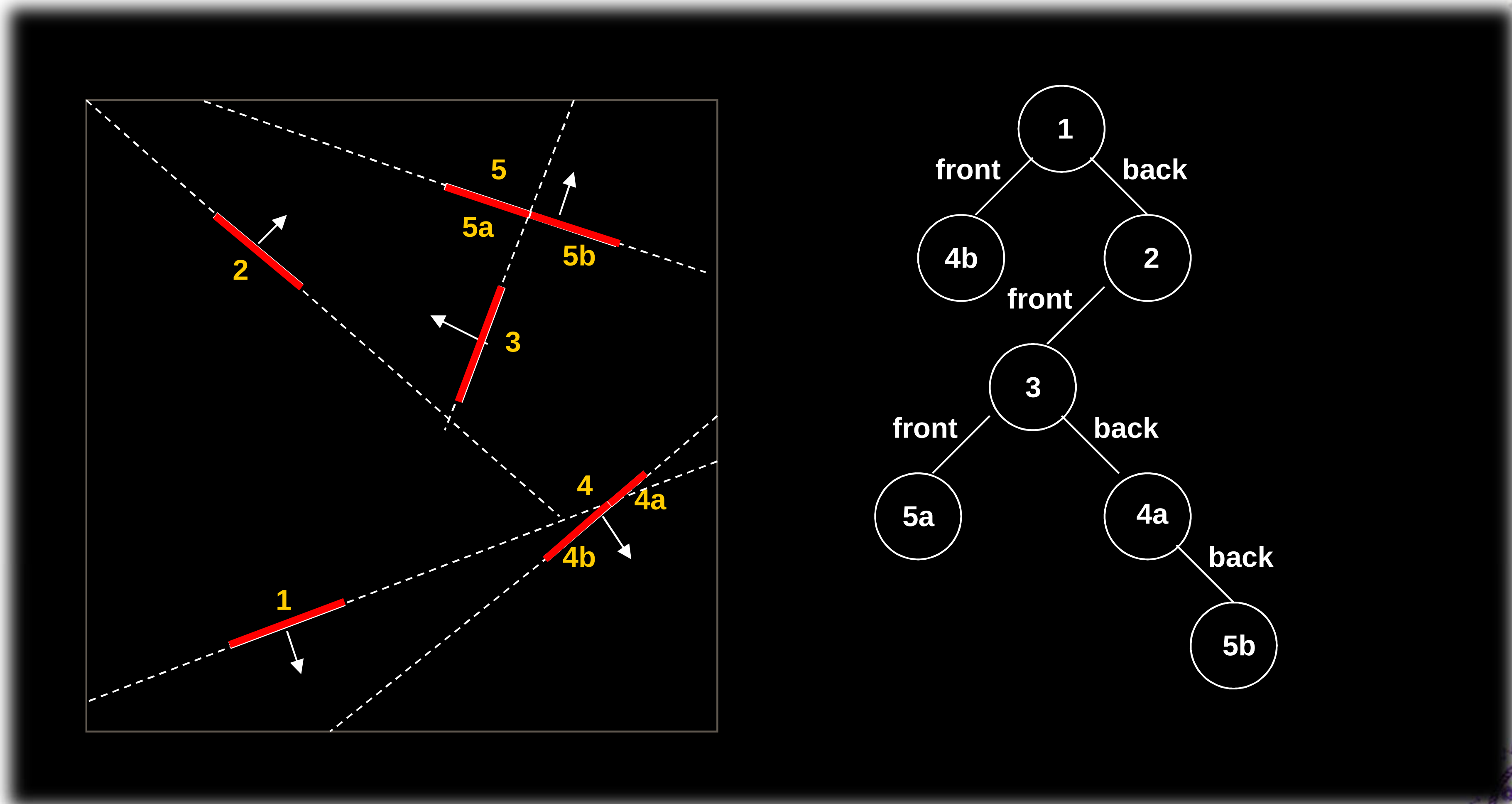
Building a BSP Tree

◆ Minimize Splitting



Building a BSP Tree

- Dynamic BSP Tree Construction



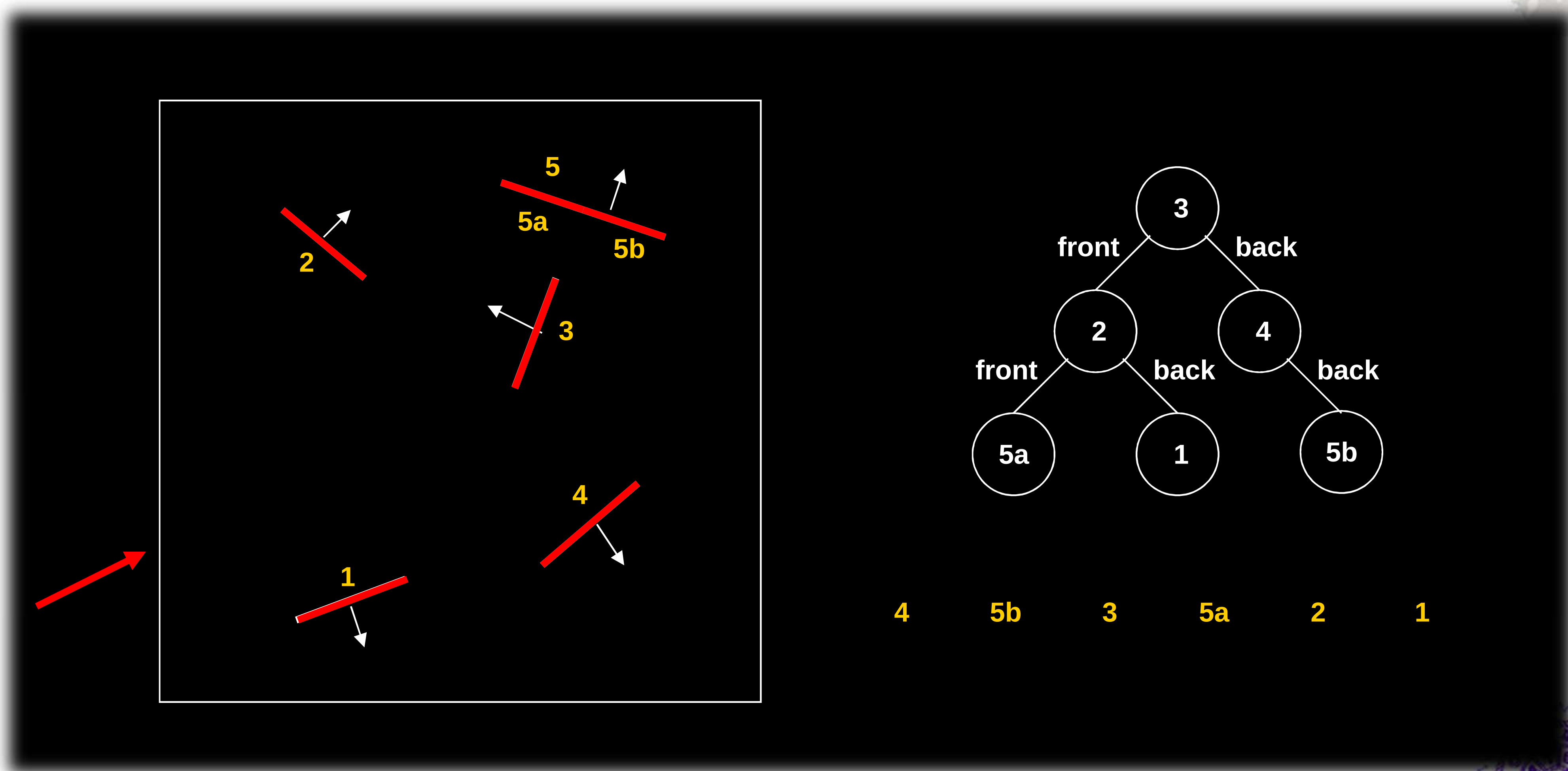
Back-to-Front Display

- **BSP Tree Traversal for Back-to-Front Display**

```
BSP_Back_to_Front(Node)
{
    if (Node == Leaf node)
        Draw(Node)
    else
        if (Viewpoint is in front of the Node)
        {
            BSP_Back_to_Front(Back(Node))
            Draw(Node)
            BSP_Back_to_Front(Front(Node))
        }
        else if (Viewpoint is in back of the Node)
        {
            BSP_Back_to_Front(Front(Node))
            Draw(Node)
            BSP_Back_to_Front(Back(Node))
        }
}
```

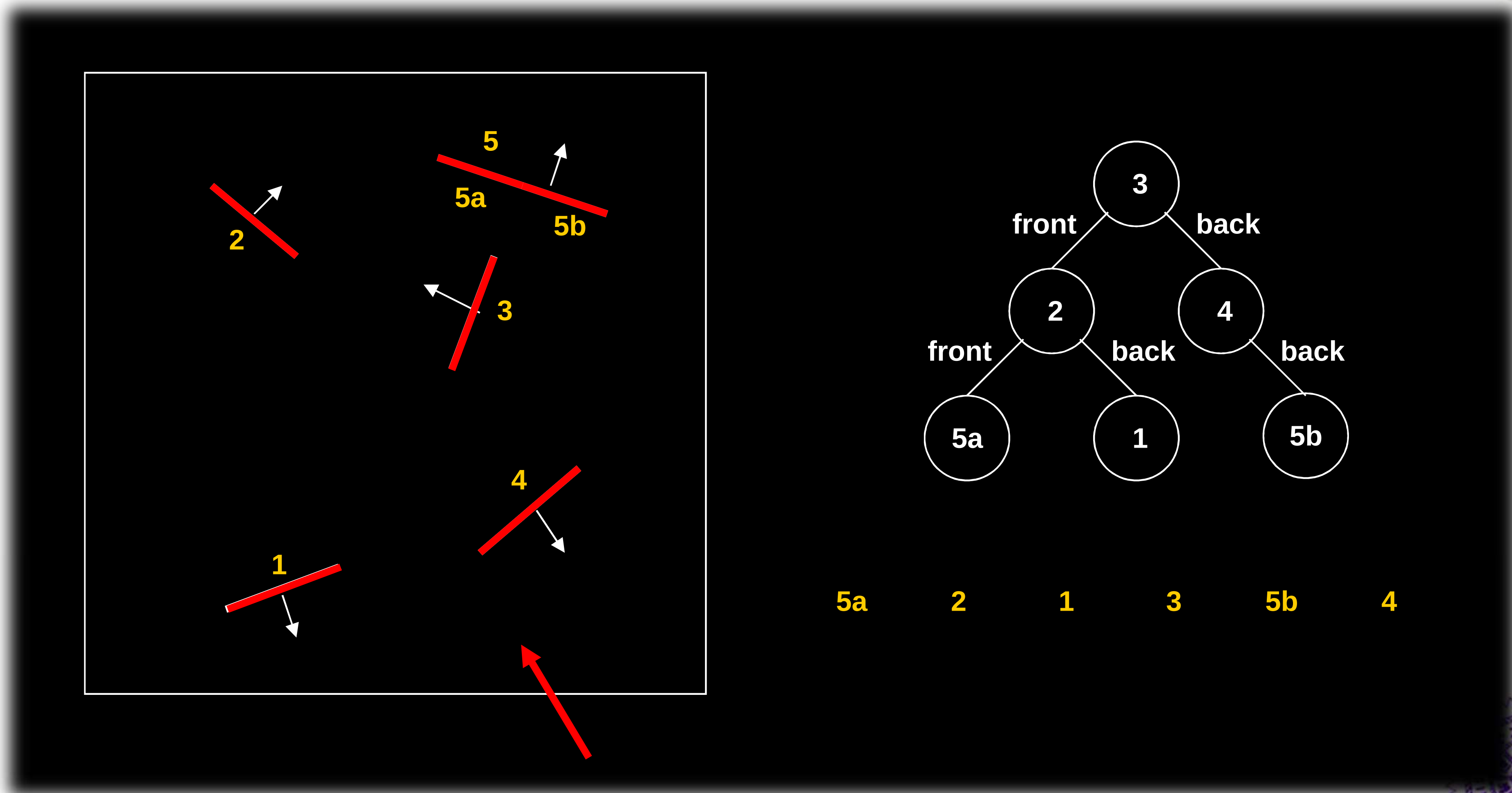
Back-to-Front Display

- Painter's Algorithm Approach



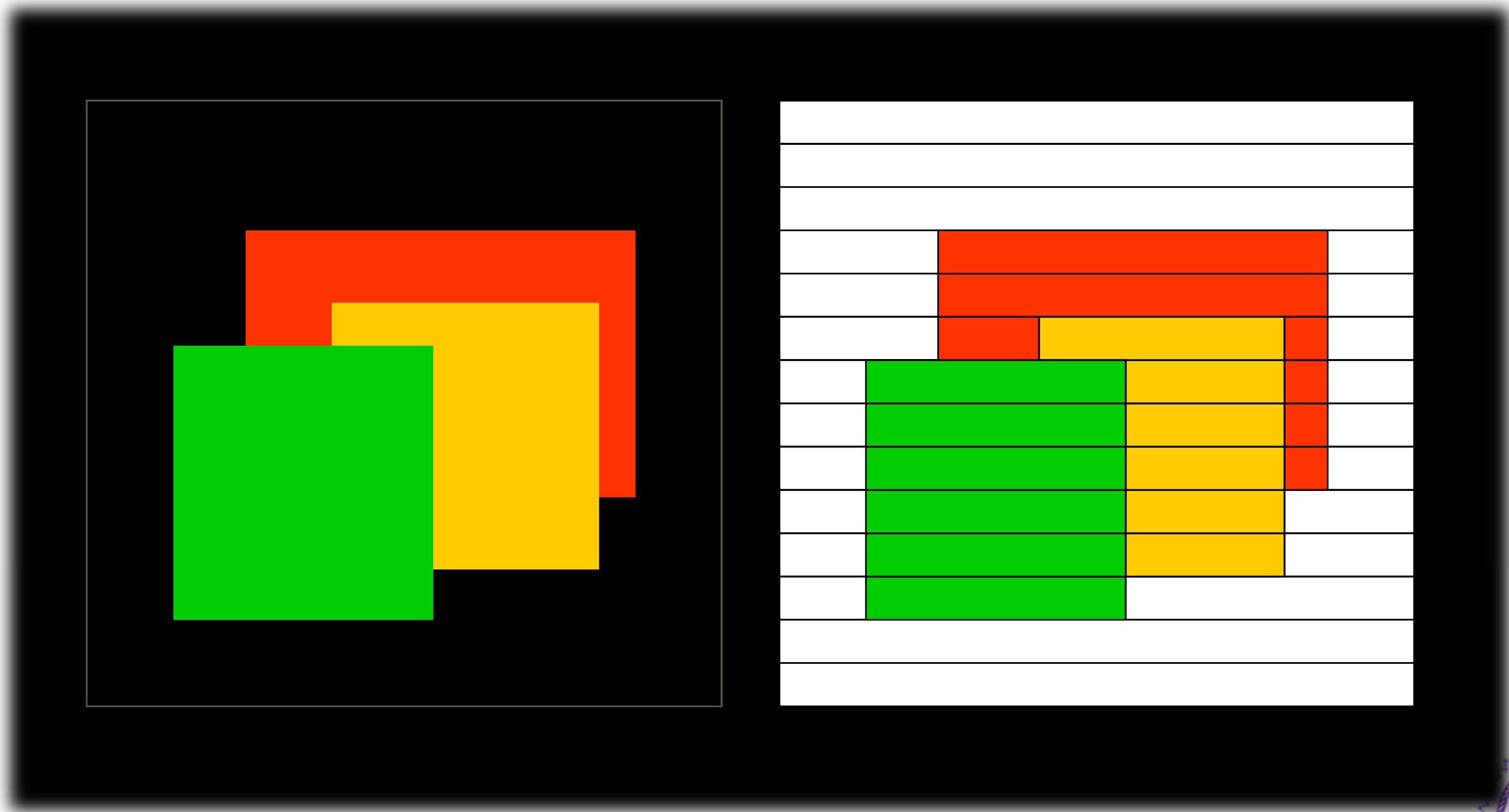
Back-to-Front Display

- **Painter's Algorithm Approach**



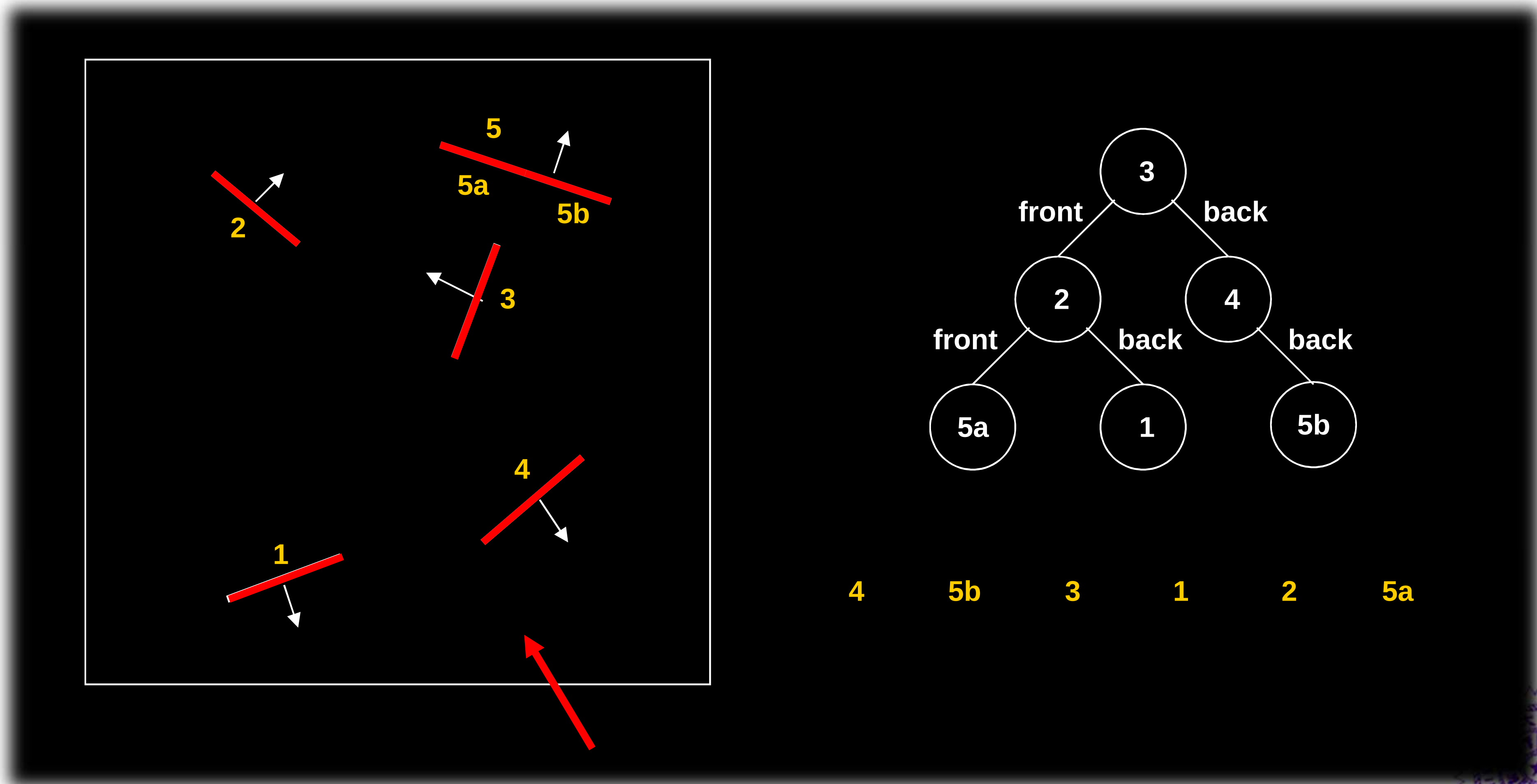
Front-to-Back Display

- Scanline Approach



Front-to-Back Display

- Scanline Approach



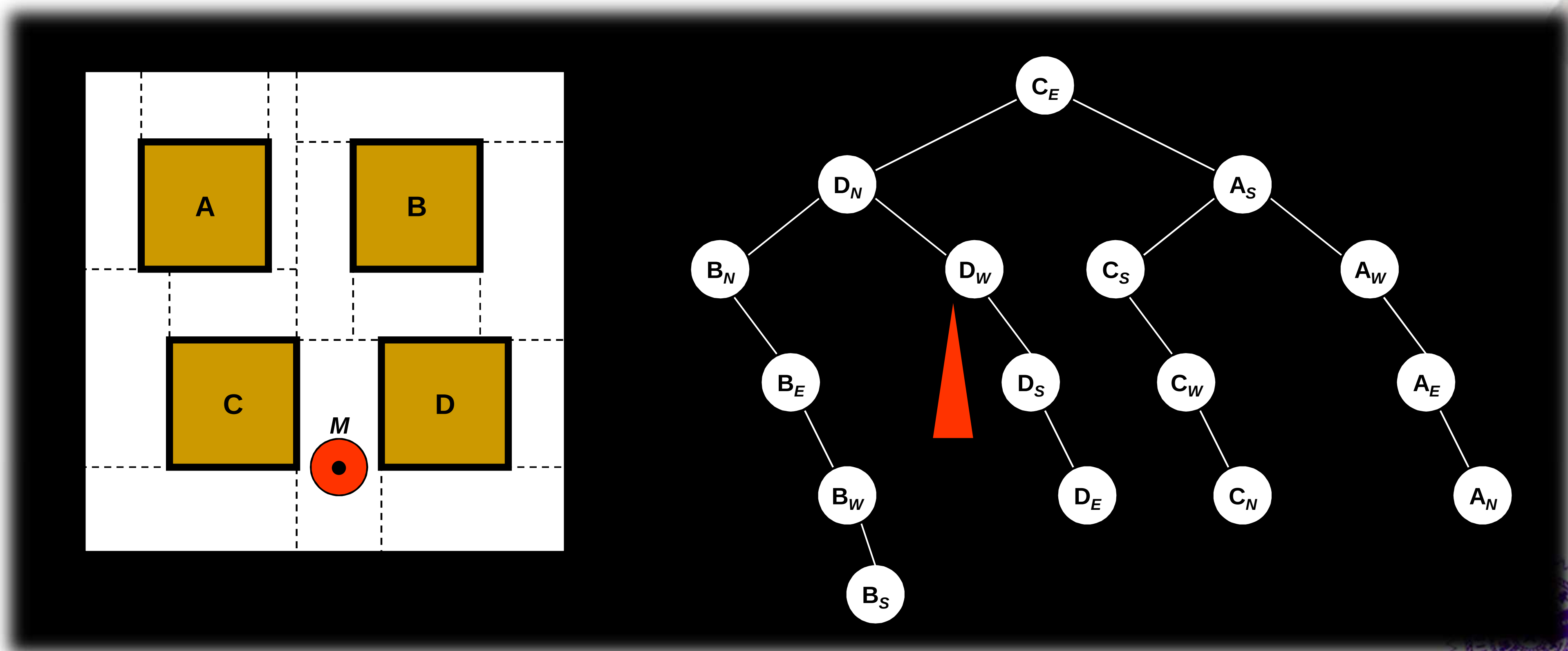
Hidden Surface Removal with a BSP Tree

- **Back-to-Front Traversal**
  - **Painter's Algorithm**
    - **Advantage: No Z buffer is required**
    - **Disadvantage: Some pixels are over-drawn**

- **Front-to-Back Traversal**
  - **Scanline Algorithm**
    - **Advantage: Only visible pixels are drawn**
    - **Disadvantage: Need to maintain a dynamic scene data structure to represent pixel masks**

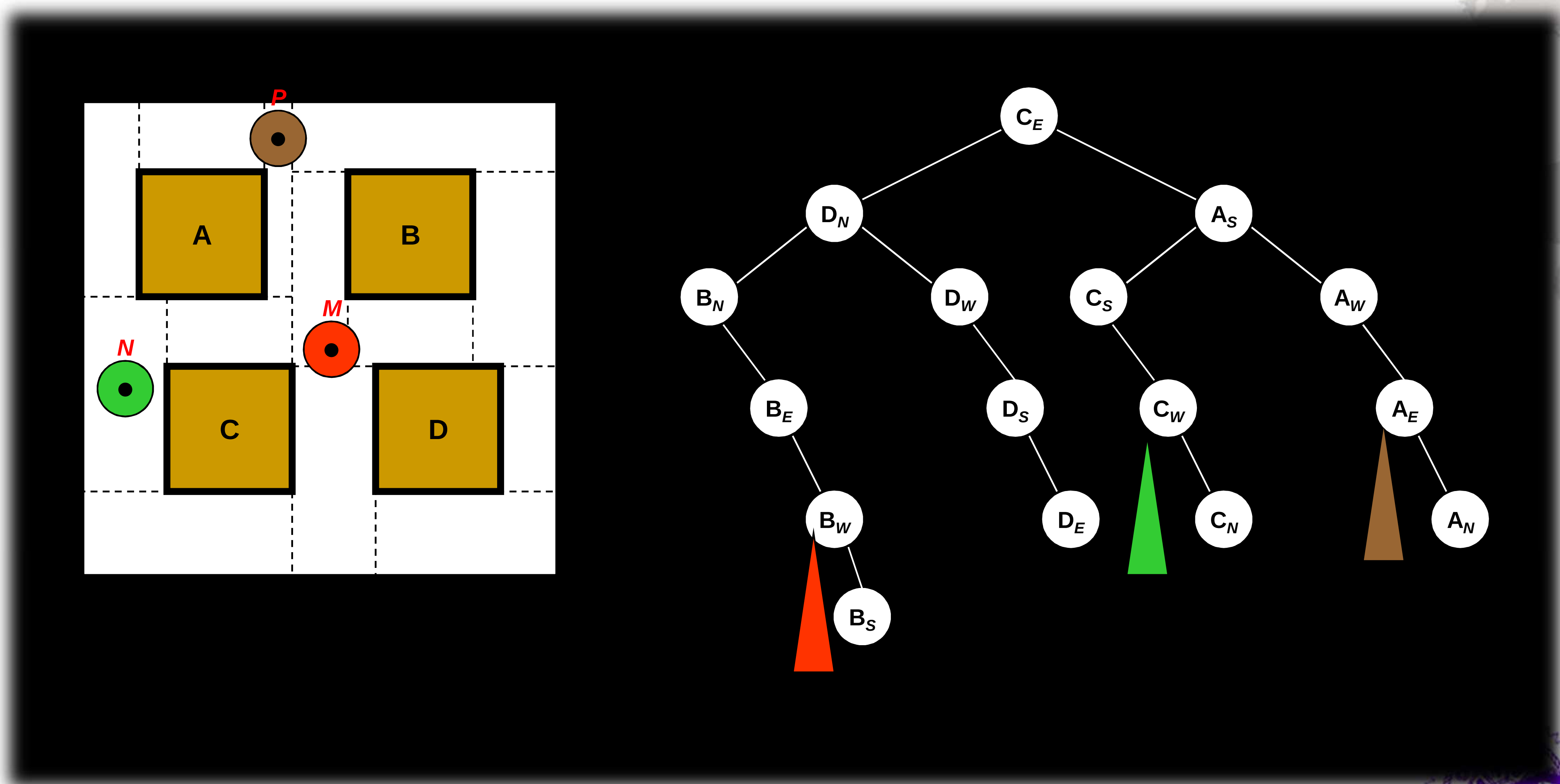
Dynamic Scene with a BSP Tree

- Start with a BSP tree containing all the static objects in the scene
- Insert the dynamic objects into the BSP tree



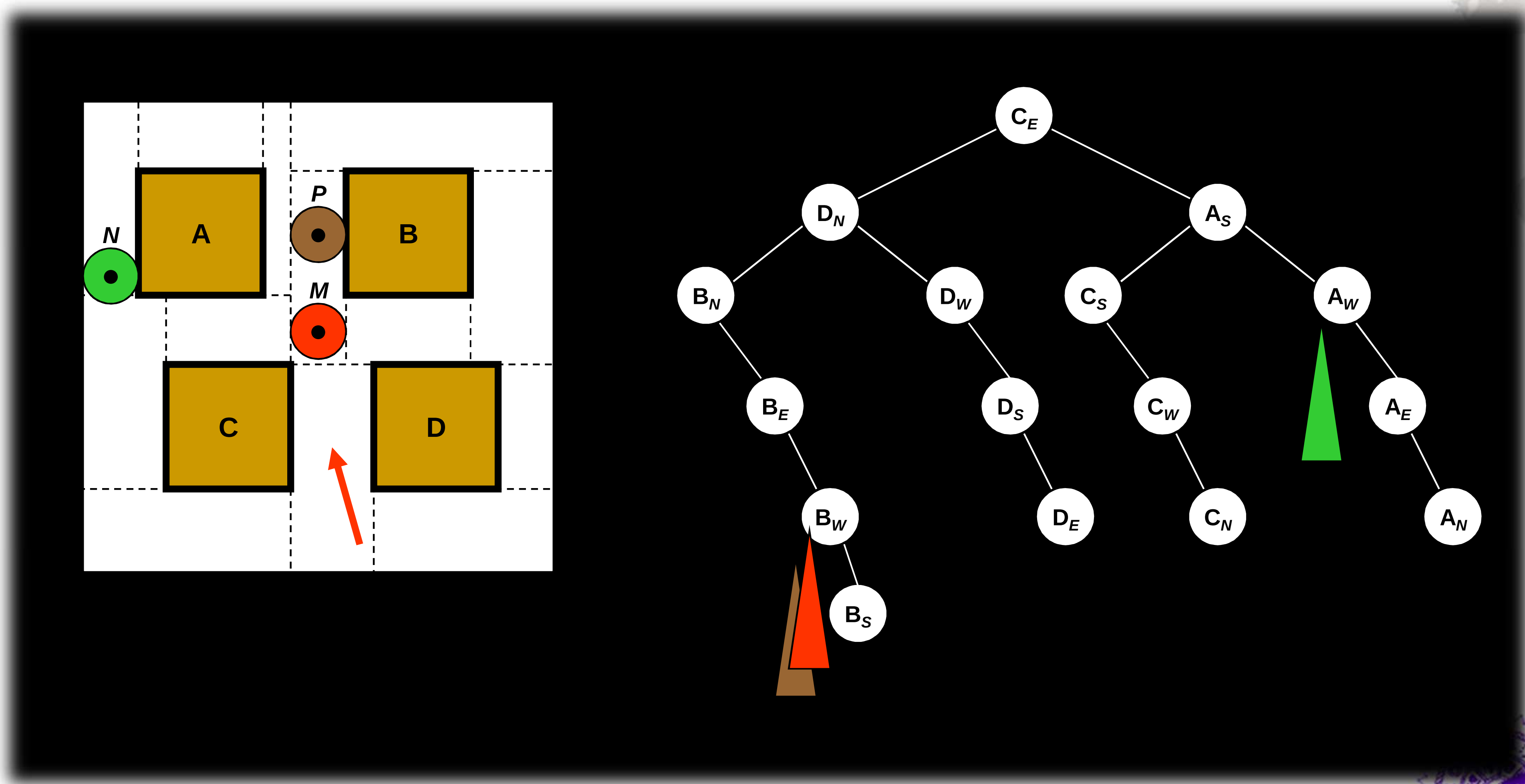
Dynamic Scene with a BSP Tree

- Doom/Quake Like Game

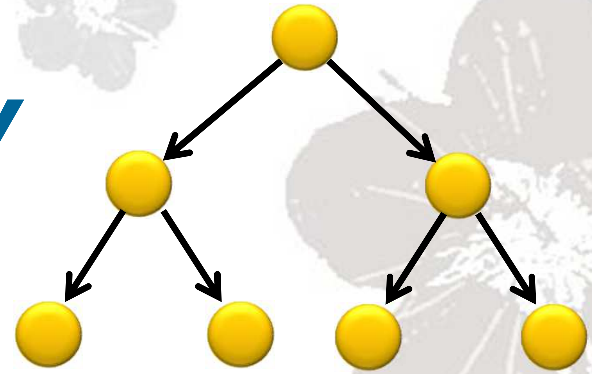


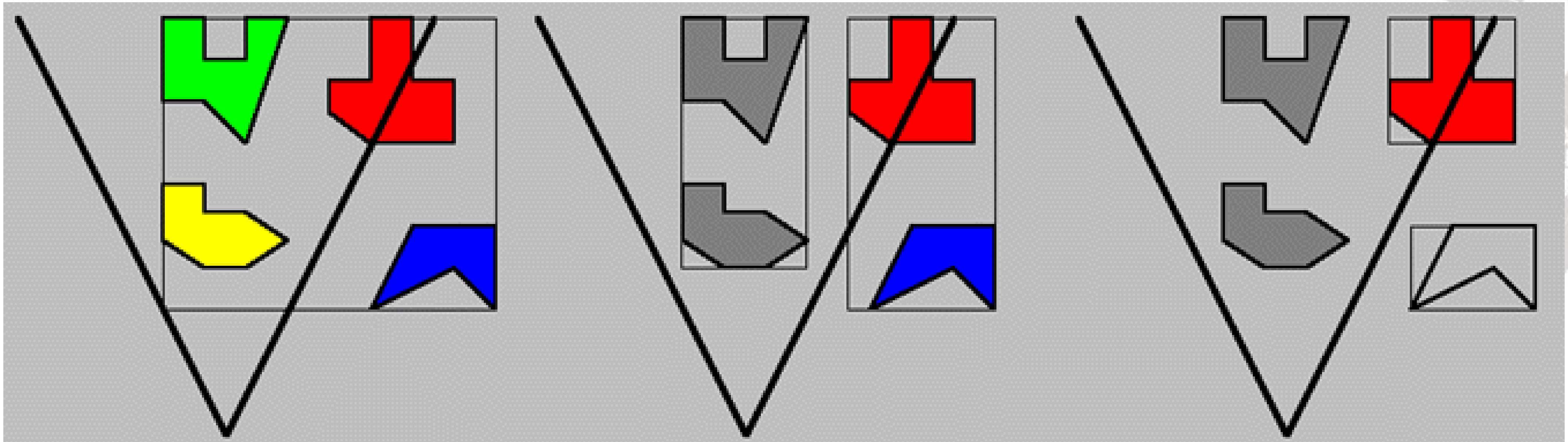
Dynamic Scene with a BSP Tree

- Doom/Quake Like Game



Bounding Volume Hierarchy

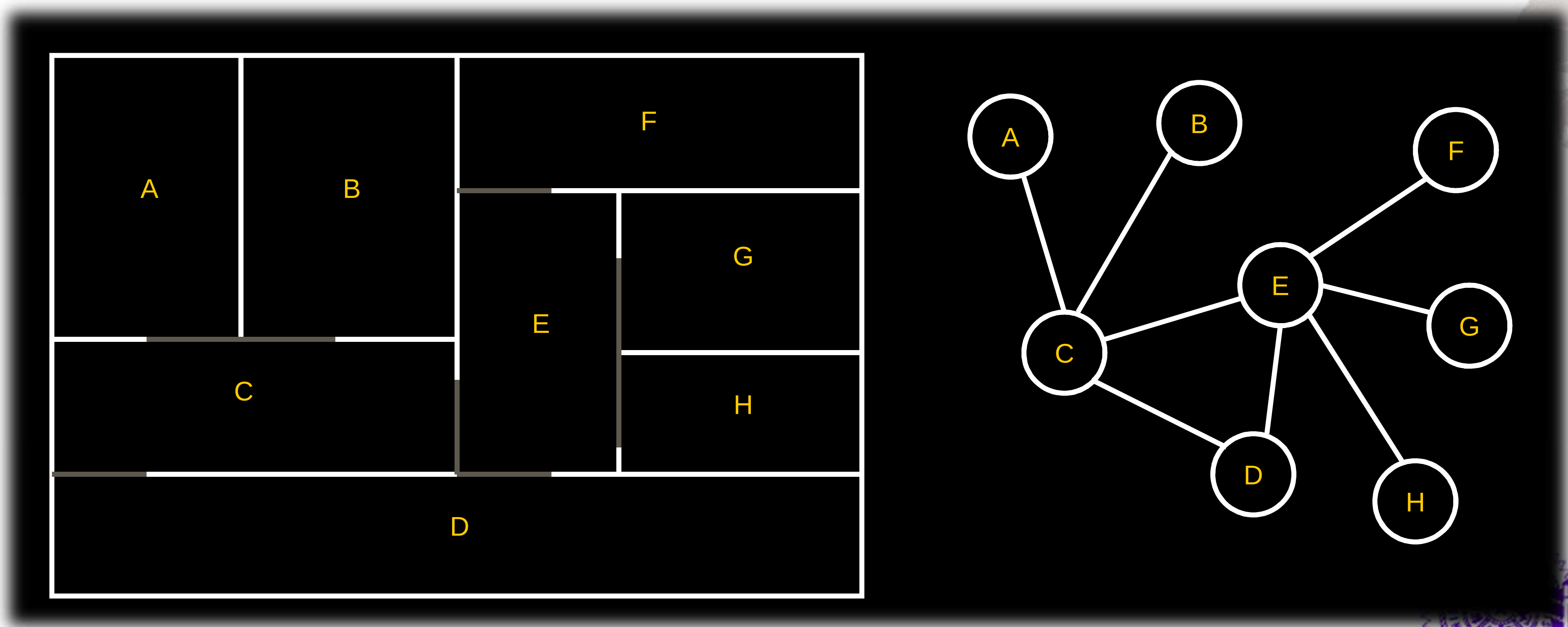




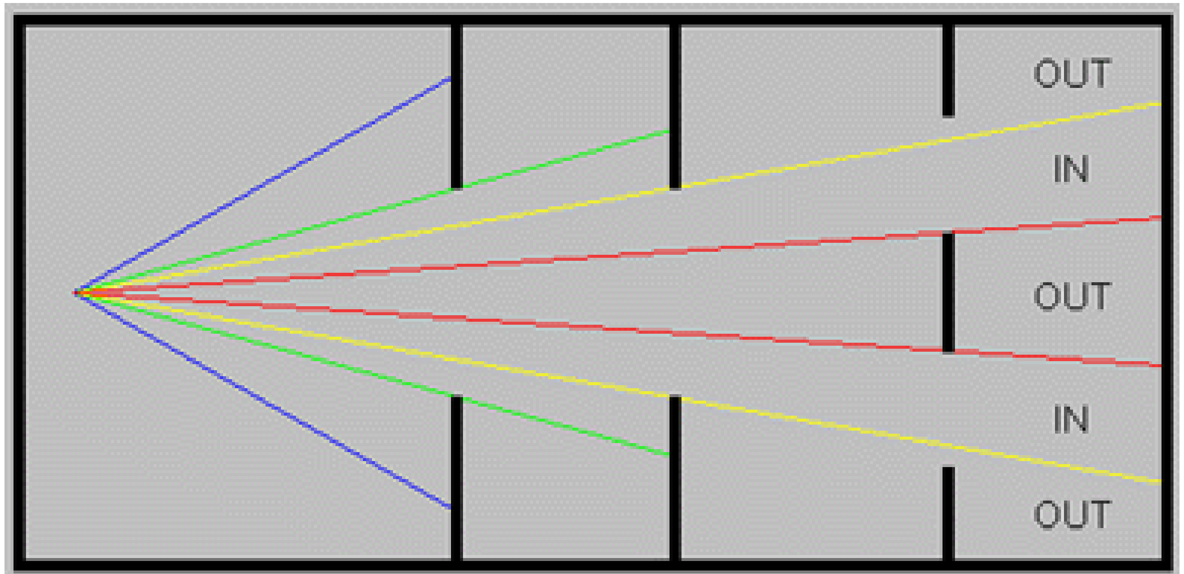
Here we show the progression of overlap tests on a simple three-level bounding-volume hierarchy. The left figure shows the top-level bounding box that encloses the entire scene; it partially overlaps so we must check its children. In the middle figure, the two child boxes are shown; the left box is completely inside so all of the objects it contains are trivially accepted (shown in gray). The right box must be traversed; the process repeats recursively. In the right figure, the lower bounding box is completely outside so its object are trivially rejected (hollowed); the upper box is partially overlapping and does not contain any child boxes, so the individual polygons must be tested for overlap (this step is sometimes omitted and the entire object is just sent to the low-level graphics pipeline).

Portal Culling

- An adjacency graph is built to represent the connections of cells. The connections are established through portals

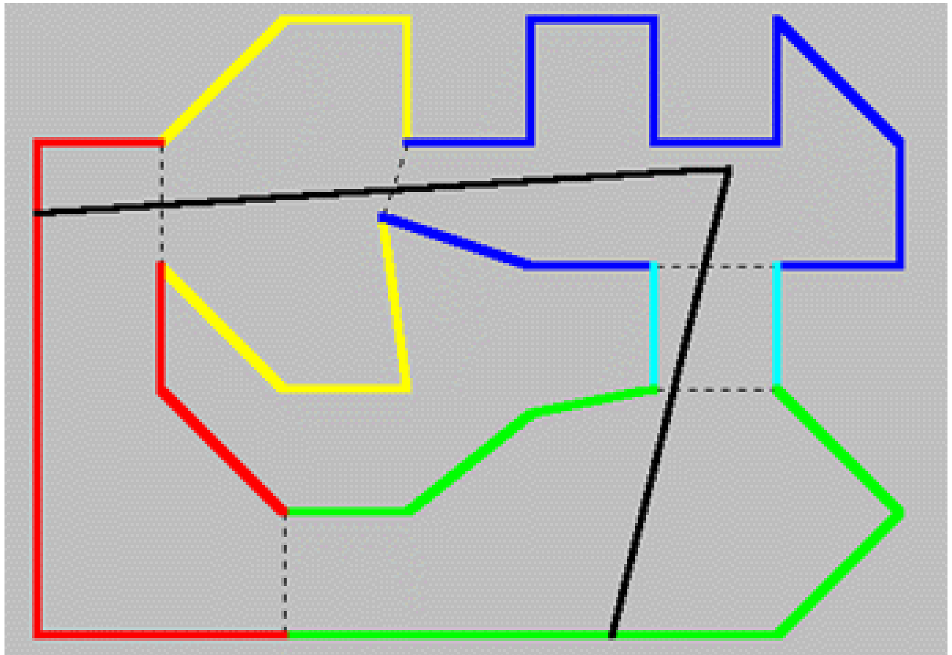


Portal Culling

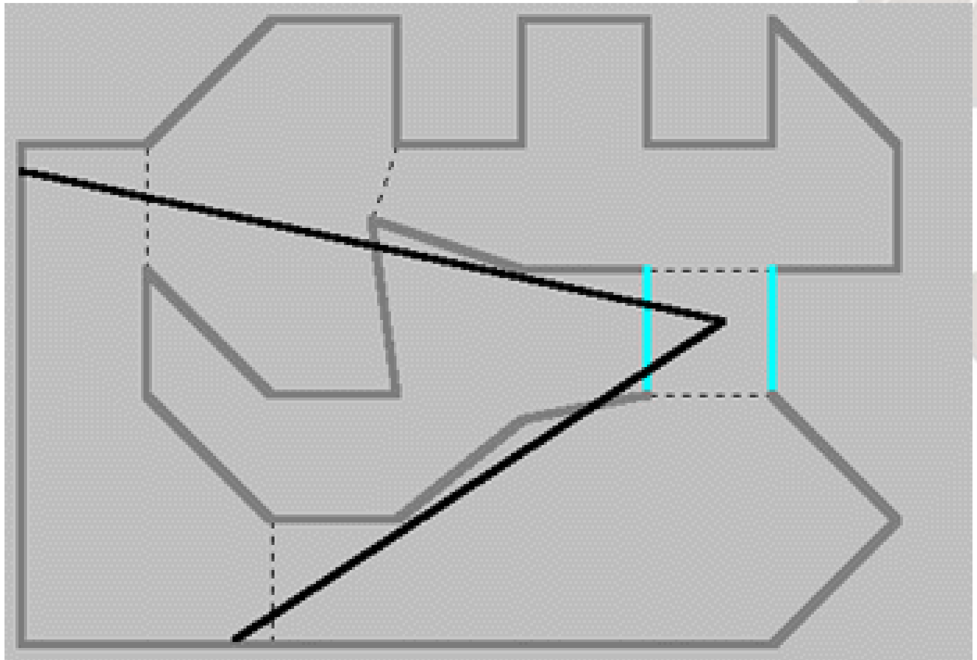


This figures illustrates the recursive nature of cells and portals. Each visible cell has at least one frustum entering it (trivial case is the first cell that contains the viewer). Objects belonging to each cell can be culled against the entering frusta. The viewer's field-of-view is indicated in blue; green lines show the frustum formed through the first portal; yellow lines indicate the second portal frustum; this final frustum is split by the last portal wall into two smaller frusta indicated with the red and yellow lines.

Portal Culling

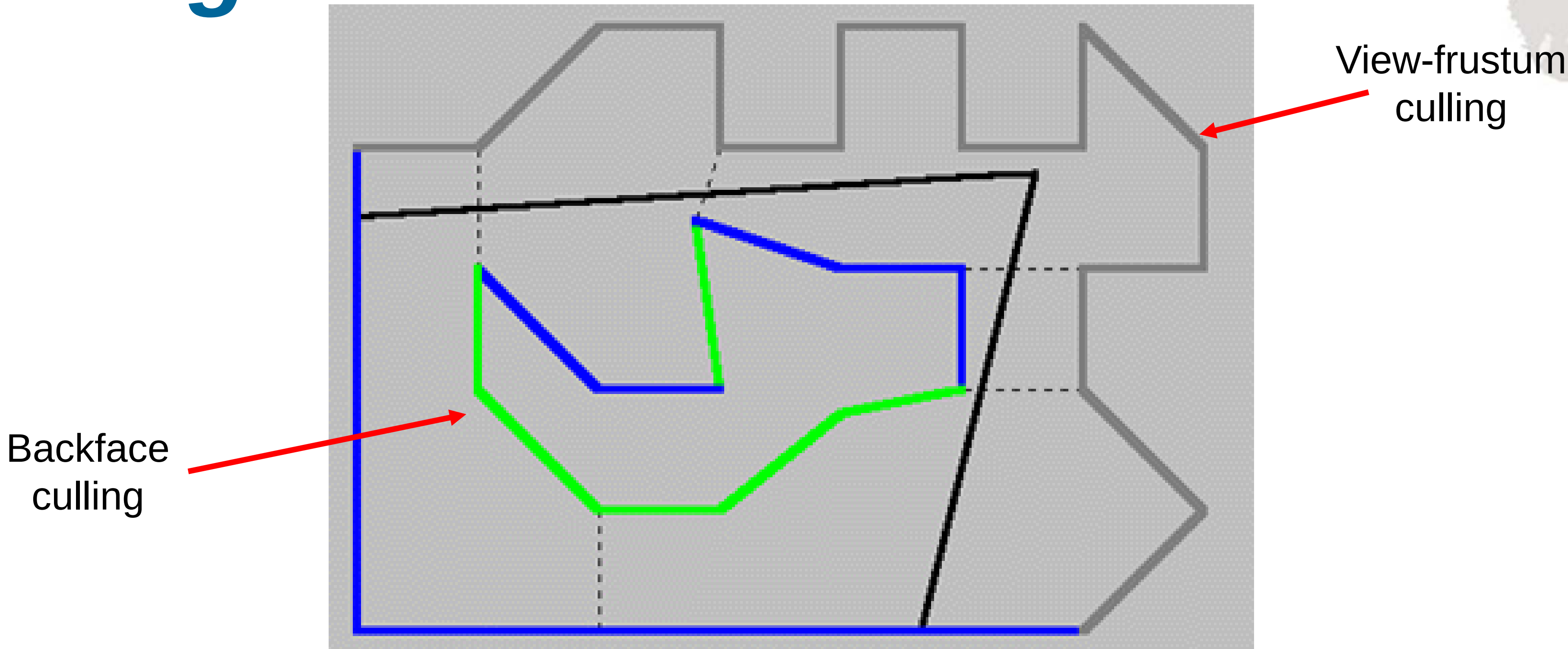


Here the world is divided into sets of polygons grouped by rooms or cells (different colors) that are separated by doorways or portals (dashed lines). Only cells visible through sequences of portals are drawn. Here there is no advantage since all cells can be seen through the portals; however, if this was used with view-frustum and backface culling we could still reduce the load.



When the viewpoint moves into a cell where long sequences of portals are no longer visible, large portions of the world are culled at a significant fraction of the cost of using other techniques (only two portal overlap tests were required). Clearly, hybrid techniques can cull even more.

View-Frustum Culling with Backface Culling



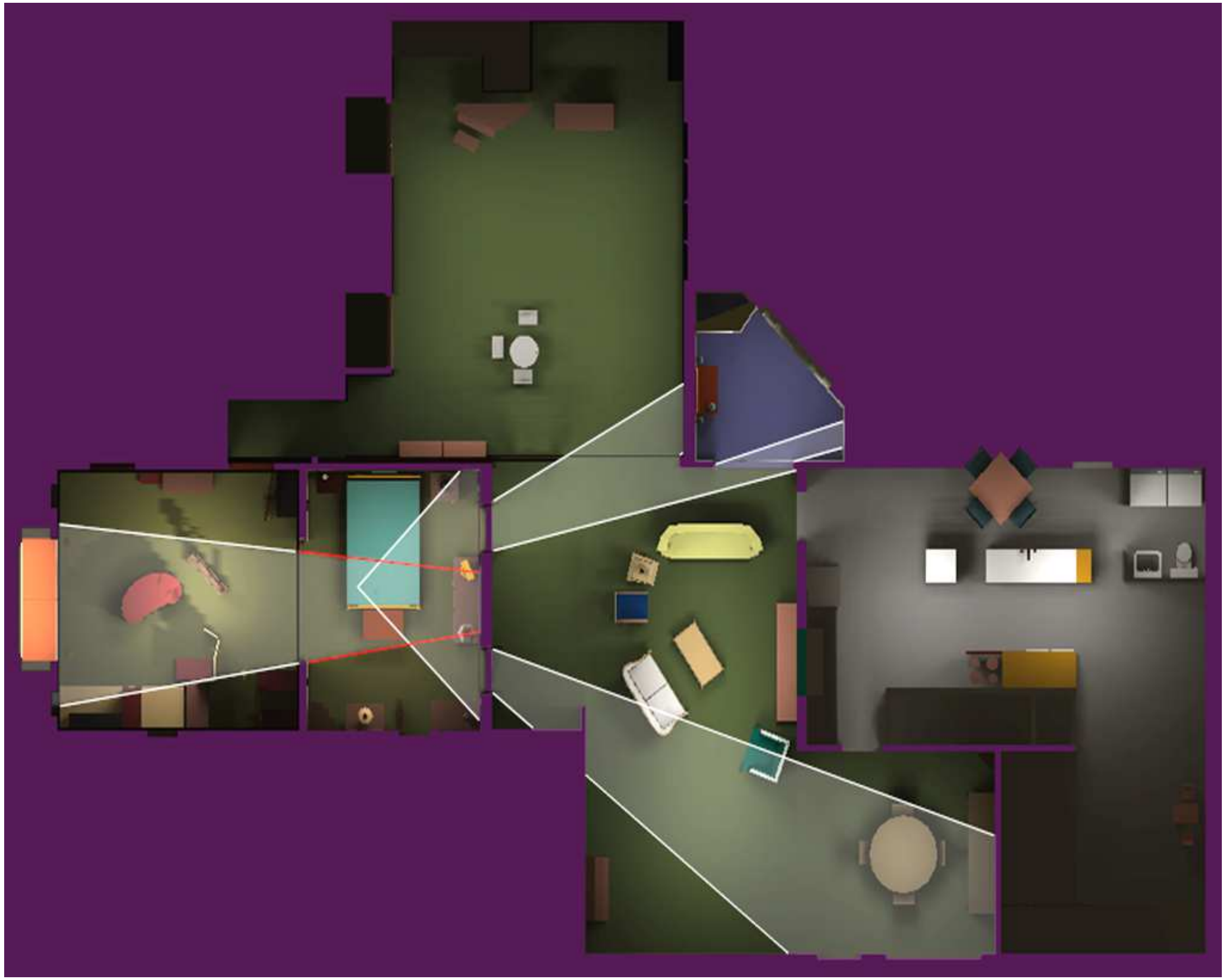
With backface culling in addition to view-frustum culling, polygons facing away from the viewer are also culled (green). Since the blue polygons are opaque, the resulting image is unaffected.

Portal Culling Example

- Luebke and Georges, I3D 95

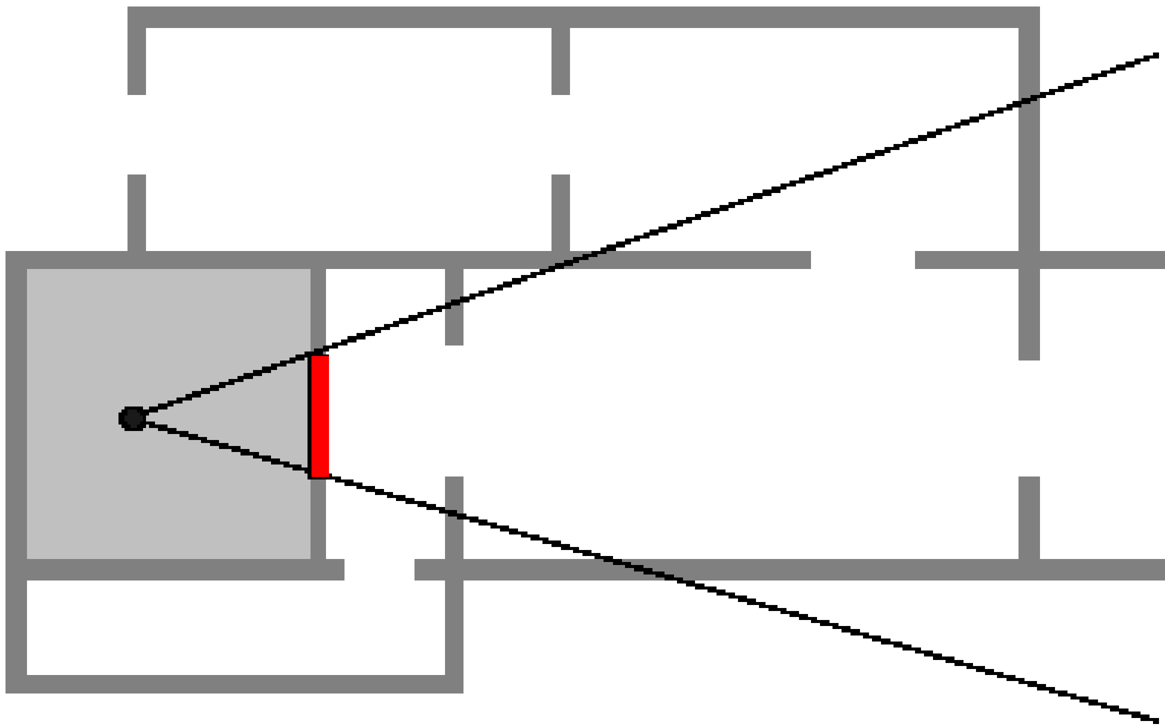
Portal Culling Example

- Top view



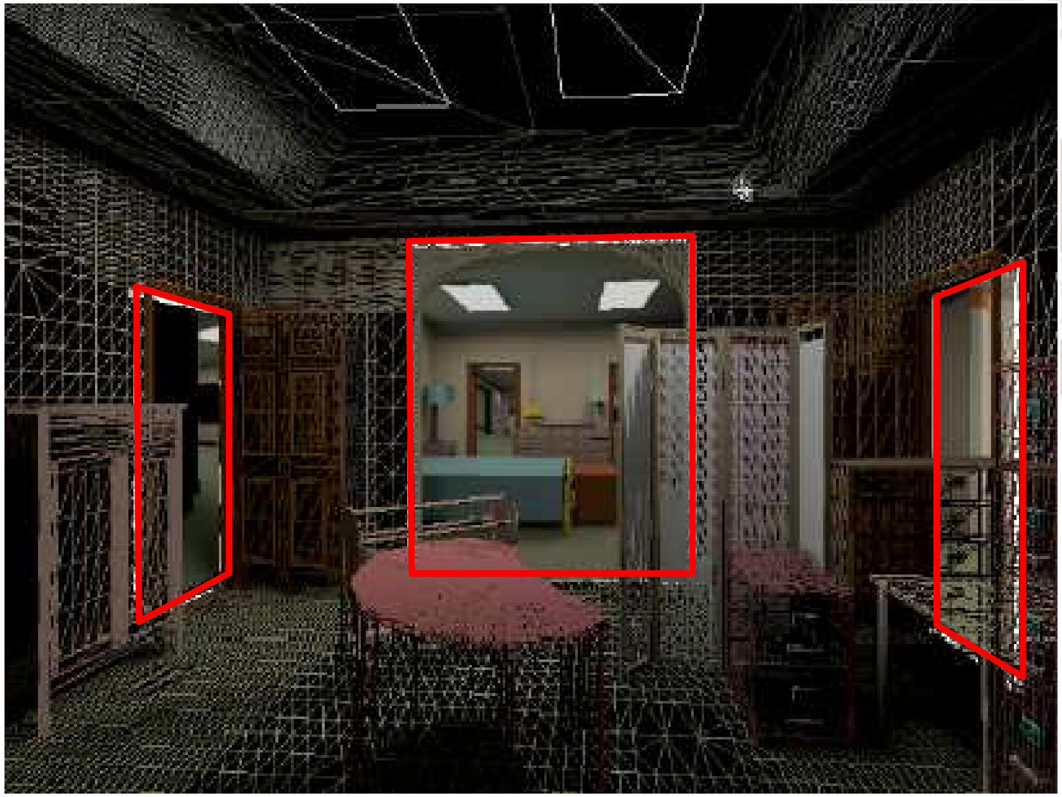
Portal Texture

- **Pre-compute portal textures to reduce rendering time during walkthrough**



Aliaga and Lastra,
IEEE Visualization '97

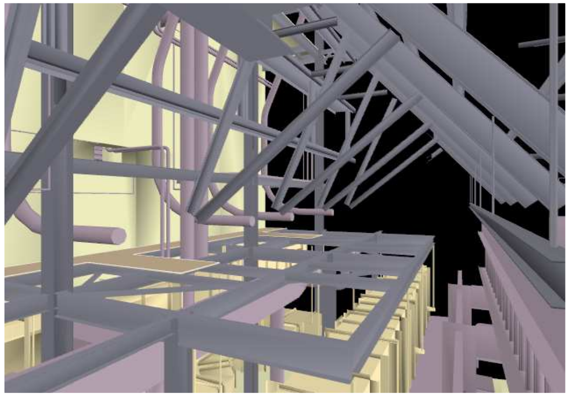
Portal Texture



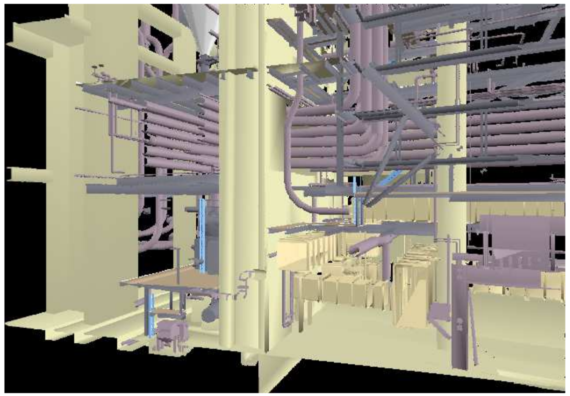
Aliaga and Lastra,
IEEE Visualization '97

Portal Texture Example

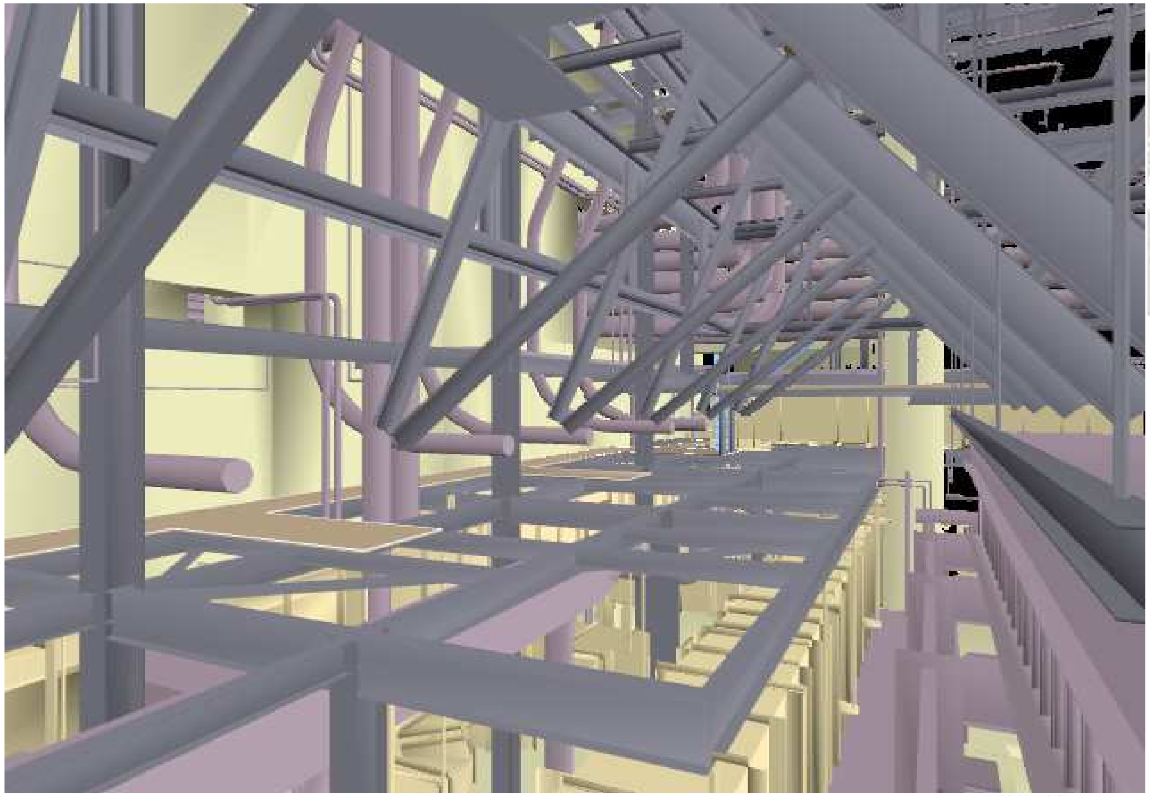
Geometry



+ =



Image



Final Scene

Q&A