Computer Graphics

6. Hidden Surface Removal, Culling and Space Partitioning

I-Chen Lin
National Yang Ming Chiao Tung Univ., Taiwan

Textbook: E.Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson
Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson

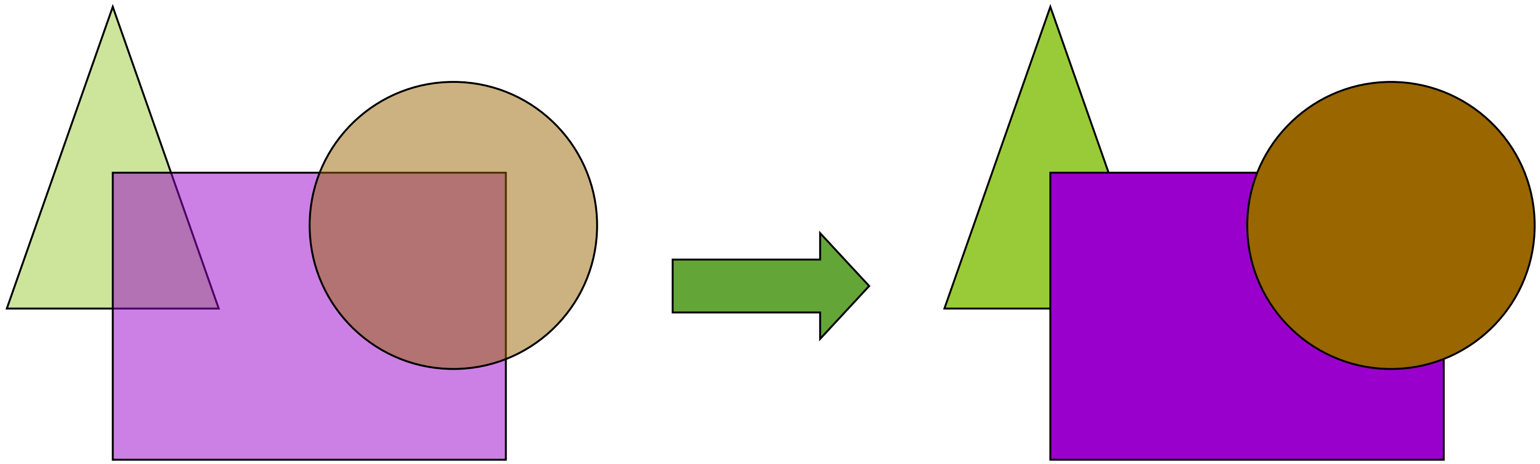
Intended Learning Outcomes

- On completion of this chapter, a student will be able to:
  - **Identify** hidden surfaces and **state** why they are concerned graphics rendering.
  - **Describe** the primary hidden surface removal methods.
  - **Compare** the object-space and image space approaches for hidden surface removal.
  - **Discuss** how to use space partitioning to assist hidden surface removal and polygon culling.

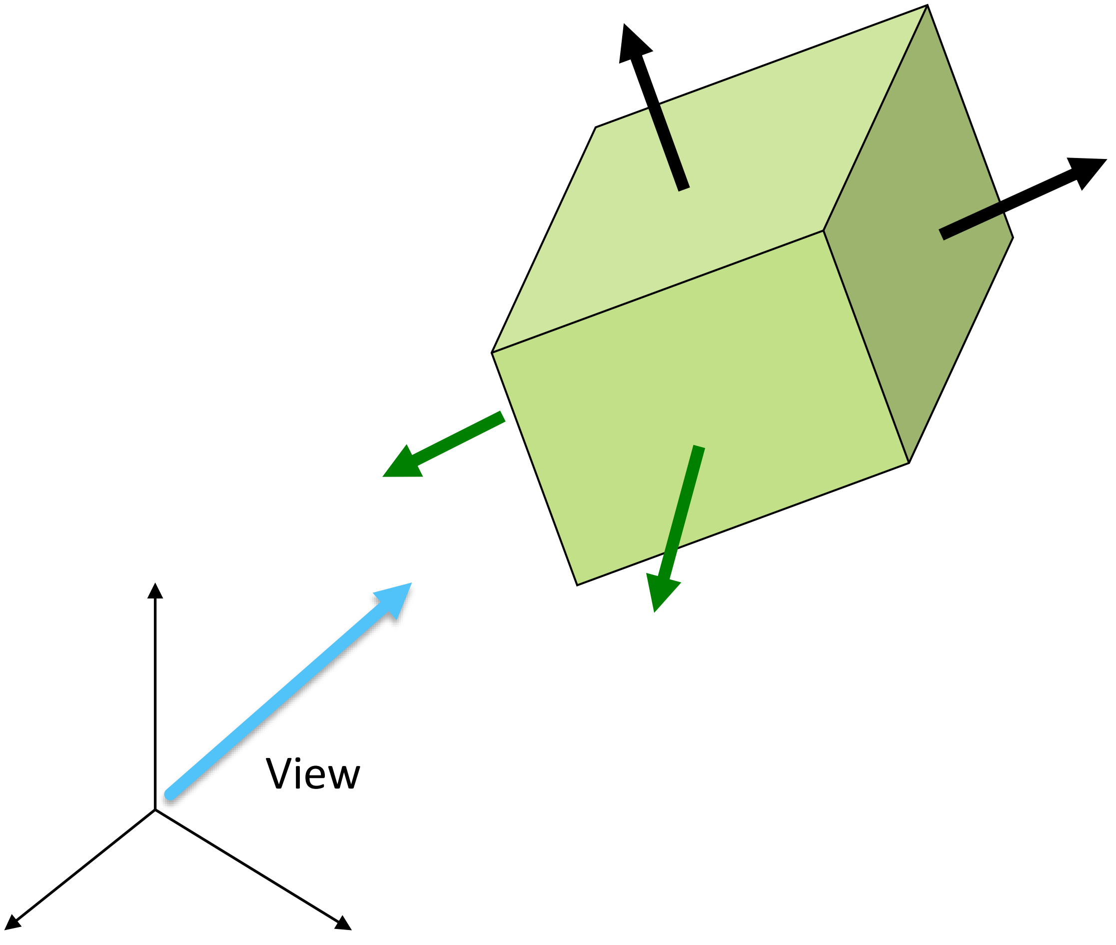
Objectives

- For a 3D wireframe viewer, we can apply viewing transformation and draw the line segments between projected point pairs.

- To fill projected polygons, we have to remove "hidden surfaces".



Back-face Removal (Culling)

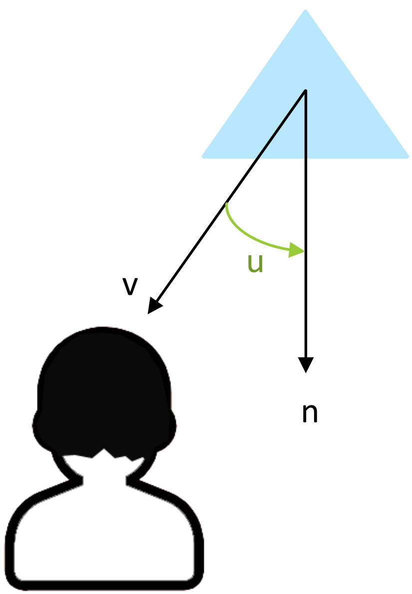


Back-face Removal (Culling)

- A face is visible iff $90 \geq \theta \geq -90$
  equivalently $\cos\theta \geq 0$ or $\boldsymbol{v} \bullet \boldsymbol{n} \geq 0$

- When $v = (0\ 0\ 1\ 0)^T$, $n = (a, b, c, 0)^T$,
  we only need to test the sign of c.

- We can enable Back-face culling in OpenGL, but it may not work correctly if we have nonconvex objects



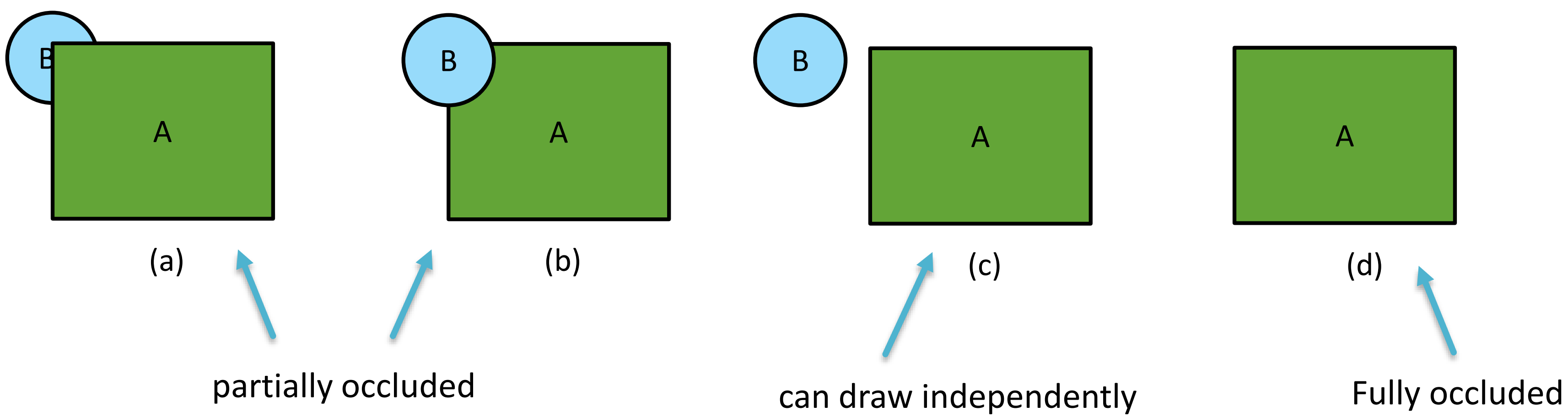
Take a look at “2D” cases

- How to hide regions behind the foreground characters?

Hidden Surface Removal

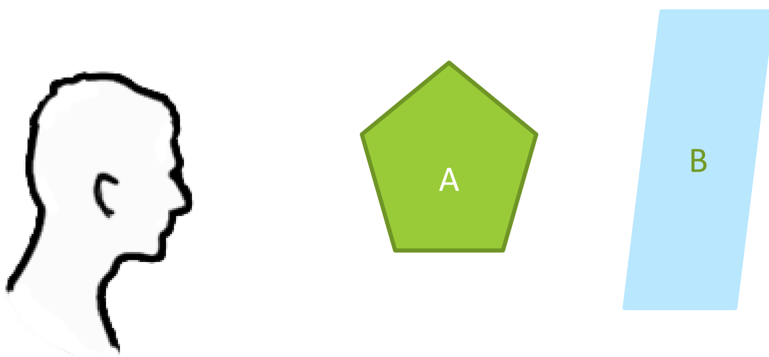
- Object-space approach: use pairwise testing between polygons (objects)

- Worst case complexity $O(n^2)$ for n polygons

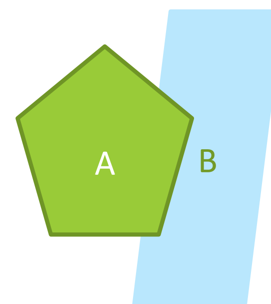


Painter's Algorithm

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



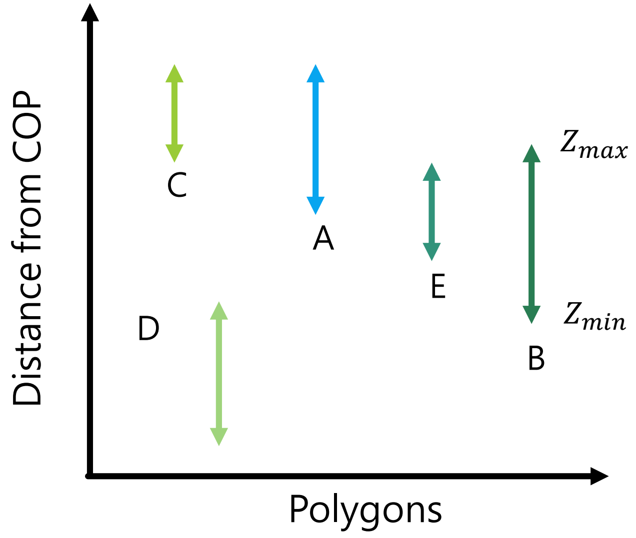
B behind A as seen by the viewer



Fill B then A

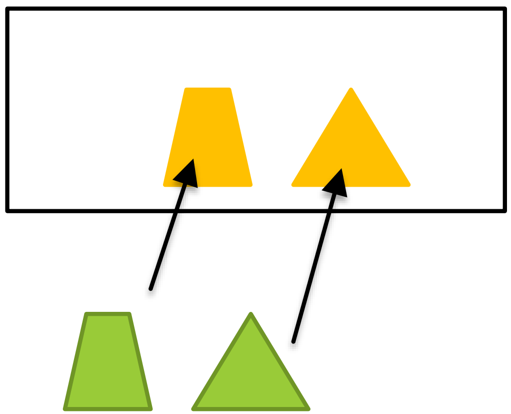
Depth Sort

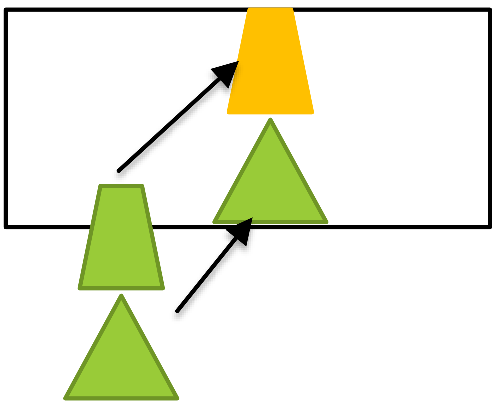
- Requires ordering polygons first
  - O(n log n) calculation for ordering
  - Not every polygon is either in front or behind all other polygons



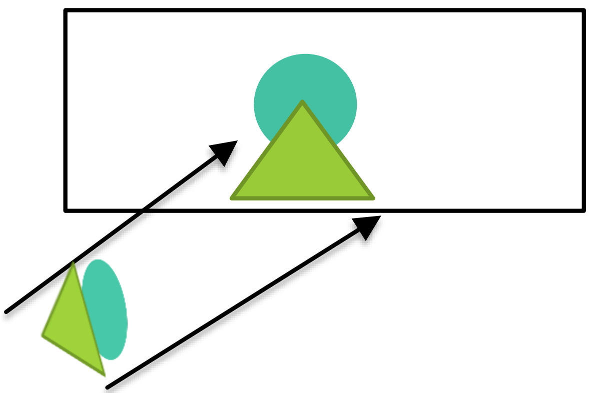
Easy Cases

- A polygon lies behind all other polygons
  - Can render
- Polygons overlap in z but not in either x or y
  - Can render independently

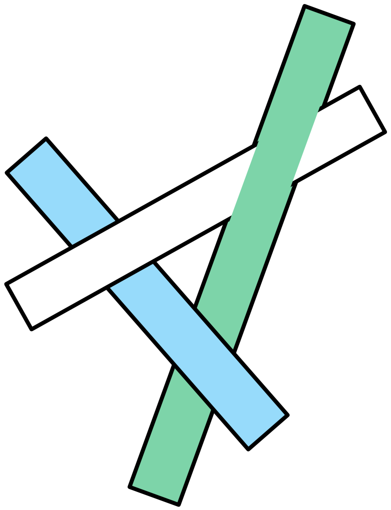




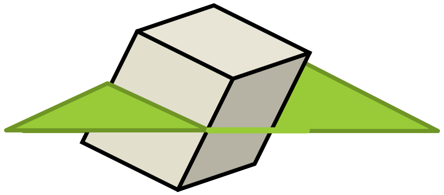
Difficult Cases



Overlap in all x, y, z directions but one is fully on one side of the other



cyclic overlap

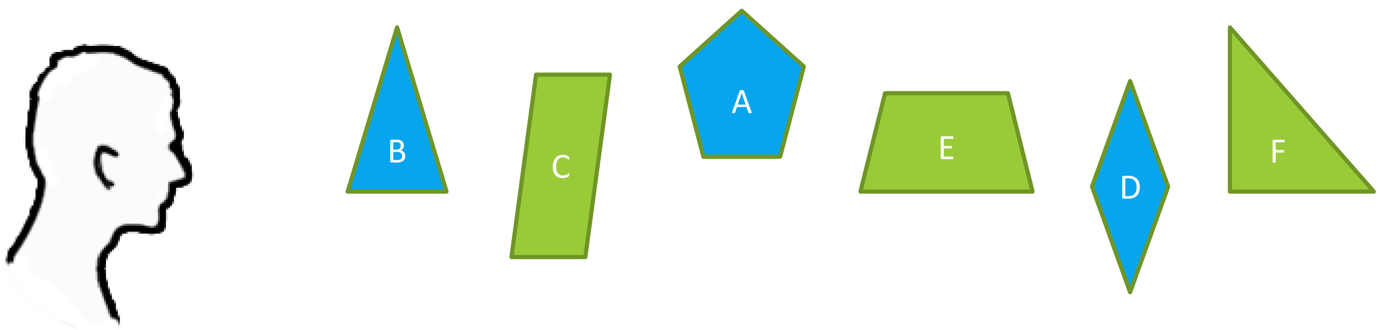


penetration

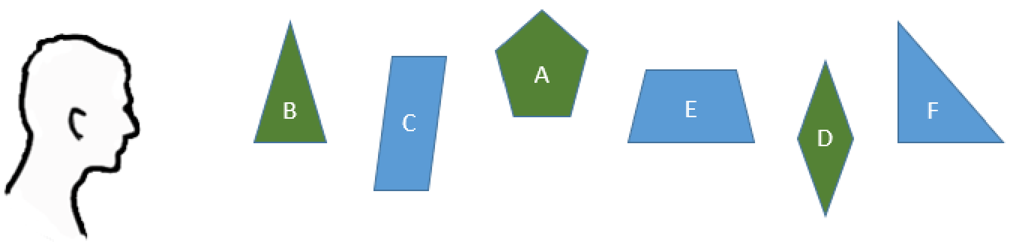
Can we disambiguate the overlapping situations?

- Let the back-to-front painter's algorithm work to more general cases.

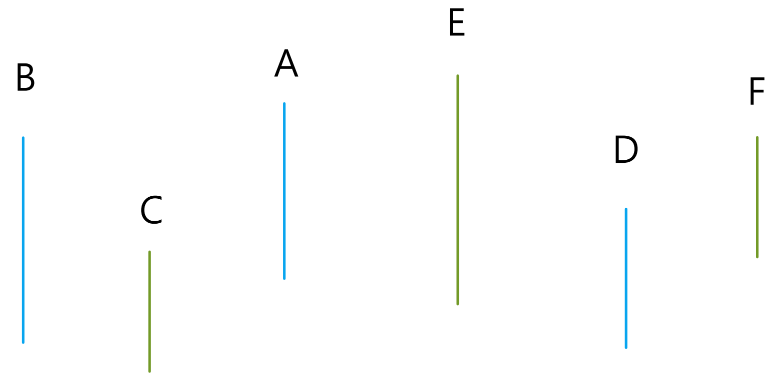
- Partition space with Binary Spatial Partition (BSP) Tree



A Simple Example



consider 6 parallel polygons



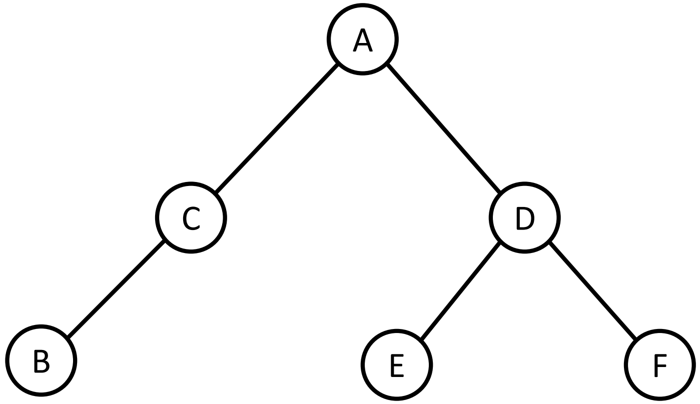
top view

The plane of A separates B and C from D, E and F

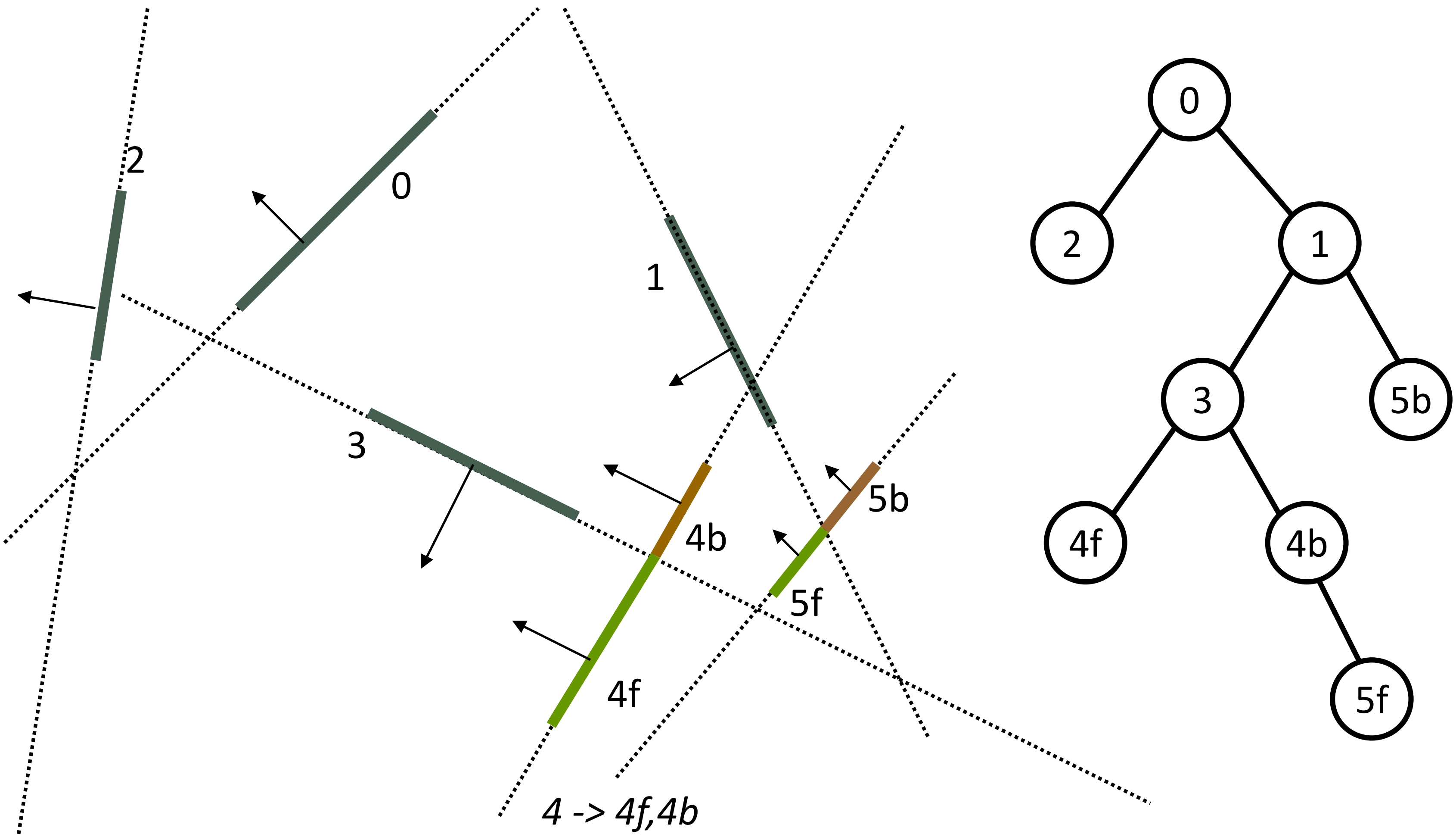
Binary Space Partitioning Tree

- Can continue recursively
  - Plane of C separates B from A
  - Plane of D separates E and F

- Can put this information in a BSP tree
  - Use for visibility and occlusion testing



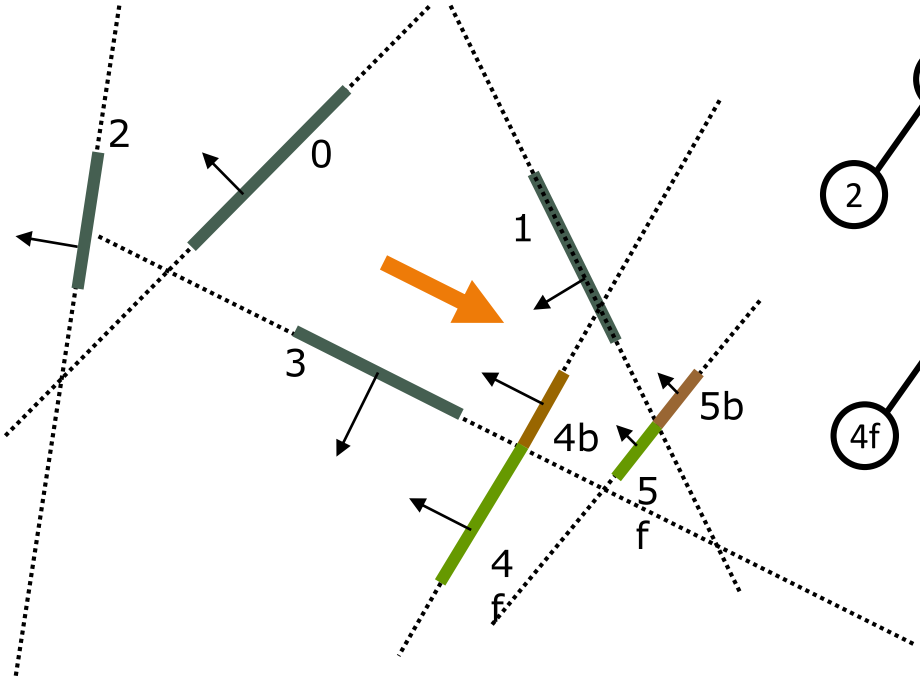
Creating a BSP tree

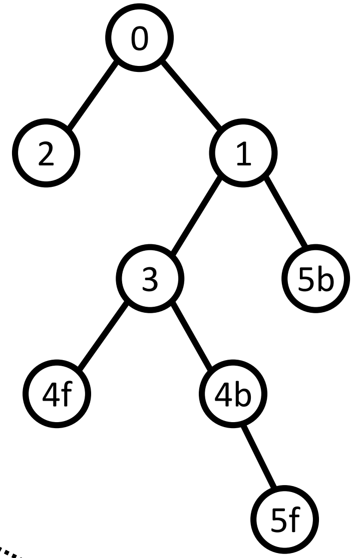


Back-to-Front Render

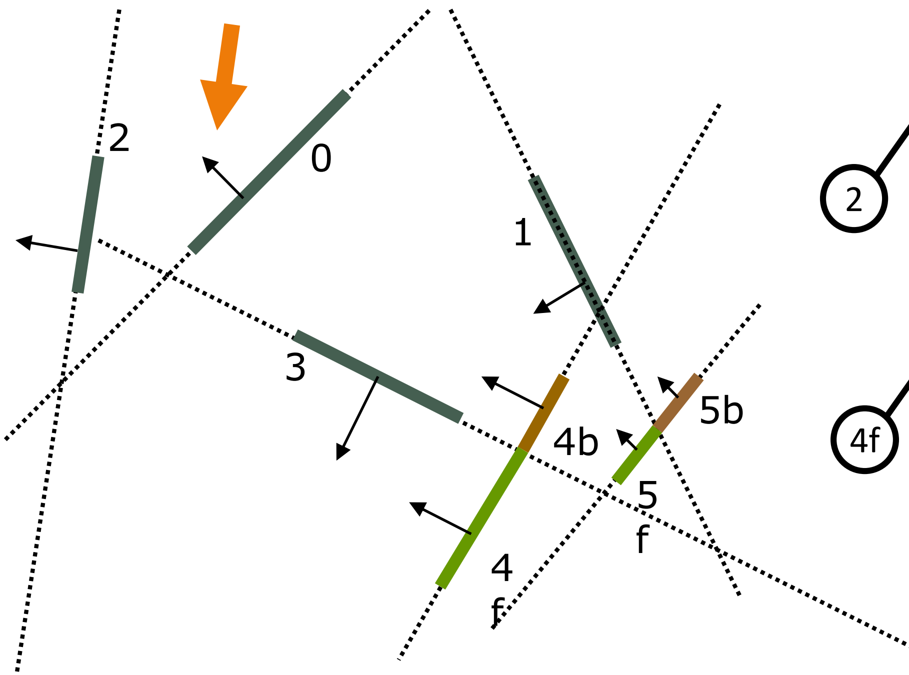
```
Render(node, view){
   if node is a leaf
    { draw this node to the screen }
   else
    if the viewpoint is in back of the dividing line
    {
    render(front subnode)
            draw node to screen
            render(back subnode)
    }
      else the viewpoint is in front of the dividing line
        {
        render (back subnode)
            draw node to screen
            render (front subnode)
    }
```

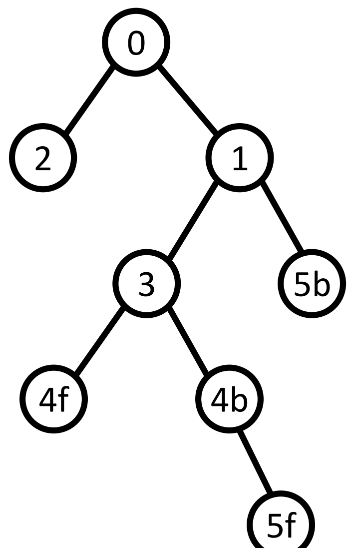
Back-to-Front Render





Back-to-Front Render

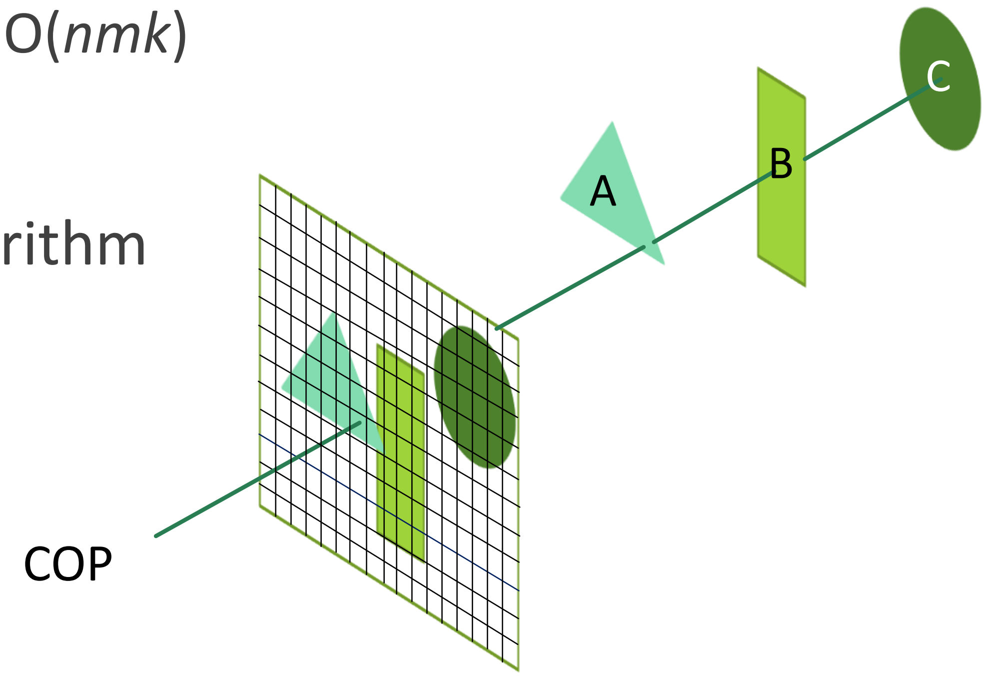




What's the limitation?

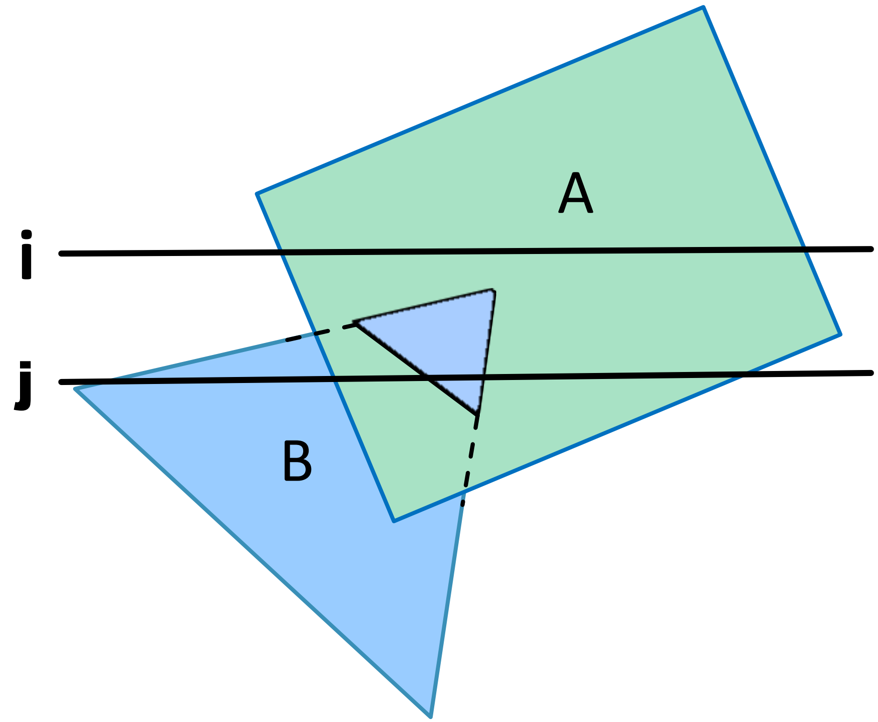
Image Space Approach

- Look at each projector (nm for an $n \times m$ frame buffer) and find closest of k polygons
- Ray casting
  - Complexity O(nmk)
- Scan-line algorithm
- z-buffer



Scan-Line Algorithm

- Can combine shading and HSR through scan-line algorithms.



scan line **i**:
no need for depth information

scan line **j**:
need depth information when A and B overlap

This technique is not considered to scale well as the number of primitives increases.

Scanlines and Z

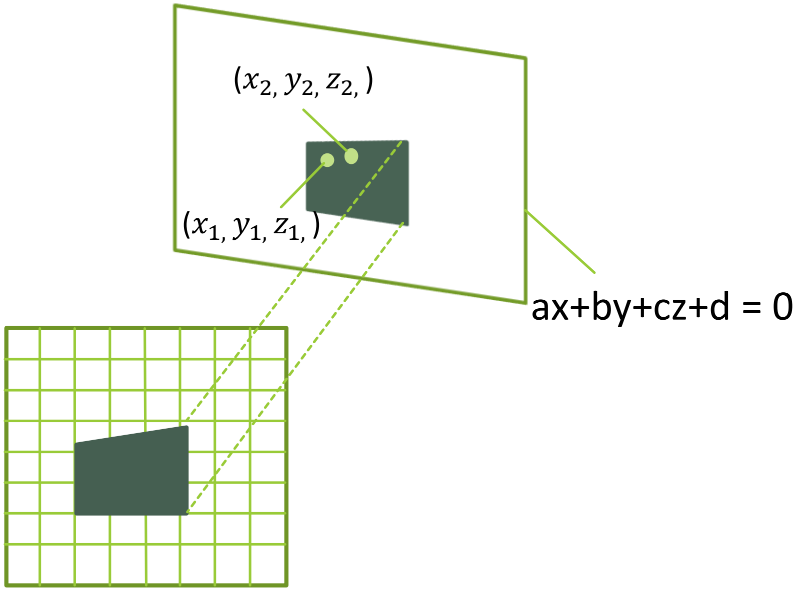
- If we work scan line by scan line as we move across a scan line, the depth changes satisfy $a\Delta x+b\Delta y+c\Delta z=0$

Along scan line

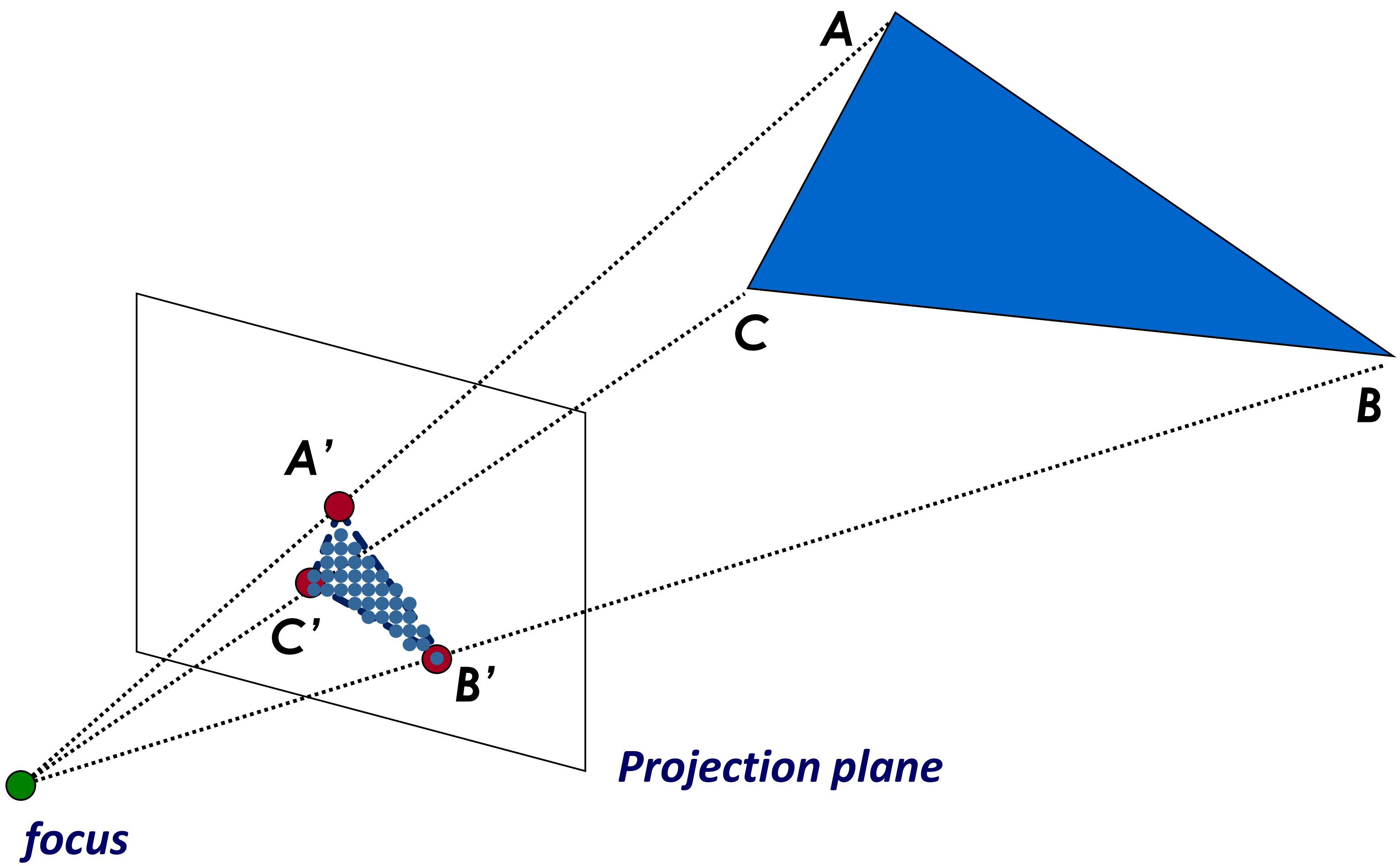
$$\Delta y = 0$$

$$\Delta z = -\frac{a}{c}\Delta x$$

In screen space $\Delta x = 1$

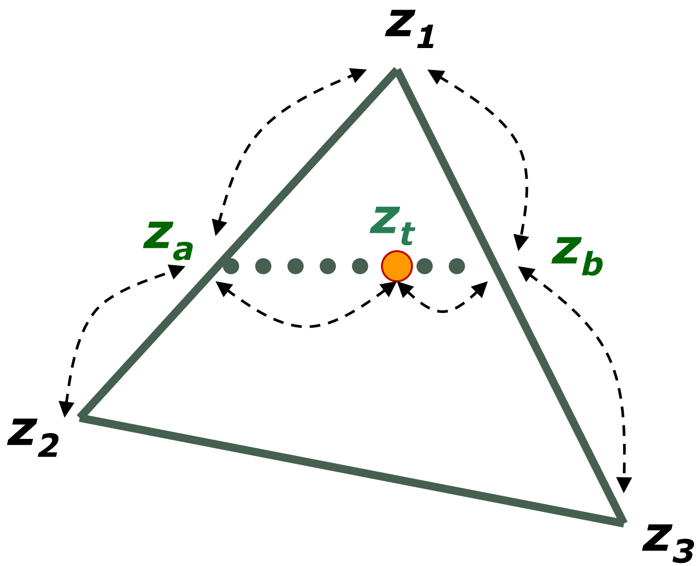


Reminder: Rasterization Operations



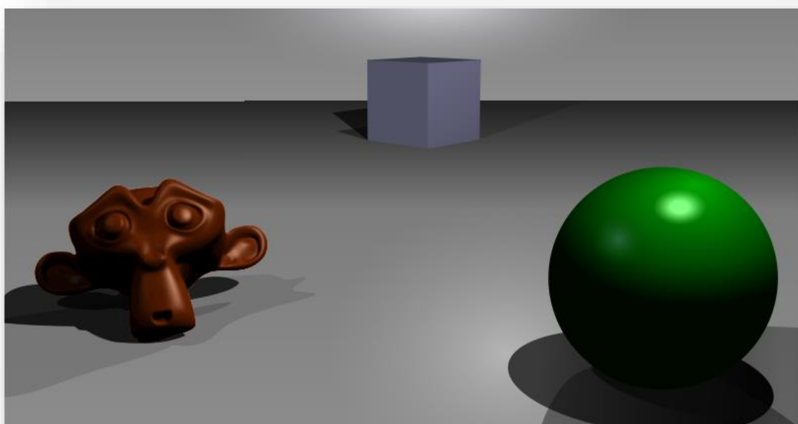
Interpolation of Z values

- Rasterizer can help us estimate variables in-between pixels (including z).



z-Buffer Algorithm

- The z or depth buffer
  - an additional buffer besides the color frame buffer.
  - storing the **depth** of the **closest** object at **each pixel** found **so far**.



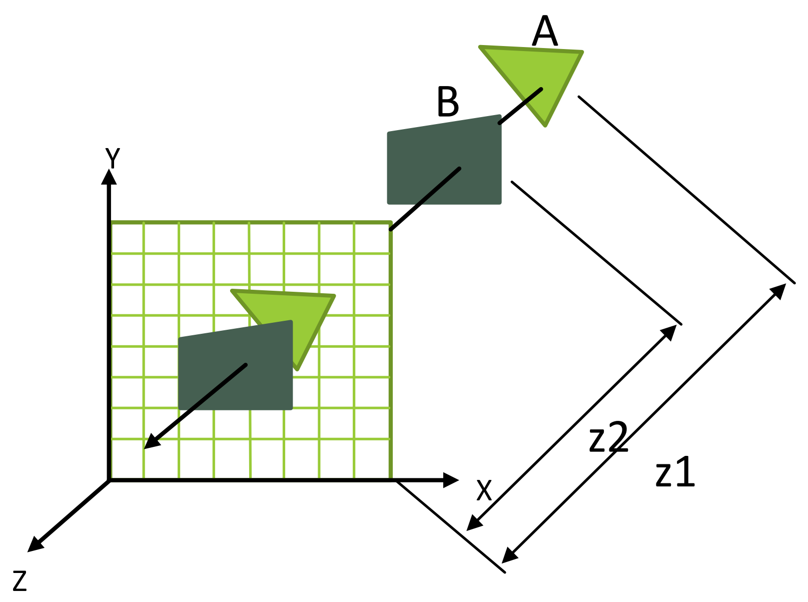
Color buffer of a 3D scene

Depth buffer of the corresponding scene

Fig. from en.wikipedia.org/wiki/Z-buffering

z-Buffer Algorithm (cont.)

- As we render each polygon, compare the depth of each pixel to depth in z buffer
  - If it is closer, place the shade of pixel in the color buffer and update z buffer.
  - If it is farther, do not update the color and z buffers .



More about Space Partitioning

- In many real-time applications, we want to eliminate as many triangles as possible within the application.
  - Reduce burden on pipeline
  - Reduce traffic on bus
- With spatial information, we can
  - Avoid rendering an object when it's unnecessary.
  - Use a coarse model when an object is far from the viewer.
  - Preload neighbor regions for seamless scene change.
  - …….
- E.g. BSP tree , Octree, ……

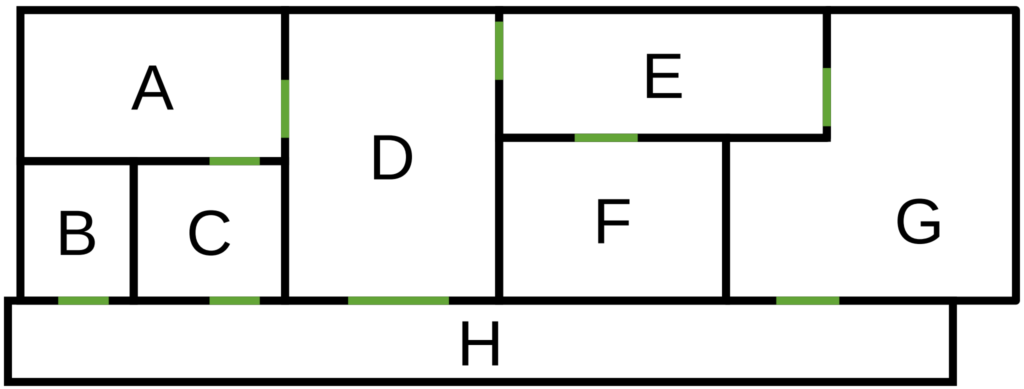
BSP-based Culling

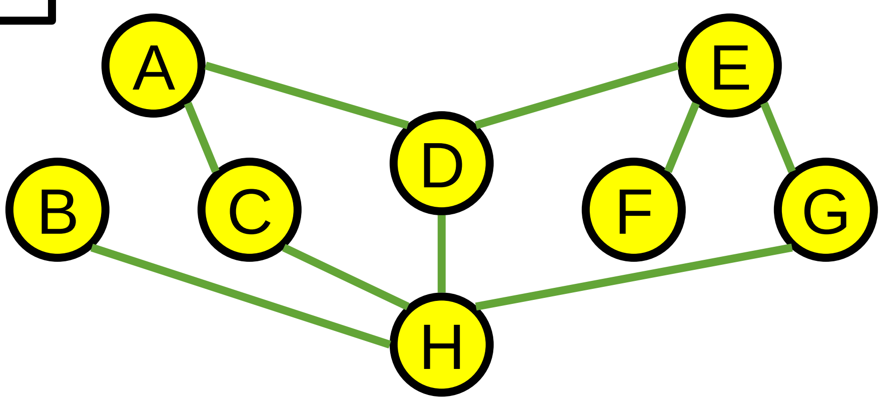
- Pervasively used in first person shooting games.
  - Doom, quake….etc.

- Visibility test
- Skip objects that are “occluded”.

Other Culling Methods

- Portal Culling
  - Walking through architectures
  - Dividing space into cells
  - Cells only see other cells through portals





Ref: www.cse.ohio-state.edu/~hwshen

Octree

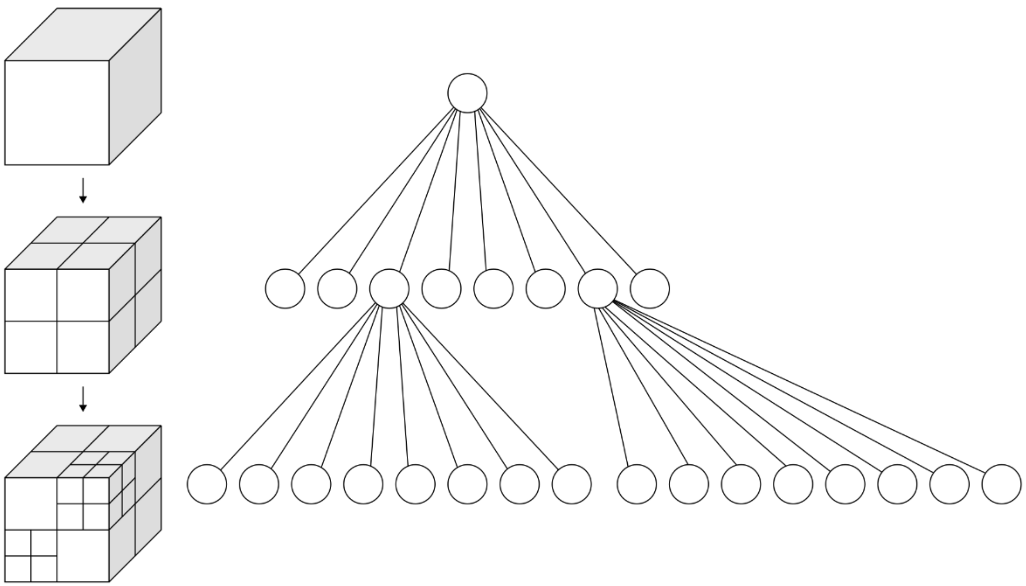


Fig. from en.wikipedia.org/wiki/Octree

Space partition for open-world 3D games

- Applying level of details (LOD) and preloading is critical for a large scene.

The End of Chapter