Introduction to Computer Graphics

7. Hidden Surface Removal & Culling

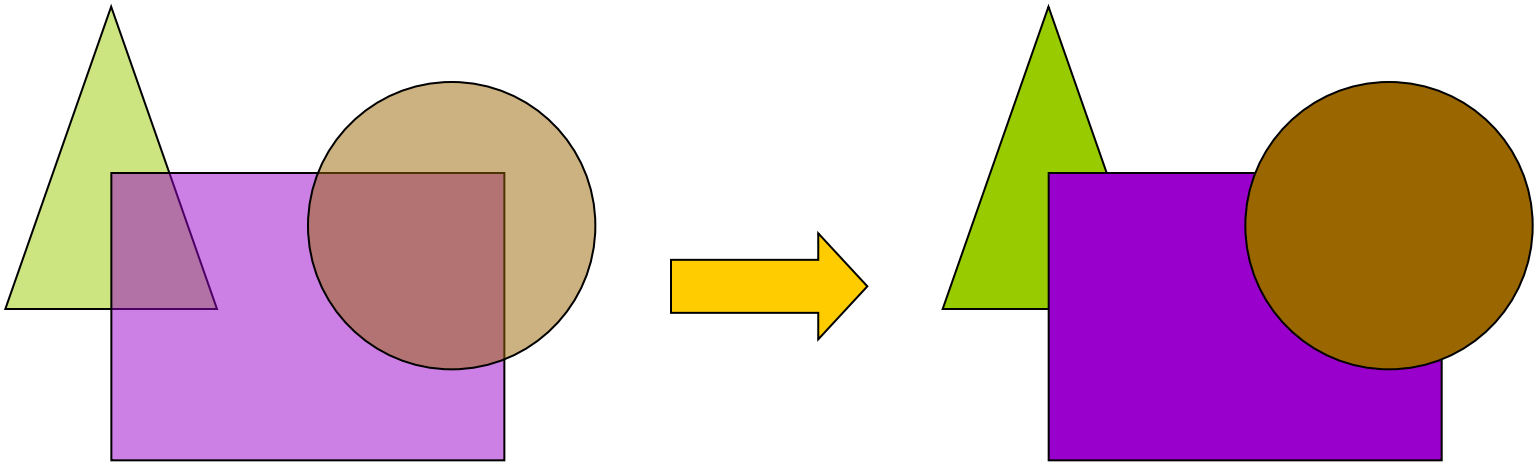
National Chiao Tung Univ, Taiwan
By: I-Chen Lin, Assistant Professor

Textbook: E.Angel, Interactive Computer Graphics, 5th Ed., Addison Wesley
Ref: Hearn and Baker, Computer Graphics, 3rd Ed., Prentice Hall

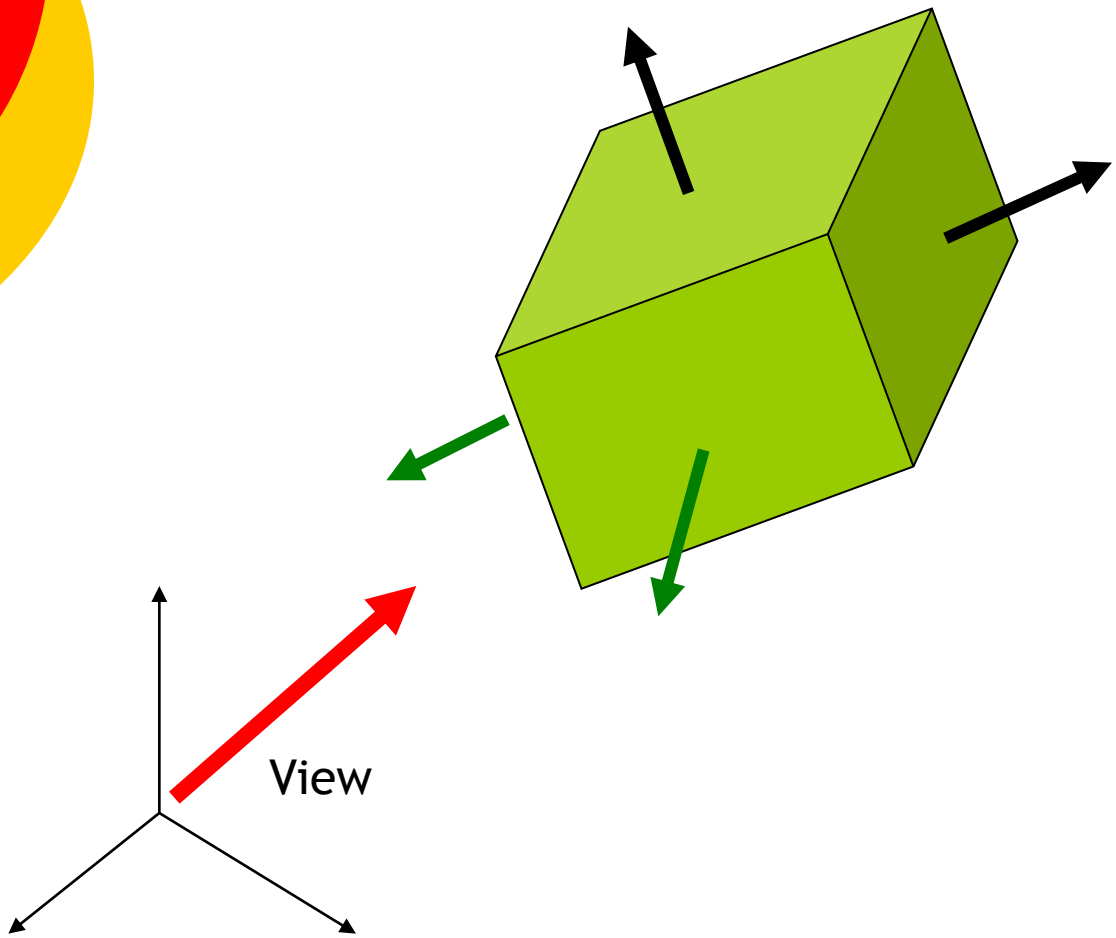
Objectives

- In Hw1 (3D wireframe displayer), we can simply draw the line segments between projected point pairs.

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

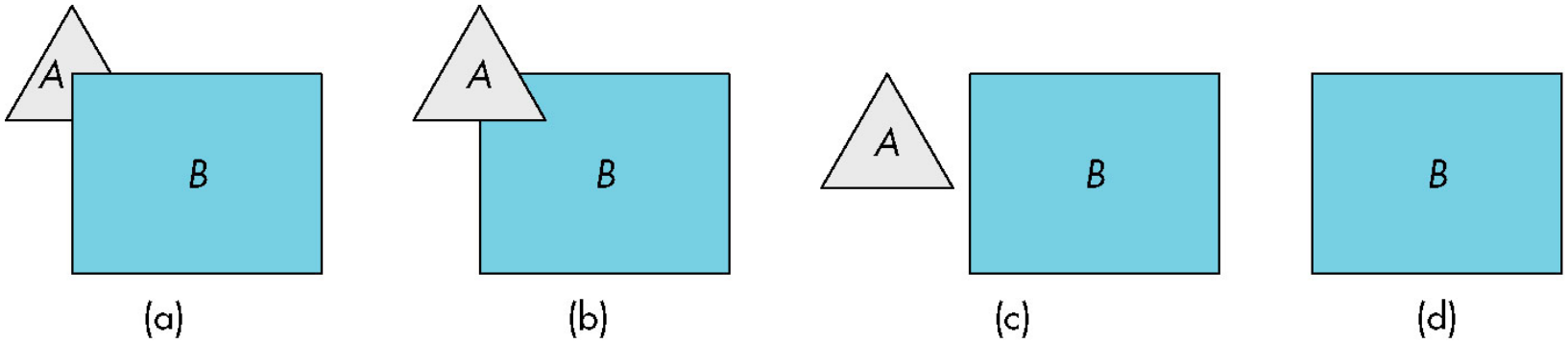


Backface Culling



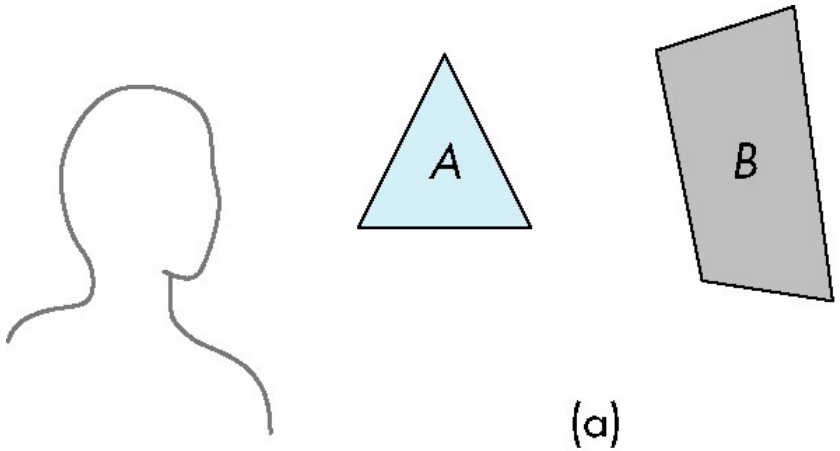
Hidden Surface Removal

- Object-space approach: use pairwise testing between polygons (objects)
- Worst case complexity O(n^2) for n polygons

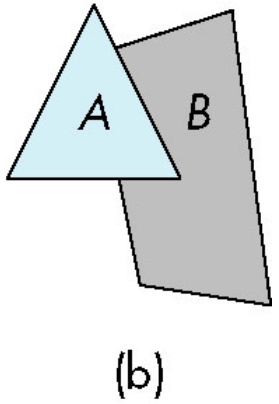


Painter's Algorithm

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



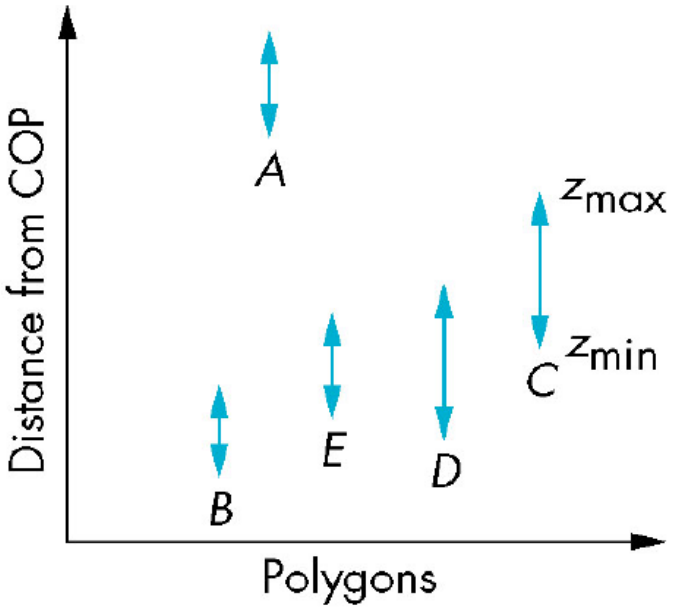
B behind A as seen by the viewer



Fill B then A

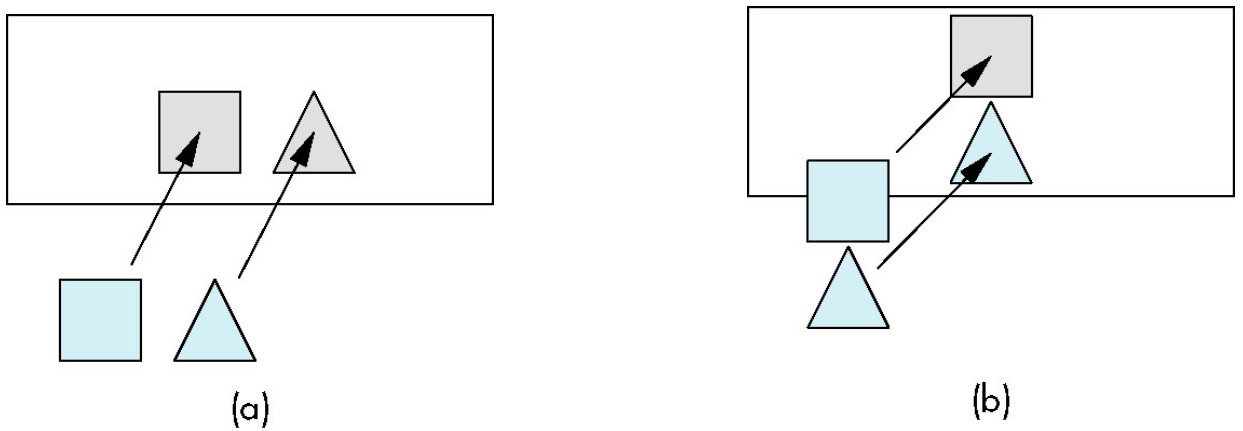
Depth Sort

- Requires ordering of polygons first
  - O(n log n) calculation for ordering
  - Not every polygon is either in front or behind all other polygons
- Order polygons and deal with easy cases first, harder later

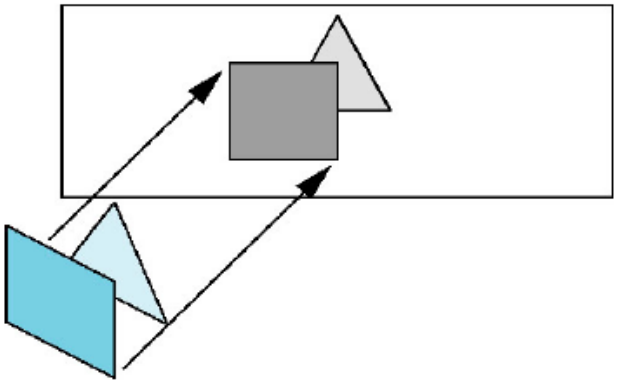


Easy Cases

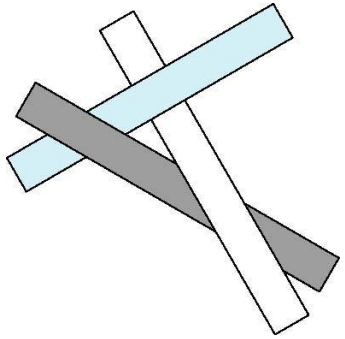
- A polygon lies behind all other polygons
  - Can render
- Polygons overlap in z but not in either x or y
  - Can render independently



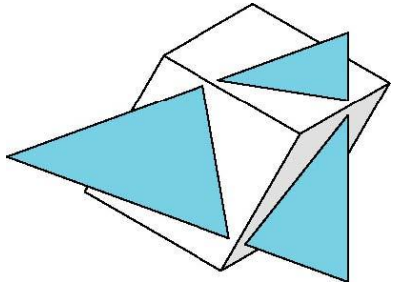
Difficult Cases



Overlap in all directions but can one is fully on one side of the other



cyclic overlap



penetration

Image Space Approach

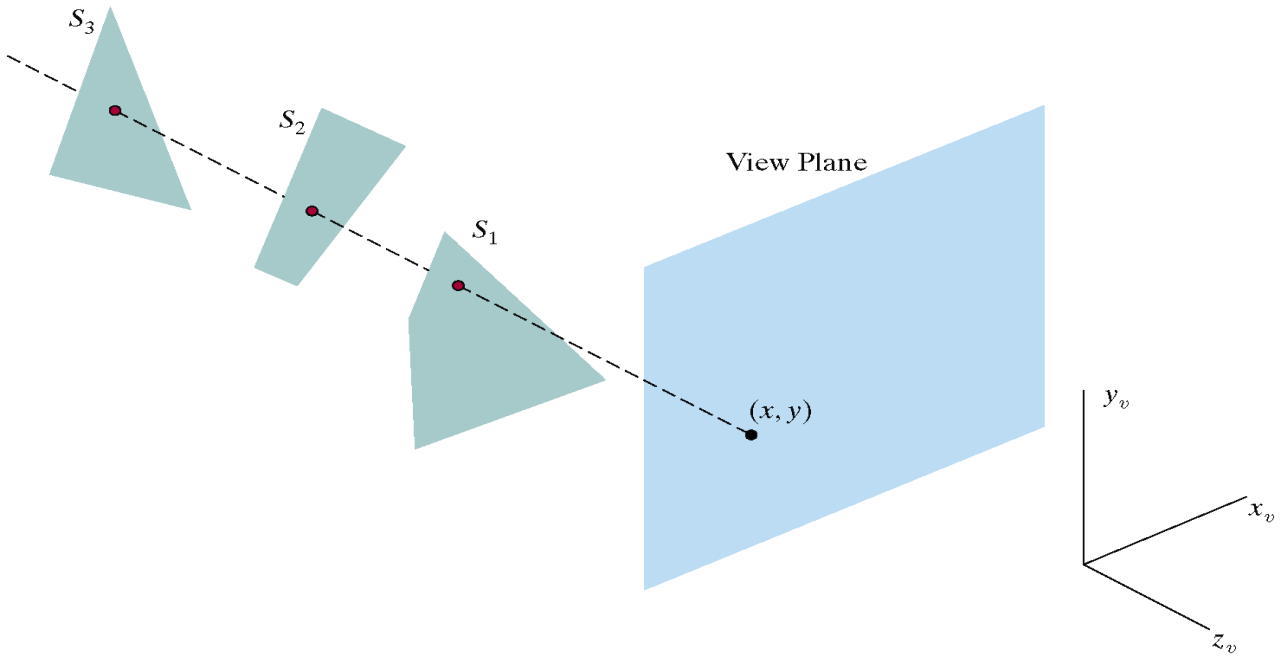


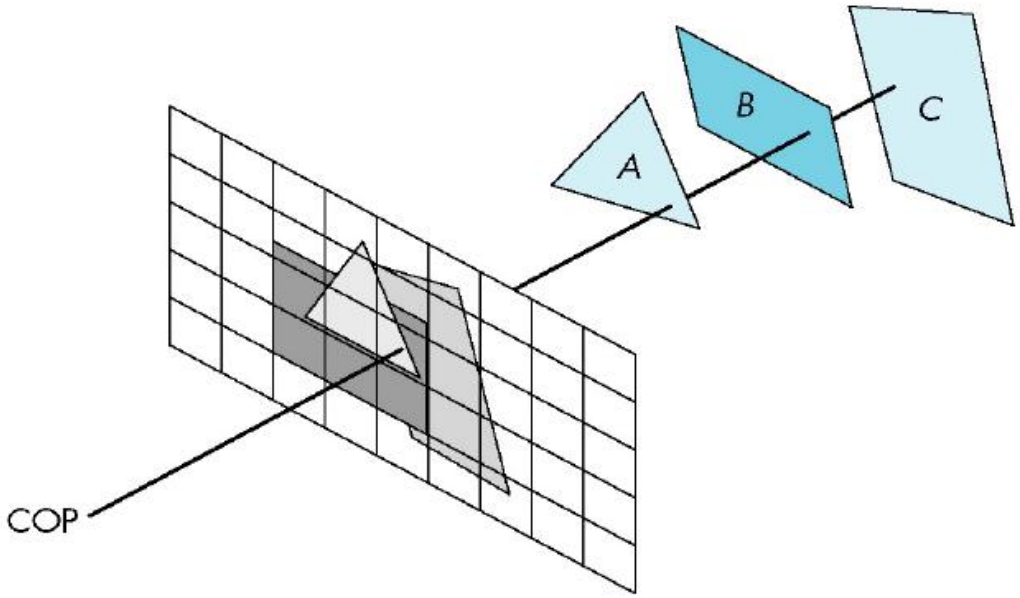
Figure 9-4

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

Image Space Approach

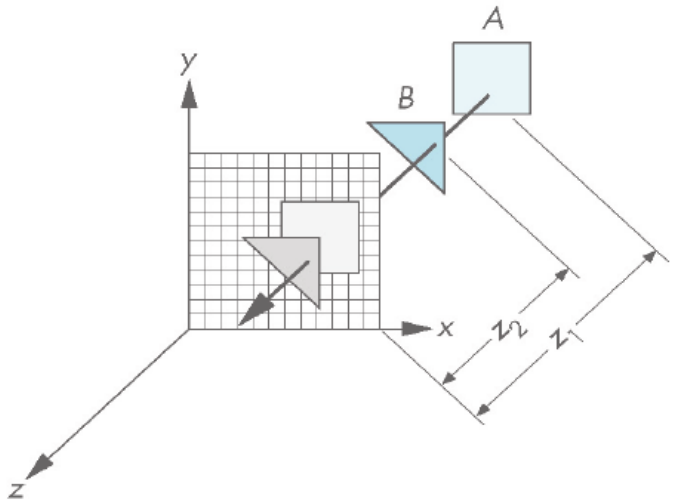
- Look at each projector (nm for an $n \times m$ frame buffer) and find closest of k polygons
- Complexity O(nmk)

- Ray casting
- z-buffer



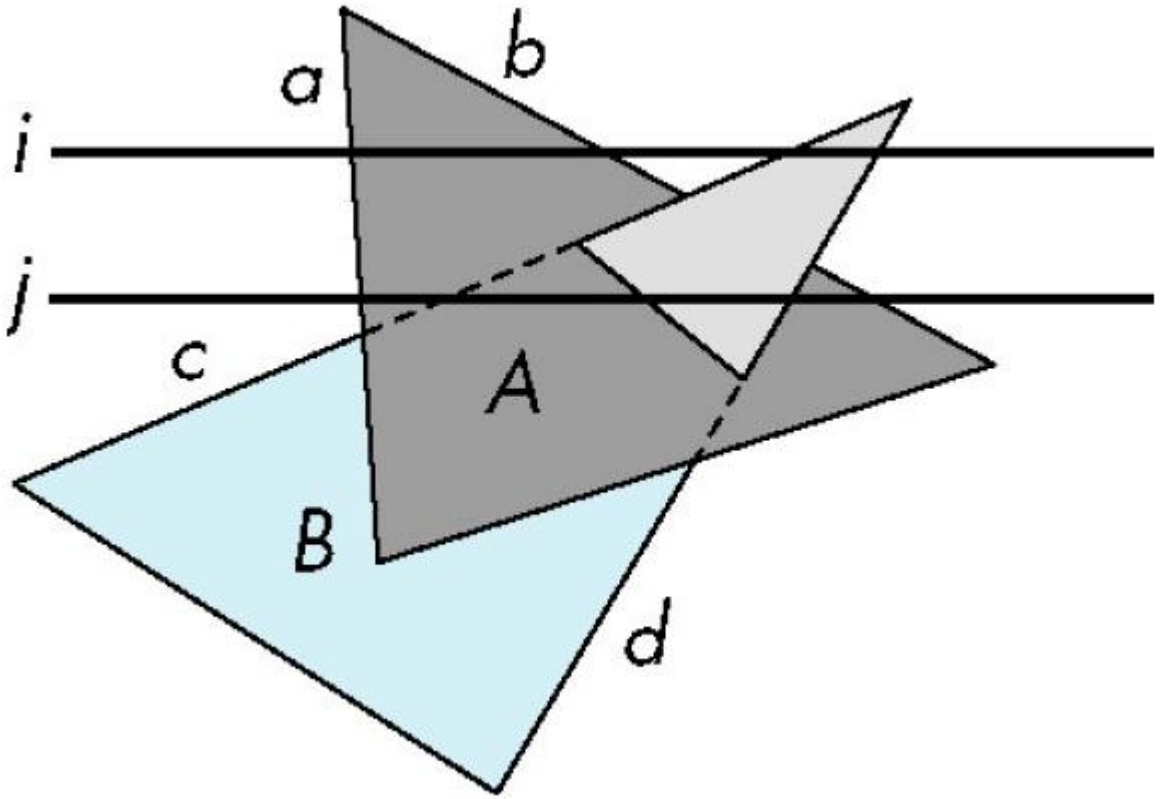
z-Buffer Algorithm

- The z or depth buffer
  - store the depth of the closest object at each pixel found so far
- As we render each polygon, compare the depth of each pixel to depth in z buffer
  - If less, place the shade of pixel in the color buffer and update z buffer

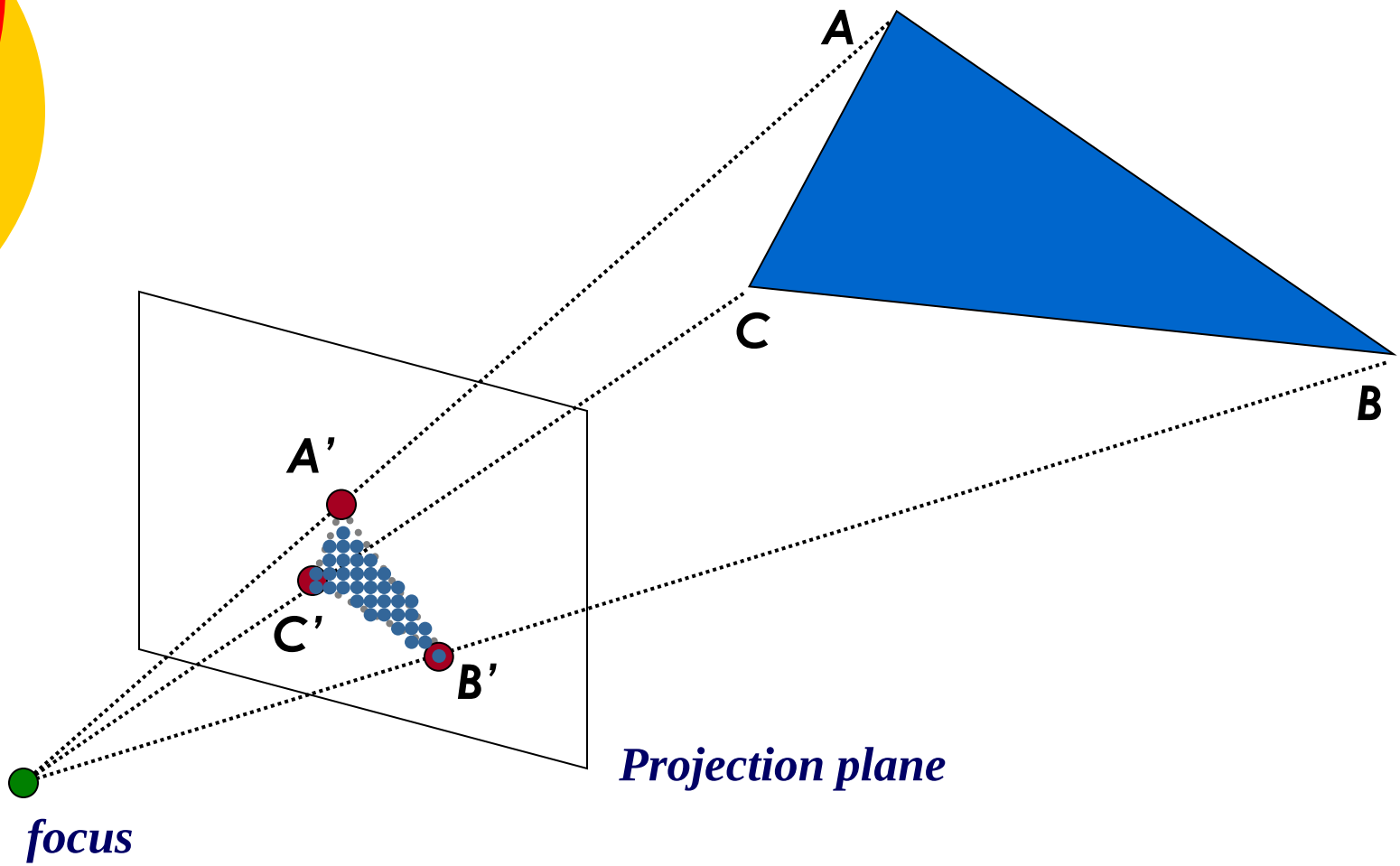


Scan-Line Algorithm

- Can combine shading and hsr through scan line algorithm



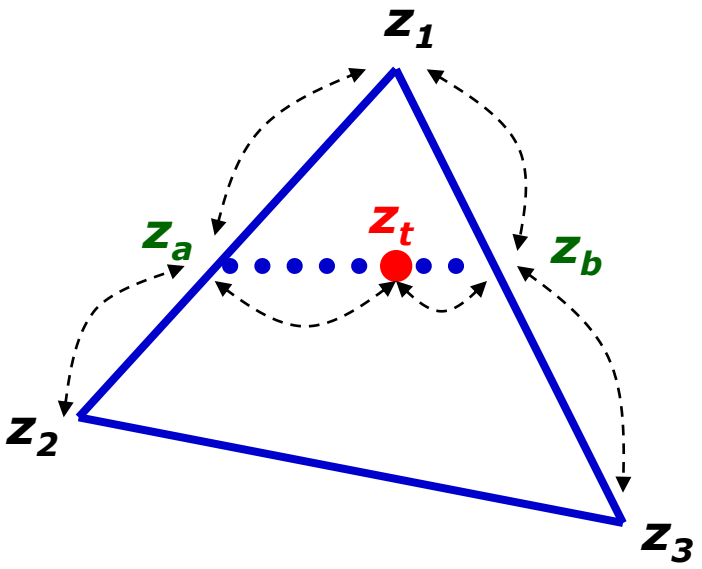
Interpolation of Z values



Interpolation of Z values

- To fill the polygon on the screen, we only fill the color and estimate the z value “pixel by pixel”.

- How to estimate z of in-between pixels ?

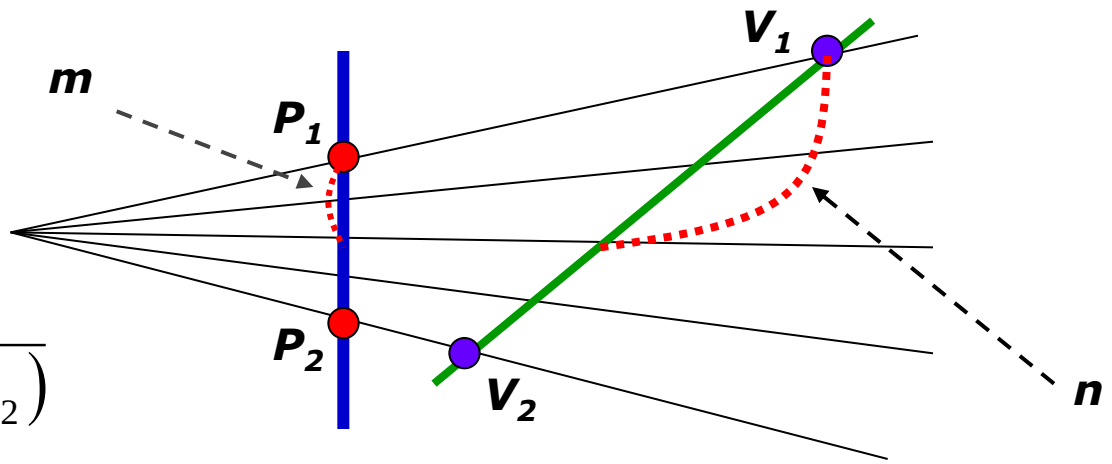


Screen Space vs. 3D Space

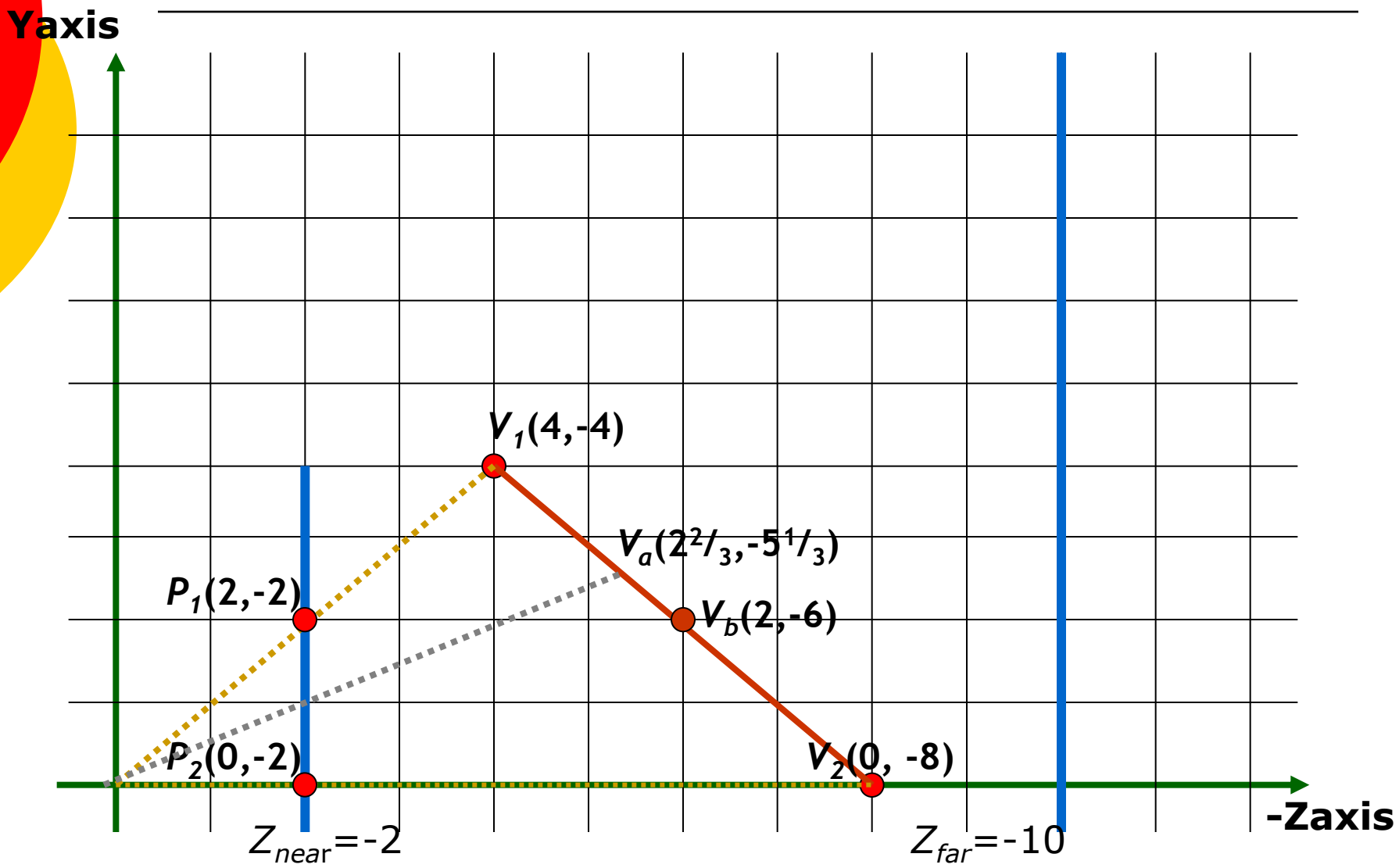
- Interpolation in screen space
  - $P(m) = P_1 + m(P_2 - P_1)$
- Interpolation in 3D space
  - $V(n) = V_1 + n(V_2 - V_1)$
  - $P_y(n) = V_y(n) \,/\, V_z(n)$

$$n \neq m$$

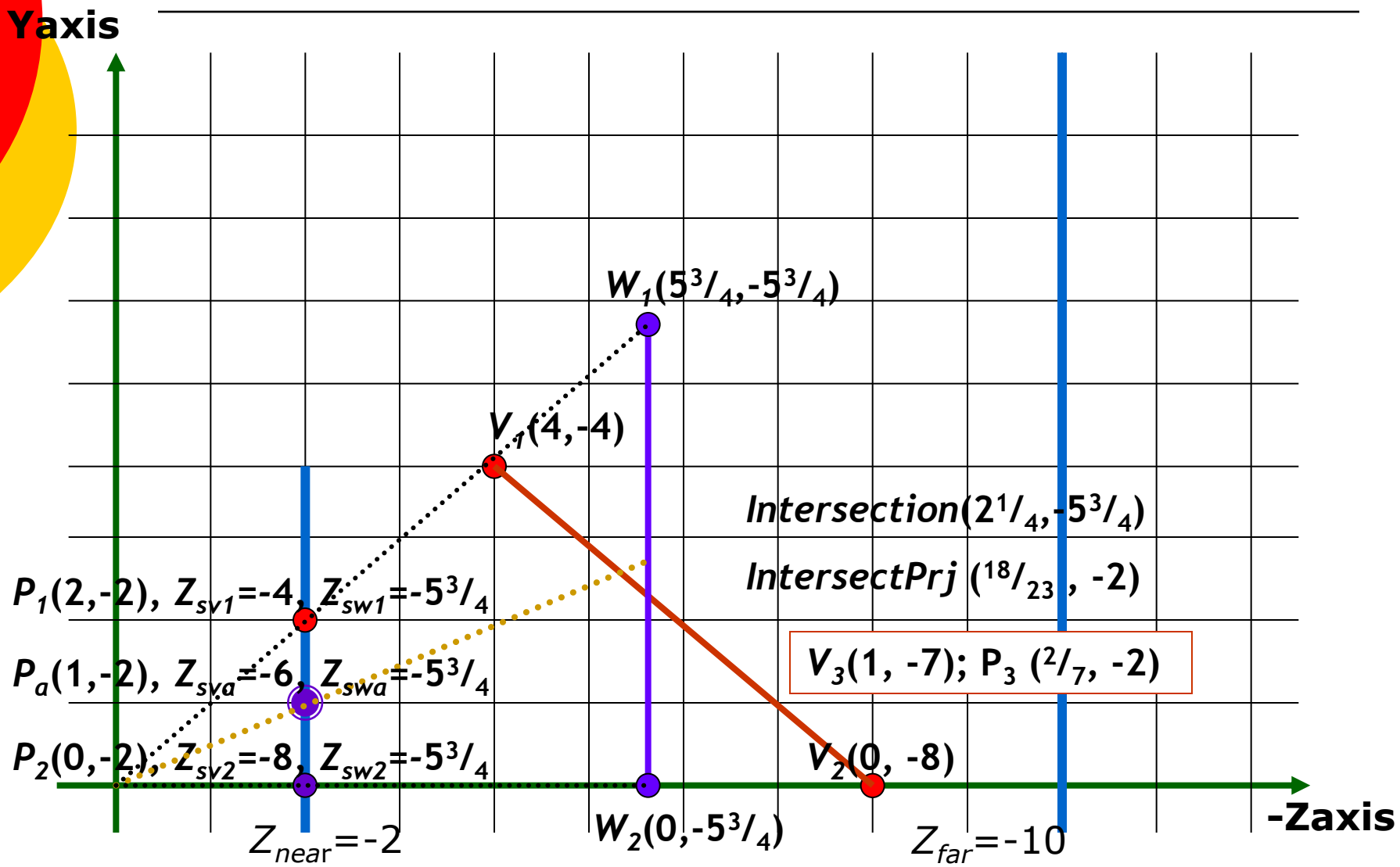
$$n = \frac{mz_1}{z_2 + m(z_1 - z_2)}$$



Screen Space vs. 3D Space

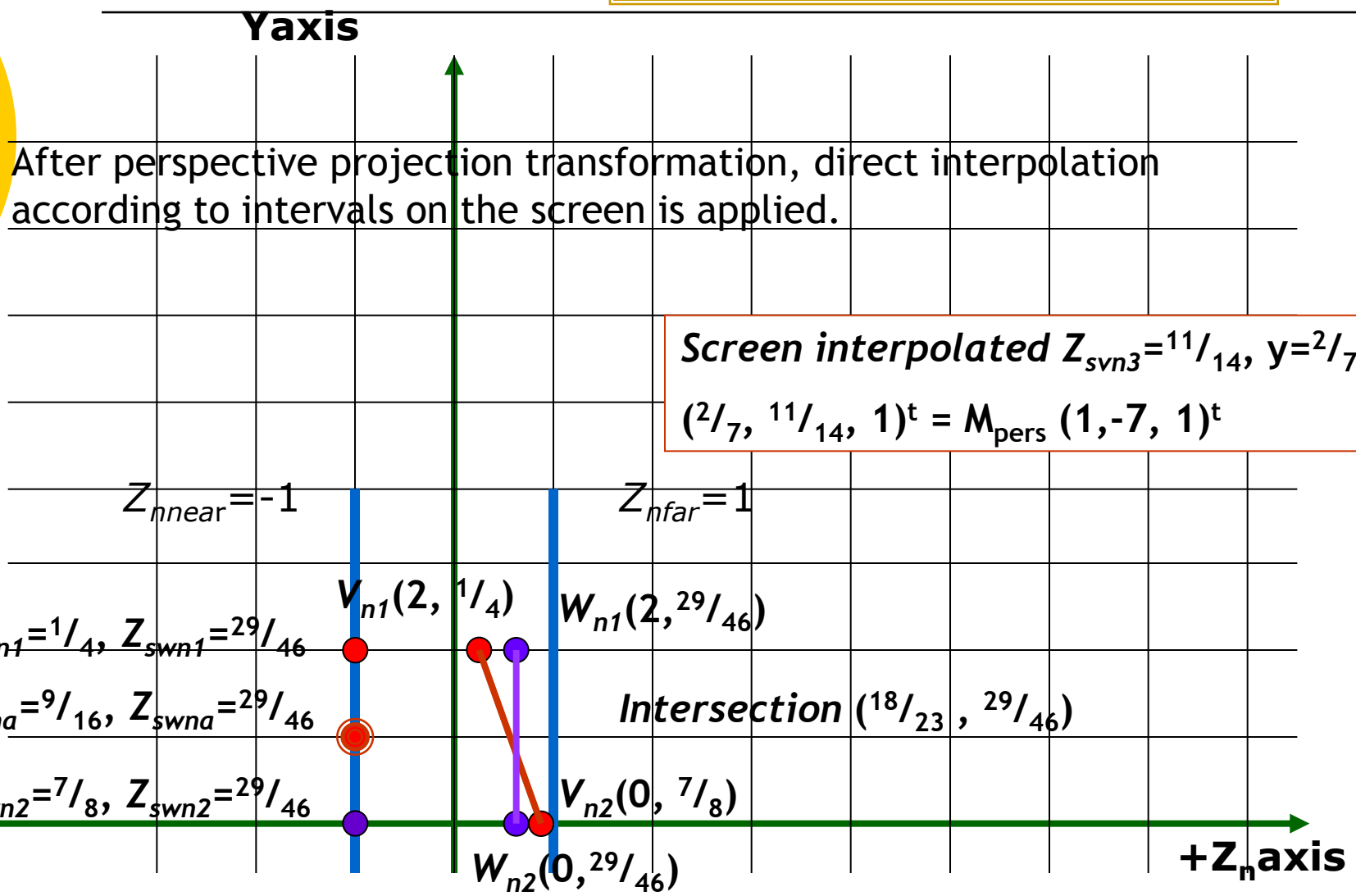


Simple Screen Interpolation



Perspective Projection Space

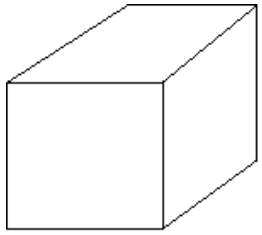
$$M_{pers} = \begin{bmatrix} -z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} & \frac{-2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



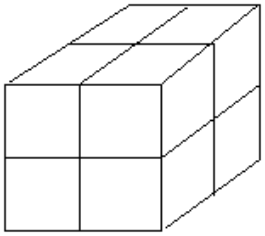
Space Partitioning

- Avoid rendering an object when it's unnecessary.
  - In many real-time applications, we want to eliminate as many objects as possible within the application.
  - Reduce burden on pipeline
  - Reduce traffic on bus
- Octree
- BSP tree

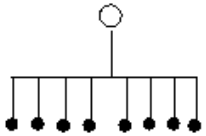
Octree

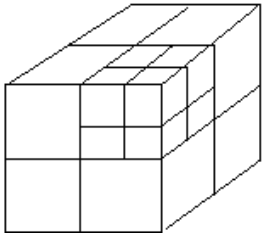


(root)

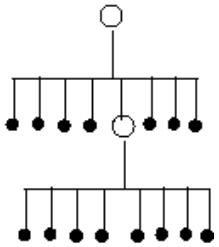


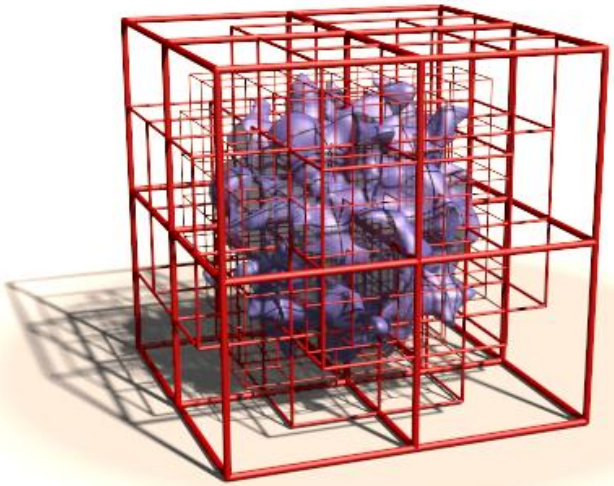
(1 level)





(2 levels)

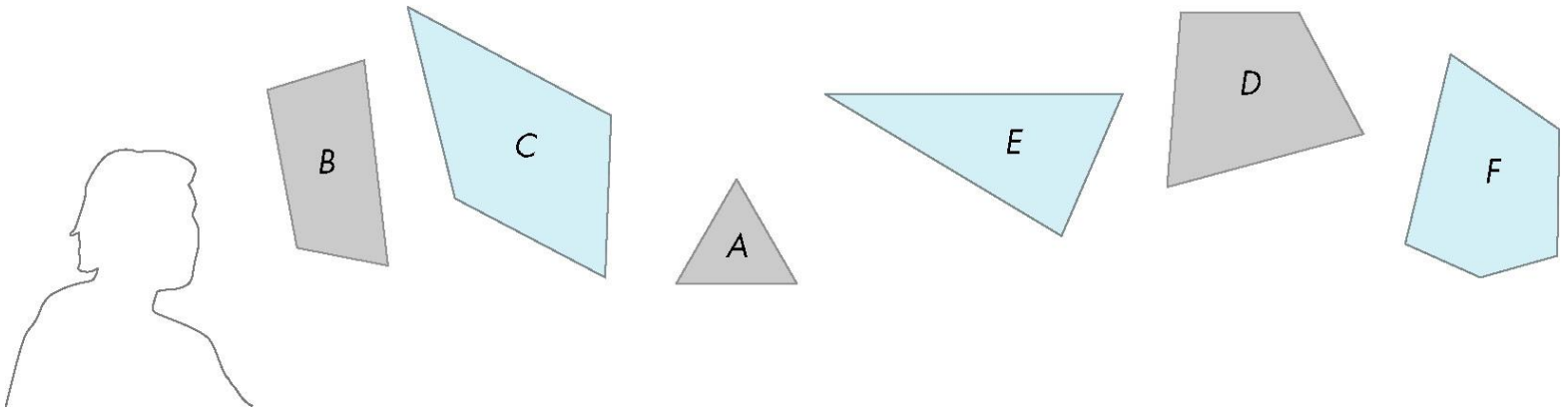




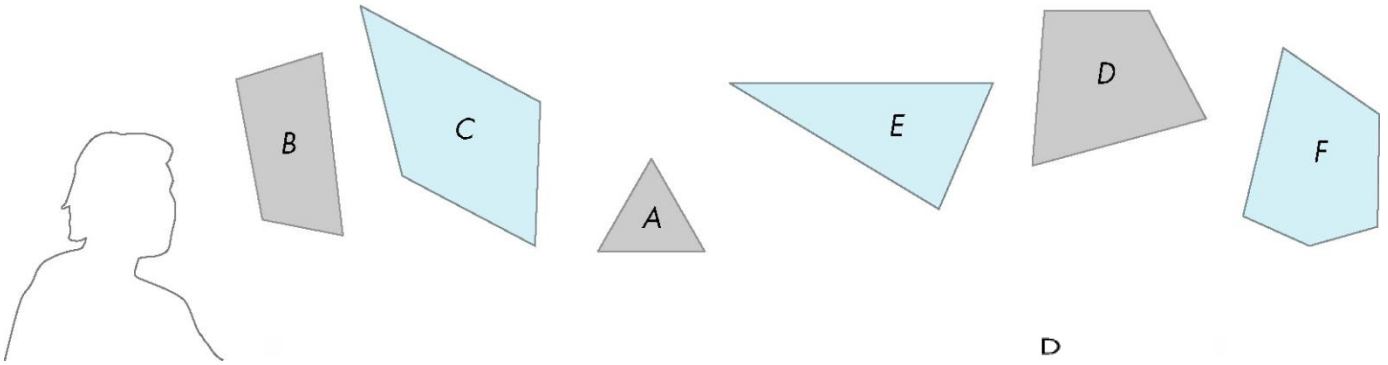
http://www.imagico.de/fast_iso/patch.html

Why do we use BSP trees?

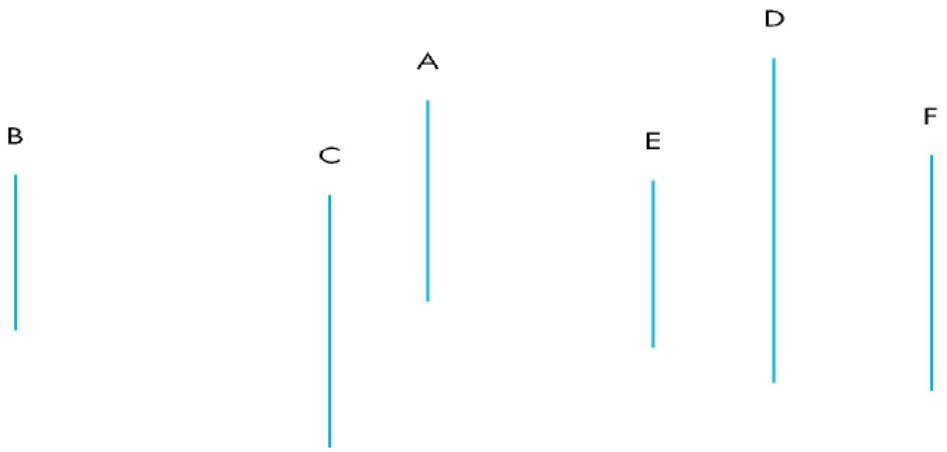
- Hidden surface removal
  - A back-to-front painter's algorithm
- Partition space with Binary Spatial Partition (BSP) Tree



A Simple Example



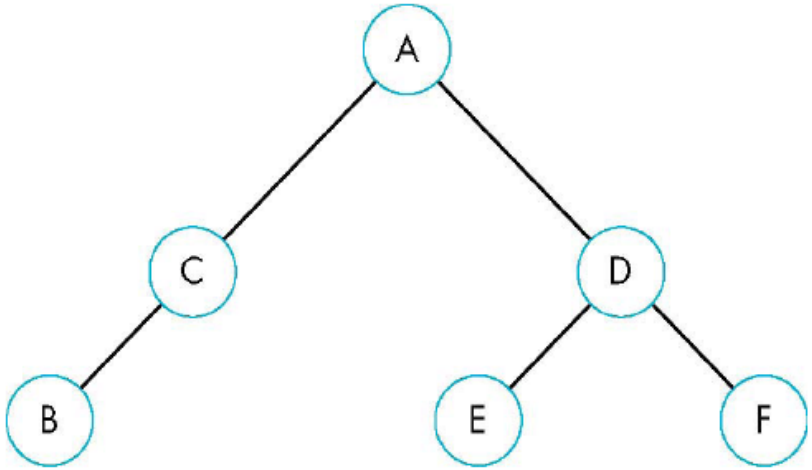
top view



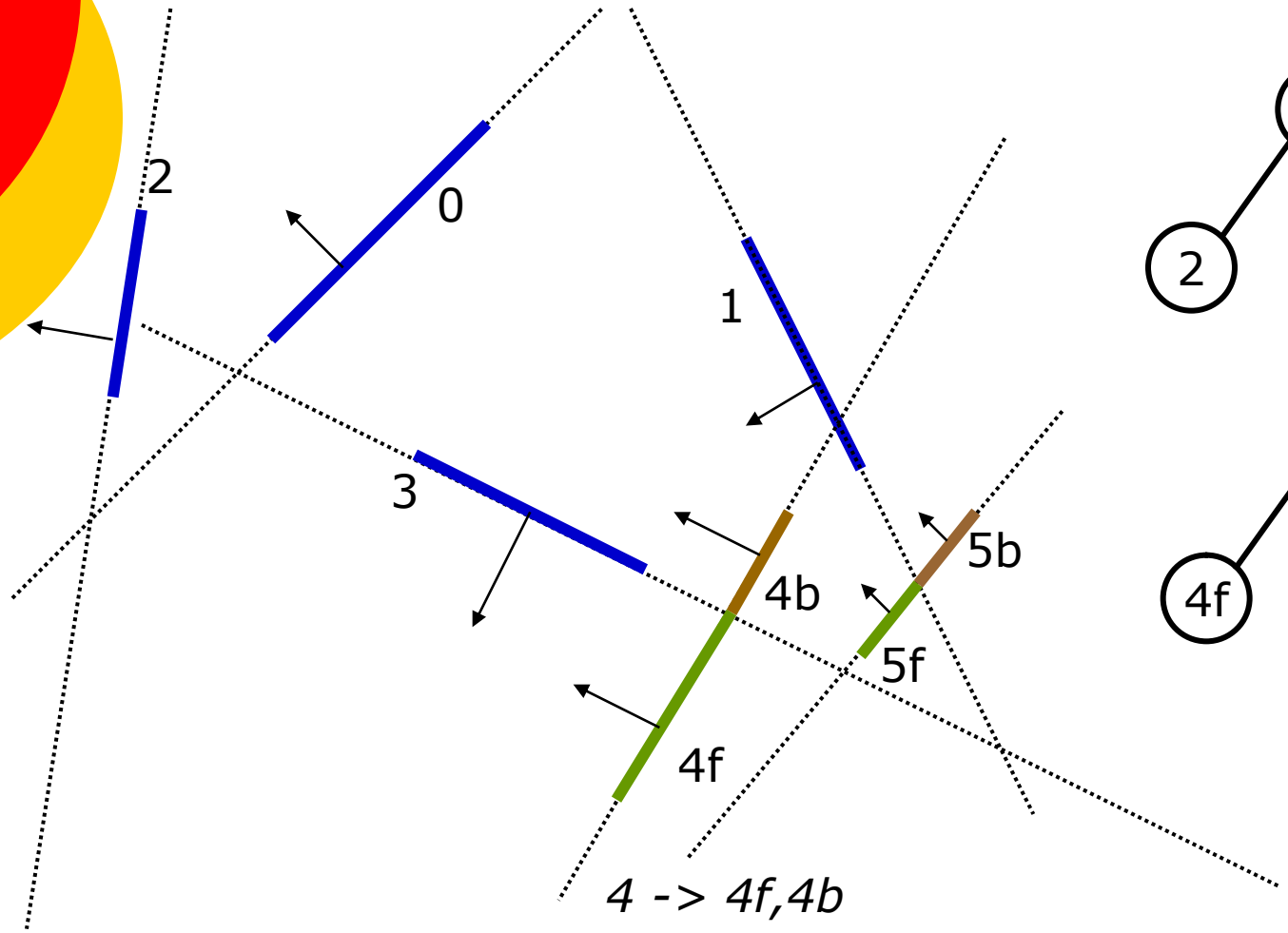
The plane of A separates B and C from D, E and F

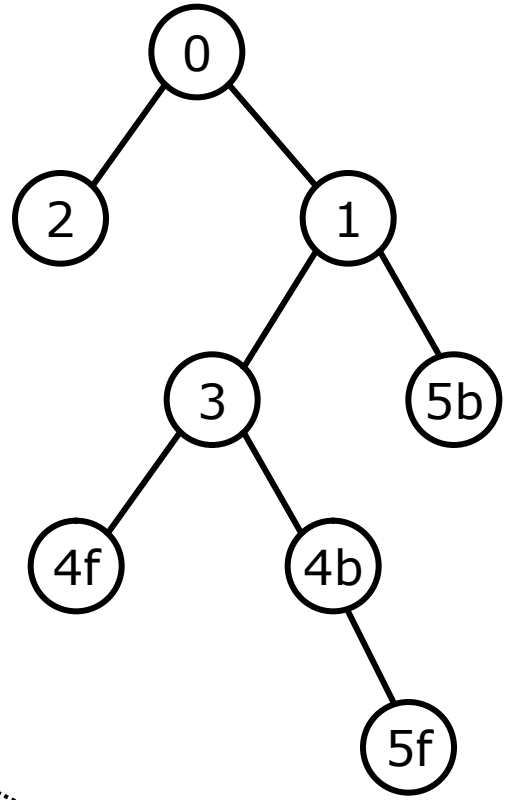
Binary Space Partitioning Tree

- 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

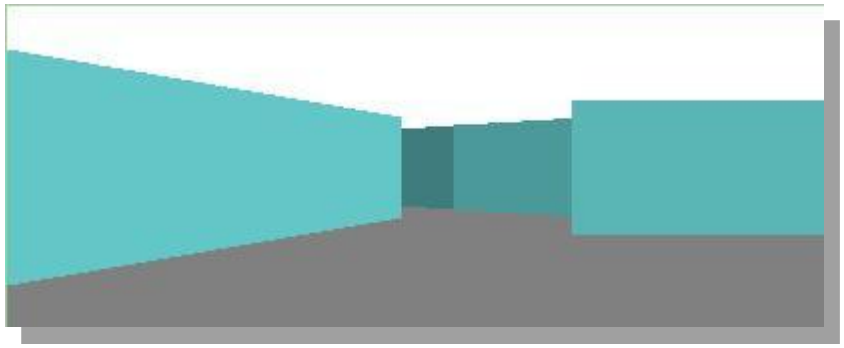


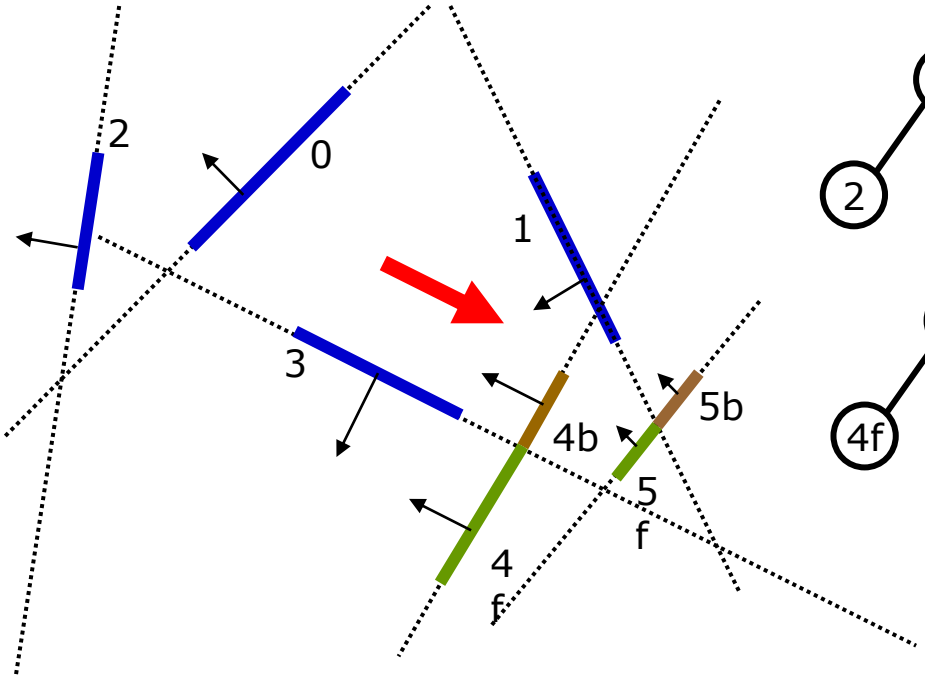


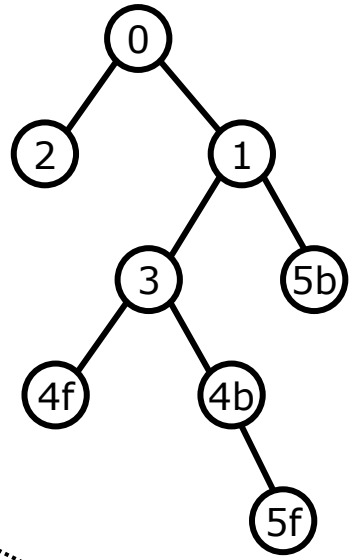
Back-to-Front Render

```
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)
        }
```

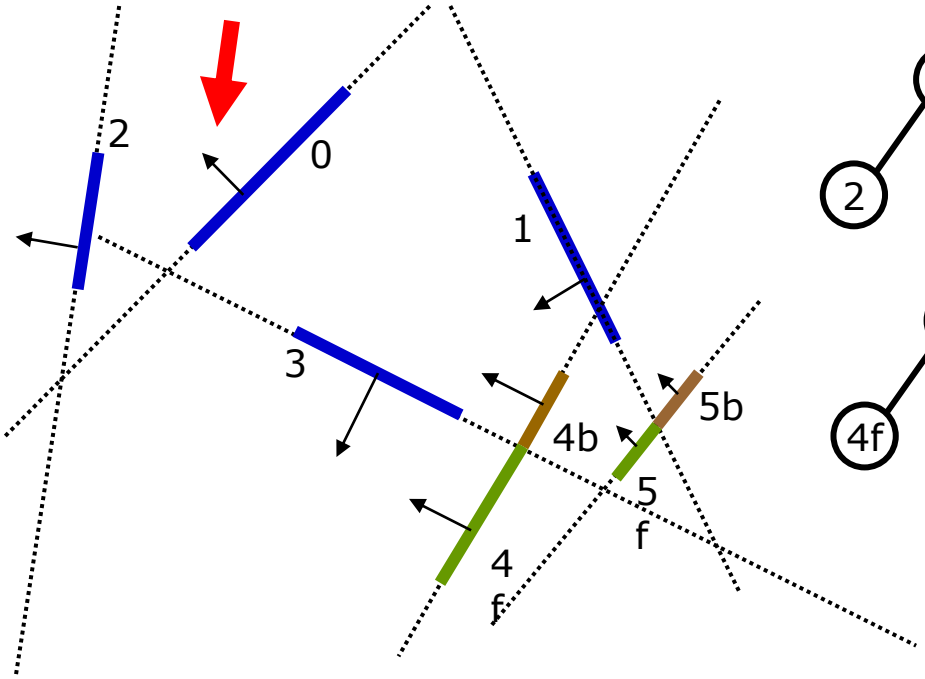
Back-to-Front Render

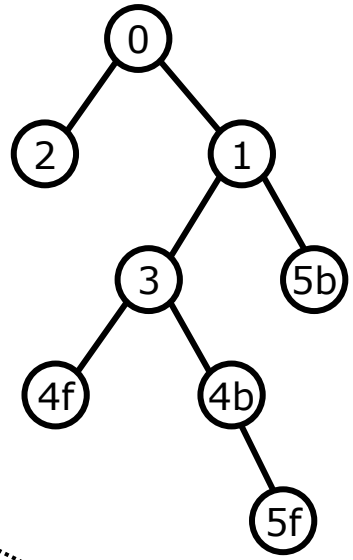






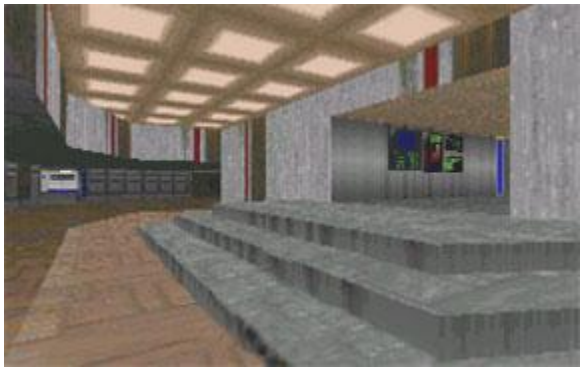
Back-to-Front Render





BSP-based Culling

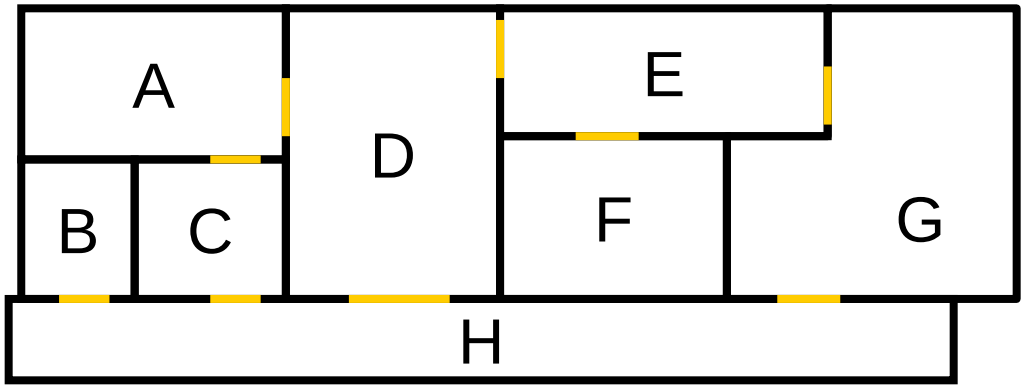
- Pervasively used in first person shooting games.
  - Doom, quake….etc.
- Visibility test
- Skip objects that are “occluded”.

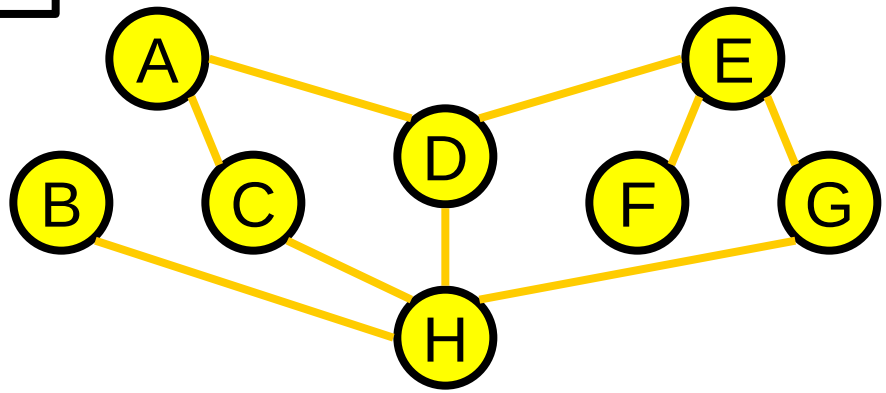


a screen shot from Doom.

Other Culling Tech.

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





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