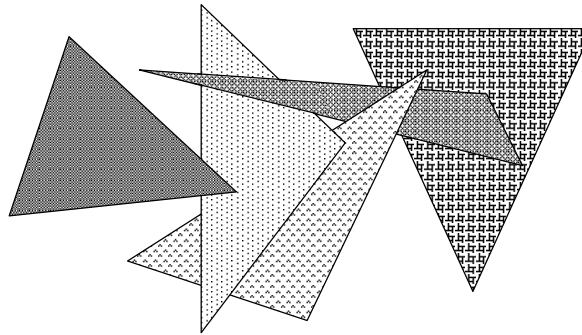


Handouts: Hidden Surface Elimination

Hidden Surface Elimination

(Visible Surface Determination)



Approaches

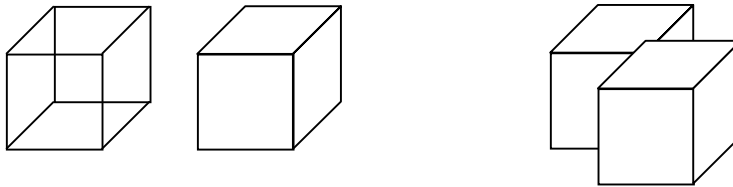


1. Back-Face Removal
2. z-Buffer (Depth-Buffer)
3. Depth-Sort
4. BSP-Tree
5. Scanline Algorithm (see book)

Handouts: Hidden Surface Elimination

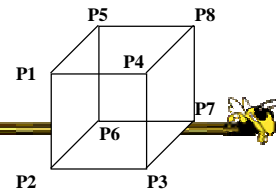
1) Back-Face Removal (Culling)

- Remove unseen polygons from convex, closed polyhedron (Cube, Sphere)
- Does not completely solve problem
 - One polyhedron may obscure another



Back Face Algorithm

- Idea. For each polygon:
 - If surface normal faces toward eye, keep
 - If surface normal faces away from eye, toss
- Adopt convention for vertex order
 - ie. assume counter-clockwise w.r.t. front
 - (P1, P2, P3, P4)
 - Compute N



Handouts: Hidden Surface Elimination

Back Face Algorithm



- Look at surface normal
 - In World Coordinates
 - | If $A x_e + B y_e + C z_e + D < 0$
The polygon is a backface.
 - After Normalizing/Perspective Projection?
 - | What is (x_e, y_e, z_e) ?

2) z-Buffer (Depth-Buffer)



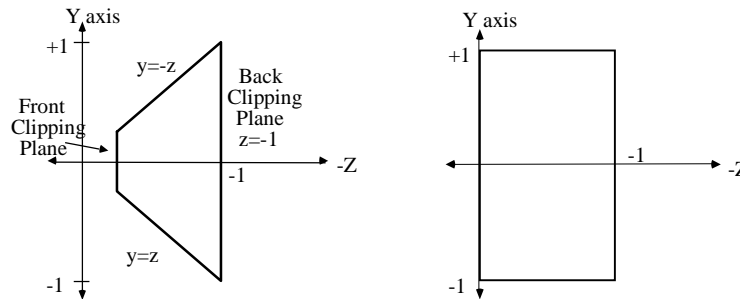
- Look at each pixel
 - Pixel shows closest object in world

- We have all info to compute $z(x,y)$

Handouts: Hidden Surface Elimination

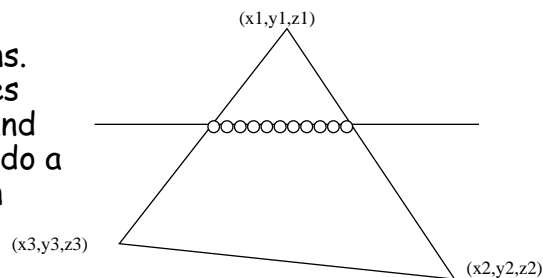
Which Z?

- Recall canonical view volumes:



Computing Pixel z-values

- Perspective projection gives z-values for vertices of polygons. To find the z-values for the boundary and interior pixels you do a linear interpolation



- Vertically: $z_{i+1} = z_i + \Delta z_v, \Delta z_v = (z_1 - z_3)/(y_1 - y_3)$
- Horizontally: $z_{i+1} = z_i + \Delta z_h, \Delta z_h = (z_1 - z_3)/(x_1 - x_3)$

Handouts: Hidden Surface Elimination

z-Buffer Algorithm



- Initialize:
 - Each z-buffer location \leftarrow Max z value
 - Each frame buffer loc. \leftarrow background color
- For each polygon:
 - Compute $z(x,y)$, depth at the pixel (x,y)
 - If $z(x,y) <$ z buffer value at pixel (x,y) then
 - $z \text{ buffer}(x,y) \leftarrow z(x,y)$
 - $\text{pixel}(x,y) \leftarrow$ color of polygon at (x,y)

z-Buffer



- Advantages
 - Linear performance
 - Polygons may be processed in any order
 - Hardware implementation \Rightarrow very fast
- Disadvantages
 - Lots of memory (nowadays ... so what?)
 - Modifications needed for antialiasing, transparency, translucency effects

Handouts: Hidden Surface Elimination

3) Depth Sort

- Sort polygons by distance
- Paint in back-to-front order
- Problems?

4) Binary Space Paritition (BSP) Trees

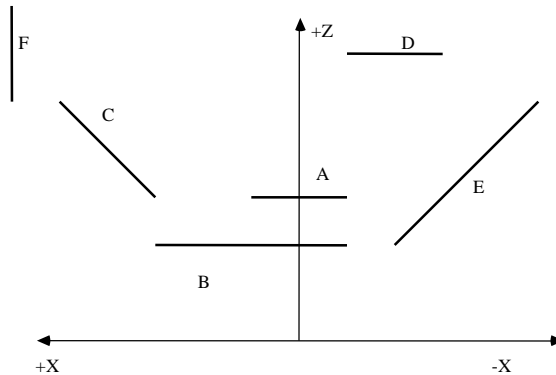
Relatively easy way to sort the polygons relative to the eyepoint

To Build a BSP Tree

1. Choose a polygon, T , and compute the equation of the plane it defines.
2. Test all the vertices of all the other polygons to determine if they are in front of, behind, or in the same plane as T . If the plane intersects a polygon, divide the polygon at the plane.
3. Polygons are placed into a binary search tree with T as the root.
4. Call the procedure recursively on the left and right subtree.

Handouts: Hidden Surface Elimination

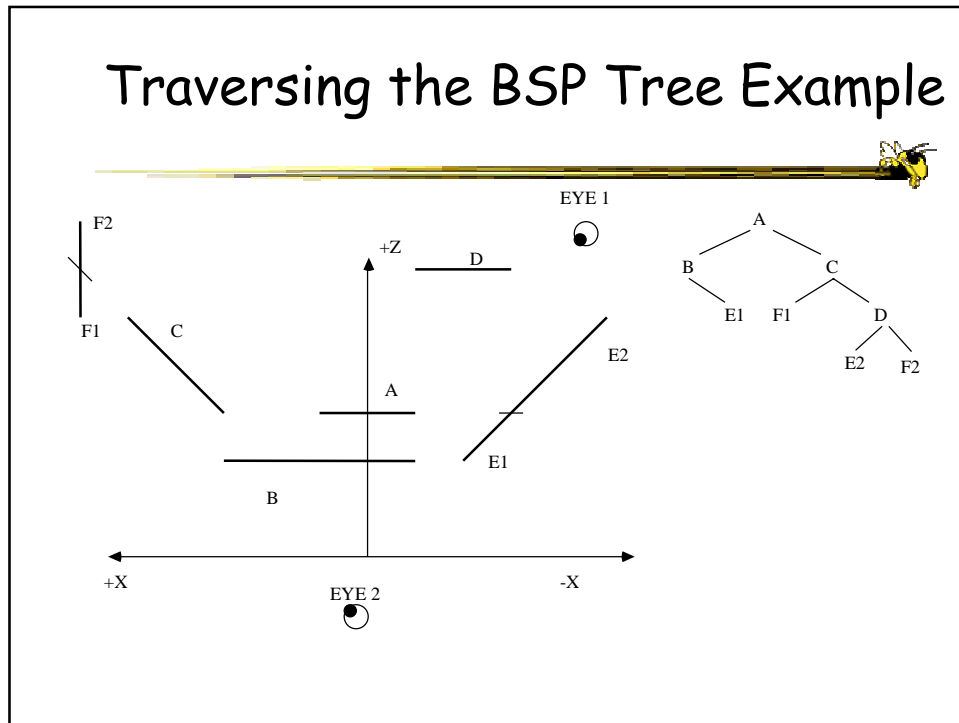
BSP Tree Example



Traversing the BSP-Tree

- Traverse the BSP tree such that the branch descended first is the side that is away from the eyepoint. This can be determined by substituting the eye point into the plane equation for the polygon at the root.
- When there is no first branch to descend, or that branch has been completed then render the polygon at this node.
- After the current node's polygon has been rendered, descend the branch that is closer to the eyepoint.

Handouts: Hidden Surface Elimination



Splitting Triangles

- If all our polygons are triangles then we always divide a triangle into more triangles when it is intersected by the plane.
- It is possible for the number of triangles to increase exponentially but in practice it is found that the increase may be as small as two fold.
- A heuristic to help minimize the number of fractures is to enter the triangles into the tree in order from largest to smallest.

