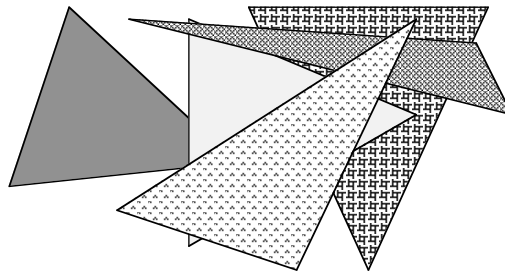




Hidden Surface Elimination

or, "Visible Surface Determination"



The problem

- Polygons and other surfaces in a 3D scene may obscure (occlude) one another
- We need to
 - remove the primitives that are hidden (HSE), OR
 - render only the primitives that are visible (VSD)



Naïve approaches

- For each pixel, intersect the ray from the eyepoint through the pixel with the objects. Draw the first object the ray hits. (Image-precision)
- For each object, determine which parts are visible and which parts are not (Object-precision)



Image- vs. Object-precision

- Image-precision
 - have to recalculate when viewpoint changes
 - generally faster but less accurate
 - $O(n \cdot p)$
- Object precision
 - done at model resolution, so more accurate
 - generally slower
 - $O(n^2)$



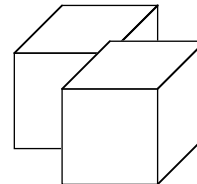
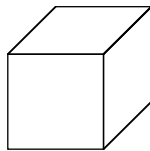
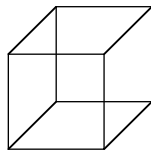
Common Approaches

1. Back-Face Removal
 - Object precision
2. z-Buffer (Depth-Buffer)
 - Image precision
3. BSP-Tree
 - Object precision



Back-Face Removal (Culling)

- Used to remove unseen polygons from a convex, closed polyhedron (Cube, Pyramid, approximated Sphere, ...)
- Only a technique to solve the hidden surface problem for single polyhedra, since one polyhedron may obscure another

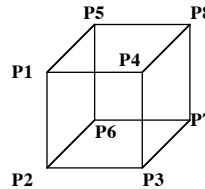




Back Face Algorithm

Before the Viewing Transformation:

1. List the vertices for each polygon clockwise with respect to the front face.
P1, P4, P3, P2
P5, P6, P7, P8



2. Compute the equation of the plane for each polygon such that the normal vector points out of the front face. (e.g. $P4P1 \times P4P3$)
3. Let (x_e, y_e, z_e) be the eyepoint (CoP)
4. If $Ax_e + By_e + Cz_e + D < 0$ The polygon is a backface.

After the Viewing Transformation:

Simply look at the z-coordinate of the Normal vector. If negative, the polygon is a backface.

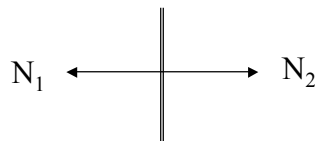
(C) Doug Bowman, Virginia Tech, 2008

7



Back-Face issues

- When culling is done depends on cost of computing plane equation vs. cost of applying the normalizing transformation
- If you have polygons that are not part of closed polyhedra, you must create two polygons in order to see each side.



(C) Doug Bowman, Virginia Tech, 2008

8



z-Buffer (Depth-Buffer)

- Concept: While rendering, keep track of the object nearest to the eye for each pixel
- Z-Buffer has memory corresponding to each pixel location. Usually ≈ 16 to 32 bits/location.

Initialize:

- Each z-buffer location \leftarrow Farthest z value
- Each frame buffer location \leftarrow background color

For each polygon:

- Compute $z(x,y)$, polygon depth at the pixel (x,y)
- If $z(x,y) >$ z buffer value at pixel (x,y) then
 - z buffer $(x,y) \leftarrow z(x,y)$
 - pixel $(x,y) \leftarrow$ color of polygon at (x,y)



z-Buffer pros and cons

Disadvantages

- Lots of extra memory
- Modifications needed to implement antialiasing, transparency, translucency effects
- Problems with precision

Advantages

- Linear performance
- Polygons may be processed in any order
- Commonly implemented in firmware/hardware so very fast.



How do you get z?

We need a z-value that preserves the depth relationship between objects after the transformation to screen coordinates.

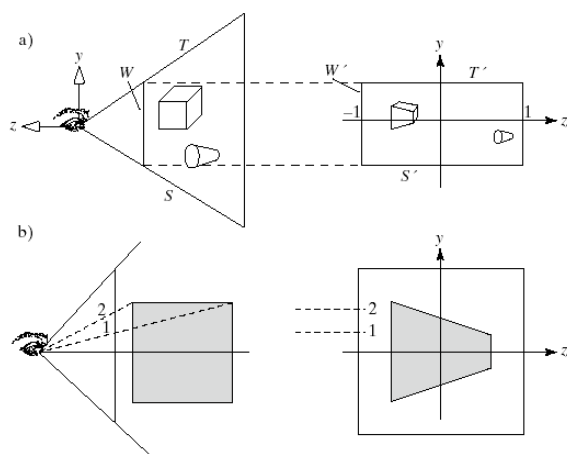
Transform vertices but maintain:

- Relative depth of points
- Linearity of lines / Planarity of polygons
- Correct perspective view



View volume transformation

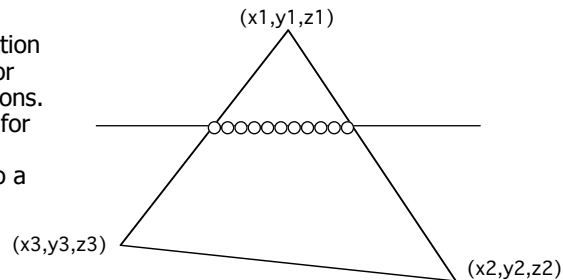
- Recall how we did perspective transformation
- z-values preserve depth; other requirements are met





Computing Pixel z-values

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



When moving vertically: $z_{i+1} = z_i + \Delta z_v$, $\Delta z_v = (z_1 - z_3) / (y_1 - y_3)$

When moving horizontally: $z_{i+1} = z_i + \Delta z_h$, $\Delta z_h = (z_1 - z_3) / (x_1 - x_3)$

This can be integrated into the polygon scan-conversion algorithm.



Binary Space Partition (BSP) Trees

- a relatively easy way to sort the polygons relative to the eyepoint
- based on splitting space into "half-planes"
- idea: we can draw the scene in the correct order for visible surfaces (back to front) if, for a polygon T, we draw all polygons behind T, then T, then all polygons in front of T
- result: viewpoint-independent hidden-surface solution

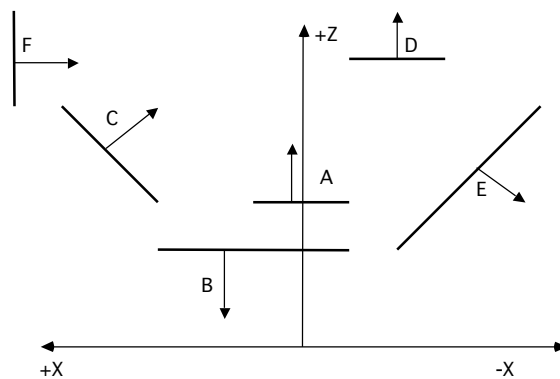


Building a BSP Tree

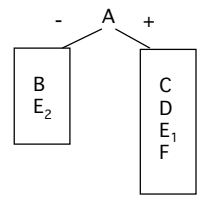
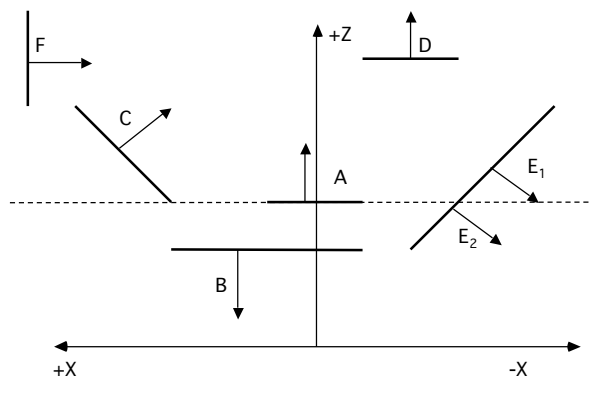
1. Choose a polygon, T , and compute the equation of the plane it defines.
2. Test all the vertices of all the other polygons to determine if they are in front of, behind, or in the same plane as T . If the plane intersects a polygon, divide the polygon at the plane.
3. Polygons are placed into a binary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



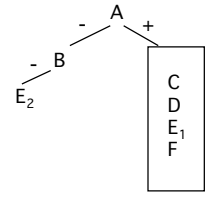
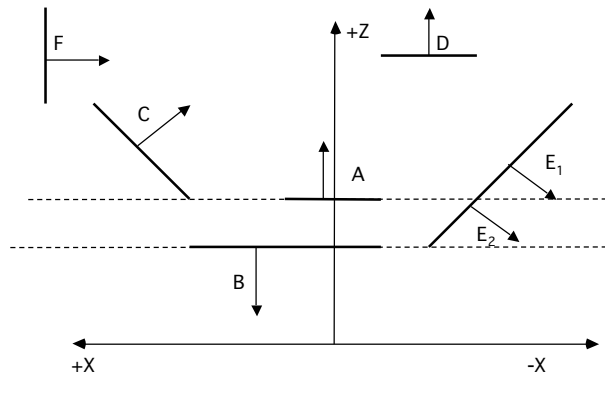
BSP Tree Example



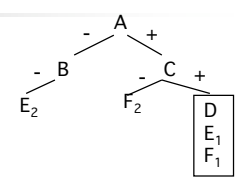
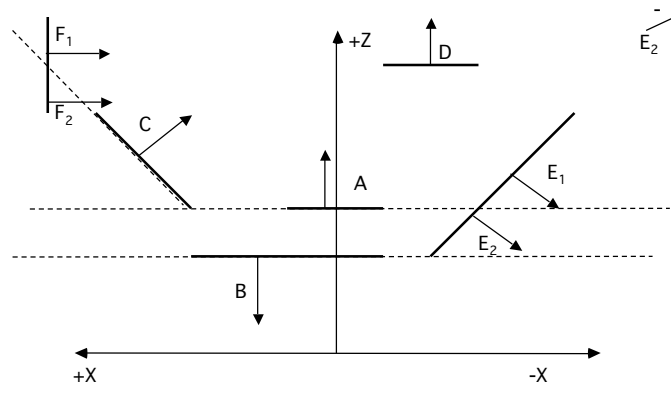
BSP Example (cont.)



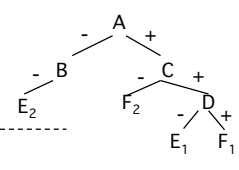
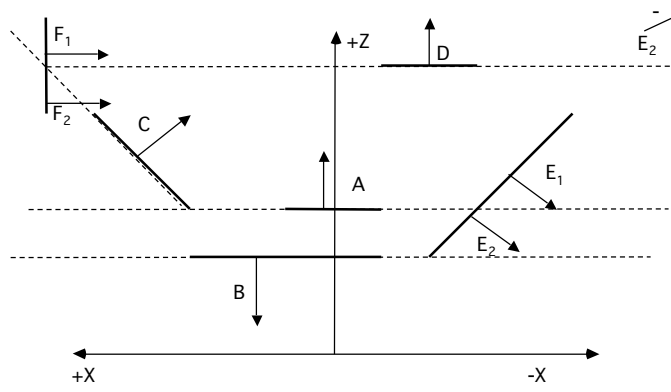
BSP Example (cont.)



BSP Example (cont.)



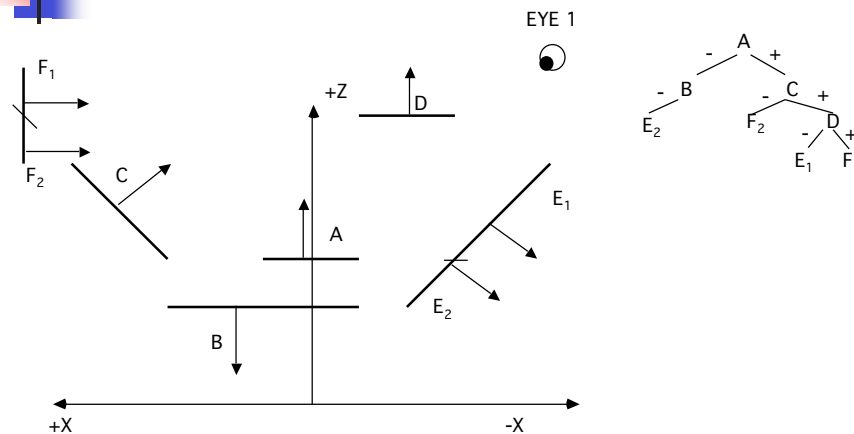
BSP Example (cont.)



Traversing the BSP-Tree

- Recursively:
 - First, descend the branch on 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.

Traversing the BSP Tree Example

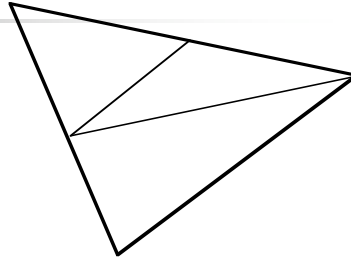


Eye 1: B, E₂, A, F₂, C, E₁, D, F₁



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.



Other HSE methods

- Image precision
 - Scan-line algorithms
- Object precision
 - Bounding volumes
 - Area subdivision / Spatial partitioning
 - Adaptive partitioning
 - Depth sort
 - View volume culling