Visible-surface detection methods

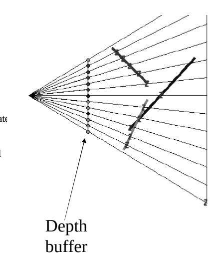
Chapter 9

Categorization

- Two categories
  - Image-space method
    - Work on the projected objects (onto the screen/framebuffer)
  - Object-space method
    - Work on the object it self
- Usually $n_{objects} << n_{pixels}$
- But the complexity in the tests also differs
- So Image-space is most common

Image based

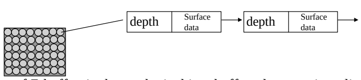
- The most common method is the Depth-Buffer Method (Z-Buffer)
- Algorithm
  - 1. initialize the depthBuffer to some value 1
  - 2. initialize the frameBuffer to backgroundcolor
  - 3. For each polygon in scene:
    - 3.1 For each projected (x,y) pixel in polygon, calculate
    - If z < depthBuffer(x,y), then depthBuffer(x,y)=z
      - frameBuffer(x,y) = color of the projected pixel



Z-Buffer

- Advantages
  - Primitives can be processed immediately (Immediate mode graphics API)
  - Well suited for HW, simple calculation per pixel
- Disadvantages
  - Visibility is coupled with sampling (Sampling = aliasing)
  - Excessive over-drawing, (the same pixel(x,y) can be accessed many times for a scene)

A-Buffer



- Extension of Z-buffer, in that each pixel in z-buffer, also contains a list of all overlapping pixels usually sorted in depth order
- Each position in the buffer can contain attributes of the surface covering the pixel:
  - Depth value,
  - Color
  - Transparency
  - Percent of area coverage
  - Surface identifier (so we can find the corresponding surface later)
- This can be used for transparency and anti-aliasing calculations.

Depth sorting

- Object space method
- Sorts surfaces in order of decreasing depth
- Surfaces are scan-converted starting with the surface of greatest depth.
- Refered to as painters algorithm
- You have all implemented it, its in the book, READ IT! (page 537-539)

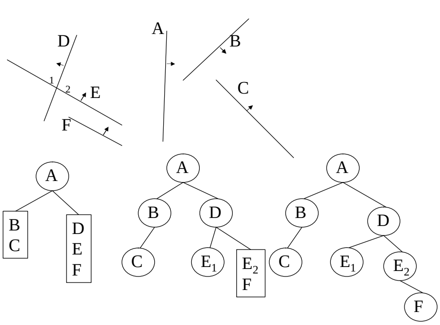
BSP-TREE

- Binary space partition tree
- Efficient when viewer moves, and objects are static
- We want to quickly determine the back to front relationship among the objects in the scene
- If we first have the green object, and then add the red, part of the green will be obscured. Therefore we cant draw the green after the red.

BSP tree

- An example of Object Space hidden surface algorithm
  – The tree is built as a preprocess, it is view independent
  – The tree is then during runtime quieried.
- All internal nodes has two children, representing front and back of the splitting line (plane in 3D)
- A 2D Example:
- Associated with each node *v* in the tree
  – A region *r(v)* and
  – A line (in 3D a plane) that intersects *r(v)*
  – A splitting plane l_n can be selected as a face of one polyhedra.
- Each internal root is defined by a splitting line (plane), dividing the space into infront of and behind the line (plane).
- Any object split by the line should be divided into separate objects.

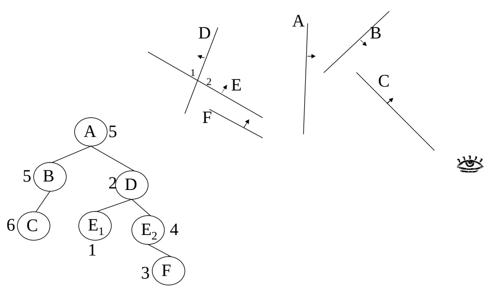
BSP Creation



BSP Traversal

- We want to render polygons in back to front order
- Inject the current viewpoint into the line (plane) equation of the root.
- Is it behind? Traverse the left tree. Otherwise select the right
- On the way back in the traversal, visit traversed nodes.

Traversal of a BSP



BSP Creation pseudo code

```
BSP_tree BSP_make(list_of_polygons plist)
   {
     if (EMPTY(plist))
       return NULL;
     else {
       root=select_and_remove_poly(plist);

       for each remaining polygon, p, in plist {
         if (p is on front of root)
           BSP_add_to_list(p, frontList)
         elseif (p is on back of root)
           BSP_add_to_list(p, backList)
         else {
           BSP_split_polygon(p, root, frontPart, backPart)
           BSP_add_to_list(frontPart, frontList)
           BSP_add_to_list(backPart, backList)
         }
       }
       return BSP_combine_tree(BSP_make(frontList),
                               root,
                               BSP_make(backList));
     }
   }
```

BSP Traversal Pseduo code

```
BSP_display(BSP_tree tree)
  {
    if (!EMPTY(tree)) {
      if (observer located on front of root) {
        BSP_display(tree->backChild);
        displayPolygon(tree->root);
        BSP_display(tree->frontChild);
      }
      else {
        BSP_display(tree->frontChild);
        displayPolygon(tree->root);
        BSP_display(tree->backChild);

      }

    }

  }
```