Lecture 8

More Hidden Surface Removal

Efficient Painter
- binary space partition (BSP) tree

Efficient Ray Casting
- spatial partitioning (uniform, octrees)
- bounding volumes
Recall last lecture ...
Also last lecture: Painter's Algorithm

Sort polygons in depth. Use the farthest vertex in each polygon as the sorting key.

Then draw polygons from "farthest" to "nearest" (back to front).

Split polygons when necessary.
More general problem: moving observer, static scene

We want to quickly find the back-to-front depth ordering AND avoid the problems of the Painter's algorithm
Recall: binary search tree (COMP 250, 251)

How to build?  How to use?
http://www.cim.mcgill.ca/~langer/250.html
Binary space partition (BSP) tree.

It is a binary tree.

Each node has a polygon $P$.  
(Number of nodes = number of polygons.)

Left descendents of $P$ are in front of $P$.

Right descendents of $P$ are in back of $P$. 
How to define/build a BSP tree?

Pick a polygon P. This is the root node of the tree.

Three cases for each remaining polygon:
1) is entirely "in front of" P
2) is entirely "in back of" P, i.e. behind P
3) intersects P's plane... in which case, split into two polygons which gives cases 1) and 2).
Example: pick \textcolor{blue}{a} for the root

\textcolor{blue}{b} intersects \textcolor{blue}{a}'s plane, so split \textcolor{blue}{b}.
\textcolor{blue}{c} is in front of \textcolor{blue}{a}. 

Convention:
Left subtree is in front.
Right subtree is in back.

b1 is in front of a.
b2 is in back of a.
c is in front of b1.
Space is partitioned into five regions.

How are they numbered?

Convention:
Left subtree is in front.
Right subtree is in back.
makeBSPtree( list of polygons ) {
    if list is empty
        return(NULL)
    else {
        select and remove a polygon \( P \) from list
        backlist := NULL
        frontlist := NULL
        for each polygon \( Q \) in list of polygons
            if all vertices of \( Q \) are in front of plane of \( P \)
                add \( Q \) to frontlist
            else if all vertices of \( Q \) are behind plane of \( P \)
                add \( Q \) to backlist
            else  //  plane \( P \) splits \( Q \)
                split \( Q \) into two polygons and add them to
                frontlist and backlist, respectively
        return   combine( makeBSPtree(frontlist),  \( P \),
                         makeBSPtree(backlist) )
    }
}
Use BSP tree to draw back-to-front

Traverse the BST tree doing depth-first-search, such that:

- draw every polygon (node).
- draw far surfaces before near surfaces. How?
displayBSPtree(root, viewer) {
    if (root != NULL) {
        if (viewer is on the front side of root plane) {
            displayBSPtree(backchild, viewer)
            drawPolygon(root)
            displayBSPtree(frontchild, viewer)
        }
        else {  // viewer is behind the root note
            displayBSPtree(frontchild, viewer)
            drawPolygon(root)  // back faced culled,
                                so not necessary
            displayBSPtree(backchild, viewer)
        }
    }
}
Q: What is the order of the leaves visited?

A: 2, 3, 4, 1, 5
Main advantage of BSP tree method (over Painter or Depth Buffer) ?

If scene is static, then we can precompute the BSP tree.

We can then quickly find back-to-front ordering from any viewer position.
Lecture 8

More Hidden Surface Removal

Efficient Painter
- binary space partition (BSP) tree

Efficient Ray Casting
- spatial partitioning (uniform, octrees)
- bounding volumes
Recall general ray casting

\[ p(t) = p_0 + (p_1 - p_0) t \]

\[ t_{\text{intersect}} = \infty \]
\[ p = \text{NULL} \]

for each polygon \{ \hspace{1cm} // \text{inefficient} \}

if (ray intersects the polygon and \( t_{\text{intersect}} < t_{\text{min}} \))
\[ t_{\text{min}} = t_{\text{intersect}} \]
\[ p = \text{polygon} \]
For each spatial cell, maintain a list of objects that intersect it.

Each object may intersect multiple cells.
Ray casting

Examine only cells that the ray intersects.
t = infinity
p = NULL

current_voxel = voxel containing the starting point of ray
while (t == infinity)
    for each surface in current voxel {
        t_intersect = distance to surface along ray
        // infinite if no intersection
        if (t_intersect < t) {
            t = t_intersect
            p = surface
        }
    }
    current_voxel = next voxel hit by the ray
t = infinity
p = NULL

current_voxel = voxel containing the starting point of ray
while (t == infinity)
    for each surface in current voxel {
        t_intersect = distance to surface along ray
        // infinite if no intersection

        if (t_intersect < t) and
            (intersection point belongs to current voxel) {
            t = t_intersect
            p = surface
        }
    }

current_voxel = next voxel hit by the ray
ASIDE: In the lecture, I doubted the stopping condition of the algorithm. But everything was fine. The "while loop" condition is that $t = \infty$. As soon as you find an intersection point that is within the current voxel, $t$ will get assigned a finite value and the algorithm will stop.
Using a coarser grid means there are typically more surfaces per voxel (bad), but fewer voxels (good).
Non-uniform spatial partition

2D - "quadtree"
Again, for each spatial cell, maintain a list of objects that intersect it.
The same ray casting algorithm works fine. But we need to specify how to compute next voxel. (Not obvious -- Exercise.)
3D - "octree"
3D - "octree"
Octrees can be an unstable representation when surfaces move e.g. animation.

e.g. what happens when the red surface moves to the right?
Lecture 8

More Hidden Surface Removal

Efficient Painter
- binary space partition (BSP) tree

Efficient Ray Casting
- spatial partitioning (uniform, octrees)
- bounding volumes
Bounding Volumes

Does the ray intersect the chair?
Bounding Volumes

IF the ray intersects the chair,
THEN it intersects the bounding volume.
IF the ray intersects the chair, THEN it intersects the bounding volume.

[Second statement is false]

IF the ray intersects the bounding volume THEN the ray intersects the chair.
IF the ray intersects the chair,
THEN it intersects the bounding volume.

\[ \sim \]

IF the ray doesn't intersect the bounding volume
THEN the ray doesn't intersect the chair.
Note the analogy to Cohen Sutherland line clipping.

A quick test can be used for trivial rejection.
Bounding Volume Hierarchy

2nd level shown

back

legs

seat
Internal nodes represent bounding volumes
Leaves represent bounding volume of a single polygon.

Q: What is the BV relationship between child and parent?
A: The child's BV is contained in the parent's BV.

Q: What is the BV relationship between siblings?
A: none
To cast a ray and find the closest object, we traverse the bounding volume hierarchy tree of the whole scene.

Use depth first search.

Q: What are we searching for?

Q: What does it mean to visit a node?
\[ p = \text{NULL} \quad // \text{pointer to polygon} \]
\[ t = \text{infinity} \quad // \text{closest point on ray} \]

\[
\text{void traverseBVH( ray, node)}\{
    \text{intersect ray with node's bounding volume}
    \text{if } 0 \leq t\_intersect < t \quad \{
        \text{if (node is a leaf) }
        \quad \text{compute } t\_intersect \\
        \quad \text{if } (0 \leq t\_intersect < t) \\
        \quad \quad \text{update } p \text{ and } t \\
        \text{else } // \text{node is a bounding volume}
        \quad \text{for each child of node} \\
        \quad \text{traverseBVH( ray, child)}
    \}\}
\}
Q: How to make this more efficient?

A: "for each child of node" loop should test closest child nodes first. (See Exercises.)
Reminder:

A1 is due Monday at 11:59 PM.