Hidden Surface Elimination: BSP trees

Outline

- Binary space partition (BSP) trees
  - Polygon-aligned
BSP Trees

- Basic idea: Preprocess geometric primitives in scene to build a spatial data structure such that tests from any viewpoint can be easily calculated later.
- Examples of tests:
  - **Visibility** for painter's algorithms
  - **Intersection testing** for ray tracing
- Generalization of binary search trees (1-D) to higher dimensions

---

**Painter’s algorithm**

Draw primitives from back to front to avoid need for depth comparisons.
BSP trees: Key property

- “Spatial sorting” keeps track of which side of lines/planes primitives are on
  - Objects on the same side as the viewer can be drawn on top of objects on the opposite side
  - Objects on one side cannot intersect objects on the other side

- “Polygon-aligned” means partitioning plane is always coplanar with a scene polygon, as opposed to arbitrarily positioned

Building 2-D “line-aligned” BSP trees

- Pick oriented line segment (i.e., has a normal) from list as the root
- Rest of lines partitioned according to which side they are on
  - “Partitioning” line placed at root of subtree
  - Sets of lines on “front” side and “back” side correspond to left & right subtrees, respectively
- Recurse on each child
Building 2-D BSP trees: Issues

• How to pick line with which to partition
• What to do with lines that cross partitioning line
  – Split them (standard)
  – Or: Put a copy on each side of boundary
• When to stop recursing
  – $n$ or fewer primitives per leaf ($n = 1$ is standard)
  – Or: Threshold on recursion depth

2-D BSP tree: Building example
2-D BSP tree: Building example

2-D BSP tree: Building example

from Foley et al.
Building 2-D BSP trees: Pseudocode

```c
BSP_tree *BSP_makeTree(line *lineList)
{

    if (lineList == NULL)
        return NULL;
    else {
        root = BSP_selectAndRemoveLine(lineList);
        backList = frontList = NULL;
        for (each line l in lineList) {
            if (l in front of root)
                BSP_addToList(l, frontList);
            else if (l in back of root)
                BSP_addToList(l, backList);
            else {
                BSP_splitLine(l, root, frontPart, backPart);
                BSP_addToList(frontPart, frontList);
                BSP_addToList(backPart, backList);
            }
        }
        return BSP_mergeTree(BSP_makeTree(frontList), root, BSP_makeTree(backList));
    }
}
```

Picking partitioning lines: Criteria

- **Painter’s algorithm**
  - Every object must be drawn → Entire tree is traversed
  - So overall tree size should be as small as possible
  - Minimize splitting
- **Ray tracing** (later in course...)
  - Several paths from root to leaves traversed looking for intersections
  - So tree depth more important than overall size
  - Balance primitives on each side
Partitioning lines: “Least-crossed” heuristic

• For painter’s algorithm, we want to choose partitioning lines that minimize the number of splits
• A particular line can be tested to see how many lines cross it and therefore would have to be split if it were the partitioning line
• Good procedure in practice:
  1. Randomly select a small number of candidate partitioning lines (e.g., 5-10 out of 1,000)
  2. Calculate number of lines that cross each candidate
  3. Use candidate with least crossings as next partition

Building 2-D BSP trees: Details

• How to parametrize partitioning line \( \mathbf{l} \) from line segment \( \mathbf{p}_1 \mathbf{p}_2 \)?
  - Homogeneous line form \( \mathbf{l} = (a, b, c)^T \) for segment is same as line it’s on
  - We can obtain \( \mathbf{l} \) from \( \mathbf{p}_1, \mathbf{p}_2 \) via \( \mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2 \)
  - Normal vector of line \( \mathbf{l} \) is \( \mathbf{n} = (a, b)^T \)

• How to decide which side of partitioning line \( \mathbf{l} \) a line segment \( \mathbf{v}_1 \mathbf{v}_2 \) is on?
  - If both points \( \mathbf{v}_1, \mathbf{v}_2 \) are on the front side of \( \mathbf{l} \), line \( \mathbf{v}_1 \mathbf{v}_2 \) is on front side
  - If both \( \mathbf{v}_1, \mathbf{v}_2 \) are on the back side, \( \mathbf{v}_1 \mathbf{v}_2 \) is on back side
  - If \( \mathbf{v}_1, \mathbf{v}_2 \) are on different sides, \( \mathbf{v}_1 \mathbf{v}_2 \) crosses partitioning line \( \mathbf{l} \)
Building 2-D BSP trees: Details

- How to split crossing lines?
  - Find intersection point $\mathbf{x}$ of $\mathbf{v}_1\mathbf{v}_2$ with $\mathbf{l}$
    - Let $\mathbf{l}'$ be homogeneous form of line defined by $\mathbf{v}_1\mathbf{v}_2$
    - By definition, we want a point $\mathbf{x}$ that is on both lines $\mathbf{l}$ and $\mathbf{l}'$. This would imply that $\mathbf{l} \cdot \mathbf{x} = \mathbf{l}' \cdot \mathbf{x} = 0$
    - Just looking at these as vectors, a dot product of 0 means that $\mathbf{x}$ is orthogonal to both $\mathbf{l}$ and $\mathbf{l}'$
    - Because cross product is orthogonal to both multiplicands, $\mathbf{x} = \mathbf{l} \times \mathbf{l}'$ satisfies this requirement and thus defines the point of intersection
  - Given intersection $\mathbf{x}$:
    - If $\mathbf{v}_1$ is on front side of $\mathbf{l}$: Output $\mathbf{v}_1 \mathbf{x}$ as front part and $\mathbf{x}\mathbf{v}_2$ as back part
    - If $\mathbf{v}_2$ is on front side of $\mathbf{l}$: Output $\mathbf{x}\mathbf{v}_2$ as front part and $\mathbf{v}_1 \mathbf{x}$ as back part

Painter’s algorithm: Tree traversal

- Want farthest-to-nearest ordering of primitives for painter’s algorithm
  - If view location is on front side of a partitioning line:
    - Lines on back side are farther
    - Lines on front side are nearer
  - If view location is on back side of a partitioning line:
    - Lines on front side are farther
    - Lines on back side are nearer

- Which side of a partitioning line $\mathbf{l}$ is a point $\mathbf{p}$ on?
  - Assuming $\mathbf{l}, \mathbf{p}$ in homogeneous form, use homogeneous line test $F(\mathbf{p}) = \mathbf{l} \cdot \mathbf{p}$
    - $F > 0 \Rightarrow \mathbf{p}$ is on back side of $\mathbf{l}$
    - $F < 0 \Rightarrow \mathbf{p}$ is on front side of $\mathbf{l}$
    - $F = 0 \Rightarrow \mathbf{p}$ is on line $\mathbf{l}$ (arbitrarily treat as back side)
void BSP_displayTree(BSP_tree *tree, point viewLocation) {
    if (tree != NULL) {
        if (viewLocation is in front of tree->root) {
            // Display back child, root, then front child
            BSP_displayTree(tree->backChild, viewLocation);
            displayLine(tree->root);
            BSP_displayTree(tree->frontChild, viewLocation);
        } else {
            // Display front child, root, then back child
            BSP_displayTree(tree->frontChild, viewLocation);
            displayLine(tree->root); // back-facing line—can cull by skipping
            BSP_displayTree(tree->backChild, viewLocation);
        }
    }
}

**Painter’s algorithm: Example BSP tree traversal**

Behind root (node 3), so display everything in front of (left subtree = nodes 1, 2, 5a), then root (node 3), then everything behind (right subtree = nodes 4 and 5b)
In front of root (node 2), so display everything behind (right subtree = node 1), then root (node 2), then everything in front of (left subtree = node 5a)
Behind root (node 4), so display everything in front of (left subtree = NULL), then root (node 4), then everything behind (right subtree = node 5b)
Painter’s algorithm: Example BSP tree traversal

**Final order: 1, 2, 5a, 3, 4, 5b**

Every node is visited from back-to-front, so this is an $O(n)$ operation ($n$ is the number of primitives after splitting)

3-D BSP Trees

- Analog of 2-D method, but now we are talking about 3-D polygon primitives and partitioning planes
- What’s different about this vs. lines?
  - Must parametrize plane from polygon
  - Point-plane “sidedness” test is analogous to point-line test
    - Just use homogeneous form of plane equation
  - Line (each edge of polygon)-plane intersection instead of line-line intersection
  - Polygon splitting instead of line splitting
3-D BSP Trees: Details

- Parametrize plane from polygon
  - Cross product of edges gets plane normal \( \mathbf{n} = (a, b, c)^T \)
  - Solve for \( d \) from single point on plane \((x, y, z)^T\) via plane equation \(ax + by + cz + d = 0\)

- Line-plane intersection
  - Unfortunately, there is no simple homogeneous form for lines in 3-D like in 2-D
  - Instead:
    - Express point \( \mathbf{p} \) on a ray as some distance \( t \) along direction \( \mathbf{d} \) from origin \( \mathbf{o} \): \( \mathbf{p} = \mathbf{o} + t\mathbf{d} \)
    - Use plane equation \( \mathbf{n} \cdot \mathbf{x} + d = 0 \), substitute \( \mathbf{o} + t\mathbf{d} \) for \( \mathbf{x} \), and solve for \( t \)
    - Then plug \( t \) back into \( \mathbf{p} = \mathbf{o} + t\mathbf{d} \) to get \( \mathbf{p} \)

BSP Trees: Notes

- Works best for static scenes
  - Moving primitives can cross partitioning lines
  - Dynamic adjustment of tree possible, but slows things down