CSE168
Computer Graphics II, Rendering

Spring 2006
Matthias Zwicker
Practical details

• Assignment 1 due today
• Problems with base code?

• Assignment 2 available on the web site
• Due May 1, 5pm
Last time

- Aggregate objects
- Acceleration structures
- Bounding volume hierarchies
- Axis aligned bounding boxes
Aggregate objects

- Object that holds groups of objects
- Stores bounding box and pointers to children
- Useful for instancing and BVH
Ray tracing cost

- Naïve approach: linear complexity per ray
- 6320 triangles * 1024 * 768 rays
Acceleration structures

- Goal: “sub-linear” complexity per ray
- Don’t touch every single object
Acceleration structures

Two fundamental approaches

• Hierarchies of groups of objects
  “object subdivision”

• Space partitioning
  “spatial subdivision”
Bounding volume hierarchies

- Hierarchies of groups of objects
Bounding volume hierarchies

- Hierarchies of groups of objects
Bounding volume hierarchies

- Hierarchies of groups of objects
Bounding volume hierarchies

- Tree structure
- Internal nodes are aggregate objects
- Leave nodes are geometric objects
- Intersection testing involves tree traversal

aggregate object
BVH construction
BVH construction

• Root
BVH construction

- Split horizontally
BVH construction

• Recursion
BVH construction

- Split vertically
BVH construction

• Split vertically

Etc....
Axis aligned bounding boxes

\[ t \in [t_{x\text{min}}, t_{x\text{max}}] \]
\[ t \in [t_{y\text{min}}, t_{y\text{max}}] \]
\[ t \in [t_{y\text{min}}, t_{y\text{max}}] \cap [t_{y\text{min}}, t_{y\text{max}}] \]
Today

- Binary space partitioning (BSP) trees
- Construction
- Optimization
- Traversal
Space partitioning

- Uniform grid
Space partitioning

• Uniform grid
Space partitioning

- Uniform grid
Space partitioning

- Uniform grid
BSP trees

- Binary space partitioning trees
- Recursively divide space into two parts
- Kd trees
  - Dividing planes are axis aligned
BSP trees

Example

• Subdivide until fewer than 3 objects in node
• Left child below split plane
• Right child above split plane
BSP trees
BSP trees
BSP trees
BSP trees
BSP trees
BSP trees
BSP trees
BSP trees
BSP trees
BSP trees
BSP trees
BSP trees

Diagram showing a BSP tree with corresponding regions and nodes labeled.
BSP tree data structure

BSP_tree : object {
    intersect( ray, &hit )
    build_tree( object *o[], bbox b )
    BSP_node *root
}

BSP_node {
    float plane_pos
    int axis
    BSP_node *above, *below
    bool is_leaf
    object *obj_array
}
BSP tree construction

```cpp
BSP_node BSP_tree::build_tree( object *o[], bbox b ) {
    if( termination criterion met ) {
        make leaf
        return leaf
    }
    find split plane
    make node
    b_above = bbox above of plane
    b_below = bbox below of plane
    o_above = objects above of plane
    o_below = objects below of plane
    node.above = build_tree( o_above, b_above )
    node.below = build_tree( o_below, b_below )
    return node
}
```
BSP tree optimization

- Where to place the split plane?
- Locally minimize cost function

Cost for no split

\[ \sum_{i=1}^{N} t_i \]

\( N \)  Number of primitives

\( t_i \)  Cost to intersect primitive \( i \)
BSP tree optimization

Cost for split

- Depends on split plane position

\[ t_t + p_B \sum_{i=1}^{N_b} t_i(b_i) + p_A \sum_{i=1}^{N_a} t_i(a_i) \]

- \( t_t \) Cost to traverse interior node
- \( p_B, p_A \) Probability to hit child below, above split
- \( N_b, N_a \) Number of primitives below, above split
BSP tree optimization

- Probability to hit child

\[ p_A = P(A|\text{root}) = \frac{S_A}{S_{\text{root}}} \]

\[ S_A, S_{\text{root}} \quad \text{Surface area of child, root} \]
BSP tree optimization

Minimizing the cost function

• Evaluate at small number of fix locations
• Advanced: evaluate at edges of object bounding boxes
• For all 3 split directions
BSP tree optimization

Tweaking implementation

• Determine parameters experimentally
  \( t_i, t_t \)

• Expect \( t_i \approx 8t_t \)

• Favor splits where one child is empty

• Allow a number of bad splits
BSP tree optimization

Tweaking your implementation

• Optimize memory layout
• BSP nodes
  - Reduce size to 8 bytes
• Tree organization
  - Store left child always immediately after parent node
BSP tree traversal

- “Front-to-back” traversal
- Traverse child nodes in order along rays
- Stop traversing as soon as surface intersection is found
- Maintain a stack of subtrees to traverse
  - More efficient than recursive function calls
BSP tree traversal
BSP tree traversal
BSP tree traversal
BSP tree traversal

Process 1,2
Stack B 3,6

Diagram showing BSP tree traversal with nodes A, B, C, D, and E, and process numbers 1, 2, 3, 4, 5, 6, 7, 8, and 9.
BSP tree traversal

![BSP tree diagram]

- **Process Stack**: 3, 6
- **Stack**: B

BSP tree traversal process with elements 1, 2, 3, 4, 5, 6, 7, 8, 9.
BSP tree traversal
BSP tree traversal

A
    D       B
      7,8    3,4
    / \    /  \  
   E   C   1,2 6 9
    \   /  5,6
     \ /   
      8    
       /  
      6   
       /  
      3   
       /  
     4   
      /  
     2   
      /  
    1

Process
3,4

Stack
C
BSP tree traversal
BSP tree traversal
BSP tree traversal
BSP tree traversal

BSP_stack_item {
    BSP_node *node
    float tmin, tmax
}

tmin
tmax
node.axis
node.plane_pos
BSP tree traversal

node = root
isect = FLT_MAX
tmin, tmax = intersect root bounding box
while( node!=0 ) {
    if( isect < tmin ) break
    if( node is not leaf ) {
        <process interior node>
    } else {
        isect = check for intersection inside leaf
        node, tmin, tmax = get item from stack
    }
}
BSP tree traversal

\[
\text{<process interior node>} = \\
\quad \text{compute ray-split plane intersection} \\
\text{<order of children>} \\
\text{<process children>}
\]

- Which child node is (potentially) hit first?
- Which child nodes are hit?
BSP tree traversal

Which child is (potentially) hit first?

```cpp
<order children> = if( e[axis] < plane_pos ) {
    first = below
    second = above
} else {
    first = above
    second = below
}
```
BSP tree traversal

Which child nodes are hit?

\[<\text{process children}>=\]
\[
\begin{array}{l}
\text{if( tsplit} > \text{tmax || tsplit} < 0) \{ \\
\quad \text{node} = \text{first} \\
\}
\end{array}
\]
BSP tree traversal

Which child nodes are hit?

```
<process children>+=
else if( tsplit< tmin ) {
    node = second
}
```
BSP tree traversal

Otherwise process first child, stack second child

\[
\text{else} \{ \\
\quad \text{node} = \text{first} \\
\quad \text{item.node} = \text{second} \\
\quad \text{item.tmin} = \text{tsplit} \\
\quad \text{item.tmax} = \text{tmax} \\
\quad \text{stack.add( item )} \\
\}
\]
BSP tree traversal

Which intersection is found first?
BSP tree traversal

Which intersection is found first?

while( node!=0 ) {
    if( isect < tmin ) break
...
BSP tree traversal

Mailboxing

- Avoid intersecting the same object multiple times
- Use unique ID tag for each ray
- Tag each object when intersecting with ray
Organizational

- Get assignment 2 from course web site!
- Implement BSP tree acceleration