Practical details

- Assignment 1 done
- Assignment 2 ready (due May 8th)

Today's Menu

- Slabs
- BSP trees
- Cost functions

Last Time

- BVH
- Assignment 2

Bounding volume hierarchies: BVH

- Bounding spheres
- Bounding boxes
- BVH traversal

Sphereflake
Bounding Box Intersection

\[
\begin{align*}
t_x^1 &= \frac{x_{\text{min}} - d_x}{d_x} & t_x^2 &= \frac{x_{\text{max}} - d_x}{d_x} \\
t_y^1 &= \frac{y_{\text{min}} - d_y}{d_y} & t_y^2 &= \frac{y_{\text{max}} - d_y}{d_y} \\
t_z^1 &= \frac{z_{\text{min}} - d_z}{d_z} & t_z^2 &= \frac{z_{\text{max}} - d_z}{d_z}
\end{align*}
\]

Hit if \( t_{\text{min}} < t_{\text{max}} \)

BVH

- Building the BVH
- Intersection

Basic BVH Intersection

```c
void intersect_bvh( ray, &hit ) {
    if ( bounding_box.hit( ray ) ) {
        if (leaf)
            leaf.intersect( ray, &hit );
        else
            for (each child box)
                intersect_bvh( ray, &hit );
    }
}
```

Building a BVH

- Bottom-up merging using a cost function
- Top-down splitting of triangle meshes

Cost Function

Cost for 1 BV:

\[
C = C_V + P_1 \cdot N_1 \cdot C_o
\]

Cost for 2 BV’s:

\[
C = C_1V + P1V \cdot N1o \cdot C_o + C2V + P2v \cdot N2o \cdot C_o
\]
**Many Triangles**

No accel: 6320 intersections / ray  
Using BVH: 2.6 intersections / ray

---

**Slabs**

- [Kay and Kajiya, "Ray Tracing Complex Scenes", SIGGRAPH 1986]  
- Generalization of bounding box concept  
- Can fit any convex object perfectly

---

**BSP Trees**

- What is it?  
- How to build it  
- How to ray trace it

---

**BSP Tree Node**

```c
struct BSP_node {
    float plane_pos;
    int axis;
    BSP_node *left, *right;
    bool is_leaf;
    Object *obj_array;
};
```

---

**Better BSP Tree Node**

```c
struct BSP_node {
    float plane_pos;
    BSP_node *left;
};
```
BSP Tree Construction

```c
subdivide( node ) {
    split along axis
    oleft = objects on left side
    setup left node
    subdivide( left node );
    oleft = objects on right side
    setup right node
    subdivide( right node );
}
```

BSP Tree Traversal

```c
Intersect root box
Compute [t_min, t_max]
Traverse tree checking nearest node first
Keep far nodes on stack
```

BSP Tree Traversal

```c
intersect_bsp() {
    [t_min, t_max] = intersect bounding box
    intersect_node( root, t_min, t_max )
}
```

BSP Tree Traversal

```c
intersect_node( node, t_min, t_max ) {
    if (node->type == leaf) {
        t = intersect node->objects
        if (t < t_max)
            done
    } else {
        if (ray.direction[ node->axis ] > 0)
            near = node->left; far = node->right
        else
            near = node->right; far = node->left;
        t = intersect node->plane
        if (t > t_max )
            intersect_node( near, t_min, t_max )
        else if (t < t_min) 
            intersect_node( far, t_min, t_max )
        else
            intersect_node( near, t_min, t )
            intersect_node( far, t, t_max )
    }
}
```

Results

<table>
<thead>
<tr>
<th></th>
<th>No BSP</th>
<th>BSP l=8, mo=10</th>
<th>BSP l=16, mo=8</th>
<th>BSP l=24, mo=8</th>
<th>BSP l=24, mo=4</th>
<th>BSP l=32, mo=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>$O(\log n)$</td>
<td>9,986,402,697</td>
<td>111,204,795</td>
<td>9,930,604</td>
<td>6,350,655</td>
<td>4,426,580</td>
</tr>
</tbody>
</table>
Better BSP Trees

- Faster implementation
- Better tree
  - Splitting dimension
  - Splitting plane location

Ad Hoc Heuristics

- Same number of objects on both sides
- Use object bounding box
- ...

Surface Area Heuristic

\[ C = \frac{C_n \cdot \sum_n SA_n + C_l \cdot \sum_l SA_l + C_o \cdot \sum_l SA_l \cdot N_l}{SA_{root}} \]

[MacDonald and Booth, 1990]

Next time

- More acceleration structures