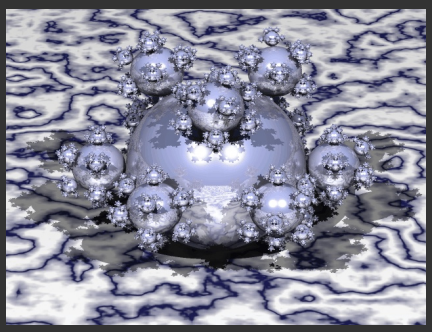
CSE168: Rendering Algorithms Acceleration Structures 2



Henrik Wann Jensen
henrik@cs.ucsd.edu

Practical details

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

Todays Menu

- Slabs
- BSP trees
- Cost functions

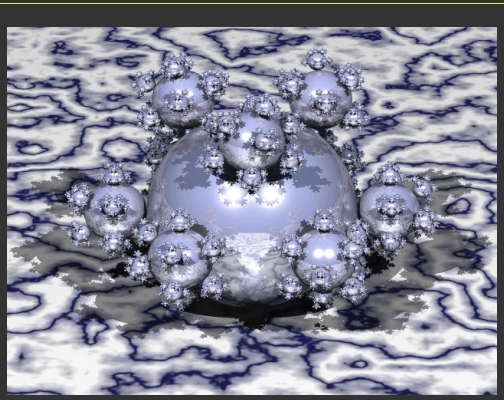
Last Time

- BVH
- Assignment 2

Last Time

- Bounding volume hierarchies: BVH
  - ⋆ Bounding spheres
  - ⋆ Bounding boxes
- BVH traversal

Sphereflake



Bounding Box Intersection

$$t_{x1} = \frac{x_{min} - \vec{o}_x}{\vec{d}_x} \quad t_{x2} = \frac{x_{max} - \vec{o}_x}{\vec{d}_x}$$
$$t_{y1} = \frac{y_{min} - \vec{o}_y}{\vec{d}_y} \quad t_{y2} = \frac{y_{max} - \vec{o}_y}{\vec{d}_y}$$
$$t_{z1} = \frac{z_{min} - \vec{o}_z}{\vec{d}_z} \quad t_{z2} = \frac{z_{max} - \vec{o}_z}{\vec{d}_z}$$

Hit if $t_{min} < t_{max}$

BVH

- Building the BVH
- Intersection

Basic BVH Intersection

```
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

Strategies

- Manual
- Heuristics
- Automatic

Building a BVH

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

Cost Function

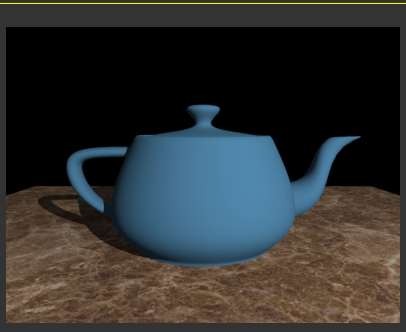
Cost for 1 BV:

$$C = C_V + P_v * N_o * C_o$$

Cost for 2 BV's:

$$C = C1_V + P1_v * N1_o * C_o + C2_V + P2_v * N2_o * C_o$$

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 Trees

- BSP = Binary Space Partitioning

BSP Tree Node

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


Better BSP Tree Node

```
struct BSP_node {
  float plane_pos;
  BSP_node *left;
}
```

BSP Tree Construction

```
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

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

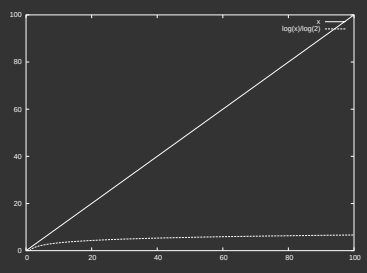

BSP Tree Traversal

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


BSP Tree Traversal

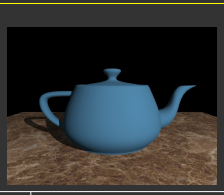
```
intersect_node( node, t_min, t_max )
{
  if (node->type == leaf) {
    t = intersect node->objects
    if (t < t_max)
      done
  } else {
    if (ray.direction[ node->axis ] > 0)
      near = node->left; far = node->right
    else
      near = node->right; far = node->left;

    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_max )
      intersect_node( far, t, t_max )
    }
  }
}
```


BSP Tree Traversal



Cost = $O(\log n)$

Results



No BSP	9,986,402,697	6320 / ray
BSP l=8, mo=10	111,204,795	70.38 / ray
BSP l=16, mo=8	11,361,140	7.19 / ray
BSP l=24, mo=8	9,930,604	6.28 / ray
BSP l=24, mo=4	6,350,655	4.02 / ray
BSP l=32, mo=2	4,426,580	2.80 / ray

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

$$C = \frac{C_n * \sum_n SA_n + C_l * \sum_l SA_l + C_o * \sum_l SA_l * N_l}{SA_{root}}$$

[MacDonald and Booth, 1990]

Next time

- More acceleration structures