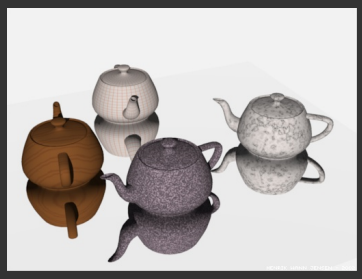
CSE168: Rendering Algorithms Acceleration Structures 1



Henrik Wann Jensen
henrik@cs.ucsd.edu

Practical details

- Assignment 1 is due tonight
- Ray tracing document now available

Todays Menu

- Triangle meshes
- BVH building
- Slabs
- Basic cost functions
- Assignment 2

Specular Teapot

Diffuse Teapot

Glass Teapot

Basic Ray Tracing

```
render()
  for each pixel p
    create primary ray through p
    trace( color, ray )
```


Basic Ray Tracing

```
trace( color, ray ) {
  for each object
    intersect object
    if hit and closest intersection
      remember hit
  if (hit)
    shade( color, hit )
```


Basic Ray Tracing

```
shade( color, hit ) {
  for all lights
    trace shadow rays
    compute diffuse and specular shading
  trace specular and refracted rays
  return color;
```


The Cost

Cost = Objects * Rays

Example: 1024x1024 image of scene with 1000 triangles

Cost is (at least) 10^9 ray-triangle intersections

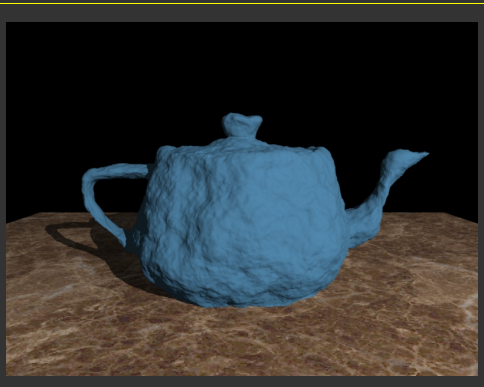
The Complexity

Measured per ray:

$$O(n) = n$$

The complexity of naive ray tracing is linear with respect to the number of objects. This is the same cost as rasterization algorithms.

The problem



Cost $\approx 10^{12}$

The Solution

Acceleration structures: A spatial organization of the objects (triangles) in the scene in order to minimize the necessary number of ray-object intersection tests.

The Strategies

- Bounding volumes
- Spatial sorting

Triangle Meshes

How to manage many triangles..

One Triangle

Basic representation:

```
struct triangle {
  Vector3f v1, v2, v3;   // vertices
  Vector3f n1, n2, n3;   // normals (optional)
  float u1,u2,u3;        // texture (optional)
  float v1,v2,v3;        // texture (optional)
};
```


One Triangle

Alternative representation;

```
struct triangle {
  int v1, v2, v3;
};

struct vertex {
  vector3f pos;
  vector3f normal;
  float u,v;
};
```


Many Triangles

One (bad) strategy (a list of triangles):

```
struct triangle {
  int v1, v2, v3;
  struct triangle *next;
};

struct vertex {
  vector3f pos;
  vector3f normal;
  float u,v;
};
```

Many Triangles

Use arrays:

```
// N is number of triangles
struct triangle tri[ N ];

// NV is number of vertices
struct vertex vert[ NV];
```

Compact and easy to index.

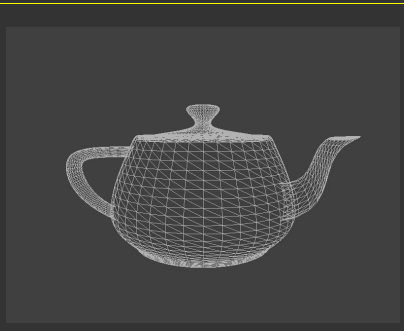
OBJ Format

```
v
v
vt
vt
vn
vn
f
usemtl
```


Many Triangles

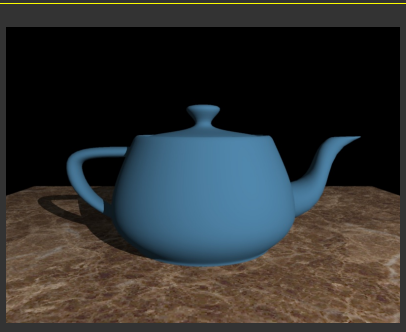
Can save a few extra bytes by using specialized alternatives such as triangle strips.

Many Triangles



This teapot is 6320 triangles

Many Triangles



6320 triangles per ray will take a long time to render

Ray Tracing Triangle Meshes

Bounding Volumes

- Use a simple object to bound the mesh
  - Spheres
  - Boxes

Bounding Spheres

```
struct bsphere {
  vector3f pos
  float radius
  object *complex_object
};
```

Fast ray-sphere intersection (only check if hit)

Bounding Boxes

```
struct bbox {
  vector3f min
  vector3f max
  object *complex_object
};
```


Ray-Box Intersection

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 $t_{min} < t_{max}$

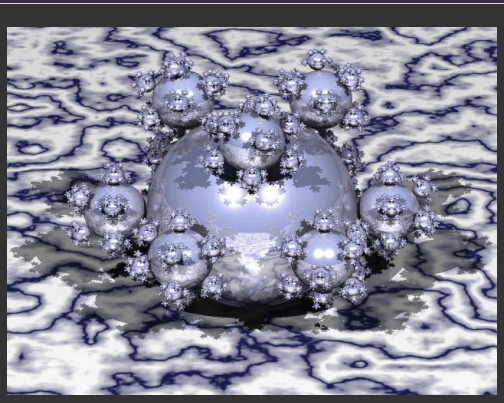
Bounding Volume Hierarchies

Idea: Break complex object into smaller pieces

Use a hierarchy of simpler volumes to bound these pieces and use volumes to bound groups of volumes.

For example a hierarchy of spheres or a boxes with triangles at the leaf nodes.

Sphereflake



Bounding Volume Hierarchies

- Building the BVH
- Intersection

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:

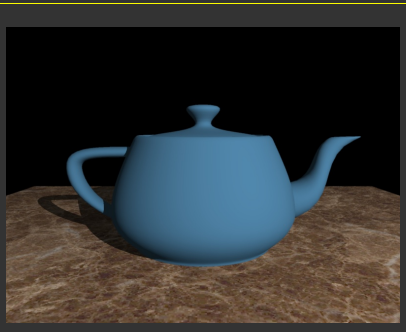
$$C = C1_V + P1_v * N1_o * C_o + C2_V + P2_v * N2_o * C_o$$

Basic 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 );
  }
}
```


BVH Efficiency



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

Next time

More acceleration structures.....