Practical details

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

Todays Menu

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

Diffuse Teapot

Glass Teapot
Basic Ray Tracing

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

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

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

Basic Ray Tracing

The Cost

```
Cost = Objects * Rays
Example: 1024x1024 image of scene with 1000 triangles
Cost is (at least) \(10^{12}\) ray-triangle intersections
```

The Complexity

```
Measured per ray:

\[ O(n) = n \]
```

The problem

Cost \(\approx 10^{12}\)

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

```c
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:

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

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

Many Triangles

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

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

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

Use arrays:

```c
// 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
t
vt
t
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

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

Fast ray-sphere intersection (only check if hit)

Bounding Boxes

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

Ray-Box Intersection

Bounding Box Intersection

\[
\begin{align*}
    t_x_1 &= \frac{\text{min}_x - \vec{o}_x}{\vec{d}_x} \\
    t_x_2 &= \frac{\text{max}_x - \vec{o}_x}{\vec{d}_x} \\
    t_y_1 &= \frac{\text{min}_y - \vec{o}_y}{\vec{d}_y} \\
    t_y_2 &= \frac{\text{max}_y - \vec{o}_y}{\vec{d}_y} \\
    t_z_1 &= \frac{\text{min}_z - \vec{o}_z}{\vec{d}_z} \\
    t_z_2 &= \frac{\text{max}_z - \vec{o}_z}{\vec{d}_z}
\end{align*}
\]

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

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.
Building a BVH

Strategies
- Manual
- Heuristics
- Automatic

Cost Function

Cost for 1 BV:
\[ C = C_V + P_v \times N_o \times C_o \]

Cost for 2 BV’s:
\[ C = C_1V + P_1v \times N_1o \times C_o + C_2V + P_2v \times N_2o \times C_o \]
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 );
    }
}
```

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......