CSE168
Computer Graphics II, Rendering

Spring 2006
Matthias Zwicker
Practical details

- Assignment 1 due Monday April 17, noon
- Lab hours Thursday 11am - 12 30pm
- Office hours Friday 3pm - 4pm, EBU3B 4114
- Handout of assignment 2 in class Monday
Last time

- Reflection
- Refraction
- Shadows
Mirror reflection

\[ L(x, d) = R_s L_i(x, r) \]

\[ r = d - 2(d \cdot n)n \]
Refraction

- At interface between dielectric materials (insulators)
- Diamond, glass, water, air
- Light travels at different speeds
- Light is bent at the interface
Refraction

• Fermat’s principle
  “The actual path between two points taken by a beam of light is the one which is traversed in the least time. “

• Snell’s law

\[ n \sin \theta = n_t \sin \phi \]
Dispersion

- Index of refraction varies with wavelength
Specular refraction

How much light is reflected and refracted? Ideal dielectric, smooth surface
Light waves

- Electromagnetic waves

Polarization

- Electric wave vector

[http://en.wikipedia.org/wiki/Polarization]
Fresnel equations

- Reflection of light polarized parallel and perpendicular to the plane of refraction

\[
\rho_\parallel = \frac{n_t \cos \theta - n \cos \phi}{n_t \cos \theta + n \cos \phi}
\]

\[
\rho_\perp = \frac{n \cos \theta - n_t \cos \phi}{n \cos \theta + n_t \cos \phi}
\]

\[
F_r(\theta) = \frac{1}{2} \left( \rho_\parallel^2 + \rho_\perp^2 \right)
\]

\[
F_t(\theta) = 1 - F_r(\theta)
\]
Glass sphere

- No Fresnel reflection
Glass sphere

- With Fresnel reflection
Shadow rays

Occluder

Shadow ray
Shadow rays

Only accept intersections if $t > \epsilon$, $\epsilon > 0.00001$

[Jensen]
color shade( hit ) {
    c = black
    for all lights L[k] {
        if( L[k] visible) {
            c += diffuse( hit, L[k] )
            c += specular( hit, L[k] )
        }
    }
    c += ambient( hit )
    trace( reflected ray, &reflected_hit)  
    c += shade( reflected_hit )
    trace( refracted ray, &refracted_hit)  
    c += shade( refracted_hit )
    return c
}
Today

- Triangle meshes
- Aggregate objects
- Acceleration structures
- Bounding volume hierarchies
Triangle meshes
Triangle meshes

class mesh : object {
    triangle t[N]
}

class triangle : object {
    vector3 v1, v2, v3       // vertices
    vector3 n1, n2, n3       // normals
    ...                     // other attributes
}
Triangle meshes

class mesh {
    triangle t[N]
    vertex v[NV]
}

class vertex {
    vector3 p    // vertices
    vector3 n    // normals
    ...          // other attributes
}

class triangle : object {
    int i1, i2, i3
    mesh *m
}
Aggregate objects

• Aggregate objects represent groups of objects
• Derived from object class

```cpp
class aggregate : object {
    bool intersect( ray, &hit )
    bounding_volume get_bv()
    object children[]
    bounding_volume bv
}

class bounding_volume {
    bool hit( ray )
}
```
Aggregate objects

```cpp
bool aggregate::intersect( ray, &hit ) {
    if( bv.hit( ray ) ) {
        bool hit_child = false
        for all children k {
            if( children[k].intersect( ray, &hit ) ) {
                hit_child = true
                if( hit.t < closest_hit.t ) {
                    closest_hit = hit
                }
            }
        }
        return hit_child
    } else {
        return false
    }
}
```
Aggregate objects

- Object hierarchies
- Instancing
- E.g., triangle meshes

```c
instance->aggregate->triangle[]
```
Acceleration structures

• How long does your ray tracer take to compute images?
Acceleration structures

- How long does your ray tracer take to compute images?
- Where does it spend most of its computation time?
Ray tracing pseudocode

for all pixels {
    computeprimary( &ray )
    for all objects {
        intersect( ray, &hit )
        if hit is closer than firsthit {
            firsthit = hit
        }
    }

    shade( firsthit )
}
Cost

- For each ray, the cost is linear in the number of objects in the scene
- Complexity $O(n)$ per ray
- Total cost: objects*rays

Example
- 1024x1024 image, 1000 triangles
- $10^9$ ray triangle intersections
Cost

- 6320 triangles
Cost

- 6320 triangles per ray
Acceleration structures

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

Two fundamental approaches

• Hierarchies of groups of objects
  “object subdivision”

• Space partitioning
  “spatial subdivision”
Acceleration structures

- Hierarchies of groups of objects
Acceleration structures

• Hierarchies of groups of objects
Acceleration structures

• Hierarchies of groups of objects
Acceleration structures

- Groups are represented by aggregate objects with bounding volumes
- “Bounding volume hierarchies”, BVH
- Logarithmic complexity $O(\log n)$
Bounding volume hierarchies

- Tree structure
- Each internal node is an aggregate object with a bounding volume
- Leave nodes are objects
Bounding volume hierarchies

- All objects in a subtree are within the bounds of its root
- Not all objects within the bounding volume of a node need to be in its subtree
- Subtrees can overlap spatially
- Subtrees are not ordered in any way
Bounding volume hierarchies

- If a node is not intersected by a ray, none of the objects in its subtree are.
- The subtree can be pruned during intersection testing.
- If a node is intersected, all children have to be tested for intersections.
Bounding volume hierarchies

• Build upon `aggregate` class
• Can construct n-ary or binary trees

class aggregate : object {
    bool intersect( ray, &hit )
    boundary_volume get_bv()
    object children[]
    bounding_volume bv
}

class bounding_volume {
    bool hit( ray )
}
Bounding volume hierarchies

- Recursive intersection

```cpp
bool aggregate::intersect( ray, &hit ) {
    if( bv.hit( ray ) ) {
        bool hit_child = false
        for all children k {
            if( children[k].intersect( ray, &hit ) ) {
                hit_child = true
                if( hit.t < closest_hit.t ) {
                    closest_hit = hit
                }
            }
        }
    }
    return hit_child
}
```
Bounding volume hierarchies

• Bounding volumes
  - Bounding boxes
  - Bounding spheres
  - Bounding anything
Axis aligned bounding boxes

class object {
    ...
    bounding_volume get_bv()
}

aggregate::get_bv() {
    bv = combined bounding volumes of all children
    return bv
}

instance::get_bv() {
    bv = transform bounding volume to world coordinates
    return bv
}
Axis aligned bounding boxes

- Fast intersection
- Intersection with axis aligned slabs
Axis aligned bounding boxes
Axis aligned bounding boxes

- Intersection test

```cpp
if( t_xmin > t_ymax ) or ( t_ymin > t_xmax ) {
    return false
} else {
    return true
}
```
Axis aligned bounding boxes

• Ray-plane intersection
  - Ray \( p(t) = e + td \)
  - Plane \( (p - a) \cdot n = 0 \)
  - Intersection \( t = \frac{(a-e) \cdot n}{d \cdot n} \)

• Axis aligned plane

\[
  t_{x_{\text{min}}} = \frac{((x_{\text{min}},0,0) - (e_x,e_y,e_z)) \cdot (1,0,0)}{(d_x,d_y,d_z) \cdot (1,0,0)}
\]

\[
  t_{x_{\text{min}}} = \frac{x_{\text{min}} - e_x}{d_x}
\]
Axis aligned bounding boxes

- Careful...
- If $d_x < 0$, ray hits $x_{\text{max}}$ intersection first, then $x_{\text{min}}$
- Additional test

```c
if( x_d >= 0 ) {
    t_xmin = ( x_min - e_x ) / d_x
    t_xmax = ( x_max - e_x ) / d_x
} else {
    t_xmin = ( x_max - e_x ) / d_x
    t_xmax = ( x_min - e_x ) / d_x
}
```
Axis aligned bounding boxes

- What happens if a ray is parallel to a coordinate axis?
- E.g., \( d_x = 0 \)
- Division by zero!

\[
    t_{x_{\min}} = \frac{x_{\min} - e_x}{0}
\]
Axis aligned bounding boxes

- Three possibilities
  1. No hit \( e_x \leq x_{min} \)
  2. Hit \( x_{min} < e_x < x_{max} \)
  3. No hit \( x_{max} \leq e_x \)
Axis aligned bounding boxes

- Rely on IEEE floating point arithmetic
  
  \(+a/0 = \infty, -a/0 = \infty\)
  
  \(a > 0\)

1. \((t_{x_{\text{min}}}, t_{x_{\text{max}}}) = (\infty, \infty)\)
2. \((t_{x_{\text{min}}}, t_{x_{\text{max}}}) = (-\infty, \infty)\)
3. \((t_{x_{\text{min}}}, t_{x_{\text{max}}}) = (-\infty, -\infty)\)
BVH construction

- Partitioning objects along coordinate axes (Shirley section 9.2.2)
- Binary tree
BVH construction
BVH construction
BVH construction

- Split horizontally
BVH construction

- Recursion
BVH construction

• Split vertically
BVH construction

- Split vertically

Etc....
class binary_bvh : aggregate {
    binary_bvh( int n, object o[] ) {
        children[0] = o[0]bv = object[0].get_bv()
    } else if ( n == 2) {
        children[0] = o[0]
        bv = combine( o[0].get_bv(), o[1].get_bv )
    } else {

BVH construction

sort o[0..n-1] along a coordinate axis
split o[0..n-1] into o1[0..k-1], o2[0..l-1]
children[0] = new binary_bvh( k-1, o1 )
children[1] = new binary_bvh( l-1, o2 )
bv = combine( children[0].get_bv(),
              children[1].get_bv() )
}

• Splitting list of objects
  - Equally in space
  - Equally in number of objects
Next time

- Incremental BVH construction
- Cost function for BVH
- Spatial subdivision