Ray Casting

Thomas Funkhouser
(courtesy of Finkelstein 9/27)
Princeton University
COS 426, Fall 2001

3D Rendering

• The color of each pixel on the view plane depends on the radiance emanating from visible surfaces

Ray Casting

• For each sample…  
  - Construct ray from eye position through view plane  
  - Find first surface intersected by ray through pixel  
  - Compute color sample based on surface radiance

Ray Casting

• Simple implementation:
  ```java
  Image RayCast(Camera camera, Scene scene, int width, int height)
  {
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++)
        for (int j = 0; j < height; j++)
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
    return image;
  }
  ```

Ray Casting

• Simple implementation:
  ```java
  Image RayCast(Camera camera, Scene scene, int width, int height)
  {
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++)
        for (int j = 0; j < height; j++)
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
    return image;
  }
  ```
Constructing Ray Through a Pixel

\[ \text{Ray: } P = P_0 + tv \]

Ray Casting

- Simple implementation:

```java
Image RayCast(Camera camera, Scene scene, int width, int height) {
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
        }
    }
    return image;
}
```

Ray-Scene Intersection

- Intersections with geometric primitives
  - Sphere
  - Triangle
  - Groups of primitives (scene)
- Acceleration techniques
  - Bounding volume hierarchies
  - Spatial partitions
    - Uniform grids
    - Octrees
    - BSP trees

Ray-Sphere Intersection

\[ \text{Ray: } P = P_0 + tv \]

\[ \text{Sphere: } |P - O|^2 - r^2 = 0 \]

Substituting for \( P \), we get:

\[ |P_0 + tv - O|^2 - r^2 = 0 \]

Solve quadratic equation:

\[ at^2 + bt + c = 0 \]

where:

\[ a = 1 \]
\[ b = 2 \cdot v \cdot (P_0 - O) \]
\[ c = |P_0 - O|^2 - r^2 = 0 \]

\[ P = P_0 + tv \]
Ray-Sphere Intersection II

Ray: \( P = P_0 + tV \)
Sphere: \( |P - O|^2 - r^2 = 0 \)

\[ L = O - P_0 \]
\[ t_{ca} = L \cdot V \]
if \( t_{ca} < 0 \) return 0
\[ d^2 = L \cdot L - t_{ca}^2 \]
if \( d^2 > r^2 \) return 0
\[ t_{tc} = \sqrt{r^2 - d^2} \]
\[ t = t_{ca} - t_{tc} \] and \( t = t_{ca} + t_{tc} \)

\( P = P_0 + tV \)

Ray-Sphere Intersection

• Need normal vector at intersection for lighting calculations

\[ N = (P - O) / ||P - O|| \]

Ray-Scene Intersection

• Intersections with geometric primitives
  - Triangle
    - Groups of primitives (scene)
  - Acceleration techniques
    - Bounding volume hierarchies
    - Spatial partitions
      - Uniform grids
      - Octrees
      - BSP trees

Ray-Triangle Intersection

• First, intersect ray with plane
• Then, check if point is inside triangle

Ray-Plane Intersection

Ray: \( P = P_0 + tV \)
Plane: \( P \cdot N + d = 0 \)

Substituting for \( P \), we get:
\( (P_0 + tV) \cdot N + d = 0 \)

Solution:
\[ t = -(P_0 \cdot N + d) / (V \cdot N) \]
\( P = P_0 + tV \)

Ray-Triangle Intersection I

• Check if point is inside triangle algebraically

For each side of triangle
\[ V_1 = T_1 - P \]
\[ V_2 = T_2 - P \]
\[ N_i = V_2 \times V_1 \]
Normalize \( N_i \)
\[ d_i = -P_0 \cdot N_i \]
if \( (P \cdot N_i + d_i) < 0 \) return FALSE; end
Ray-Triangle Intersection II

- Check if point is inside triangle parametrically

Compute $\alpha, \beta$:

$P = \alpha (T_2 - T_1) + \beta (T_3 - T_1)$

Check if point inside triangle:

$0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$

$\alpha + \beta \leq 1$

Other Ray-Primitive Intersections

- Cone, cylinder, ellipsoid:
  - Similar to sphere
- Box
  - Intersect 3 front-facing planes, return closest
- Convex polygon
  - Same as triangle (check point-in-polygon algebraically)
- Concave polygon
  - Same plane intersection
  - More complex point-in-polygon test

Ray-Scene Intersection

- Find intersection with front-most primitive in group

```c
Intersection FindIntersection(Ray ray, Scene scene) {
    min_t = infinity
    min_primitive = NULL
    For each primitive in scene {
        t = Intersect(ray, primitive);
        if (t < min_t) then
            min_t = t
            min_primitive = primitive
    }
    return Intersection(min_t, min_primitive)
}
```

Ray-Scene Intersection

- Intersections with geometric primitives
  - Sphere
  - Triangle
  - Groups of primitives (scene)
- Acceleration techniques
  - Bounding volume hierarchies
  - Spatial partitions
    - Uniform grids
    - Octrees
    - BSP trees

Bounding Volumes

- Check for intersection with simple shape first
  - If ray doesn’t intersect bounding volume, then it doesn’t intersect its contents

Bounding Volume Hierarchies I

- Build hierarchy of bounding volumes
  - Bounding volume of interior node contains all children
**Bounding Volume Hierarchies**
- Use hierarchy to accelerate ray intersections
  - Intersect node contents only if hit bounding volume

**Bounding Volume Hierarchies III**
- Sort hits & detect early termination

```c
FindIntersection(Ray ray, Node node)
{
    // Find intersections with child node bounding volumes
    ... 
    // Sort intersections front to back
    ... 
    // Process intersections (checking for early termination)
    min_t = infinity;
    foreach(intersected child i)
    {
        if (min_t < bv_t[i]) break;
        shape_t = FindIntersection(ray, child);
        if (shape_t < min_t) { min_t = shape_t; }
    }
    return min_t;
}
```

**Ray-Scene Intersection**
- Intersections with geometric primitives
  - Sphere
  - Triangle
  - Groups of primitives (scene)
- Acceleration techniques
  - Bounding volume hierarchies
  - Spatial partitions
    - Uniform grids
    - Octrees
    - BSP trees

**Uniform Grid**
- Construct uniform grid over scene
  - Index primitives according to overlaps with grid cells

**Uniform Grid**
- Trace rays through grid cells
  - Fast
  - Incremental

- Potential problem:
  - How choose suitable grid resolution?
  - Too little benefit if grid is too coarse
  - Too much cost if grid is too fine
Ray-Scene Intersection

- Intersections with geometric primitives
  - Sphere
  - Triangle
  - Groups of primitives (scene)
- Acceleration techniques
  - Bounding volume hierarchies
    - Uniform grids
    - Octrees
    - BSP trees

Octree

- Construct adaptive grid over scene
  - Recursively subdivide box-shaped cells into 8 octants
  - Index primitives by overlaps with cells

Generally fewer cells

Octree

- Trace rays through neighbor cells
  - Fewer cells
  - More complex neighbor finding
Trade-off fewer cells for more expensive traversal

Ray-Scene Intersection

- Intersections with geometric primitives
  - Sphere
  - Triangle
  - Groups of primitives (scene)
- Acceleration techniques
  - Bounding volume hierarchies
    - Uniform grids
    - Octrees
    - BSP trees

Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Every cell is a convex polyhedron

Binary Space Partition (BSP) Tree

- Simple recursive algorithms
  - Example: point finding
Binary Space Partition (BSP) Tree

- Trace rays by recursion on tree
  - BSP construction enables simple front-to-back traversal

Binary Space Partition (BSP) Tree

RayTreeIntersect(Ray ray, Node node, double min, double max)
{}
  if (Node is a leaf)
    return intersection of closest primitive in cell, or NULL if none
  else
    dist = distance of the ray point to split plane of node
    near_child = child of node that contains the origin of Ray
    far_child = other child of node
    if the interval to look is on near side
      return RayTreeIntersect(ray, near_child, min, max)
    else if the interval to look is on far side
      return RayTreeIntersect(ray, far_child, min, max)
    else if the interval to look is on both side
      if (RayTreeIntersect(ray, near_child, min, dist)) return …;
        else return RayTreeIntersect(ray, far_child, dist, max)

Other Accelerations

- Screen space coherence
  - Check last hit first
  - Beam tracing
  - Pencil tracing
  - Cone tracing
- Memory coherence
  - Large scenes
- Parallelism
  - Ray casting is “embarrassingly parallelizable”
  - etc.

Acceleration

- Intersection acceleration techniques are important
  - Bounding volume hierarchies
  - Spatial partitions
- General concepts
  - Sort objects spatially
  - Make trivial rejections quick
  - Utilize coherence when possible

Expected time is sub-linear in number of primitives

Summary

- Writing a simple ray casting renderer is easy
  - Generate rays
  - Intersection tests
  - Lighting calculations

Next Time is Illumination!

Without Illumination

With Illumination