Ray Casting

Tom Funkhouser
Princeton University
COS 426, Spring 2006

3D Rendering

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

Ray Casting

• For each sample …
  o Construct ray from eye position through view plane
  o Find first surface intersected by ray through pixel
  o 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;
}
```
**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**

- Ray: $P = P_0 + tV$
- Sphere: $|P - O|^2 - r^2 = 0$

```
Ray: $P = P_0 + tV$
Sphere: $|P - O|^2 - r^2 = 0$
```

Substituting for $P$, we get:

$|P_0 + tV - O|^2 - r^2 = 0$

Solve quadratic equation:

$a t^2 + bt + c = 0$

where:

- $a = 1$
- $b = 2V \cdot (P_0 - O)$
- $c = |P_0 - O|^2 - r^2 = 0$

$P = P_0 + tV$

**Ray-Sphere Intersection I**

- Algebraic Method

```
Ray: $P = P_0 + tV$
Sphere: $|P - O|^2 - r^2 = 0$
```

Substituting for $P$, we get:

$|P_0 + tV - O|^2 - r^2 = 0$

Solve quadratic equation:

$a t^2 + bt + c = 0$

where:

- $a = 1$
- $b = 2V \cdot (P_0 - O)$
- $c = |P_0 - O|^2 - r^2 = 0$

$P = P_0 + tV$
**Ray-Sphere Intersection**

- Need normal vector at intersection for lighting calculations

\[ \mathbf{N} = \frac{(\mathbf{P} - \mathbf{O})}{||\mathbf{P} - \mathbf{O}||} \]

**Ray-Sphere Intersection II**

Ray: \( \mathbf{P} = \mathbf{P}_0 + t\mathbf{V} \)
Sphere: \( ||\mathbf{P} - \mathbf{O}||^2 - r^2 = 0 \)

- Geometric Method

\[
\begin{align*}
L &= \mathbf{O} - \mathbf{P}_0 \\
t_a &= \frac{L \cdot \mathbf{V}}{L \cdot L} \\
t &= t_a \\
\mathbf{P} &= \mathbf{P}_0 + t\mathbf{V} \\
\end{align*}
\]

**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-Plane Intersection**

Ray: \( \mathbf{P} = \mathbf{P}_0 + t\mathbf{V} \)
Plane: \( \mathbf{P} \cdot \mathbf{N} + d = 0 \)

- Algebraic Method

Substituting for \( \mathbf{P} \), we get:
\[
(\mathbf{P}_0 + t\mathbf{V}) \cdot \mathbf{N} + d = 0
\]

Solution:
\[
\begin{align*}
t &= \frac{-(\mathbf{P}_0 \cdot \mathbf{N} + d)}{\mathbf{V} \cdot \mathbf{N}} \\
\mathbf{P} &= \mathbf{P}_0 + t\mathbf{V}
\end{align*}
\]

**Ray-Triangle Intersection I**

- Check if point is inside triangle algebraically

For each side of triangle
\[
\begin{align*}
\mathbf{V}_1 &= \mathbf{T}_1 - \mathbf{P}_0 \\
\mathbf{V}_2 &= \mathbf{T}_2 - \mathbf{P}_0 \\
\mathbf{N}_i &= \mathbf{V}_i \times \mathbf{V}_2 \\
\text{Normalize } \mathbf{N}_i
\end{align*}
\]

if \( (\mathbf{P} - \mathbf{P}_0) \cdot \mathbf{N}_i < 0 \)
return FALSE;
end

**Ray-Triangle Intersection**

- First, intersect ray with plane
- Then, check if point is inside triangle
Ray-Triangle Intersection II

- Check if point is inside triangle parametrically
  
  Compute “barycentric coordinates” α, β:
  
  \[ \alpha = \frac{\text{Area}(T_1T_2P)}{\text{Area}(T_1T_2T_3)} \]
  \[ \beta = \frac{\text{Area}(T_1P)}{\text{Area}(T_1T_2T_3)} \]
  
  \[ \text{Area}(T_1T_2T_3) = \frac{1}{2} (T2-T1) \times (T3-T1) \]
  
  Check if point inside triangle, 0 ≤ α ≤ 1 and 0 ≤ β ≤ 1
  \[ α + β ≤ 1 \]

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
IntersectionFindIntersection(Ray ray, Scene scene)
{
    min_t = infinity
    min_primitive = NULL
    For each primitive in scene {
        t = Intersect(ray, primitive);
        if (t > 0 && t < min_t) then
            min_primitive = primitive
            min_t = t
    }
    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

```
Ray-Sphere Intersection
```

Bounding Volumes

- Check for intersection with simple shape first

```
Ray-Box Intersection
```
Bounding Volumes

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

Bounding Volumes

- Sort hits & detect early termination

```
FindIntersection(Ray ray, Scene scene)
{
    // Find intersections with bounding volumes
    // Sort intersections from to back
    // Process intersections (checking for early termination)
    min_t = infinity;
    for each intersected bounding volume i {
        if (min_t < bv_t[i]) break;
        shape_t = FindIntersection(ray, bounding volume contents);
        if (shape_t < min_t) min_t = shape_t;
    }
    return min_t;
}
```

Bounding Volumes

- Use hierarchy to accelerate ray intersections
  - Intersect node contents only if hit bounding volume

Bounding Volumes

- Check for intersection with simple shape first
  - If ray doesn't intersect bounding volume, then it doesn't intersect its contents
  - If found another hit closer than hit with bounding box, then can skip checking contents of bounding box

Bounding Volume Hierarchies

- Build hierarchy of bounding volumes
  - Bounding volume of interior node contains all children

```
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;
    for each 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;
}
```

Bounding Volume Hierarchies

- Traverse scene nodes recursively

```
FindIntersection(Ray ray, Scene scene)
{
    // Find intersections with bounding volumes
    // Sort intersections front to back
    // Process intersections (checking for early termination)
    min_t = infinity;
    for each intersected bounding volume i {
        min_t = infinity;
        for each intersected child i {
            if (min_t < bv_t[i]) break;
            shape_t = FindIntersection(ray, bounding volume contents);
            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
      - 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

  Only check primitives in intersected grid cells

Potential problem:

- How choose suitable grid resolution?

Uniform Grid

- Potential problem:
  - 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

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

- 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

Ray/TreeIntersect(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 Ray/TreeIntersect(ray, near_child, min, max)
    else if the interval to look is on far side
      return Ray/TreeIntersect(ray, far_child, min, max)
    else if the interval to look is on both side
      if (Ray/TreeIntersect(ray, near_child, min, dist)) return ...;
      else return Ray/TreeIntersect(ray, far_child, dist, max)
  }
**Other Accelerations**

- Screen space coherence
  - Check last hit first
  - Beam tracing
  - Pencil 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

```
Image RayCast(Camera camera, Scene scene, int width, int height)

Ray ray = ConstructRayThroughPixel(camera, i, j);
image[i][j] = GetColor(hit);
```

**Heckbert’s business card ray trace**

```
typedef struct{double x, y, z} vec; vec U, black, amb = [.02, .02, .02];
struct sphere{ vec cen, color; }
```

---

**Next Time is Illumination!**

Without Illumination  With Illumination