Ray Casting

Adam Finkelstein
Princeton University
COS 426, Spring 2003

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;
}
```
Constructing Ray Through a Pixel

- \( P = P_0 + tV \)

- \( \theta = \) frustum half-angle
- \( d = \) distance to view plane
- \( \text{right} = \) towards \( x \) up

\[
P_1 = P_0 + d\text{towards} - d\tan(\theta)\text{right}
\]

\[
P_2 = P_0 + d\text{towards} + d\tan(\theta)\text{right}
\]

\[
P = P_1 + (i/\text{width} + 0.5) \cdot (P_2 - P_1)
\]

\[
V = \frac{(P - P_0)}{\|P - P_0\|}
\]

Ray Casting

- Simple implementation:

```java
public 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:

\[
a t^2 + bt + c = 0
\]

where:

\[
a = 1
\]

\[
b = 2 V \cdot (P_0 - O)
\]

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

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

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

\[ L = O - P_0 \]
\[ t = \frac{L \cdot V}{d^2} \]
\[ d = \frac{L \cdot L - t^2}{2} \]
\[ t_{\text{max}} = \frac{d}{2} \]
\[ t_{\text{target}} = \sqrt{r^2 - d^2} \]
\[ t = t_{\text{max}} - t_{\text{target}} \text{ and } t_{\text{max}} + t_{\text{target}} \]
\[ P = P_0 + tV \]

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

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

Solution:
\[ t = \frac{(P_0 \cdot N + d)}{(V \cdot N)} \]
\[ P = P_0 + tV \]

Ray-Triangle Intersection I

Check if point is inside triangle algebraically

For each side of triangle:
\[ V_i = T_i - P \]
\[ N_i = V_j \times V_k \]
Normalize \( N_i \)
if \( (P - P_0) \cdot N_i < 0 \)
return FALSE;
end

Ray-Sphere Intersection

- Need normal vector at intersection for lighting calculations

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

$$ 0 \leq \alpha \leq 1 \text{ and } 0 \leq \beta \leq 1 $$

$$ \alpha + \beta \leq 1 $$

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

Intersection FindIntersection(Ray ray, Scene scene)
{
  min_t = \infty
  minPrimitive = NULL
  For each primitive in scene {
    t = Intersect(ray, primitive);
    if (t > 0 && t < min_t) then
      minPrimitive = primitive
      min_t = t
  }
  return Intersection(min_t, minPrimitive)
}

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

Bounding Volumes
• Check for intersection with simple shape first
Bounding Volumes

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

Still need to check for intersections with shape.

Bounding Volume Hierarchies I

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

Bounding Volume Hierarchies II

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

Bounding Volume Hierarchies III

- Sort hits & detect early termination

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
  
  Only check primitives in intersected grid cells

Uniform Grid

- 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
    - Spatial partitions
      - 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
  - Spatial partitions
    - Uniform grids
    - BSP trees

Binary Space Partition (BSP) Tree

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

- Simple recursive algorithms
  - Example: point finding

```c
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 sides
            if (RayTreeIntersect(ray, near_child, min, dist)) return …;
            else return RayTreeIntersect(ray, far_child, dist, max)
} 
```

Binary Space Partition (BSP) Tree

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

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

Heckbert's business card ray tracer

- typedef struct{double x,y,z}vec;vec U,black,ambient(0,0,0.25);struct sphere{ vec cen,color;double rad,kd,ks,kt,kl,ir};*s,*best,sph[5]=

Next Time is Illumination!