Ray Casting

Adam Finkelstein
Princeton University
COS 426, Spring 2005

3D Rendering

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

Simplest method is ray casting

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

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

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-Sphere Intersection

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:
\[ at^2 + bt + c = 0 \]
where:
\[ a = 1 \]
\[ b = 2V \cdot (P_0 - O) \]
\[ c = |P_0 - O|^2 - r^2 = 0 \]

\[ P = P_0 + tV \]

Ray-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 I

Algebraic Method
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
\( t_{hc} = \sqrt{r^2 - d^2} \)
\( t = t_{ca} - t_{hc} \) and \( t_{ca} + t_{hc} \)
\( P = P_0 + tV \)

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-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

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

Ray-Triangle Intersection I

• Check if point is inside triangle algebraically

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

• Check if point is inside triangle parametrically

![Ray-Triangle Intersection II Diagram]

Compute "barycentric coordinates" \( \alpha, \beta \):

\[
\alpha = \frac{\text{Area}(T_1T_2P)}{\text{Area}(T_1T_2T_3)}
\]
\[
\beta = \frac{\text{Area}(T_1PT_3)}{\text{Area}(T_1T_2T_3)}
\]

Area\((T_1T_2T_3) = \frac{1}{2} (T_2-T_1) \times (T_3-T_1)\)

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:
  o Similar to sphere

• Box
  o Intersect 3 front-facing planes, return closest

• Convex polygon
  o Same as triangle (check point-in-polygon algebraically)

• Concave polygon
  o Same plane intersection
  o More complex point-in-polygon test

Ray-Scene Intersection

• Find intersection with front-most primitive in group

![Ray-Scene Intersection Diagram]

Ray-Scene Intersection

• Intersections with geometric primitives
  o Sphere
  o Triangle
  o Groups of primitives (scene)

  » Acceleration techniques
    o Bounding volume hierarchies
    o Spatial partitions
      » Uniform grids
      » Octrees
      » BSP trees

Bounding Volumes

• Check for intersection with simple shape first

![Bounding Volumes Diagram]
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

```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;
    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;
}
```

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
    - Uniform grids
    - Octrees
    - BSP trees

**Binary Space Partition (BSP) Tree**

- Simple recursive algorithms
  - Example: point finding
- Trace rays by recursion on tree
  - BSP construction enables simple front-to-back traversal
- Recursively partition space by planes
  - Every cell is a convex polyhedron
- Intersections with geometric primitives
  - Sphere
  - Triangle
  - Groups of primitives (scene)
- Acceleration techniques
  - Bounding volume hierarchies
    - Uniform grids
    - Octrees
    - BSP trees
- Screen space coherence
  - Check last hit first
  - Beam tracing
  - Pencil tracing
  - Cone tracing
- Memory coherence
  - Large scenes
- Parallelism
  - Ray casting is "embarrassingly parallelizable"
  - etc.

```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 side
            if (RayTreeIntersect(ray, near_child, min, dist)) return ...;
                else return RayTreeIntersect(ray, far_child, dist, max)
}
```
Acceleration

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

Expected time is sub-linear in number of primitives

Summary

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

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;
}

Heckbert’s business card ray tracer

• typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{ vec cen,color; double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={{0.,6.,.5,1.1,.1,.9, .05,2.85,0.1,.7,.8,.5,.1,.5,.2,1., .7,3.0,.05,1.2,1.8,-.5,1.8,1.,3,.7,0.0,1.2,3.6,15.1,.8,.1,.7,.0,.0,.6,.1,.5,.3,.1,12., .8,1., 1.5,0.0,.0,.5,.15,.5},vec c;double u,vw,wt;vec U,vcomb(a,A,B);vec A; B;[return A.x * B.x A.y B.y + A.z B.z;vec vcomb(a,A,B);vec A;B;vec U;}
t dendrite u=..vcomb(x,A,B);double a=vec A; B;}
struct sphere*intersect(P,D)
{
    best=0; tmin=1e30; s=sph[5]; while(s->cen)
    if((e=s->kl*vcomb(-1.,P,s->cen)))>
    intersect(P,D),else return amb;color=vcomb(e, s->ks,trace(level,P, vcomb(2*d,N,D)),s->kd),
vcomb(s->kl,U,black));
}

Next Time is Illumination!

Without Illumination

With Illumination

/*minray!*/