Ray Casting

Thomas Funkhouser
Princeton University
C0S 426, Fall 2000

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 …
  - 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

Ray: \( P = P_0 + tV \)

• 2D Example

\[ \Theta = \text{frustum half-angle} \]
\[ d = \text{distance to view plane} \]

right = towards x up

\[ P1 = P_0 + d \times \text{towards} - d \times \tan(\Theta) \times \text{right} \]
\[ P2 = P_0 + d \times \text{towards} + d \times \tan(\Theta) \times \text{right} \]

\[ P = P1 + (i/\text{width} + 0.5) \times 2d \times \tan(\Theta) \times \text{right} \]
\[ V = (P - P_0) / \|P - P_0\| \]
Ray Casting

- Simple implementation:

```java
def 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-Sphere Intersection I

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

Algebraic Method

Substituting for \( P \), we get:
\[ |P_0 + tV - O|^2 - r^2 = 0 \]

Solve quadratic equation:
\[ at^2 + bt + c = 0 \]
where:
\[ a = 1 \]
\[ b = 2 \mathbf{V} \cdot (P_0 - \mathbf{O}) \]
\[ c = |P_0 - \mathbf{C}|^2 - r^2 = 0 \]

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

Ray-Sphere Intersection

- Need normal vector at intersection for lighting calculations

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

- Intersections with geometric primitives
  - Sphere
    - 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:
\[ t = -\frac{(P_0 \cdot N + d)}{(V \cdot N)} \]
\[ P = P_0 + tV \]

Ray-Triangle Intersection I

- Check if point is inside triangle algebraically

For each side of triangle
\[ V_1 = T_1 - P \]
\[ V_2 = T_2 - P \]
\[ N_1 = V_2 \times V_1 \]
Normalize \( N_1 \)
\[ d_1 = -P_0 \cdot N_1 \]
if \( ((P \cdot N_1 + d_1) < 0) \) return FALSE;
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.
  $$ 0 \leq \alpha \leq 1 \text{ and } 0 \leq \beta \leq 1 $$

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 = infinity
    min_primitive = NULL
    For each primitive in scene {
        t = Intersect(ray, primitive);
        if (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

Next Time!
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)
{
  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;
}