Ray Casting

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

- 2D Example

\[ \varpi = \text{frustum half-angle} \]
\[ d = \text{distance to view plane} \]
\[ \text{right} = \text{towards up} \]

\[ P_1 = P_0 + d \times \text{towards} \]
\[ P_2 = P_0 + d \times \text{towards} + d \times \tan(\varpi) \times \text{right} \]

\[ P = P_1 + (i + 0.5) \times \text{width} \times \frac{2 \times \tan(\varpi) \times \text{right}}{P_2 - P_1} \]

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

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

\[ \text{Ray: } P = P_0 + tv \]
\[ \text{Sphere: } |P - C|^2 - r^2 = 0 \]

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

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

If ray direction is normalized!
Ray-Sphere Intersection
- Need normal vector at intersection for lighting calculations

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

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:
  \[ t = \frac{-\langle P_0 \cdot N + d \rangle}{v \cdot N} \]

Ray-Triangle Intersection I
- Check if point is inside triangle geometrically

\[ A \times B \text{ will point in the opposite direction from } C \times B! \]

Ray-Triangle Intersection II
- Check if point is inside triangle parametrically

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

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

- 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 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
• 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
  • 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

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
  - 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
  - Recursive descent – don't do 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
  - Spatial partitions
    - Octrees
    - BSP trees
Binary Space Partition (BSP) Tree

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

- Simple recursive algorithms
  - Example: point location

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

- Intersection acceleration techniques are important
  - Bounding volume hierarchies
  - Spatial partitions

- General concepts
  - Sort objects spatially
  - Make trivial rejections quick
  - Utilize coherence when possible

Other Accelerations

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

- Memory coherence
  - Large scenes

- Parallelism
  - Ray casting is "embarrassingly parallel"

Acceleration

- 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)
{
  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;
}

Next Time is Illumination!

Without Illumination  With Illumination