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

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

WHY?
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

\[ \theta = \text{frustum half-angle} \]
\[ d = \text{distance to view plane} \]
\[ \text{right} = \text{towards} \times \text{up} \]
\[ P_1 = P_0 + d\text{towards} - d\tan(\theta)\text{right} \]
\[ P_2 = P_0 + d\text{towards} + d\tan(\theta)\text{right} \]
\[ P = P_1 + (i/width + 0.5) * (P_2 - P_1) \]
\[ V = (P - P_0) / \|P - P_0\| \]

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-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: \(|P - C|^2 - r^2 = 0\)

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

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

\( P = P_0 + tV \)
Ray-Sphere Intersection

- Need normal vector at intersection for lighting calculations

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

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

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

Solution:
\[ t = -\frac{\mathbf{P}_0 \cdot \mathbf{N} + d}{\mathbf{V} \cdot \mathbf{N}} \]
\[ \mathbf{P} = \mathbf{P}_0 + t \mathbf{V} \]
Ray-Triangle Intersection I
- Check if point is inside triangle geometrically

Ax'B' will point in the opposite direction from CxB!

\[
\text{SameSide}(p_1, p_2, a, b):
\begin{align*}
  cp1 &= \text{Cross} (b-a, p_1-a) \\
  cp2 &= \text{Cross} (b-a, p_2-a) \\
  \text{return Dot}(cp1, cp2) &\geq 0
\end{align*}
\]

\[
\text{PointInTriangle}(p, a, b, c):
\begin{align*}
  \text{return SameSide}(p, a, b, c) \land \\
  \text{SameSide}(p, b, a, c) \land \\
  \text{SameSide}(p, c, a, b)
\end{align*}
\]

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

\[
\text{Intersection FindIntersection}(\text{Ray} \ ray, \text{Scene} \ scene) \{
\begin{align*}
  \text{min}_t &= \text{infinity} \\
  \text{min}_\text{primitive} &= \text{NULL} \\
  \text{For each primitive in scene} \{ \\
  t &= \text{Intersect}(\text{ray}, \text{primitive}); \text{if } (t > 0 \&\& t < \text{min}_t) \text{ then } \\
  \text{min}_\text{primitive} &= \text{primitive} \\
  \text{min}_t &= t
  \}\}
\end{align*}
\]

\[
\text{return Intersection}(\text{min}_t, \text{min}_\text{primitive})
\]
### 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

- 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

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

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

- Trace rays through neighbor cells
  - Fewer cells
  - Recursive descent – don’t do neighbor finding...

- Trade-off fewer cells for more expensive traversal

- Generally fewer cells

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
Binary Space Partition (BSP) Tree

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

Example: point location

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, dist, 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)
}
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”
  - 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

ImageRayCast(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
Next Time is Illumination!

Without Illumination  With Illumination