Foundations of Computer Graphics

Online Lecture 9: Ray Tracing 1

History and Basic Ray Casting

Ravi Ramamoorthi

Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Interreflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)
- Realistic Materials (Velvet, Paints, Glass)
- ...

Ray Tracing

- Different Approach to Image Synthesis as compared to Hardware pipeline (OpenGL)
- Pixel by Pixel instead of Object by Object
- Easy to compute shadows/transparency/etc

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
- Optimizations

Ray Tracing: History

- Appel 68
- Whitted 80 [recursive ray tracing]
  - Landmark in computer graphics
  - Lots of work on various geometric primitives
  - Lots of work on accelerations
- Current Research
  - Real-Time raytracing (historically, slow technique)
  - Ray tracing architecture

Ray Tracing History

- “An improved illumination model for shaded display” by T. Whitted, CACM 1980
- 512x512, VAX 11/780
- 74 min, today real-time

Turner Whitted 1980.
Spheres and Checkerboard
Outline in Code

Image Raytrace (Camera cam, Scene scene, int width, int height) 
{

    Image image = new Image (width, height) ;
    for (int i = 0 ; i < height ; i++) 
    for (int j = 0 ; j < width ; j++) 
    
        Ray ray = RayThruPixel (cam, i, j) ;
        Intersection hit = Intersect (ray, scene) ;
        image[i][j] = FindColor (hit) ;

    return image ;

}

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
- Optimizations

Ray Casting

Produce same images as with OpenGL
- Visibility per pixel instead of Z-buffer
- Find nearest object by shooting rays into scene
- Shade it as in standard OpenGL

Comparison to hardware scan-line

- Per-pixel evaluation, per-pixel rays (not scan-convert each object). On face of it, costly
- But good for walkthroughs of extremely large models (amortize preprocessing, low complexity)
- More complex shading, lighting effects possible

Foundations of Computer Graphics

Online Lecture 9: Ray Tracing 1

Core Algorithm: Shadows and Reflections

Ravi Ramamoorthi
Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
- Optimizations

Shadows

Shadows: Numerical Issues
- Numerical inaccuracy may cause intersection to be below surface (effect exaggerated in figure)
- Causing surface to incorrectly shadow itself
- Move a little towards light before shooting shadow ray

Mirror Reflections/Refractions

Generate reflected ray in mirror direction,
Get reflections and refractions of objects

Recursive Ray Tracing

For each pixel
- Trace Primary Eye Ray, find intersection
- Trace Secondary Shadow Ray(s) to all light(s)
  - Color = Visible ? Illumination Model : 0 ;
- Trace Reflected Ray
  - Color += reflectivity * Color of reflected ray

Problems with Recursion

- Reflection rays may be traced forever
- Generally, set maximum recursion depth
- Same for transmitted rays (take refraction into account)
Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Interreflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)

Discussed in this lecture
Not discussed but possible with distribution ray tracing
Hard (but not impossible) with ray tracing; radiosity methods

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
- Optimizations

Ray/Object Intersections

- Heart of Ray Tracer
  - One of the main initial research areas
  - Optimized routines for wide variety of primitives
- Various types of info
  - Shadow rays: Intersection/No Intersection
  - Primary rays: Point of intersection, material, normals
  - Texture coordinates
- Work out examples
  - Triangle, sphere, polygon, general implicit surface

Ray-Sphere Intersection

\[
\text{ray} = P = P_o + Pt
\]
\[
\text{sphere} = (P - C)(P - C) - r^2 = 0
\]

Substitute
**Ray-Sphere Intersection**

\[
\text{ray} = P_0 + Pt \\
\text{sphere} = (P - C)(P - C) - r^2 = 0
\]

Substitute

\[
\text{ray} = P_0 + Pt \\
\text{sphere} = (P_0 + Pt - C)(P_0 + Pt - C) - r^2 = 0
\]

Simplify

\[
t^2(P_0, P) + 2t(P_0 - C) + (P_0 - C)(P_0 - C) - r^2 = 0
\]

Solve quadratic equations for \( t \)

- 2 real positive roots: pick smaller root
- Both roots same: tangent to sphere
- One positive, one negative root: ray origin inside sphere (pick + root)
- Complex roots: no intersection (check discriminant of equation first)

**Ray-Sphere Intersection**

\[
\text{ray} = P_0 + Pt \\
\text{sphere} = (P - C)(P - C) - r^2 = 0
\]

Substitute

\[
\text{ray} = P_0 + Pt \\
\text{sphere} = (P_0 + Pt - C)(P_0 + Pt - C) - r^2 = 0
\]

Simplify

\[
t^2(P_0, P) + 2t(P_0 - C) + (P_0 - C)(P_0 - C) - r^2 = 0
\]

**Ray-Triangle Intersection**

- One approach: Ray-Plane intersection, then check if inside triangle
- Plane equation:

**Ray-Triangle Intersection**

- One approach: Ray-Plane intersection, then check if inside triangle
- Plane equation:
Ray-Triangle Intersection

- One approach: Ray-Plane intersection, then check if inside triangle
- Plane equation:
  \[ p \cdot n = A n = 0 \]

Ray inside Triangle

- Once intersect with plane, need to find if in triangle
- Many possibilities for triangles, general polygons
- We find parametrically [barycentric coordinates]. Also useful for other applications (texture mapping)

Other primitives

- Much early work in ray tracing focused on ray-primitive intersection tests
- Cones, cylinders, ellipsoids
- Boxes (especially useful for bounding boxes)
- General planar polygons
- Many more

Ray-Tracing Transformed Objects

We have an optimized ray-sphere test
- But we want to ray trace an ellipsoid...

Solution: Ellipsoid transforms sphere
- Apply inverse transform to ray, use ray-sphere
- Allows for instancing (traffic jam of cars)

Mathematical details worked out next
Transformed Objects

- Consider a general 4x4 transform $M$ (matrix stacks)
- Apply inverse transform $M^{-1}$ to ray
  - Locations stored and transform in homogeneous coordinates
  - Vectors (ray directions) have homogeneous coordinate set to 0 [so there is no action because of translations]
- Do standard ray-surface intersection as modified
- Transform intersection back to actual coordinates
  - Intersection point $p$ transforms as $Mp$
  - Normals $n$ transform as $M^{-1}n$. Do all this before lighting

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
- Optimizations
- Current Research

Accelerated Structures

Bounding boxes (possibly hierarchical)
- If no intersection bounding box, needn’t check objects

Spatial Hierarchies (Oct-trees, kd trees, BSP trees)
Acceleration Structures: Grids

Acceleration and Regular Grids

- Simplest acceleration, for example 5x5x5 grid
- For each grid cell, store overlapping triangles
- March ray along grid (need to be careful with this), test against each triangle in grid cell
- More sophisticated: kd-tree, oct-tree bsp-tree
- Or use (hierarchical) bounding boxes