Ray Tracing

Computer Graphics Seminar

Ashraf Abbasov
1st year MSc Computer Science student
What is Ray?
Ray Casting

\[ P(t) = A + \text{direction} \times t \]
General idea of Ray Tracing
Shadows

Reflection

Refraction

The role of the illumination model is to determine how much light is reflected to the viewer from a visible point on a surface as a function of light source direction and strength, viewer position, surface orientation, and surface properties. The shading calculations can be performed on three scales: microscopic, local, and global. Although the exact nature of reflection from surfaces is best explained in terms of microscopic interactions between light rays and the surface [3], most shaders produce excellent results using aggregate local surface data. Unfortunately, these models are usually limited in scope, i.e., they look only at light source and surface orientations, while ignoring the overall setting in which the surface is placed. The reason that shaders tend to operate on local data is that traditional visible surface algorithms cannot provide the necessary global data.

A shading model is presented here that uses global information to calculate intensities. Then, to support this shader, extensions to a ray tracing visible surface algorithm are presented.

1. Conventional Models

The simplest visible surface algorithms use shaders based on Lambert's cosine law. The intensity of the reflected light is proportional to the dot product of the surface normal and the light source direction, simulating a perfect diffuser and yielding a reasonable looking approximation to a dull, matte surface. A more sophisticated model is the one devised by Bui-Tuong Phong [8]. Intensity from Phong's model is given by

\[ I = I_a + k_d \sum_{j=1}^{n} (N \cdot L_j) + k_r \sum_{j=1}^{n} (N \cdot L_j)^2, \]
Shadows
Reflection

[Diagram showing light source, eye, shadow ray, and reflection ray]
Refraction
Rendered Image

74 minutes
Ray-Sphere intersection

\[ P(t) = A + t \cdot b \]

\[ x^2 + y^2 + z^2 = R^2 \]
Ray-Sphere intersection

\[ t^2 \mathbf{b} \cdot \mathbf{b} + 2t \mathbf{b} \cdot (\mathbf{A} - \mathbf{C}) + (\mathbf{A} - \mathbf{C}) \cdot (\mathbf{A} - \mathbf{C}) - R^2 = 0 \]
Rasterization vs Ray Tracing

Rasterization loop:
For each object:
  For each pixel:
    closer?

Ray Tracing loop:
For each pixel:
  For each object:
    closer?
Rasterization
Ray Tracing
Bounding Volume Hierarchy

$O(\log n)$
BVH ALGORITHM
Massive Improvement in Search Efficiency
Ray Tracing Effects

Hard Shadows

Soft Shadows

Global Illumination

Glossy Reflection

Ambient Occlusion
Hard Shadows

Point light

Ray rom intersection point to light

Black if something in the way

Illuminated if nothing in the way
Soft Shadows

Aerial Light

Multiple rays from intersection point to various points on the light

Similar to stochastic ray tracing

Some lights blocked - penumbra - soft shadows

All lights blocked - umbra - full shadow
Global illumination

Color bleeding/Indirect lighting/Interreflection

Several rays bouncing around
Glossy Reflection

Several Reflection Rays

Like stochastic ray tracing
Glossy Reflection

Several Reflection Rays

Like stochastic
Ambient Occlusion

Several rays to a certain distance

To make crevices darken
Ray Tracing Pipeline
Ray Tracing Shaders

- Ray Generation Shader
- Intersection Shader
- Any-hit Shader
- Closest Hit Shader
- Miss shader
Ray Generation Shader

Ray generation shaders ("raygen") begin all ray tracing work. A raygen shader runs on a 2D grid of threads, much like a compute shader, and is the starting point for tracing rays into the scene. It is also responsible for writing the final output from the ray tracing algorithm out to memory.
Intersection Shader

Intersection shaders implement arbitrary ray-primitive intersection algorithms. They are useful to allow applications to intersect rays with different kinds of primitives (e.g., spheres) that do not have built-in support. Triangle primitives have built-in support and don’t require an intersection shader.
Intersection Shader

Intersection shaders implement arbitrary ray-primitive intersection algorithms. They are useful to allow applications to intersect rays with different kinds of primitives (e.g., spheres) that do not have built-in support. Triangle primitives have built-in support and don’t require an intersection shader.
Hit Shaders

Hit shaders are invoked when a ray-primitive intersection is found. They are responsible for computing the interactions that happen at an intersection point (e.g., light-material interactions for graphics applications) and can spawn new rays as needed.

There are two kinds of hit shaders: any-hit shaders are invoked on all intersections of a ray with scene primitives, in an arbitrary order, and can reject intersections in addition to computing shading data. Closest-hit shaders invoke only on the closest intersection point along the ray.
Miss Shaders

Miss shaders are invoked when no intersection is found for a given ray.
Ray Tracing Pipeline

[Diagram of ray tracing pipeline with steps such as Ray Generation Shader, Call to TraceRay(), Acceleration Traversal, Hit a Node?, Intersection Shader, Is Opaque?, Any-Hit Shader, Hit Accepted ReportHit(), Update Hit Distance, and Closest Hit Distance.]
Ray Tracing Hardware
RT Cores

Ray-bounding volume

Ray-triangle intersection
One METRO Exodus Frame

NVIDIA PASCAL™
NVIDIA GeForce®
GTX 1080 Ti

NVIDIA TURING™
GeForce RTX™ 2080
NO RT CORE

TURING RTX
GeForce RTX 2080
RT CORE+DLSS

TIME

FP32 Cores    INT32 Cores    RT Cores    Tensor Cores
Denoising for Ray Tracing

Ray Traced images can be very noisy

Films do 3000 rays per pixel but still have noise

Nvidia GPUs can do Real-Time Denoising. (NRD)

  Designed to Use One Ray or Less Per Pixel

  Diffuse, Specular or Reflections, Infinite Light Source Shadows
References


[5] Ray Tracing in One Weekend, 2020, Peter Shirley
Thank You For Listening
Any Questions?