Basic Ray Tracing
Rendering: Reality

Eye acts as pinhole camera

Photons from light hit objects
Rendering: Reality

Eye acts as pinhole camera

Photons from light hit objects
Bounce everywhere

Extremely few hit eye, form image

one lightbulb = $10^{19}$ photons/sec
Synthetic Pinhole Camera

Useful abstraction: virtual image plane

[Diagram of synthetic pinhole camera]
Rendering: Ray Tracing

Reverse of reality
• shoot rays through image plane
• see what they hit
• secondary rays for:
  • reflections
  • shadows
• Embarrassingly parallel
Local Illumination

Simplifying assumptions:
• ignore everything except eye, light, and object
• no shadows, reflections, etc
“Ray Tracing is Slow”

Very true in the past; still true today
But real-time ray tracing is coming
Side Note

What does NVidia’s RTX hardware do?
Why Slow?

Naïve algorithm: $O(NR)$
- $R$: number of rays
- $N$: number of objects

But rays can be cast in parallel
- each ray $O(N)$
- even faster with good culling
Why Slow?

Despite being parallel:

1. poor cache coherence
   - nearby rays can hit different geometry

2. unpredictable
   - must **shade** pixels whose rays hit object
   - may require tracing rays **recursively**
Basic Algorithm

For each pixel:
• shoot ray from camera through pixel
• find first object it hits
• if it hits something
  • shade that pixel
  • shoot secondary rays
Shoot Rays From Camera

Ray has **origin** and **direction**

Points on ray are the **positive span**

How to create a ray?
Shoot Rays From Camera

Creating a ray:

• origin is eye

• pick direction to pierce center of pixel
Whitted-style Ray Tracing

- Turner Whitted introduced ray tracing to graphics in 1980
  - Combines eye ray tracing + rays to light and recursive tracing

- Algorithm:
  1. For each pixel, trace primary ray in direction $\mathbf{V}$ to the first visible surface.
  2. For each intersection trace secondary rays:
     - Shadow in direction $\mathbf{L}$ to light sources
     - Reflected in direction $\mathbf{R}$
     - Refracted (transmitted) in direction $\mathbf{T}$
  3. Calculate shading of pixel based on light attenuation
Find First Object Hit By Ray

Collision detection: find all values of $t$ where ray hits object boundary

Take smallest positive value of $t$
When Did We Hit an Object?

How do we know?
How can we calculate this efficiently?
Efficient Approximations

Multiple approximate checks eliminate candidates more efficiently than a single, accurate check

Checks (in order):

- Ray-Plane intersection
- Ray-Triangle intersection
- Position of intersection on triangle
Ray-Plane Collision Detection

Plane specified by:
- point on plane
- plane normal

In-class Activity:
Use the plane equation to determine where point Q is based on the ray origin P and direction \( \vec{d} \)
\[ N \cdot Q + d = 0 \]
\[ N \cdot (P + \vec{d}(t)) + d = 0 \]
\[ N \cdot P + N \cdot \vec{d}(t) = -d \]
\[ N \cdot \vec{d}(t) = - (d + N \cdot P) \]
\[ t = - \frac{N \cdot P + d}{N \cdot \vec{d}} \]
\[ Q = P + \vec{d}(t) \]
Ray-Triangle Collision Detection

• Intersect ray with triangle’s supporting plane:
  \[ N = (A - C) \times (B - C) \]
• Check if inside triangle
How to Check if Inside?

- Using triangle edges
- Using barycentric coordinates
- Using projections
Ray-Triangle Collision Detection

Normal:

\[ \hat{n} = \frac{(B-A) \times (C-A)}{\| (B-A) \times (C-A) \|} \]
Idea: if P inside, must be left of line AB

How can we determine if point Q is to the left or right of a triangle edge?

Ray-Triangle Collision Detection

Normal:

\[ \hat{n} = \frac{(B-A) \times (C-A)}{\| (B-A) \times (C-A) \|} \]
Intuition

Cross product will point in opposite direction if point Q is to the right
Therefore dot product will now be negative
($\cos\theta < 0$ if $\theta > 90^\circ$)
Inside-Outside Test

Check that point Q is to the left of all edges:

\[ ((B-A) \times (Q-A)) \cdot n \geq 0 \]
\[ ((C-B) \times (Q-B)) \cdot n \geq 0 \]
\[ ((A-C) \times (Q-C)) \cdot n \geq 0 \]

If it passes all three tests, it is inside the triangle.
Caveat

The inside-outside method will have some corner cases that result in unshaded fragments on the first assignment!
Barycentric Coordinates

Affine frame defined by origin \((t = c)\) and vectors from \(c\) \((v = a-c, w = b-c)\)

Point can be represented using area coordinates \(\alpha, \beta, \gamma\) (ratio between sub-area and total triangle area):

\[ Q = \alpha a + \beta b + \gamma c \]
Barycentric Coordinates

What does these area coordinates tell us?

\[ \alpha = \frac{A_a}{A} \]
\[ \beta = \frac{A_b}{A} \]
\[ \gamma = \frac{A_c}{A} \]
Barycentric Coordinates

If point Q’s
\(\alpha, \beta, \gamma \geq 0\)
and
\(\alpha + \beta + \gamma = 1\)
then Q is within the triangle!
Barycentric Coordinates

Proportional to lengths of crossproducts:

\[ A_a = \frac{1}{2} \|((C-B) \times (Q-B))\| \]
\[ A_b = \frac{1}{2} \|((A-C) \times (Q-C))\| \]
\[ A_c = \frac{1}{2} \|((B-A) \times (Q-A))\| \]
Beyond Triangle Intersections...

- Barycentric coordinates can interpolate
  - Vertex properties
  - Material properties
  - Texture coordinates
  - Normals

\[ k_d(Q) = \alpha k_d(A) + \beta k_d(B) + \gamma k_d(C) \]

- Used everywhere!
Barycentric Coordinates in 2D

Project down into 2D and compute barycentric coordinates
Möller-Trumbore Triangle Intersect

- Introduced as an optimized triangle-ray intersection test
- Based on the barycentric parameterization
  - Direction of ray intersection from ray origin becomes 3rd axis (uw are barycentric axes)
- Still commonly used

Full details here:
Ray Tracing: Shading

- Shading colors the pixels.
- Color depends on:
  - Object material
  - Incoming lights
  - Angle of viewer
Object Materials

Different materials can behave very differently

• opaque vs translucent vs transparent
• shiny vs dull

We classify different responses to light into “types”
Emissive Lighting

Light generated within material
Diffuse Reflection

Light comes in, bounces out randomly

Typical for “rough” unpolished materials
View angle doesn’t matter
Specular Reflection

Light reflects perfectly

Typical for smooth, “polished” surfaces
General Opaque Materials

Diffuse-specular spectrum:
What About Translucent?

Subsurface Scattering
What About Translucent?

Subsurface Scattering
Refraction
What About Translucent?

Subsurface Scattering
Refraction
Structural Color

... Not today.
Phong Shading Model

We’ll talk about the specific math behind shading models later. For now, let’s focus on the “ray-tracing” aspect of shading…
How do secondary rays work?
Ray Tracing: Shading

Let $I(P, d)$ be the intensity along ray $P + td$

$$I(P, d) = I_{\text{direct}} + I_{\text{reflected}} + I_{\text{transmitted}}$$

- $I_{\text{direct}}$ computed from Phong model
- $I_{\text{reflected}} = k_r I(Q, R)$
- $I_{\text{transmitted}} = k_t I(Q, T)$
Reflection and Transmission

Law of reflection:
\[ \theta_i = \theta_r \]

Snell’s law of refraction:
\[ \eta_i \sin \theta_i = \eta_t \sin \theta_t \]

(\( \eta \) is index of refraction)
What is this effect?
Total Internal Reflection

- Occurs if:
  - \( \eta_i > \eta_t \) (density of current medium > density of other medium)
  - \( \Theta_i > \Theta_c \) (angle of incidence > critical angle)
- Critical angle is an angle of incidence that provides an angle of refraction of 90°
- No transmission occurs — only reflection
Critical Angle in TIR

• If $\theta_t = 90^\circ$, light moves along boundary surface
• If $\theta_t > 90^\circ$, light is reflected within current medium
Light and Shadow Attenuation

Light attenuation:
  • Light farther from the source contributes less intensity

Shadow attenuation:
  • If light source is blocked from point on an object, object is in shadow
  • Attenuation is 0 if completely blocked
  • Some attenuation for translucent objects
Light Attenuation

Real light attenuation: inverse square law

Tends to look bad: too dim or washed out

So, we cheat:

\[ d \text{ is light-to-point distance} \]

Tweak constant & linear terms to taste:

\[ f_{atten}(d) = \frac{1}{a + bd + cd^2} \]
Shooting Shadow Rays

Camera → Image

Light Source → Shadow Ray

Scene Object

View Ray
In-Class Activity

Based on the information discussed in class, write high-level pseudocode to describe the steps in Whitted ray tracing.

Include consideration for ray termination and computing indices of refraction and TIR.