Ray Tracing Basics I

Computer Graphics as Virtual Photography

**Photography:**
- real scene
- camera (captures light)
- photo processing
- Photographic print

**Computer Graphics:**
- 3D models
- camera model (focuses simulated lighting)
- tone reproduction
- synthetic image
Ray Tracing in the real world

- Light is emitted from light source
- Bounces off of the environment
- Assumptions
  - Light travels in straight rays
  - Path of light changes based on object interaction.
  - Can simulate using basic geometry.
- Some light will reach and be focused by camera onto film plane.
  - Lots of light will not!
  - In image synthesis, we are only interested in the light that does

Backwards Ray Tracing

- Light rays are traced backward from the eye (center of projection), through a viewing plane, into scene to see what it hits.
- The pixel is then set to the color values returned by the ray.
- This color is a result of the object hit by the ray.
Ray Tracing - Basics

Sometimes you don’t hit an object
Ray Tracing - Basics

Sometimes you do

Ray Tracing - Basics

- If you do hit an object, additional rays are spawned and sent into world to determine color at intersection point
  - Shadow ray
  - Reflected ray
  - Transmitted ray
Ray Tracing - Basics

- **Shadow ray**
  - Ray spawned toward each light source to see if point is in shadow.
Ray Tracing

- Reflective Ray

Ray Tracing

- Transmitted ray
Ray Tracing

- Ray Tracing incorporates into a single framework:
  - Hidden surface removal
  - Shadow computation
  - Reflection of light
  - Refraction of light
  - Global Specular Interaction
- Extremely elegant and compact
Ray Tracing Basics

- Basic Ray Tracing -- Example

Ray Tracing Assignment

- For Checkpoint 2:
  - Trace rays through camera model
  - Using ray tracing for visible surface determination.

- Questions -- Break
Ray Tracing through the Camera

- Issues
  - Ray Geometry
  - Object-Ray Intersection
  - Projection

Introducing Ray

- Use mathematical description of a ray and objects to determine intersection.
- Parametric representation of a ray:
  - Origin of ray, \( P_o = (x_o, y_o, z_o) \)
  - Direction \( D = (dx, dy, dz) \)
  - \( Ray (\omega) = P_o + \omega D \)
- If \( D \) is normalized, then \( \omega \) will be the distance from origin of the ray.
Ray-Object Intersection

- Most of the computation in ray tracing is determining ray object-intersection
- When a ray intersects an object, we need to know:
  - Point of intersection
  - Normal of surface at point of intersection

Ray-Sphere Intersection

- The Sphere
  - A sphere can be defined by:
    - Center \((x_c, y_c, z_c)\)
    - Radius \(r\)
  - Equation of a point \((x_s, y_s, z_s)\) on a sphere:
    \[
    (x_s - x_c)^2 + (y_s - y_c)^2 + (z_s - z_c)^2 = r^2
    \]
Ray-Sphere Intersection

- Ray - Sphere Intersection
  - Substituting ray equation for \((x_s, y_s, z_s)\)
  - We get:
    \[ A \omega^2 + B \omega + C = 0 \]
  - where
    \[ A = dx^2 + dy^2 + dz^2 \]
    \[ B = 2(dx(x_o - x_c) + dy(y_o - y_c) + dz(z_o - z_c)) \]
    \[ C = (x_o - x_c)^2 + (y_o - y_c)^2 + (z_o - z_c)^2 - r^2 \]

- Using the Quadratic Formula
  \[ \omega = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \]

  - Note: \(\omega\) must be positive, otherwise the intersection is BEHIND the origin of the ray
Ray-Sphere Intersection

Note: If \( \mathbf{D} \) is normalized

\[ A = dx^2 + dy^2 + dz^2 = 1 \] and

\[ \omega = \frac{-B \pm \sqrt{B^2 - 4C}}{2} \]

Ray-Sphere Intersection

- If \( B^2 - 4C \) is:
  - \(< 0\) – no real root, no intersection
  - \( = 0\) – one root, ray intersects at sphere’s surface
  - \( > 0\) – two roots, ray goes through sphere.

Use least positive root
Ray-Sphere Intersection

- Once we found a $\omega_i$ for the point of intersection, the actual point is:
  - $(x, y, z) = (x_0 + dx \cdot \omega_i, y_0 + dy \cdot \omega_i, z_0 + dz \cdot \omega_i)$
- The normal at the point of intersection is:
  - $(x_n, y_n, z_n) = ((x_i - x_c)/r, (y_i - y_c)/r, (z_i - z_c)/r)$
  - (We divide by $r$ to normalize!)

Ray-Plane Intersection

- A plane can be defined by:
  - A normal vector and a point on the plane
- It has the equation
  $$Ax + By + Cz + F = 0$$
- where $P_n = (A, B, C)$ gives the normal and if normalized $(A^2 + B^2 + C^2 = 1)$, $F$ will the shortest distance to the plane from the origin of world.
Ray-Plane Intersection

- Ray - Plane Intersection
  - For plane with equation:
    \[ Ax + By + Cz + F = 0 \]
  - Plug in equation for ray and we get
    \[
    \omega = \frac{-(Ax_o + By_o + Cz_o + F)}{Adx + Bdy + Cdz} = \frac{-(P_n \cdot P_0 + F)}{(P_n \cdot D)}
    \]

Ray-Plane Intersection

- If \((P_n \cdot D)\) is
  - 0 – then ray is parallel to plane, no intersection
- If \(\omega\) is
  - \(< 0\) – then the ray intersects behind the origin of the ray...ignore!
  - \(> 0\) – calculate the point of intersection
Ray-Plane Intersection

- Once we found a $\omega_i$ for the point of intersection, the actual point is:
  - $(x_i, y_i, z_i) = (x_0 + dx \cdot \omega_i, y_0 + dy \cdot \omega_i, z_0 + dz \cdot \omega_i)$
- And we already have the normal at the point of intersection is:
  - $P_n = (A, B, C)$

Ray-Polygon Intersection

- Find the plane in which the polygon sits
- Find the point of intersection between the ray and the plane
- If point of intersection is found, see if it lies within the boundaries of the polygon.
Ray-Polygon Intersection

- Find the plane in which the polygon sits
  - A plane can be defined by:
    - A normal vector and a point
  - And has the equation
    \[ Ax + By + Cz + F = 0 \]
  - where \( P_n = (A, B, C) \) gives the normal and if normalized \( (A^2 + B^2 + C^2 = 1) \), \( F \) will give the shortest distance to the plane from the origin of the world.

Ray-Polygon Intersection

- Find the point of intersection between the ray and the plane
  - Done previously
- See if point of intersection lies within the boundaries of the polygon.
  - One algorithm:
    - Draw line from \( P \) to each polygon vertex
    - Measure angles between lines
      - Recall: \( \mathbf{A} \cdot \mathbf{B} = |\mathbf{A}| |\mathbf{B}| \cos \theta \)
      - If sum of angles between lines is 360°, polygon contains \( P \)
Other Intersections

- To add other geometric primitives to your ray tracer
  - Must mathematically derive the point of intersection between a ray and geometric primitive.

- Questions?

Ray Tracing through the Camera

- Issues
  - Ray Geometry
  - Object-Ray Intersection
  - Projection
Ray Tracing through a camera

1. Set up your scene
   - Determine position / orientation of objects in scene.

2. Spawn a ray and send into scene
   - Define ray direction (remember to normalize)
   - Check for closest intersection
   - Calculate and return color

3. Display or save final image
Introducing Ray

- Use mathematical description of a ray and objects to determine intersection.
- Parametric representation of a ray:
  - Origin of ray, \( P_o = (x_o, y_o, z_o) \)
  - Direction \( D = (dx, dy, dz) \)
  - \( \text{Ray} (\omega) = P_o + \omega D \)
- If \( D \) is normalized, then \( \omega \) will be the distance from origin of the ray.

Graphics Pipeline
Camera Transformations

\[
M = \begin{bmatrix}
  u_x & u_y & u_z - \text{eye}\cdot u \\
  v_x & v_y & v_z - \text{eye}\cdot v \\
  n_x & n_y & n_z - \text{eye}\cdot n \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]

- \((u_x, u_y, u_z)\) are coordinates of unit vector \(u\) w.r.t. world space
- Similar for \(v\), \(n\),
- \((\text{eye})\) is the origin of view space w.r.t. world space
- If ups are aligned, simply use negative eye location values in the fourth column

Graphics Pipeline

1. 3D Object Coordinates
2. Object Transformation
3. 3D World Coordinates
4. 3D World Transformation
5. 3D Eye Coordinates
6. 3D Clipping
7. 2D Screen Coordinates
8. 2D Screen Transformation
9. Window to Viewport Mapping
10. 2D Eye Coordinates
11. Projection
12. 3D Eye Coordinates
Projection

- Note: Projection not required as this will be done as part of the ray tracing process

\[
\begin{bmatrix}
P_u \\
P_v \\
P_n \\
1
\end{bmatrix}
= M
\begin{bmatrix}
p_x \\
p_y \\
p_z \\
1
\end{bmatrix}
\]

Spawning rays through camera

- Coordinate spaces
  - Can do in camera space or world space
  - Camera space
    - Must transform all objects/lights to camera space
  - World space
    - Must transform initial rays to world space
Projection in Camera Space

- The role of cameras can be described as projecting a 3D scene onto a 2D plane

Converting to World Space

\[
\begin{bmatrix}
P_x \\
P_y \\
P_z \\
1
\end{bmatrix}
= M^{-1}
\begin{bmatrix}
P_u \\
P_v \\
P_n \\
1
\end{bmatrix}
\]
Inverting a 4x4 Matrix

- Code samples from
  - Graphics Gems
  - Ken Perlin

- Available on Web
  - Will link on DIARY

Tips – World Space

- Need only transform the location of 1st “pixel” location on image plane and dx, dy, and dz as you move across and down the plane
Tips – Calculating Color

- Find point of intersection
  - Good Safety tip – only consider intersections if they occur past the image plane.
- If intersection
  - Return color
  - Of object

Displaying your image

- You don’t really need the full power of a 3D API to do ray tracing
  - Just need the ability to write color values to pixels
  - Some of the matrix operation routines may be helpful.
Displaying your image

- OpenGL
  - glDrawPixels();
  - Chapter 10, Hill
  - Chapter 8, OpenGL, red book

Displaying your image

- C library
  - Netppm
    - Netpbm is a freeware toolkit/library for manipulation of graphic images, including conversion of images between a variety of different formats.
    - http://netpbm.sourceforge.net/
  - Java
    - Java2D
      - java.awt.image
      - javax.imageio
  - Questions?