Class Topics and Objectives
- Photo-realistic image generation
- Ray Tracing!
- Learn and implement the algorithms needed to create ray traced images
- Serious numerical computing and programming class
- Assumes you know CG I material
- Go to web site

Class Structure
- Weekly lectures & reading assignments
- 7 programming assignments
- Post images and code on the web
- Usually due Sunday 11:59 PM
  - Discuss problems/questions Thursday before
- No tests
- Grad students give presentations
- Paper summary and one question

Grading
- Graduate Section
  - Programming Assignments - 80%
  - In-class Presentation - 10%
  - In-class Assignments - 10%
- Undergraduate Section
  - Programming Assignments - 90%
  - In-class Assignments - 10%
- Late policy
  - 1 week late ➔ 1 letter grade down
  - 2 letter grades down after that
Ray Casting
- Determines visible surfaces by tracing “light” rays from the viewer’s eye to the objects
- View plane is divided by a pixel grid
- The eye ray is fired from the center of projection through each pixel

Ray Tracing
- Extension of ray casting
- Idea: Continue to bounce the ray in the scene
- Shoot rays to light sources
- Simple and powerful
- Reflections, shadows, transparency and multiple light sources

Ray Tracing Diagrams

Slide Credits
- Kevin Suffern - University of Technology, Sydney, Australia
- G. Drew Kessler & Larry Hodges - Georgia Institute of Technology
- Fredo Durand & Barb Cutler - MIT
- Computer Graphics I

Slide Credits
- Kevin Suffern - University of Technology, Sydney, Australia
- G. Drew Kessler & Larry Hodges - Georgia Institute of Technology
- Fredo Durand & Barb Cutler - MIT
- Computer Graphics I
**First Ray-Traced Image**

Whitted 1980

---

**Issues**

- Ray-object intersections
- Complex, hierarchical models (CSG?)
- Transformations
- Camera models
- Recursive algorithms
- Surface physics (shading models)
- Color representations
- Light representations
- Sampling, anti-aliasing and filtering
- Geometric optics
- Acceleration techniques
- Texture mapping

---

The primary ray is defined in world coordinates by:

- The camera location $o$ (the ray origin)
- A unit direction vector $d$

The expression for the primary ray is

$$ p = o + td $$

where $t$ is the ray parameter.

### Ray representation

- Two variants:
  - Parametric
  - Cartesian (normalized to unit)
- Parametric Form
  - $P(t) = o + td$
  - $P(t) = o + t_{min}d$

Note

The vector $d$ must be converted to a unit vector before it is used in the camera ray.
Notice the following difference between parallel and perspective viewing in ray tracing:

In parallel viewing, each primary ray has a different origin, but the same direction.

In perspective viewing, each primary ray has the same origin, but a different direction.

We index the pixels horizontally (from left to right) with

\[ j : 0 \leq j \leq w_{\text{res}} - 1, \]

and vertically (from top to bottom) with

\[ k : 0 \leq k \leq h_{\text{res}} - 1. \]
Calculating Primary Rays

- Given (in world coordinates)
  - Camera (eye point) location \( \mathbf{O} \)
  - Camera view out direction (\( \mathbf{Z} \))
  - Camera view up vector (\( \mathbf{Y} \))
  - Distance to image plane (\( d \))
  - Horizontal camera view angle (\( \theta \))
  - Pixel resolution of image plane (\( h_{\text{res}}, v_{\text{res}} \))
- Calculate set of rays (\( d \)) that equally samples the image plane

Calculate Preliminary Values

- Camera view side direction (\( \mathbf{X} \))
  - \( \mathbf{Z} \parallel \mathbf{Y} \)
- Horizontal length of image plane (\( s_j \))
  - Next slide
- Vertical length of image plane (\( s_k \))
  - \( s_k = s_j \times (v_{\text{res}} / h_{\text{res}}) \)
  - Assume square pixels

Calculating \( s_j \)

- \( h = d \times \tan(\theta/2) \)
- \( s_j = 2h \)
- \( s_j = 2d \times \tan(\theta/2) \)

Position of top left pixel (\( \mathbf{P}_{0,0} \))

- \( \mathbf{O} + d \times \mathbf{Z} \times (s_j/2) \parallel \mathbf{X} + (s_k/2) \parallel \mathbf{Y} \)

All in world coordinates!
Calculate Those Rays!

- \( P_{0,0} + S_j X_v - S_k Y_v \) sweeps out image plane
- \( 0 \leq j \leq S_j; 0 \leq k \leq S_k \)

for \((j=0; j++; j < h_{res})\)
for \((k=0; k++; k < v_{res})\) { 
  \( d_{j,k} = (P_{0,0} + S_j j/(h_{res}-1)) \times X_v \)
  \(- S_k k/(v_{res}-1)) \times Y_v \) - O;
  \( d'_{j,k} = \frac{d_{j,k}}{|d_{j,k}|} \);
  Image[i,j] = ray_trace(\( d'_{j,k} \), Scene);
}
The following three foils illustrate the effect of changing the image resolution on the field of view, when the pixel size is kept the same.

It is common practice in ray tracers to specify the field of view with angles. You can still do this with appropriate conversions.
Parameters

- X and Y resolution of image
- Camera location & direction
- Distance between camera & image plane
- Camera view angle
- Distance between pixels
- These are not independent!
- Goal: Choose your independent variables and calculate your d's

Ray-Sphere Intersection

G. Drew Kessler
Larry Hodges
Georgia Institute of Technology
Ray/Sphere Intersection
(Algebraic Solution)

Ray is defined by $R(t) = R_o + R_d \cdot t$ where $t > 0$.
- $R_o =$ Origin of ray at $(x_o, y_o, z_o)$
- $R_d =$ Direction of ray $[x_d, y_d, z_d]$ (unit vector)

Sphere's surface is defined by the set of points $\{(x_s, y_s, z_s)\}$ satisfying the equation:

$$(x_s - x_c)^2 + (y_s - y_c)^2 + (z_s - z_c)^2 - r_s^2 = 0$$

Center of sphere: $(x_c, y_c, z_c)$
Radius of sphere: $r_s$

Possible Cases of Ray/Sphere Intersection

1. Ray intersects sphere twice with $t > 0$
2. Ray tangent to sphere
3. Ray intersects sphere with $t < 0$
4. Ray originates inside sphere
5. Ray does not intersect sphere

Solving For $t$

Substitute the basic ray equation:
- $x = x_o + x_d \cdot t$
- $y = y_o + y_d \cdot t$
- $z = z_o + z_d \cdot t$

into the equation of the sphere:

$$(x_o + x_d \cdot t - x_c)^2 + (y_o + y_d \cdot t - y_c)^2 + (z_o + z_d \cdot t - z_c)^2 - r_s^2 = 0$$

This is a quadratic equation in $t$: $At^2 + Bt + C = 0$, where
- $A = x_d^2 + y_d^2 + z_d^2$
- $B = 2[x_d(x_o - x_c) + y_d(y_o - y_c) + z_d(z_o - z_c)]$
- $C = (x_o - x_c)^2 + (y_o - y_c)^2 + (z_o - z_c)^2 - r_s^2$

Note: $A = 1$

Relation of $t$ to Intersection

We want the smallest positive $t$ - call it $t_1$.

$\text{Discriminant} = 0$

$$t_1 = \frac{-B}{2A}$$

$\text{Discriminant} < 0$

$$t_1 = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$
**Actual Intersection**

Intersection point, \((x_i, y_i, z_i) = (x_o + x_d \cdot t_i, y_o + y_d \cdot t_i, z_o + z_d \cdot t_i)\)

Unit vector normal to the surface at this point is
\[ N = \left(\frac{x_i - x_c}{r_s}, \frac{y_i - y_c}{r_s}, \frac{z_i - z_c}{r_s}\right) \]

If the ray originates inside the sphere, \(N\) should be negated so that it points back toward the center.

**Summary (Algebraic Solution)**

1. Calculate A, B and C of the quadratic intersection equation
2. Calculate discriminant (If \(< 0\), then no intersection)
3. Calculate \(t_0\)
4. If \(t_0 < 0\), then calculate \(t_1\) (If \(t_1 < 0\), no intersection point on ray)
5. Calculate intersection point
6. Calculate normal vector at point

Helpful pointers:
- Precompute \(r_s^2\)
- Precompute \(1/r_s\)
- If computed \(t\) is very small then, due to rounding error, you may not have a valid intersection
**Barycentric definition of a triangle**

- \( P(\alpha, \beta, \gamma) = \alpha a + \beta b + \gamma c \)
  - with \( \alpha + \beta + \gamma = 1 \)
  - \( 0 < \alpha < 1 \)
  - \( 0 < \beta < 1 \)
  - \( 0 < \gamma < 1 \)

**Given P, how can we compute \( \alpha, \beta, \gamma \)?**

- Compute the areas of the opposite subtriangle
  - Ratio with complete area
    - \( \alpha = \frac{A_2}{A} \)
    - \( \beta = \frac{A_3}{A} \)
    - \( \gamma = \frac{A_1}{A} \)

Use signed areas for points outside the triangle.

**Simplify**

- Since \( \alpha + \beta + \gamma = 1 \)
  - we can write \( \alpha = 1 - \beta - \gamma \)
- \( P(\beta, \gamma) = (1 - \beta - \gamma) a + \beta b + \gamma c \)
- \( P(\beta, \gamma) = a + \beta (b-a) + \gamma (c-a) \)

**Simplify**

- Non-orthogonal coordinate system of the plane
How do we use it for intersection?

- Insert ray equation into barycentric expression of triangle
- \( P(t) = a + \beta(b-a) + \gamma(c-a) \)
- Intersection if \( \beta + \gamma < 1 \); \( 0 < \beta \) and \( 0 < \gamma \)

Intersection

- \( R_x + D_x = a_x + \beta(b_x-a_x) + \gamma(c_x-a_x) \)
- \( R_y + D_y = a_y + \beta(b_y-a_y) + \gamma(c_y-a_y) \)
- \( R_z + D_z = a_z + \beta(b_z-a_z) + \gamma(c_z-a_z) \)

Matrix form

\[
\begin{bmatrix}
\alpha_1 - b_1 & a_1 - c_1 & D_x & \gamma \\
\alpha_2 - b_2 & a_2 - c_2 & D_y & \gamma \\
\alpha_3 - b_3 & a_3 - c_3 & D_z & \gamma \\
\end{bmatrix}
\]

Matrix A

Cramer’s rule

- \( | | \) denotes the determinant

\[
\begin{bmatrix}
\alpha_1 - b_1 & a_1 - c_1 & D_x \\
\alpha_2 - b_2 & a_2 - c_2 & D_y \\
\alpha_3 - b_3 & a_3 - c_3 & D_z \\
\end{bmatrix}
\]

- \( |A| \) \( \rightarrow \) determinant of matrix A
Calculate Intersection Point
- Is \( n + b \leq 1 ? \)
- Are \( n \) and \( b \) both non-negative?
- Is \( t \) non-negative?
- If so, you’ve got an intersection!
- \( P = R + tD \)

Modular Functionality
- Read and write image files
- Create hierarchical geometric models
- Support several geometric primitives
- Geometric calculations & parameters
  - Ray-object intersections
  - Normals
  - Bounding boxes
  - Color & surface properties
  - Texture maps
- Intersect arbitrary ray with scene
  - Stop at first intersection (shadow rays)
- Adaptive sampling of image plane
- Light properties

Design Your Ray Tracer!
- “Novice programmers often neglect the design phase, instead diving into coding without giving thought to the evolution of a piece of software over time. The result is a haphazard, poorly modularized code which is difficult to maintain and modify. A few minutes of planning short-term and long-term goals at the beginning is time well spent.”
  - Paul Heckbert, “Writing a Ray Tracer”
- Read this chapter!
Progression of Assignments
- Basic ray tracer
  - Triangle/sphere intersection. No shading.
- Hierarchical geometric model
  - Triangle mesh. Still no shading.
- Simple shading and point light sources
- Acceleration techniques
- Adaptive super-sampling/anti-aliasing
- Shadows and reflections
- 2D/3D texture mapping
- No transparency/refraction :-(

Wrap Up
- First programming assignment
  - Due 10/10/04
  - Go to web page
- Grad students need to pick a presentation date and paper for next week