Advanced Rendering Techniques

Dr. David Breen
Online
Wednesday 6PM → 8:50PM

Presentation 1
3/30/22
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/ICG material
- Learn about other rendering algorithms
Class Structure

- Weekly lectures & reading assignments
- 7 regular programming assignments
- 1 extra credit assignment
- Post images on the web
  - E-mail the URL to david@cs.drexel.edu
- Upload code to Bb Vista
- Final exam on material *not* covered by assignments
Grading

- Programming Assignments - 90%
- Final Exam – 10%
- Late policy
  - 1 point/day
  - Maximum 5 points off
Go To Web Sites

- Class web site
- Previous pictures web site
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
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

Fig. 11. An eye ray $E$ propagated through a scene. Many of the intersections spawn reflected, transmitted, and shadow rays.

Fig. 12. The ray tree in schematic form.

The Ray Tree
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

$$p = o + t \ d$$

where $t$ is the ray parameter.

Ray representation

- Two vectors:
  - Origin
  - Direction (normalized is better)
- Parametric line
  - $P(t) = R + t \ D$
Note

The vector $d$ must be converted to a unit vector before it is used in the camera ray.
Left-handed system!
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

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

and vertically (from top to bottom) with

\[ k : 0 \leq k \leq v_{\text{res}} - 1. \]

The \((x_p, y_p)\) coordinates of the lower left corner of the \((j, k)\) pixel are
Calculating Primary Rays

Center of projection

Window
Calculating Primary Rays

- Given (in world coordinates)
  - Camera (eye point) location $\mathbf{O}$
  - Camera view out direction $(\mathbf{Z}_v)$
  - Camera view up vector $(\mathbf{V}_{up})$
  - Distance to image plane (d)
  - Horizontal camera view angle ($\theta$)
  - Pixel resolution of image plane ($h_{res}, v_{res}$)

- Calculate set of rays ($\mathbf{d}$) that equally samples the image plane
Calculate Preliminary Values

- Camera view side direction ($X_v$)
  - $X_v = Z_v \times \text{Vup}$ (left-handed system)
- Make sure that $Y_v$ is orthogonal to $X_v$ & $Z_v$
  - $Y_v = X_v \times Z_v$
- Be sure to normalize $X_v$, $Y_v$ & $Z_v$
  - Horizontal length of image plane ($s_j$)
  - Next slide
Calculating $s_j$

- $h/d = \tan(\theta/2)$
- $h = d \cdot \tan(\theta/2)$
- $s_j = 2h$
- $s_j = 2d \cdot \tan(\theta/2)$
Calculate Preliminary Values

- Vertical length of image plane \((s_k)\)
  
  - \(s_k = s_j \cdot \left(\frac{v_{\text{res}}}{h_{\text{res}}}\right)\)
  
  - Assume square pixels
Calculate Preliminary Values

- Position of top left pixel \((P_{0,0})\)
- \(\mathbf{O} + d \times \mathbf{Z}_v - (S_j/2) \times \mathbf{X}_v + (S_k/2) \times \mathbf{Y}_v\)

All in world coordinates!
Calculate Those Rays!

- \( \mathbf{P}_{0,0} + \alpha \mathbf{X}_v - \beta \mathbf{Y}_v \) sweeps out image plane
- \( 0 \leq \alpha \leq S_j; 0 \leq \beta \leq S_k \)

for (j=0; j++; j < h_{res})
  for (k=0; k++; k < v_{res}) {
    \[ \mathbf{d}_{j,k} = (\mathbf{P}_{0,0} + S_j*(j/(h_{res}-1)) \times \mathbf{X}_v - S_k*(k/(v_{res}-1)) \times \mathbf{Y}_v) - \mathbf{O}; \]
    \[ \mathbf{d}'_{j,k} = \mathbf{d}_{j,k} / \| \mathbf{d}_{j,k} \|; \]
    Image[j,k] = ray_trace(\mathbf{O}, \mathbf{d}'_{j,k}, \text{Scene});
  }

Calculate Those Rays!

for \( j = 0; j < h_{\text{res}} \)  
  for \( k = 0; k < v_{\text{res}} \) {  
    \[
    d_{j,k} = (P_{0,0} + S_j \times (j / (h_{\text{res}} - 1)) \times X_v \nonumber \\
    - S_k \times (k / (v_{\text{res}} - 1)) \times Y_v) - O; 
    \]
    \[
    d'_{j,k} = d_{j,k} / \| d_{j,k} \|; \quad \nonumber 
    \]
    Image[j,k] = ray_trace(O, d'_{j,k}, Scene);  
}
Perspective projection

In ray tracing we do not need to explicitly use the perspective projection, because it is built into the primary rays.

As these emanate from camera position, this acts as the centre of projection.
Size of the pixels

What effect does changing the physical size of the pixels have on the image?

For a specified image resolution \((h_{\text{res}}, v_{\text{res}})\), the size of the window is proportional to the size \(s\) of the pixels.

For a fixed view plane distance \(d\), the field of view is proportional to \(s\).

For fixed \(s\), the size of the window is proportional to \(h_{\text{res}}\) and \(v_{\text{res}}\).

Increasing \(h_{\text{res}}\) and \(v_{\text{res}}\) increases the field of view of the camera, provided \(s\) and \(d\) are kept the same.

Some of these effects are illustrated in the following figures.
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.
The three windows superimposed
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
I recommend setting …

- X and Y resolution of image
  - \((h_{\text{res}}, v_{\text{res}})\)
- Camera location & orientation
  - \(\mathbf{O} \& \mathbf{Z}_v \& \mathbf{V}_{\text{up}}\)
- Distance between camera & image plane
  - \(d\) (a positive scalar, e.g. 10)
- Camera view angle
  - \(\theta\)
Ray-Sphere Intersection

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

Ray is defined by $ R(t) = R_0 + R_d*t $ where $ t > 0 $.  

- $ R_0 = $ 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:

\[ (x_0 + x_d t - x_c)^2 + (y_0 + y_d t - y_c)^2 + (z_0 + z_d t - z_c)^2 - r_s^2 = 0 \]

This is a quadratic equation in \( t \): \( A t^2 + B t + C = 0 \), where

\[ A = x_d^2 + y_d^2 + z_d^2 \]
\[ B = 2[x_d(x_0 - x_c) + y_d(y_0 - y_c) + z_d(z_0 - z_c)] \]
\[ C = (x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r_s^2 \]

Note: \( A = 1 \)
Relation of $t$ to Intersection

We want the smallest positive $t$ - call it $t_i$

Discriminant $= 0$

$$t_0 = \left(\frac{-B - \sqrt{B^2 - 4AC}}{2}\right)$$

$$t_1 = \left(\frac{-B + \sqrt{B^2 - 4AC}}{2}\right)$$

Discriminant $< 0$
Actual Intersection

Intersection point,

\[(x_i, y_i, z_i) = (x_o+x_d*t_i, y_o+y_d*t_i, z_o+z_d*t_i)\]

Unit vector normal to the surface at this point is

\[N = [(x_i - x_c) / r_s, (y_i - y_c) / r_s, (z_i - z_c) / r_s]\]

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
Ray-Triangle Intersection

Fredo Durand
Barb Cutler
MIT
Barycentric definition of a plane

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

[Moebius, 1827]
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
  \[
  \alpha = \frac{A_a}{A}, \quad \beta = \frac{A_b}{A} \quad \gamma = \frac{A_c}{A}
  \]

Use signed areas for points outside the triangle.
Simplify

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

- $P(\beta, \gamma) = (1 - \beta - \gamma) \ a + \beta b + \gamma c$
- $P(\beta, \gamma) = a + \beta(b-a) + \gamma(c-a)$
- 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 $\beta + \gamma \leq 1; \ 0 \leq \beta$ and $0 \leq \gamma$
Intersection

- \( R_x + tD_x = a_x + \beta (b_x - a_x) + \gamma (c_x - a_x) \)
- \( R_y + tD_y = a_y + \beta (b_y - a_y) + \gamma (c_y - a_y) \)
- \( R_z + tD_z = a_z + \beta (b_z - a_z) + \gamma (c_z - a_z) \)
Matrix form

- $R_x + tD_x = a_x + \beta (b_x - a_x) + \gamma (c_x - a_x)$
- $R_y + tD_y = a_y + \beta (b_y - a_y) + \gamma (c_y - a_y)$
- $R_z + tD_z = a_z + \beta (b_z - a_z) + \gamma (c_z - a_z)$

$$\begin{bmatrix} a_x - b_x & a_x - c_x & D_x \\ a_y - b_y & a_y - c_y & D_y \\ a_z - b_z & a_z - c_z & D_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - R_x \\ a_y - R_y \\ a_z - R_z \end{bmatrix}$$
Cramer’s rule

\[ \begin{vmatrix} a_x - R_x & a_x - c_x & D_x \\ a_y - R_y & a_y - c_y & D_y \\ a_z - R_z & a_z - c_z & D_z \end{vmatrix} \]

\[ \begin{vmatrix} a_x - b_x & a_x - R_x & D_x \\ a_y - b_y & a_y - R_y & D_y \\ a_z - b_z & a_z - R_z & D_z \end{vmatrix} \]

\[ \begin{vmatrix} a_x - b_x & a_x - c_x & a_x - R_x \\ a_y - b_y & a_y - c_y & a_y - R_y \\ a_z - b_z & a_z - c_z & a_z - R_z \end{vmatrix} \]

\( |A| \rightarrow \text{determinant of matrix } A \)
Advantage

- Efficient
- Store no plane equation
- Get the barycentric coordinates for free
  - Useful for interpolation, texture mapping
Calculate Intersection Point

- Are $\beta$ and $\gamma$ both non-negative?
- Is $\beta + \gamma \leq 1$?
- Is $t$ non-negative?
- If so, you’ve got an intersection!
- $P = R + tD$
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.”

Modular Functionality

- Read and write image files
- Create hierarchical geometric models with transformations
- 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)
- Recursive generation and summation of rays
- Adaptive sampling of image plane
- Light properties
Possible Software Structure

P. Heckbert
Writing a Ray Tracer
Progression of Assignments

- Basic ray tracer with spheres and triangles
  - Triangle/sphere intersection. No shading.
- Simple shading and point light sources
- Acceleration techniques
- Adaptive super-sampling/anti-aliasing
- Shadows and reflections
- Transparency and refraction
- 2D/3D texture mapping
Wrap Up

- First programming assignment
  - Due 4/10/22
  - Go to web page
- Questions?