

Ray Tracing I - Basic Algorithm

Lecture #02: Tuesday, 21 January 2003
Lecturer: Greg Humphreys
Scribe: John Tran
Reviewer: Joe Mama

This lecture covers the basics of ray-surface intersections. We will pretend there is only one object we are intersecting with, and next lecture, we will come up with acceleration techniques for large numbers of objects.

1 Books

This is a list of books about Ray Tracing that could be useful for further reading.

- *Introduction to Ray Tracing*, by Andrew Glassner
- *Radiosity and Realistic Image Synthesis*, by Michael Cohen
- *Realistic Ray Tracing*, by Pete Shirley
- *Radiosity and Global Illumination*, by François Sillion and Claude Puech
- *Realistic Image Synthesis Using Photon Mapping*, by Henrik Wann Jensen

2 Background

Ray Tracing is not a new idea! The Greeks argued about whether light rays came from the source or the eye. The latter argument was based on the idea of *feelers* that were sent from the eye, went into space and *touched* objects, and return. This is how we do ray tracing today.

The Greeks also came up with the idea of *Plenoptic modelling*, which came from the Greek work “plenom,” which was the medium through which the *feelers* moved. Nowadays, “plenom” means “fullness.” (ie. plentiful)

Also in physics, Gauss used to trace rays through lenses to come up with his famous lens equations.

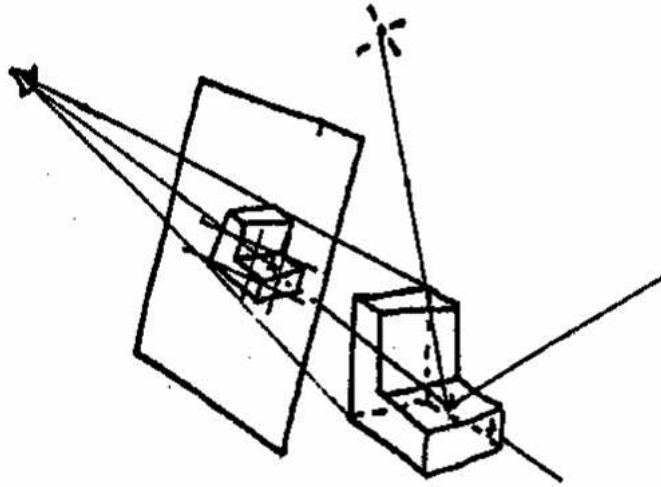


Figure 1: Ray casting

3 Three key properties of light

These three assumptions about light make our lives easier for graphics.

1. Light travels in straight lines. Exceptions include bending in a very strong gravitational field or media having a continuously varying index of refraction.
2. Rays do not interfere with each other. In other words, photons do not collide.
3. Recipricosity. The physics of light path traversal is invariant under direction. Physicists trace from the source to the eye, but in graphics we trace from the eye to the source.

4 First Big Idea in Ray Tracing

In 1968, Arthur Appel came up with the idea of *ray casting* which is essentially unchanged today.

1. Generate an image by sending one ray per pixel and figuring out which object each ray hits first
2. Check for shadows by sending a ray to the light

Why trace from the eye? Computational efficiency! You generate less rays this way, and you only generate rays which are necessary for the final image. Also, you only have one camera, even though you could have multiple lights in your scene.

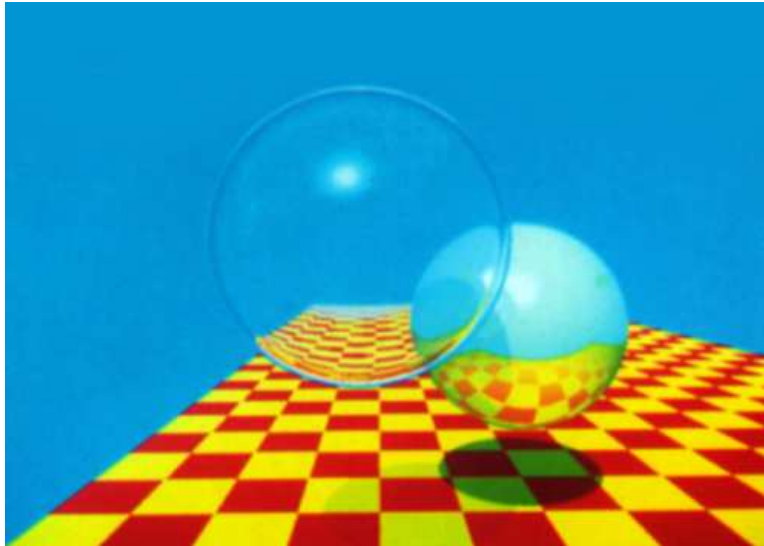


Figure 2: Recursive ray tracing (reflection and refraction)

5 Second Big Idea in Ray Tracing

In 1979, Turner Whitted came up with *recursive ray tracing*, which then allowed reflection and refraction. Figure 2 shows an image from that paper.

Note that the glass sphere actually simulates a hollow glass sphere in real life.

6 Structure of a Ray Tracer

Some of the first ray tracers were very small. You can even write a ray tracer on the back of a business card! LRT is about 20,000 lines of code.

The basic ray tracing architecture looks something like figure 3.

The scene is described as a Renderman Interface Bytestream (RIB) ASCII text file. This scene description is parsed into shapes, lights, materials, and camera parameters. The camera generates the eye rays and intersects them with the shapes that were also generated from the parser.

If there is a hit, a Surf is returned to the integrator. Surf is a term from differential geometry that represents a little piece of the surface that is hit that contains enough information about the local geometry to shade it. This way, the geometry is separated from the shading. The surf contains the surface normal, position, tangent vectors, and parametric texture coordinates.

The surf is passed to the integrator class, which returns a color to the camera, as well as possibly generating reflecting or refracting rays. This class is called an “integrator” because it integrates over the incident lighting in the scene, which usually reduces to a

Ray Tracing Architecture

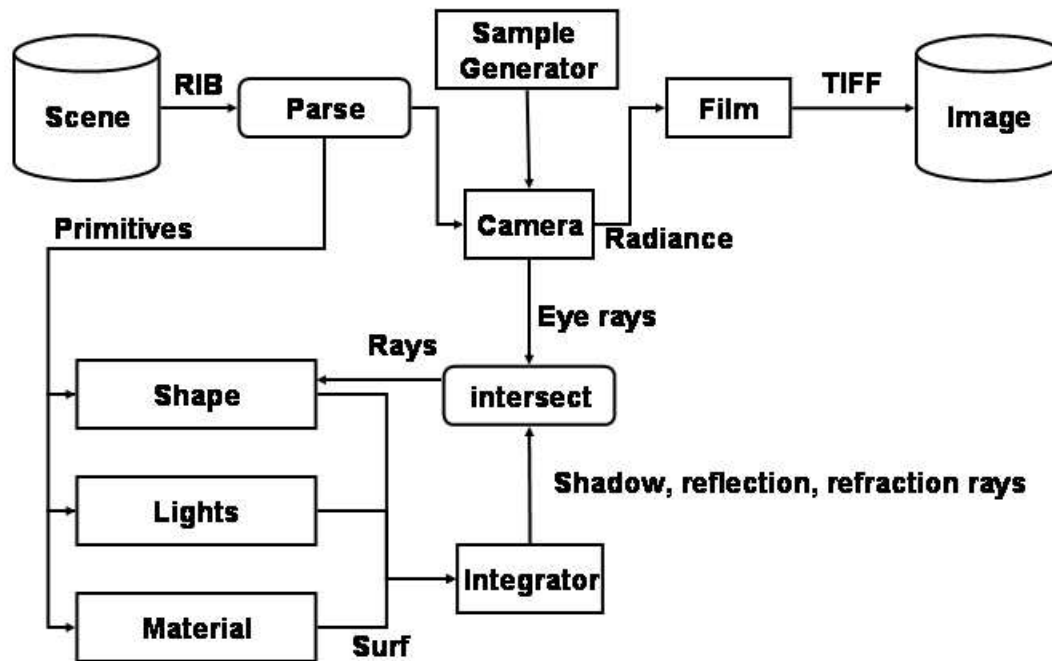


Image Synthesis Lecture 2

Greg Humphreys, Spring 2003

Figure 3: LRT Ray Tracing Architecture

summation.

The colors are passed from the camera to the film, which has certain response characteristics to imitate film. The film then writes out a TIFF image.

The sample generator decides where the rays go. This class, along with the integrator, are the sources of complexity in the system, whereas most of the other classes are fairly straightforward.

7 Intersecting Rays with Objects

7.1 Ray-Plane Intersection

What is a ray? A ray is a semi-infinite line with an origin O and a direction D . Points on this line satisfy the equation

$$P = O + tD, 0 \leq t \leq \infty \quad (1)$$

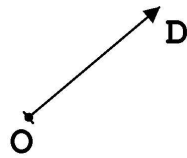


Figure 4: Ray

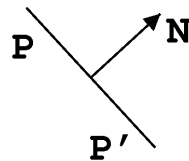


Figure 5: Plane

What is a plane? A plane can be described by a point and a normal. The plane equation ($ax + by + cz + d = 0$) is enough to describe a plane algebraically.

In vector form, we can describe the ray as:

$$(P - P') \bullet N = 0 \quad (2)$$

To intersect, P must satisfy both equations. Substituting Equation 1 into Equation 2, we get

$$(P - P') \bullet N = ((O + tD) - P') \bullet N = 0 \quad (3)$$

$$(O - P') \bullet N = (O - P') \bullet N + tD \bullet N = 0 \quad (4)$$

$$t = -\frac{(O - P') \bullet N}{D \bullet N} \quad (5)$$

Plugging this t back into the plane equation, we can find the intersection point. If $D \bullet N = 0$, then the line is parallel to the plane. If $t < 0$, then there is no intersection.

7.2 Ray-Slab Intersection

A slab is defined as the space between two parallel planes. To determine the intersection, simply find the intersection of the ray with both planes and take the lowest t that is still positive.

One thing to note is that the normals for both planes will be the same. Therefore, $D \bullet N$ need only be computed once.

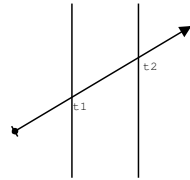


Figure 6: Ray-Slab Intersection

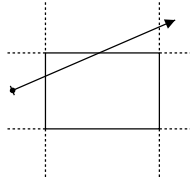


Figure 7: Ray-Box Intersection

7.3 Ray-Box Intersection

A box is equivalent to two slabs.

For a more general case, let us look at the ray-convex polyhedron intersection

7.4 Ray-Convex Polyhedron Intersection

With a convex polyhedron, we have to check the ray with all planes. How do we find the *first* point of intersection? We could substitute the intersections into the plane equations to determine which side of the planes the intersections lie in, or we could classify the intersection points as entering or leaving. This can be done by looking at the sign of the ray direction dotted with the normal of the plane, $D \bullet N$. For example, we can classify $D \bullet N < 0$ as entering.

$$\text{entering} : D \bullet N < 0 \quad (6)$$

$$\text{exiting} : D \bullet N \geq 0 \quad (7)$$

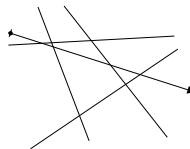


Figure 8: Ray-Convex Polyhedron

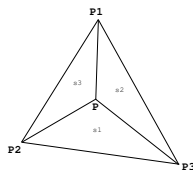


Figure 9: A Triangle

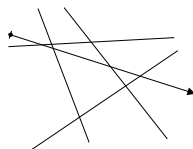


Figure 10: Ray-Polygon

The last entering point is the first point of intersection t_1 , and the first exiting point is t_2 .

Again, $D \bullet N$ is already computed from the plane intersection, so this computation is done basically for free.

7.5 Ray-Triangle Intersection

We can represent a triangle by its *barycentric* coordinates.

$$P = s_1P_1 + s_2P_2 + s_3P_3 \quad (8)$$

$$0 \leq s_i \leq 1 \quad (9)$$

$$\sum s_i = 1 \quad (10)$$

The triangle defines a plane. So we can intersect the ray with the plane and find the point P , then solve for the barycentric coordinates to determine if P is inside the triangle, checking if the barycentric coordinates are valid. To do this, we write the triangle equation as a matrix equation and solve for s_1 , s_2 , and s_3 .

$$\begin{bmatrix} P_1 & P_2 & P_3 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} P \end{bmatrix} \quad (11)$$

LRT uses something slightly different to test ray-triangle intersections.

7.6 Ray-Polygon Intersection

1. Find intersection with plane defined by polygon

```

inside( vert *v, int n, float x, float y )
{
    int cross=0; float x0, y0, x1, y1;

    x0 = v[n-1].x - x;
    y0 = v[n-1].y - y;
    while (n--) {
        x1 = v->x - x;
        y1 = v->y - y;
        if (y0 > 0) {
            if (y1 <= y1*x0) cross++;
        } else {
            if (y1 > 0 && x0*y1 > y0*x1) cross++;
        }
        x0 = x1; y0 = y1; v++;
    }
    return cross & 1;
}

```

Figure 11: Point In Polygon Intersection

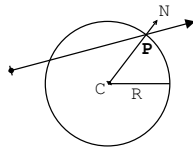


Figure 12: Ray-Sphere

2. Test point-in-polygon by first projecting the polygon on to the x-y plane (or any other plane depending on the orientation of the polygon), and then doing a 2D point-in-polygon test

For the point-in-polygon test, we use the Jordan-Curve theorem, which states that if we have a point in a polygon and we shoot a ray in the plane of that polygon, then that ray will cross the edges of the polygon an odd number of times. The following is an implementatino of this algorithm.

8 Ray-Sphere Intersection

What is a sphere? We can write the equation for a sphere as:

$$(P - C)^2 - R^2 = 0 \quad (12)$$

Substituting the ray equation in, we get:

$$((O + tD) - C)^2 - R^2 = 0 \quad (13)$$

This is a quadratic equation in t :

$$at^2 + bt + c = 0 \quad (14)$$

where $a = D^2$, $b = 2(O - C) \cdot D$, and $c = (O - C)^2 - R^2$.

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (15)$$

The determinant then tells you how many intersections there are.

- $b^2 - 4ac < 0$: no intersections
- $b^2 - 4ac = 0$: one intersection, the ray is tangent
- $b^2 - 4ac > 0$: two intersections

We can save computation by computing the root which requires no subtraction first, and using the fact that $r_1 r_2 = \frac{c}{A}$. (LRT does this)

$$t_0 = \frac{q}{A} \begin{cases} q = \frac{1}{2}(B - \sqrt{B^2 - 4ac}) & \text{if } B < 0 \\ q = \frac{1}{2}(B + \sqrt{B^2 - 4ac}) & \text{if } B \geq 0 \end{cases} \quad (16)$$

$$t_0 t_1 = \frac{C}{A} \implies t_1 = \frac{C}{q} \quad (17)$$

In LRT, we need more than just the hit point. We need the surface normal N , surface tangents S, T , and surface parameters u, v .

For a sphere, we can find these by noting that u and v are the latitude and longitude.

$$N = P - C \quad (18)$$

$$\begin{aligned} x &= \cos \theta \cos \phi & \theta &= \arctan(x, y) \\ y &= \sin \theta \cos \phi & \phi &= \arcsin(z) \\ z &= \sin \phi \end{aligned}$$

9 Ray-Implicit Surface Intersection

What is the pattern in all of this? To intersect a ray with a surface, just come up with some algebraic or vector form for that surface and plug the ray into that surface.

$$f(x, y, z) = 0 \quad (19)$$

$$\begin{aligned}x &= x_O + x_D t \\y &= y_O + y_D t \\z &= z_O + z_D t\end{aligned}$$

$$f^*(t) = 0 \tag{20}$$

f^* is a univariate equation (x_O and x_D are the same for any given ray). We can solve for t using any root finder. Note that we have to find *all* roots, not just one.

10 Ray-Algebraic Surface Intersection

An algebraic surface is an implicit surface whose function is a *polynomial*. In other words, f is a polynomial, and f^* is also a polynomial.

There are lots of specialized techniques for polynomial root finding when you know that your function is a polynomial.

11 Not so much of a Summary

Jim Kajiya was the first to note that once we have an object-oriented interface to geometry, we can ray trace *anything!*