Ray Tracing I: Basics

Today

■ **pbrt overview**
■ **Basic algorithms**
■ **Ray-surface intersection**
■ **Efficiency**
■ **Floating-point error**

Next lecture

■ Accelerating ray tracing of large numbers of geometric primitives
Light Rays

Three ideas about light rays

1. Light travels in straight lines (mostly)
2. Light rays do not interfere with each other if they cross (light is invisible!)
3. Light rays travel from the light sources to the eye (but the physics is invariant under path reversal - reciprocity).
Ray Tracing in Computer Graphics

Appel 1968 - Ray casting

1. Generate an image by casting one ray per pixel
2. Check for shadows by sending a ray to the light
“An improved
Illumination model
for shaded display”
T. Whitted,
CACM 1980

1. Always send ray
to the light source (unless
glass or mirror)

2. Recursively generate
reflected rays (mirror) and
transmitted rays (glass)

Time:
- VAX 11/780 (1979) 74m
- PC (2006) 6s
- GPU (2012) 1/30s
PBRT Overview
Table 1.1: Main Interface Types. Most of pbrt is implemented in terms of 10 key abstract base classes, listed here. Implementations of each of these can easily be added to the system to extend its functionality.

<table>
<thead>
<tr>
<th>Base class</th>
<th>Directory</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape</td>
<td>shapes/</td>
<td>3.1</td>
</tr>
<tr>
<td>Aggregate</td>
<td>accelerators/</td>
<td>4.2</td>
</tr>
<tr>
<td>Camera</td>
<td>cameras/</td>
<td>6.1</td>
</tr>
<tr>
<td>Sampler</td>
<td>samplers/</td>
<td>7.2</td>
</tr>
<tr>
<td>Filter</td>
<td>filters/</td>
<td>7.8</td>
</tr>
<tr>
<td>Material</td>
<td>materials/</td>
<td>9.2</td>
</tr>
<tr>
<td>Texture</td>
<td>textures/</td>
<td>10.3</td>
</tr>
<tr>
<td>Medium</td>
<td>media/</td>
<td>11.3</td>
</tr>
<tr>
<td>Light</td>
<td>lights/</td>
<td>12.2</td>
</tr>
<tr>
<td>Integrator</td>
<td>integrators/</td>
<td>1.3.3</td>
</tr>
</tbody>
</table>
Figure 1.17: Class Relationships for the Main Rendering Loop in the SamplerIntegrator::Render() Method in core/integrator.cpp. The Sampler provides a sequence of sample values, one for each image sample to be taken. The Camera turns a sample into a corresponding ray from the film plane, and the Li() method implementation computes the radiance along that ray arriving at the film. The sample and its radiance are given to the Film, which stores their contribution in an image. This process repeats until the Sampler has provided as many samples as are necessary to generate the final image.
Figure 1.19: Class Relationships for Surface Integration. The main rendering loop in the SamplerIntegrator computes a camera ray and passes it to the $Li()$ method, which returns the radiance along that ray arriving at the ray’s origin. After finding the closest intersection, it computes the material properties at the intersection point, representing them in the form of a BSDF. It then uses the Lights in the Scene to determine the illumination there. Together, these give the information needed to compute the radiance reflected back along the ray at the intersection point.
Mirror, depth 2
Mirror, depth 3
Mirror, depth 10
Glass, depth 1
Glass, depth 2
Glass, depth 3
Glass, depth 10
Bidirectional paths
Shape Interface (Simplified)

class Shape {
    public:
        Bounds3f ObjectBound() const;
        Bounds3f WorldBound() const;
        bool Intersect(const Ray &ray, Float *tHit,
                        SurfaceInteraction *isect,
                        bool testAlphaTexture) const;
        bool IntersectP(const Ray &ray,
                        bool testAlphaTexture);
        Float Area() const;
        // ...
};
class SurfaceInteraction {
    Point3f p;
    Float time;
    Vector3f pError;
    Normal3f n;

    Point2f uv;
    Vector3f dpdu, dpdv;
    Normal3f dndu, dndv;

    struct {
        Normal3f n;
        Vector3f dpdu, dpdv;
        Normal3f dndu, dndv;
    } shading;

    // ...
};
Ray-Surface Intersection
Ray-Plane Intersection

Ray

\[ r(t) = o + t\overrightarrow{d} \]

Want \( t \) where the ray intersects the plane

Plane

\[ (p - p') \cdot \overrightarrow{n} = 0 \]

\[ ax + by + cz + d = 0 \]
Ray-Plane Intersection

Ray: \( r(t) = o + t \vec{d} \)

Plane: \( (p - p') \cdot \vec{n} = 0 \)

Substitute ray equation into plane equation:

\[
(p - r(t)) \cdot \vec{n} = (p - (o + t \vec{d})) \cdot \vec{n} = 0
\]

\[
t = -\frac{(o - p) \cdot \vec{n}}{\vec{d} \cdot \vec{n}}
\]
Finding The Closest Intersection

\[ r(0) \]

\[ o \]

\[ r(t) \]

\[ \overrightarrow{d} \]

\[ r(t_0) \]

\[ r(t_1) \]
Optimizing Ray-Box

\[ t = -\frac{(o - p) \cdot \vec{n}}{d \cdot \vec{n}} \]

\[ t = -\frac{o_x - p_x}{d_x} \]
What About Rays Parallel to a Plane?

Math says: \[ t = -\frac{o_x - p_x}{d_x} \]

IEEE Floating Point says:
1. positive num / 0 = +Inf
2. +Inf > all other floats
3. -Inf < all other floats
4. negative num / 0 = -Inf
5. -Inf < all other floats

\[ t = -\frac{o_x - p_x}{0} = \pm\infty \]
Ray-Sphere Intersection

Ray: \[ r(t) = \mathbf{o} + t \mathbf{d} \]

Sphere: \[ \| \mathbf{p} - \mathbf{c} \|^2 - r^2 = 0 \]
\[ (\mathbf{o} + t \mathbf{d} - \mathbf{c})^2 - r^2 = 0 \]

\[ a = \mathbf{d} \cdot \mathbf{d} \]
\[ b = 2(\mathbf{o} - \mathbf{c}) \cdot \mathbf{d} \]
\[ c = ((\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c})) - r^2 \]
Quadrics
The signed area of the parallelogram given by the vectors $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ is given by

$$\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = (x_1y_2) - (x_2y_1)$$

Half of this area is the area of the triangle they specify.
Half of this area is the area of the triangle they specify

\[ \frac{1}{2} \begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{vmatrix} = \frac{1}{2} ((x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)) \]

Area of a triangle with vertices \( p_0 = (x_0, y_0) \) etc.
Barycentric Coordinates

\[ a_0(p) = \text{Area}(p_1, p_2, p) \]
\[ a_1(p) = \text{Area}(p_2, p_0, p) \]
\[ a_2(p) = \text{Area}(p_0, p_1, p) \]

Define barycentric coordinates:

\[ b_i = \frac{a_i}{\text{Area}(p_0, p_1, p_2)}, \quad 0 \leq i \leq 2 \]

\( p \) is inside the triangle if

\[ 0 \leq b_0 \leq 1, \text{ and } 0 \leq b_1 \leq 1 \]
Ray-Triangle Intersection

3 points define a plane: \( p = b_0 p_0 + b_1 p_1 + b_2 p_2 \)

Find ray-plane intersection point

Test whether that point is inside the triangle

\[
\begin{bmatrix}
  p_0 & p_1 & p_2
\end{bmatrix}
\begin{bmatrix}
  b_0 \\
  b_1 \\
  b_2
\end{bmatrix} = \begin{bmatrix}
  p
\end{bmatrix}
\]

\[
\begin{bmatrix}
  b_0 \\
  b_1 \\
  b_2
\end{bmatrix} = \begin{bmatrix}
  p_0 & p_1 & p_2
\end{bmatrix}^{-1} \begin{bmatrix}
  p
\end{bmatrix}
\]

Inside if \( b_0 > 0, b_1 > 0, b_2 > 0 \)
Ray Triangle Intersection

\[ \mathbf{o} + t \mathbf{d} = (1 - b_1 - b_2) \mathbf{p}_0 + b_1 \mathbf{p}_1 + b_2 \mathbf{p}_2 \]

\[
\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\mathbf{s}_1 \cdot \mathbf{e}_1} \begin{bmatrix} \mathbf{s}_2 \cdot \mathbf{e}_2 \\ \mathbf{s}_1 \cdot \mathbf{s} \\ \mathbf{s}_2 \cdot \mathbf{d} \end{bmatrix}
\]

Where

\[
\begin{align*}
\mathbf{e}_1 &= \mathbf{p}_1 - \mathbf{p}_0 \\
\mathbf{e}_2 &= \mathbf{p}_2 - \mathbf{p}_0 \\
\mathbf{s} &= \mathbf{o} - \mathbf{p}_0 \\
\mathbf{s}_1 &= \mathbf{d} \times \mathbf{e}_2 \\
\mathbf{s}_2 &= \mathbf{s} \times \mathbf{e}_1
\end{align*}
\]

[Möller and Trumbore 1997]
Efficiency
Contemporary Bathroom Scene

525M triangles
12.9B ray-triangle tests
~26 tests per triangle
3.2B hit (~25%)
~30 minutes with 8 threads
5m 6s in ray/triangle (17%)
Conserve Memory vs. Coherence

```cpp
class Triangle {
    // ...
    int *v;
    TriangleMesh *mesh;
};

class TriangleMesh {
    Point3f *P;
    // ...
};
```

pbrt Triangle

Alternative Triangle
Additional Techniques

Early outs

- Detect non-intersection partway through
  e.g. compute t before testing inside

Batch intersecting \( m \) rays with \( n \) primitives

- Reduce redundant computation
- Allows SIMD / vector processing
Floating-Point Error
Floating-point Representation

Scientific notation  \[ \pm 1.m \times 2^e \]
- with a fixed sized mantissa (23-bits),
- a limited exponent range (8-bits, e-127),
- signed bit

\[ 2.5 = 1.25 \times 2^1 = 1.01_b \times 2^1 \]
\[ 1/3 \approx 1.0101010101010101010101011_b \times 2^{-2} \]
\[ 0 = ? \]
Floating-point Representation
Roundoff Error
Catastrophic Cancellation
Catastrophic Cancellation

A - B vs A' - B' ...
Effect of Roundoff Error

CS348b Lecture 2
Pat Hanrahan / Matt Pharr, Spring 2019
Effect of Floating-Point Roundoff Error
Round-off Error Remedies

Use double (fp64) rather than float (fp32)
- Can help, but also avoids the root problem
- Increases memory use, reduces performance

Ignore reintersection with the last object hit
- Only works for flat objects (e.g. triangles)
- No help if model has coincident triangles

Have a $t_{\text{min}}$ along ray to ignore close intersections
- What value should $t_{\text{min}}$ take?
Refine Intersection Point, Bound Error

See pbrt 3.9 for details
Ray-Implicit Surface Intersection

Implicit surface

\[ f(x, y, z) = 0 \]

Substitute ray equation

\[ x = o_x + td_x \]
\[ y = o_y + td_y \]
\[ z = o_z + td_z \]

Univariate root finding

\[ f^*(t) = 0 \]

Fedkiw et al.