Lecture 6 - “Path Tracing”

Welcome!
Today's Agenda:

- Introduction
- Path Tracing
Introduction

Previously in Advanced Graphics

The Rendering Equation:

\[ L(s \leftarrow x) = L_E(s \leftarrow x) + \int_A f_r(s \leftarrow x \leftarrow x') L(x \leftarrow x') G(x \leftrightarrow x') \, dA(x') \]

"The light that travels from x to s is the light that x emits plus the light that x reflects."

"three-point formulation"

\[ A: \text{all points } x' \text{ in the scene} \]
\[ f_r: \text{BRDF} \]
\[ L: \text{radiance} \]
\[ G: \text{‘geometry factor’} \]

BRDF: takes irradiance (from \( x' \)), calculated by:

\[ L(x \leftarrow x') G(x \leftrightarrow x') \]
returns radiance (towards \( s \)).
Introduction

Previously in Advanced Graphics

The Rendering Equation (hemispherical formulation):

\[ L_o(x, \omega_o) = L_E(x, \omega_o) + \int_{\Omega} f_r(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos \theta_i \, d\omega_i \]

...which models light transport as it happens in the real world, by summing:

- Direct illumination: \( L_E(x, \omega_o) \)
- Indirect illumination, or reflected light: \( \int_{\Omega} f_r(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos \theta_i \, d\omega_i \)

We used quantities flux \( \Phi \) (joules per second), radiance \( L \) (flux per \( m^2 \) per sr) and irradiance \( E \) (flux per \( m^2 \)). Radiance and irradiance are continuous values.
Introduction

Previously in Advanced Graphics

Particle transport:

As an alternative to discrete flux / radiance / irradiance, we can reason about light transport in terms of particle transport.

- Flux then becomes the number of emitted photons;
- Radiance the number of photons travelling through a unit area in a unit direction;
- Irradiance the number of photons arriving on a unit area.

A BRDF tells us how many particles are absorbed, and how outgoing particles are distributed. The distribution depends on the incident and exitant direction.
Introduction

Previously in Advanced Graphics

Probabilities:

We can also reason about the behavior of a single photon. In that case, the BRDF tells us the *probability* of a photon being absorbed, or leaving in a certain direction.
Introduction

Previously in Advanced Graphics

Turning physics into code:

- ‘Flux’ becomes ‘photons per second’
- Count becomes probability

To complicate things, radiance and irradiance are ‘continuous values’ or ‘densities’:

“At point X we have a density of 50 photons per m²”

“In direction ω₀, we have a flow of 10 photons per steradian”

Likewise, our probabilities are going to be densities.
Introduction

Bidirectional Reflectance Distribution Function

BRDF: function describing the relation between radiance emitted in direction $\omega_o$ and irradiance arriving from direction $\omega_i$:

$$f_r(\omega_o, \omega_i) = \frac{L_o(\omega_o)}{E_i(\omega_i)} = \frac{L_o(\omega_o)}{L_i(\omega_i) \cos \theta_i} = \frac{\text{outgoing radiance}}{\text{incoming irradiance}}$$

Or, if spatially variant:

$$f_r(x, \omega_o, \omega_i) = \frac{L_o(x, \omega_o)}{E_i(x, \omega_i)} = \frac{L_o(x, \omega_o)}{L_i(x, \omega_i) \cos \theta_i}$$

Properties:

- Should be positive: $f_r(\omega_o, \omega_i) \geq 0$
- Helmholz reciprocity should be obeyed: $f_r(\omega_o, \omega_i) = f_r(\omega_i, \omega_o)$
- Energy should be conserved: $\int_\Omega f_r(\omega_o, \omega_i) \cos \theta_o \, d\omega_o \leq 1$
Introduction

Bidirectional Reflectance Distribution Function

The diffuse BRDF is:

\[ f_r(\omega_o, \omega_i) = \frac{\text{albedo}}{\pi} \]

So, for a total irradiance \( E \) at surface point \( x \), the outgoing radiance \( L_o = E_i \frac{\text{albedo}}{\pi} \). Why the \( \pi \)?

Energy conservation: \( E_o \leq E_i \)

Suppose we have a directional light parallel to \( \hat{n} \), with intensity 1. Then: \( E_i = L_i = 1 \). Suppose our BRDF = \( \frac{\text{albedo}}{1} \).

Then, for \( \text{albedo} = 1 \) we get: \( E_o = \int_{\Omega} L_i f_r(\omega_o, \omega_i) \cos \omega_o \, d\omega_o = \int_{\Omega} \cos \omega_o \, d\omega_o \)

Now: \( \int_{\Omega} \cos \omega_o \, d\omega_o = \pi \Rightarrow E_o = \pi E_i. \)
Introduction

Bidirectional Reflectance Distribution Function

Mirror / Perfect specular:
Reflects light in a fixed direction.

For a given incoming direction $\omega_i$, all light is emitted in a single infinitesimal set of directions. The specular BRDF is a Dirac function:

$$f_r(x, \omega_o, \omega_i) = \begin{cases} \infty, & \text{along reflected vector} \\ 0, & \text{otherwise} \end{cases}$$

This is not practical, and therefore we will handle the pure specular case (reflection and refraction) separately.
Previously in Advanced Graphics

Monte Carlo integration:

- Soft shadows: randomly sample the area of a light source;
- Glossy reflections: randomly sample the directions in a cone;
- Depth of field: randomly sample the aperture;
- Motion blur: randomly sample frame time.

In the case of the rendering equation, we are dealing with a **recursive integral**.

**Path tracing**: evaluating this integral using a **random walk**.
Today's Agenda:

- Introduction
- Path Tracing
Path Tracing

Solving the Rendering Equation

\[ L_o(x, \omega_o) = L_E(x, \omega_o) + \int_{\Omega} f_r(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos \theta_i \, d\omega_i \]

Let's start with direct illumination:

For a screen pixel, diffuse surface point \( p \) with normal \( \vec{N} \) is directly visible. What is the radiance travelling via \( p \) towards the eye?

Answer:

\[ L_o(p, \omega_o) = \int_{\Omega} f_r(p, \omega_o, \omega_i) L_d(p, \omega_i) \cos \theta_i \, d\omega_i \]
Path Tracing

Solving the Rendering Equation

\[ L_o(p, \omega_o) = L_E(x, \omega_o) + \int \omega_i \frac{albedo}{\pi} L_d(p, \omega_i) \cos \theta_i \, d\omega_i \]

Let's start with direct illumination:

For a screen pixel, diffuse surface point \( p \) with normal \( N \) is directly visible. What is the radiance travelling via \( p \) towards the eye?

**Answer:**

\[ L_o(p, \omega_o) = \int \omega_i \frac{albedo}{\pi} L_d(p, \omega_i) \cos \theta_i \, d\omega_i \]

In other words: the sum of radiance (scaled by \( \cos \theta_i \) to convert to irradiance) arriving from all directions over the hemisphere, divided by \( \pi \).

**Q:** What about distance attenuation?

**A:** A far-away light is found by fewer directions \( \omega_i \): it's solid angle on the hemisphere is smaller.

**Q:** What happened to \( \omega_o \)?

**A:** The BRDF is independent of \( \omega_o \) (it doesn’t appear in the equation), but as \( \omega_i \) approaches the horizon, \( \cos \theta_i \) approaches zero.
Path Tracing

Direct Illumination

\[
L_0(p, \omega_o) = \int_{\Omega} f_r(p, \omega_o, \omega_i) L_d(p, \omega_i) \cos \theta_i \, d\omega_i
\]

We can solve this integral using Monte-Carlo integration:

- Chose \( N \) random directions over the hemisphere for \( p \)
- Find the first surface in each direction by tracing a ray
- If the surface is light emitting: add it to the sum
- Divide the sum by \( N \) and multiply by \( 2\pi \)

\[
L_0(p, \omega_o) \approx \frac{2\pi}{N} \sum_{i=1}^{N} f_r(p, \omega_o, \omega_i) L_d(p, \omega_i) \cos \theta_i
\]
Direct Illumination

\[ L_0(p, \omega_o) \approx \frac{2\pi}{N} \sum_{i=1}^{N} f_r(p, \omega_o, \omega_i) L_d(p, \omega_i) \cos \theta \]

Questions:
- Why do we multiply by \(2\pi\)?
- What is the radiance \(L_d(p, \omega_i)\) towards \(p\) for e.g. a 100W light?
- What is the irradiance \(E\) at \(p\) from this light?

We integrate over the hemisphere, which has an area of \(2\pi\).

Do not confuse this with the \(1/\pi\) factor in the BRDF, which doesn't compensate for the surface of the hemisphere, but the integral of \(\cos \theta\) over the hemisphere (\(\pi\)).

\(L\) is per steradian; \(L_d(p, \omega_i)\) is proportional to the solid angle of the light as seen from \(p\), so: \(~(A_{disc}/r^2)\).
Direct Illumination

\[ L_o(p, \omega_o) = \int_\Omega f_r(p, \omega_o, \omega_i) L_d(p, \omega_i) \cos \theta_i \, d\omega_i \]

In many directions, we will not find light sources. We can improve our estimate by sampling the lights separately.

\[ L_o(p, \omega_i) = \sum_{j=1}^{lights} \int_\Omega f_r(p, \omega_o, \omega_i) L_d^j(p, \omega_i) \cos \theta_i \, d\omega_i \]

Obviously, sampling the entire hemisphere for each light is not necessary; we can sample the area of the light instead:

\[ L_o(p, \omega_i) = \sum_{j=1}^{lights} \int_A f_r(p, \omega_o, \omega_i) L_d^j(p, \omega_i) \cos \theta_i \, d\omega_i \]
Path Tracing

Direct Illumination

\[ L_o(p, \omega_i) = \sum_{j=1}^{\text{lights}} \int_A f_r(p, \omega_o, \omega_i) L^j_d(p, \omega_i) \cos \theta_i \, d\omega_i \]

Using Monte-Carlo:

\[ L_o(p, \omega_i) \approx \frac{1}{N} \sum_{i=1}^{N} f_r(p, \omega_o, P) L^i_d(p, P) V(p \leftrightarrow P) \]

where

- \( L^j_d(p, P) \) is the direct light towards \( p \) from random point \( P \) on random light \( J \)
- \( V(p \leftrightarrow P) \) is the mutual visibility between \( p \) and \( P \)
- \( A_{L_d} \) is the area of this light source
- \( \frac{(A_{L_d} \cos \theta_i \cos \theta_o)}{\| p - P \|^2} \) is the area of the light source projected on the hemisphere, which approximates the solid angle of the light.

Recall:

\[ V_A = \int_A^B f(x) \, dx \approx \frac{B - A}{N} \sum_{i=1}^{N} f(x_i) \]
Path Tracing

Direct Illumination

We now have two methods to estimate direct illumination using Monte Carlo integration:

1. By random sampling the hemisphere:

   \[ L_o(p, \omega_o) \approx \frac{2\pi}{N} \sum_{i=1}^{N} f_r(p, \omega_o, \Omega_i) L_d(p, \Omega_i) \cos \theta_i \]

2. By sampling the lights directly:

   \[ L_o(p, \omega_i) \approx \frac{\text{lights}}{N} \sum_{i=1}^{N} f_r(p, \omega_o, P) L'_d(p, P) V(p \leftrightarrow P) \frac{A_L L_d \cos \theta_i \cos \theta_o}{\| p - P \|^2} \]

For \( N = \infty \), these yield the same result.
Path Tracing

Direct Illumination

We now have two methods to estimate direct illumination using Monte Carlo integration:

1. By random sampling the hemisphere:

\[
L_o(p, \omega_o) \approx \frac{2\pi}{N} \sum_{i=1}^{N} f_r(p, \omega_o, \Omega_i) \cdot L_d(p, \Omega_i) \cdot \cos \theta_i
\]

2. By sampling the lights directly (three point formulation):

\[
L_o(s \leftarrow p) \approx \frac{\text{lights}}{N} \sum_{i=1}^{N} f_r(s \leftarrow p \leftarrow Q) \cdot L_d^l(p \leftarrow Q) \cdot V(p \leftarrow Q) \cdot \frac{A_{L_d^l} \cdot \cos \theta_i \cdot \cos \theta_o}{\| p - Q \| ^2}
\]

For \( N = \infty \), these yield the same result.
Verification

Method 1 in a small C# ray tracing framework:

In: Ray ray, with members O, D, N, t.
Already calculated: intersection point I = O + t * D.

Vector3 R = RTTools.DiffuseReflection( ray.N);
Ray rayToHemisphere = new Ray( I + R * EPSILON, R, 1e34f );
Scene.Intersect( rayToHemisphere );
if (rayToHemisphere.objIdx == LIGHT)
{
    Vector3 BRDF = material.diffuse * INVPI;
    float cos_i = Vector3.Dot( R, ray.N );
    return 2.0f * PI * BRDF * Scene.lightColor * cos_i;
}
Path Tracing

\[ L_o(p, \omega_i) \approx \text{lights} \ast \frac{1}{N} \sum_{i=1}^{N} f_r(p, \omega_o, P) L_d^i(p, P) V(p \leftrightarrow P) \frac{A_{Ld} \cos \theta_i \cos \theta_o}{\| p - P \|^2} \]

Verification

Method 2 in a small C# ray tracing framework:

```csharp
// construct vector to random point on light
Vector3 L = Scene.RandomPointOnLight() - I;
float dist = L.Length();
L /= dist;
float cos_o = Vector3.Dot(-L, lightNormal);
float cos_i = Vector3.Dot(L, ray.N);
if ((cos_o <= 0) || (cos_i <= 0)) return BLACK;

// light is not behind surface point, trace shadow ray
Ray r = new Ray(I + EPSILON * L, L, dist - 2 * EPSILON);
Scene.Intersect(r);
if (r.objIdx != -1) return Vector3.Zero;

// light is visible (V(p,p')=1); calculate transport
Vector3 BRDF = material.diffuse * INVPI;
float solidAngle = (cos_o * Scene.LIGHTAREA) / (dist * dist);
return BRDF * lightCount * Scene.lightColor * solidAngle * cos_i;
```
Path Tracing

```
0.1s
```
Path Tracing

```
0.5s
```
Path Tracing

```
... (more code here) ...
```
Path Tracing

```c

// Advanced Graphics - Path Tracing

// Scene parameters

// ray tracing

// Intersecting the plane

// Path Tracing

// Diffuse shading

// Sample a new point

// Compute the new point

// End of path tracing
```

30.0s
Rendering using Monte Carlo Integration

In the demonstration, we sampled each light using only 1 sample. The (very noisy) result is directly visualized.

To get a better estimate, we average the result of several frames (and thus: several samples).

Observations:

1. The light sampling estimator is much better than the hemisphere estimator;
2. Relatively few samples are sufficient for a recognizable image;
3. Noise reduces over time, but we quickly get diminishing returns.
Path Tracing

Indirect Light

Returning to the full rendering equation:

\[
L_o(x, \omega_o) = L_F(x, \omega_o) + \int_{\Omega} f_r(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos \theta_i \, d\omega_i
\]

We know how to evaluate direct lighting arriving at \(x\) from all directions \(\omega_i\):

\[
L_o(p, \omega_o) = \int_{\Omega} f_r(p, \omega_o, \omega_i) L_d(p, \omega_i) \cos \theta_i \, d\omega_i
\]

What remains is indirect light.
This is the light that is not emitted by the surface in direction \(\omega_i\), but reflected.
Path Tracing

Indirect Light

\[ L_0(x, \omega_o) = L_E(x, \omega_o) + \int_{\Omega} f_r(x, \omega_o, \omega_i) \, l_i(x, \omega_i) \cos \theta_i \, d\omega_i \]

Let’s expand / reorganize this:

\[ L_0(x, \omega_o^x) = L_E(x, \omega_o^x) \]

\[ L_0(p, \omega_o) \approx \frac{2\pi}{N} \sum_{i=1}^{N} f_r(p, \omega_o, \Omega_i) \, l_d(p, \Omega_i) \cos \theta_i \]

\[ \int_{\Omega} L_E(y, \omega_o^y) \, f_r(x, \omega_o^x, \omega_i^x) \, \cos \theta_i \, d\omega_i^x \]

\[ \int_{\Omega} \int_{\Omega} L_E(z, \omega_o^z) \, f_r(y, \omega_o^y, \omega_i^y) \, \cos \theta_i \, f_r(x, \omega_o^x, \omega_i^x) \, \cos \theta_i \, d\omega_i^x \, d\omega_i^y \]

\[ \int_{\Omega} \int_{\Omega} \int_{\Omega} \cdots \]

\[ \approx \cdots \]
Indirect Light

One particle finding the light via a surface:

\[ I, N = \text{Trace}( \text{ray} ); \]
\[ R = \text{DiffuseReflection}( N ); \]
\[ \text{lightColor} = \text{Trace}( \text{new Ray}( I, R ) ); \]
\[ \text{return } \text{dot}( R, N ) \cdot \frac{\text{albedo}}{\pi} \cdot \text{lightColor} \cdot 2\pi; \]

One particle finding the light via two surfaces:

\[ I_1, N_1 = \text{Trace}( \text{ray} ); \]
\[ R_1 = \text{DiffuseReflection}( N_1 ); \]
\[ I_2, N_2 = \text{Trace}( \text{new Ray}( I_1, R_1 ) ); \]
\[ R_2 = \text{DiffuseReflection}( N_2 ); \]
\[ \text{lightColor} = \text{Trace}( \text{new Ray}( I_2, R_2 ) ); \]
\[ \text{return } \text{dot}( R_1, N_1 ) \cdot \frac{\text{albedo}}{\pi} \cdot \text{lightColor} \cdot 2\pi \cdot \text{dot}( R_2, N_2 ) \cdot \frac{\text{albedo}}{\pi} \cdot 2\pi \cdot \text{lightColor}; \]
Path Tracing

Path Tracing Algorithm

```c
Color Sample( Ray ray ) {
    // trace ray
    I, N, material = Trace( ray );
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight) return material.emittance;
    // continue in random direction
    R = DiffuseReflection( N );
    Ray newRay( I, R );
    // update throughput
    BRDF = material.albedo / PI;
    Ei = Sample( newRay ) * dot( N, R ); // irradiance
    return PI * 2.0f * BRDF * Ei;
}
```

```c
// trace ray
I, N, material = Trace( ray );
```

```c
// terminate if ray left the scene
if (ray.NOHIT) return BLACK;
```

```c
// terminate if we hit a light source
if (material.isLight) return material.emittance;
```

```c
// continue in random direction
R = DiffuseReflection( N );
```

```c
Ray newRay( I, R );
```

```c
// update throughput
BRDF = material.albedo / PI;
```

```c
Ei = Sample( newRay ) * dot( N, R ); // irradiance
```

```c:return PI * 2.0f * BRDF * Ei;
```
Path Tracing

```
# Path Tracing

# Advanced Graphics

```
Path Tracing
Path Tracing
Path Tracing

Particle Transport

The random walk is analogous to particle transport:

- a particle leaves the camera
- at each surface, energy is absorbed proportional to 1-albedo (‘surface color’)
- at each surface, the particle picks a new direction
- at a light, the path transfers energy to the camera.

In the simulation, particles seem to travel backwards. This is valid because of the Helmholtz reciprocity.

Notice that longer paths tend to return less energy.
Particle Transport - Mirrors

Handling a pure specular surface:

A particle that encounters a mirror continues in a deterministic way.

Question:
- What happens at a red mirror?
- What happens if a material is only half reflective?
Path Tracing

Particle Transport - Glass

Handling dielectrics:

Dielectrics reflect and transmit light.
In the ray tracer, we handled this using two rays.

A particle must choose.

The probability of each choice is calculated using the Fresnel equations.

```c
Color Sample( Ray ray )
{
    // trace ray
    I, N, material = Trace( ray );
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight) return emittance;
    // surface interaction
    if (material.isMirror)
    {
        // continue in fixed direction
        Ray r( I, Reflect( N ) );
        return material.albedo * Sample( r );
    }
    // continue in random direction
    R = DiffuseReflection( N );
    BRDF = material.albedo / PI;
    Ray r( I, R );
    // update throughput
    Ei = Sample( r ) * (N∙R);
    return PI * 2.0f * BRDF * Ei;
}
```
Path Tracing

```c
L (depth < MAXDEPTH)

c = Inside ? 1 : -1
R = wV / wC; wD != wV
wDz = 1.0F - wC.y * wC.y
(b, h)

if (wDz < 0.0F)

R = c * R + (1.0F - c) * W
E = diffuse;
true;

if (refr) && (depth < MAXDEPTH)

E = true;

if (brentf - EvaluateDiffuse( L, M ) * Perlin + ats_factor * diffuse =emax;}
int weight = Misc( directr, brentf );
E = (weight * costheta) / directr; /* radiance = SampleLight( Brand, I, RL, RBig, .x + radiance.y + radiance.z ) > 0 ) && (dot( N, R )
E = true;

d = SampleDir( diffuse, N, r1, r2, s1, s2, pdf1 };
E = bvec * ( dot( N, R ) / pdf1 )
```
Path Tracing

```
encL (depth < MAXDEPTH) {
  t = Inside ? t : t + n;
  r = NT / s; b = rT;
  x1 = rB - n * s;
  x2 = rT - n * s;
  x = aR - bT;
  r = x1 + (b - 1.0) * x2;
  c = true;
  refr = r * diffuse;
  if (c && refr) && (depth < MAXDEPTH) {
    encL (depth);
  } else {
    c = true;
    N = refr;
    if (N.N < 0) {
      N *= -1;
    }
    refn = N * diffuse;
    if (refn > 0.0) {
      if (survival < Probability)
      if (Estimation = doing it properly, closely)
      if (true)
      refr = EvaluateDiffuse (L, N) + PDF;
      at factor = diffuse = MAX2;
      at weight = Misc (directly, bended);
      at convolution = dot (N, V);
      E = ((weight + convoluted) / direct(Ref)) * (radial);
      random walk = done properly, closely following the
      inverse)
      at
table brdf = SampleDiffuse (diffuse, N, r1, r2, s, pdf)
      if
      pdf = 1; // E = brdf * (dot (N, R) / pdf)
    }
  }
}
```
Today’s Agenda:

- Introduction
- Path Tracing
INFOMAGR – Advanced Graphics

Jacco Bikker - November 2021 - February 2022

END of “Path Tracing”

next lecture: “GPU Ray Tracing (1)”