\[
I(x, x') = g(x, x') \left[ \epsilon(x, x') + \int_{\mathcal{S}} \rho(x, x', x'') I(x', x'') dx'' \right]
\]
Today's Agenda:

- Introduction
- Stratification
- Next Event Estimation
- Importance Sampling
Introduction

Previously in Advanced Graphics
Introduction
Introduction

Today in Advanced Graphics:

- Stratification
- Next Event Estimation
- Importance Sampling
- Multiple Importance Sampling
- Resampled Importance Sampling*

Aim:

- to get a better image with the same number of samples
- to increase the efficiency of a path tracer
- to reduce variance in the estimate

Requirement:

- produce the correct image

*: If time permits
Today's Agenda:

- Introduction
- Stratification
- Next Event Estimation
- Importance Sampling
Stratification

Uniform Random Sampling

To sample a light source, we draw two random values in the range 0..1.

The resulting 2D positions are not uniformly distributed over the area.

We can improve uniformity using *stratification*: one sample is placed in each stratum.
Stratification

Uniform Random Sampling

To sample a light source, we draw two random values in the range 0..1.

The resulting 2D positions are not uniformly distributed over the area.

We can improve uniformity using stratification: one sample is placed in each stratum.

For 4x4 strata:

\[
\begin{align*}
\text{stratum}_x &= (\text{idx} \% 4) \times 0.25 \quad / \quad \text{idx} = 0..15 \\
\text{stratum}_y &= (\text{idx} / 4) \times 0.25 \\
r0 &= \text{Rand()} \times 0.25 \\
r1 &= \text{Rand()} \times 0.25 \\
P &= \text{vec2}(\text{stratum}_x + r0, \text{stratum}_y + r1)
\end{align*}
\]
Stratification
Use Cases

Stratification can be applied to any Monte Carlo process:

- Anti-aliasing (sampling the pixel)
- Depth of field (sampling the lens)
- Motion blur (sampling time)
- Soft shadows (sampling area lights)
- Diffuse reflections (sampling the hemisphere)

However, there are problems:

- We need to take one sample per stratum
- Stratum count: higher is better, but with diminishing returns
- Combining stratification for e.g. depth of field and soft shadows leads to correlation of the samples, unless we stratify the 4D space - which leads to a very large number of strata: the curse of dimensionality.
Stratification

Troubleshooting Path Tracing Experiments

When experimenting with stratification and other variance reduction methods you will frequently produce incorrect images.

Tip:

Keep a simple reference path tracer without any tricks. Compare your output to this reference solution frequently.
Today's Agenda:

- Introduction
- Stratification
- Next Event Estimation
- Importance Sampling
Next Event Estimation

Recall the rendering equation:

\[ \mathbf{L}_o(x, \omega_o) = \mathbf{L}_E(x, \omega_o) + \int_{\Omega} \mathbf{f}_r(x, \omega_o, \omega_i) \mathbf{L}_i(x, \omega_i) \cos \theta_i \ d\omega_i \]

Also recall that we had two ways to sample direct illumination:

- Integrating over the hemisphere
- Integrating over the lights

Can we apply this to the full rendering equation, instead of just direct illumination?
Incoming direct light

\[ = \int_{\Omega} L_d(x, \omega_i) \cos \theta_i \, d\omega_i \]

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

\( -\frac{1}{2}\pi \)

\( +\frac{1}{2}\pi \)
Incoming direct light

\[
\frac{-\pi}{2} \leq \theta_i \leq \frac{\pi}{2}
\]

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

\[
= \int_{\Omega} L_d(x, \omega_i) \cos \theta_i \, d\omega_i
\]

\[
= \int_{A..B} L_d(x, \omega_i) \cos \theta_i \, d\omega_i + \int_{C..D} L_d(x, \omega_i) \cos \theta_i \, d\omega_i
\]
Incoming **direct + indirect** light
Incoming \textbf{direct + indirect} light
Next Event Estimation

Observation: light travelling via any vertex on the path consists of indirect light and direct light \textit{for that vertex}.

Next Event Estimation: sampling direct and indirect separately.
Next Event Estimation

Per surface interaction, we trace *two* random rays.

- Ray A returns (via point $x$) the energy reflected by $y$ (estimates indirect light for $x$).
- Ray B returns the direct illumination on point $x$ (estimates direct light on $x$).
- Ray C returns the direct illumination on point $y$, which will reach the sensor via ray A.
- Ray D leaves the scene.
Next Event Estimation

When a ray for indirect illumination stumbles upon a light, the path is terminated and no energy is transported via ray D:

This way, we prevent accounting for direct illumination on point $y$ twice.
Next Event Estimation

We thus split the hemisphere into two distinct areas:

1. The area that has the projection of the light source on it;
2. The area that is not covered by this projection.

We can now safely send a ray to each of these areas and sum whatever we find there.

(or: we integrate over these non-overlapping areas and sum the energy we receive via both to determine the energy we receive over the entire hemisphere)

Area 1:
Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

Area 2:
Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.
Area 1:
Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

Area 2:
Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.
Area 1:
Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

Area 2:
Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.
Next Event Estimation

```c
Color Sample( Ray ray )
{
    // trace ray
    I, N, material = Trace( ray );
    BRDF = material.albedo / PI;
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight) return BLACK;
    // sample a random light source
    L, Nl, dist, A = RandomPointOnLight();
    Ray lr( I, L, dist );
    if (N.L > 0 && Nl.--L > 0) if (!Trace( lr ))
    {
        solidAngle = ((Nl.--L) * A) / dist^2;
        Ld = lightColor * solidAngle * BRDF * N.L;
    }
    // continue random walk
    R = DiffuseReflection( N );
    Ray r( I, R );
    Ei = Sample( r ) * (N.R);
    return PI * 2.0f * BRDF * Ei + Ld;
}
```
Advanced Graphics – Variance Reduction

```c
C
L (depth < FPDFSTEP)

r = Inside ? T : L;
R = N x / Nt, S = N / Nt;

+ S x = N x - R x, B = R x,
+ T x = S x + (R y + S y),
+ F x = (D x - R x) - (S x - N x),
+ R = (D x - R x) - (S x - N x),
+ E = diffuse;
+ true;
+ refl = refl + refr) && (depth < FPDFSTEP)
+ R, B};
+ refl = " diffuse;
+ true;
+ WADEPPTH)
+ survival = SurviveProbability_diffuse;
+ estimation = doing it properly, (India)
+
+ Radiance = SampleLight; Brand, I, N, A, Blight;
+ x + radiance.y = radiance.z) > 0) && (color * v = true;
+ int brdf = EvaluateDiffuse( L, N ) * "survival";
+ at factor = diffuse * INVERSE;
+ at weight = M2( directI, brnder );
+ at confusion = dot(t, b); ( E = ((weight + cosTheta2) / directPdf) * (radius - 1))
+ window walk = done properly, closely following beautiful
+ (diffuse)
+
+ int brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &S);
+ pdf1
+ E = "brdf" * (dot(N, R) / pdf1);
```
Next Event Estimation

Some vertices require special attention:

- If the first vertex after the camera is emissive, its energy can’t be reflected to the camera.
- For specular surfaces, the BRDF to a light is always 0.

Since a light ray doesn’t make sense for specular vertices, we will include emission from a vertex directly following a specular vertex.

The same goes for the first vertex after the camera: if this is emissive, we will also include this.

*This means we need to keep track of the type of the previous vertex during the random walk.*
Color Sample( Ray ray, bool lastSpecular )
{
    // trace ray
    I, N, material = Trace( ray );
    BRDF = material.albedo / PI;
    // terminate if ray left the scene
    if (ray.NOHIT) return BLACK;
    // terminate if we hit a light source
    if (material.isLight)
        if (lastSpecular) return material.emissive;
        else return BLACK;
    // sample a random light source
    L, NL, dist, A = RandomPointOnLight();
    Ray lr( I, L, dist );
    if (N·L > 0 && NL·-L > 0) if (!Trace( lr ))
    {
        solidAngle = ((NL·-L) * A) / dist^2;
        Ld = lightColor * solidAngle * BRDF * N·L;
    }
    // continue random walk
    R = DiffuseReflection( N );
    Ray r( I, R );
    Ei = Sample( r, false ) * (N·R);
    return PI * 2.0f * BRDF * Ei + Ld;
Today's Agenda:

- Introduction
- Stratification
- Next Event Estimation
- Importance Sampling
Importance Sampling

Important Sampling for Monte Carlo

Monte Carlo integration:

\[ V_A = \int_A^B f(x) \, dx = E(f(X)) \approx \frac{1}{N} \sum_{i=1}^{N} f(X_i) \]

Example 1: rolling two dice, \( D_1 \) and \( D_2 \), the outcome is \( 6D_1 + D_2 \). What is the expected value? (average die value is 3.5, so the answer is of course 3.5 * 6 + 3.5 = 24.5)

\[ V = \frac{1}{N} \sum_{i=1}^{N} f(D_1) + g(D_2) \text{ where: } D_1, D_2 \in \{1,2,3,4,5,6\}, f(x) = 6x, g(x) = x \]
Importance Sampling

Importance Sampling for Monte Carlo

Changing the experiment slightly: each sample is one roll of one die.

Using Monte Carlo:

$$ V = \frac{1}{N} \sum_{i=1}^{N} \frac{f(T, D)}{0.5} \text{ where: } D \in \{1, 2, 3, 4, 5, 6\}, \quad T \in \{0, 1\} $$

for (int i = 0; i < 1000; i++)
{
    int D1 = IRand(6) + 1;
    int D2 = IRand(6) + 1;
    float f = (float)(6 * D1 + D2);
    total += f;
    rolls++;
}

for (int i = 0; i < 2000; i++)
{
    int D = IRand(6) + 1;
    int T = IRand(2);
    float f = (float)((5 * T + 1) * D) / 0.5f;
    total += f;
    rolls++;
}

0.5: Probability of using die $T$. 

importance = SampleLight(brand, 1, &Al, &B); 
int weight = M(1, direct, &B, &r); 
for (int i = 0; i < 2000; i++)
{
    int D = IRand(6) + 1;
    int T = IRand(2);
    float f = (float)((5 * T + 1) * D) / 0.5f;
    total += f;
    rolls++;
}

for( int i = 0; i < 2000; i++ )
{
    int D = IRand( 6 ) + 1;
    int T = IRand( 2 );
    float f = (float)((5 * T + 1) * D) / 0.5f;
    total += f;
    rolls++;
}
Importance Sampling

Importance Sampling for Monte Carlo

What happens when we don't pick each die with the same probability?

- we get the correct answer;
- we get lower variance.

```c
for( int i = 0; i < 1000; i++ )
{
    int D = IRand( 6 ) + 1;
    float r = Rand();
    int T = (r < 0.8f) ? 0 : 1;
    float p = (T == 0) ? 0.8f : 0.2f;
    float f = (float)((5 * T + 1) * D) / p;
    total += f;
    rolls++;
}
```
INFOMAGR – Advanced Graphics

Jacco Bikker - November 2018 - February 2019

END of “Variance Reduction”

next lecture: “GPU Ray Tracing (2)”