ics ≹(depth < NACD

: = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt ° n D, N); ≫)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (dd)

= * diffuse = true;

. :fl + refr)) && (de

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, # // _]]

e.x + radiance.y + radiance.z) $\rightarrow I(\chi)$

v = true;

at brdfPdf = EvaluateDiffuse(L, N) * Psu at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf) = (rac

andom walk - done properly, closely following See /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true: $\epsilon(x,x')$

INFOMAGR – Advanced Graphics

Jacco Bikker - November 2021 - February 2022

Lecture 8,9- "Variance Reduction"

Welcome!



ics & (depth < NoCOS

z = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt o D, N); 0)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Tr) R = (D * nnt - N * (dd)

= * diffuse = true;

. :fl + refr)) && (depth < MODEL

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed Hf; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && closed

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Random Samples
- Next Event Estimation
- Importance Sampling
- Russian Roulette



Introduction

Previously in Advanced Graphics



v = true; at brdfPdf = EvaluateDiffuse(L, N) * Ps at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

ndom walk - done properly, closely following SAL ive)

, t33 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; .pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



Introduction

hics & (depth < MOCO:

c = inside / 1 / / ht = nt / nc, ddn bs2t = 1.0f - nnt / r D, N); D)

at a = nt - nc, b = nt = r at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (000

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, closed
if;
radiance = SampleLight(&rand, I, &L, &ll)
e.x + radiance.y + radiance.z) > 0) && ()

v = true;

at brdfPdf = EvaluateDiffuse(L, N) * Pau at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:





Introduction

at a = ni

), N);

AXDEPTH)

v = true;

/ive)

lf;

survive = SurvivalProbability(dif

radiance = SampleLight(&rand, I e.x + radiance.y + radiance.z) > 0

at brdfPdf = EvaluateDiffuse(L, |

E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely fo

at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

Today in Advanced Graphics:

- Blue Noise
- **Next Event Estimation**
- Importance Sampling
- Multiple Importance Sampling
- **Resampled Importance Sampling**

- 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

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 1 urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true:

Stratification

Aim:







ics & (depth < NoCOS

z = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt o D, N); 0)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Tr) R = (D * nnt - N * (dd)

= * diffuse = true;

. :fl + refr)) && (depth < MODEL

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed Hf; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && closed

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Random Samples
- Next Event Estimation
- Importance Sampling
- Russian Roulette



sics & (depth < MacCor

: = inside ? 1 ()) ht = nt / nc, ddn ss2t = 1.0f - nnt ° (), N); ∂)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (dd)

= * diffuse; = true;

• efl + refr)) && (depth < MOXDEPII

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvise at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (cost)

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

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.





hics & (depth < ⊅0000

: = inside ? 1 1 1 1 ht = nt / nc, ddm 1 bs2t = 1.0f - nmt 1 nm 2, N); 3)

at a = nt - nc, b = nt + n at Tr = 1 - (R0 + (1 - R0) Fr) R = (D = nnt - N = (ddn

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTI

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, &t, e.x + radiance.y + radiance.z) > 0) &&

v = true;

sion = true

at brdfPdf = EvaluateDiffuse(L, N) Pa at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely fol /ive)

; at3 brdf = SampleDiffuse(diffuse, N, rl urvive; pdf; n = E * brdf * (dot(N, R) / pdf);

Randomness

By the way...

What is a good random number?

rand

int rand (void); Generate random number

Returns a pseudo-random integral number in the range between 0 and RAND MAX.

This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called. This algorithm uses a seed to generate the series, which should be initialized to some distinctive value using function srand.

RAND_MAX is a constant defined in <cstdlib>.

A typical way to generate trivial pseudo-random numbers in a determined range using rand is to use the modulo of the returned value by the range span and add the initial value of the range:

1 v1 = rand() % 100;	// v1 in the range 0 to 99
2 v2 = rand() % 100 + 1;	// v2 in the range 1 to 100
³ v3 = rand() % 30 + 1985;	// v3 in the range 1985-2014

BAD. What if RAND_MAX is 65535, and we want a number in the range 0..50000? The range 50000..65535 will overlap 0..15535...

<cstdlib>

Notice though that this modulo operation does not generate uniformly distributed random numbers in the span (since in most cases this operation makes lower numbers slightly more likely).

C++ supports a wide range of powerful tools to generate random and pseudo-random numbers (see <random> for more info).





hics & (depth < NACCC

: = inside ? 1 | 1 | 1 ht = nt / nc, ddn = 1 os2t = 1.0f - nnt = nn D, N); B)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N * (ddn

= * diffuse; = true;

efl + refr)) && (depth < NOCCO

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffd estimation - doing it properly, it if; radiance = SampleLight(&rand, I, &L 2.x + radiance.y + radiance.z) > 0)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Pa at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely foll /ive)

. t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Randomness

By the way...

What is a good random number?

Generators

Pseudo-random number engines (templates) Generators that use an algorithm to generate pseudo-random numbers based on an initial seed: Inear_congruential_engine Linear congruential random number engine (class template) mersenne_twister_engine Mersenne twister random number engine (class template) subtract_with_carry_engine Subtract-with-carry random number engine (class template)

Engine adaptors

 They adapt an engine, modifying the way numbers are generated with it:

 discard_block_engine
 Discard-block random number engine adaptor (class template)

 independent_bits_engine
 Independent-bits random number engine adaptor (class template)

 shuffle_order_engine
 Shuffle-order random number engine adaptor (class template)

Pseudo-random number engines (instantiations)

Particular instantiations of generator engines and adaptors:		
default_random_engine Default random engine (class)		
minstd_rand	Minimal Standard minstd_rand generator (class)	
minstd_rand0	Minimal Standard minstd_rand0 generator (class)	
mt19937	Mersenne Twister 19937 generator (class)	





sics & (depth < Motos)

t = inside 7 1 1 1 0 nt = nt / nc, ddn 1 1 ps2t = 1.0f - n⊓t 1 0 D, N); ≫)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0) Tr) R = (D ⁼ nnt - N

= * diffuse; = true;

-:fl + refr)) && (depth < MODEPT)

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed Hf; radiance = SampleLight(&rand, I, &L, &Lis e.x + radiance.y + radiance.z) > 0) & closed e.x + radiance.z) + radiance.z

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (PSUR);

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Randomness

By the way...

What is a good random number?

Consider Marsaglia's xor32*:

uint xorshift32(uint& state)

state ^= state << 13; state ^= state >> 17; state ^= state << 5; return state;

Xor32, plus: float RandomFloat(uint& s) { return xorshift32(s) * 2.3283064365387e-10f; } Seeding: Try the 'WangHash'.

*: Marsaglia, 2003. "Xorshift RNGs". Journal of Statistical Software.





sics & (depth < Motoon

: = inside ? 1 1 1 2 ht = nt / nc, ddn 552t = 1.0f - nnt " r 2, N); 3)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N = (dd)

= * diffuse; = true;

• efl + refr)) && (depth < MANDEPT

D, N); refl * E * diffu = true;

AXDEPTH)

sion = true:

```
survive = SurvivalProbability( diffuse
estimation - doing it properly
if;
radiance = SampleLight( &rand, I, &L, &lie
e.x + radiance.y + radiance.z) > 0) && (doing to the set of the s
```

w = true; at brdfPdf = EvaluateDiffuse(L, N) * Psu at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following /ive)

```
,
t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, dpdf )
rrvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
```

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:

stratum_x = (idx % 4) * 0.25 // idx = 0..15
stratum_y = (idx / 4) * 0.25
r0 = Rand() * 0.25
r1 = Rand() * 0.25
P = vec2(stratum_x + r0, stratum_y + r1)





sics & (depth < Moos

: = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - n⊓t 2, N); 3)

at a = nt - nc, b = nt = nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (ddn

* diffuse; = true;

. efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &II e.x + radiance.y + radiance.z) > 0) && (d)

w = true; at brdfPdf = EvaluateDiffuse(L, N) =

at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following S /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, addi urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



tics & (depth < Notici⊓

nt a = nt - nc, b = nt + n nt Tr = 1 - (R0 + (1 - R0 Tr) R = (D ⁼ nnt - N ⁼ (ddn

= * diffuse; = true;

-:fl + refr)) && (depth < MA)

D, N); refl * E * diffuse = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, close if; radiance = SampleLight(&rand, I, &L, e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, N) * F at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely followin /ive)

. t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

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*.





Blue Noise

tics & (depth < Mox000

t = inside 7 1 1 1 0 ht = nt / nc, ddn 0 0 bs2t = 1.0f - nnt 0 0, N); ∂)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0) Fr) R = (D ⁺ nnt - N - (dd)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed Hf; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && closed

v = true; at brdfPdf = EvaluateDiffuse(L, N) Pourvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (rad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dpdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Uniform Random Numbers

Stratification helps, because it improves the *uniformity* of random numbers.

Other approaches to achieve this:

Poisson-disc distributions

Also known as: blue noise.









https://www.arnoldrenderer.com/research/dither_abstract.pdf

nics & (depth < Notoon

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (dd)

= * diffuse; = true;

efl + refr)) && (depth < MAXDEPT

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) 2.x + radiance.y + radiance.z) > 0) && (doing)

v = true;

at brdfPdf = EvaluateDiffuse(L, N) * Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (psi

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

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.



ics & (depth < NoCOS

z = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt o D, N); 0)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Tr) R = (D * nnt - N * (dd)

= * diffuse = true;

. :fl + refr)) && (depth < MODEL

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed Hf; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && closed

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Random Samples
- Next Event Estimation
- Importance Sampling
- Russian Roulette



NEE

Next Event Estimation

Recall the rendering equation:

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

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

 J_{Ω}

lights

...and the way we sampled it using Monte Carlo:

lωi

 $L_o(x)$

$$L_o(p,\omega_o) = \int f_r(p,\omega_o,\omega_i) L_d(p,\omega_i) \cos\theta_i dx$$

refl * E

= true:

survive = SurvivalProbability(diff adiance = SampleLight $L(s \leftarrow x) = \sum_{i=1}^{5} \int_{A_i} f_r(s \leftarrow x \leftarrow x'_j) L(x \leftarrow x'_j) G(x \leftrightarrow x'_j) dA(x')$ integrating at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L)

integrating over

the hemisphere

E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely foll /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1 pdf; 1 = E * brdf * (dot(N, R) / pdf);

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

Vector3 L = RandomPointOnLight() - I; float dist = L.Length(); L /= dist; float cos o = Dot(-L, lightNormal); float cos i = Dot(L, ray.N); if (cos o <= 0 || cos i <= 0) return BLACK;</pre> // trace shadow ray Ray r = new Ray(...); Scene.Intersect(r); if (r.objIdx != -1) return BLACK; // V(p,p')=1; calculate transport Vector3 BRDF = material.diffuse * INVPI; float solidAngle = ...; return solidAngle * BRDF * lightColor * cos i;

$$\int_{A}^{B} f(x) dx \approx \frac{B-A}{N} \sum_{i=1}^{N} f(X)$$





 $-\frac{1}{2}\pi$

Incoming <u>direct light</u>

NEE

sics & (depth < Motocraa

c = inside / 1 ht = nt / nc, ddn bs2t = 1.0f - nnt 2, N); 3)

at a = nt - nc, b = nt = nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N ⁼ (ddn

= * diffuse; = true;

efl + refr)) && (depth < MAXDEE

D, N); refl * E * diffuse = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, If; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && doctory

v = true; at brdfPdf =

at brdfPdf = EvaluateDiffuse(L, N) * Psurvio at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (PS

andom walk - done properly, closely following SOO /ive)

, t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf) prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



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



 $\boldsymbol{\chi}$

 $+\frac{1}{2}\pi$

NEE



at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, & urvive; pdf; i = E * brdf * (dot(N, R) / pdf); sion = true:



22

NEE

AXDEPTH)



v = true; at brdfPdf = EvaluateDiffuse(L, N) 🤅 P at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

/ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, & urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:





 $-\frac{1}{2}\pi$

NEE

Incoming <u>direct + indirect</u> light

tics & (depth < Modern

t = inside / 1 ht = nt / nc, ddn bs2t = 1.0f - nnt), N); »)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N * (ddn

= * diffuse; = true;

-:**fl + refr)) && (depth < MAXDE**

D, N); refl * E * diffuse = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &L) 2.x + radiance.y + radiance.z) > 0 && ()

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Pount at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely $\overline{1/2}\pi$

, H33 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apd urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



 $+\frac{1}{2}\pi$



nics & (depth < PACES

: = inside ? 1 : 1.0 ht = nt / nc, ddn ss2t = 1.0f - n∩t * 2, N); ≫)

at a = nt - nc, b = nt + nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N - (dd)

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed ff; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) &

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sovi /ive)

, H3 Brdf = SampleDiffuse(diffuse, N, r1, r2, &R, Spdf) prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Next Event Estimation

Observation: light travelling via any vertex on the path consists of indirect light and direct light *for that vertex.*

Next Event Estimation: sampling direct and indirect separately.





at a = n

), N);

AXDEPTH)

v = true;

/ive)

lf;

refl * E * diffuse;

survive = SurvivalProbability(di

radiance = SampleLight(&rand, I

e.x + radiance.y + radiance.z)

at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf

at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf) andom walk - done properly, closely follo

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.





t3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apd urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:



ics & (depth < NACCO

c = inside ? 1 ()) ht = nt / nc, ddn (os2t = 1.0f - nnt " r 2, N); ≫)

at a = nt - nc, b = nt - n at Tr = 1 - (R0 + (1 - R0 Ir) R = (D = nnt - N = (ddn)

= * diffuse; = true;

efl + refr)) && (depth < MONDEPTH

D, N); refl * E * diffus = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly f; radiance = SampleLight(&rand, I, &L, &L) 2.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurvit at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * ();

andom walk - done properly, closely following 300. /ive)

st3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, Brdf)
urvive;
pdf;
n = E * brdf * (dot(N, R) / pdf);
sion = true:

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.





27

ics & (depth < Modern

: = inside ? 1 1 1 1 ht = nt / nc, ddn ps2t = 1.0f - nnt ? r 2, N); 3)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N * (dom

= * diffuse; = true;

• efl + refr)) && (depth < MADDEP

), N); refl * E * diffus = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closi if; radiance = SampleLight(&rand, I, &L, 2.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psum at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following 3000 /ive)

, t3 Brdf = SampleDiffuse(diffuse, N, r1, r2, &R, apdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

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.



 $\boldsymbol{\chi}$



sion = true:

NEE	Area 1:
	Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.
rics A (depth & Moderna)	
<pre>= inside = 1 = 1 = 1 = 1 = 1 = 1 = 1 = 1 = 1 =</pre>	Area 2:
$\begin{array}{l} \mathbf{t} = \mathbf{n} \mathbf{t} - \mathbf{n} \mathbf{c} \cdot \mathbf{b} + \mathbf{n} \mathbf{t} \\ \mathbf{t} \mathbf{T} = 1 - 1 \mathbf{f} 2 \mathbf{\pi} \\ \mathbf{t} \mathbf{r} \\ \mathbf{r} \end{pmatrix}_{R} = (\mathbf{D} = \mathbf{n} 1 2 \mathbf{\pi} \mathbf{n} + \mathbf{n} \mathbf{n} \mathbf{n} \mathbf{n} \mathbf{n} \mathbf{n} \mathbf{n} \mathbf{n}$	$+\frac{1}{2}\pi$ Send a ray in a random direction on the
: * diffuse; = true;	source.
<pre>f1 + refr)) && (depth < MAXDEPRIND), N); ref1 * E * diffuse; = true; WXDEPTH) survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &lightson e.x + radiance.y + radiance.z) > 0) && dot</pre>	
v = true; et brdfPdf = āvgluateDiffuse(L, N) Psurvive et3 factor = ā/2ult * InvPi; t weight = Mic2(digertPdf, bodfDdf))	$+\frac{1}{2}\pi$
E * ((weight * cosThetaOut) / directPdf) * (radiance	$+\frac{1}{2}\pi$
andom walk - done properly, closely following Social vive)	
at3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 8pdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf);	X

VIS1

Next Event Estimation

tics & (depth < ™ocosin

c = inside ? 1 . . . ht = nt / nc, ddn bs2t = 1.0f - nnt 2, N); 2)

at a = nt - nc, b = nt + r at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (000

= * diffuse; = true;

-:fl + refr)) && (depth < MAXDEPT

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &llow e.x + radiance.y + radiance.z) > 0) && _____

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (read);

andom walk - done properly, closely following Same /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 8pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Color Sample(Ray ray) // trace ray I, N, material = FindNearest(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 $(N \cdot L > 0 \& N \cdot L > 0)$ if $(! \circ C \cdot L = 0)$ if $(! \circ C \cdot L = 0)$ solidAngle = $((Nl \cdot -L) * A) / dist^2;$ Ld = lightColor * solidAngle * BRDF * $N \cdot L * lightCount;$

// continue random walk R = DiffuseReflection(N); Ray r(I, R); Ei = Sample(r) * (N·R);

}

return PI * 2.0f * BRDF * Ei + Ld;



hics & (depth < ⊅0000

: = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt 2, N); 3)

at a = nt - nc, b = nt + r at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N = (dd)

* diffuse; = true;

. efl + refr)) && (depth < MODECCI

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed ff; radiance = SampleLight(&rand, I, &L, &L) e.x + radiance.y + radiance.z) > 0) && ()

v = true;

at brdfPdf = EvaluateDiffuse(L, N) Psur at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following S /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dodf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:





hics & (depth < ≯VXC)

: = inside / l ht = nt / nc, ddn bs2t = 1.0f - nmt / D, N); B)

at a = nt - nc, b = nt = n at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N - (ddn

= * diffuse; = true:

. efl + refr)) && (depth < NOXDEPTIO

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse)
estimation - doing it properly, classed
f;
radiance = SampleLight(&rand, I, &L, &ll
e.x + radiance.y + radiance.z) > 0) && ()

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pau at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following S. /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, SR, dpdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); Sion = true:





hics & (depth < NOCC

c = inside ? 1 : . . ht = nt / nc, ddn bs2t = 1.0f - nnt D, N); B)

at a = nt - nc, b = nt + r at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N ⁻ (dd)

= * diffuse = true:

efl + refr)) && (depth < MONDEPTH)

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &light e.x + radiance.y + radiance.z) > 0) && (doing)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following S. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, addf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:





sics & (depth < Modern

= inside ? 1 1 1 0 ht = nt / nc, ddn bs2t = 1.0f - nnt = n D, N); B)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N = (ddn

= * diffuse; = true;

. efl + refr)) && (depth < MADDE

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffs: estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L e.x + radiance.y + radiance.z) > 0) 8

w = true; at brdfPdf = EvaluateDiffuse(L, N) P at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely following Sa /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, Updf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

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.



ics & (depth < MAXDON

: = inside ? 1 1 1 1 ht = nt / nc, ddn 1 1 s2t = 1.0f - nnt " nn 2, N); 8)

NEE

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Ir) R = (D ⁺ nnt - N - (dd)

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly,
doing it properly,
doing it properly,
f;
adiance = SampleLight(&rand, I, &L, &light)
e.x + radiance.y + radiance.z) > 0) && (doing it)

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (1800)

andom walk - done properly, closely following SAC. /ive)

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 8pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

```
// 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 \cdot L > 0 \& N \cdot L > 0) if (!Trace( lr ))
   solidAngle = ((Nl \cdot -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;
```

Color Sample(Ray ray, bool lastSpecular)



Further Reading

HOME PROJECTS PUBLICATIONS PERSONAL

hics & (depth < ≫ooss

c = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt 2, N); 2)

at a = nt - nc, b = nt + r at Tr = 1 - (R0 + (1 - R0 Tr) R = (D = nnt - N = (000

= * diffuse; = true;

efl + refr)) && (depth < MOXDEP

D, N); refl * E * diffuse; = true;

AXDEPTH)

v = true;

survive = SurvivalProbability(diffuse estimation - doing it properly, closed H; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0 & doctory

Part 1:

at brdfPdf = EvaluateDiffuse(L, N) Part ht3 fattors:/diffuce.ompf2.com/2019/12/11/probability-theory-for-physically-based-rendering ht weight = Mis2(directPdf, brdfPdf) ht cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

ndom walk - done prop**Part2:**

bhttps://paccro.pupf2rcom/2019/12/13/probability-theory-for-physically-based-rendering-part-2/
urvive;
pdf;
n = E * brdf * (dot(N, R) / pdf);
sion = true:



Posted in <u>Article</u>, <u>Tutorial</u>
<u>18 Comments</u>

Probability Theory for Physically Based Rendering

by jbikker on December 11, 2019

Introduction

Rendering frequently involves the evaluation of multidimensional definite integrals: e.g., the visibility of an area light, radiance arriving over the area of a pixel, radiance arriving over a period of time, and the irradiance arriving over the hemisphere of a surface point. Evaluation of these integrals is typically done using Monte-Carlo integration, where the integral is replaced by the expected value of a stochastic



ics & (depth < NoCOS

z = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt - n O, N); 0)

at a = nt - nc, b = nt - n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N * (dd)

= * diffuse; = true;

efl + refr)) && (depth < 100

D, N); •eti * E - diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse .estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &I) .x + radiance.y + radiance.z) > 0) &&

v = true;

at brdfPdf = EvaluateDiffuse(L, N) * Psurv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf) * (ra

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Random Samples
- Next Event Estimation
- Importance Sampling
- Russian Roulette



Importance Sampling for Monte Carlo

Monte Carlo integration:

$$V_A = \int_A^B f(x) \, dx = (B-A) \, E(f(X)) \approx \frac{B-A}{N} \sum_{i=1}^N f(X)$$

Example 1: rolling two dice D_1 and D_2 , the outcome is $6D_1 + D_2$. What is the expected value of this experiment?

(Answer: average die value is 3.5, so the answer is 3.5 * 6 + 3.5 = 24.5)

Using Monte Carlo:

w = true; at brdfPdf = EvaluateDiffuse(L, N) * P at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

survive = SurvivalProbability(diff

radiance = SampleLight(&rand, I, & e.x + radiance.y + radiance.z) > 0)

at a = nt

), N);

AXDEPTH)

lf;

efl + refr)) && (depth

refl * E * diffuse;

andom walk - done properly, closely following S /ive)

, t33 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; .pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

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



Importance Sampling for Monte Carlo

at a = nt

efl + refr)) && (depth <

), N); refl * E * diffuse;

AXDEPTH)

survive = SurvivalProbability(di lf: radiance = SampleLight(&rand, I, &L e.x + radiance.y + radiance.z) >

v = true; at brdfPdf = EvaluateDiffuse(L, at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf

andom walk - done properly, closely foll /ive)

```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, 8
urvive;
pdf;
1 = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

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

 $V = \frac{1}{N} \sum_{n=1}^{N} \frac{f(T, D)}{C}$ where: $D \in \{1,2,3,4,5,6\}, T \in \{0,1\}, f(t,d) = (5t+1) d$

0.5: Probability of using die T.

```
for( int i = 0; i < 1000; i++ )</pre>
                                        for( int i = 0; i < 2000; i++ )</pre>
  int D1 = IRand(6) + 1;
                                           int D = IRand(6) + 1;
   int D2 = IRand(6) + 1;
                                           int T = IRand( 2 );
   float f = (float)(6 * D1 + D2);
                                           float f = (float)((5 * T + 1) * D) / 0.5f;
   total += f;
                                           total += f;
                                           rolls++;
   rolls++;
```



Importance Sampling for Monte Carlo

for(int i = 0; i < 1000; i++)</pre>

float D1 prob = 0.8f;

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

inside t = nt / nc, ddm s2t = 1.0f - nnt (, N);)

at a = nt - nc, b = nt - rc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N - (00)

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPTOD

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(%rand, I, %L, % 2.x + radiance.y + radiance.z) > 0) %%

v = true;

at brdfPdf = EvaluateDiffuse(L, N) = = at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf);

at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely follo /ive)

```
,
H33 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, apdf
urvive;
.pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

int D = IRand(6) + 1; float r = Rand(); // uniform 0..1 int T = (r < D1_prob) ? 0 : 1; float p = (T == 0) ? D1_prob : (1 - D1_prob); float f = (float)((5 * T + 1) * D) / p; total += f; rolls++;

we get the correct answer;we get lower variance.



ics & (depth < Maxourn

t = inside ? 1 nt = nt / nc, ddn os2t = 1.0f - n⊓t 2, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 Tr) R = (D = nnt - N

= * diffuse = true;

efl + refr)) && (depth < MANDEPTH

D, N); refl * E * diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (();

andom walk - done properly, closely following 340 /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf) prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Importance Sampling for Monte Carlo

Example 2: sampling two area lights.

Sampling the large light with a greater probability yields a better estimate.



sics & (depth < MARCET

: = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt ~ 1 0, N); 3)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0) Fr) R = (D ⁼ nnt - N - (dd)

= * diffuse = true;

. :fl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, doing if; radiance = SampleLight(&rand, I, &L, &ii) e.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (();

andom walk - done properly, closely following S. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dpdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Importance Sampling for Monte Carlo

Example 3: sampling an integral.

Considering the previous experiments, which stratum should be sample more often?





Importance Sampling for Monte Carlo

Example 3: sampling an integral.

= inside | | ht = nt / nc, ddn ps2t = 1.0f - nnt D, N); D)

at a = nt - nc, b = nt - r at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (dd)

= * diffuse = true;

. :fl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L, &Light) e.x + radiance.y + radiance.z) > 0) && (doing)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (();

andom walk - done properly, closely following S. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dpdf | urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Considering the previous experiments, which stratum should be sample more often?





at a = ni at Tr = 1 - (R0

efl + refr)) && (dep

), N); refl * E * diffuse;

AXDEPTH)

survive = SurvivalProbability(dit lf: radiance = SampleLight(&rand, I .x + radiance.y + radiance.z)

v = true: at brdfPdf = EvaluateDiffuse(at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely foll /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2) urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true:

Importance Sampling for Monte Carlo

Example 3: sampling an integral.

2

1

3

Considering the previous experiments, which stratum should be sample more often?

When using 8 strata and a uniform random distribution, each stratum will be sampled with a 0.125 probability. When using 8 strata and a non-uniform sampling scheme, the sum of the sampling probabilities must be 1. Good sampling probabilities are obtained by simply following the function we're sampling. Note: we must normalize.

8

7

We don't have to use these probabilities; any set of non-zero probabilities will work, but with greater variance. This includes any approximation of the function we're sampling, whether this approximation is good or not.





6

5

Importance Sampling for Monte Carlo

Example 3: sampling an integral.

: = inside | ht = nt / nc, ddn os2t = 1.0f - nnt -D, N); D)

at a = nt - nc, b = nt at Tr = 1 - (R0 + (1 - R0 Fr) R = (D = nnt - N

= * diffuse = true;

efl + refr)) && (depth < MAXDEPT

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(&rand, I, &L, &Light) e.x + radiance.y + radiance.z) > 0) && (doing)

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) ();

andom walk - done properly, closely following Se /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, dodf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Considering the previous experiments, which stratum should be sample more often?

If we go from 8 to infinite strata, the probability of sampling a stratum becomes 0.

This is where we introduce the PDF, or *probability density function*. On a continuous domain, the probability of sampling a specific X is 0 (just like radiance arriving at a point is 0).

However, we can say something about the probability of choosing X in a *part of the domain*, by integrating the pdf over the subdomain. The pdf is a *probability density*.



Importance Sampling for Monte Carlo

Example 4: sampling the hemisphere.

: = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt ? D, N); ∂)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N - (dd)

= * diffuse = true;

efl + refr)) && (depth < MODEPTI

D, N); refl * E * diffus = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, if; radiance = SampleLight(&rand, I, &L, &L()) e.x + radiance.y + radiance.z) > 0) &&

w = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (real

andom walk - done properly, closely following Soul /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, Bpdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:





nics & (depth ≤ MAXON

c = inside ? 1 ht = nt / nc, ddn os2t = 1.0f - nnt 0, N); 3)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - RC) Tr) R = (D = nnt - N _ (100

= * diffuse = true;

-:fl + refr)) && (depth < MAXDEPTH

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, doing if; adiance = SampleLight(&rand, I, &L, &II) e.x + radiance.y + radiance.z) > 0) && doing

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) ((A)

andom walk - done properly, closely following Soul /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Importance Sampling for Monte Carlo

n

Example 4: sampling the hemisphere.

1/2π



 $\sqrt{2\pi}$

Importance Sampling for Monte Carlo

Monte Carlo without importance sampling:

$$E(f(X)) \approx \frac{1}{N} \sum_{i=1}^{N} f(X)$$

With importance sampling:

), N); refl * E * diffuse;

at a = nt - nc

AXDEPTH)

survive = SurvivalProbability(dit radiance = SampleLight(&rand, I,) e.x + radiance.y + radiance.z) > 0

v = true; at brdfPdf = EvaluateDiffuse(L, N at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf) at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely follow /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &; urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf); sion = true:



Here, p(x) is the *probability density function* (PDF).



Properties of a valid PDF p(x):

- 1. p(x) > 0 for all $x \in D$ where $f(x) \neq 0$
- 2. $\int_D p(x)d\mu(x) = 1$

at a = nt

efl + refr)) && (depth

), N); refl * E * diffuse;

AXDEPTH)

sion = true:

survive = SurvivalProbability lf: radiance = SampleLight(&rand, e.x + radiance.y + radiance.z)

v = true: at brdfPdf = EvaluateDiffuse(at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely foll /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf);

Probability Density Function

$1/_{2}\pi$ $-\frac{1}{2}\pi$

Note: p(x) *is a density, not a probability; it can (and will) exceed 1 for some x.*

Applied to direct light sampling:

p(x) = C for the part of the hemisphere covered by the light source

 \rightarrow C = 1 / solid angle to ensure p(x) integrates to 1

 \rightarrow Since samples are divided by p(x), we multiply by 1/(1/solid angle):

$$L_o(p,\omega_i) \approx lights * \frac{1}{N} \sum_{i=1}^N f_r(p,\omega_o,P) L_d^J(p,P) V(p \leftrightarrow P) \underbrace{\frac{A_{L_d^J} \cos \theta_i \cos \theta_o}{\|p-P\|^2}}_{\|p-P\|^2}$$



ics

c = inside ? 1 = 1 = 1 ht = nt / nc, ddn bs2t = 1.0f - nnt = 0 D, N); D)

at a = nt - nc, b = nt + nc at Tr = 1 - (R0 + (1 - R0 Γ r) R = (D = nnt - N = (100)

= * diffuse; = true;

• efl + refr)) && (depth < MAXDE

), N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed ff; radiance = SampleLight(&rand, I, &L, &L) e.x + radiance.y + radiance.z) > 0) &

v = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Probability Density Function

Applied to hemisphere sampling:

Light arriving over the hemisphere is cosine weighted.

→ Without further knowledge of the environment, the ideal PDF is the cosine function.

PDF: $p(\theta) = \cos \theta$

Question: how do we normalize this?

$$\int_{\Omega} \cos\theta d\theta = \pi \quad \Rightarrow \quad \int_{\Omega} \frac{\cos\theta}{\pi} d\theta = 1$$

Question: how do we choose random directions using this PDF?



nics & (depth < ™XXXXXIII

: = inside } 1 () | ht = nt / nc, ddn = bs2t = 1.0f - nπt D, N); 3)

at a = nt - nc, b = nt - nc, at Tr = 1 - (R0 + (1 - R0 Γr) R = (D = nnt - N = (dd)

= * diffuse; = true;

. efl + refr)) && (depth < MAXDEPii

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly, close
if;
radiance = SampleLight(&rand, I, &L, &l
e.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely follo /ive)

```
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

Cosine-weighted Random Direction

Without deriving this in detail:

A cosine-weighted random distribution is obtained by generating points on the unit disc, and projecting the disc on the unit hemisphere. In code:

float3 CosineWeightedDiffuseReflection()

```
float r0 = Rand(), r1 = Rand();
float r = sqrt( r0 );
float theta = 2 * PI * r1;
float x = r * cosf( theta );
float y = r * sinf( theta );
return float3( x, y, sqrt( 1 - r0 ) );
```

Note: you still have to transform this to tangent space.



Color Sample(Ray ray)

I, N, material = Trace(ray);

// continue in random direction

if (ray.NOHIT) return BLACK;

R = DiffuseReflection(N);

BRDF = material.albedo / PI;

 $Ei = Sample(r) * (N \cdot R) / PDF;$

// terminate if ray left the scene

// terminate if we hit a light source

if (material.isLight) return emittance;

// trace ray

Ray r(I, R);

// update throughput

PDF = 1 / (2 * PI);

return BRDF * Ei;

Importance Sampling

```
at a = nt - nc,
```

```
refl * E * diffuse;
```

```
AXDEPTH)
```

```
survive = SurvivalProbability
radiance = SampleLight( &rand)
e.x + radiance.y + radiance.z) > 0
```

```
v = true;
```

at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf) at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely follow /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, Apr urvive; pdf; n = E * brdf * (dot(N, R) / pdf);sion = true:

```
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;
- // continue in random direction
- R = CosineWeightedDiffuseReflection(N);

```
Ray r( I, R );
// update throughput
BRDF = material.albedo / PI;
PDF = (N \cdot R) / PI;
Ei = Sample( r ) * (N·R) / PDF;
return BRDF * Ei;
```



ics & (depth < NoCOS

z = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt o D, N); 0)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Tr) R = (D * nnt - N * (dd)

= * diffuse = true;

. :fl + refr)) && (depth < MODEL

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed Hf; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && closed

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Random Samples
- Next Event Estimation
- Importance Sampling
- Russian Roulette



```
efl + refr)) && (dept)
```

```
), N );
refl * E * diffuse;
```

```
AXDEPTH)
```

```
survive = SurvivalProbability( diff
radiance = SampleLight( &rand
e.x + radiance.y + radiance.z)
```

```
v = true;
at brdfPdf = EvaluateDiffuse( L
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, br
```

```
andom walk - done properly, Josely foll
/ive)
```

```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &
urvive;
pdf;
1 = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

```
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 \cdot L > 0 \& N \cdot L > 0) if (!Trace( lr ))
                    solidAngle = ((N1 \cdot -L) * A) / dist^2;
                    Ld = lightColor * solidAngle * BRDF * N.L;
                 // continue random walk
                R = DiffuseReflection( N );
                Ray r(I, R);
                Ei = Sample(r) * (N \cdot R);
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directurn PI * 2.0f * BRDF * Ei + Ld;
```

```
Color Sample( Ray ray )
   T = (1, 1, 1, 1), E = (0, 0, 0);
   while (1)
      I, N, material = Trace( ray );
      BRDF = material.albedo / PI;
      if (ray.NOHIT) break;
      if (material.isLight) break;
      // sample a random light source
      L, Nl, dist, A = RandomPointOnLight();
      Ray lr( I, L, dist );
      if (N \cdot L > 0 \& N \cdot L > 0) if (!Trace(lr))
         solidAngle = ((N1 \cdot -L) * A) / dist^2;
         lightPDF = 1 / solidAngle;
         E += T * (N·L / lightPDF) * BRDF * lightColor;
      // continue random walk
      R = DiffuseReflection( N );
      hemiPDF = 1 / (PI * 2.0f);
      ray = Ray(I, R);
      T *= ((N \cdot R) / hemiPDF) * BRDF;
   return E;
```

nics & (depth < Motos)

: = inside ? 1 ; ... ht = nt / nc, ddn os2t = 1.0f - n∩t ° 2, N); ≥)

at a = nt - nc, b = nt + r at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N * (dd

= * diffuse; = true;

. efl + refr)) && (depth < MAXDERII

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly if; radiance = SampleLight(%rand, I, %L, % 2.x + radiance.y + radiance.z) > 0) %%

w = true; at brdfPdf = EvaluateDiffuse(L, N) at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf

at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following SM /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Russian Roulette

Core idea:

The longer a path becomes, the less energy it transports.

Killing half of 16 rays is easy; what do we do with a single path?

 \rightarrow Kill it with a probability of 50%.



8 rays, returning 16 Watts of radiance each, 128 Watts in total.

= 4 rays, returning 32 Watts of radiance each, 128 Watts in total.



at a = nt

efl + refr)) && (depth

), N); refl * E * diffuse;

AXDEPTH)

sion = true:

survive = SurvivalProbability(dit lf; radiance = SampleLight(&rand, I, .x + radiance.y + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, | at3 factor = diffuse * INVPI at weight = Mis2(directPdf, brdfPdf at cosThetaOut = dot(N, L);

E * ((weight * cosThetaOut) / directPdf

= 4 rays, returning 32 Watts of radiance each, 128 Watts in total. andom walk - done properly, closely foll /ive)

at3 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 1 urvive; pdf; 1 = E * brdf * (dot(N, R) / pdf);

Russian Roulette

Russian roulette is applied to the random walk.

Most basic implementation: just before you start calculating the next random direction, you decide if the path lives or dies.

8 rays, returning 16 Watts of radiance each, 128 Watts in total.



ics & (depth < PACCO

at a = nt - nc, b = nt - n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D * nnt - N - (0

= * diffuse; = true;

. efl + refr)) && (depth < MAX



at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPd

andom walk - done properly, closely following /ive)

, t3 brdf = SampleDiffuse(diffuse, N, r1, r2, SR, Spdf) urvive; pdf; n = E * brdf * (dot(N, R) / pdf);

Better Russian Roulette

The termination probability of 50% is arbitrary. *Any probability is statistically correct.*

However: for 50% survival rate, survivors scale up by $2\left(=\frac{1}{50\%}\right)$. → In general, for a survival probability ρ , survivors scale up by $\frac{1}{\rho}$.

We can choose the survival probability *per path*. It is typically linked to albedo: the color of the last vertex. A good survival probability is:

$$\rho_{survive} = clamp\left(\frac{red + green + blue}{3}, 0.1, 0.9\right)$$

Note that $\rho_{survive} > 0$ to prevent bias. Also note that $\rho = 1$ is never a good idea.

Better: $\rho_{survive} = clamp(max(red, green, blue), 0, 1)$



ics & (depth < Motos

: = inside ? l | | | ht = nt / nc, ddn | os2t = 1.0f - nnt ?), N); ?)

at a = nt - nc, b = nt - e at Tr = 1 - (R0 + (1 - R0 Ir) R = (D = nnt - N - (dd)

= * diffuse; = true;

. efl + refr)) && (depth < MAXIII

D, N); refl * E * diffu = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) + radiance.z) > 0) && doing to e.x + radiance.y + radiance.z) + radiance.z)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Psurviv at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (??)

andom walk - done properly, closely following Sec /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

RR and **Next** Event Estimation

A path that gets terminated gets to keep the energy accumulated with Next Event Estimation.

We are applying Russian roulette to indirect illumination only.

D





RR

z = inside nt = nt / nc, ddr os2t = 1.0f - nnt 0, N); 0)

at a = nt - nc, b = nt - n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D ⁼ nnt - N ⁻ (dd)

= * diffuse; = true;

```
.
efl + refr)) && (depth < MAXDEPT
```

D, N); refl * E * diffuse; = true;

AXDEPTH)

```
survive = SurvivalProbability( diffuse
estimation - doing it properly, closed
f;
radiance = SampleLight( &rand, I, &L &L
e.x + radiance.y + radiance.z) > 0) &&
```

w = true; at brdfPdf = EvaluateDiffuse(L, N) Ps at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

```
andom walk - done properly, closely followin
/ive)
```

```
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, & return E;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true:
```

Color Sample(Ray ray) T = (1, 1, 1), E = (0, 0, 0);while (1) I, N, material = Trace(ray); BRDF = material.albedo / PI; if (ray.NOHIT) break; if (material.isLight) break; // sample a random light source L, Nl, dist, A = RandomPointOnLight(); Ray lr(I, L, dist); if $(N \cdot L > 0 \& Nl \cdot -L > 0)$ if (!Trace(lr))solidAngle = $((Nl \cdot -L) * A) / dist^2;$ lightPDF = 1 / solidAngle; E += T * (N·L / lightPDF) * BRDF * lightColor; // Russian Roulette

> p = SurvivalProb(material.albedo); if (p < Rand()) break; else /* whew still alive */ T *= 1/p; // continue random walk R = DiffuseReflection(N); hemiPDF = 1 / (PI * 2.0f); ray = Ray(I, R); T *= ((N·R) / hemiPDF) * BRDF;

Service Se

ics & (depth < NoCOS

z = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt o D, N); 0)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 Tr) R = (D * nnt - N * (dd)

= * diffuse = true;

. :fl + refr)) && (depth < MODEL

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed Hf; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) && closed

v = true; at brdfPdf = EvaluateDiffuse(L, N) Psurvi at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

andom walk - done properly, closely following Sou. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Today's Agenda:

- Introduction
- Random Samples
- Next Event Estimation
- Importance Sampling
- Russian Roulette



hics & (depth < Motos

: = inside ? 1 |]] ht = nt / nc, ddn =] bs2t = 1.0f - nnt = on D, N); B)

at a = nt - nc, b = nt + nc at Tr = 1 - (R0 + (1 - R0) Fr) R = (D ⁼ nnt - N = (ddn)

= * diffuse; = true;

-:fl + refr)) && (depth < MONDEPTH

D, N); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) e.x + radiance.y + radiance.z) > 0) &&

v = true; at brdfPdf = EvaluateDiffuse(L, N) Pourvive at3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) (nad

andom walk - done properly, closely following Sec. /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

INFOMAGR – Advanced Graphics

Jacco Bikker - November 2021 - February 2022

END of "Variance Reduction (2)"

next lecture: "Various"

