tics & (depth < >voca

: = inside ? 1 ()) ht = nt / nc, ddn bs2t = 1.0f - nnt " ∩ 2, N); 2)

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

= * diffuse = true;

efl + refr)) && (depth < MAXDEPTI

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly ff; radiance = SampleLight(&rand, I c.x + radiance.y + radiance.z) > **1 (2**

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 Sec /ive)

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

= g(x, x')

INFOMAGR – Advanced Graphics

Jacco Bikker - November 2021 - February 2022

Lecture 15 - "Filtering"

(x,x',x'')I(x',x'')dx''

Welcome!



ics & (depth < MAXDE)

: = inside ? l | ... ht = nt / nc, ddn os2t = 1.0f - nnt = n O, N); 0)

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 * diffus = true;

AXDEPTH)

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

w = 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) (and

andom walk - done properly, closely following Sour /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
- Ingredients
- Future Work



sics & (depth < ≯vocc

: = inside ? 1 ht = nt / nc, ddn ss2t = 1.0f - nnt ∩ n D, N); ≷)

at a = nt - nc, b = Ht = n at Tr = 1 - (R0 + (1 - R0 Fr) R = (D - nnt - N - (33)

= * diffuse; = true;

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

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

AXDEPTH)

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

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) (rad)

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

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



tics & (depth < ≥vocci

- inside - i ddn - i so nt = nt / nc, ddn - i so sost = 1.0f - nnt - ont 2, N); 3)

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

= * diffuse; = true;

efl + refr)) && (depth < MADE

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

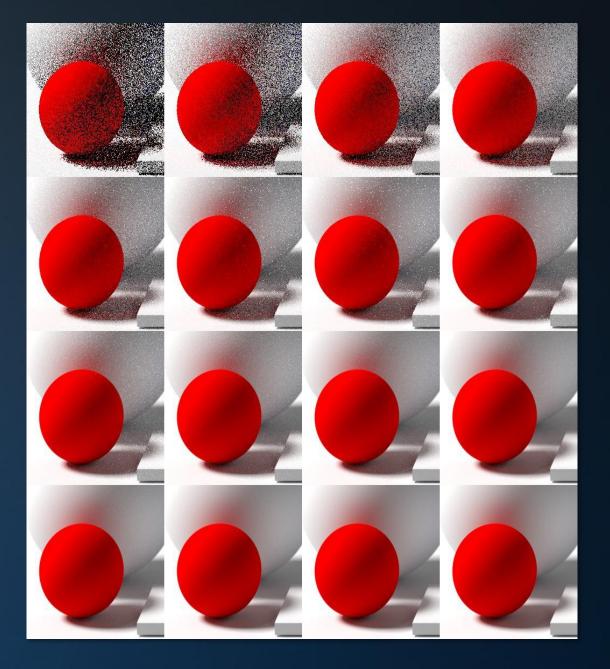
AXDEPTH)

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

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) * (radial);

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

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





Last Time

ics & (depth < ⊅00000

: = inside ? 1 1 1 0 ht = nt / nc, ddn 0 1 ss2t = 1.0f - nnt ° n 2, N); ≫)

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 * diffus = true;

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly f; radiance = SampleLight(&rand, I, &L 2.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) (nad

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

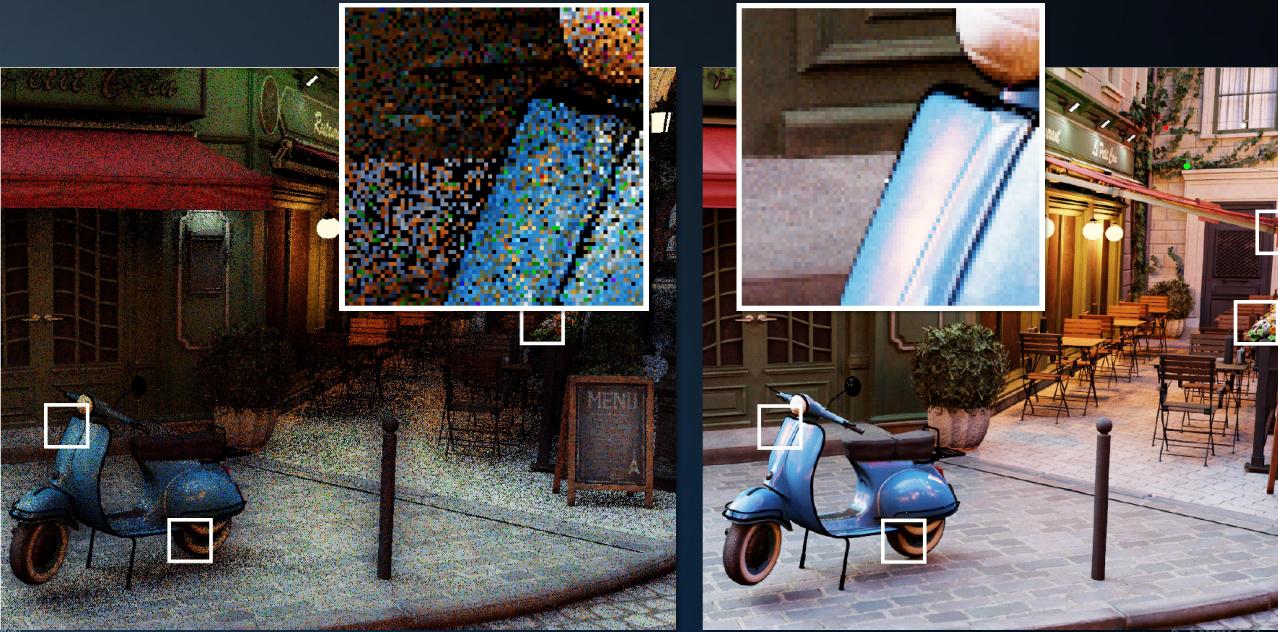
; at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Towards Noise-free Path Tracing

"Work smarter, not harder": generate better samples / send rays where they matter.

Extreme case: ReSTIR, which spends a lot of effort on deciding where to send a shadow ray.





st3 brdf = SampleDiffuse(diffuse, N, r1, "Rearchitecting Spatiotemporal Resampling for Production", Wyman & Panteleev, 2021.
pdf;
n = E * brdf * (dot(N, R) / pdf);
sign = true:



Last Time

We tried everything

...But with an 8spp budget, it's still noisy.

- There is somewhat uniform noise left
- 'Fireflies' indicate presence of 'improbable paths'.





7

Last Time

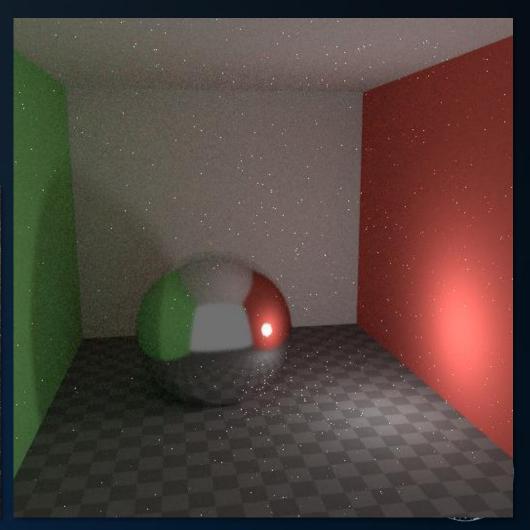
We tried everything

...But with an 8spp budget, it's still noisy.

- There is somewhat uniform noise left
- 'Fireflies' indicate presence of 'improbable paths'.







Last Time

tics & (depth < Not000

: = inside ? 1 1 1 1 ht = nt / nc, ddn 4 1 ps2t = 1.0f - nmt 4 nn D, N); D)

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

= * diffuse; = true;

efl + refr)) && (depth < MODEPTI

), N); refl * E * diffu: = 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) * Ps 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:

Suppressing Fireflies

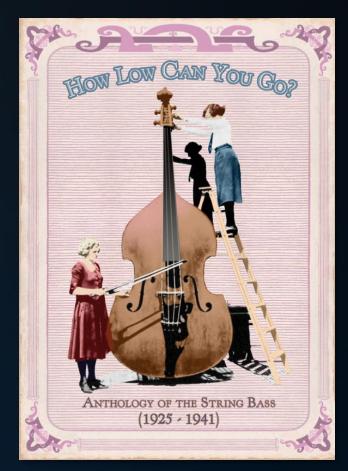
"A firefly is easily recognized in the final image: it is a pixel with a value that differs significantly from its neighbors."

- Is this always true?
- How to fix it?

Is that still correct?

Firefly suppression introduces bias in our estimator.

- Spread out the removed energy over the image / neighborhood
- Just wait it out (additional samples will improve the average)
- Do some adaptive sampling (detect high variance)
 - Just accept it.





Last Time

ics & (depth < MoxOC

: = inside ? 1 = 1 00 ht = nt / nc, ddn = 0 s2t = 1.0f - nnt = n 2, N); ≫)

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

= * diffuse; = true;

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

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

AXDEPTH)

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

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) (real

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

: at3 brdf = SampleDiffuse(diffuse, N, r1, r2*: The Iray Light Transport Simulation and Rendering System, Section 5.5. Keller et al., 2017. pdf; pdf * (det(N, R,) (adf);

. = E * brdf * (dot(N, R) / pdf); sion = true:

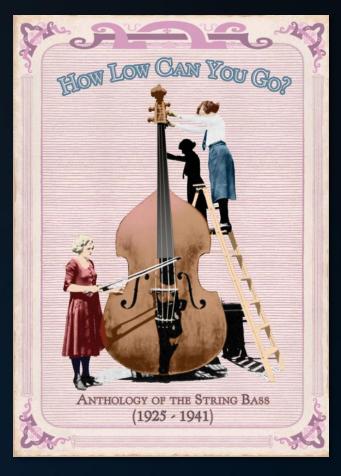
Suppressing Fireflies

"A firefly is easily recognized in the final image: it is a pixel with a value that differs significantly from its neighbors."

Better approach: *clamp*.*

e.g., in Lighthouse 2:

#define CLAMPINTENSITY(E) \
 if (dot(E, E) > 25) E = 5 * normalize(E);





Last Time

hics & (depth < ™0000

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

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

= * diffuse; = true;

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

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

AXDEPTH)

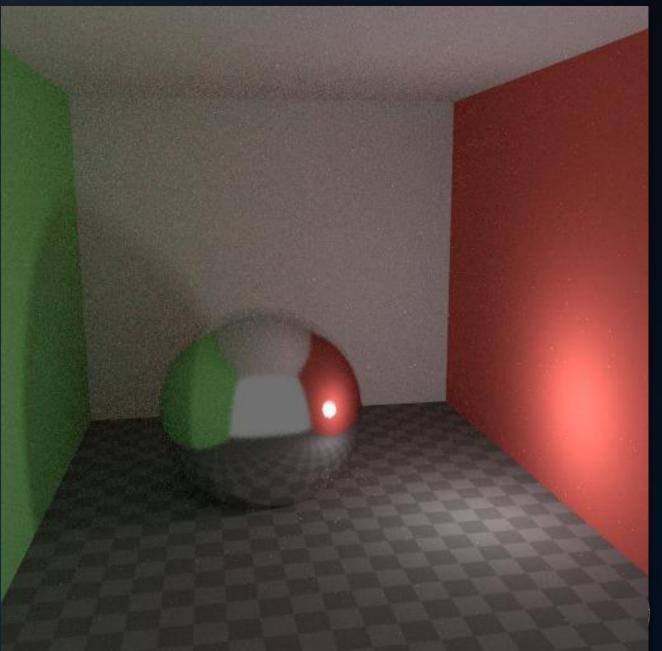
survive = SurvivalProbability(diffuse
estimation - doing it properly
if;
radiance = SampleLight(&rand, I, &L, &light)
extinue.x + radiance.y + radiance.z) > 0) &&

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)

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

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





tics & (depth < NOCCO

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

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

= * diffuse = true;

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

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

AXDEPTH)

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

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) * (nad)

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

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

Today's Agenda:

Noise

- Ingredients
- Future Work



Ingredients

sics & (depth < NOCC)

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

= * diffuse; = true;

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

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

AXDEPTH)

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

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 fol /ive)

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

Reducing the Problem - Filtering

Core idea:

Exploit the fact that illumination is typically low-frequent: Nearby pixels tend to converge to similar values, so we should be able to use information gathered for one pixel to improve the estimate of the next.

Essentially, we are increasing the number of samples per pixel, by including the neighbors.

Note:

Unless neighboring pixels actually converge to the same value, filtering introduces bias.

Filtering thus trades variance for bias.



Ingredients



z = inside ? 1 ht = nt / nc, ddn bs2t = 1.0f - nnt on D, N); 3)

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

= * diffuse; = true;

efl + refr)) && (depth < MODECCO

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

AXDEPTH)

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

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

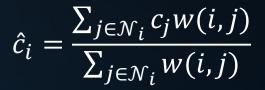
andom walk - done properly, closely following Sec. /ive)

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

Filter kernels

For the actual filtering, we apply a kernel.

Pixel FilteredValue(i_x , i_y , halfWidth) sum = 0 summedWeight = 0for $j_x = i_x$ - halfWidth to i_x + halfWidth for $j_y = i_y$ - halfWidth to i_y + halfWidth $sum += \text{ReadPixel}(j_x, j_y) * \text{weight}(j_x, j_y)$ $summedWeight += \text{weight}(j_x, j_y)$





Ingredients



AXDEPTH)

= true

sion = true

lf: adiance

Filter kernels

For the actual filtering, we apply a kernel.

Pixel FilteredValue(i_x , i_y , halfWidth) sum = 0summedWeight = 0

 $\hat{c}_i = \frac{\sum_{j \in \mathcal{N}_i} c_j w(i, j)}{\sum_{j \in \mathcal{N}_i} w(i, j)}$

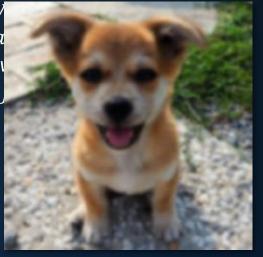




um umnsum



 $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$



 $\frac{1}{22} \begin{bmatrix} 1 & 3 & 1 \\ 3 & 6 & 3 \\ 1 & 3 & 1 \end{bmatrix}$



Ingredients



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

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

= * diffuse; = true;

efl + refr)) && (depth < MODECCO

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; 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) * (

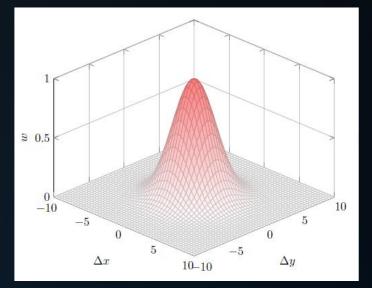
andom walk - done properly, closely followin /ive)

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

Filter kernels

For the actual filtering, we apply a kernel.

Pixel FilteredValue(i_x , i_y , halfWidth) sum = 0 summedWeight = 0for $j_x = i_x$ - halfWidth to i_x + halfWidth for $j_y = i_y$ - halfWidth to i_y + halfWidth $sum += \text{ReadPixel}(j_x, j_y) * \text{weight}(j_x, j_y)$ $summedWeight += \text{weight}(j_x, j_y)$



Here, weight or w is the weight function. We could simply use the Gaussian kernel:

 $w(i,j) = \exp\left(\frac{-\|p_i - p_j\|^2}{2\sigma_d^2}\right),$

where p_i and p_j are screen space positions and σ_d is the spatial standard deviation of the Gaussian kernel.



Ingredients



: = inside ? 1 = 1.3 ht = nt / nc, ddn = 3 ss2t = 1.0f - nnt ⊂ n 2, N); ≷)

at a = nt - nc, b = nt - nc at Tr = 1 - (R0 + (1 - R0 rr) R = (D = nnt - N - (d0)

= * diffuse; = true;

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

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

AXDEPTH)

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

andom walk - done properly, closely fo /ive)

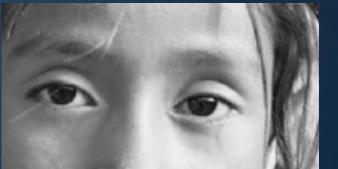
; at3 brdf = SampleDiffuse(diffuse, N, r1 urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Filter kernels

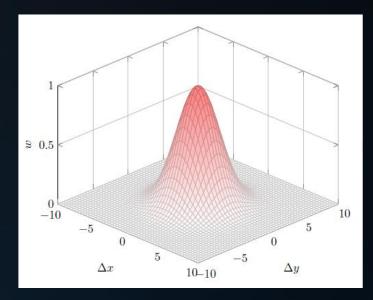
A Gaussian filter (as well as other low-pass filters) blurs out high frequency details.

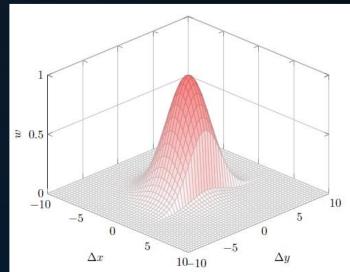
We can improve on this using a non-linear bilateral filter*.

$$\psi(i,j) = \exp\left(\frac{-\left\|p_i - p_j\right\|^2}{2\sigma_d^2}\right) \times \exp\left(\frac{-\left\|c_i - c_j\right\|^2}{2\sigma_r^2}\right)$$













*: Tomasi & Manduchi, Bilateral filtering for gray and color images. ICCV '98.

Ingredients



: = inside ? 1 a 1 d nt = nt / nc, ddn a d os2t = 1.0f - nnt " nn), N); ≫)

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

= * diffuse; = true;

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

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed) if; 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) / directPd

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

; it3 brdf = SampleDiffuse(diffuse, N, r1, r2, prvive; pdf; n = E + brdf * (dot(N, R) / pdf); sion = true:

Filter kernels

The bilateral filter takes the color of nearby pixels into account. We can take this further, by taking an arbitrary set of features into account.

The cross bilateral filter*:

$$\Psi(i,j) = \exp\left(\frac{-\|p_i - p_j\|^2}{2\sigma_d^2}\right) \times \prod_{k=1}^K \exp\left(\frac{-\|f_{k,i} - f_{k,j}\|^2}{2\sigma_k^2}\right)$$

Here, $f_{k,i}$ is the k'th feature vector at pixel i and σ_k is the bandwidth parameter for feature k.

Note that we can use noise-free features to smooth noisy features. Example of a low-noise feature: normals at the primary intersection point. Example of a noisy feature: indirect illumination at the primary intersection point.



*: Eisemann & Durand. Flash photography enhancement via intrinsic relighting. ACM Trans. Graph. 23, 3 (Aug. 2004).

Ingredients



: = inside ? 1 : 1 : 1 ht = nt / nc, ddn : 1 ss2t = 1.0f - nnt " n ,, N); ≫)

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

= * diffuse; = true;

efl + refr)) && (depth < MOG

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

AXDEPTH)

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

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:

Filter kernels, digest

Filtering adds samples to a pixel by 'borrowing' them from neighbors. Filtering trades variance for bias.

We can improve the quality of the borrowed samples using a weight:

- Further away = less relevant
- Different normal, different material, ... = less relevant

Some considerations:

Should we take accumulated or individual samples from neighbors?Depth of field and AA seriously affect our options.



Ingredients





= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPII

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, closed if; radiance = SampleLight(&rand, I, &L, &light) 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 f /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1 urvive; pdf; n = E * brdf * (dot(N, R) / pdf); icn = true; Separating illumination into layers allows us to filter each layer separately. This prevents bleeding, and allows for layer-specific kernel sizes.

We can also separate albedo from illumination.

Indirect illumination as a feature: A path tracer allows us to conveniently split direct from indirect, and bounce 1 from bounce 2.





Ingredients





= * diffuse; = true;

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

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

AXDEPTH)

survive = SurvivalProbability(diffuse estimation - doing it properly, close 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 following Sa /ive)

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

Separating albedo from illumination



Adding this separation to an existing renderer:

- store albedo at the primary intersection (simple material property);
 - at the end of the pipeline: illumination = sample / max(epsilon, albedo).



Reprojection

Core idea:

Ingredients



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

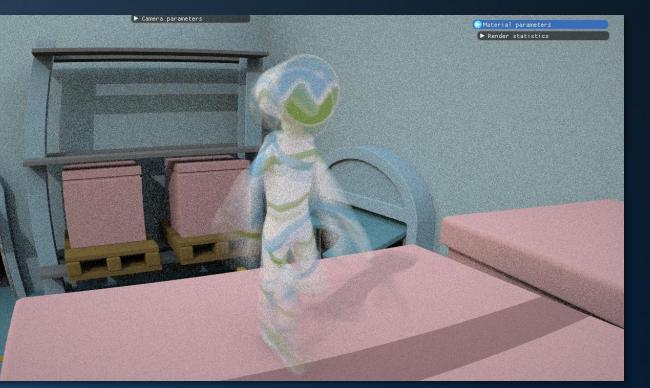
AXDEPTH)

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) * Ps 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, urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true: In an animation, samples taken for the previous frame are meaningful for the current frame. *We can supply the filter with more data by looking back in time.*





Reprojection

Core idea:

Ingredients



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) && (doing to be a closed)

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 /ive)

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

In an animation, samples taken for the previous frame are meaningful for the current frame. *We can supply the filter with more data by looking back in time.*

Problem: in an animation, the camera and/or the geometry moves. We need to find the location of a pixel in the previous frame(s).

Solution: use the camera matrices.

$$M_{4x4} \begin{pmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{pmatrix} = \begin{pmatrix} x_{screen} \\ y_{screen} \\ z_{screen} \\ 1 \end{pmatrix} \rightarrow M_{4x4}^{-1} \begin{pmatrix} x_{screen} \\ y_{screen} \\ z_{screen} \\ 1 \end{pmatrix} = \begin{pmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{pmatrix}$$

(finally, apply the matrix of the previous frame to obtain the screen location in the previous frame.)





https://www.shadertoy.com/view/ldtGW

Ingredients



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

AXDEPTH)

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

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 S /ive)

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

Reprojection

Reprojection using camera matrices:

- fails if we have animation
- will not work with depth of field
- will not work with speculars.

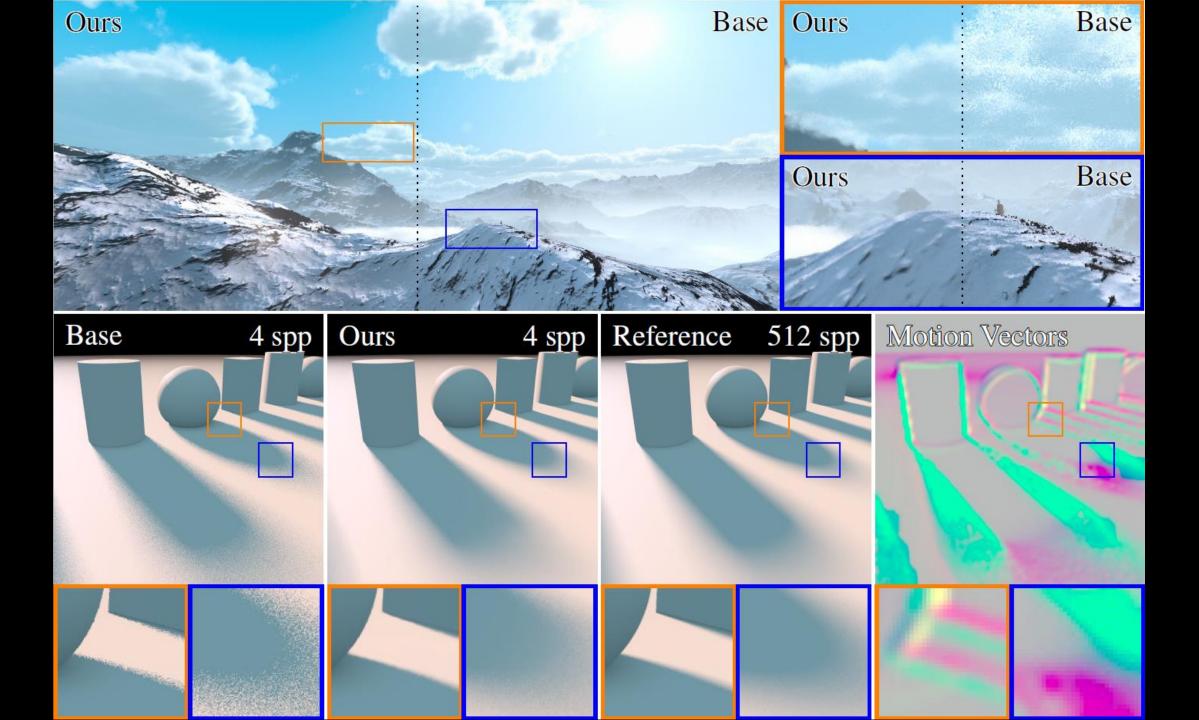
A recent paper proposes an alternative*:

For each pixel (i,j), find the shift to similar pixels in the neighborhood by comparing a small patch of pixels around (i,j) to pixels at some distance.

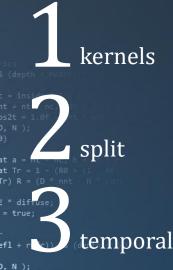
Note: this idea is not new, but the paper makes it efficient using a hierarchical process, where down-sampled versions of the image are used to increase the size of the search window.

*: Fast Temporal Reprojection without Motion Vectors. Hanika & Tessari, 2021.





Ingredients



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) Psi 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, urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Caching in world space

Instead of searching the current pixel in the previous frame in screen space, we can also maintain a cache in world space*.

Path space filtering:

- Store information in a 3D grid
- Map the grid cells to a hash map
- Update grid cells for each vertex that 'visits' it

Note that a single cell may still receive shading information for surfaces with different normals.

*: Binder et al., Massively Parallel Path Space Filtering, 2019.



Adaptive Sampling

to realtime rendering.

Some pixels need more samples than others.

(to reach a certain variance level)

Ingredients

kernels split temporal

radiance.v + radiance

adaptive

EvaluateDiffuse at3 factor at weight = Mis2(directPdf, brdfPd at cosThetaOut = dot(N, L E * ((weight * cosThetaOut) / directProvide the cost of the cos

andom walk - done properly

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

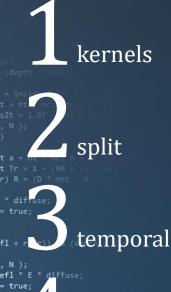
*: A Survey of Adaptive Sampling in Realistic Image Synthesis, M. Sik, 2013.

Adaptive Sampling^{*} aims to estimate which pixels still need work.

Note that reliable variance estimation requires more than



Ingredients



AXDEP urvive = stivaladaptive estimation - doing it property

n; radiance = SampleLight(&rand, I, &L, &L r.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) / direct?

andom walk - done properly, closely fo /ive)

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

Variance-guided Filtering*

A variance estimate is also useful for steering the filter kernel size:

- A pixel with low variance can use a small kernel *(which prevents overblurring)*
- A pixel with high variance needs a larger kernel *(to include more samples from neighbors)*

SVGF combines bilateral filtering with variance guided kernel sizes and temporal reprojection.



*: Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. Schied et al., 2017.



Ingredients



https://dspace.library.uu.nl/bitstream/handle/1874/366198/Beyond%20SVGF.pdf at weight = Mis2(directPdf, brdfPdf

/ive)

at brdfPdf = EvaluateDiffuse(L, N

E * ((weight * cosThetaOut) / directPdf)

at3 factor = diffuse * INVPI;

at cosThetaOut = dot(N, L);

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



Ingredients





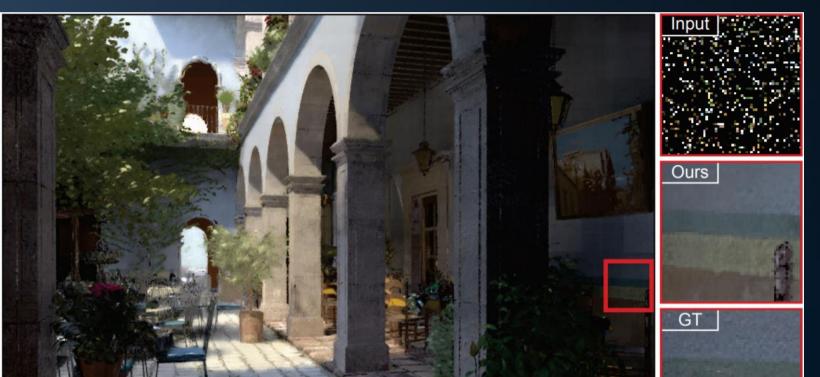
andom walk - done properly, closely follo /ive)

; at3 brdf = SampleDiffuse(diffuse, N, r1, r; urvive; pdf; n = E * brdf * (dot(N, R) / pdf); ien = true;

Machine Learning

Neural networks can be used to filter path tracing noise.

E.g., by learning optimal filter parameters: A Machine Learning Approach for Filtering Monte Carlo Noise. Kalantari et al., 2015.





Ingredients





andom walk - done properly, closely follo /ive)

Machine Learning

Reinforcement Learning can be used to importance sample based on experience.

E.g., by learning light transport while rendering: Learning Light Transport the Reinforced Way. Dahm & Keller, 2017.





Ingredients



Machine Learning

Reinforcement Learning can be used to importance sample based on experience.

temporal adaptive



at3 brdf = SampleDiffuse(diffuse. pdf = E * brdf * (dot(N, R) / pdf); Reinforcement Learning for rendering is often referred to as path guiding: Path Guiding in Production. Vorba et al., 2019 (SIGGRAPH 2019 course).





Ingredients

kernels split temporal adaptive

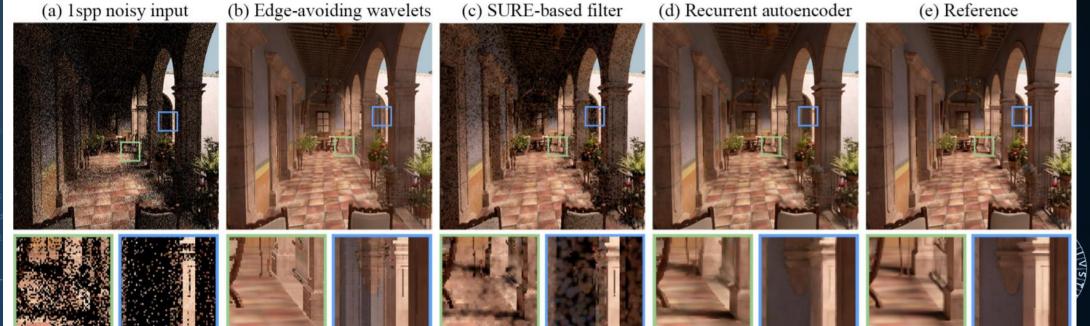
= true brdfhat valuateDiffuse(L, N) 3 factor =) ffuse * INVPI; weight = V 2(iearfning); cosinctadut = dot(N, L); * ((weight * cosThetaOut) / directPd

ndom walk - done properly, closely /ive)

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

And finally: convolutional neural networks.

Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. Disney / Pixar, University of California: Bako et al., 2019. Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder. NVIDIA, several universities: Chaitanya et al., 2017.



tics & (depth < NOCCO

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

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

= * diffuse = true;

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

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

AXDEPTH)

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

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) * (nad)

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

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

Today's Agenda:

Noise

- Ingredients
- Future Work



Digest

Filtering, practical

First of all, provide a high-quality render:

- Few samples can still be HQ samples
- Many filters get more expensive with high spp counts \rightarrow spend more time per sample

Prepare your input:

- Separate albedo and illumination
- Separate direct and indirect light
- Suppress outliers
- Supply 'feature buffers' for the bilateral kernels
- Use a pinhole camera postpone AA / DOF
- Reproject; go temporal.

Filter:

- Some form of bilateral
 - Steer kernel size with variance estimation
 - Ideally: sample-based; pixel-based if this is too slow



nside ? 1 = 1 nt / nc, ddn = 1.0f - nnt);

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

= * diffuse; = true;

• efl + refr)) && (depth < MAXDEPTI

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

AXDEPTH)

survive = SurvivalProbability(diffuse
estimation - doing it properly
if;
radiance = SampleLight(&rand, I, &L, &

e.x + radiance.y + radiance.z) > 0) 88

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) / <u>directPdf</u>

andom walk - done properly, closely follo

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

Digest

Filtering, open problems

Not easy to do:

- DOF, AA
- Transparency

Considerations for real-time:

- Mind temporal stability
- Don't make it too crisp
- Make some (uniform) noise a feature
- Consider using DLSS

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

v = true;

AXDEPTH)

- at brdfPdf = EvaluateDiffuse(L, N) * Psurvis 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)

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



hics ≹ (depth < NACCS

t = inside } l ht = nt / nc, ddn bs2t = 1.0f - n⊓t - n D, N); ∂)

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

= * diffuse; = true;

• •fl + refr)) && (denth < Max

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

AXDEPTH)

survive = SurvivalProbability(diff. estimation - doing it properly, cl 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) / direc

andom walk - done properly, close /ive)

, H33 brdf = SampleDiffuse(diffuse, N, r1, r2, 8R, 8pdf urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:





tics & (depth < NOCCO

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

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

= * diffuse = true;

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

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

AXDEPTH)

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

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) * (nad)

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

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

Today's Agenda:

Noise

- Ingredients
- Future Work



hics & (depth < No000

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

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

= * 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) 2.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)

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

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

INFOMAGR – Advanced Graphics

Jacco Bikker - November 2021 – February 2022

END of "Filtering"

next lecture: "Bits & Pieces, Exam Training"

