REAL-TIME RAY TRACING OF CORRECT* SOFT SHADOWS

(* without a shadow of a doubt)

Stephen Hill, Lucasfilm

Morgan McGuire, NVIDIA

Eric Heitz, Unity Technologies

[Stephen's section]

This talk is about our recent I3D paper, with more details about the implementation and performance.

We'll also be asking and answering the question: what is a *correct* soft shadow?

(Apologies for the pun in the title.)

BACKSTORY

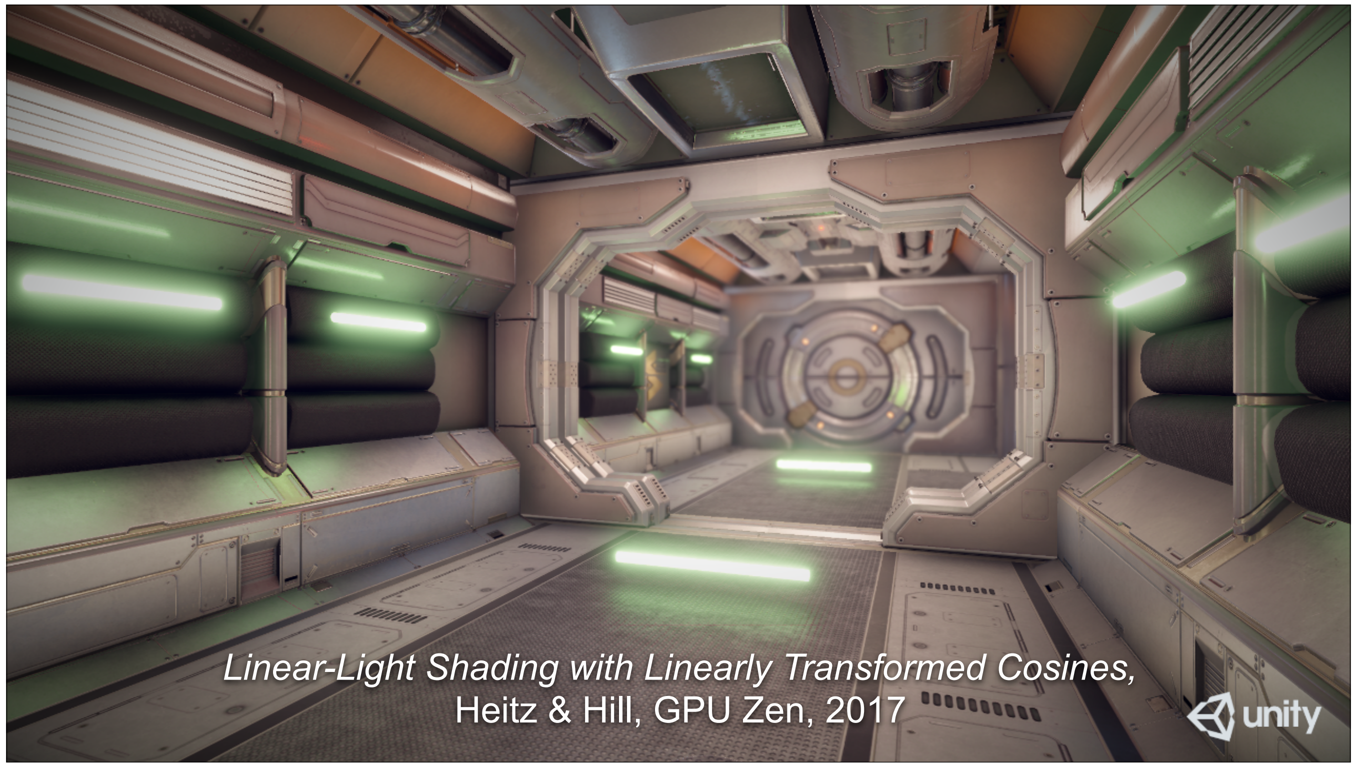
First let me recap the journey we’ve been on from two years ago…

Real-Time Polygonal Light Shading with Linearly Transformed Cosines,
Heitz et al., SIGGRAPH 2016 technical paper

It all started with a collaboration, between Eric, Jonathan Dupuy, myself and Dave Neubelt.

We developed an efficient solution for shading with polygonal lights, based on a new spherical distribution: Linearly Transformed Cosines (LTCs).

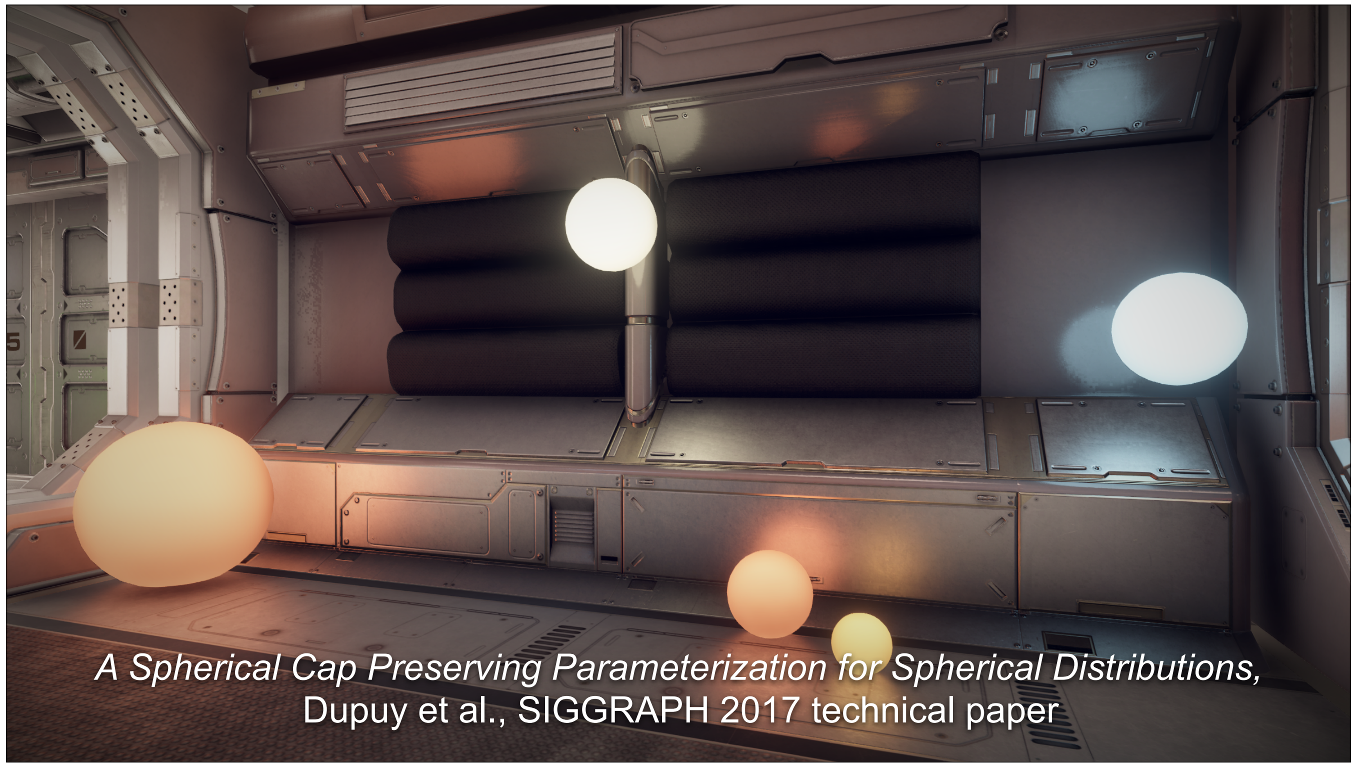
More on this in a bit.


Linear-Light Shading with Linearly Transformed Cosines,
Heitz & Hill, GPU Zen, 2017
unity

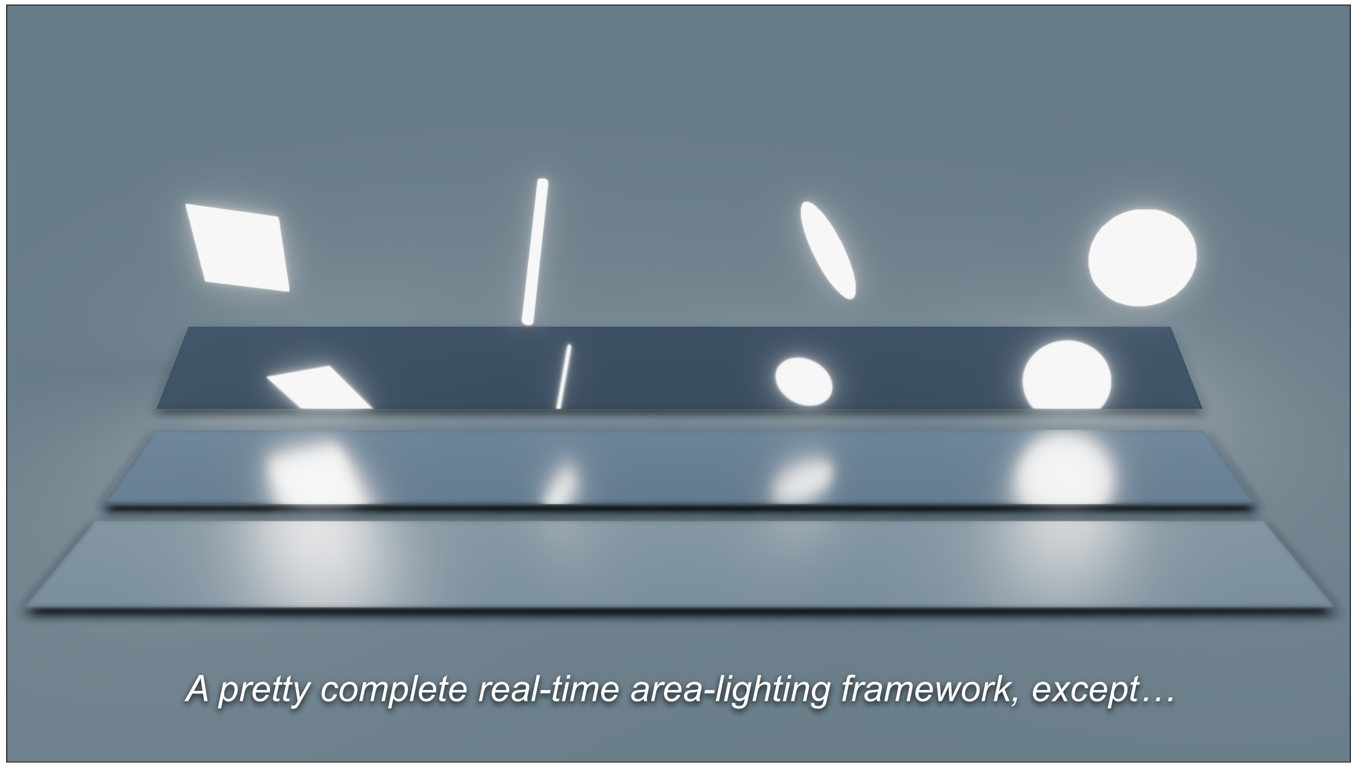
Eric and I later extended this to line lights…

Real-Time Line- and Disk-Light Shading with Linearly Transformed Cosines,
Heitz & Hill, Physically Based Shading course, SIGGRAPH 2017

…and disk lights too.


A Spherical Cap Preserving Parameterization for Spherical Distributions,
Dupuy et al., SIGGRAPH 2017 technical paper

In parallel, the folks at Unity developed another spherical distribution that has good properties for sphere lights.


A pretty complete real-time area-lighting framework, except…

So, after all of this, we arrived at a pretty complete area-lighting framework that supports a range of common area-light types.

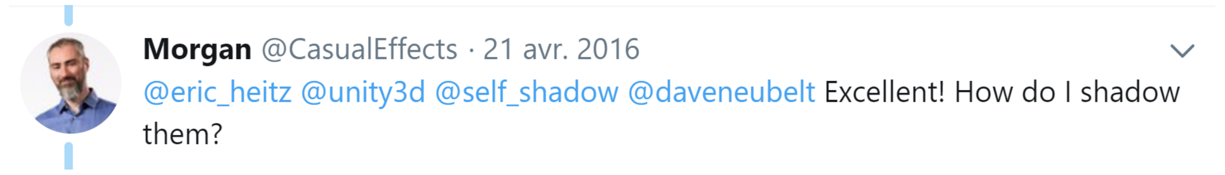
BACKSTORY

Morgan @CasualEffects · 21 avr. 2016

@eric_heitz @unity3d @self_shadow @daveneubelt Excellent! How do I shadow them?

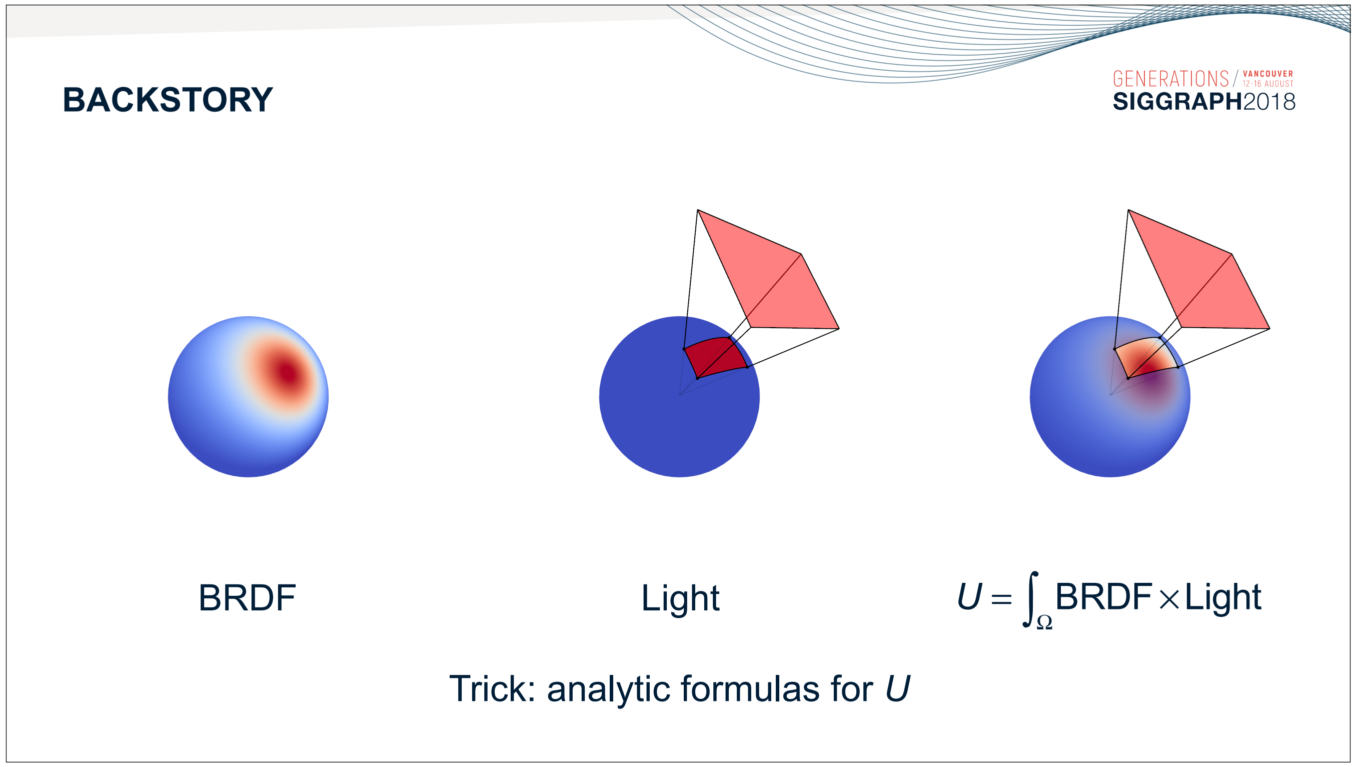
The major limitation was that these lights cannot be shadowed, at least accurately.

BACKSTORY


Morgan @CasualEffects · 21 avr. 2016
@eric_heitz @unity3d @self_shadow @daveneubelt Excellent! How do I shadow them?

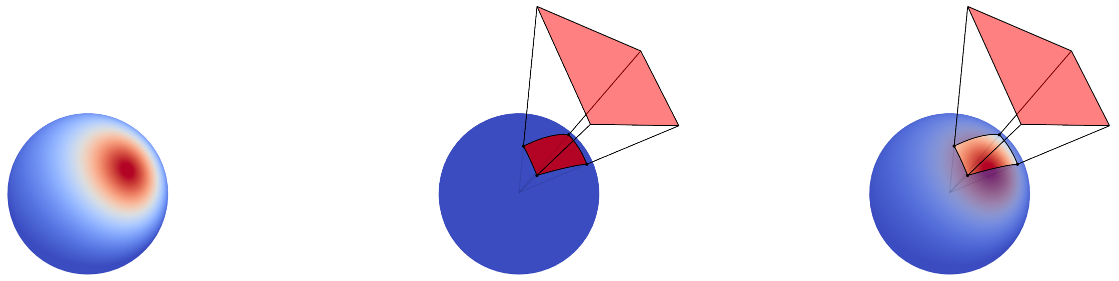
Thanks Morgan!

Thanks Morgan!



BACKSTORY

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018



Trick: analytic formulas for *U*

The reason why our area lights cannot be shadowed is related to how they work.

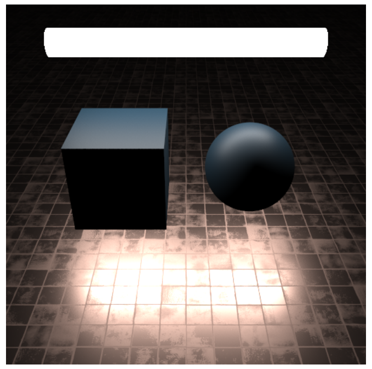
Shading a pixel with an area light means considering two spherical functions — the BRDF and the light — and computing their integral.

The trick with our framework was to find analytical solutions for this integral, thanks to new spherical distributions that we designed.

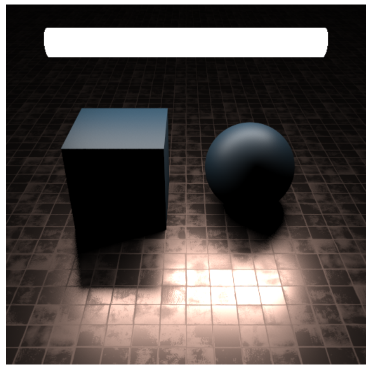
BACKSTORY

$U = \int_\Omega \text{BRDF} \times \text{Light}$

$S = \int_\Omega \text{BRDF} \times \text{Light} \times \text{Visibility}$



$\overset{?}{\longrightarrow}$



Unshadowed illumination
(what we have)

Shadowed illumination
(what we want)

However, we were only able to compute *U*, the *unshadowed* direct illumination. What we really want is *S*, the *shadowed* direct illumination. The difference, of course, is the presence of a *visibility* function.

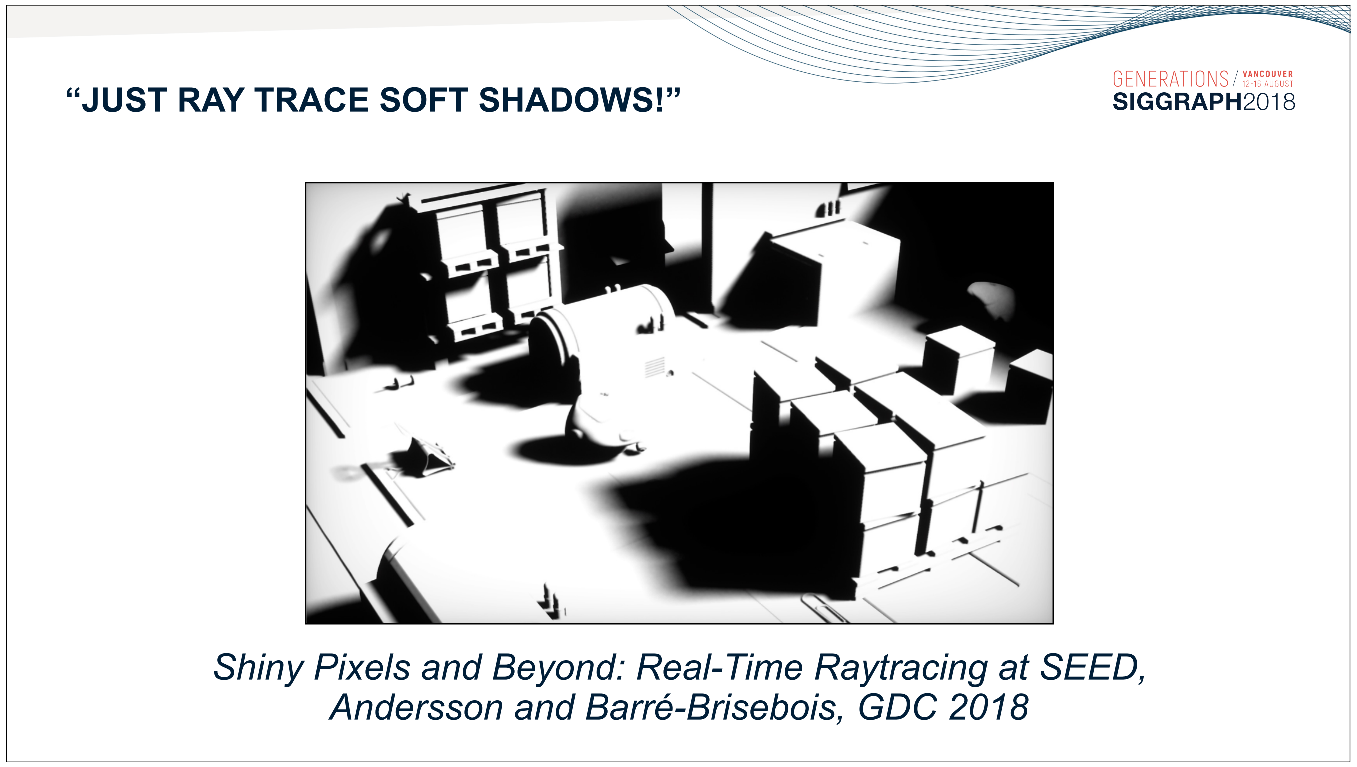
This is where we started: we had a good solution for *U*, but what we want is *S*.

What should we do?

OPTION 1: “JUST RAY TRACE SOFT SHADOWS!”

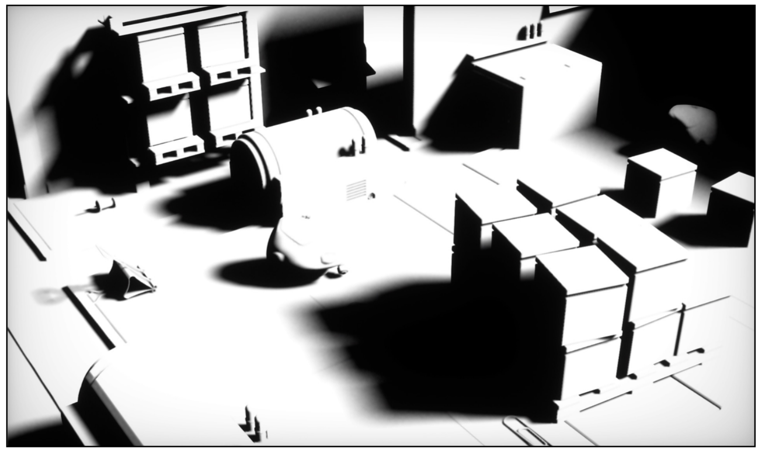
Now, you might think: well, let’s use DirectX Raytracing (DXR).

Ray tracing will surely give us nice-looking results…



"JUST RAY TRACE SOFT SHADOWS!"

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

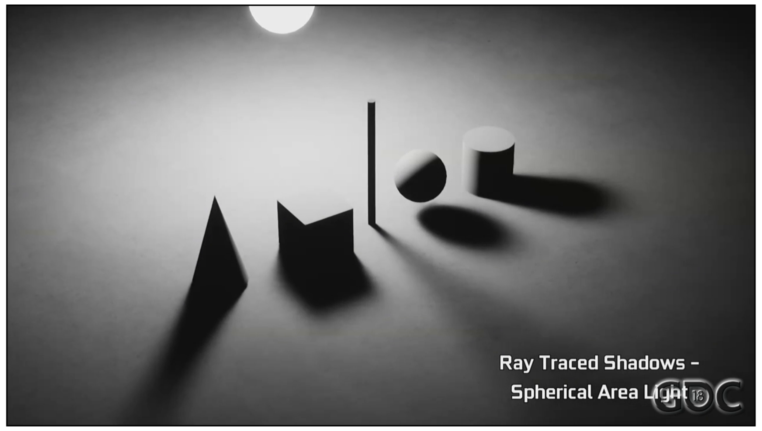


Shiny Pixels and Beyond: Real-Time Raytracing at SEED,
Andersson and Barré-Brisebois, GDC 2018

…and that's exactly what some recent demos have done.

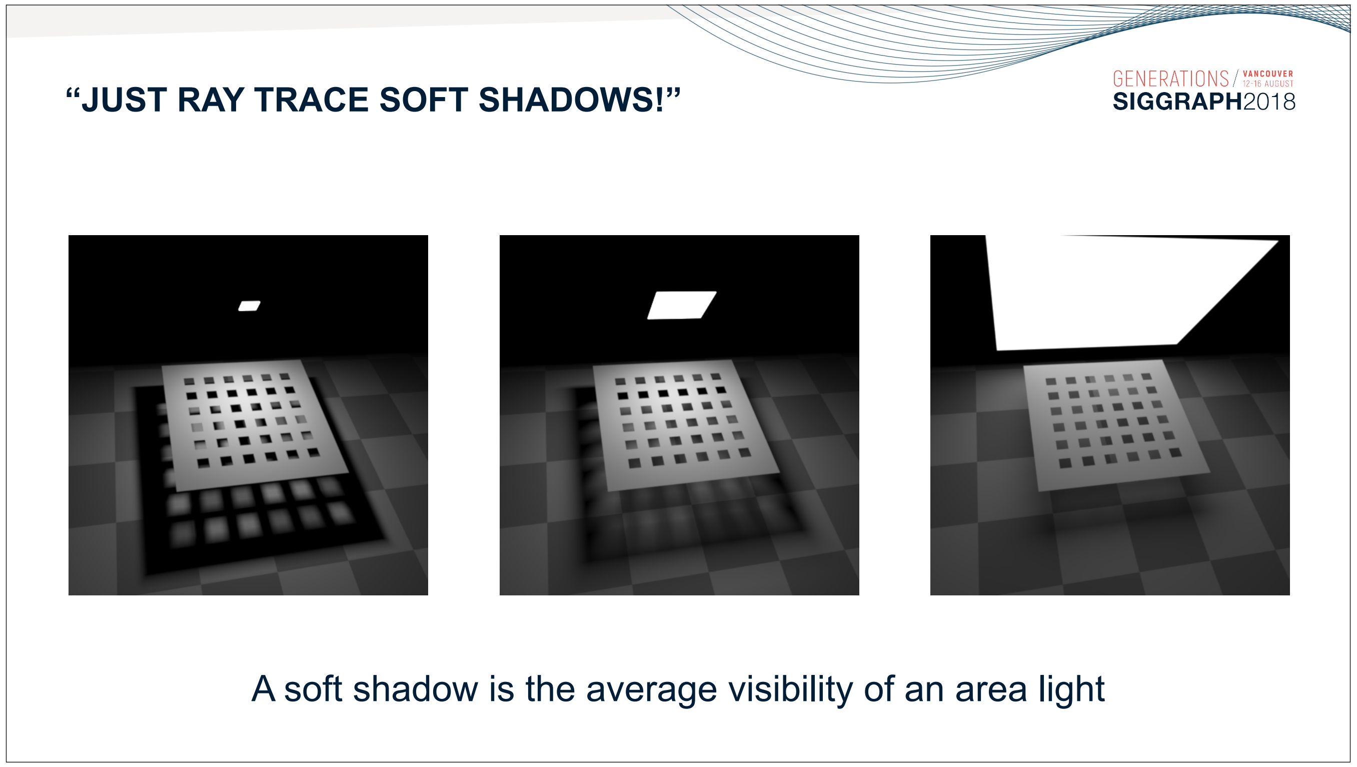
Here, the folks from SEED did ray-traced shadows — using cone sampling of a spherical cap for sun shadows — and the results look good.

"JUST RAY TRACE SOFT SHADOWS!"



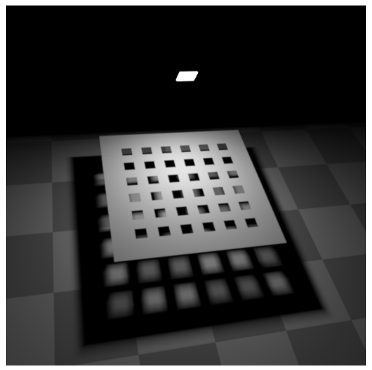
Ray Tracing in Games with NVIDIA RTX,
Llamas and Liu, GDC 2018

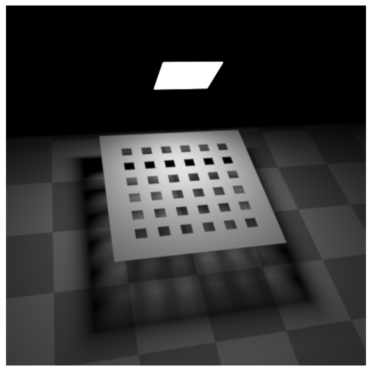
Similarly, NVIDIA showed a demo of soft shadows for a sphere light, doing essentially the same thing.

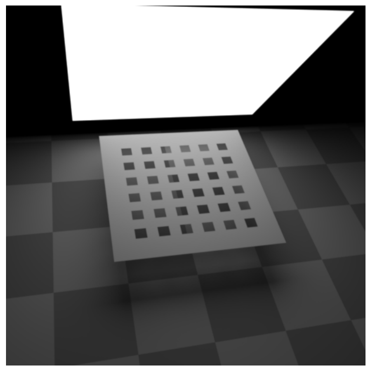


“JUST RAY TRACE SOFT SHADOWS!”

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018



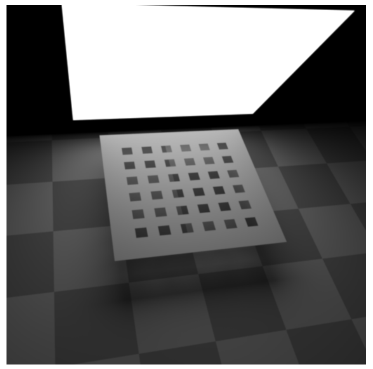


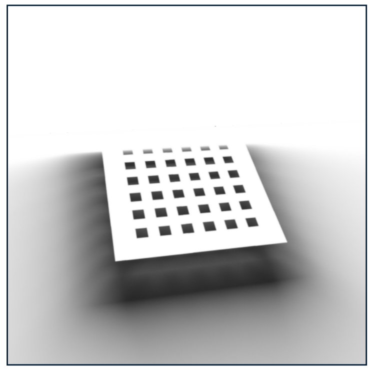


A soft shadow is the average visibility of an area light

So this gives us a definition of what “soft shadows” means: a soft shadow is the average visibility of an area light.

“JUST RAY TRACE SOFT SHADOWS!”





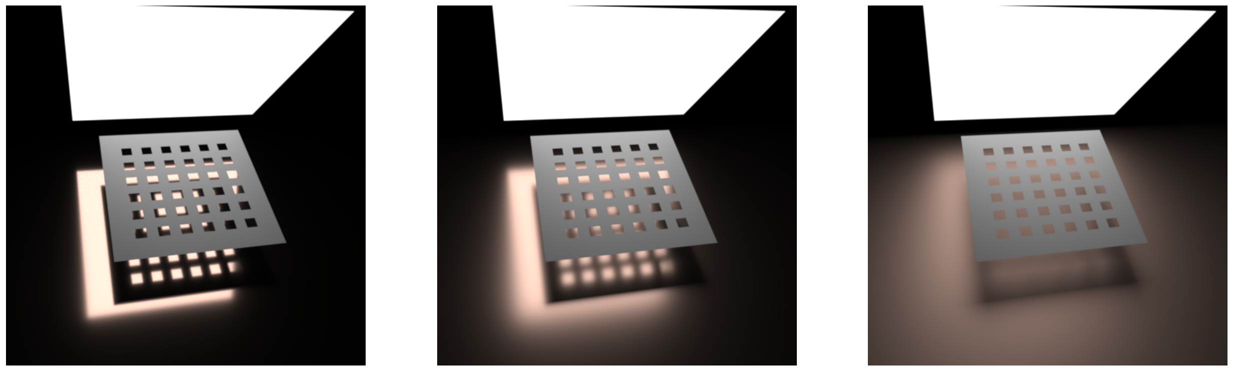
$$\text{soft shadow} = \int_{\text{Light}} \text{Visibility} \,/ \int_{\text{Light}} 1$$

This is defined mathematically by this equation.

This is a purely geometric quantity. Therefore, it could be baked if the light and receiver are static.

“JUST RAY TRACE SOFT SHADOWS!”

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

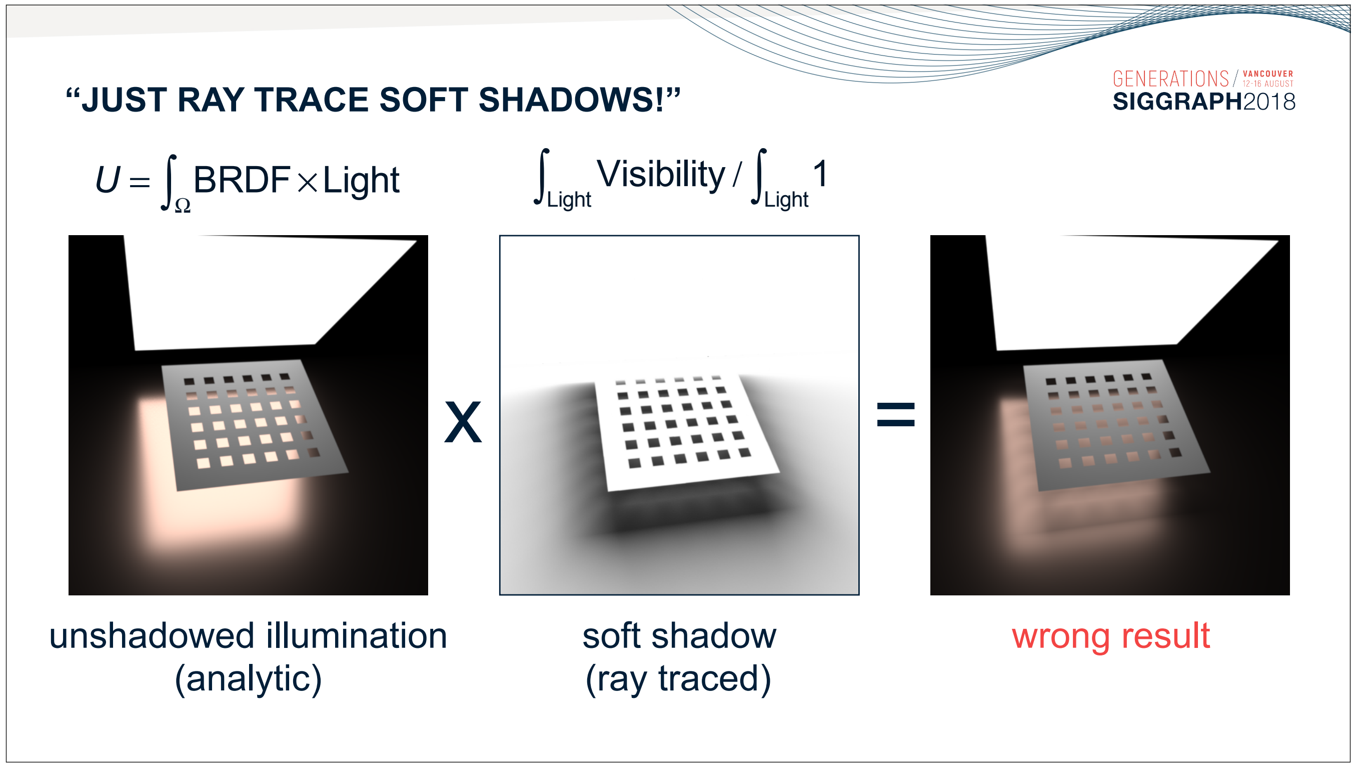


What about the BRDF?

But there’s a problem: we haven’t thought about other surface properties.

For instance, what if the receiver is glossy? Here we can see that the softness of the shadow changes with the BRDF.

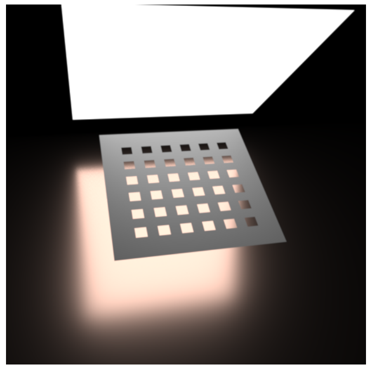
This is easy to understand if we think about a perfect mirror. You will never see a soft shadow on a mirror because light is reflected in a single direction (for a given point), resulting in a binary result, i.e. hard shadows.



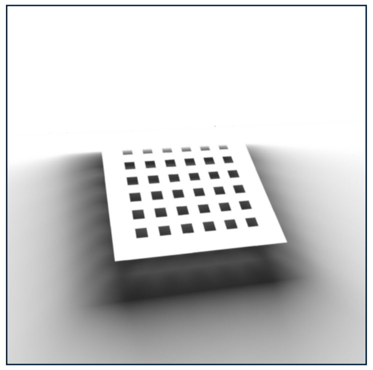
"JUST RAY TRACE SOFT SHADOWS!"

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

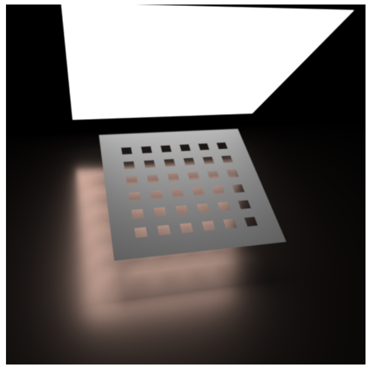
$U = \int_{\Omega} \text{BRDF} \times \text{Light}$ $\quad \int_{\text{Light}} \text{Visibility} / \int_{\text{Light}} 1$



X



=



unshadowed illumination (analytic)

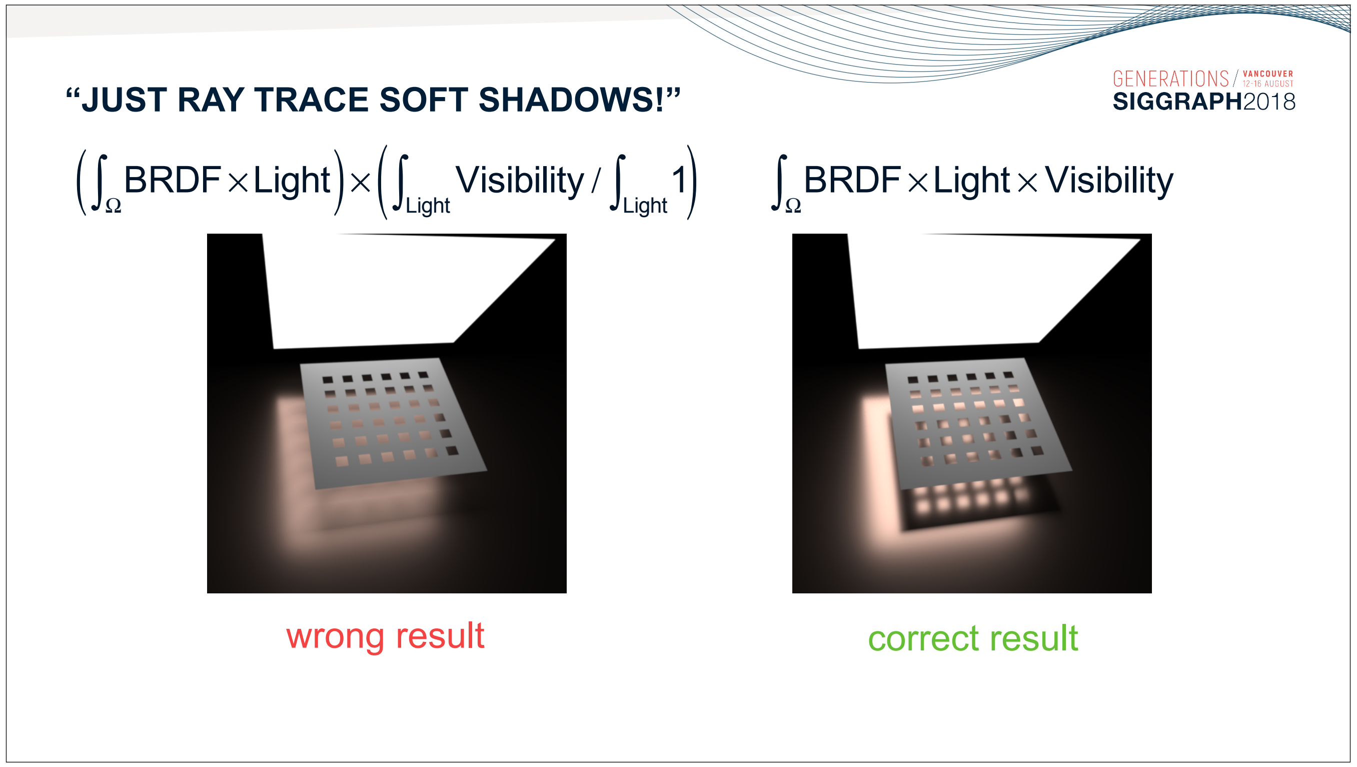
soft shadow (ray traced)

wrong result

This is a real-time attempt at reproducing the offline result in the middle of the previous slide.

Here we're using our real-time area-lighting framework to compute *U*, and a real-time soft-shadow algorithm (e.g. GPU ray tracing) to modulate it.

The result looks obviously wrong.

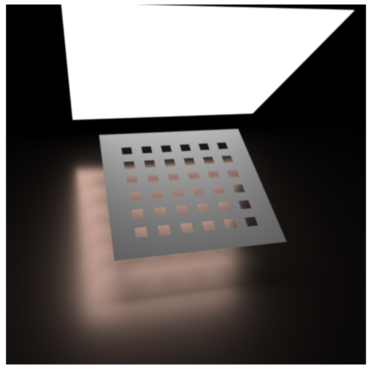


“JUST RAY TRACE SOFT SHADOWS!”

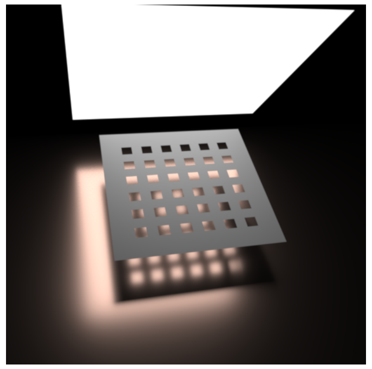
GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

$\left(\int_{\Omega} \text{BRDF} \times \text{Light}\right) \times \left(\int_{\text{Light}} \text{Visibility} \,/ \int_{\text{Light}} 1\right)$

$\int_{\Omega} \text{BRDF} \times \text{Light} \times \text{Visibility}$



wrong result

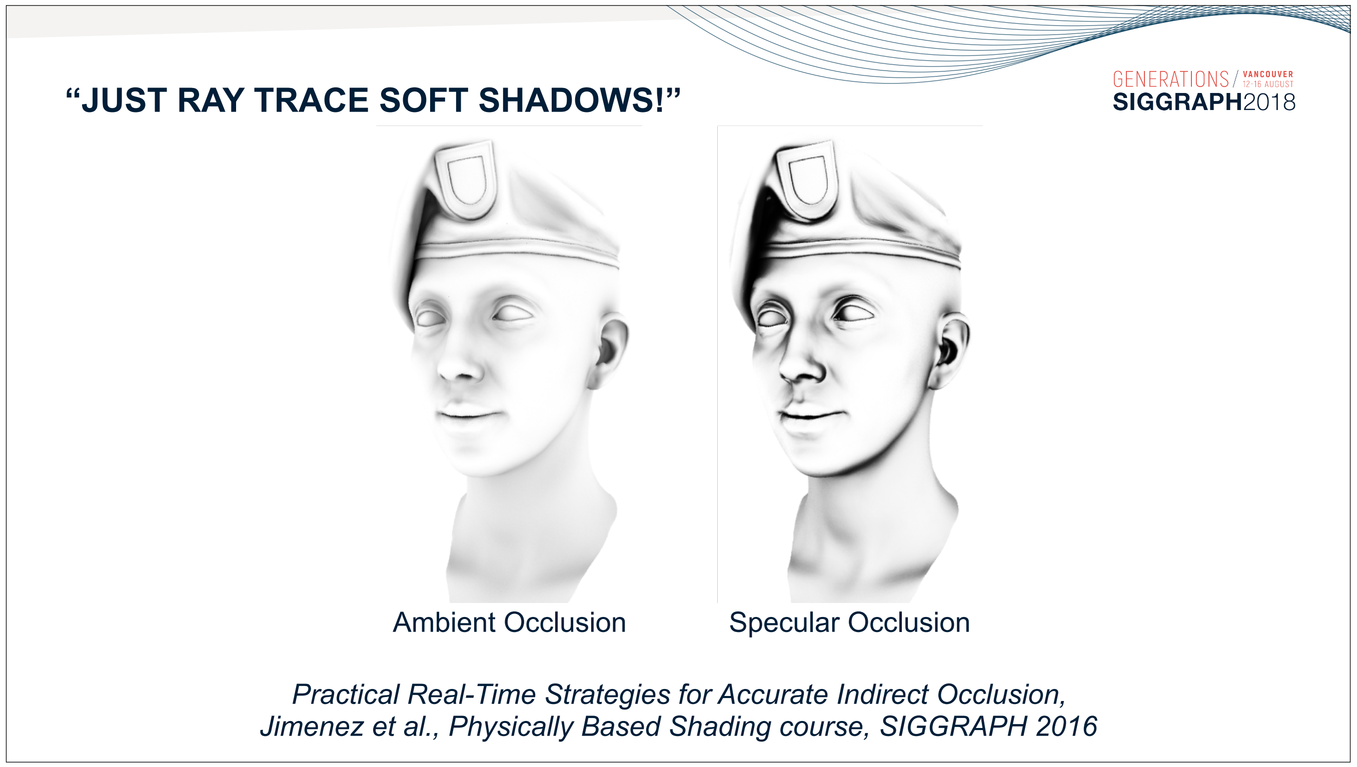


correct result

The reason is that we’re not computing the right equation.

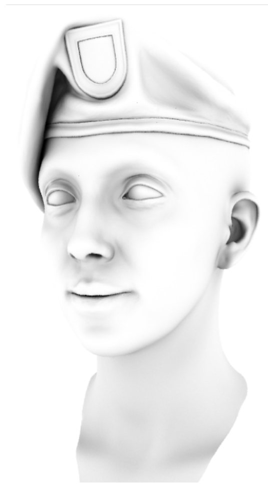
This shows that “shooting rays towards the light source” is not the right approach. It may work okay in some cases, but in others it’s obviously wrong.

With this in mind, it’s sad to think of GPU ray tracing being ‘wasted’ on computing the wrong result.

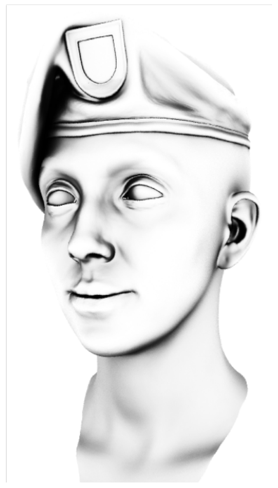


“JUST RAY TRACE SOFT SHADOWS!”

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018



Ambient Occlusion



Specular Occlusion

Practical Real-Time Strategies for Accurate Indirect Occlusion,
Jimenez et al., Physically Based Shading course, SIGGRAPH 2016

You might be thinking that “well, in real-time, we don’t care about correctness”.

However, it’s important that we have an approach that takes the material into account, for *consistency*. This is one of the big payoffs of physically based shading, even if we’re not aiming for photo realism: consistent results = less artist frustration.

Ambient Occlusion vs. Specular Occlusion is an analogous situation, just without a local light source (or an all-encompassing spherical one). In this case we already know that we should be taking the material into account, so we should do the same for area shadows!

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

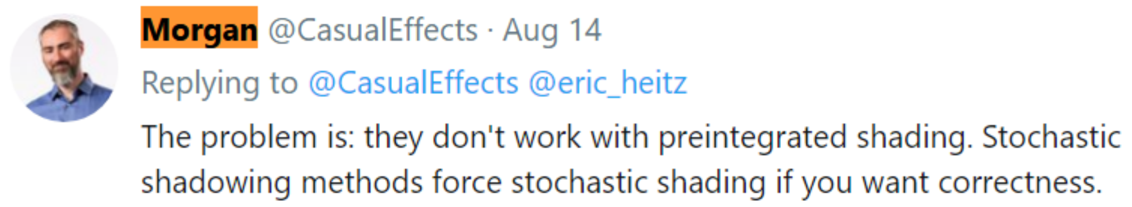
OPTION 2: “GO FULLY STOCHASTIC!”

This leads us to the next option to consider for our problem: use a Monte Carlo estimator of the direct illumination. This is what the offline guys do, and we’re guaranteed to be correct.

This option seems extreme, but let me explain why we might have to do this.

“GO FULLY STOCHASTIC!”

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

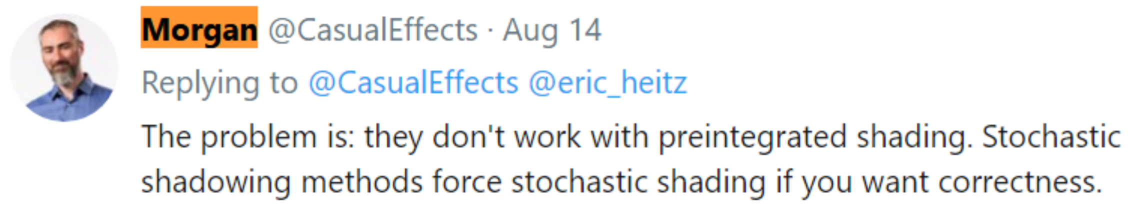

Morgan @CasualEffects · Aug 14

Replying to @CasualEffects @eric_heitz

The problem is: they don't work with preintegrated shading. Stochastic shadowing methods force stochastic shading if you want correctness.

Here’s another tweet from Morgan.

“GO FULLY STOCHASTIC!”



Morgan @CasualEffects · Aug 14

Replying to @CasualEffects @eric_heitz

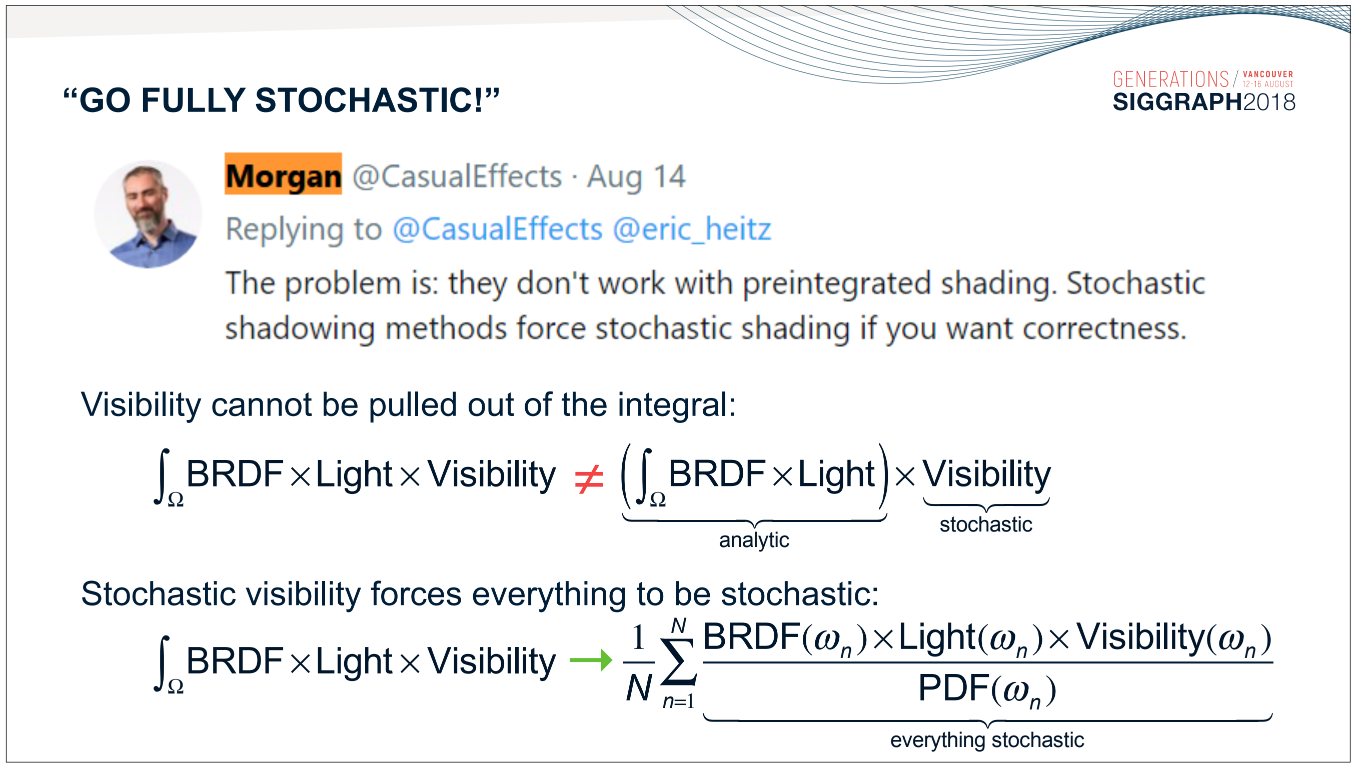
The problem is: they don't work with preintegrated shading. Stochastic shadowing methods force stochastic shading if you want correctness.

Visibility cannot be pulled out of the integral:

$$\int_\Omega \text{BRDF} \times \text{Light} \times \text{Visibility} \neq \underbrace{\left(\int_\Omega \text{BRDF} \times \text{Light}\right)}_{\text{analytic}} \times \underbrace{\text{Visibility}}_{\text{stochastic}}$$

He’s pointing out a fundamental limitation of our area-lighting framework: if we want to compute the right result, visibility needs to be inside the integral.

(With soft shadows, we just saw that computing a separate visibility factor doesn’t work.)



"GO FULLY STOCHASTIC!"

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

Morgan @CasualEffects · Aug 14
Replying to @CasualEffects @eric_heitz
The problem is: they don't work with preintegrated shading. Stochastic shadowing methods force stochastic shading if you want correctness.

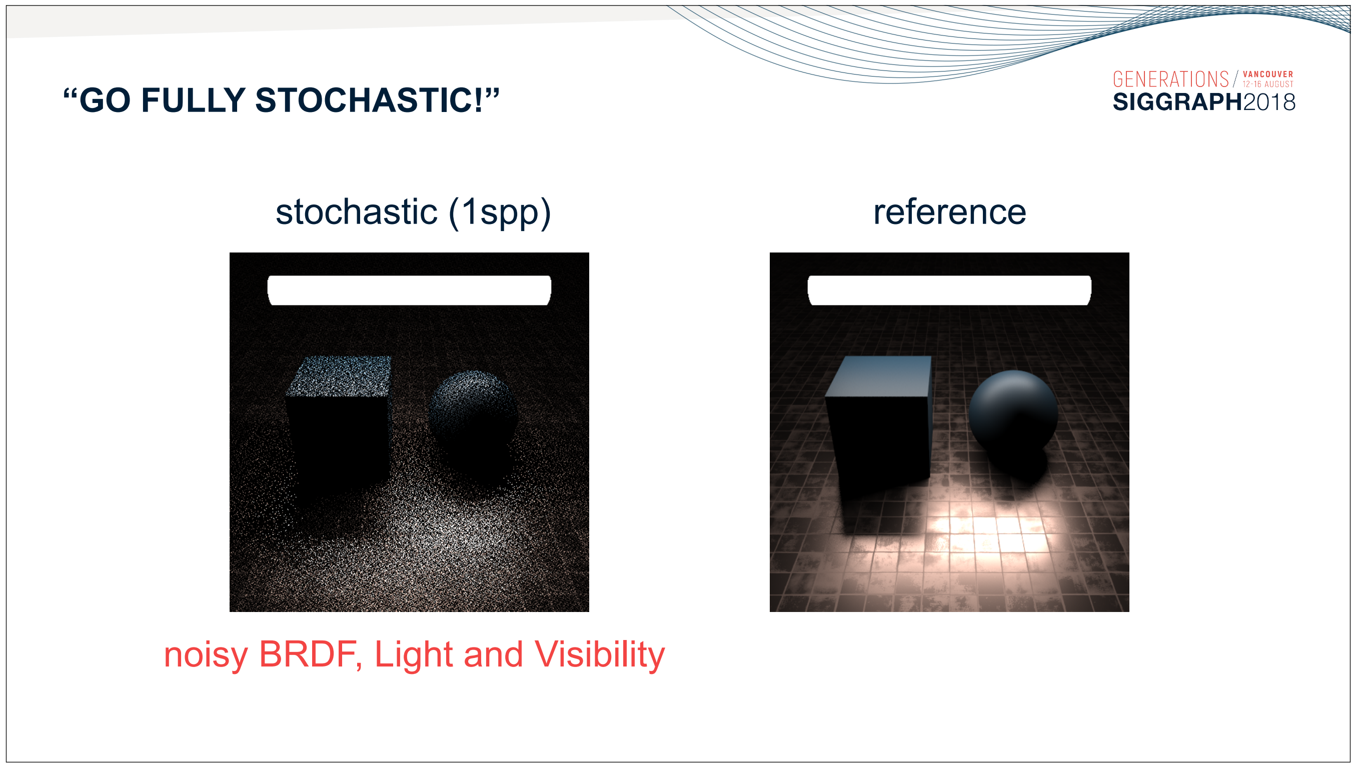
Visibility cannot be pulled out of the integral:

$$\int_\Omega \text{BRDF} \times \text{Light} \times \text{Visibility} \neq \underbrace{\left(\int_\Omega \text{BRDF} \times \text{Light}\right)}_{\text{analytic}} \times \underbrace{\text{Visibility}}_{\text{stochastic}}$$

Stochastic visibility forces everything to be stochastic:

$$\int_\Omega \text{BRDF} \times \text{Light} \times \text{Visibility} \rightarrow \frac{1}{N}\sum_{n=1}^{N} \underbrace{\frac{\text{BRDF}(\omega_n) \times \text{Light}(\omega_n) \times \text{Visibility}(\omega_n)}{\text{PDF}(\omega_n)}}_{\text{everything stochastic}}$$

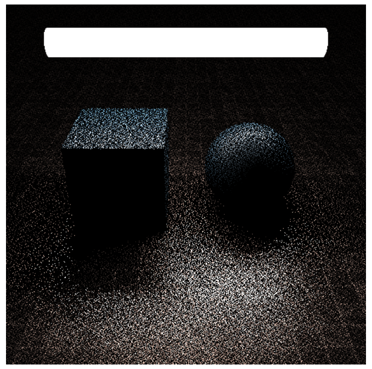
But if we do this, it forces the other terms to be stochastic too.



"GO FULLY STOCHASTIC!"

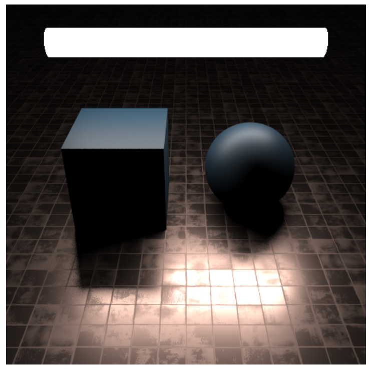
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

stochastic (1spp)



noisy BRDF, Light and Visibility

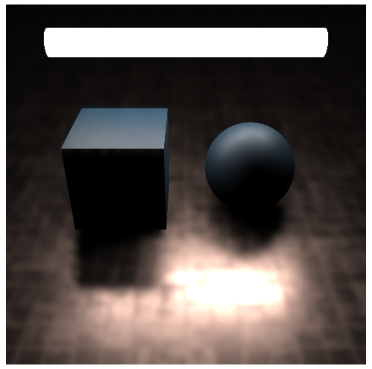
reference



Unfortunately, for real-time, we can't yet afford many samples — perhaps just a few per pixel. This means we will have to deal with lots of noise.

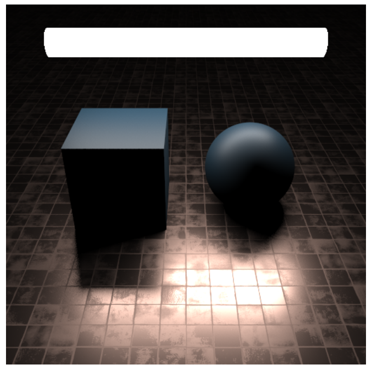
“GO FULLY STOCHASTIC!”

stochastic (1spp) + denoising



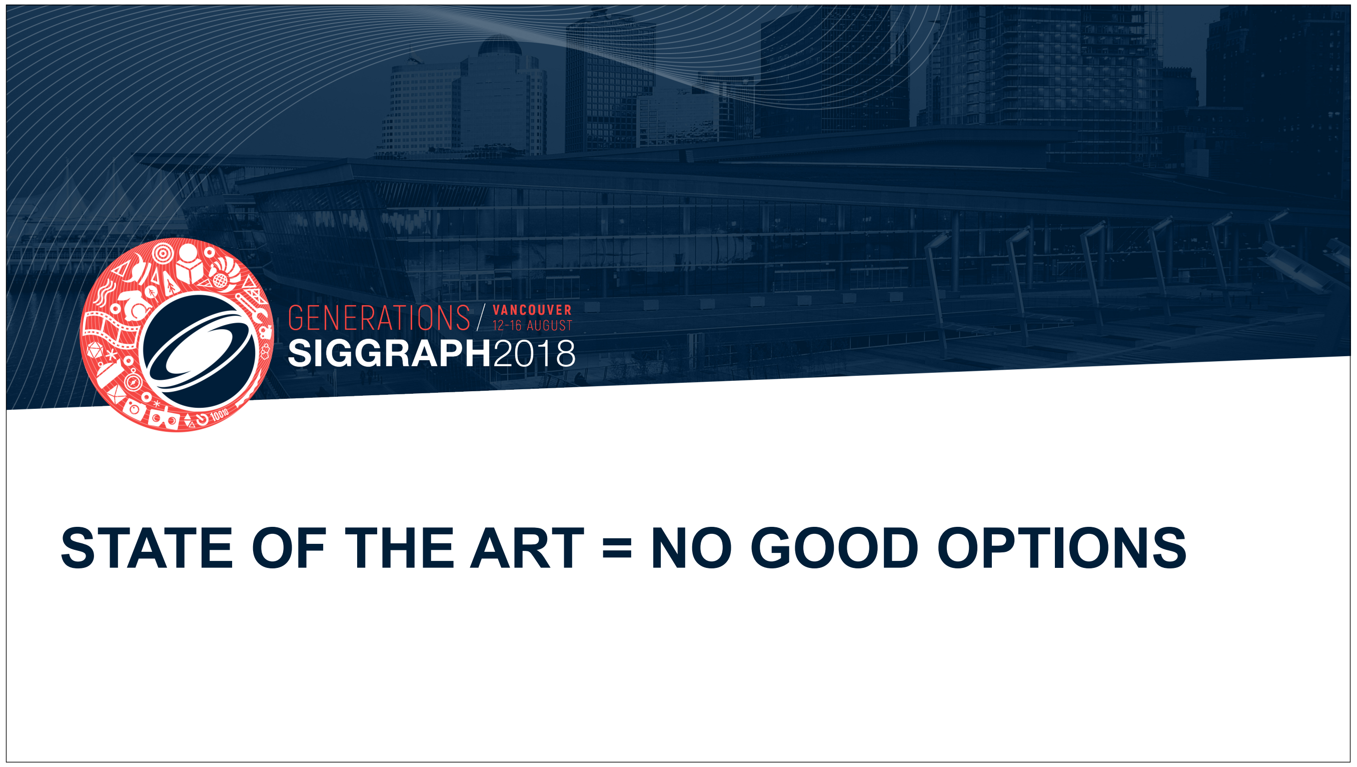
overblurring

reference



The usual solution is to use a denoiser, but it needs to be <u>very</u> aggressive to get rid of the noise. As a result, we’ll get some blurring.

Since our BRDF is part of the integral, this means that we’ll blur the shading too. High-frequency details (e.g. from normal maps) could get lost as a result.


GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

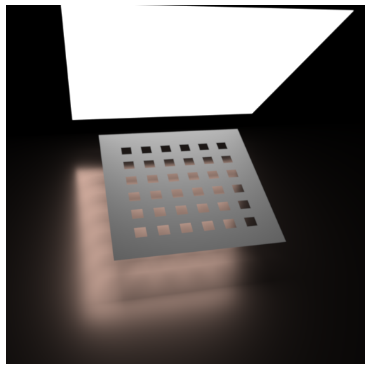
STATE OF THE ART = NO GOOD OPTIONS

So in summary, it seems that we’re screwed.

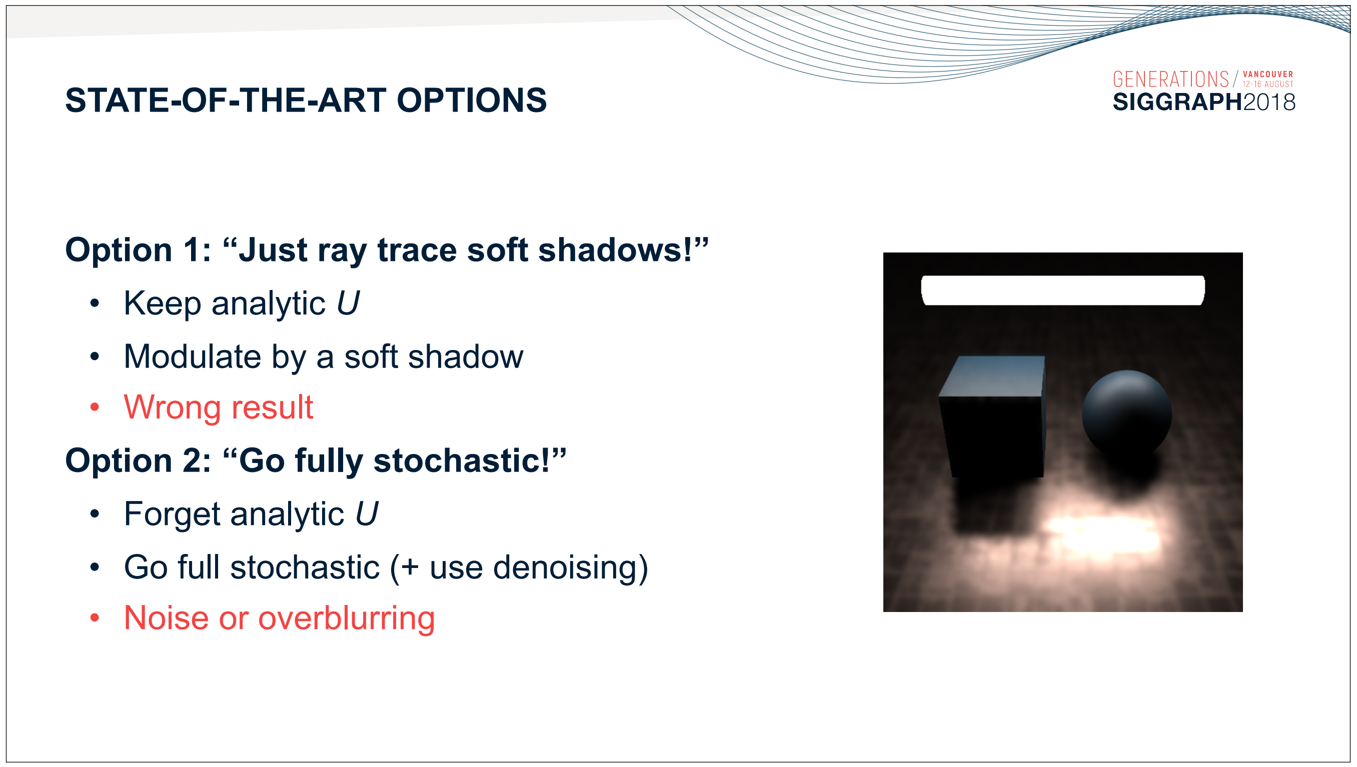
STATE-OF-THE-ART OPTIONS

Option 1: “Just ray trace soft shadows!”

- Keep analytic *U*
- Modulate by a soft shadow
- Wrong result



If we use some sort of soft-shadow algorithm, it will work with our analytical area lighting but we’ll get the wrong result.



STATE-OF-THE-ART OPTIONS

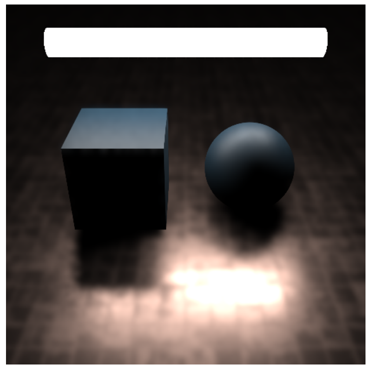
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

Option 1: "Just ray trace soft shadows!"

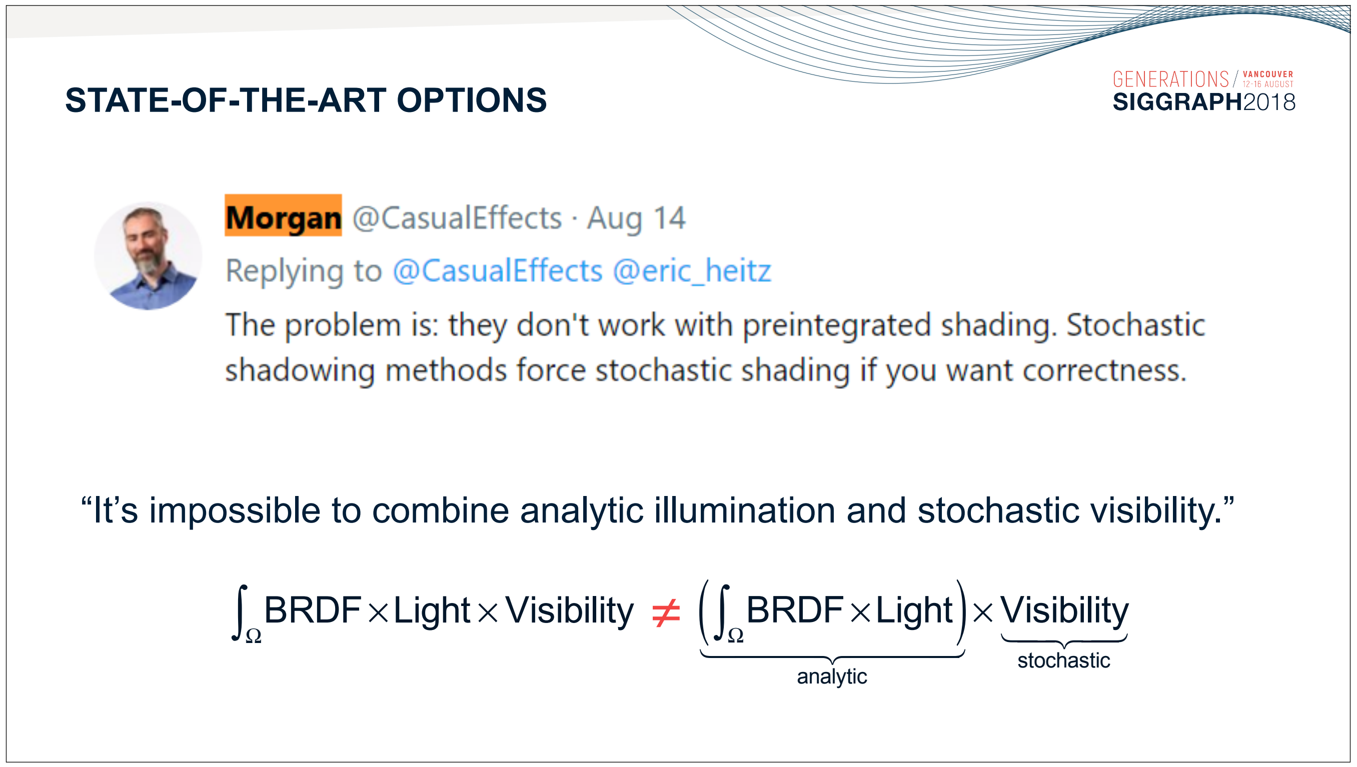
- Keep analytic *U*
- Modulate by a soft shadow
- Wrong result

Option 2: "Go fully stochastic!"

- Forget analytic *U*
- Go full stochastic (+ use denoising)
- Noise or overblurring



Or we can go 'fully stochastic', which means giving up on our analytic method and also having to contend with noise or overblurring.



STATE-OF-THE-ART OPTIONS

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

Morgan @CasualEffects · Aug 14
Replying to @CasualEffects @eric_heitz
The problem is: they don't work with preintegrated shading. Stochastic shadowing methods force stochastic shading if you want correctness.

“It’s impossible to combine analytic illumination and stochastic visibility.”

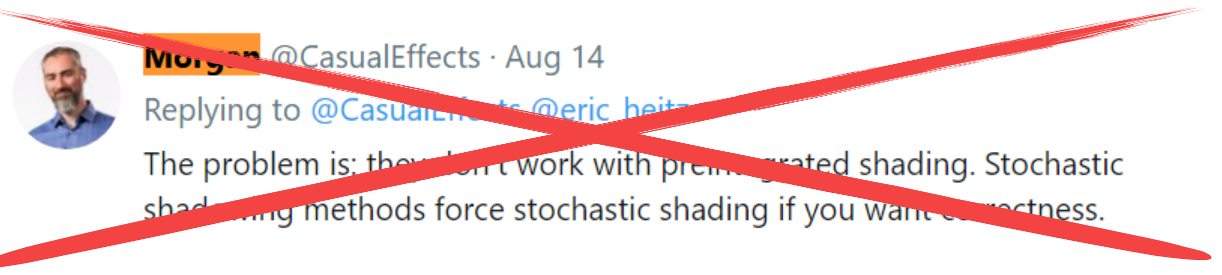
$$\int_\Omega \text{BRDF}\times\text{Light}\times\text{Visibility} \neq \underbrace{\left(\int_\Omega \text{BRDF}\times\text{Light}\right)}_{\text{analytic}}\times\underbrace{\text{Visibility}}_{\text{stochastic}}$$

The dilemma is due to what Morgan pointed out here.

Because we cannot pull visibility out of the integral, we are doomed to either do it approximately or make everything stochastic.

STATE-OF-THE-ART OPTIONS

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018



~~Morgan @CasualEffects · Aug 14~~
~~Replying to @CasualEffects @eric_heitz~~
~~The problem is: they don't work with preintegrated shading. Stochastic shadowing methods force stochastic shading if you want correctness.~~

We found a way to do it right

$$\int_\Omega \text{BRDF}\times\text{Light}\times\text{Visibility} = \underbrace{\left(\int_\Omega \text{BRDF}\times\text{Light}\right)}_{\text{analytic}}\times\underbrace{\text{Visibility}}_{\text{stochastic}}$$

Our breakthrough was when we realised that this was actually a misconception.

We found a way to pull visibility out of the integral without breaking the equation. Thanks to this, we don't have to compromise and we obtain something much better instead.

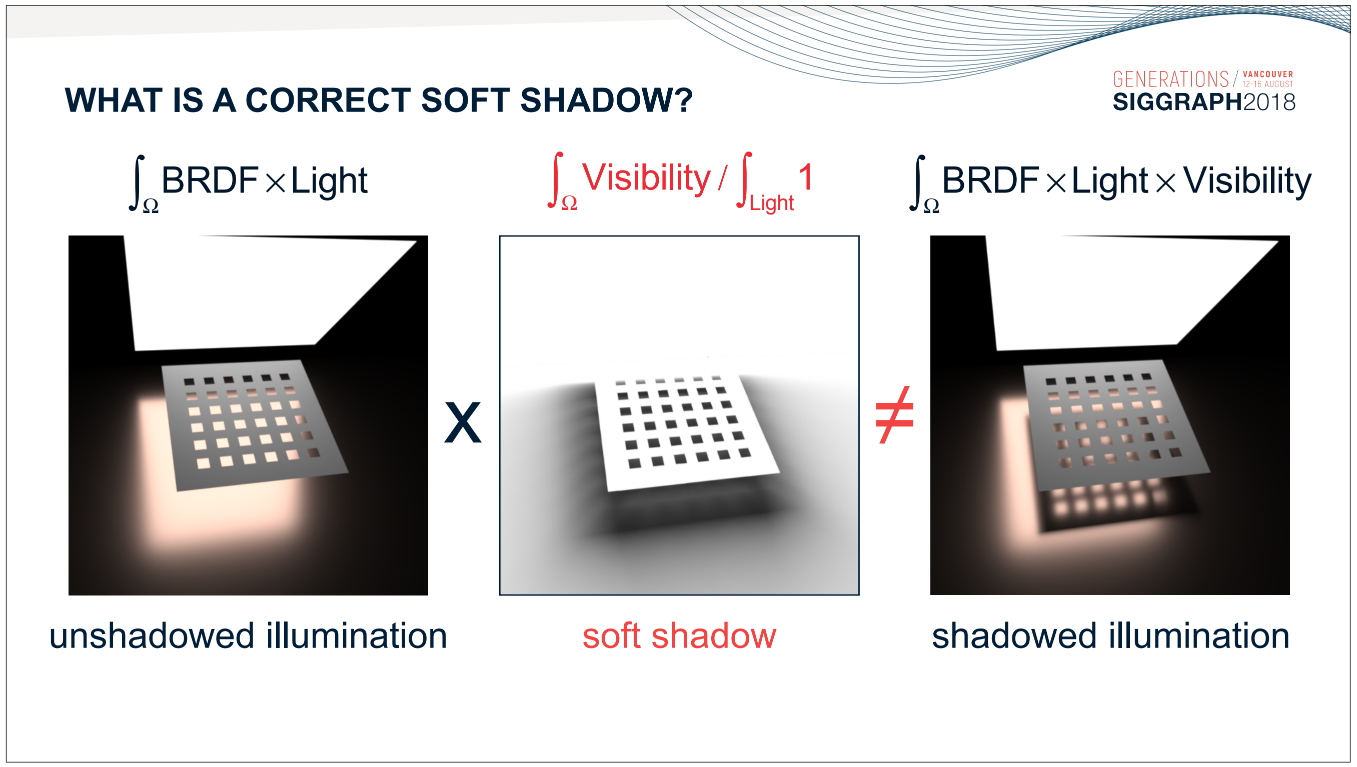
So, the cornerstone of our method is to solve this equation…

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

WHAT IS A CORRECT SOFT SHADOW?

…which is equivalent to asking: “What is a correct soft shadow?"

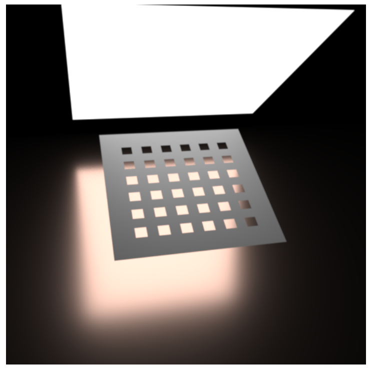
This is the question that I was teasing you with in the beginning.



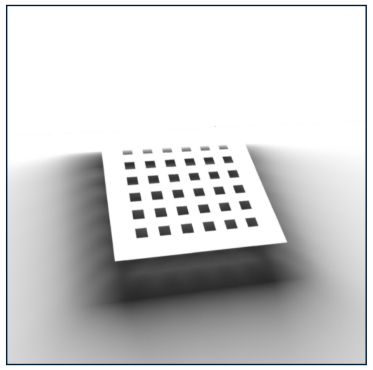
GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

WHAT IS A CORRECT SOFT SHADOW?

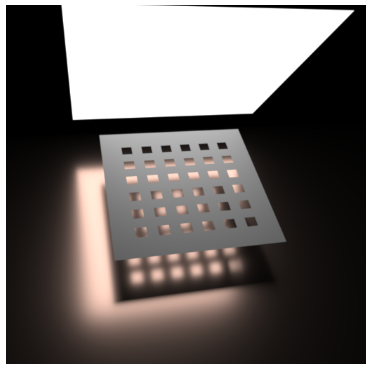
$\int_\Omega \text{BRDF} \times \text{Light}$ | $\int_\Omega \text{Visibility} / \int_{\text{Light}} 1$ | $\int_\Omega \text{BRDF} \times \text{Light} \times \text{Visibility}$


unshadowed illumination

X


soft shadow

≠


shadowed illumination

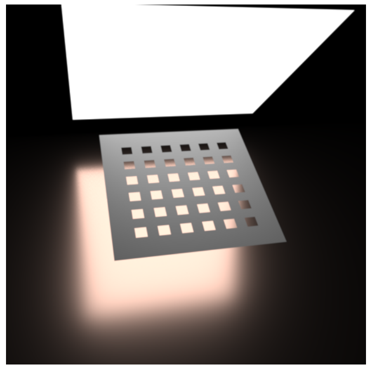
We have seen that soft shadows do not give the right result, so we might wonder...

WHAT IS A CORRECT SOFT SHADOW?

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

$\int_{\Omega} \mathrm{BRDF} \times \mathrm{Light}$

$\int_{\Omega} \mathrm{BRDF} \times \mathrm{Light} \times \mathrm{Visibility}$

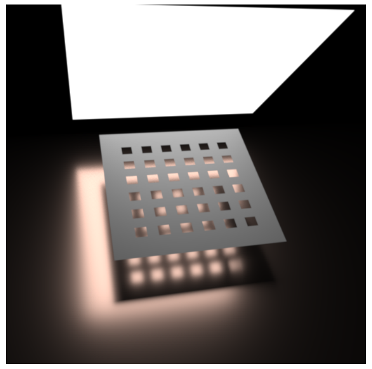

unshadowed illumination

x

?

“correct” soft shadow

=

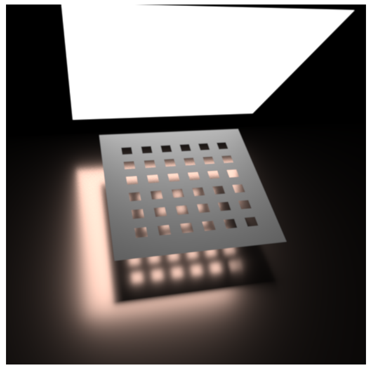
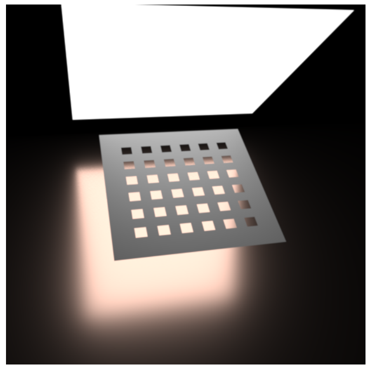

shadowed illumination

...what should we compute instead? What goes here in the middle?

WHAT IS A CORRECT SOFT SHADOW?

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

$$? = \int_\Omega \text{BRDF} \times \text{Light} \times \text{Visibility} \; / \; \int_\Omega \text{BRDF} \times \text{Light}$$

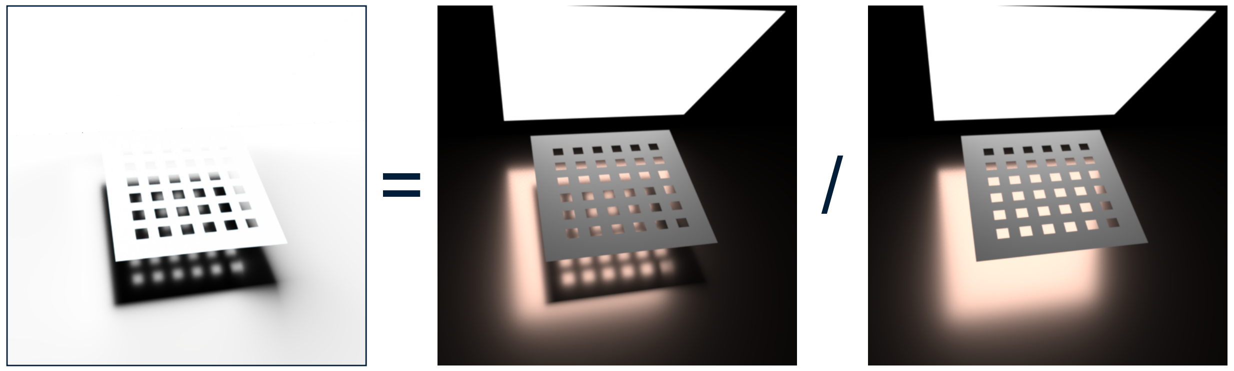
“correct” soft shadow = shadowed illumination / unshadowed illumination

Obvious trick: what if we put the first image on the other side of the equals sign?

We get the shadowed illumination divided by the unshadowed illumination.

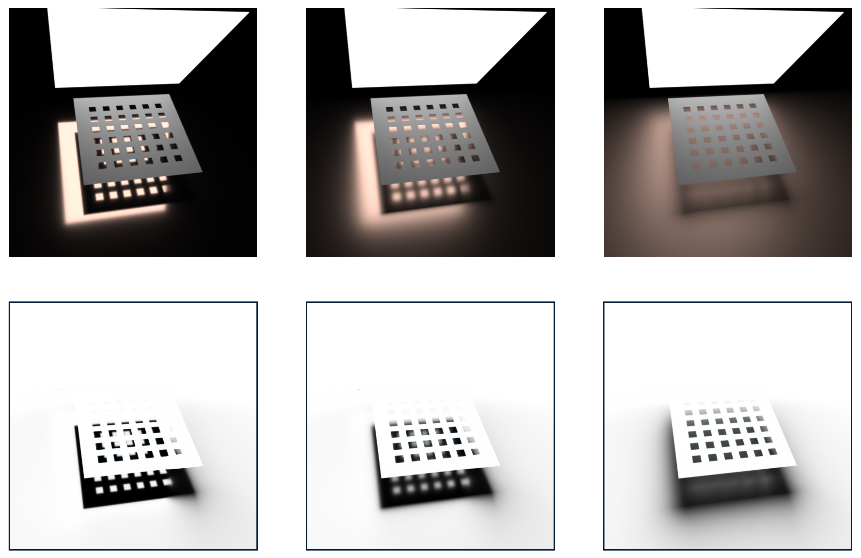
WHAT IS A CORRECT SOFT SHADOW?

$$\text{“correct” soft shadow} = \frac{\int_\Omega \text{BRDF} \times \text{Light} \times \text{Visibility}}{\int_\Omega \text{BRDF} \times \text{Light}}$$

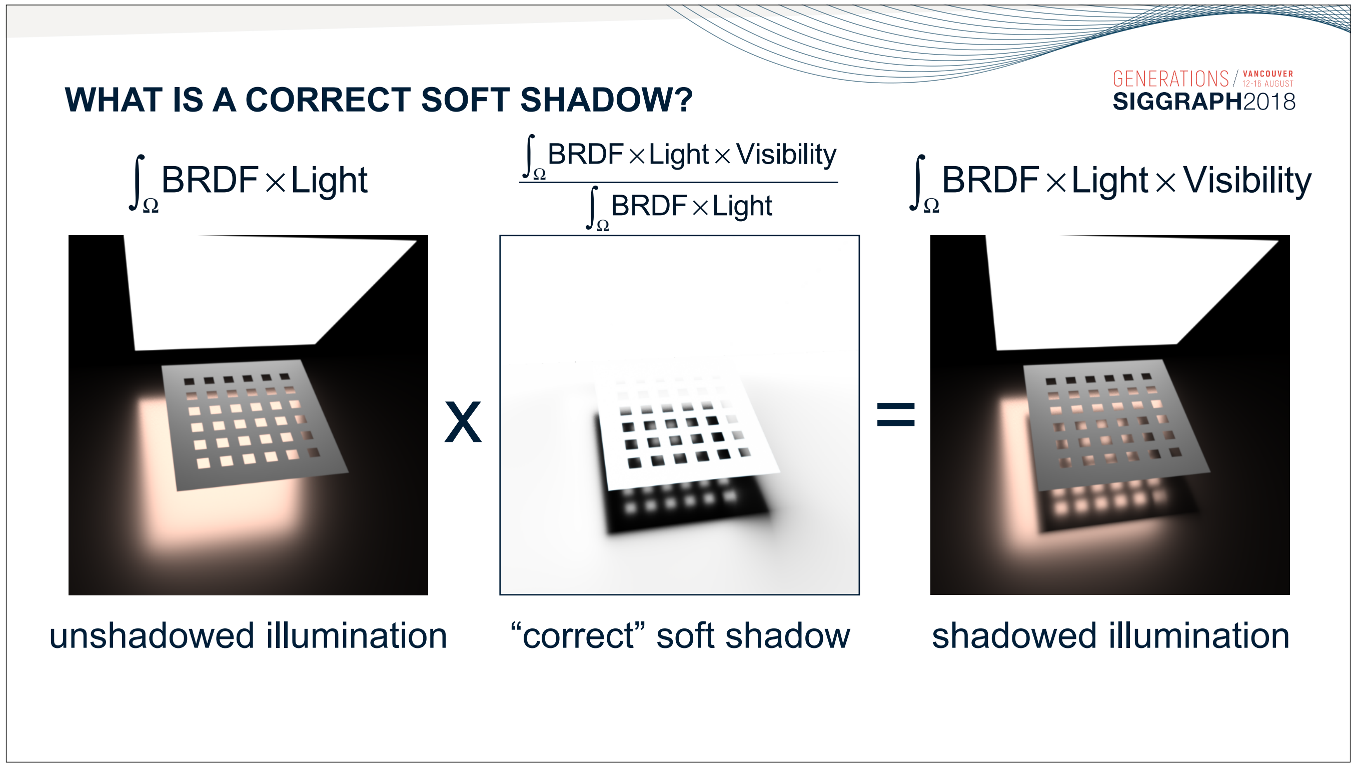

“correct” soft shadow = shadowed illumination / unshadowed illumination

If we compute this, we obtain is an image whose values are always between 0 and 1.

WHAT IS A CORRECT SOFT SHADOW?



With this definition, the *shadow image* has the right softness when the BRDF changes, and it matches the shadow that we effectively see in the reference image.



WHAT IS A CORRECT SOFT SHADOW?

GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

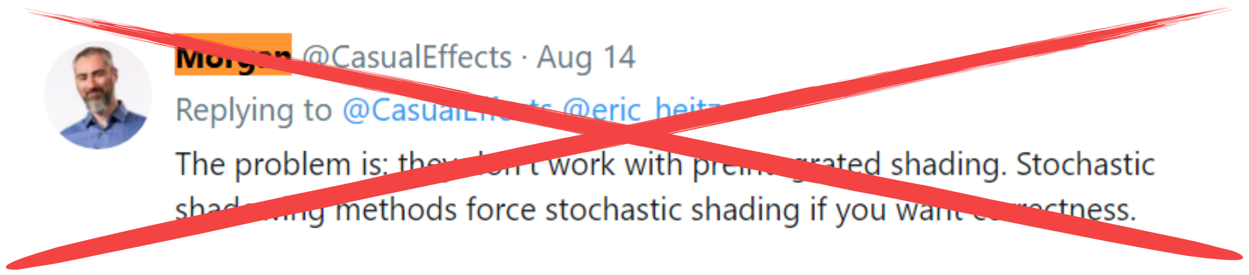
$$\int_\Omega \mathrm{BRDF} \times \mathrm{Light} \quad \times \quad \frac{\int_\Omega \mathrm{BRDF} \times \mathrm{Light} \times \mathrm{Visibility}}{\int_\Omega \mathrm{BRDF} \times \mathrm{Light}} \quad = \quad \int_\Omega \mathrm{BRDF} \times \mathrm{Light} \times \mathrm{Visibility}$$

unshadowed illumination X "correct" soft shadow = shadowed illumination

And, mathematically, we can see that things are obviously correct: a · b/a = b.

WHAT IS A CORRECT SOFT SHADOW?

GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018



~~Morgan @CasualEffects · Aug 14~~

~~Replying to @CasualEffects @eric_heitz~~

~~The problem is: they don't work with preintegrated shading. Stochastic shadowing methods force stochastic shading if you want correctness.~~

We found a way to do it right

$$\int_\Omega \text{BRDF}\times\text{Light}\times\text{Visibility} = \underbrace{\left(\int_\Omega \text{BRDF}\times\text{Light}\right)}_{\text{analytic}}\times\underbrace{\frac{\int_\Omega \text{BRDF}\times\text{Light}\times\text{Visibility}}{\int_\Omega \text{BRDF}\times\text{Light}}}_{\text{stochastic}}$$

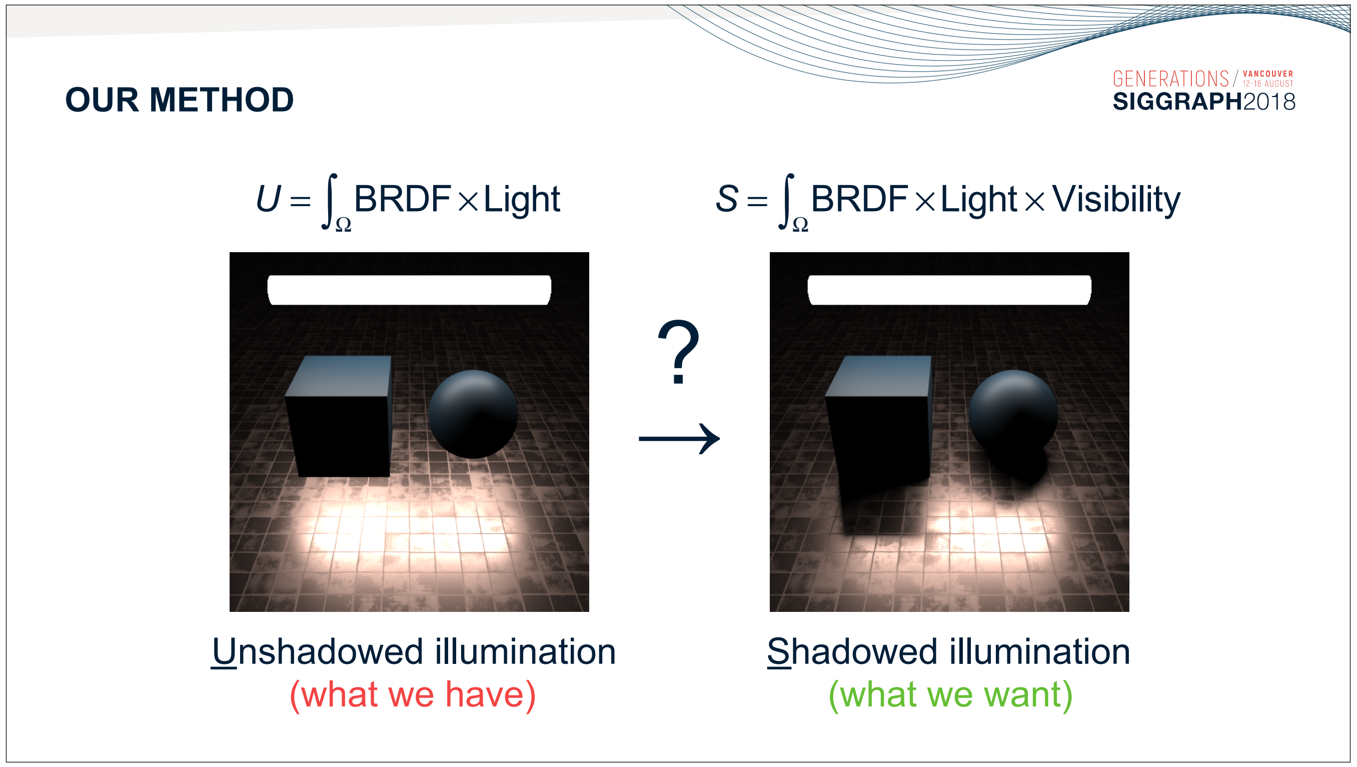
By defining the visibility in this way, we obtain a stochastic visibility term that can be computed outside of the integral.

In summary: we get the right result without making everything stochastic!

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

OUR METHOD

I'll now cover how our method works in practice.

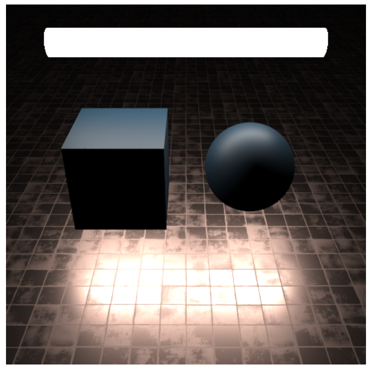


OUR METHOD

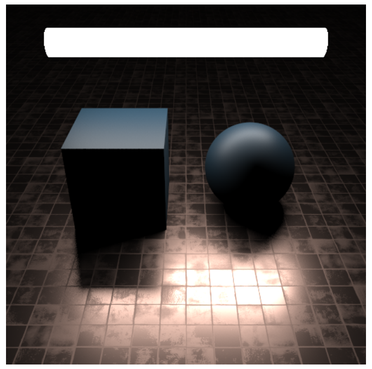
GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

$U = \int_\Omega \text{BRDF} \times \text{Light}$

$S = \int_\Omega \text{BRDF} \times \text{Light} \times \text{Visibility}$



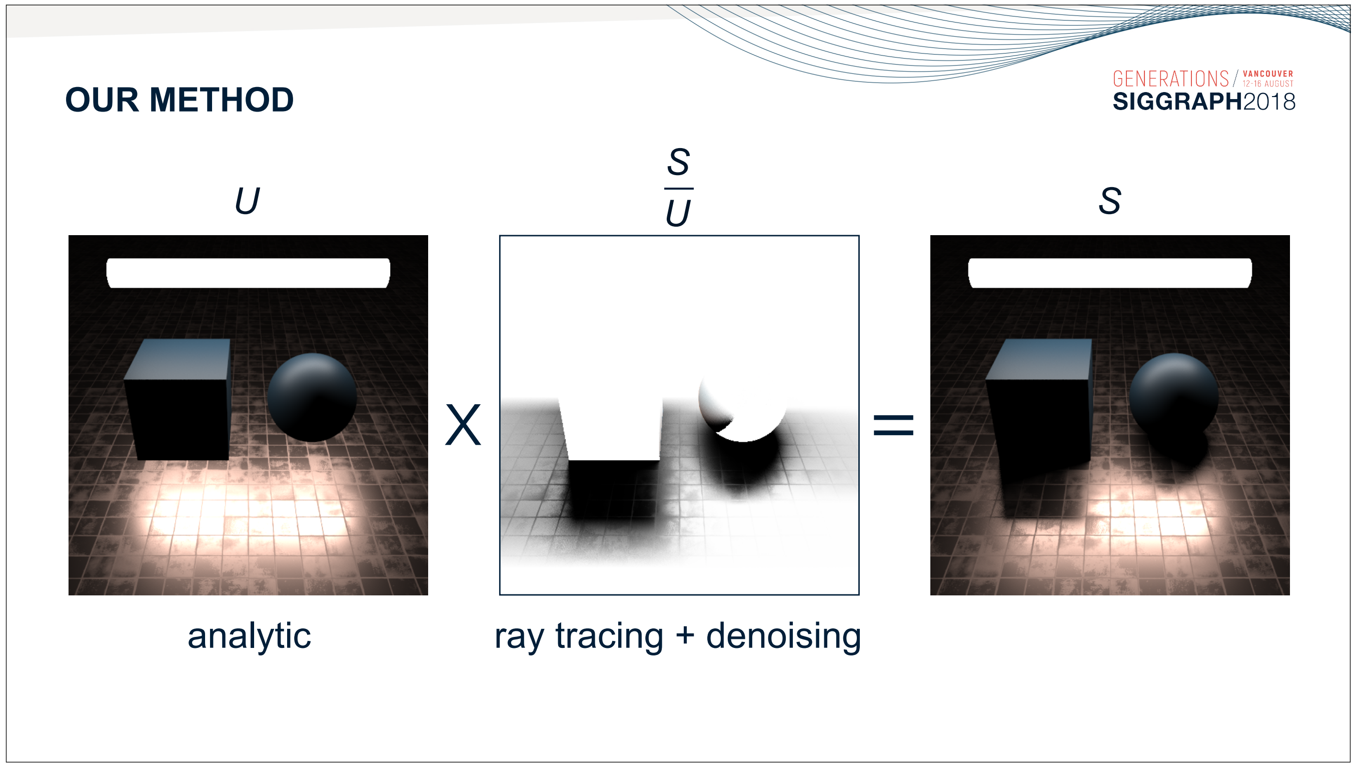
$\overset{?}{\longrightarrow}$



Unshadowed illumination
(what we have)

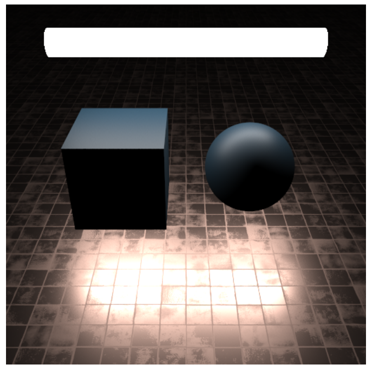
Shadowed illumination
(what we want)

Here's our problem again: we want to obtain *S* given that we have an analytic solution for *U*.

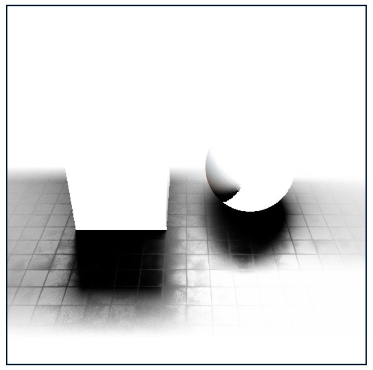


OUR METHOD

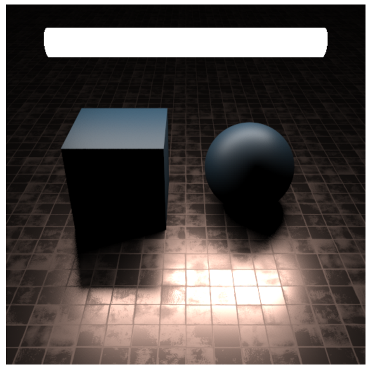
$$U \times \frac{S}{U} = S$$



analytic



ray tracing + denoising



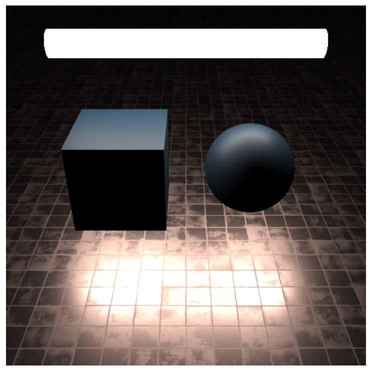
With the formulation that we have seen, we obtain $S = U \cdot S/U$.

By itself, this formulation isn't very useful: we are defining the result we want to compute as a function of the result we want to compute. It's self-referential.

Of course, since we don't know S, we cannot directly compute S/U. The trick is that we use ray tracing and denoising to obtain a good *estimate* of S/U.

OUR METHOD

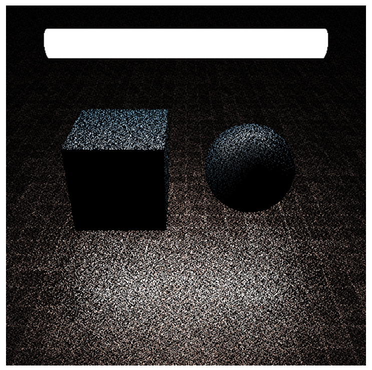
U



$\int_{\Omega} \mathrm{BRDF} \times \mathrm{Light}$

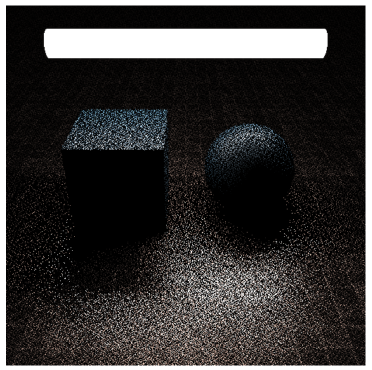
analytic

U_N



$\frac{1}{N}\sum_{n=1}^{N} \frac{\mathrm{BRDF}(\omega_n) \times \mathrm{Light}(\omega_n)}{\mathrm{PDF}(\omega_n)}$

S_N



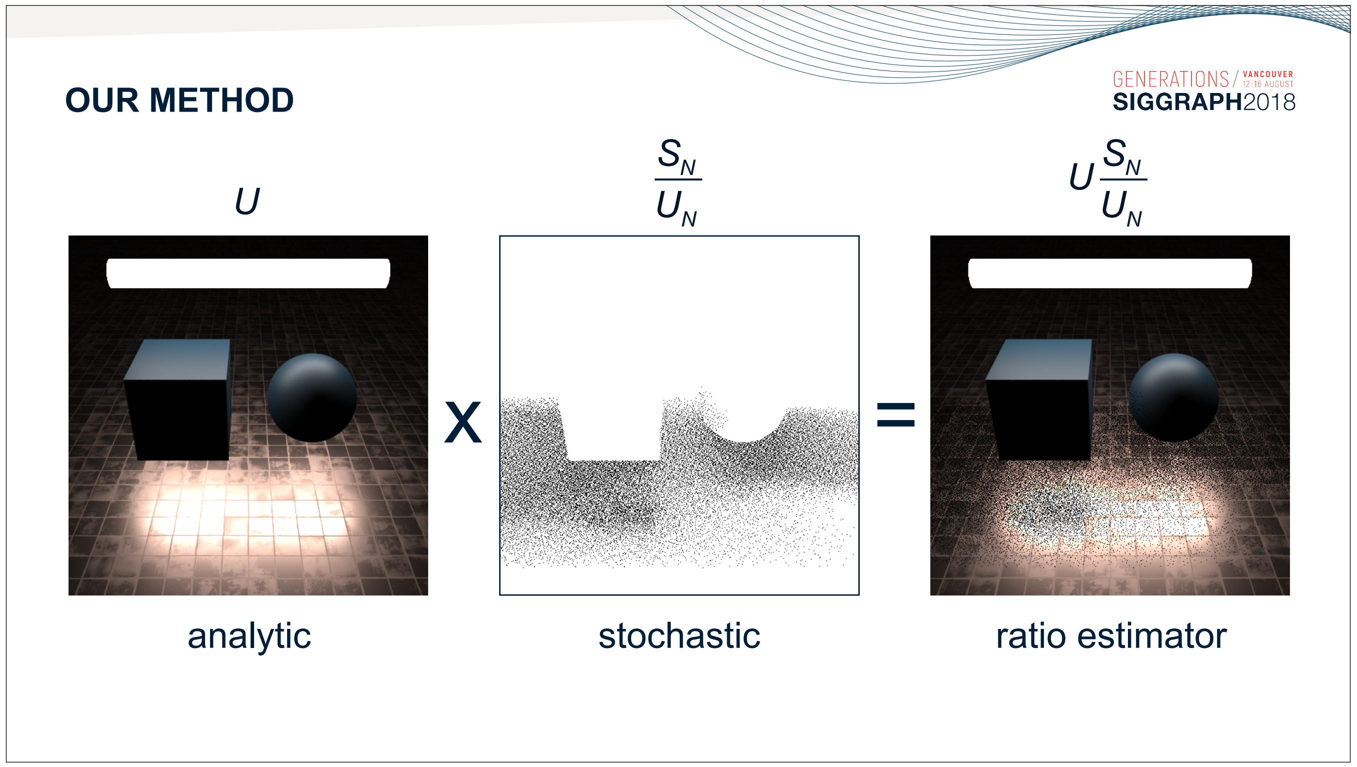
$\frac{1}{N}\sum_{n=1}^{N} \frac{\mathrm{BRDF}(\omega_n) \times \mathrm{Light}(\omega_n) \times \mathrm{Visibility}(\omega_n)}{\mathrm{PDF}(\omega_n)}$

stochastic with the same random numbers

Our algorithm computes three images:
* the analytic U using our area-lighting framework
* a stochastic estimate U_N of U
* and a stochastic estimate S_N of S.

The subscript N stands for the number of samples per pixel.

An important detail is that we use the same random numbers for U_N and S_N. This way, variance is correlated…



OUR METHOD

U $\times$ $\frac{S_N}{U_N}$ $=$ $U\frac{S_N}{U_N}$

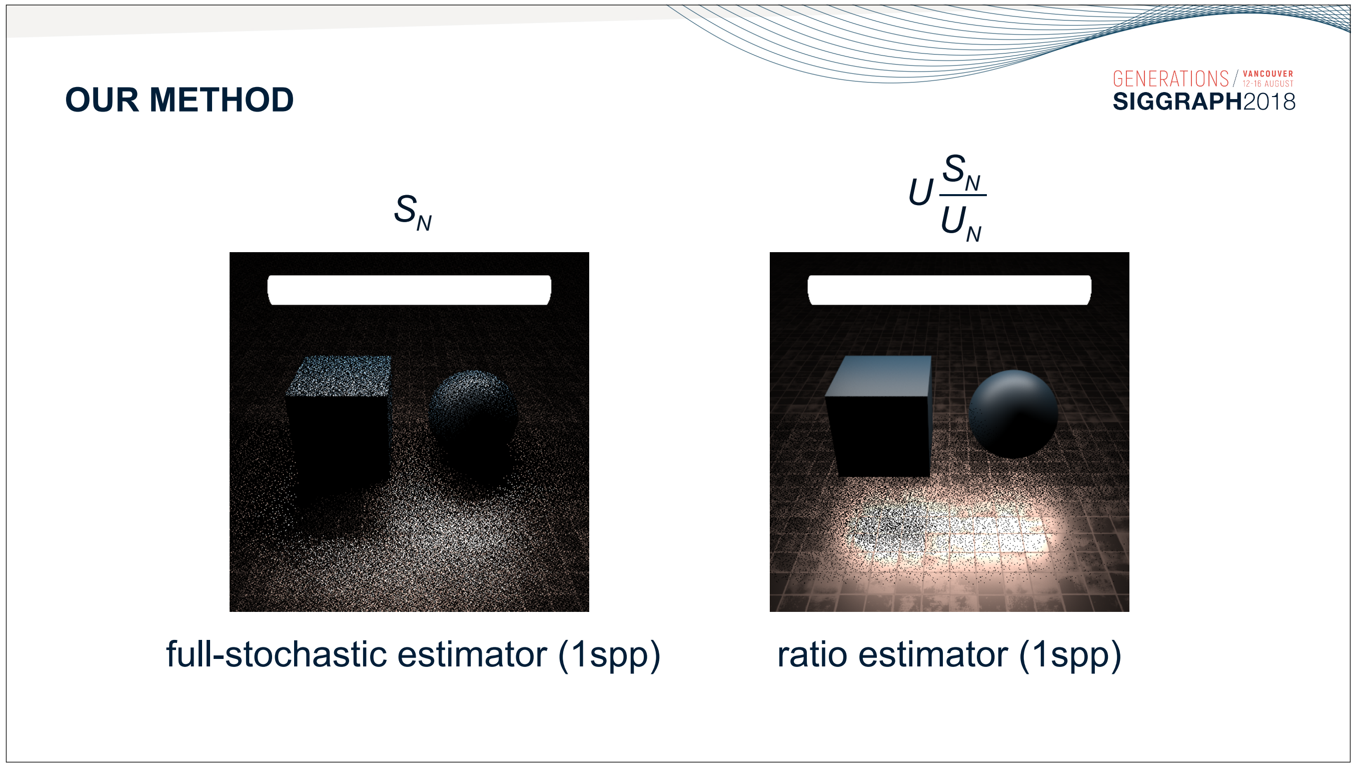
analytic

stochastic

ratio estimator

…and cancels out when we divide S_N by U_N.

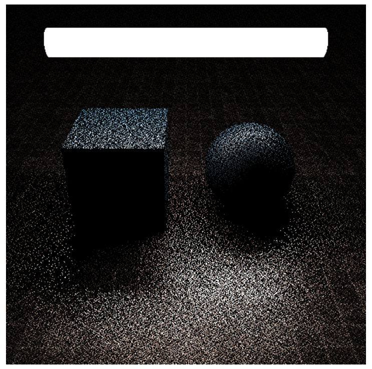
The only remaining noise is in the shadows.



OUR METHOD

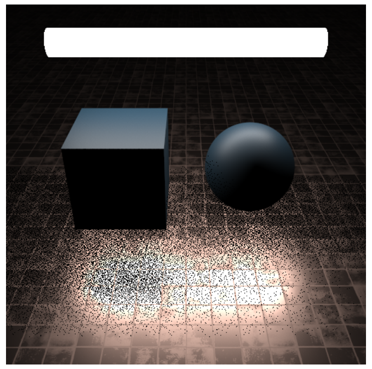
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

S_N



full-stochastic estimator (1spp)

$U\frac{S_N}{U_N}$



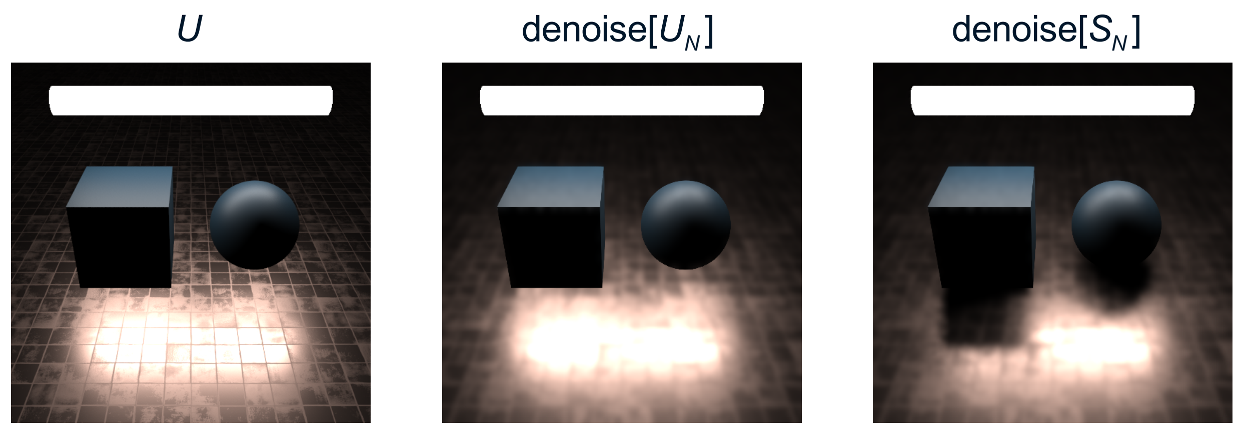
ratio estimator (1spp)

This yields a Monte-Carlo estimator called a *ratio estimator*.

It’s better than the classic estimator because all of the information related to shading is analytic and noise free — only the visibility is stochastic.

OUR METHOD

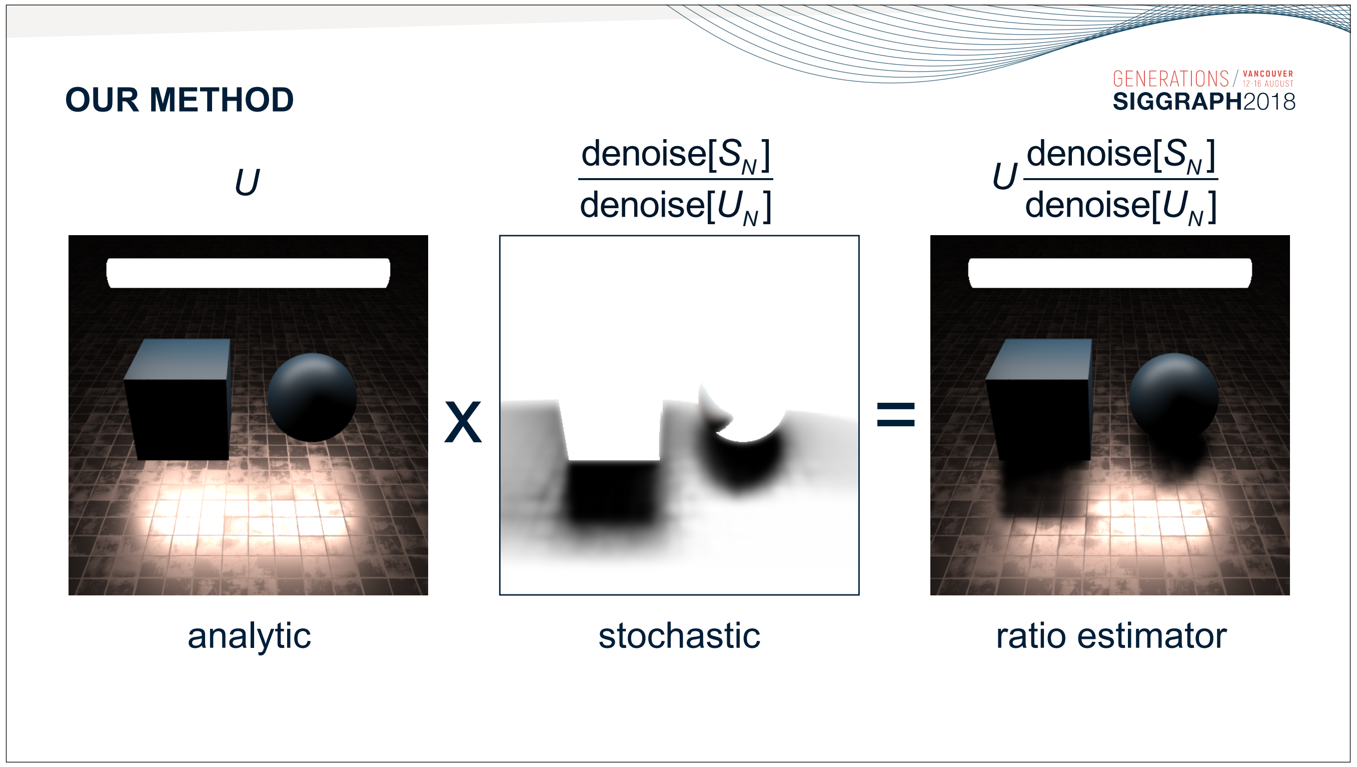
GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018



denoising with the same per-pixel filters

Still, we don’t want noise, so we apply a real-time denoiser to U_N and S_N.

As with the random numbers, we apply the same denoising kernel to both images.



OUR METHOD

U × $\frac{\text{denoise}[S_N]}{\text{denoise}[U_N]}$ = $U\frac{\text{denoise}[S_N]}{\text{denoise}[U_N]}$

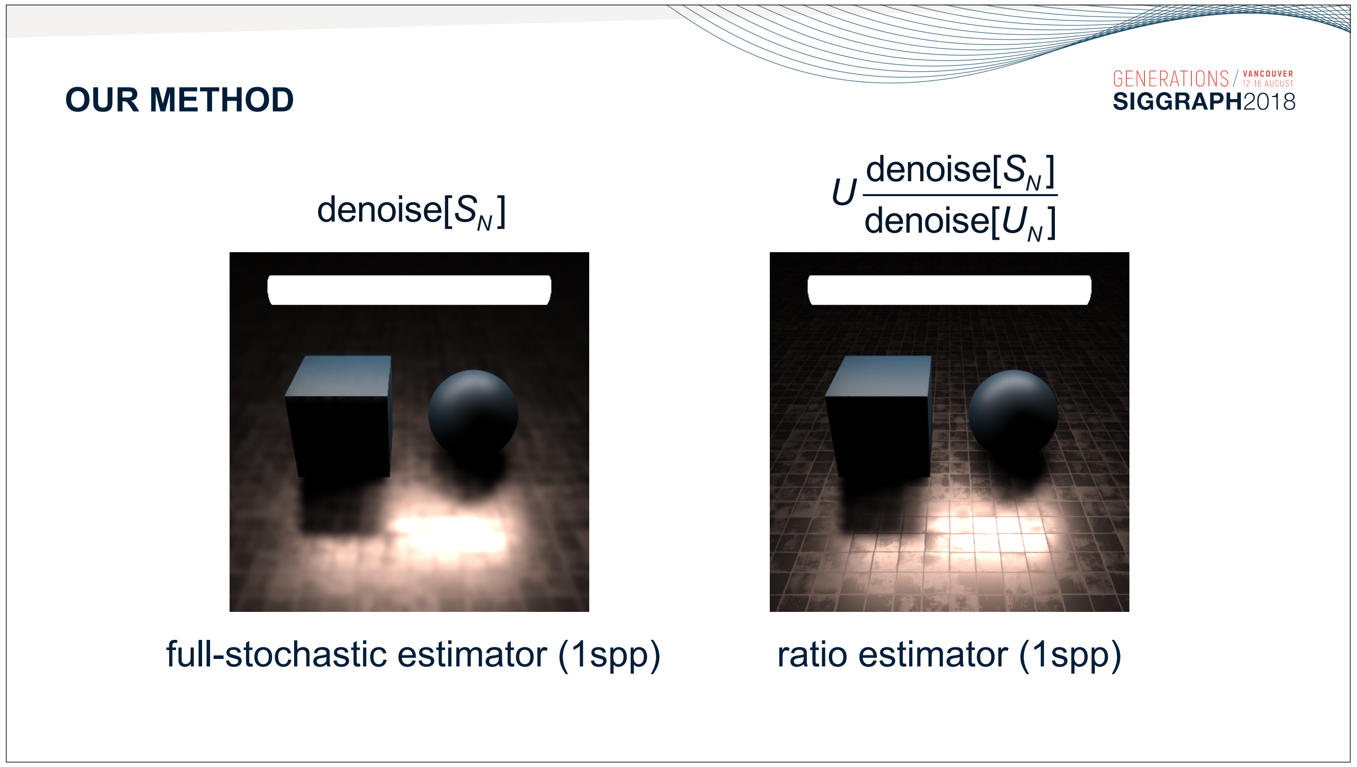
analytic

stochastic

ratio estimator

This gives us a denoised shadow image, which is blurred because of the aggressive denoiser.

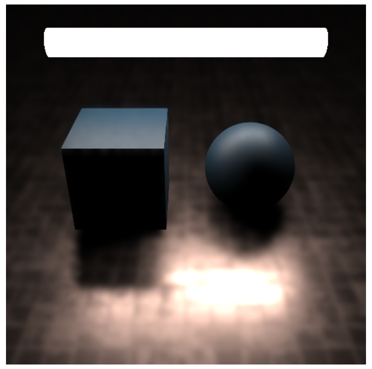
But, importantly, the analytic shading remains sharp.



OUR METHOD

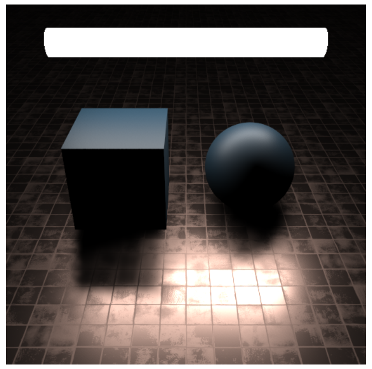
GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

$\text{denoise}[S_N]$



full-stochastic estimator (1spp)

$U\dfrac{\text{denoise}[S_N]}{\text{denoise}[U_N]}$



ratio estimator (1spp)

Here we can see that we get a much better result than if we had gone with option 2, where everything is stochastic.

So, even with a low number of samples and a low-quality denoiser, we still get good-looking results.

OUR METHOD

Stochastic Screen-Space Reflections, Stachowiak,
Advances in Real-Time Rendering in Games course, SIGGRAPH 2015

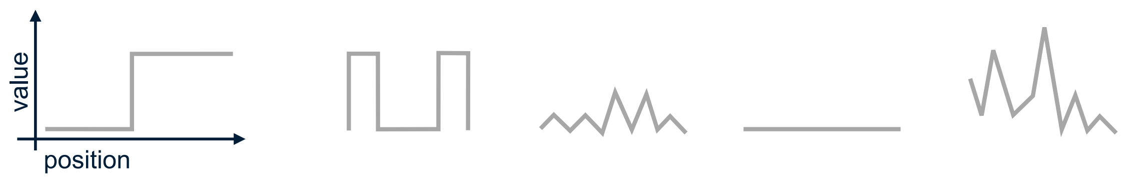
This isn't the first time that we've seen the ratio estimator in this course. Tomasz Stachowiak's screen-space reflections used this formulation for variance reduction back in 2015. He didn't refer to it as a ratio estimator, but now we have a name for it.

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

DENOISING

Okay, so we have a solution, but there are some practical details that I'd like to dive into now. The first is denoising.

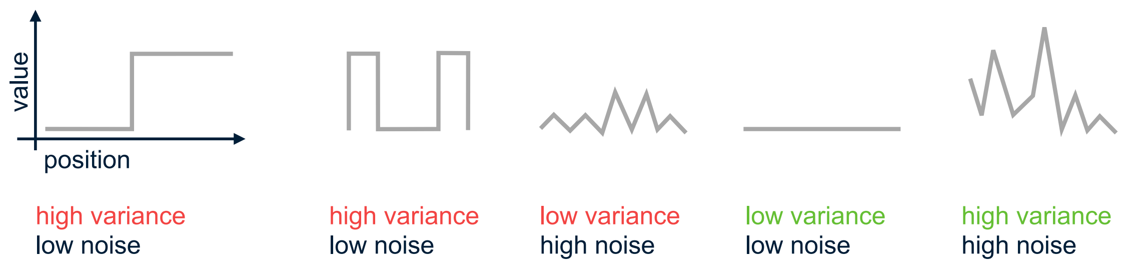
DENOISING



For our denoiser, we’re going to be applying a bilateral filter. For this, we need to decide how much to blur. We should do this based on the amount of noise in some way.

DENOISING

GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018



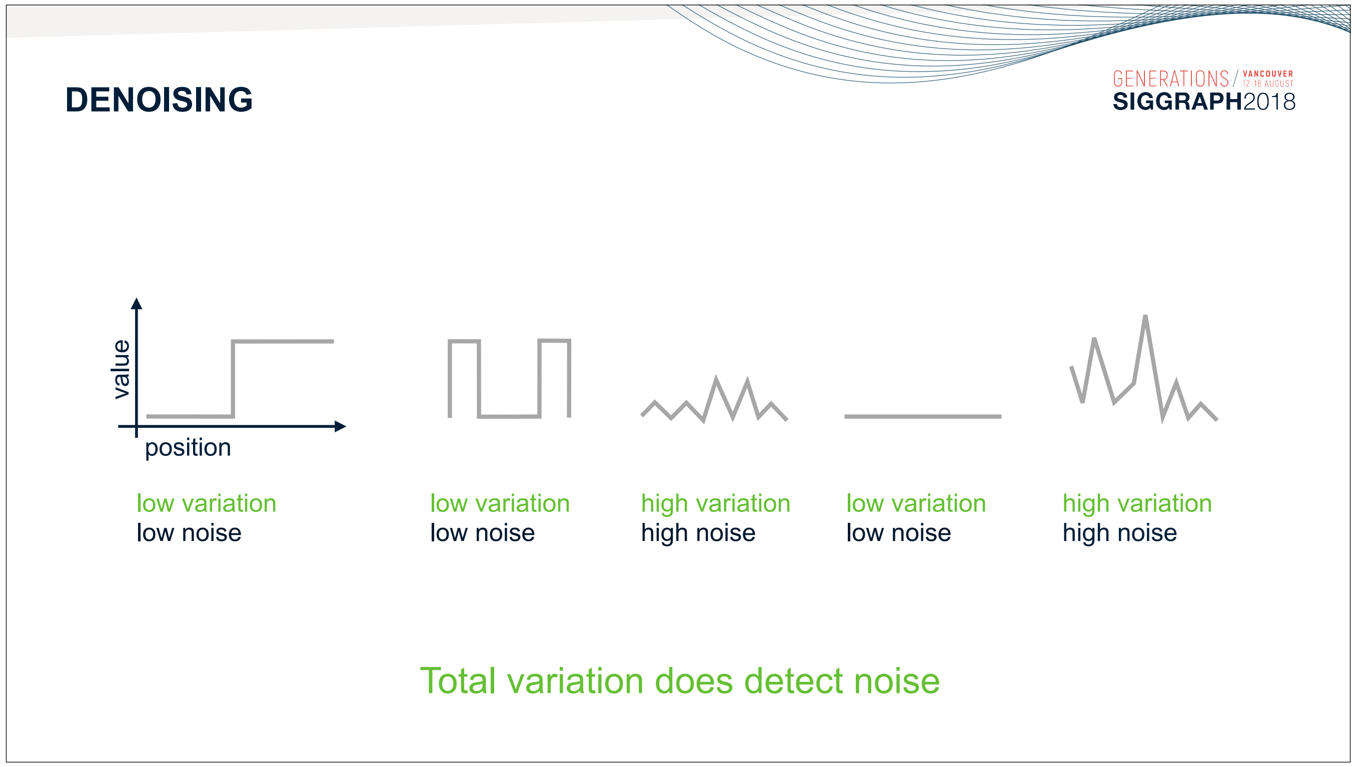
Variance doesn’t detect noise

Noise is variation. But despite a common misconception, statistical *variance* does not measure this kind of variation!

Variance tells you how many of the values are far from the average, but it doesn’t care about the relative spatial location of those deviations.

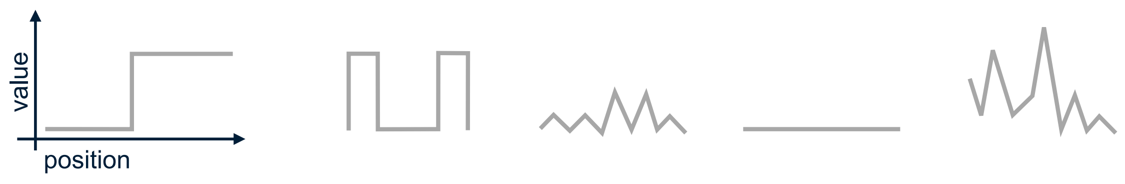
For these 1D schematics, you can see that variance doesn’t tell us anything about noise.

Some previous denoisers used variance as a metric and this is why it’s a mistake.



DENOISING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

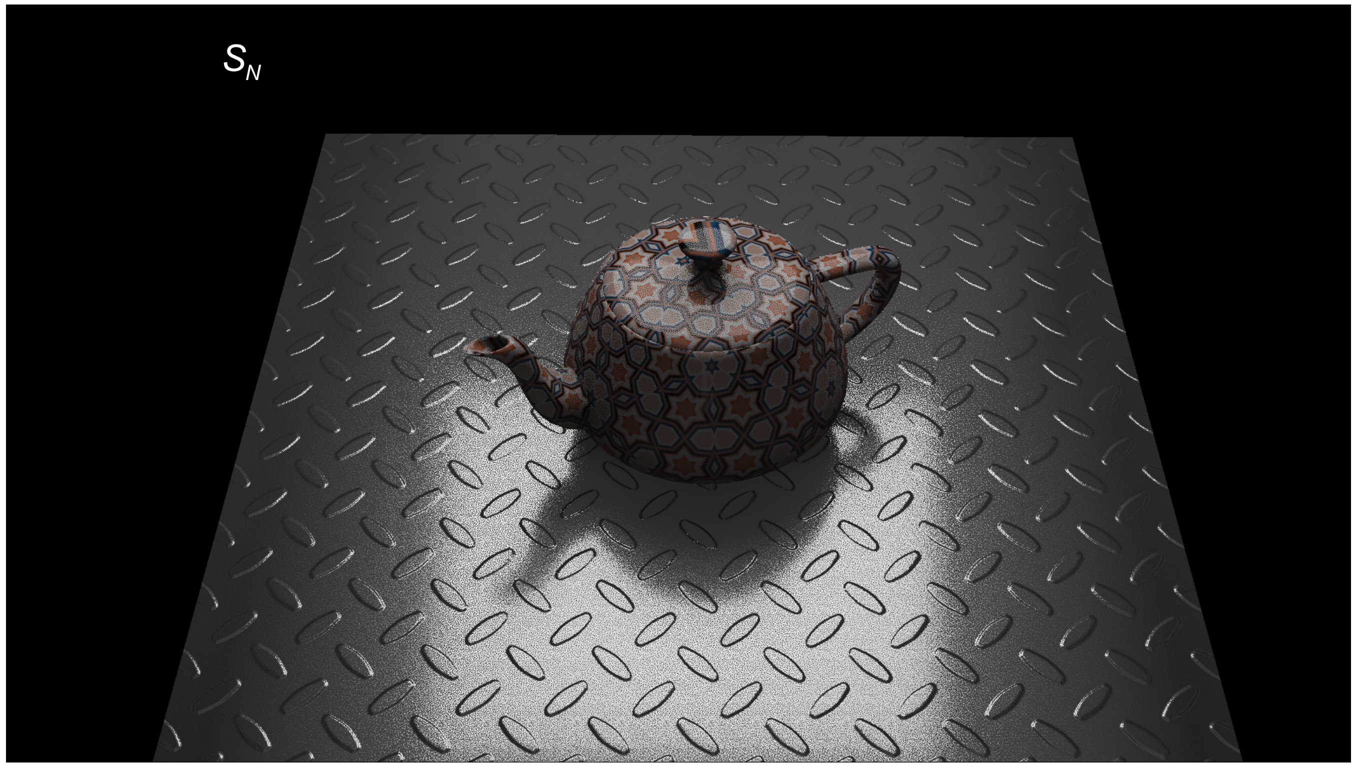


low variation	low variation	high variation	low variation	high variation
low noise	low noise	high noise	low noise	high noise

Total variation does detect noise

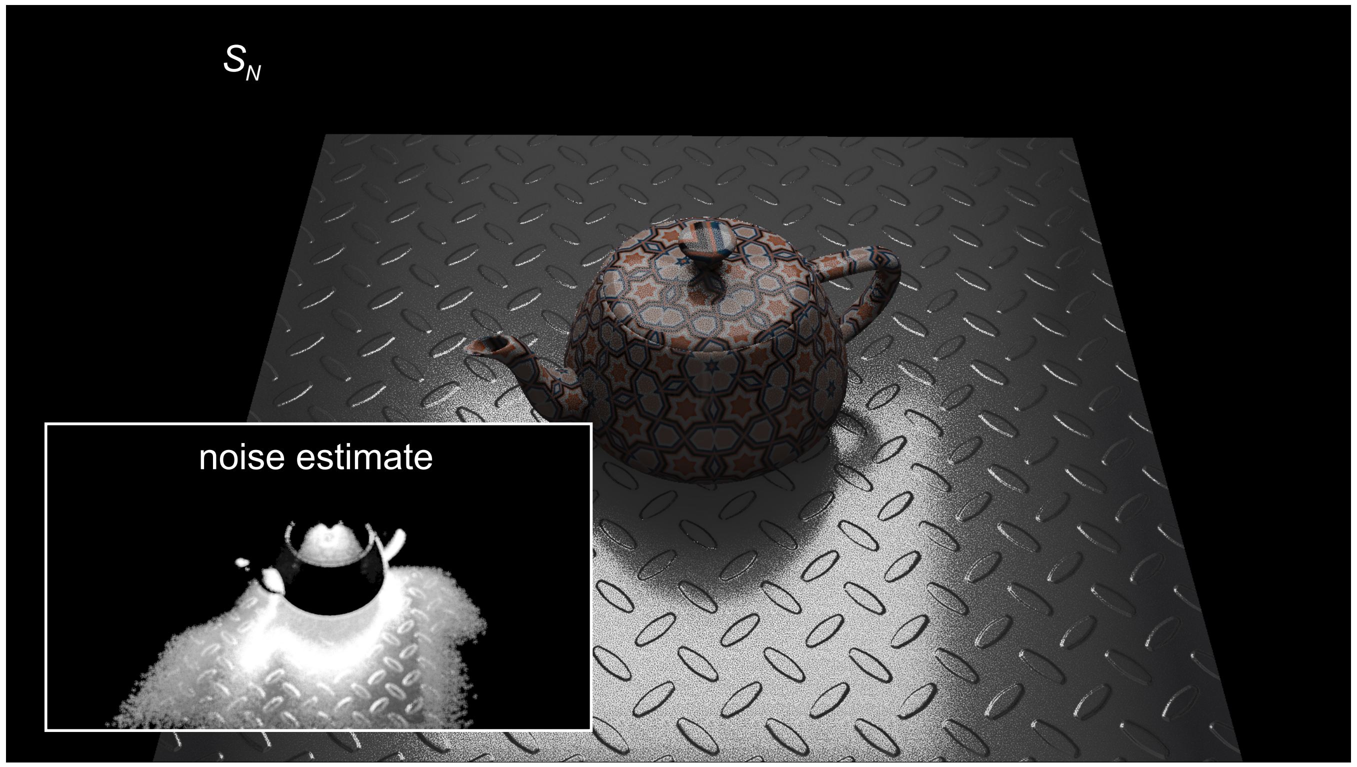
A better measure is called *total variation* in the statistics literature. This measures the variation of acceleration (inflections). We discuss the math behind this in our paper.

This is an estimate of noise along a line. To estimate noise in a 2D rectangular patch, we take random lines through the centre pixel and average their results. That randomness means that our noise estimate is itself noisy, so we blur it a little.

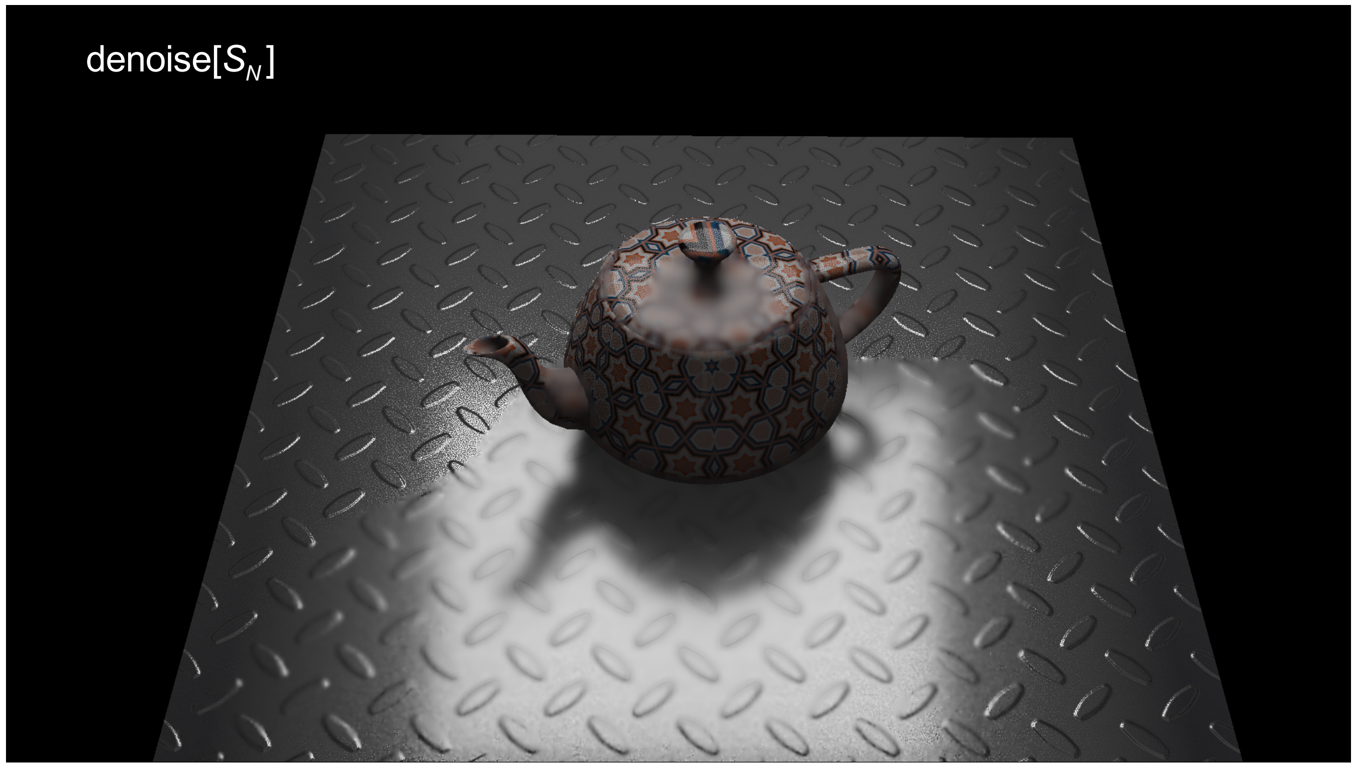


Since we’re going to denoise two images — U_N and S_N (shown here) — and take their ratio, we only care about noise that differs between them.

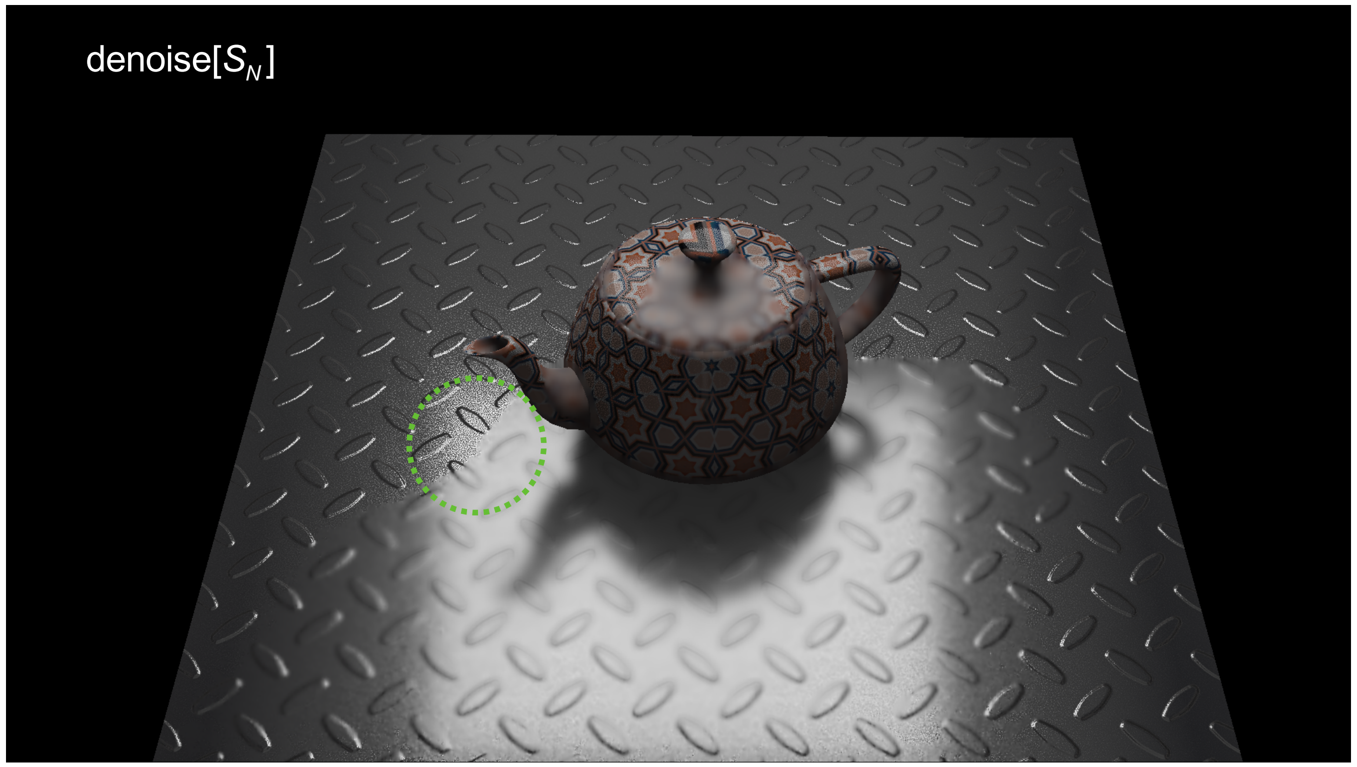
So, we measure total variation of the *ratio,* and then use that to control the filter applied to each image. This avoids spending bandwidth denoising areas that aren’t in shadow.



Here is the noise estimate. Note that it has correctly identified areas where the shadow term is noisy.

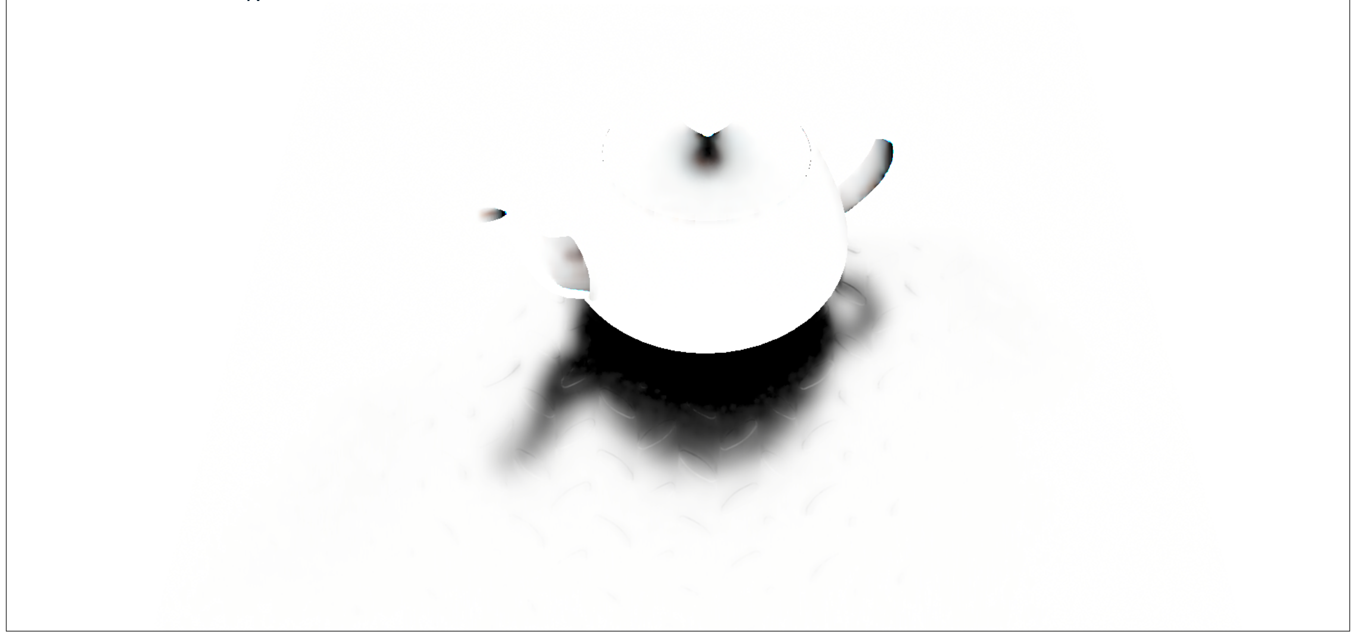


Here is the result of denoising based on that estimate, but only where there are shadows.

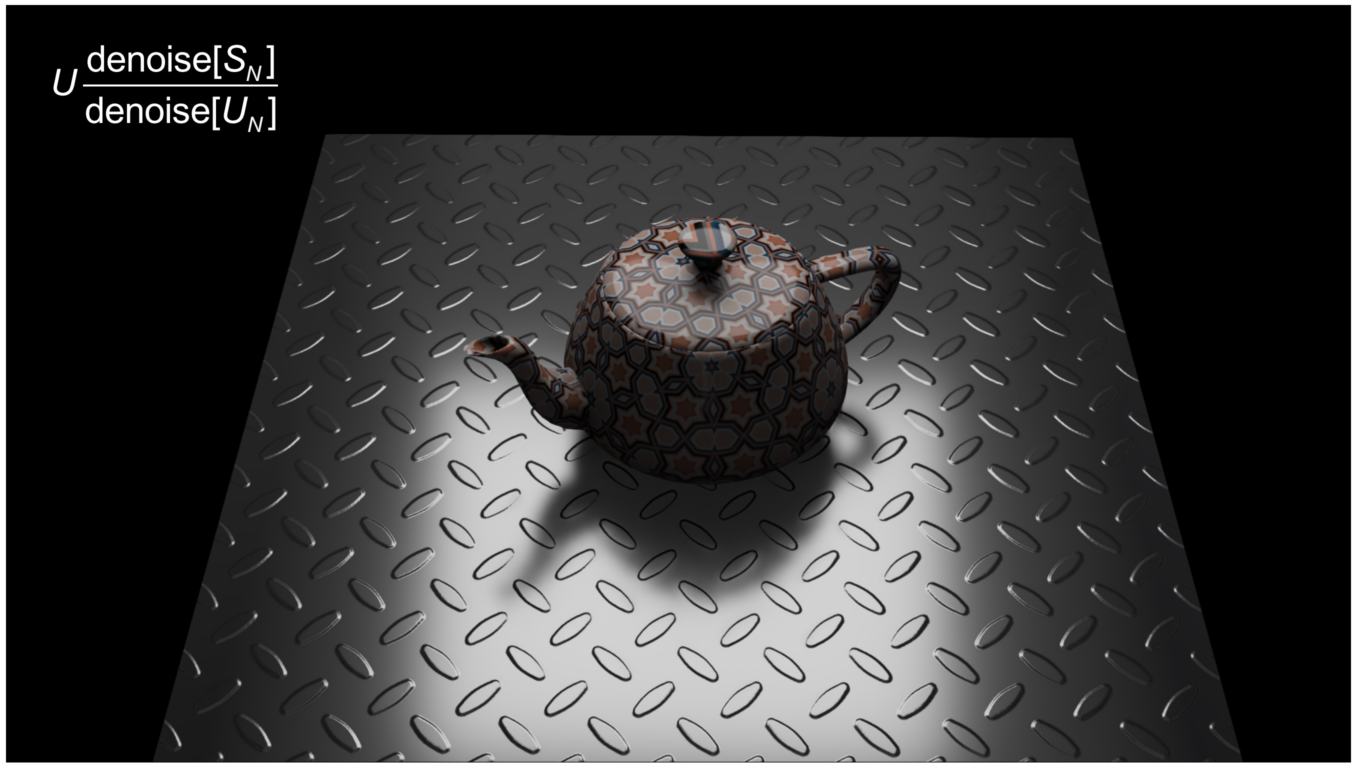


This highlighted area shows where the filter hasn't been applied, because there's no <u>shadow</u> noise. The noise is just in the lighting, which is the same in both S_N and U_N, so it will cancel when we compute the ratio.

$$\frac{\text{denoise}[S_N]}{\text{denoise}[U_N]}$$



Here is the denoised ratio (shadow image).



Finally, here's the end result that combines our analytic term, *U*, with the denoised shadows.

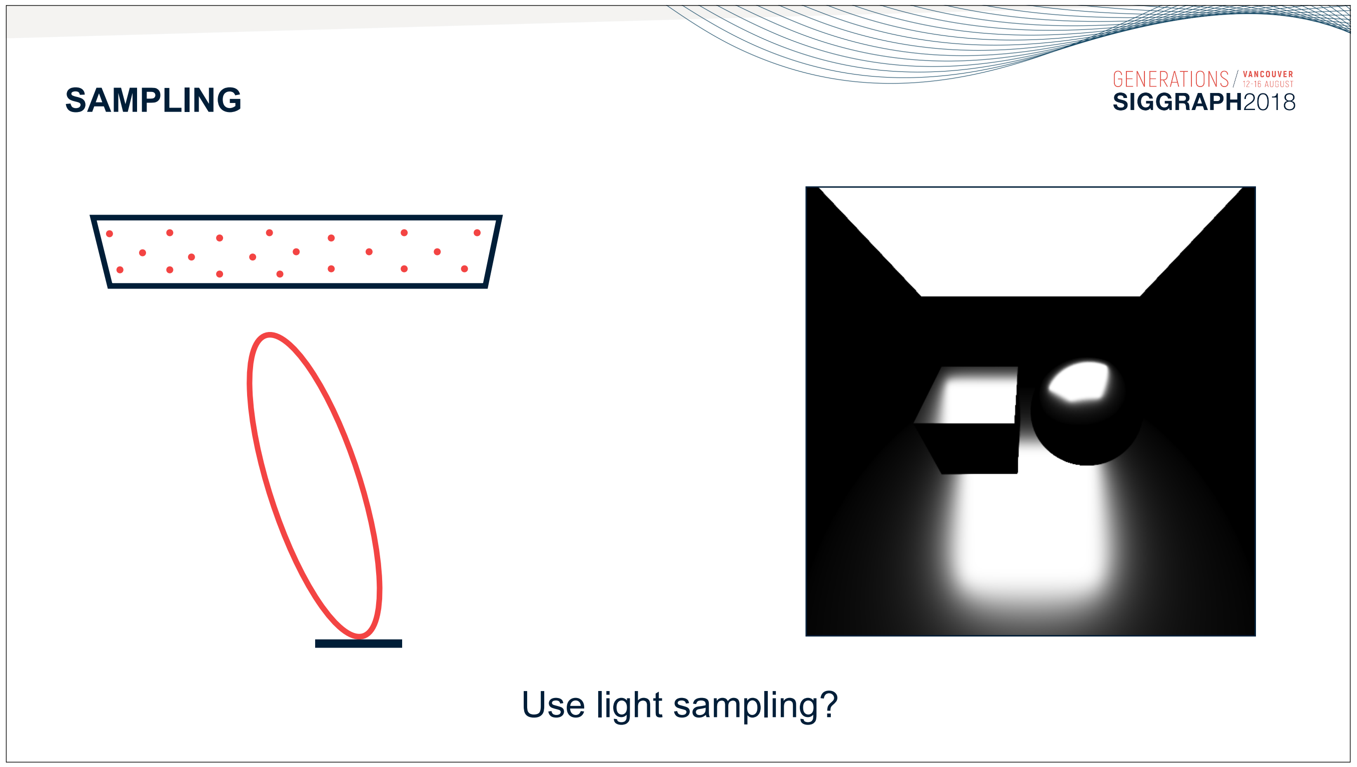
In practice, we denoise both signals simultaneously using two render targets, and the final pass directly computes and applies the ratio.

We've released the full source code for our implementation.

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

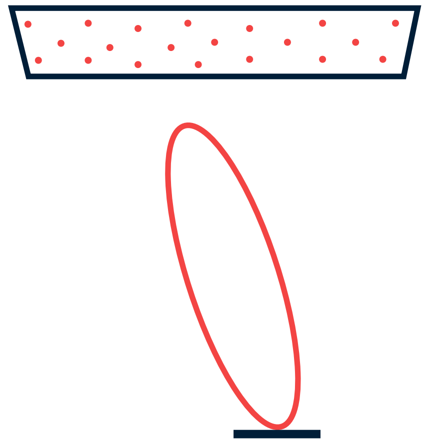
SAMPLING

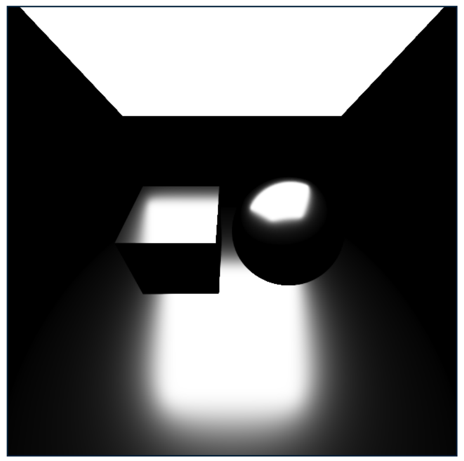
One remaining and critical detail is how to sample the visibility. In other words: “What shadow rays should we trace?”



SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

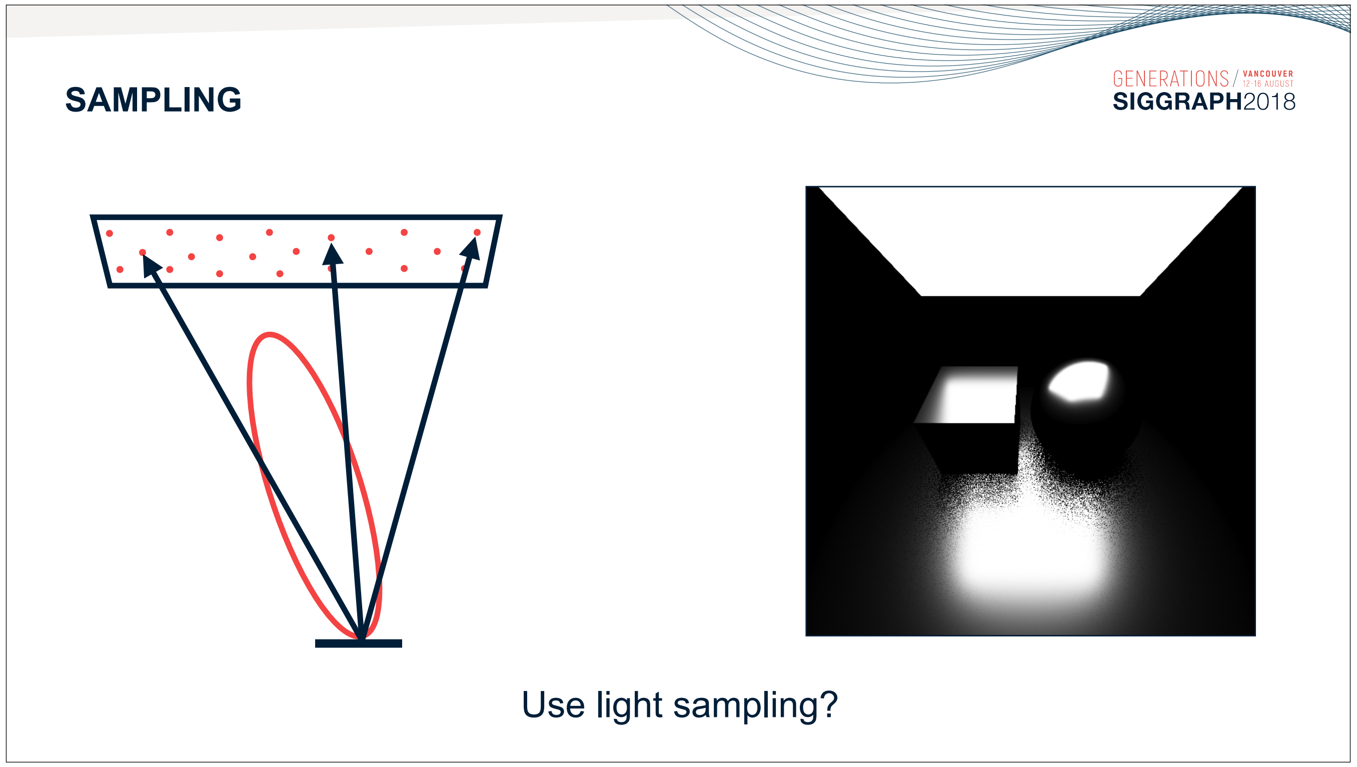




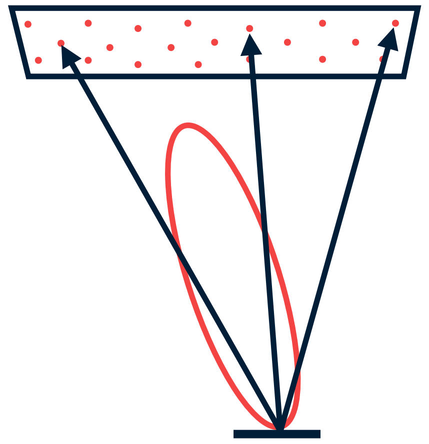
Use light sampling?

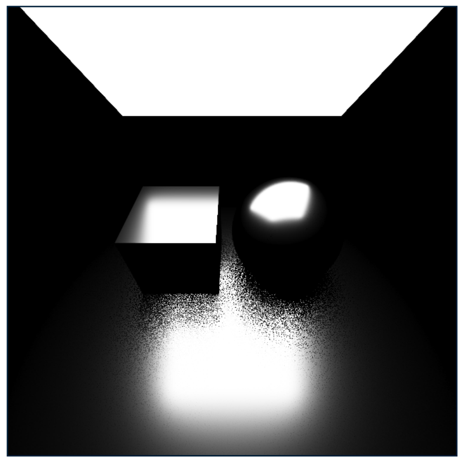
We could generate samples in the direction of the light, much like the GDC demos.

This guarantees that we will hit the light, but let’s see how it performs for the glossy conductor we have here.


SAMPLING

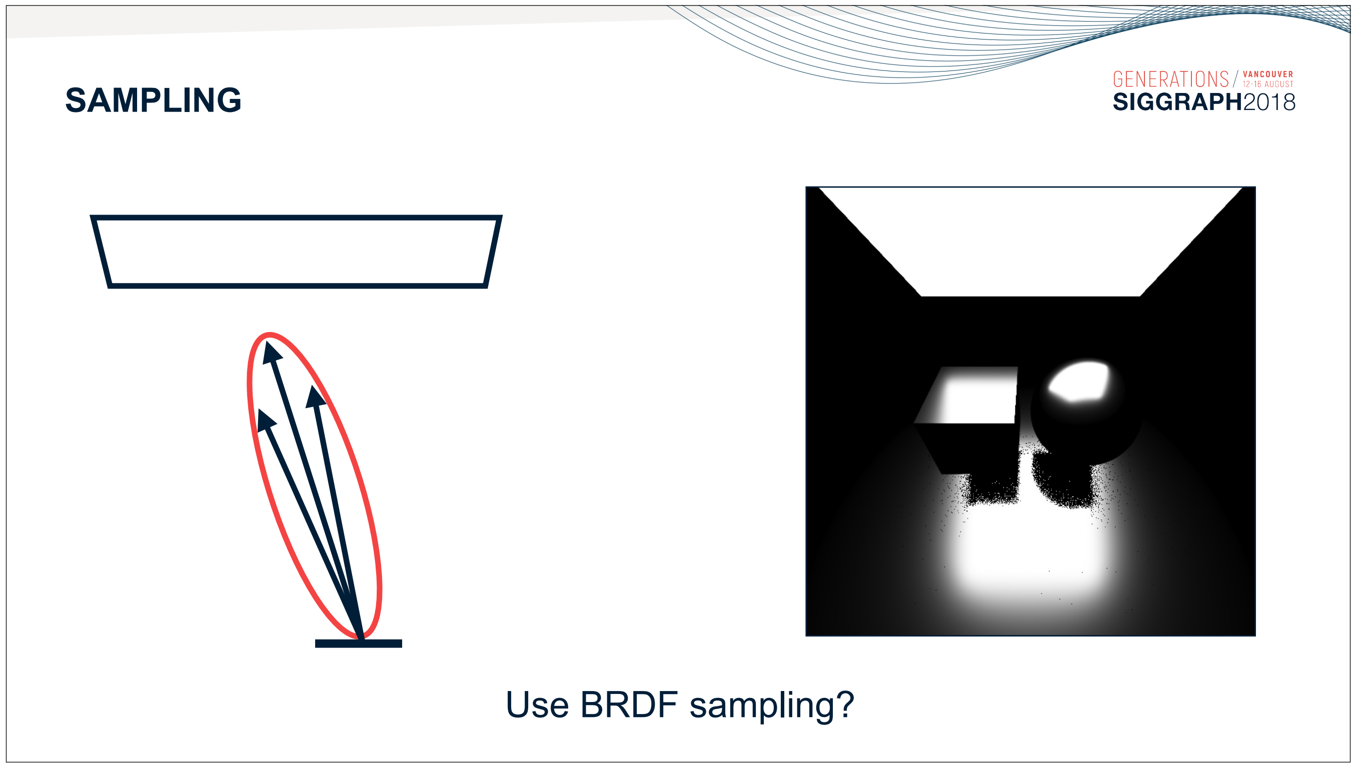
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018





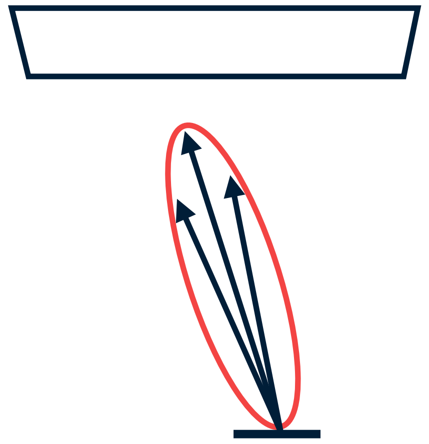
Use light sampling?

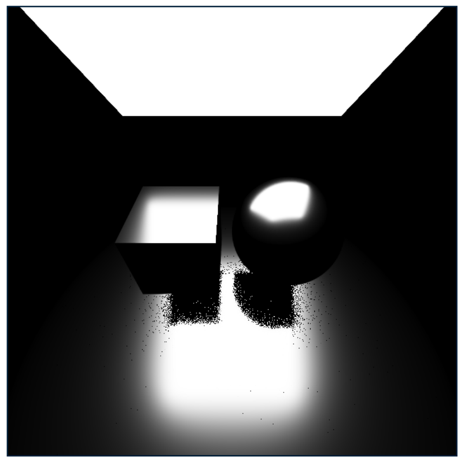
Our light sampling strategy often generates directions that are not near the peak of the specular lobe. This leads leads to slow convergence, hence we have noisy results with a low number of samples. What we should be seeing is a sharper shadow in the highlight.



SAMPLING

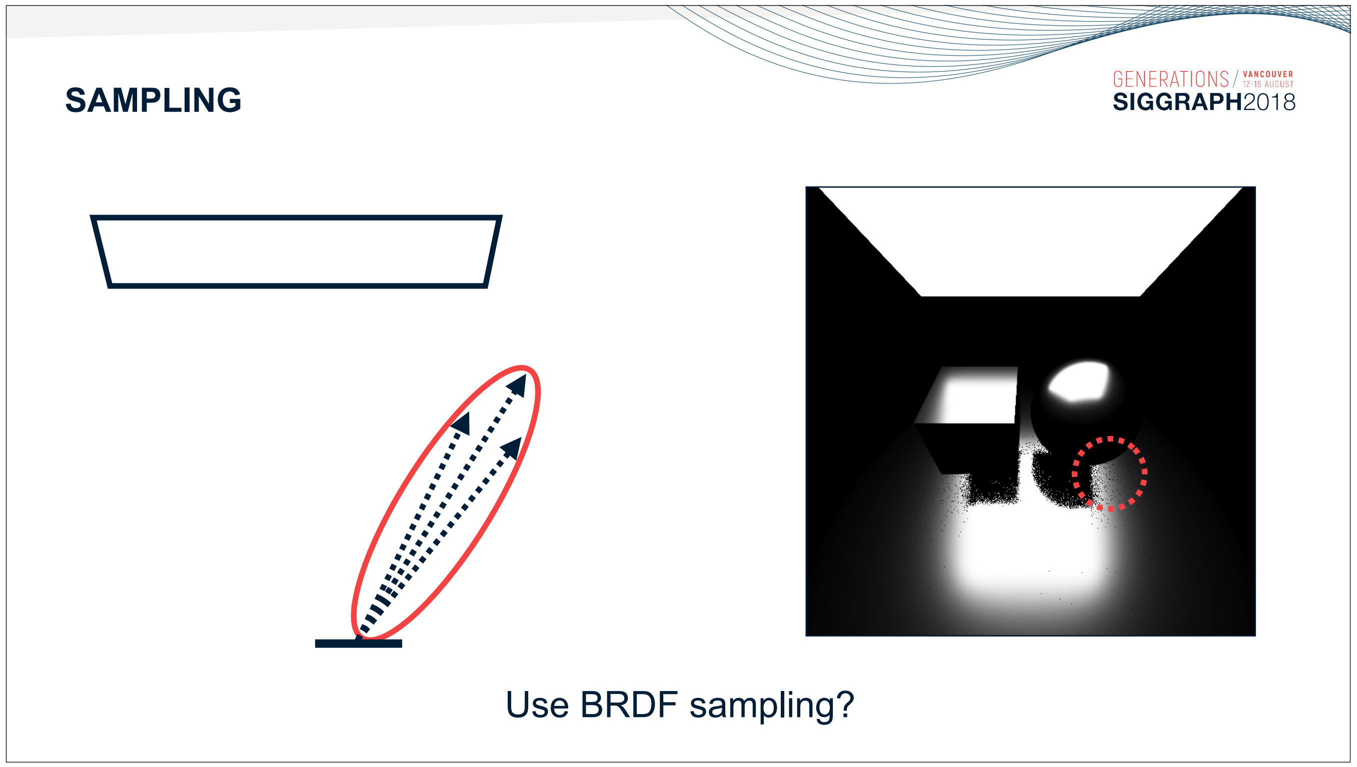
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018



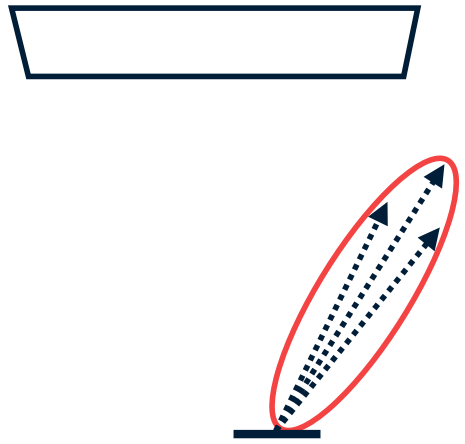


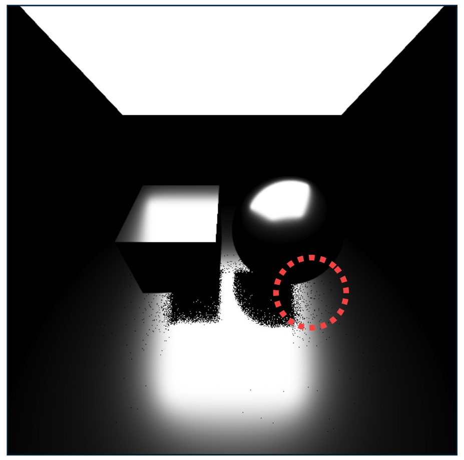
Use BRDF sampling?

If instead we importance sample the specular lobe, we get a more defined shadow in the highlight.



SAMPLING





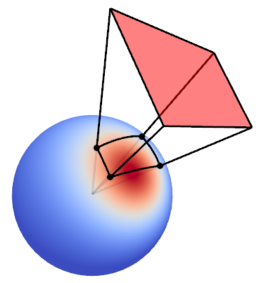
Use BRDF sampling?

However, we’re now missing shadowing near the edge of the highlight. This is due to shadow rays missing the light — in this case, we don’t have a valid light hit.

Again, with enough samples, we’ll converge to the right result, but it would be nice if we could get more mileage out of just a handful of samples.

Fortunately, there’s an easy way to combine the strengths of both sampling strategies…

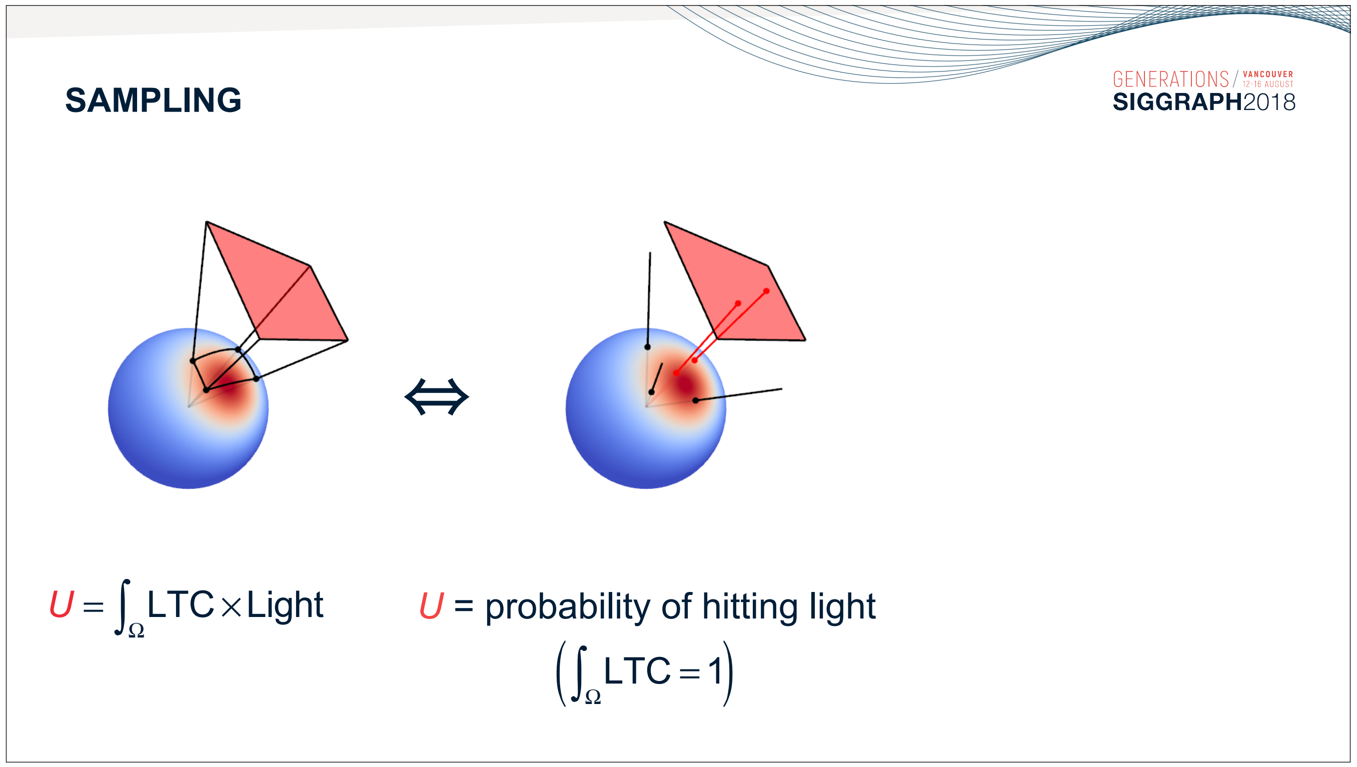
SAMPLING



$$U = \int_{\Omega} \text{LTC} \times \text{Light}$$

The trick is to use other information that we already have: our analytic *U* term.

The LTC part of this lobe (not including the directional albedo)…



SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

$\Leftrightarrow$

$$U = \int_{\Omega} \mathrm{LTC} \times \mathrm{Light}$$

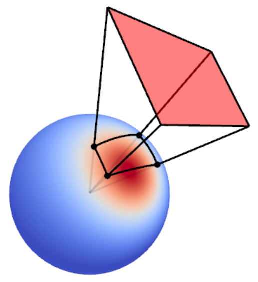
U = probability of hitting light

$$\left(\int_{\Omega} \mathrm{LTC} = 1\right)$$

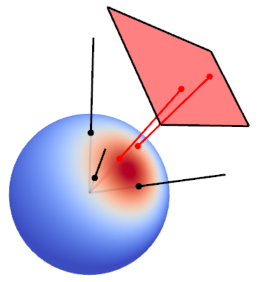
…is equivalent to the probability of our BRDF visibility samples (drawn from the LTC distribution) hitting the light.

Note: the LTC lobe incorporates the projection cosine and integrates to 1 over the sphere.

SAMPLING



$\Leftrightarrow$



$\Rightarrow$

```
if randf() ≤ U:
  // BRDF sample
else:
  // Light sample

pdf = lerp(pdf_Light,
           pdf_BRDF, U)
```

$U = \int_\Omega \mathrm{LTC} \times \mathrm{Light}$

U = probability of hitting light

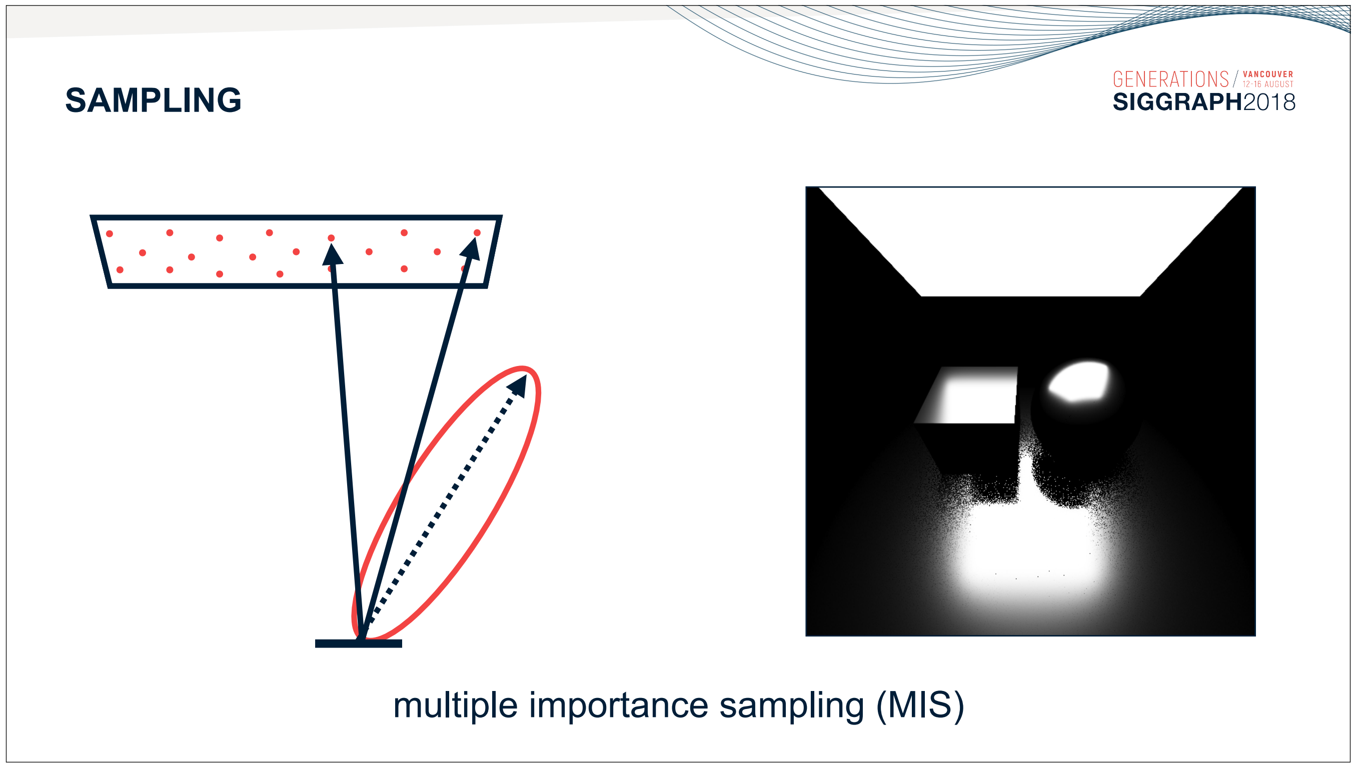
$\left(\int_\Omega \mathrm{LTC} = 1\right)$

sample based on U

We can use this *hit probability* to decide when to BRDF sample and when to light sample.

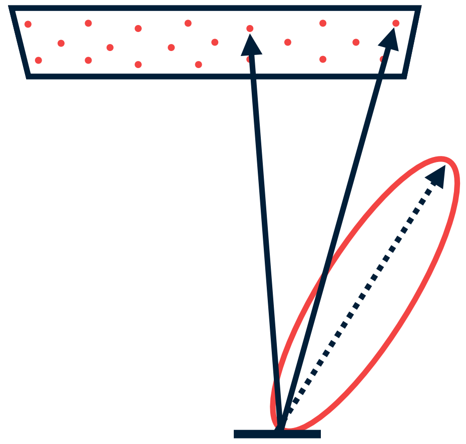
So, not only are we able to use the strengths of the unshadowed analytic lighting and stochastic shadows, but we can also use the analytic result to improve our sampling. This is a nice symbiosis.

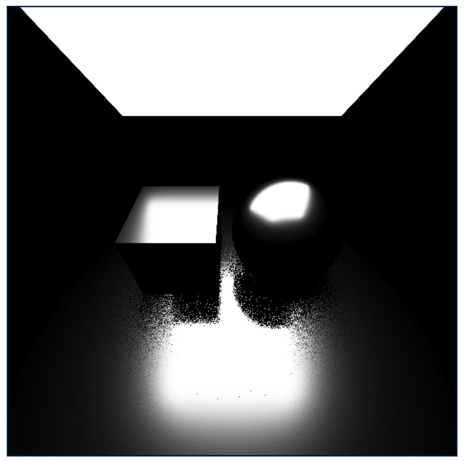
The sampling PDF ends up being a linear blend between the light and BRDF PDFs.



SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

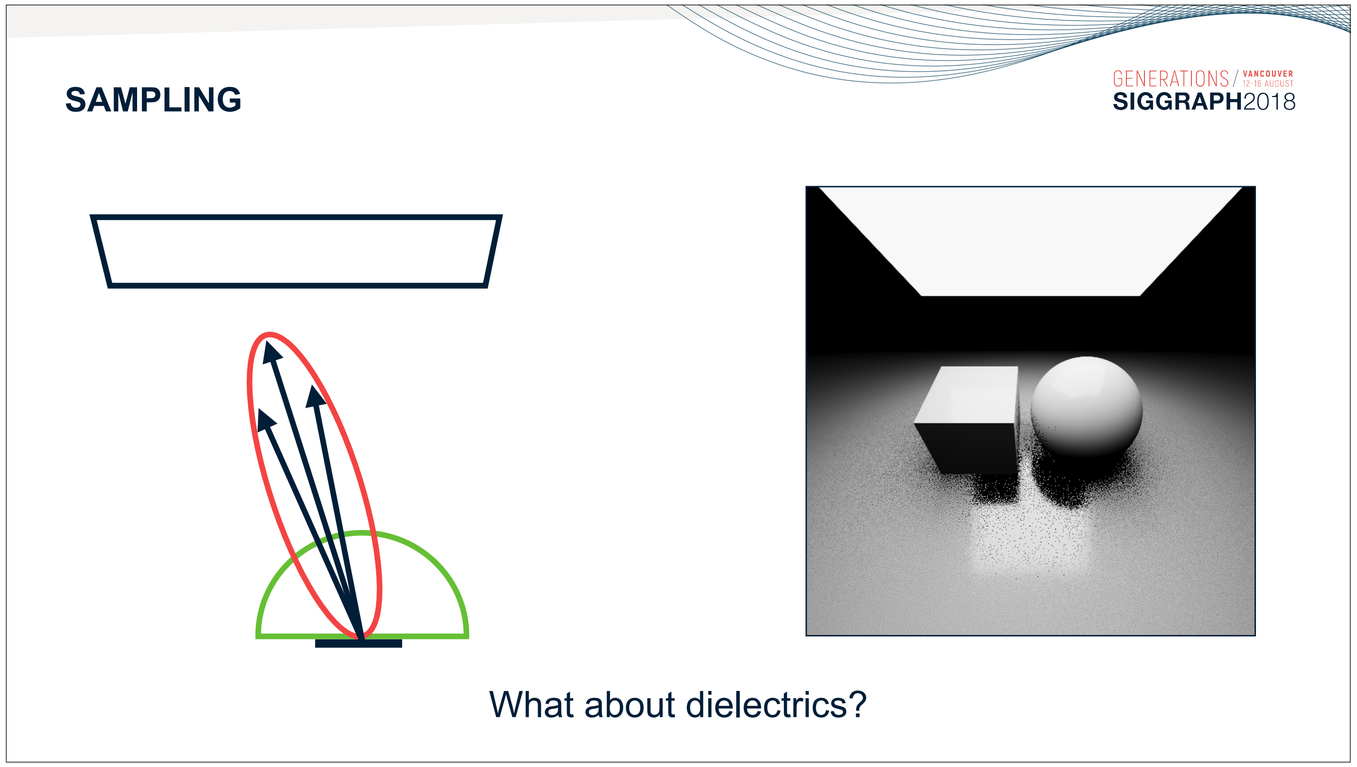




multiple importance sampling (MIS)

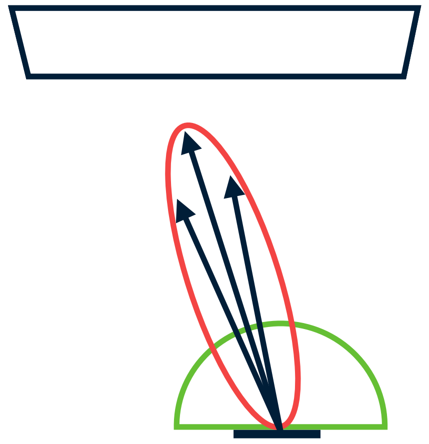
What we end up with here is a blend between mostly BRDF sampling in the centre of the highlight — where there is a high probability of BRDF samples hitting the light — and mostly light sampling near the edge of the highlight.

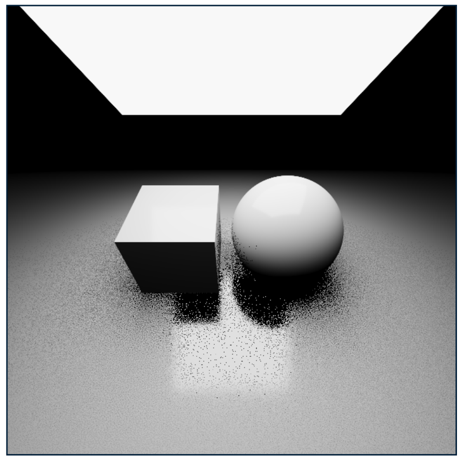
This combined strategy is a form of *multiple importance sampling* (MIS).



SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

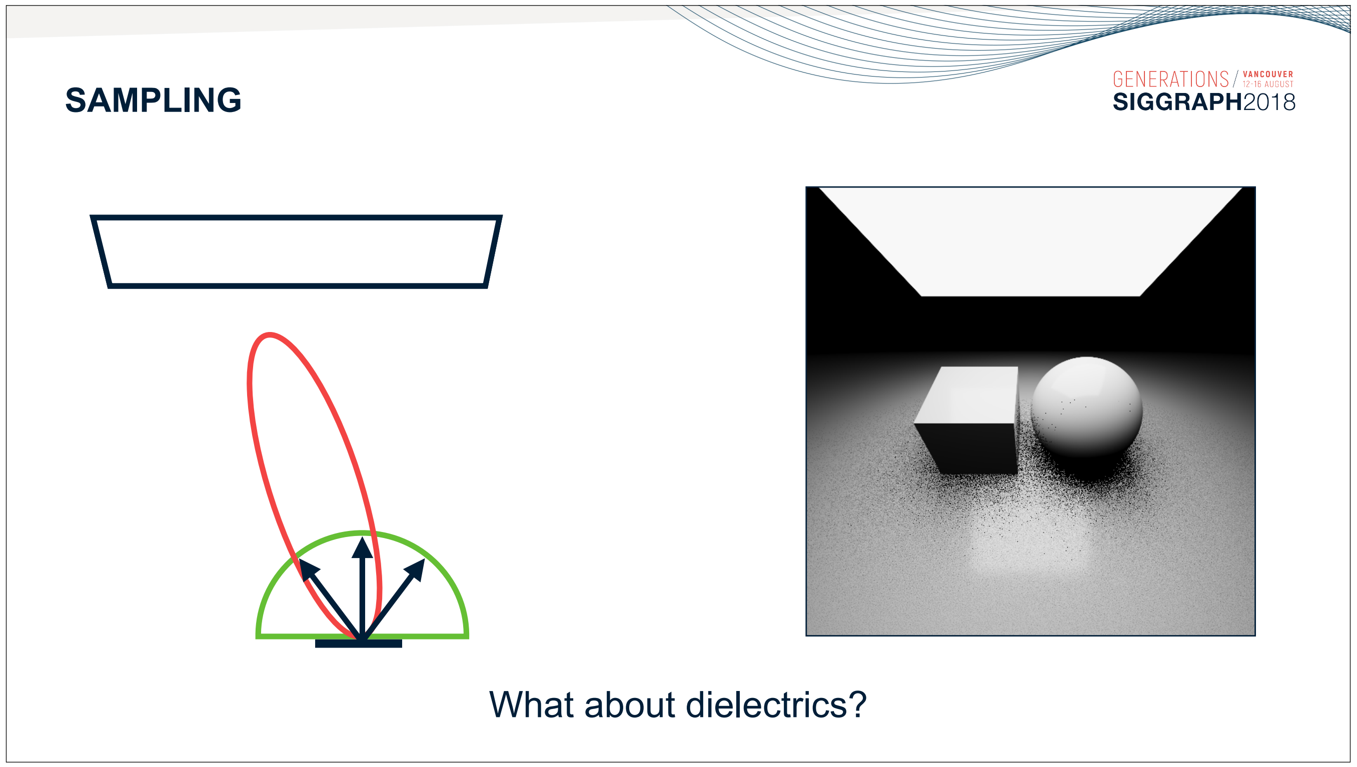




What about dielectrics?

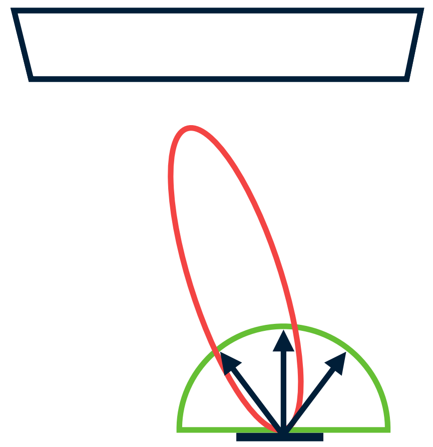
So, that covers single-lobe materials, but what should we do with dielectrics, which are typically modelled using both a specular and a diffuse lobe.

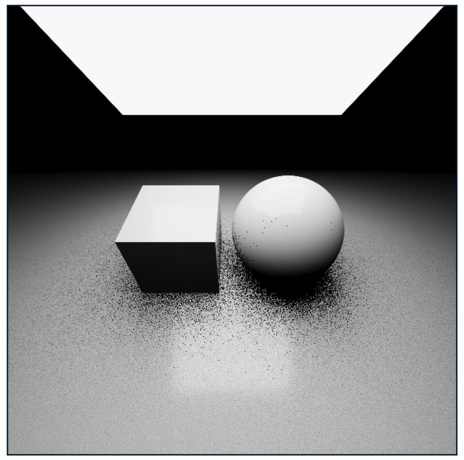
If we stick with sampling just the specular lobe then we get this result. Inside the highlight, the specular shadows are handled well, but diffuse is underrepresented.



SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

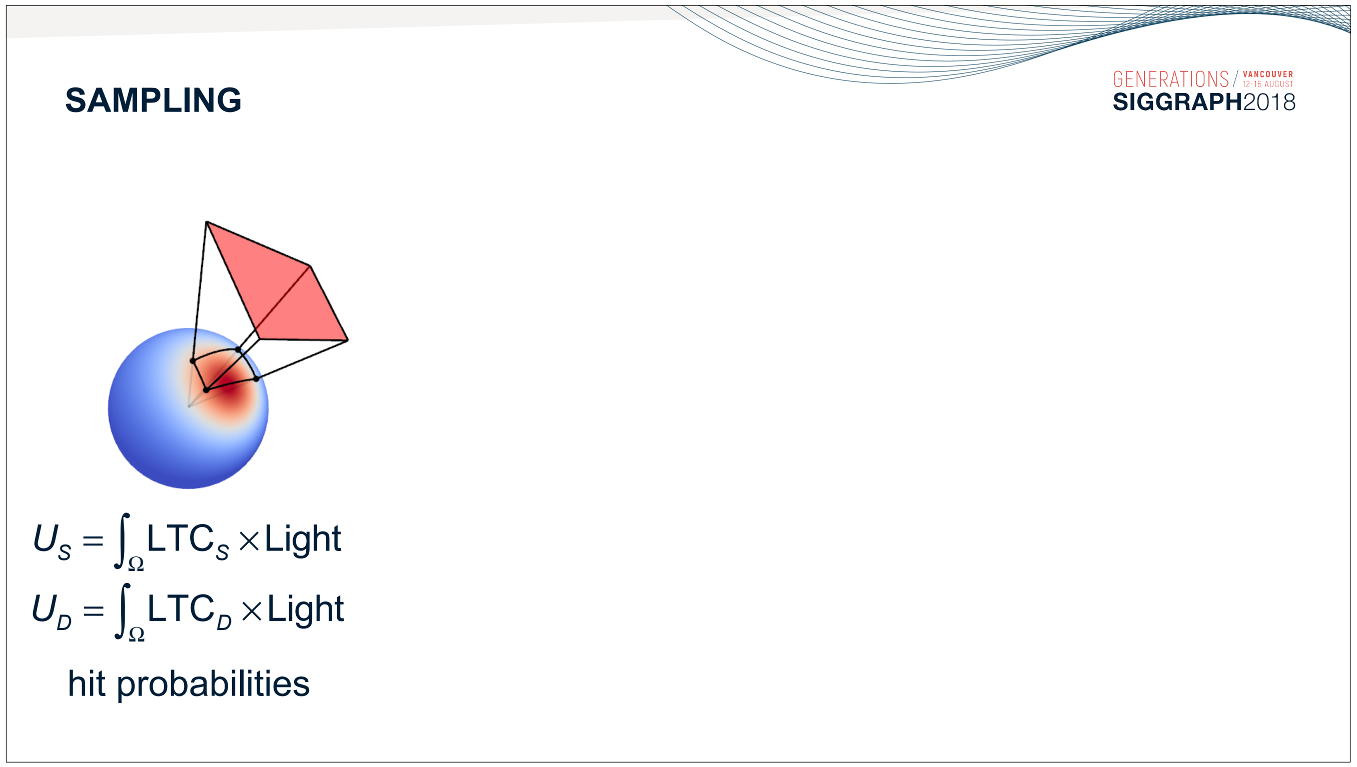




What about dielectrics?

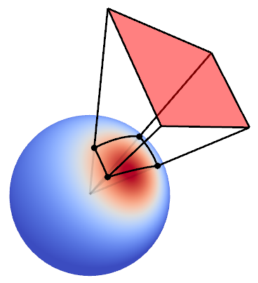
If we use the diffuse lobe for sampling then we have the same problem as before: noise, due to slow convergence (with respect to the specular lobe).

Fortunately, we can extend our multiple importance sampling to address this…



SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018



$$U_S = \int_\Omega \mathrm{LTC}_S \times \mathrm{Light}$$

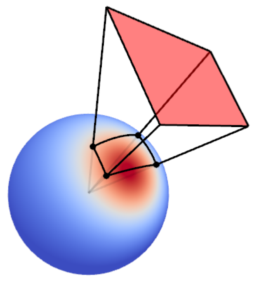
$$U_D = \int_\Omega \mathrm{LTC}_D \times \mathrm{Light}$$

hit probabilities

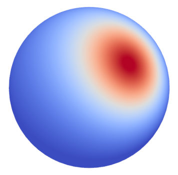
We can use the analytic results for both of the lobes, specular (U_S) and diffuse (U_D).

SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018



+



$$U_S = \int_\Omega \mathrm{LTC}_S \times \mathrm{Light}$$
$$U_D = \int_\Omega \mathrm{LTC}_D \times \mathrm{Light}$$

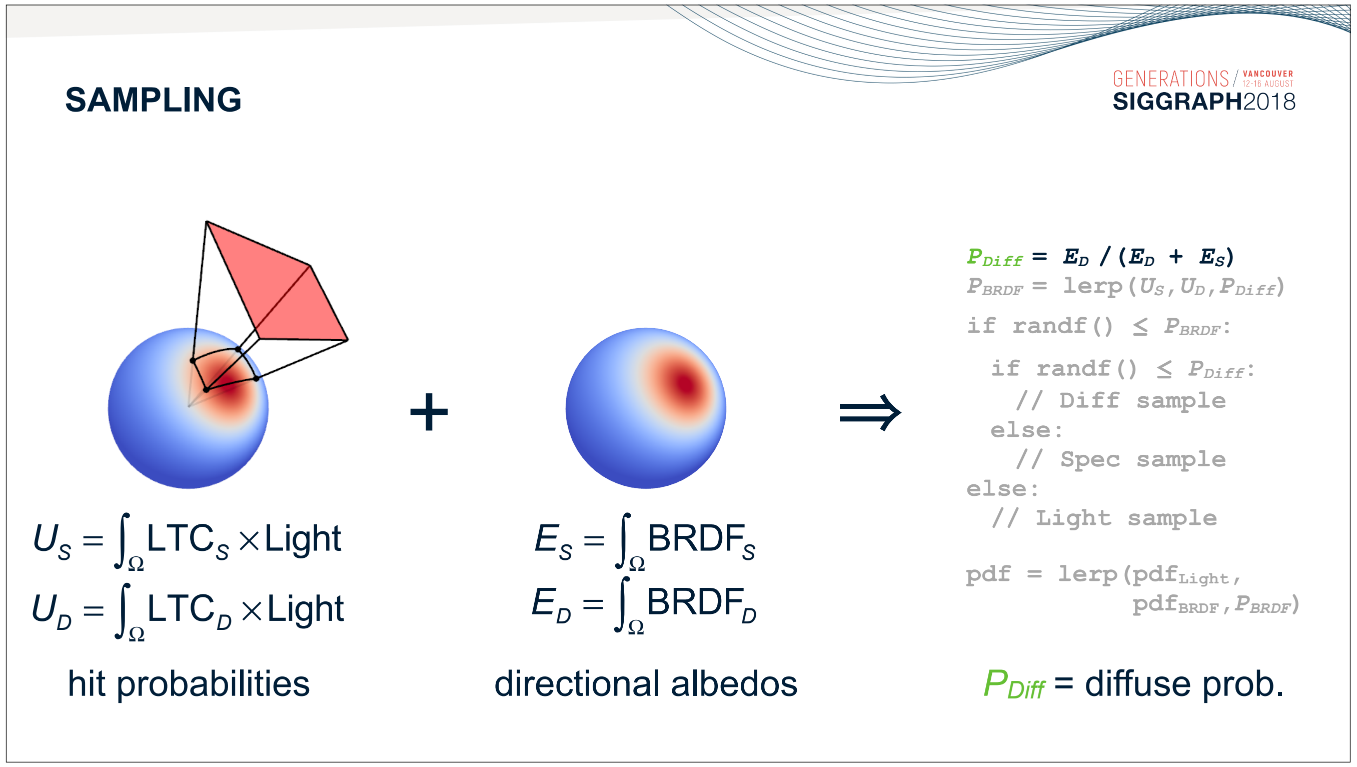
hit probabilities

$$E_S = \int_\Omega \mathrm{BRDF}_S$$
$$E_D = \int_\Omega \mathrm{BRDF}_D$$

directional albedos

Together with their directional albedos, E_S and E_D.

Note: these are already available as part of the full analytic area lighting computation. E_S is equivalent to the *Environment BRDF* / *Preintegrated DFG* term that's commonly used with prefiltered environmental lighting. For purely Lambertian diffuse, E_D is simply the diffuse colour / albedo of the material.



SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

$$U_S = \int_\Omega \text{LTC}_S \times \text{Light}$$
$$U_D = \int_\Omega \text{LTC}_D \times \text{Light}$$

hit probabilities

$+$

$$E_S = \int_\Omega \text{BRDF}_S$$
$$E_D = \int_\Omega \text{BRDF}_D$$

directional albedos

$\Rightarrow$

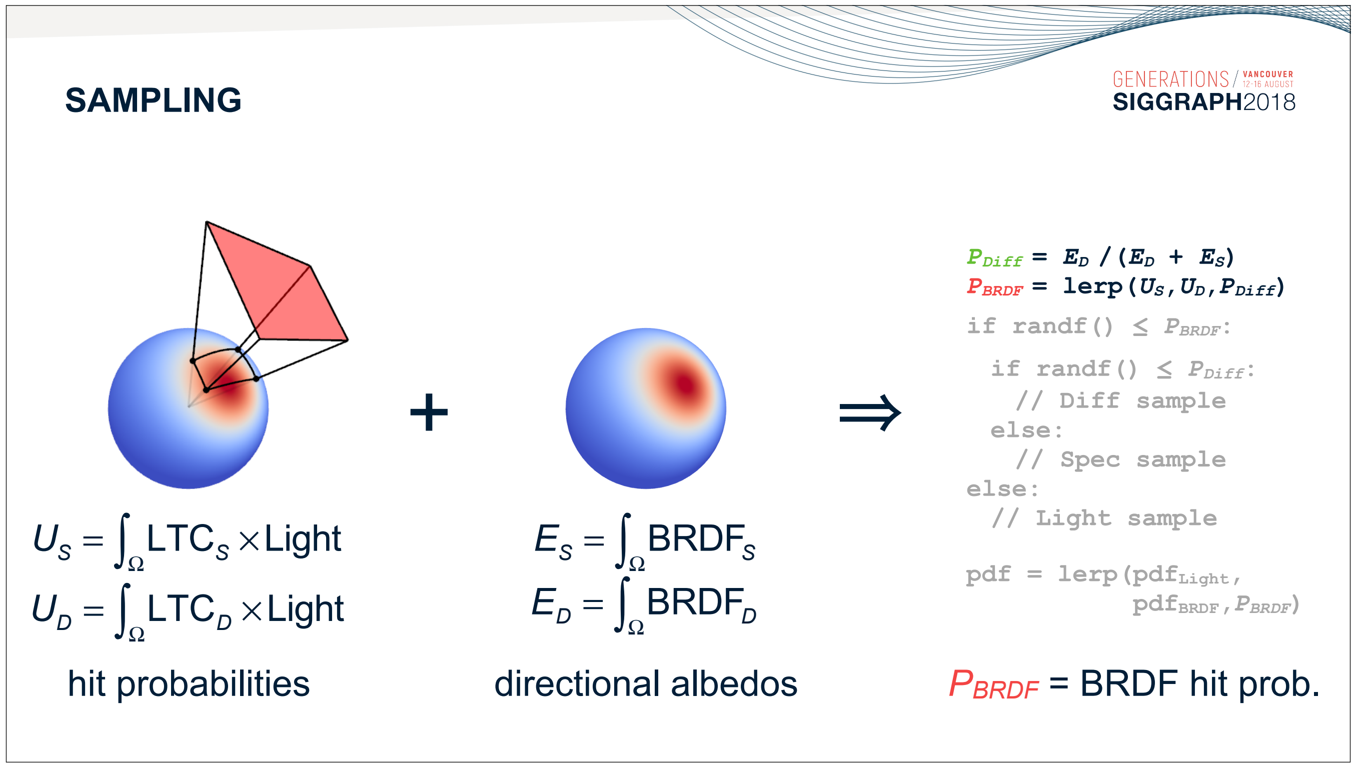
```
$P_{Diff}$ = $E_D$ / ($E_D$ + $E_S$)
$P_{BRDF}$ = lerp($U_S$,$U_D$,$P_{Diff}$)
if randf() ≤ $P_{BRDF}$:
  if randf() ≤ $P_{Diff}$:
    // Diff sample
  else:
    // Spec sample
else:
  // Light sample

pdf = lerp(pdf$_{Light}$,
           pdf$_{BRDF}$,$P_{BRDF}$)
```

P_{Diff} = diffuse prob.

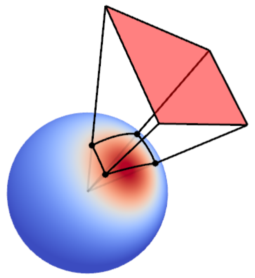
First, using the directional albedos, we can calculate the proportion of reflected light that's diffuse. We can use this is a probability of sampling the diffuse lobe, vs the specular lobe.

This way, we will 'divvy up' samples depending on the relative intensity of specular and diffuse. For instance, more samples will go to specular at grazing angles, due to Fresnel.



SAMPLING

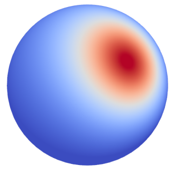
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018



$$U_S = \int_\Omega \text{LTC}_S \times \text{Light}$$
$$U_D = \int_\Omega \text{LTC}_D \times \text{Light}$$

hit probabilities

+



$$E_S = \int_\Omega \text{BRDF}_S$$
$$E_D = \int_\Omega \text{BRDF}_D$$

directional albedos

$\Rightarrow$

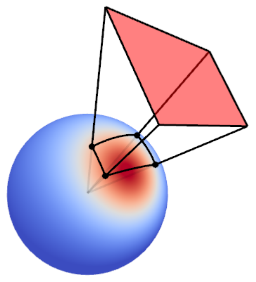
```
$P_{Diff}$ = $E_D$ / ($E_D$ + $E_S$)
$P_{BRDF}$ = lerp($U_S$,$U_D$,$P_{Diff}$)
if randf() ≤ $P_{BRDF}$:
  if randf() ≤ $P_{Diff}$:
    // Diff sample
  else:
    // Spec sample
else:
  // Light sample

pdf = lerp(pdf$_{Light}$,
           pdf$_{BRDF}$,$P_{BRDF}$)
```

P_{BRDF} = BRDF hit prob.

Next, the overall BRDF hit probability, P_{BRDF}, is simply a linear blend of the specular and diffuse hit probabilities, based on P_{Diff}.

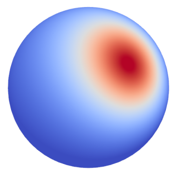
SAMPLING



$$U_S = \int_\Omega \mathrm{LTC}_S \times \mathrm{Light}$$
$$U_D = \int_\Omega \mathrm{LTC}_D \times \mathrm{Light}$$

hit probabilities

+



$$E_S = \int_\Omega \mathrm{BRDF}_S$$
$$E_D = \int_\Omega \mathrm{BRDF}_D$$

directional albedos

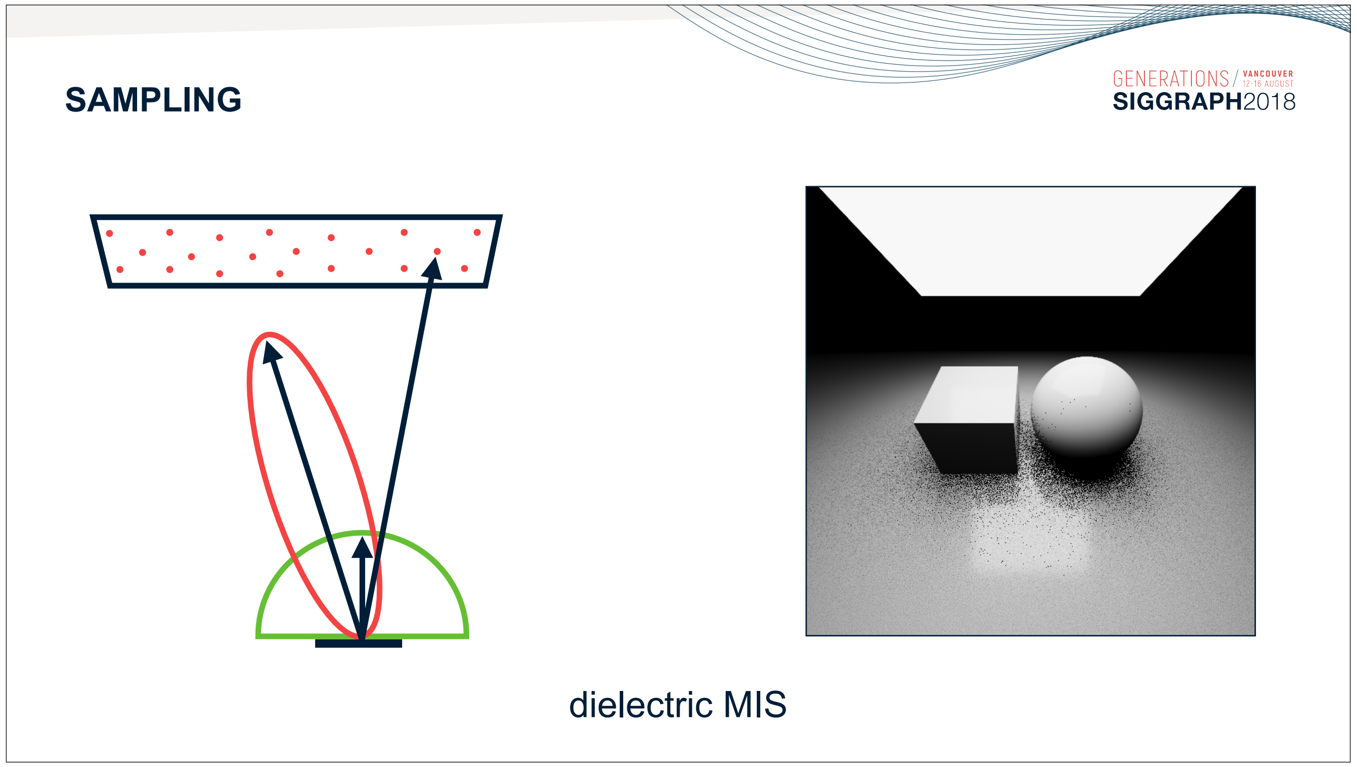
$\Rightarrow$

```
$P_{Diff}$ = $E_D$ / ($E_D$ + $E_S$)
$P_{BRDF}$ = lerp($U_S$,$U_D$,$P_{Diff}$)
if randf() ≤ $P_{BRDF}$:
  if randf() ≤ $P_{Diff}$:
    // Diff sample
  else:
    // Spec sample
else:
  // Light sample

pdf = lerp(pdf_Light,
           pdf_BRDF, $P_{BRDF}$)
```

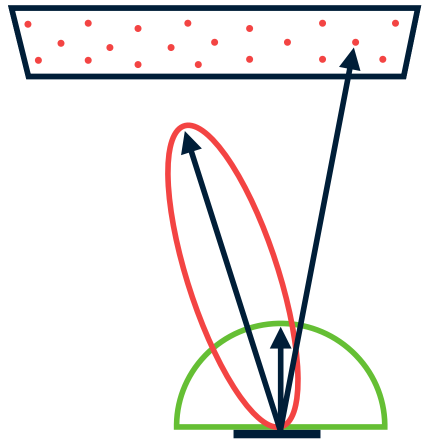
sample ∝ P_{Diff}, P_{BRDF}

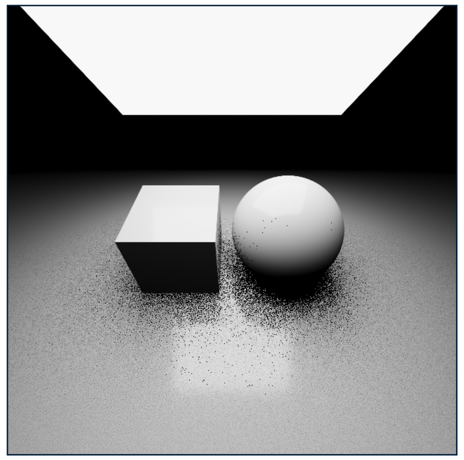
Finally, we can generate samples in proportion to these probabilities: selecting between light and BRDF samples based on P_{BRDF}, and selecting the type of BRDF sample (diffuse or specular) based on P_{Diff}.



SAMPLING

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018





dielectric MIS

This provides a good balance of sampling strategies while being simple to implement.

During development, we also explored using *product importance sampling*, but this was more expensive and caused us to lose the benefits of blue noise jittering across screen pixels. Still, this is an area that’s worth revisiting in the future.

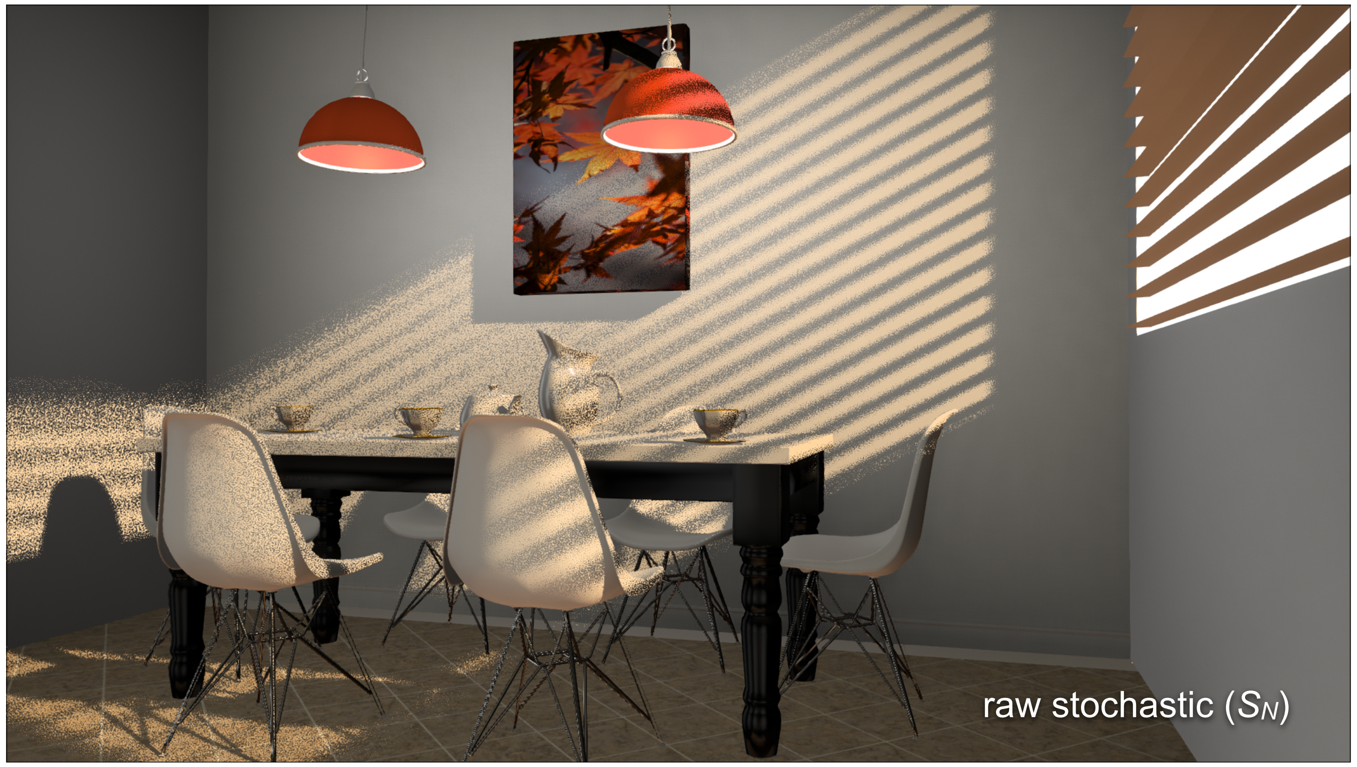
GENERATIONS / VANCOUVER 12-16 AUGUST

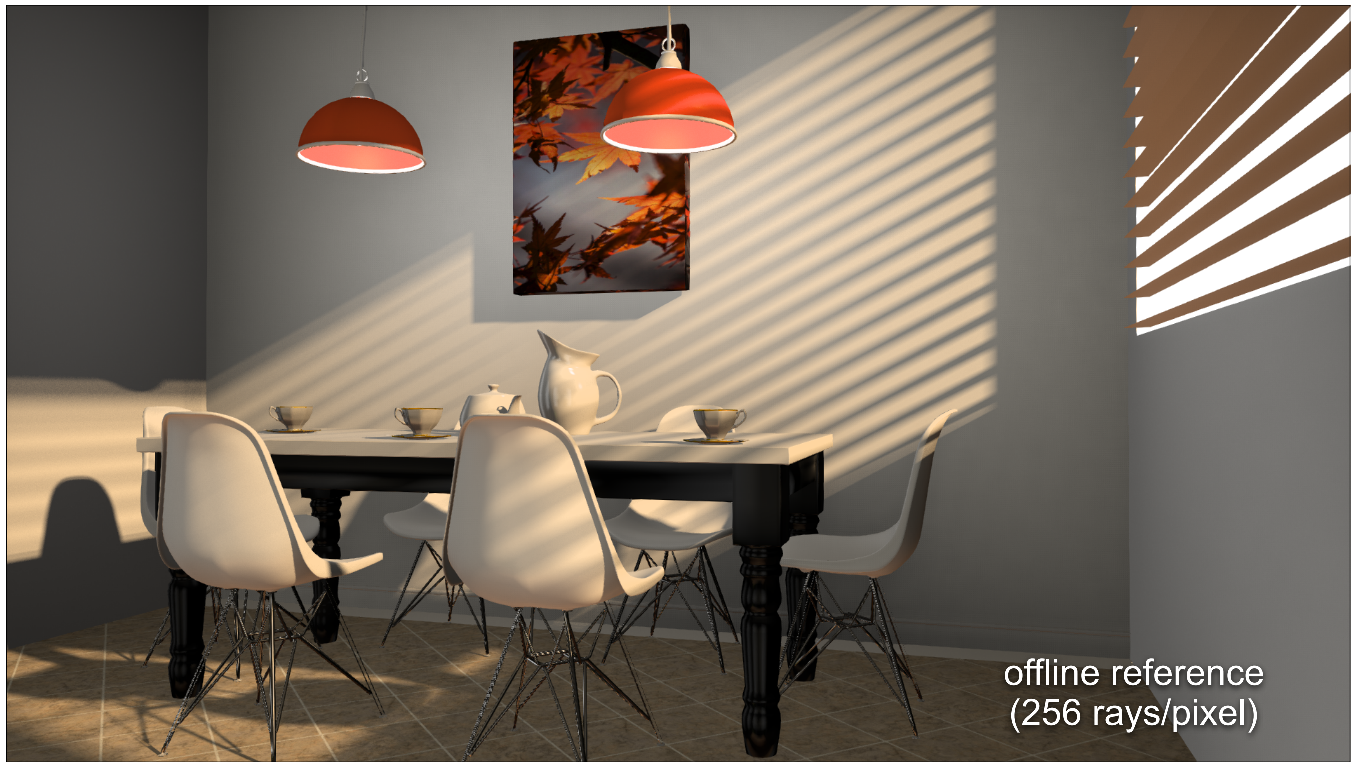
SIGGRAPH2018

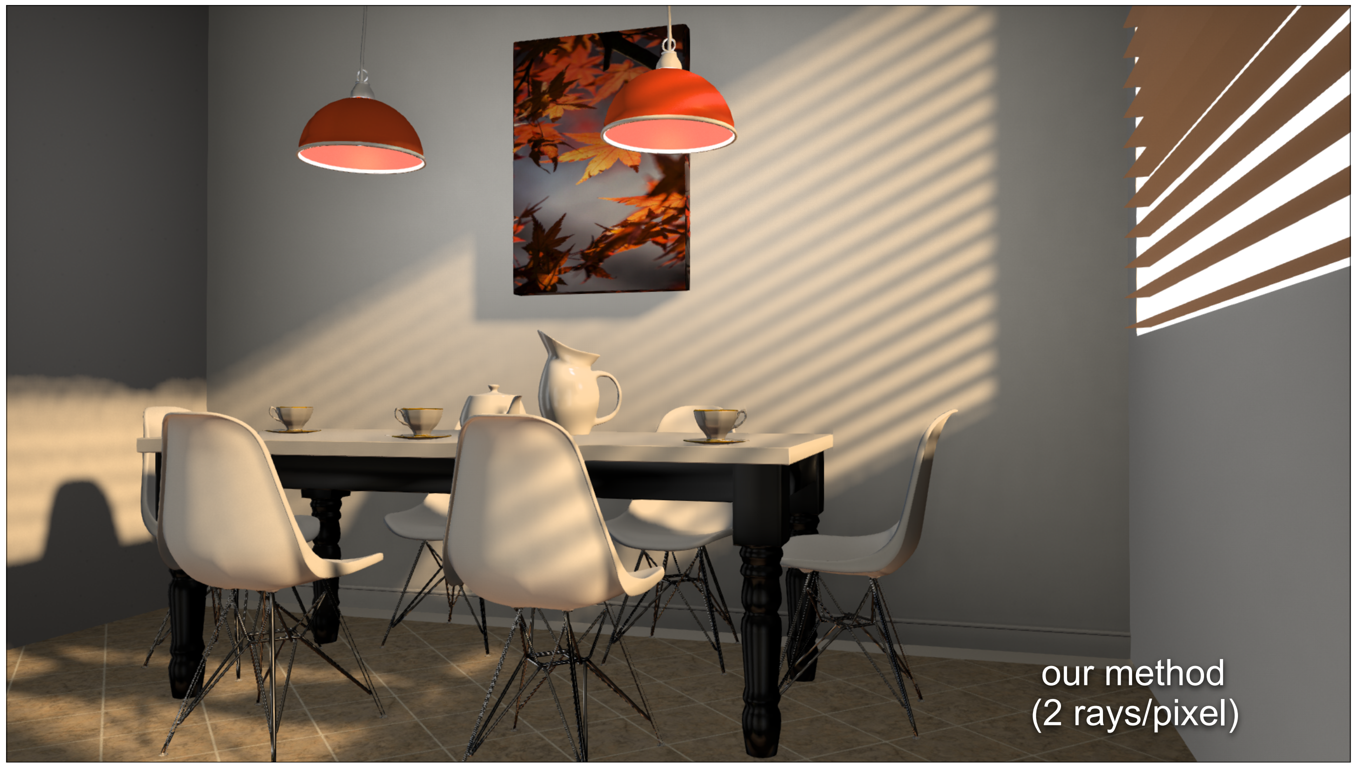
RESULTS

[Morgan’s section]

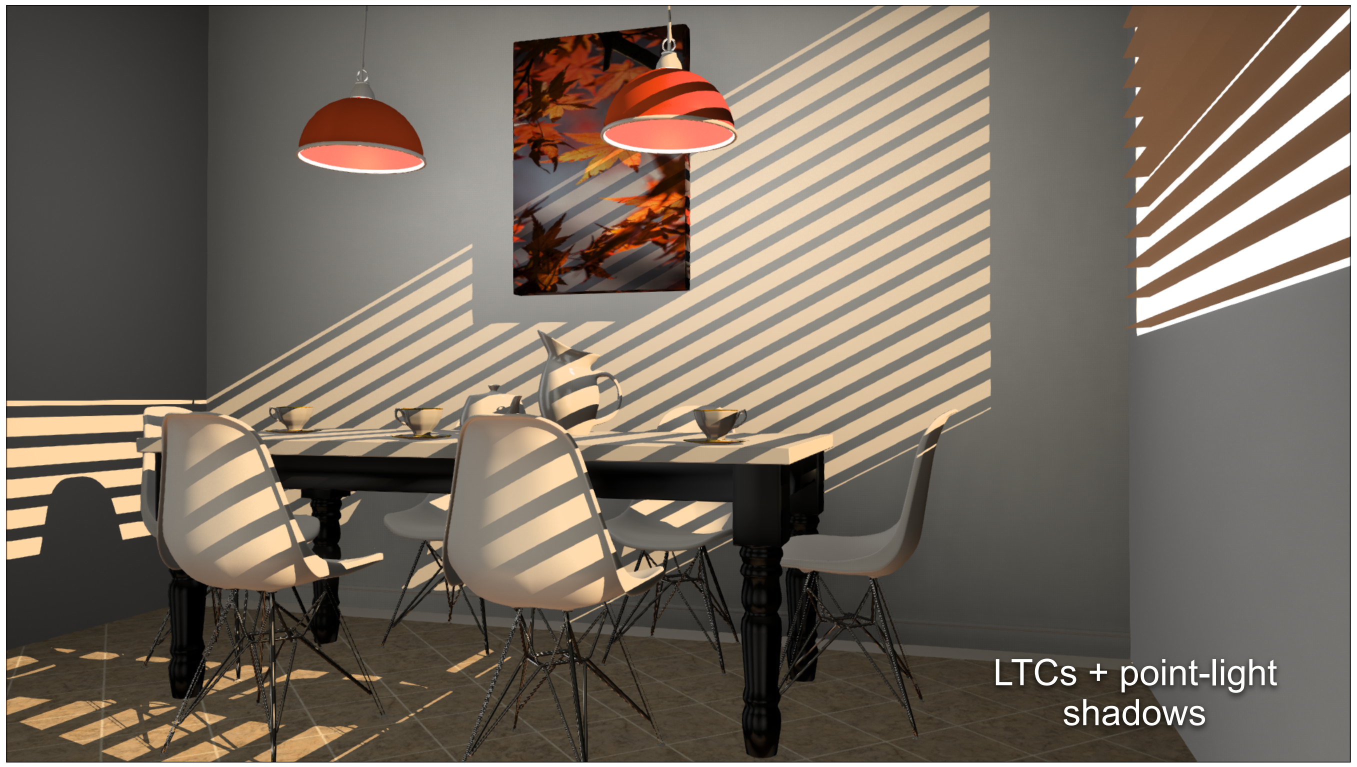
So, how does it look?


raw stochastic (S_N)

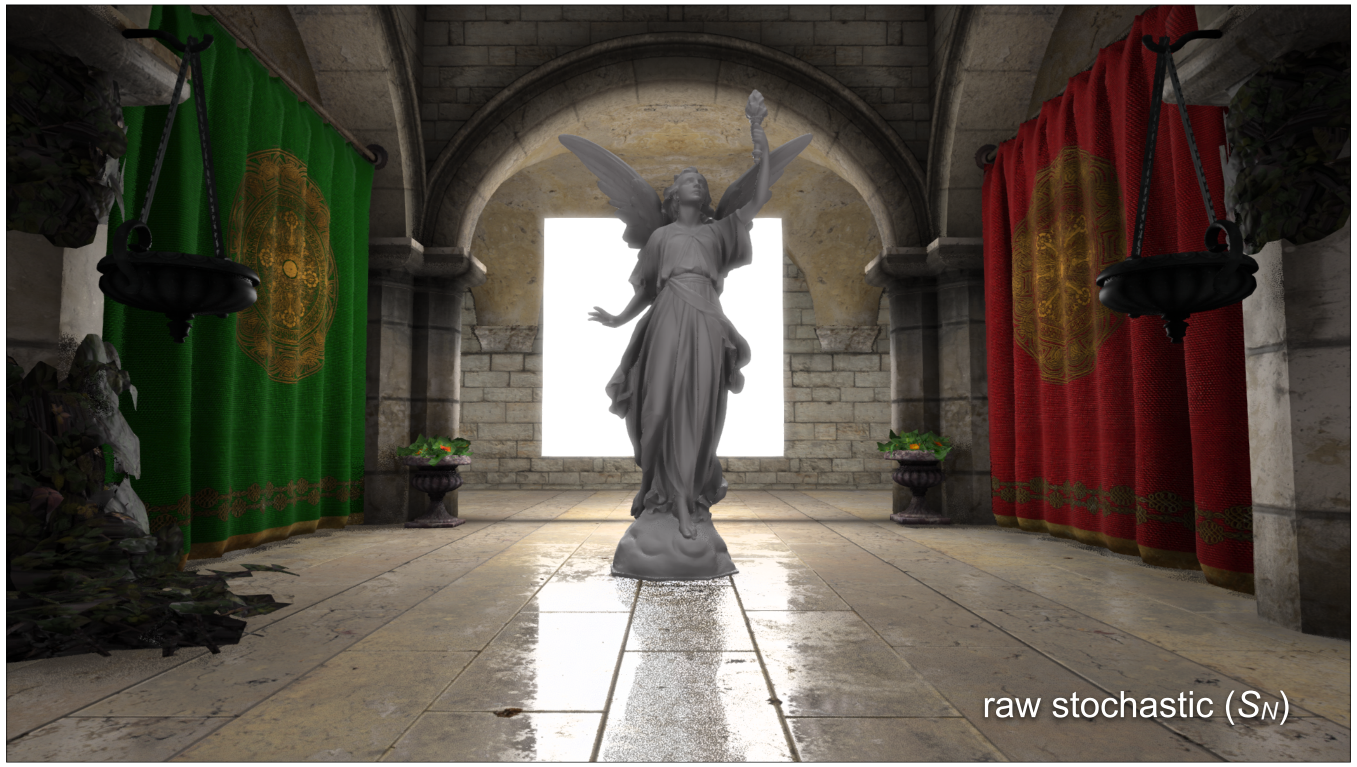

offline reference
(256 rays/pixel)

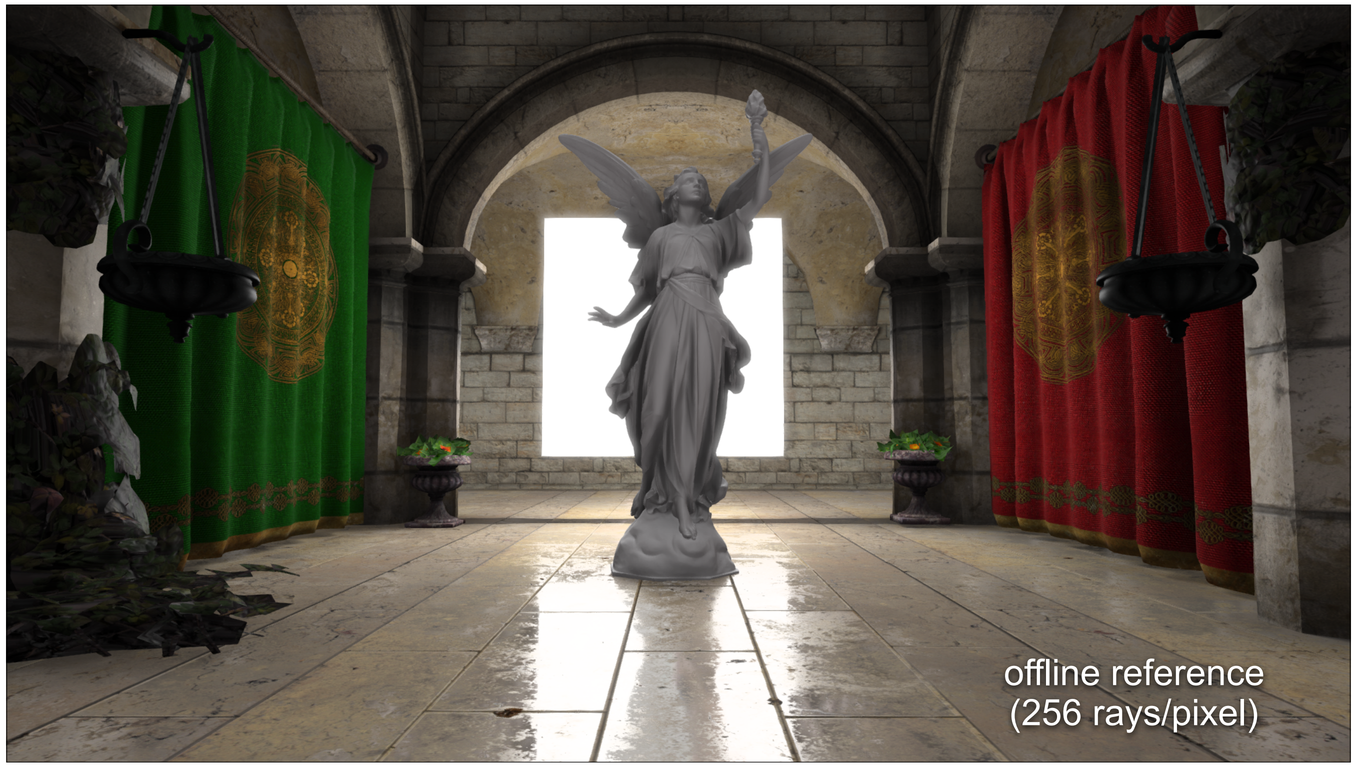


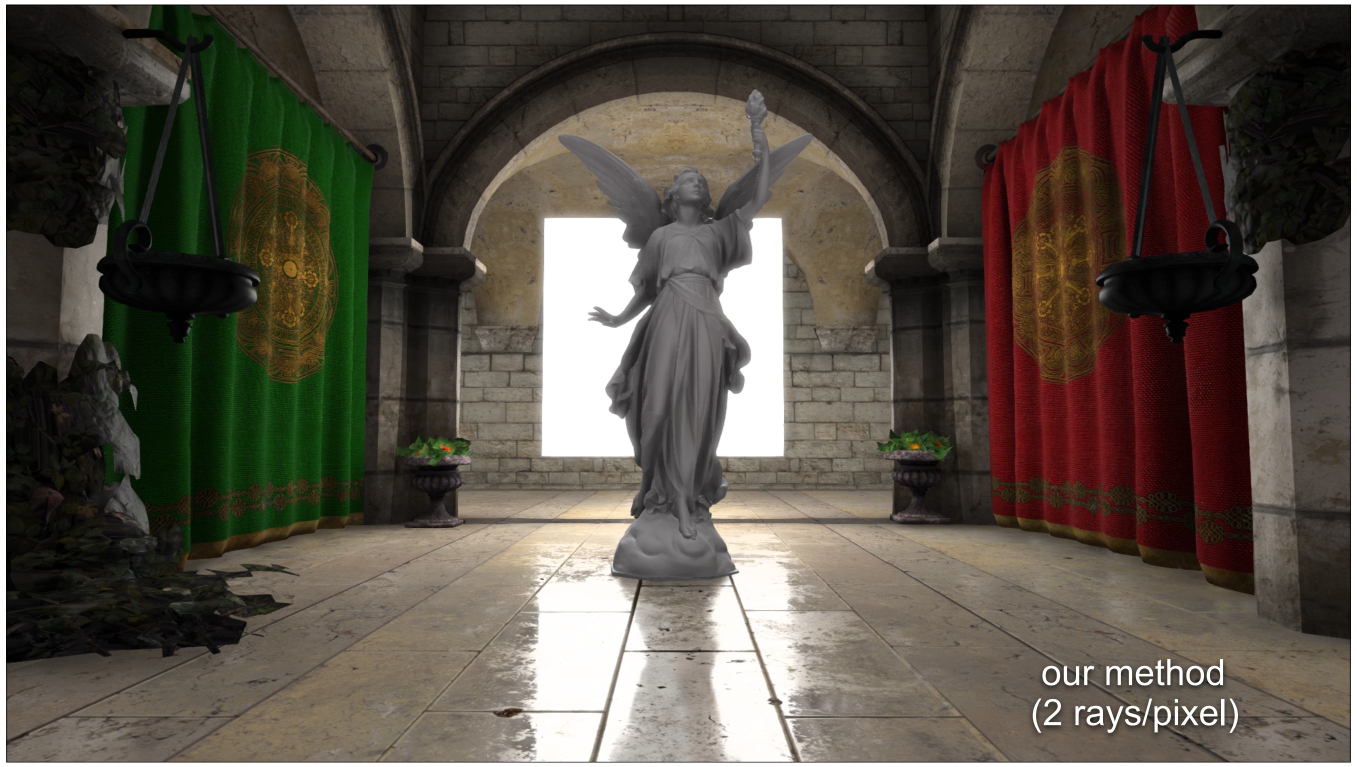
Using 254 fewer rays at each pixel!

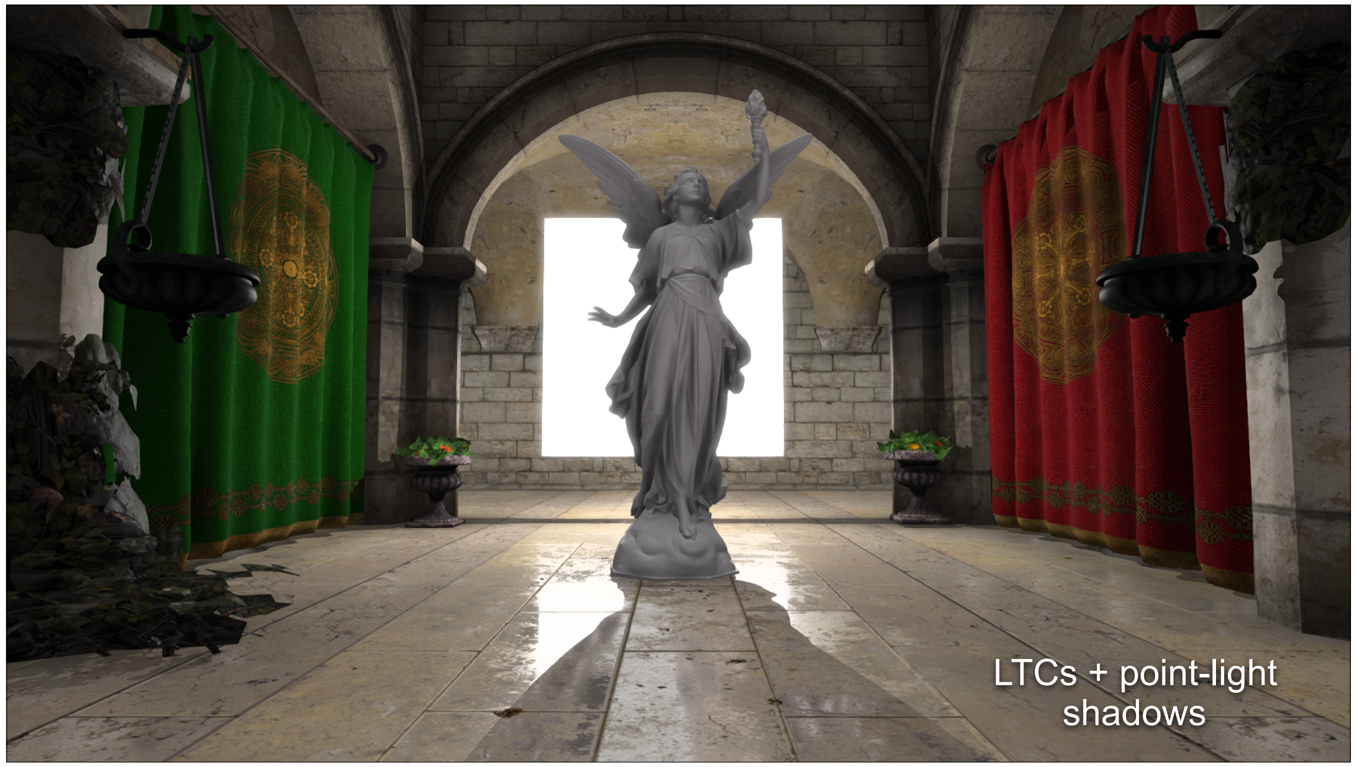


Point shadows lose all of the distance softening and “wrap lighting” produced naturally by an area light.

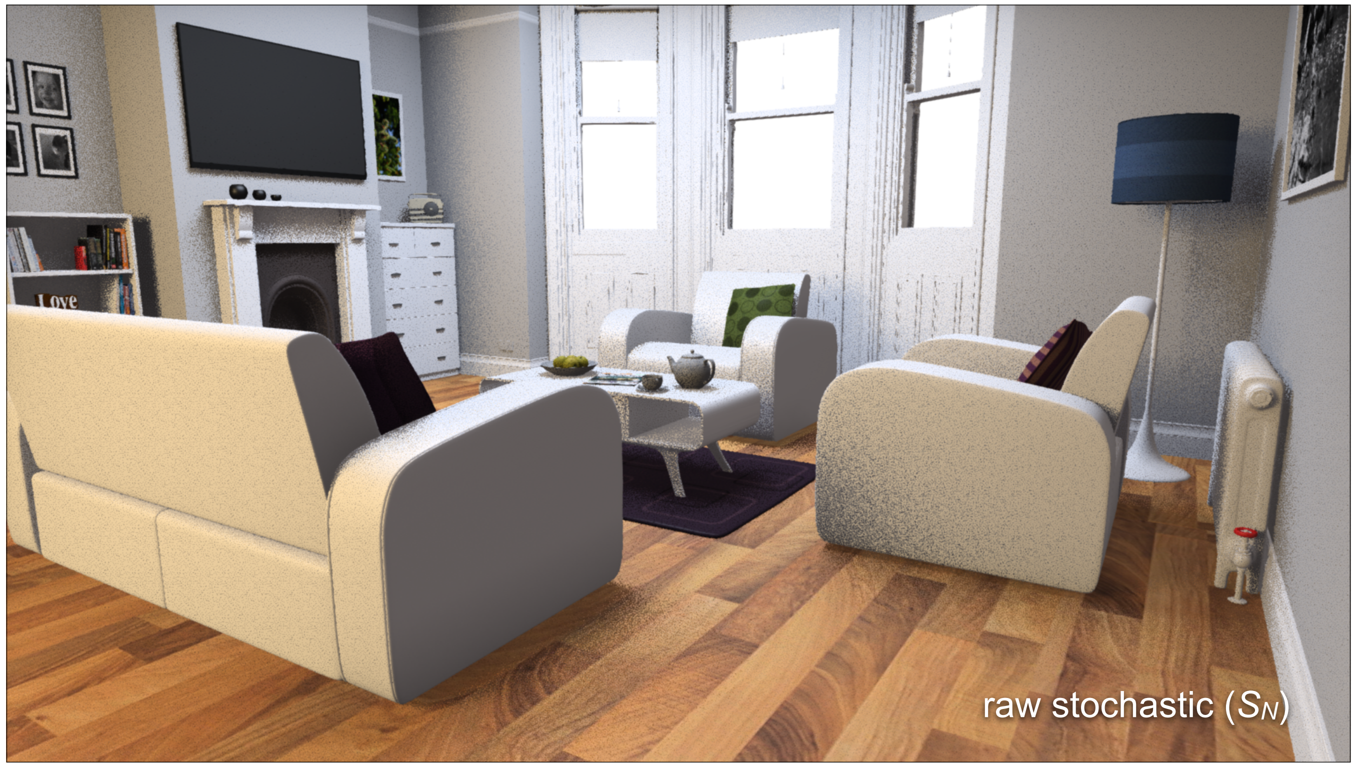

raw stochastic (S_N)


offline reference
(256 rays/pixel)

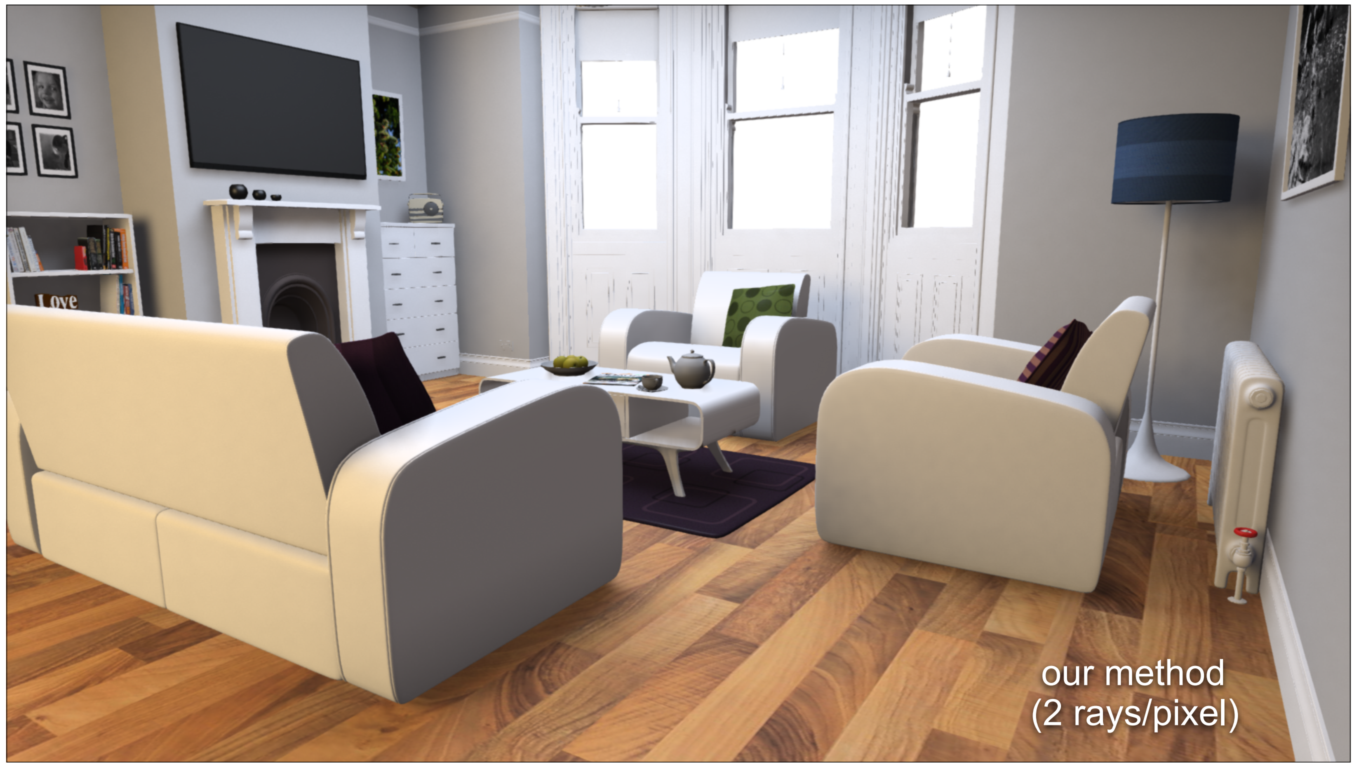

our method
(2 rays/pixel)



Shadow map point shadows just look ridiculous here on an area light. They both under and over shadow.

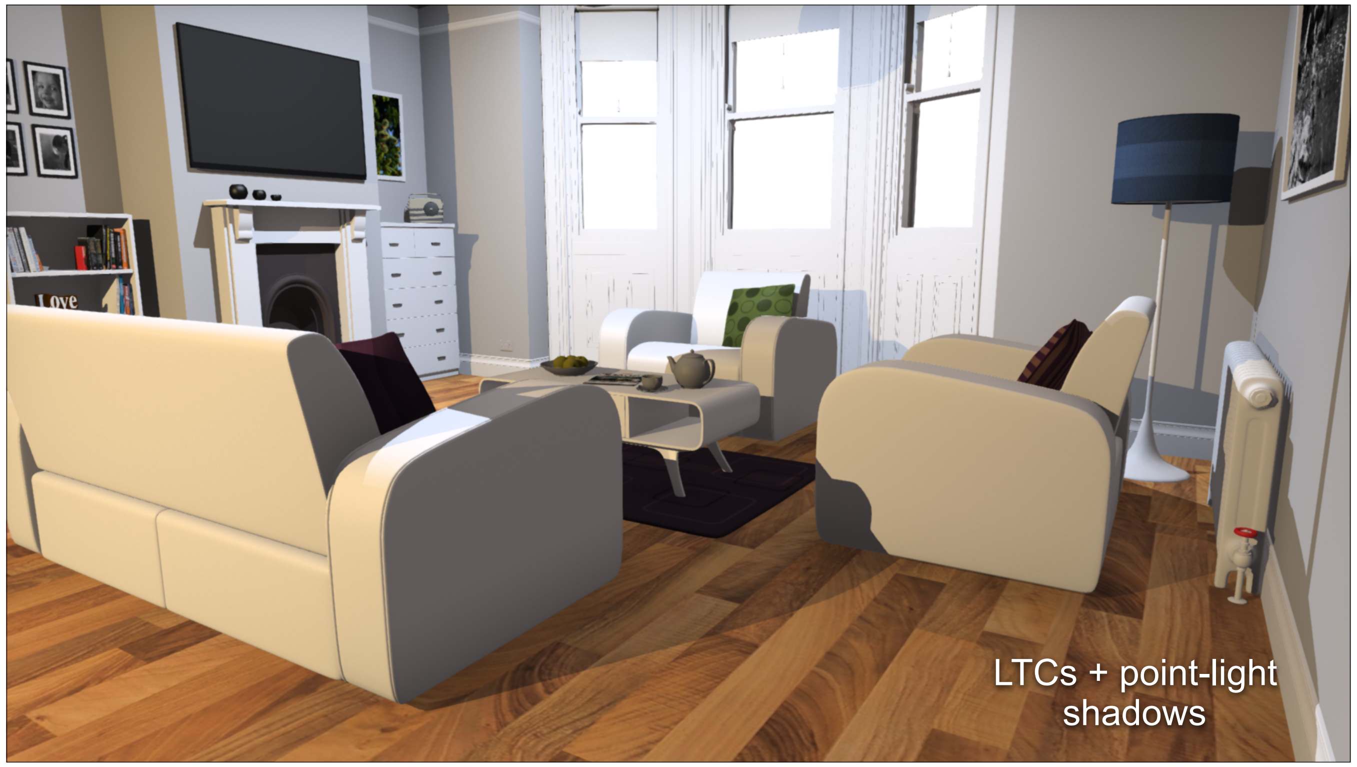

raw stochastic (S_N)
Love

Love
offline reference
(256 rays/pixel)



This looks like a path traced image, but in fact there’s no global illumination or ambient occlusion here, just direct illumination from two area lights. So, you can get a lot of coverage and a natural look from properly shadowed area lights.

We have one yellow area light off to the left, and the entire ceiling is a grey area light representing the bounce light. All of the ‘ambient occlusion’ is just the naturally occurring soft shadow from that ceiling light, and the very soft falloff of shadows and self-shadowing due to huge area sources is what creates the ‘global’ look. This is very art directable and very natural.

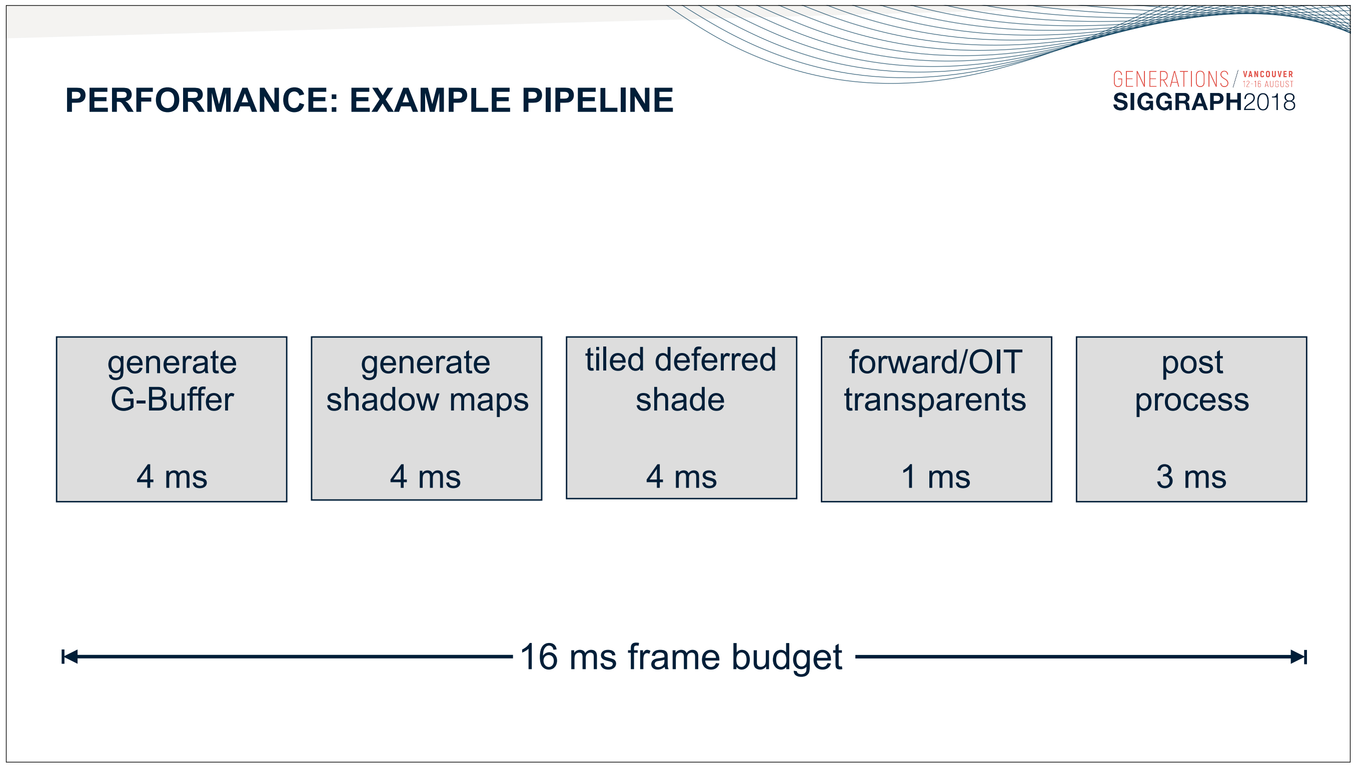


With point light shadows you get none of that natural look. It looks like ‘computer graphics’. :)

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

PERFORMANCE

To get a context for lighting times…

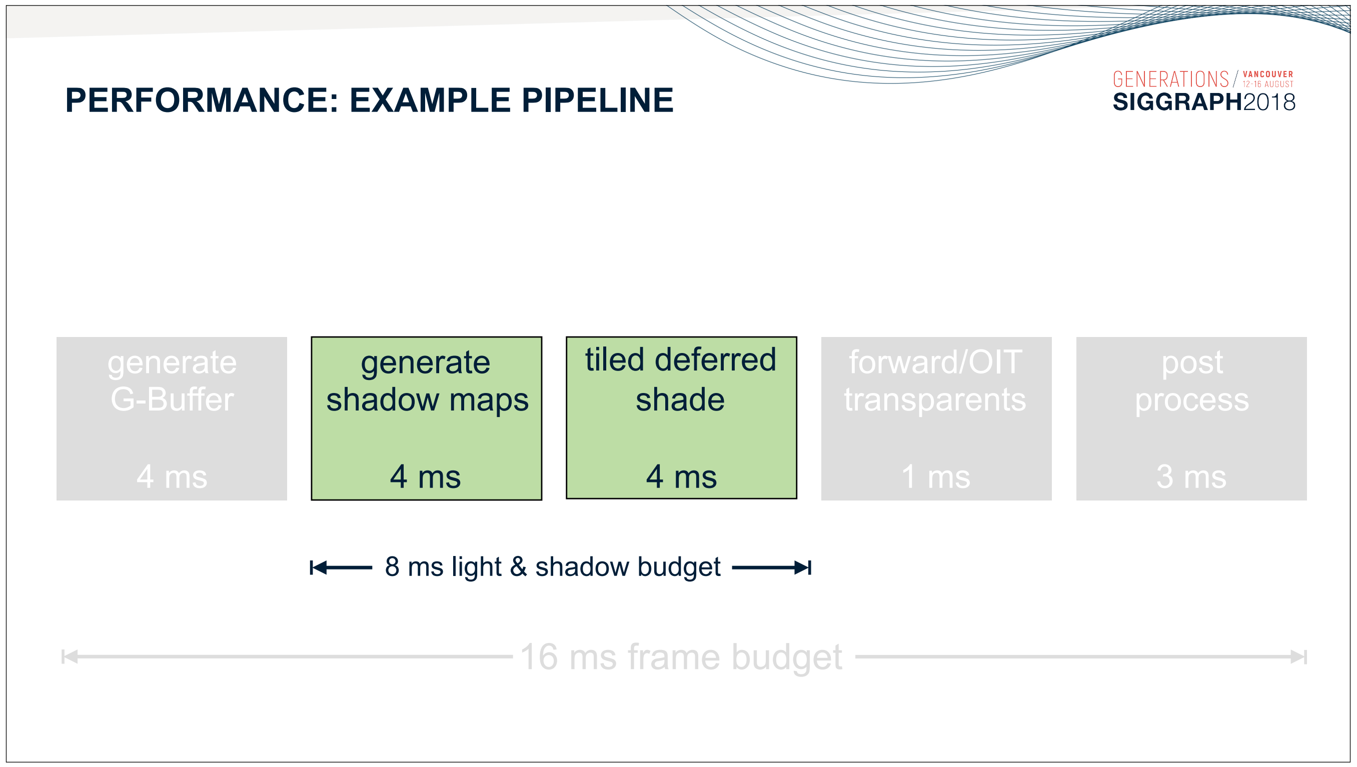


PERFORMANCE: EXAMPLE PIPELINE

generate G-Buffer 4 ms → generate shadow maps 4 ms → tiled deferred shade 4 ms → forward/OIT transparents 1 ms → post process 3 ms

16 ms frame budget

Here’s a straw person 60 fps pipeline for deferred shading. Each stage’s timing will vary based on the target platform and kind of content. For simplicity, I’m glossing over some things like AO passes, but I think we can agree that this is in the ballpark.



PERFORMANCE: EXAMPLE PIPELINE

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

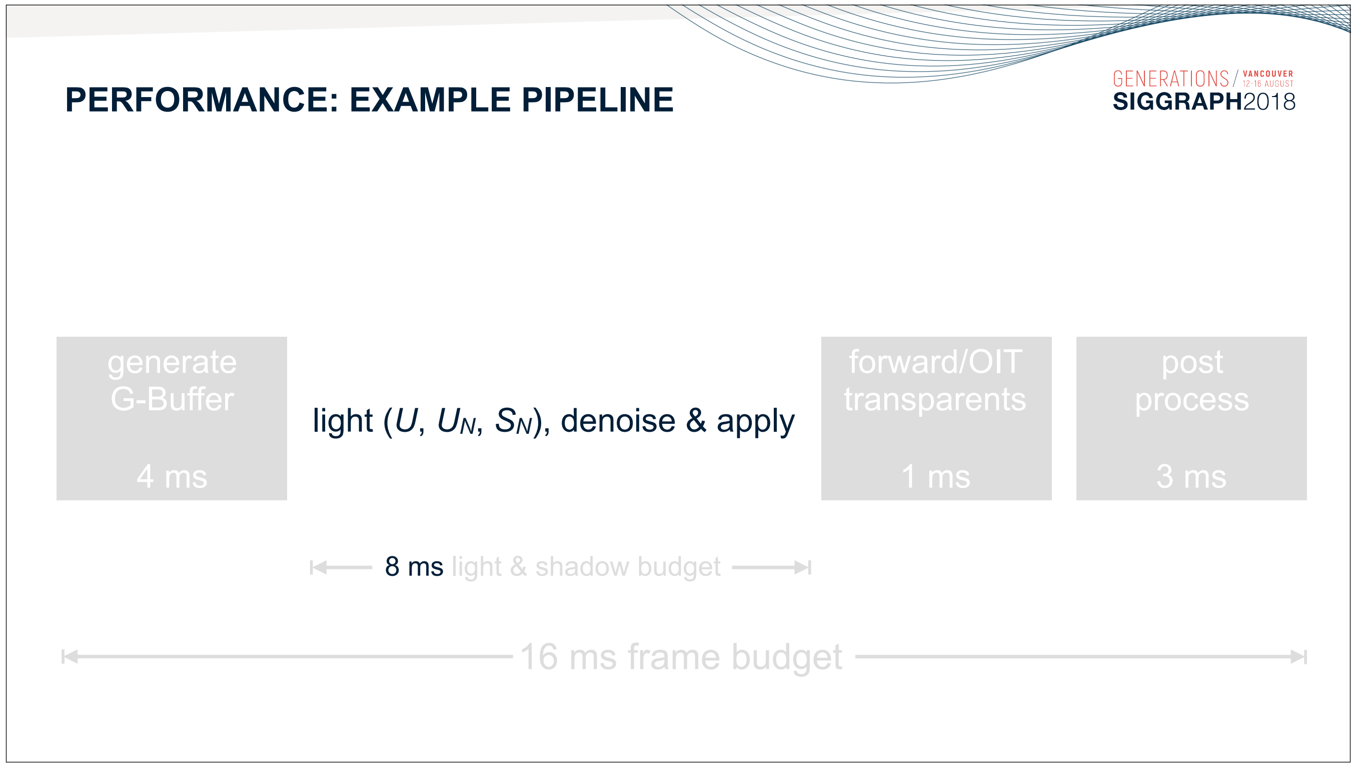
generate G-Buffer	generate shadow maps	tiled deferred shade	forward/OIT transparents	post process
4 ms	4 ms	4 ms	1 ms	3 ms

8 ms light & shadow budget

16 ms frame budget

You obviously don't need shadow maps if you're ray tracing the shadows, so our method immediately gives you back the 4 ms from the shadow map pass.

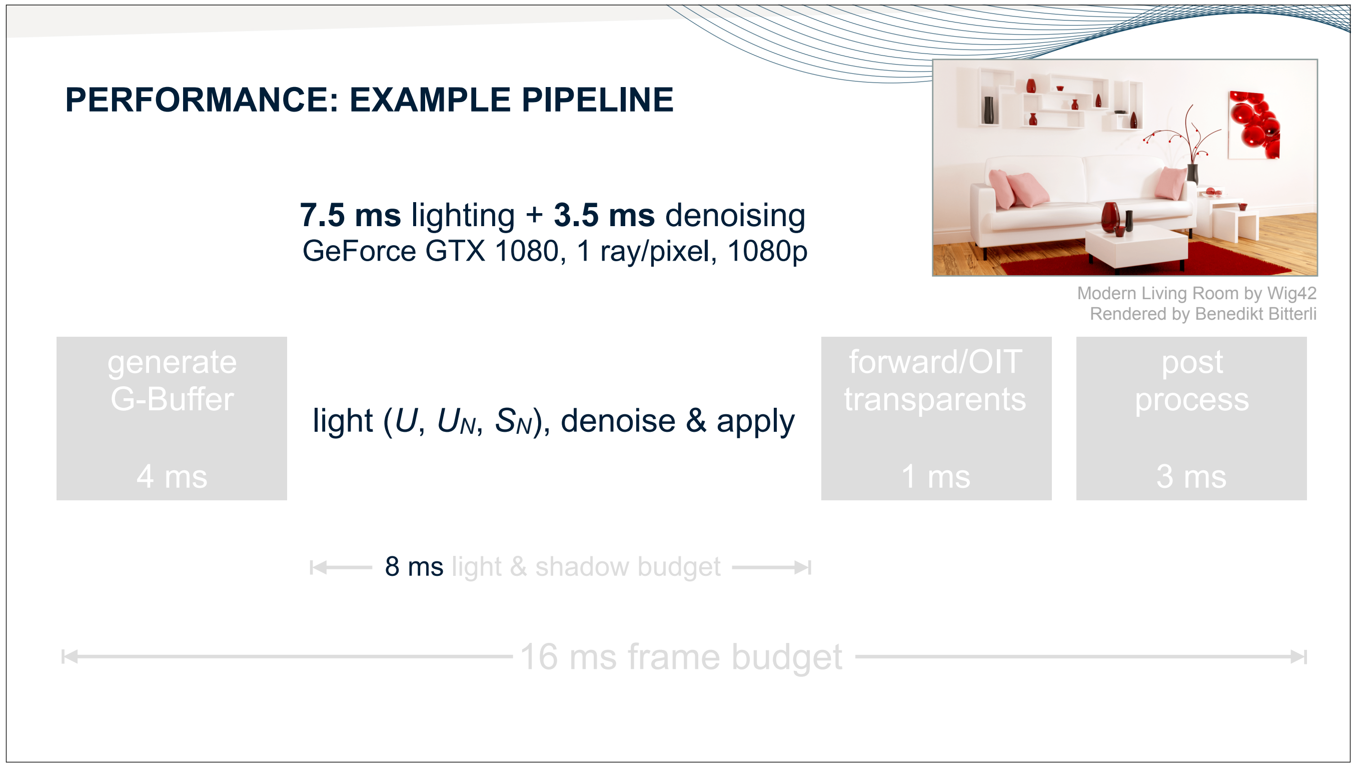
But it isn't just shadowing: we're computing all of the shading in our ratio estimator generation. So we recover the time of the entire tiled deferred shading pass as well. Combined, those are about half the frame time.



PERFORMANCE: EXAMPLE PIPELINE

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018
generate G-Buffer
4 ms
light (U, U_N, S_N), denoise & apply
forward/OIT transparents
1 ms
post process
3 ms
8 ms light & shadow budget
16 ms frame budget

Two punchlines: 1) at 60 fps you don't get the full 16.7ms for lighting and shadowing; you get maybe half of that, and 2) because this method does both, you could spend that entire 8-9 ms on area lights (at 1080p, at 60 Hz; VR or 4K rendering obviously scales your budget down).



PERFORMANCE: EXAMPLE PIPELINE

7.5 ms lighting + **3.5 ms** denoising
GeForce GTX 1080, 1 ray/pixel, 1080p

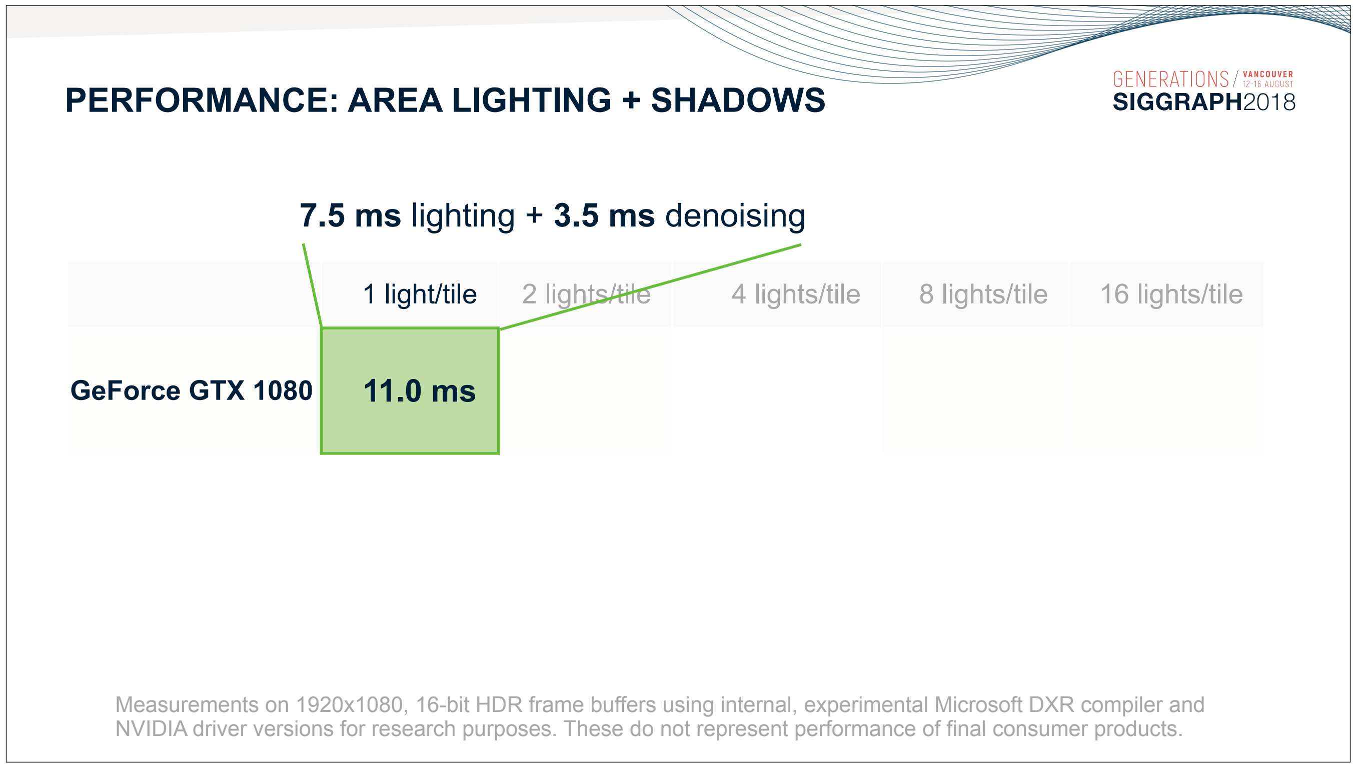
Modern Living Room by Wig42
Rendered by Benedikt Bitterli

Well, for a highly tessellated scene and a large 1m^2 light, this costs about 7.5 ms for the light computation using one ray per pixel in a DXR shader and about 3.5 ms for the denoising on a GeForce GTX 1080.

It is no surprise that we don't meet the 8 ms lighting budget when ray tracing on a Pascal GPUs, but going over budget by only 3 ms in order to turn point-rasterised point lights into correct ray-traced area lights is actually pretty good.

My real concern here is that you *can* distribute the rays across lights, but we really want one ray, per pixel, per light in the tile for good coverage. Area lights have a large visual impact, so you don't need quite as many as you do for point lights, but I still want to see 2-4 area lights per tile, not just the one light in 8 ms shown here.

Let's try scaling this up to that point…



PERFORMANCE: AREA LIGHTING + SHADOWS

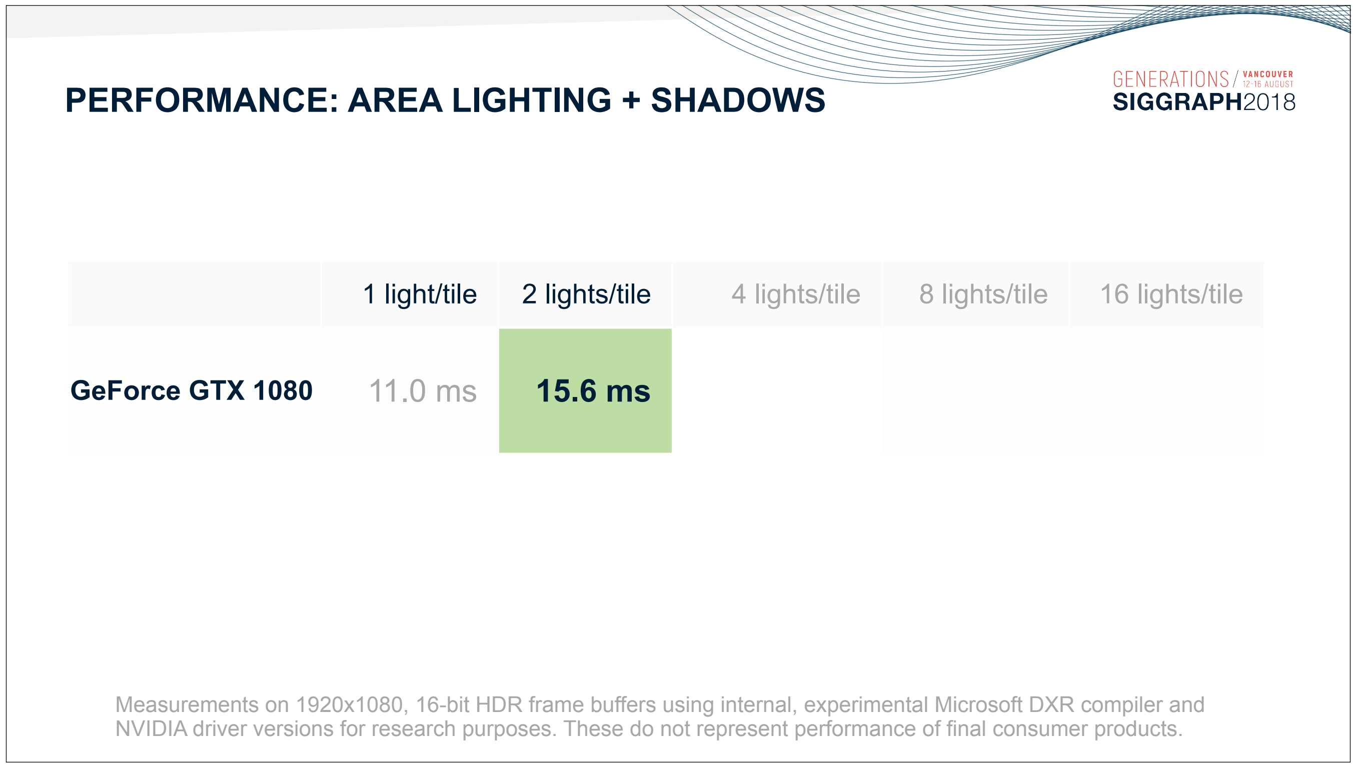
GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

7.5 ms lighting + **3.5 ms** denoising

	1 light/tile	2 lights/tile	4 lights/tile	8 lights/tile	16 lights/tile
GeForce GTX 1080	**11.0 ms**				

Measurements on 1920x1080, 16-bit HDR frame buffers using internal, experimental Microsoft DXR compiler and NVIDIA driver versions for research purposes. These do not represent performance of final consumer products.

Here's the 11.0 ms for one ray. If we spend one ray per light, then rays/pixel is lights/tile, so I'll label the table with that.



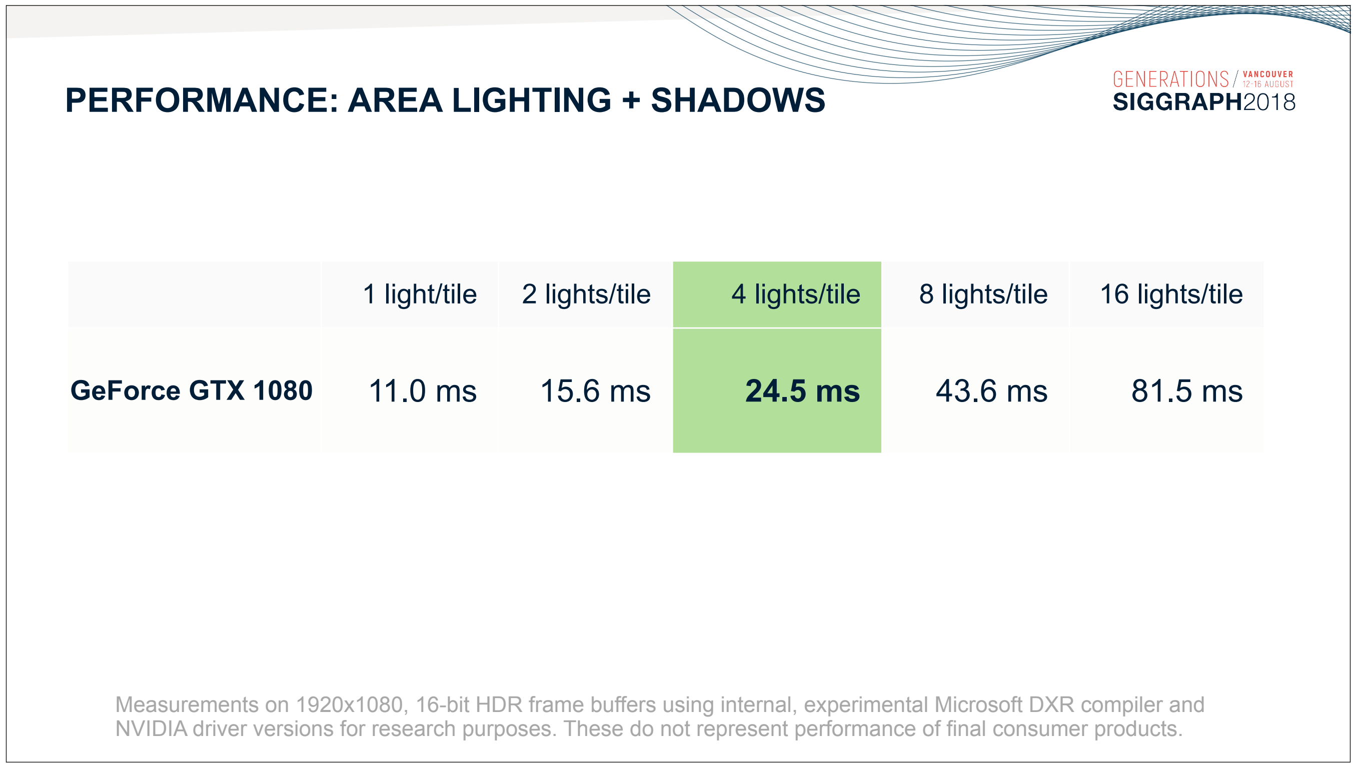
PERFORMANCE: AREA LIGHTING + SHADOWS

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

	1 light/tile	2 lights/tile	4 lights/tile	8 lights/tile	16 lights/tile
GeForce GTX 1080	11.0 ms	**15.6 ms**			

Measurements on 1920x1080, 16-bit HDR frame buffers using internal, experimental Microsoft DXR compiler and NVIDIA driver versions for research purposes. These do not represent performance of final consumer products.

At 2 lights/tile we measured about 15 and a half ms…



PERFORMANCE: AREA LIGHTING + SHADOWS

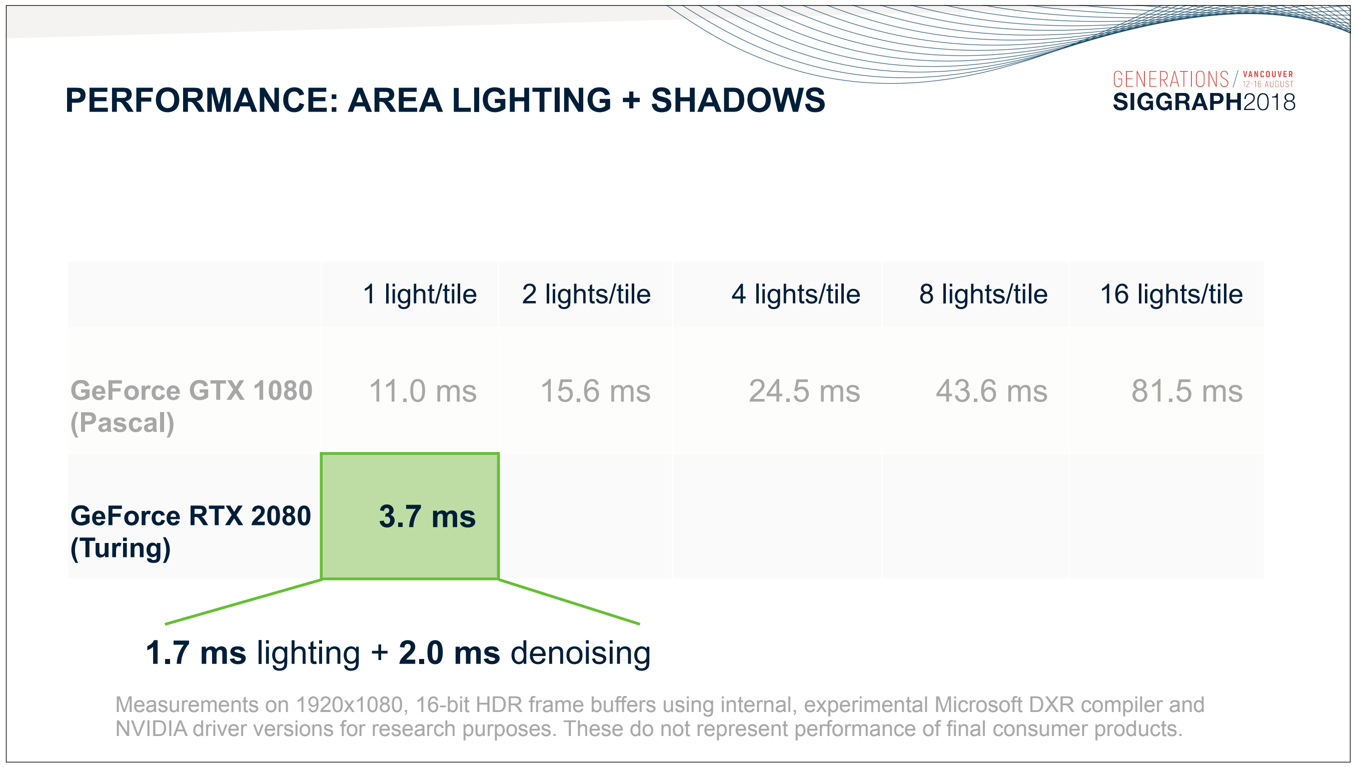
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

	1 light/tile	2 lights/tile	4 lights/tile	8 lights/tile	16 lights/tile
GeForce GTX 1080	11.0 ms	15.6 ms	**24.5 ms**	43.6 ms	81.5 ms

Measurements on 1920x1080, 16-bit HDR frame buffers using internal, experimental Microsoft DXR compiler and NVIDIA driver versions for research purposes. These do not represent performance of final consumer products.

…24.5 ms at our target of 4 lights per tile, and then proportionally higher costs for ridiculous numbers of area lights.

At four lights per tile, I wanted to hit 8 ms, and we're at 25 ms today. That's not deployable on Pascal, but this kind of limitation was a big motivation for an architecture like NVIDIA's Turing chips. We have an algorithm that is bottlenecked on ray tracing (ray casting, really), so if we move ray tracing into dedicated, asynchronous cores then our method becomes practical, and maybe faster than rasterising and filtering giant shadow maps.



PERFORMANCE: AREA LIGHTING + SHADOWS

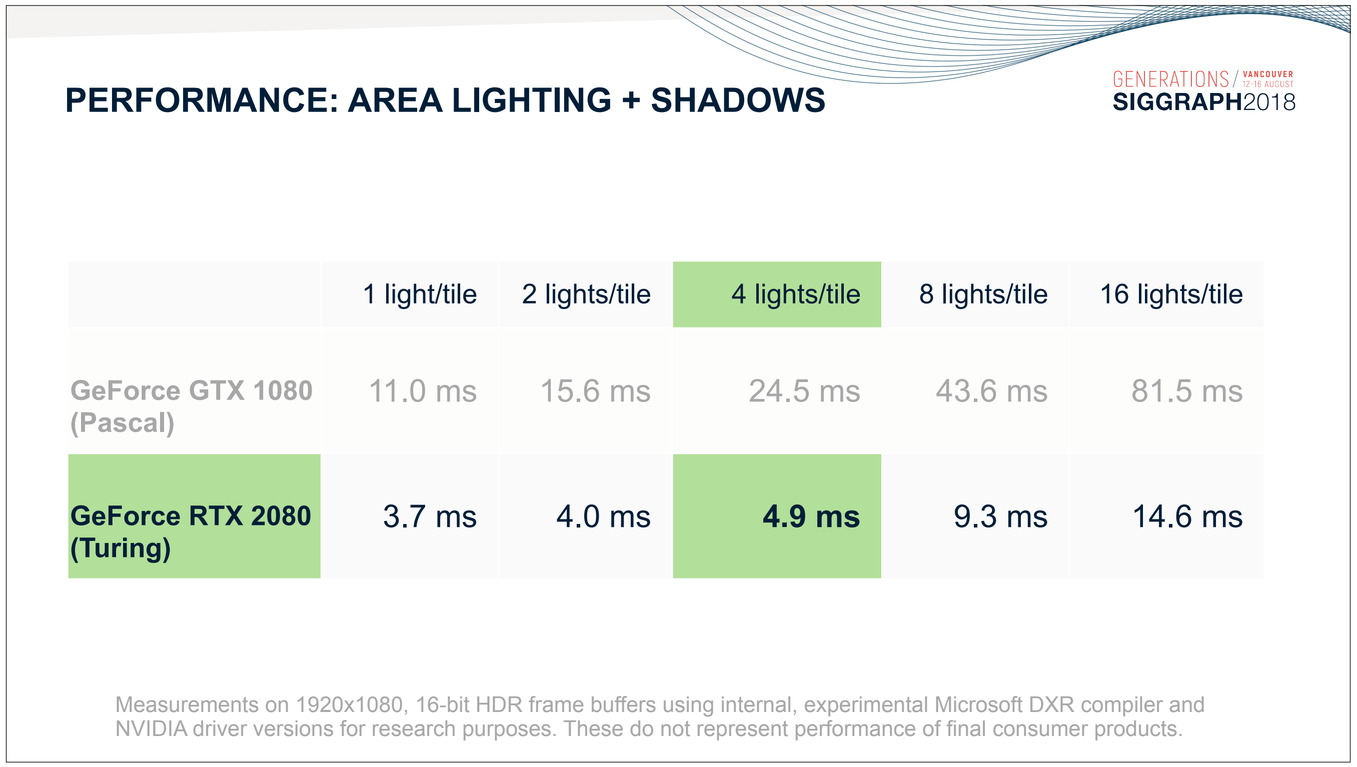
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

	1 light/tile	2 lights/tile	4 lights/tile	8 lights/tile	16 lights/tile
GeForce GTX 1080 (Pascal)	11.0 ms	15.6 ms	24.5 ms	43.6 ms	81.5 ms
GeForce RTX 2080 (Turing)	**3.7 ms**				

1.7 ms lighting + **2.0 ms** denoising

Measurements on 1920x1080, 16-bit HDR frame buffers using internal, experimental Microsoft DXR compiler and NVIDIA driver versions for research purposes. These do not represent performance of final consumer products.

On the new NVIDIA GeForce RTX 2080 GPU, at one light per tile, we see 1.7 ms for the lighting pass and 2.0 ms for the denoising. That’s a 4x speedup for our algorithm.



PERFORMANCE: AREA LIGHTING + SHADOWS

GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

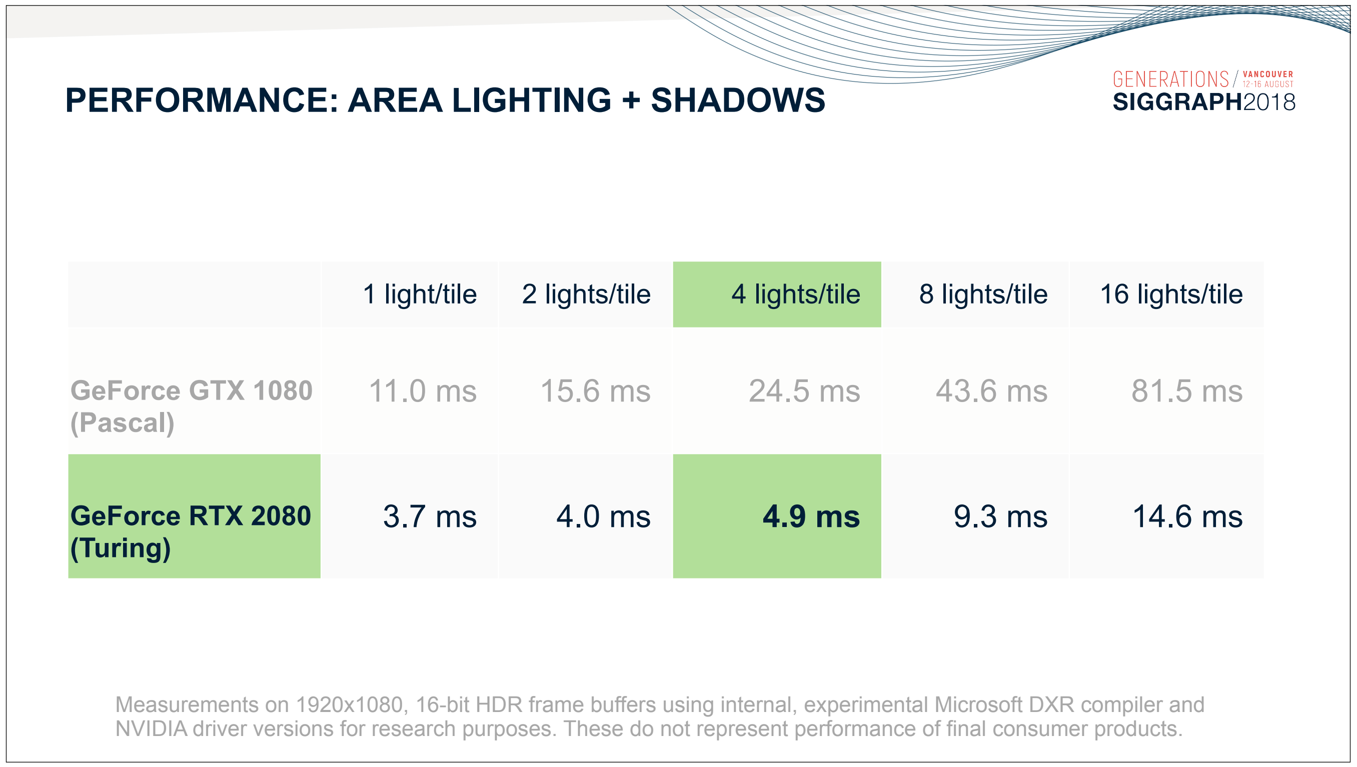
	1 light/tile	2 lights/tile	4 lights/tile	8 lights/tile	16 lights/tile
GeForce GTX 1080 (Pascal)	11.0 ms	15.6 ms	24.5 ms	43.6 ms	81.5 ms
GeForce RTX 2080 (Turing)	3.7 ms	4.0 ms	**4.9 ms**	9.3 ms	14.6 ms

Measurements on 1920x1080, 16-bit HDR frame buffers using internal, experimental Microsoft DXR compiler and NVIDIA driver versions for research purposes. These do not represent performance of final consumer products.

At 2 lights per tile, it takes 4 ms total. At the critical 4 lights per tile, where I wanted to beat 8 ms “some day”: we’re at 4.9 ms, well under our hypothetical 8 ms budget! And then 9 and 15 ms for higher light counts.

I measured this on Turing using DXR, but this isn’t about NVIDIA’s GPUs or Microsoft’s APIs in isolation. The whole industry is going in the same direction. I believe every API, engine and processor is going to support ray tracing soon.

So, give me one more minute to relate some observations about how you might optimise algorithms like this for a world of real-time ray tracing, and then we’ll wrap up the whole session.



PERFORMANCE: AREA LIGHTING + SHADOWS

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

	1 light/tile	2 lights/tile	4 lights/tile	8 lights/tile	16 lights/tile
GeForce GTX 1080 (Pascal)	11.0 ms	15.6 ms	24.5 ms	43.6 ms	81.5 ms
GeForce RTX 2080 (Turing)	3.7 ms	4.0 ms	**4.9 ms**	9.3 ms	14.6 ms

Measurements on 1920x1080, 16-bit HDR frame buffers using internal, experimental Microsoft DXR compiler and NVIDIA driver versions for research purposes. These do not represent performance of final consumer products.

It is early days for dedicated HW-accelerated ray tracing, but we've already seen this:

Ray tracing is a high-latency operation that runs on a separate core. Think of it like a really expensive texture fetch. The same kinds of optimisations apply.

You need enough work to cover that latency, so that you can fill the machine and have the SM (CUDA core) do other processing on other threads while waiting for the ray trace to return.

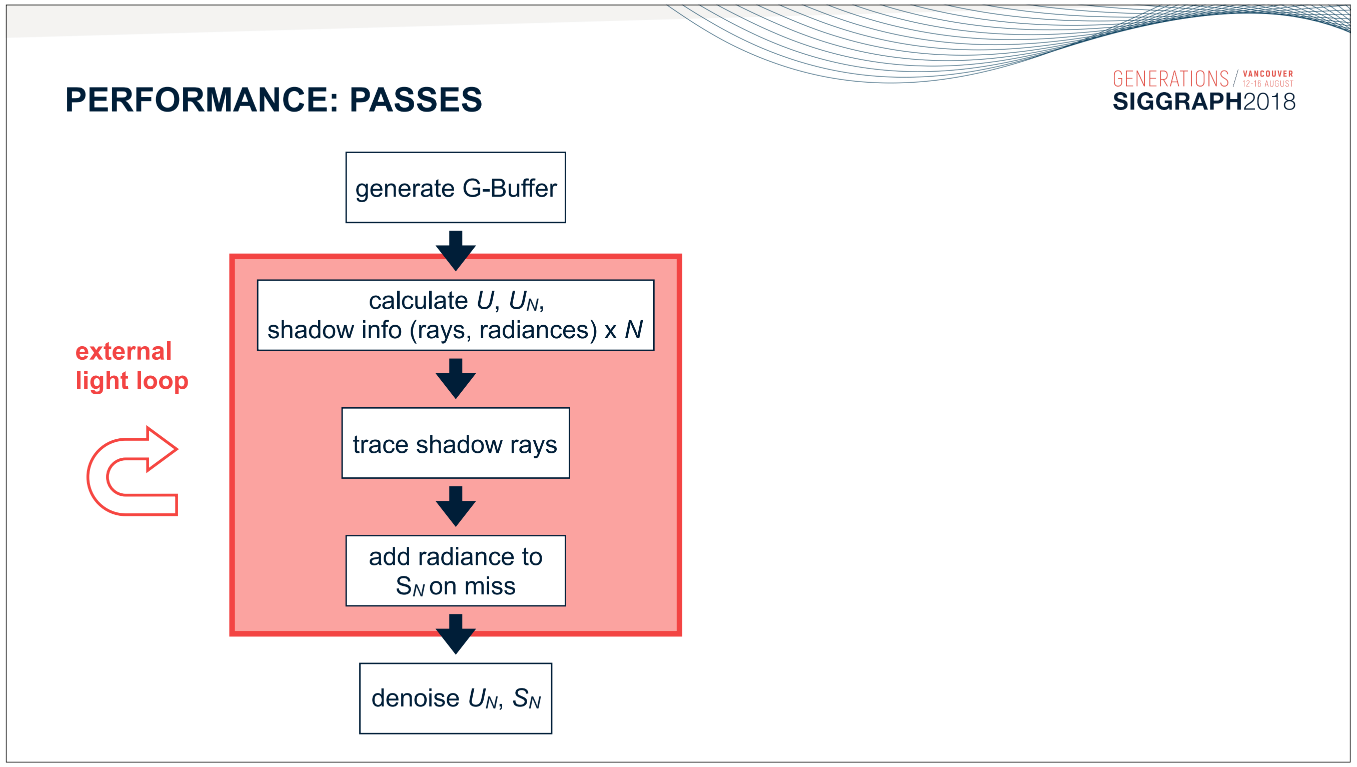
A key consideration is the live state of your program: you want a low register count when you hit that ray trace call in order to get a lot of threads in flight simultaneously, and the trace is going to need some of the registers for itself.

GDDR6 gives you a lot more bandwidth, but you're sharing that with the ray-tracing cores. So, as always, you must balance compute, bandwidth and registers.

4 lights per tile is a scalability sweet spot for our current implementation because the unrolled shader loop balances the fixed read/write cost, register file size and instruction cache for that constant.

Multiple importance sampling is a source of divergence in a program. You have to structure it carefully to keep your threads in sync across a warp even when they're sampling different PDFs. MIS also forces a lot of normalisation terms, which means division operations. On Turing there's a lot of arithmetic throughput, but that still needs to be hidden behind the long memory and ray-trace operations.

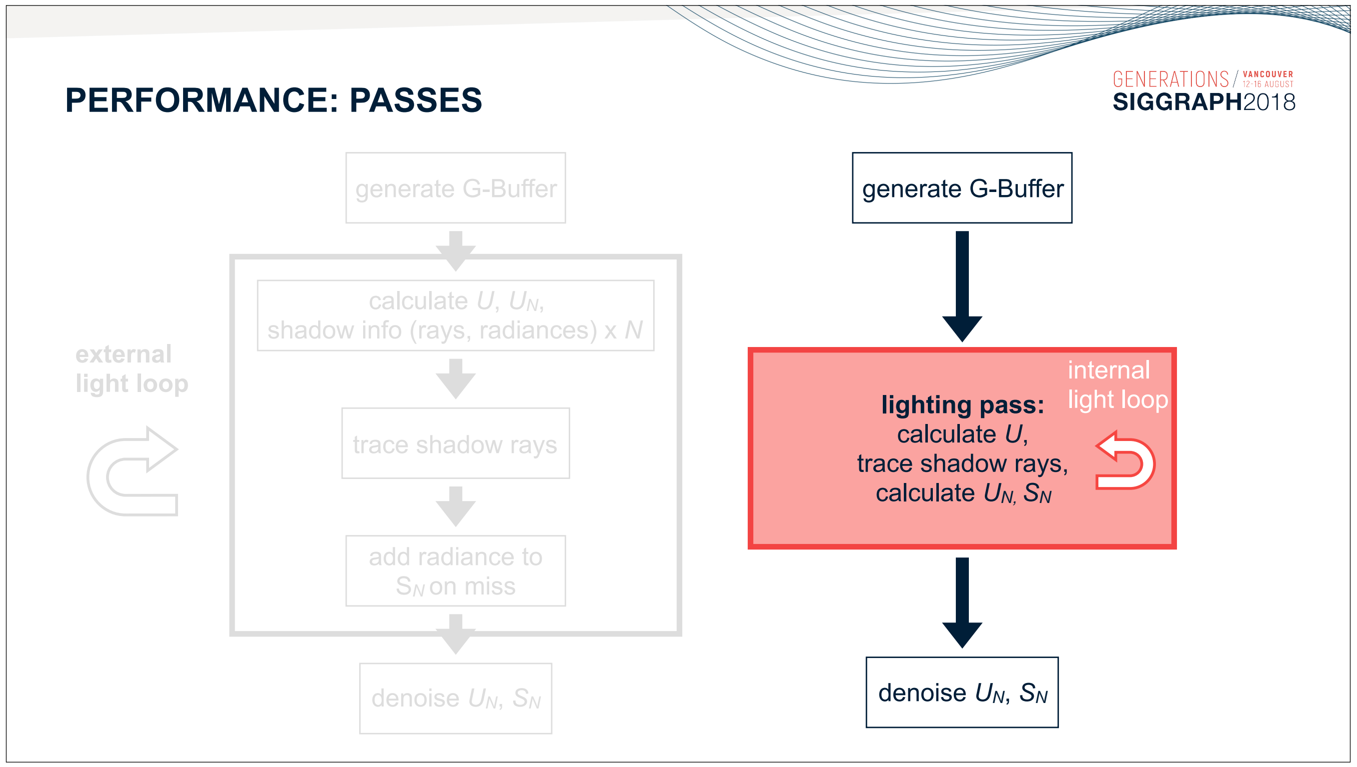
PERFORMANCE: PASSES


GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018
generate G-Buffer
calculate U, U_N, shadow info (rays, radiances) x N
trace shadow rays
add radiance to S_N on miss
denoise U_N, S_N
external light loop

Collectively, these issues mean that the most significant design choice for any ray-tracing kernel is wavefront vs. inline shading.

A wavefront design makes one pass per light. It explicitly rolls all state out to textures, casts *wavefronts* of rays for all pixels at once, and then reads state back in to shade. We did this in our initial I3D implementation, which is online.

It minimises live state to get good occupancy so that we can fully use the ray tracing cores, but it heavily magnifies the G-Buffer and shading buffer bandwidth cost.



PERFORMANCE: PASSES

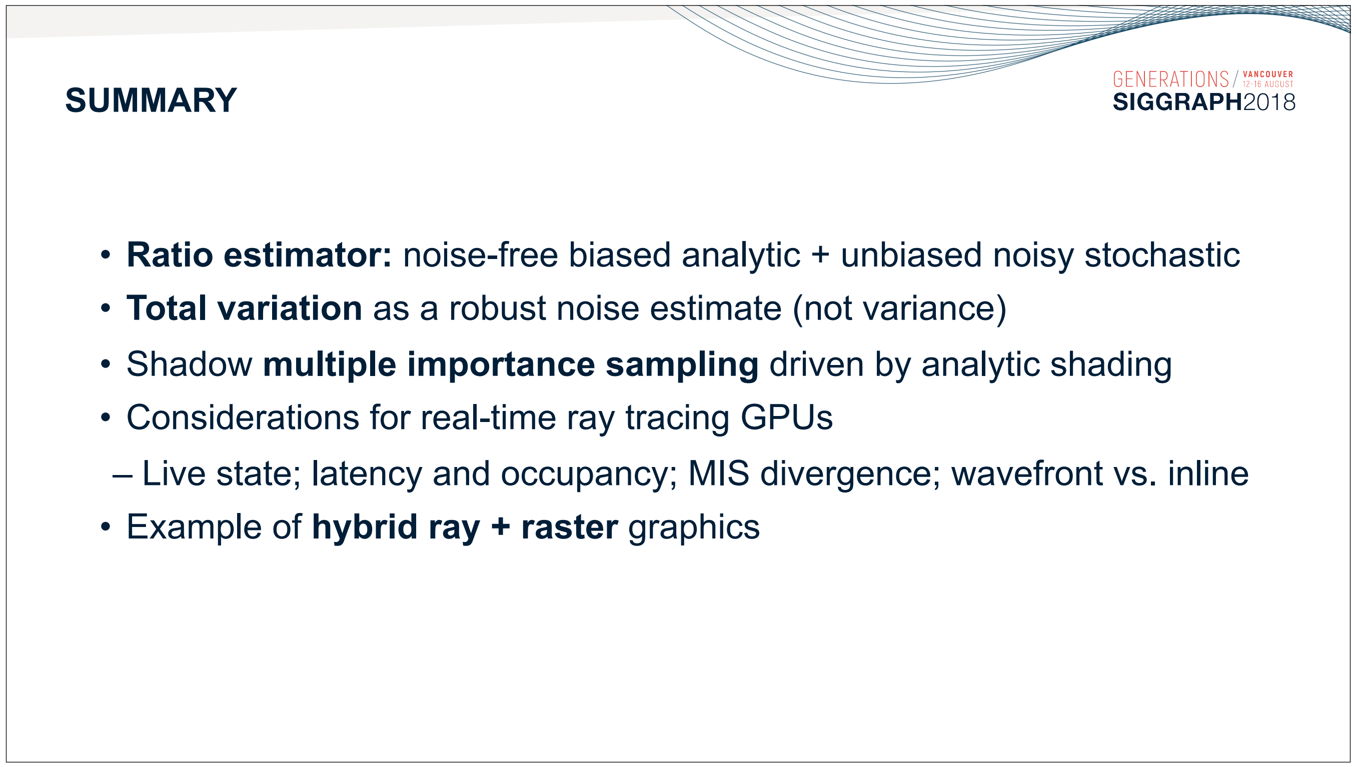
GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018
generate G-Buffer
calculate U, U_N, shadow info (rays, radiances) x N
external light loop
trace shadow rays
add radiance to S_N on miss
denoise U_N, S_N
generate G-Buffer
lighting pass: calculate U, trace shadow rays, calculate U_N, S_N
internal light loop
denoise U_N, S_N

The inline method makes a single draw call that has a light loop in the shader, which performs both ray casting and shading. Our new SIGGRAPH implementation follows this approach.

It conserves memory bandwidth in the main shader so that the ray-tracing cores can use it, but the larger live state means more register pressure, so the optimisation challenge was conserving registers to getting enough occupancy to put all of those rays in flight simultaneously.

We're still working with both approaches and will report when we have a strong recommendation of which is favoured for the consumer Turing GPUs.

SUMMARY



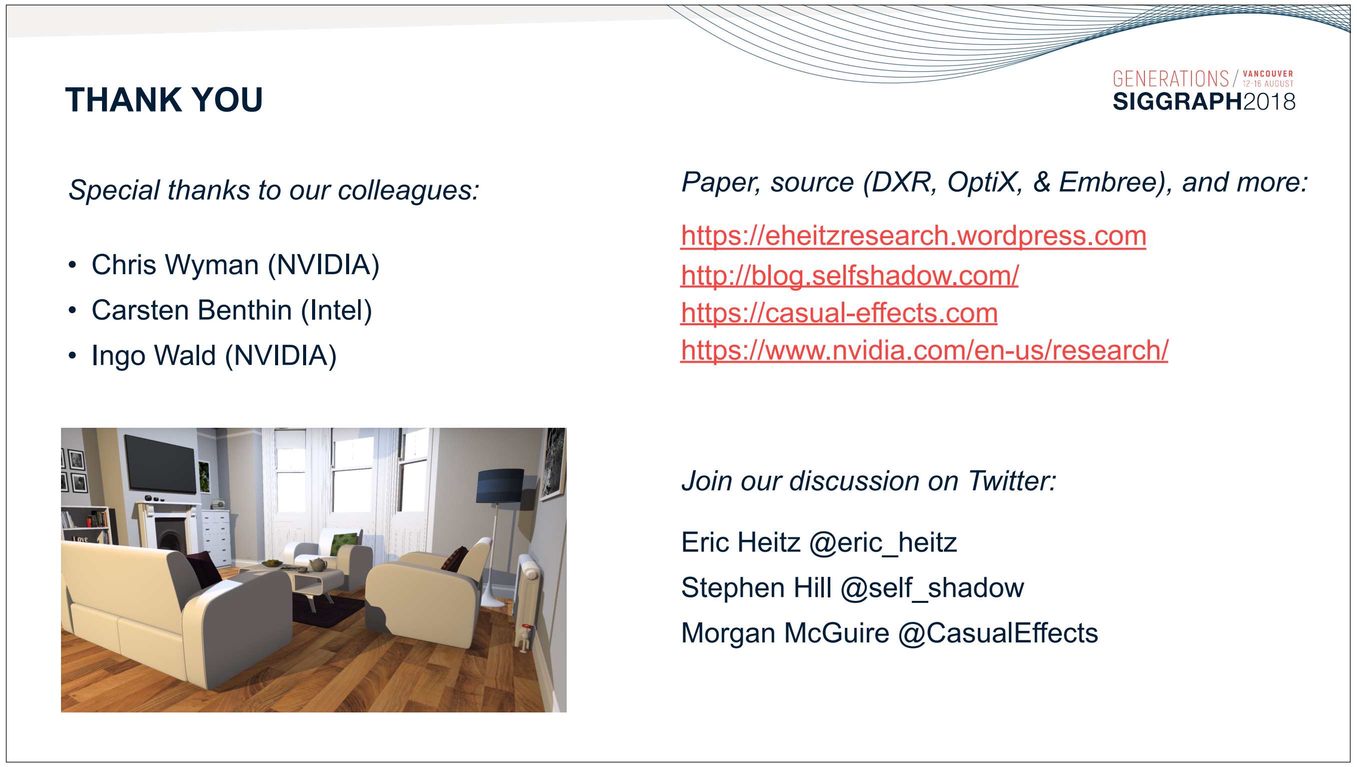
SUMMARY

GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

- **Ratio estimator:** noise-free biased analytic + unbiased noisy stochastic
- **Total variation** as a robust noise estimate (not variance)
- Shadow **multiple importance sampling** driven by analytic shading
- Considerations for real-time ray tracing GPUs
  – Live state; latency and occupancy; MIS divergence; wavefront vs. inline
- Example of **hybrid ray + raster** graphics

In summary, we presented the following…

We hope that these elements are useful for other, similar problems, and are looking at participating media, subsurface scattering, global illumination and ambient occlusion next.



THANK YOU

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018

Special thanks to our colleagues:

- Chris Wyman (NVIDIA)
- Carsten Benthin (Intel)
- Ingo Wald (NVIDIA)

Paper, source (DXR, OptiX, & Embree), and more:

https://eheitzresearch.wordpress.com
http://blog.selfshadow.com/
https://casual-effects.com
https://www.nvidia.com/en-us/research/

Join our discussion on Twitter:

Eric Heitz @eric_heitz
Stephen Hill @self_shadow
Morgan McGuire @CasualEffects

Finally, I'd like to acknowledge our colleagues at Intel and NVIDIA who helped us optimise the various implementations.