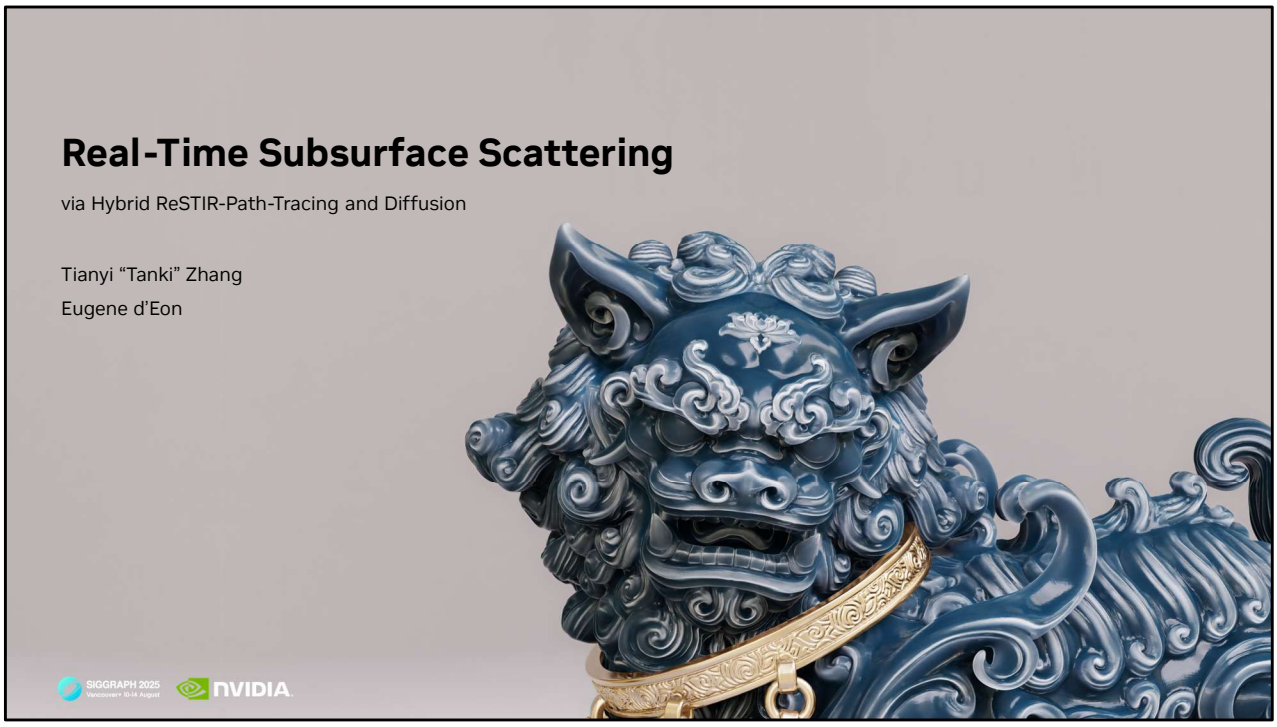
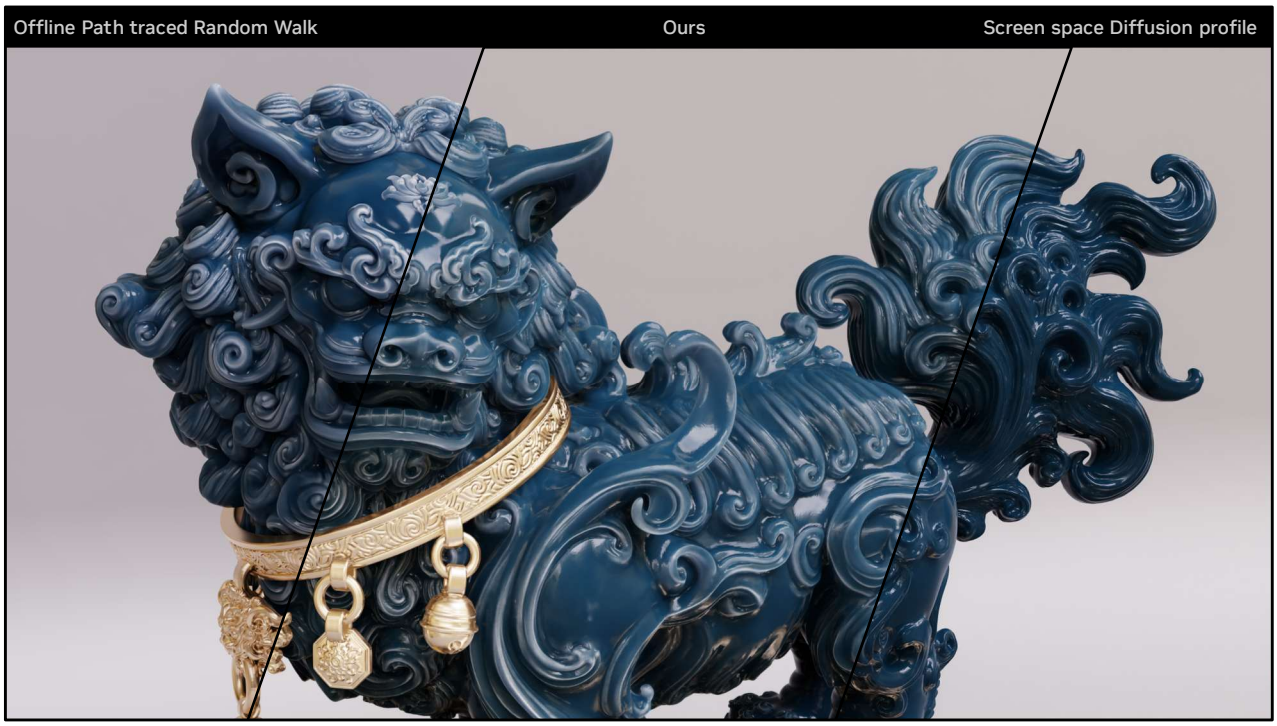


Hello everyone. Welcome to Advances in Real-Time Rendering in Games. I'm Tianyi Zhang, usually go by Tanki. I and Eugene have been working on the Omniverse RTX renderer. While Eugene is not here for personal reasons, I'm excited to present Eugene's and my work on real-time subsurface scattering. This talk covers how we achieve realistic SSS in real-time by combining path-traced light transport with diffusion profiles. You'll see how our hybrid method delivers high-quality SSS fast enough for current generation pipelines. Let’s dive in!

Let me start by showing you what we can achieve. This stone lion statue is rendered with our technique using only a uniform transmission color and scattering distance—no texture mapping. All the rich detail you see comes from physically-based geometry interaction.



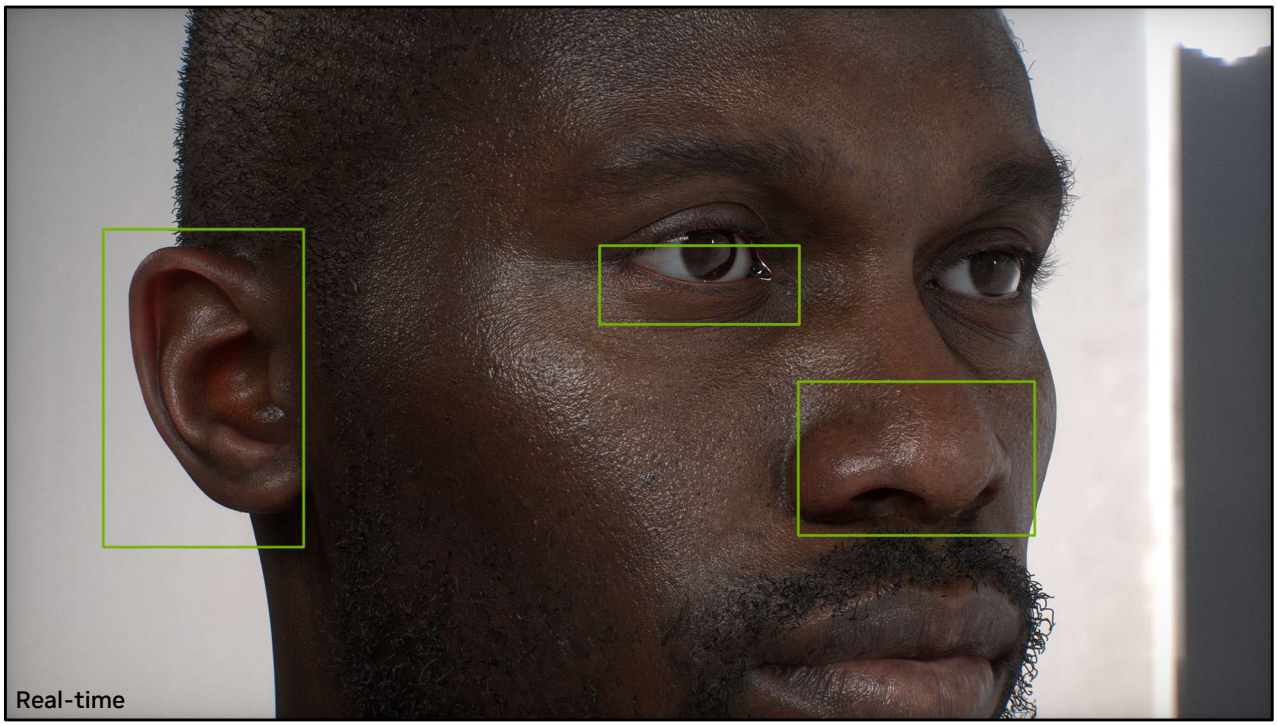
What's particularly exciting is that our result is closer to ground truth offline path tracing than state-of-the-art diffusion profile approaches.

Here's another example with these marble chess pieces, showing similar improvements in realism.

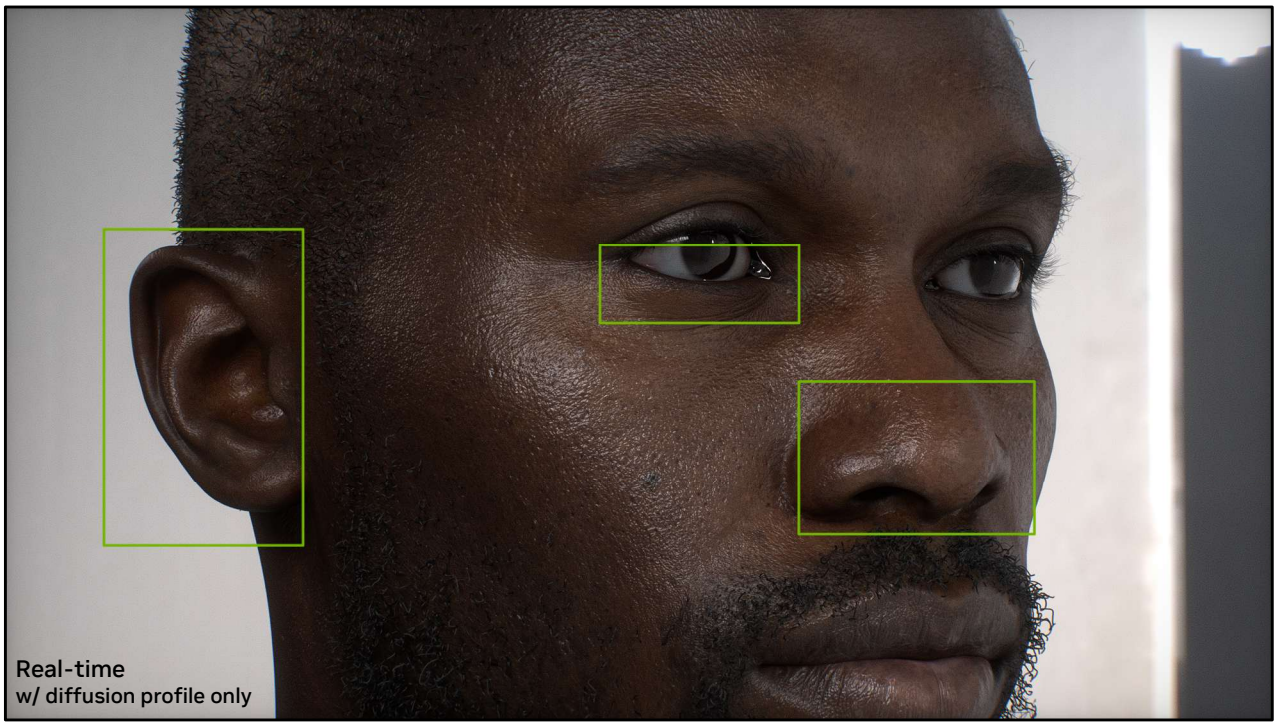
Moving to a different material—these candles have cylinder lights at the top representing flames, causing light to bleed through and create that characteristic glow you'd expect from real wax.

And here we have candies that combine subsurface scattering with rough transmission for a semi-translucent material.

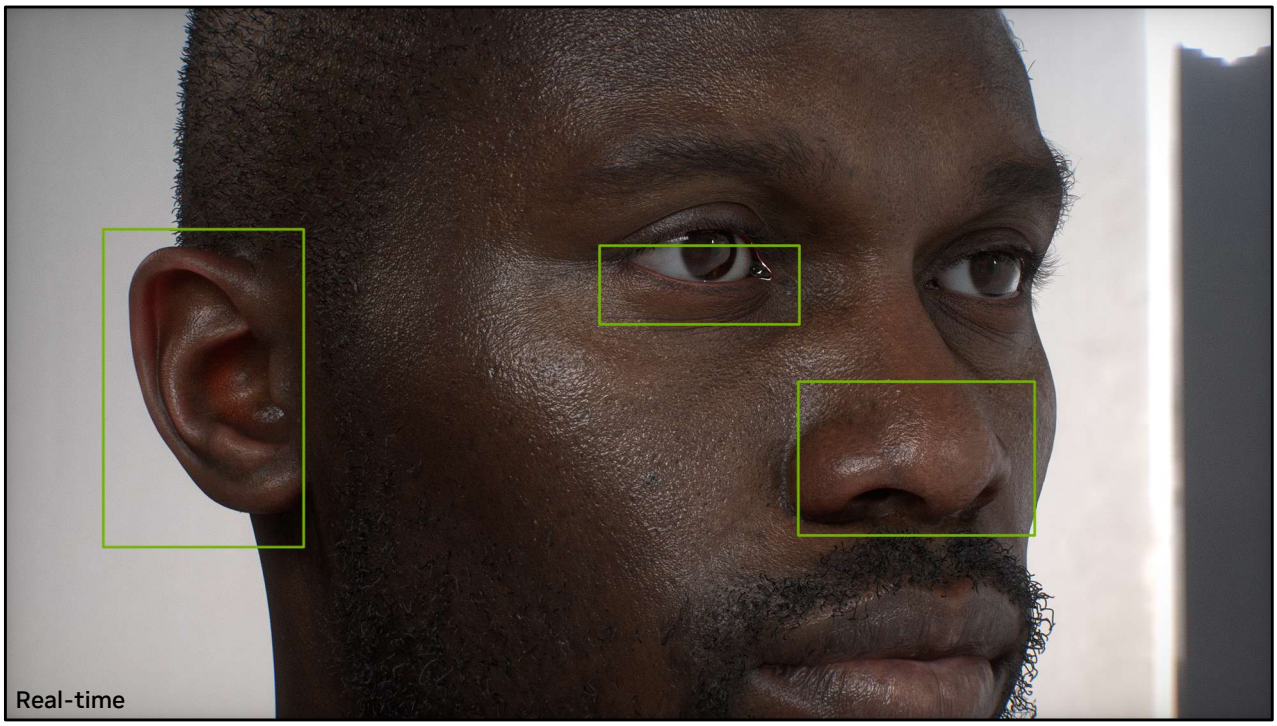
Asset modified from https://skfb.ly/6V7vG under CC 4.0 (https://creativecommons.org/licenses/by/4.0/)



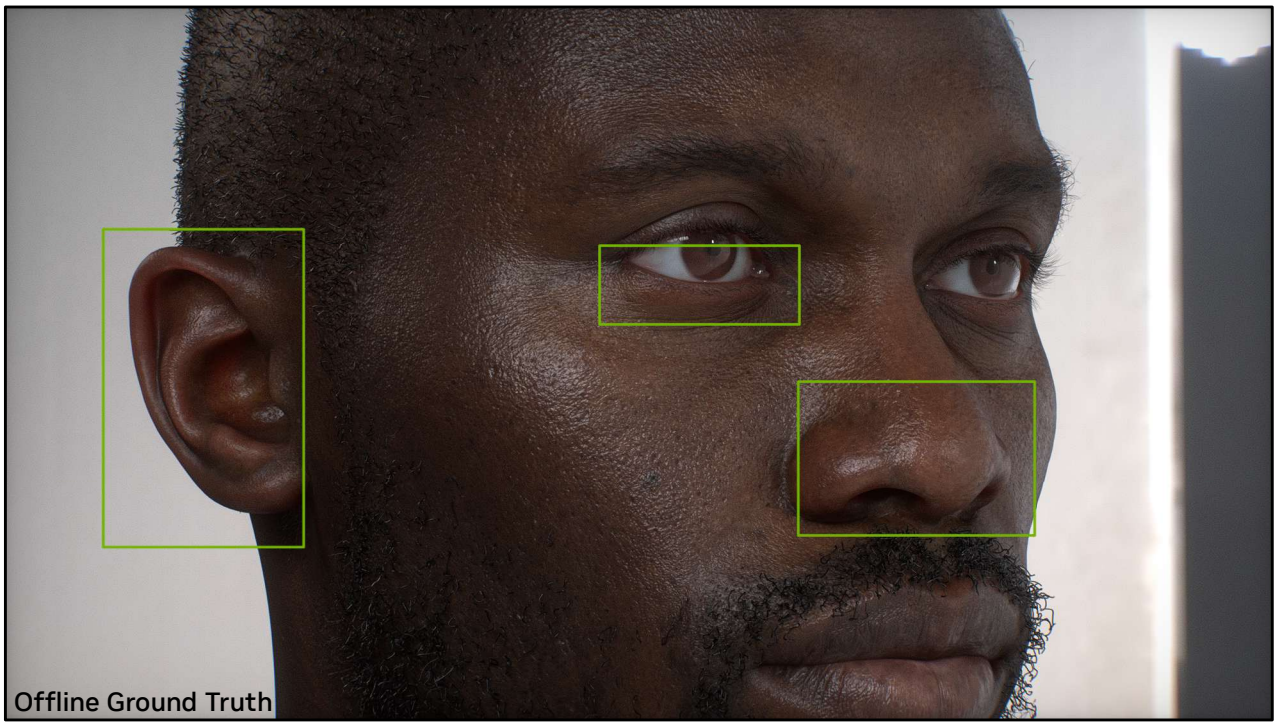
Now, since we're talking about subsurface scattering, we must show digital humans. This shot is rendered in real-time with our approach. Pay close attention to the ear, nose, and eyelids as I switch to the next slide.



Here's the same scene without our method—notice how all those subtle translucent effects completely disappear.



Let me flip back to our result, and now let's compare this against offline path tracing.



While there are some detail differences, our skin rendering captures significantly more detail with much closer ground truth matching.

Specular transmission with SSS

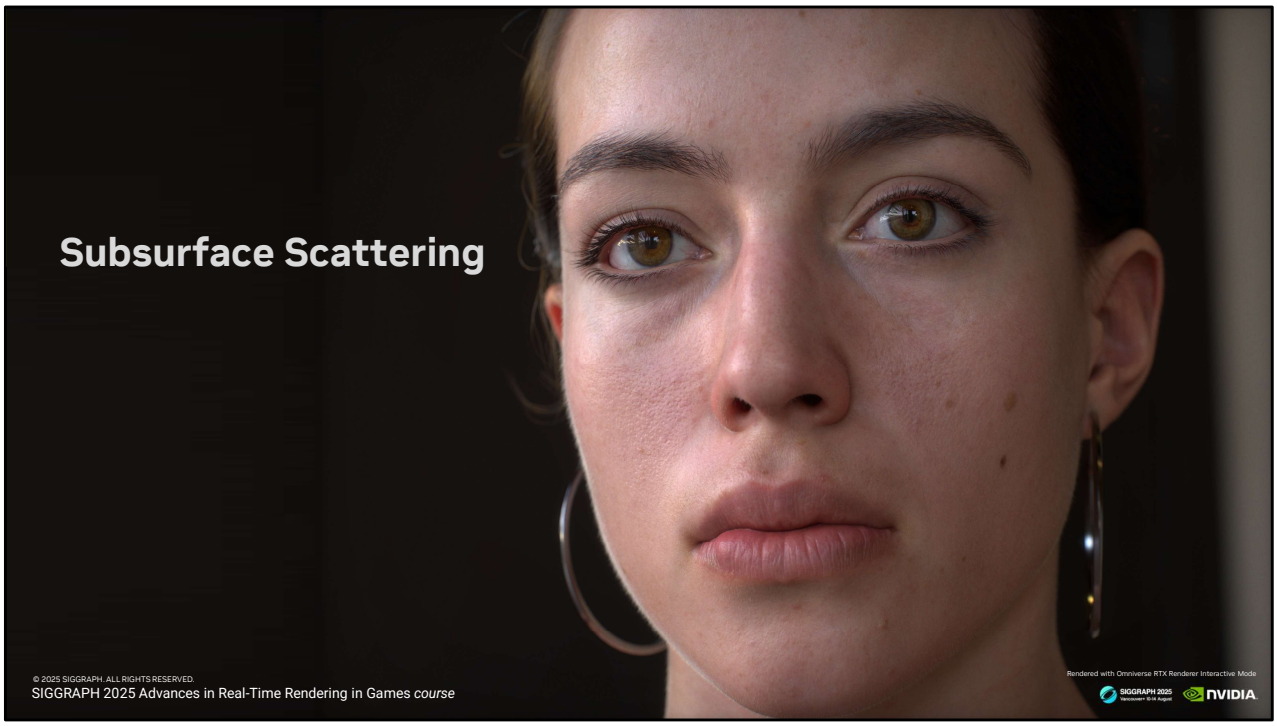
Before we dive into the technical details, I want to highlight one more thing about material richness.

This is how the candies look without any transmission.

This shows only transmission, no subsurface scattering.

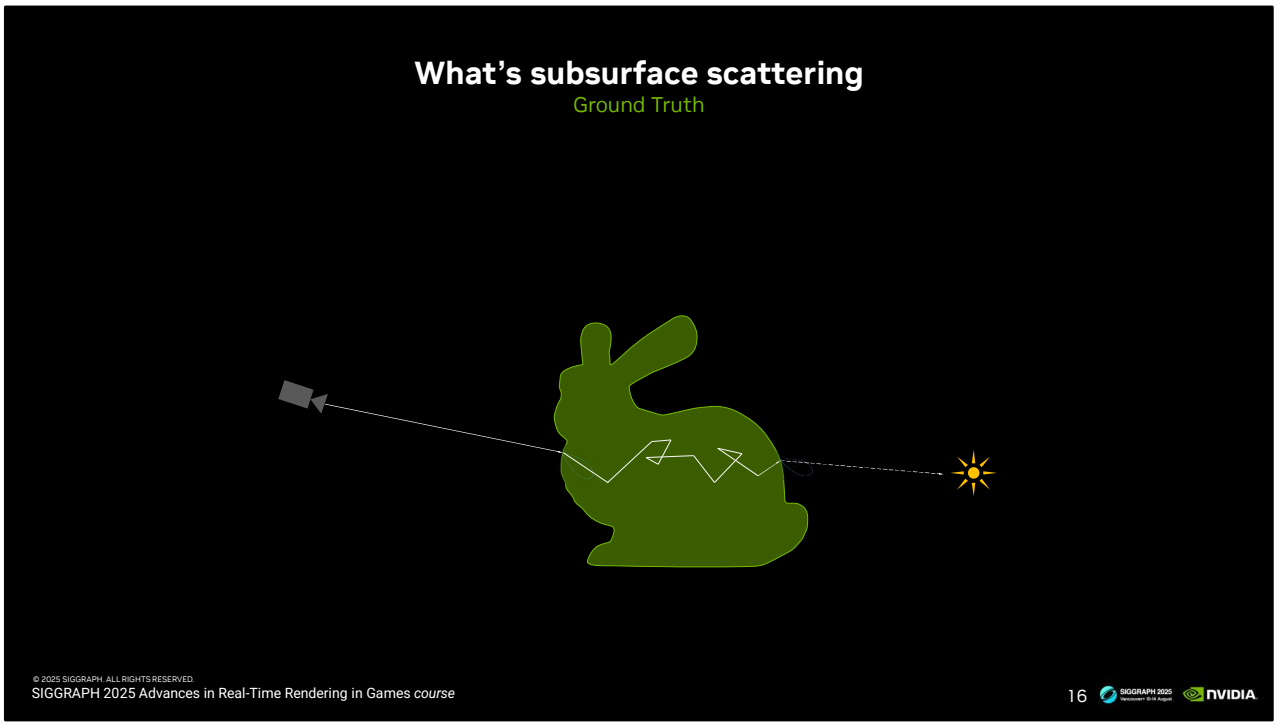
Specular transmission with SSS

But combining both effects gives us those delicious-looking candies we saw earlier!

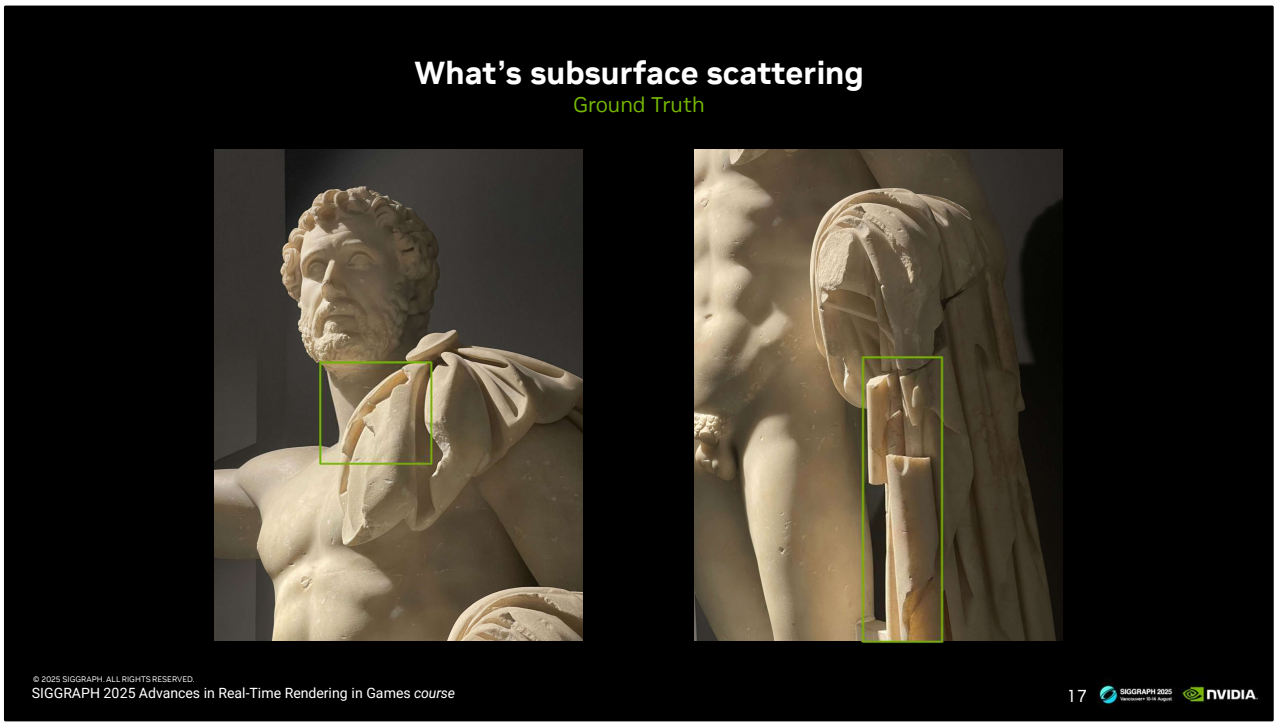


Alright, now let's officially begin with the fundamentals behind subsurface scattering.

NVIDIA



Subsurface scattering is fundamentally volume light transport inside objects. Light penetrates the surface, travels through translucent material, scatters internally, then exits at different points. This internal journey creates the soft appearance and color bleeding we see in materials like skin, wax, and marble.



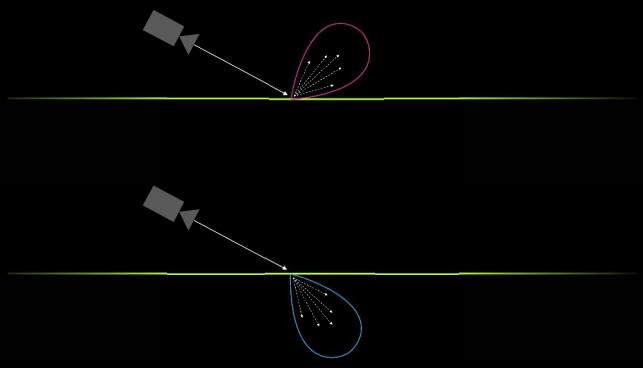
I was in Italy recently and seeing those beautiful roman statues full of SSS, so here we are.

You can see, when light hits the surface, a portion refracts into the marble. Inside, it might scatter off mineral particles many times and lose some energy to absorption. Eventually some of that light finds its way back out of the surface at another point. The result is that the statue's shadows and shading are softened by this internal blur of light, and you can also see light bleed through. Sometimes, it's hard to differentiate SSS from GI. But in this case, I stood there for 10mins studying the lighting, just to make sure it was SSS before I took the shot, so I am certain those circled spots are SSS. The glow on the leg, is from the GI.

What's subsurface scattering

Ground Truth

- A homogeneous volume bound by geometry interface
  - Geometry boundary - transmission BSDF

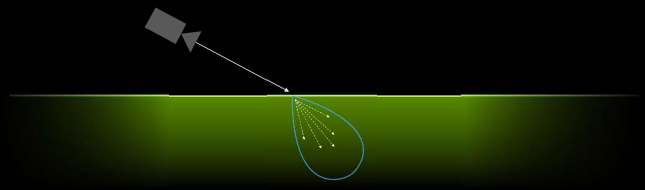


More formally, SSS refers to light transport within solid objects that behave as bounded volumes. We assume homogeneous material properties throughout the interior, with the geometry acting as boundaries. When rays hit surfaces, they can either reflect via the reflection BSDF—shown in pink—or enter via the transmission BSDF in blue.

What's subsurface scattering

Ground Truth

- A homogeneous volume bound by geometry interface
  - Geometry boundary - transmission BSDF
  - A solid geometry - Volume light transport

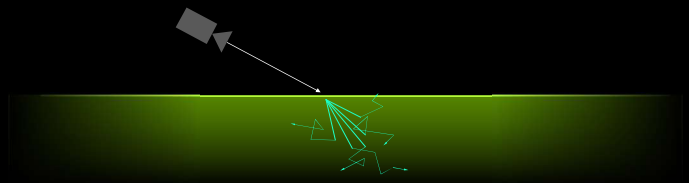


Once inside, light undergoes volume transport just like participating media. So effectively, SSS is volume scattering that happens after surface transmission, where light can scatter multiple times internally.

What's subsurface scattering

Establishing Ground Truth

- A homogeneous volume bound by geometry interface
  - Geometry boundary - transmission BSDF
  - A solid geometry - Volume light transport
- Ground truth: random walk

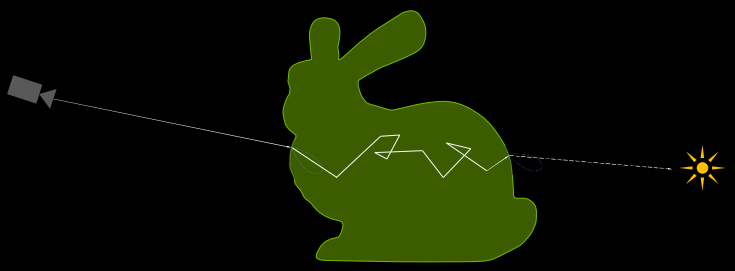


High-quality offline renderers handle this through random walks: for each entering ray, they stochastically sample scattering distances and directions, repeating this process until the photon either gets absorbed or exits.

What's subsurface scattering

Establishing Ground Truth

- A homogeneous volume bound by geometry interface
  - Geometry boundary - transmission BSDF
  - A solid geometry - Volume light transport
- Ground truth: random walk

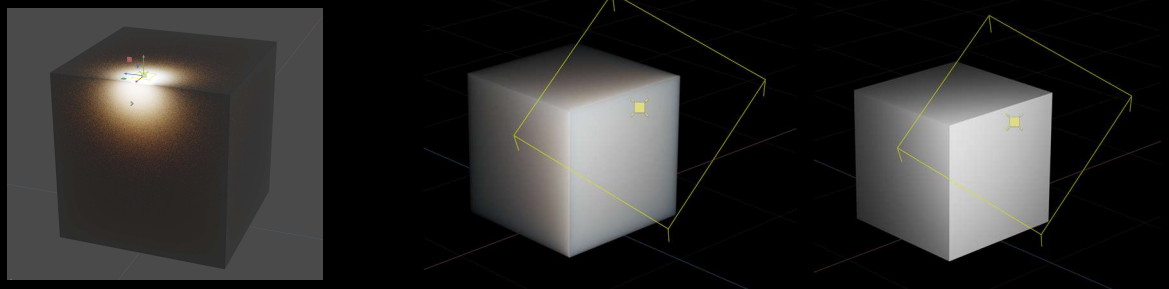


These random walks require many samples to converge, but they serve as our accuracy reference—we'll keep comparing our hybrid method against these random-walk simulations to ensure we capture the important effects.

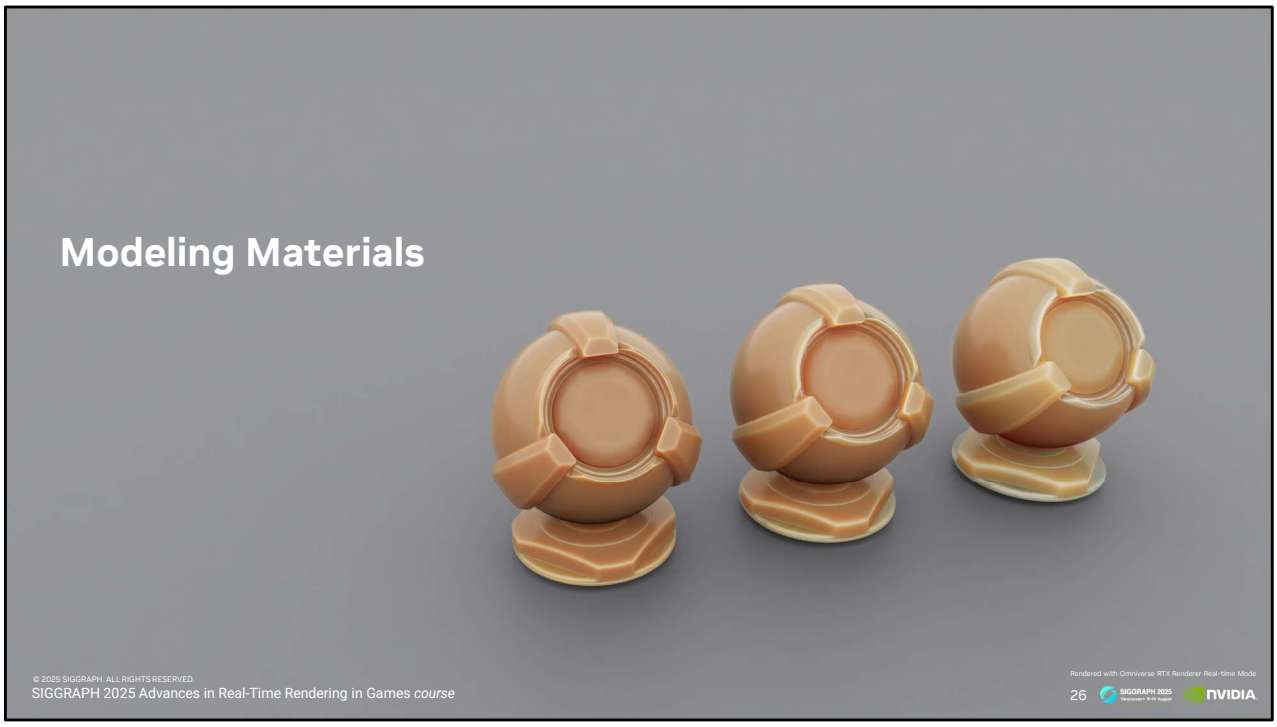
What's subsurface scattering

Ground Truth

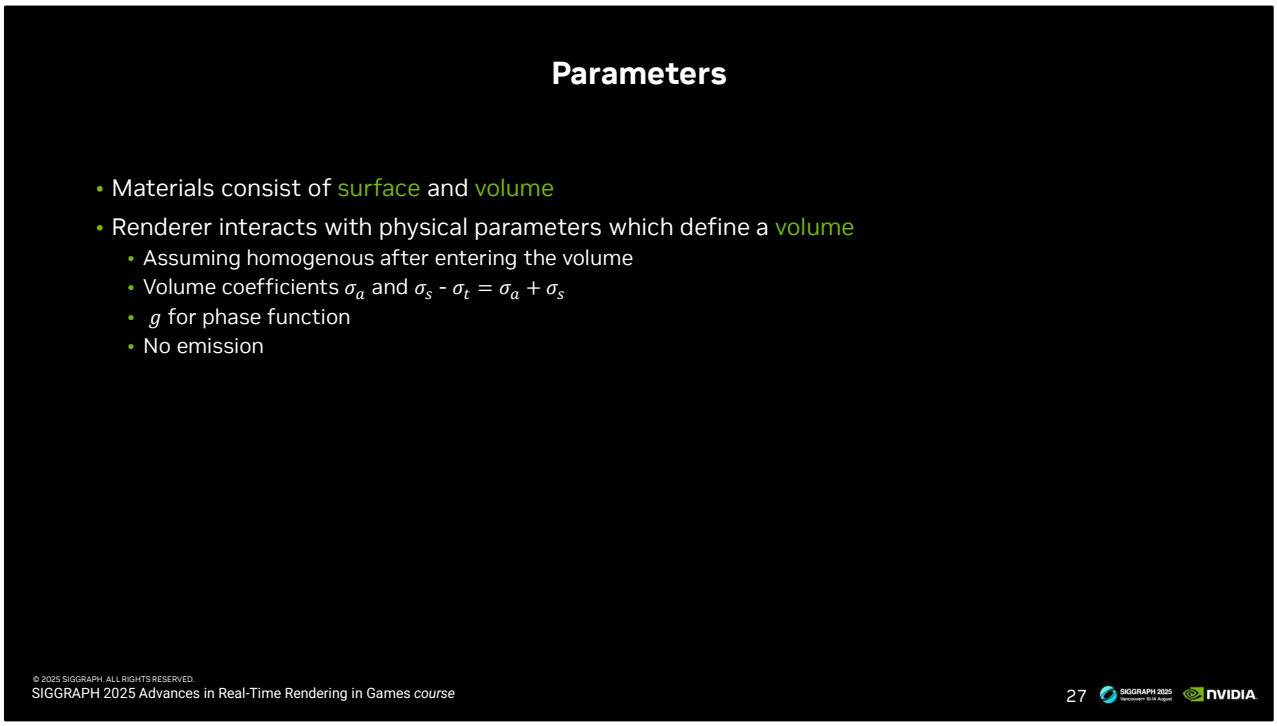
- A homogeneous volume bound by geometry interface
  - Geometry boundary - transmission BSDF
  - A solid geometry - Volume light transport
- Ground truth: random walk



Let me show you two examples of what random walks produce. On the left, we have a small rectangular light placed almost against the surface—you can see how light bleeds through and scatters before exiting. In the middle, a large light from above creates those signature bright edges, making the cube appear almost like smoke. Compare this to the solid cube on the right under identical lighting.



Now that we know what SSS entails physically, let's talk about how we model SSS materials. For production SSS materials, we need specific methods that are both physically-based and artist-friendly.



When we think about a material, it's common that we are thinking about the surface property of the appearance. To define SSS properly, we need to explicitly define its volume behavior – How things behave within the mesh.

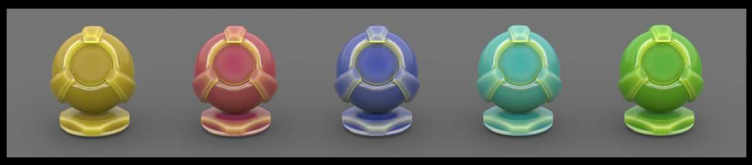
We assume homogeneous volumes with properties determined at ray entry—this is common even for offline rendering. While not perfectly accurate for complex materials like human flesh with its varying tissues, it's a reasonable approximation that provides performance benefits. Renderers ultimately need absorption coefficient, scattering coefficient, and anisotropy.

Unlike actual participating media, SSS does not have emission. We expect the emission is pre-integrated and expressed as part of the surface properties.

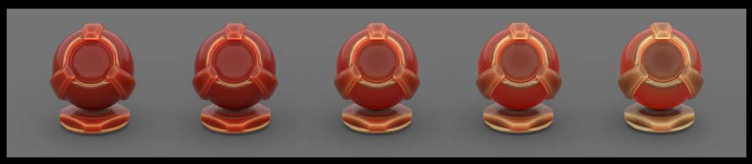
Parameters

- Provide a default higher level, artist-friendly mapping
  - Color - C
  - Scatter distance (mean free path) - mfp
  - Uniform scale for scatter distance
  - Anisotropy - g

Various color - C
Same scatter distance - mfp



Same color - C
Various scatter distance - mfp



However, these parameters aren't intuitive for artists, so we provide higher-level color mapping parameters instead. Color reflects the general appearance, while smaller mean free path values create denser materials that eventually approach reflection-only behavior. Those parameters can be texture mapping too, to offer maximum flexibility for artist.

Here is some examples. You can see color reflects the general appearance of the material, while with smaller mfp, the material behave more and more dense, which eventually becomes almost like a reflection only material.

Parameters

- Provide a default higher level, artist-friendly mapping
  - Color - C
  - Scatter distance (mean free path) - mfp
  - Uniform scale for scatter distance
  - Anisotropy - g

$$S = 4.09712 + 4.20863\,C - \sqrt{9.59217 + 41.6808\,C + 17.7126\,C^2}$$

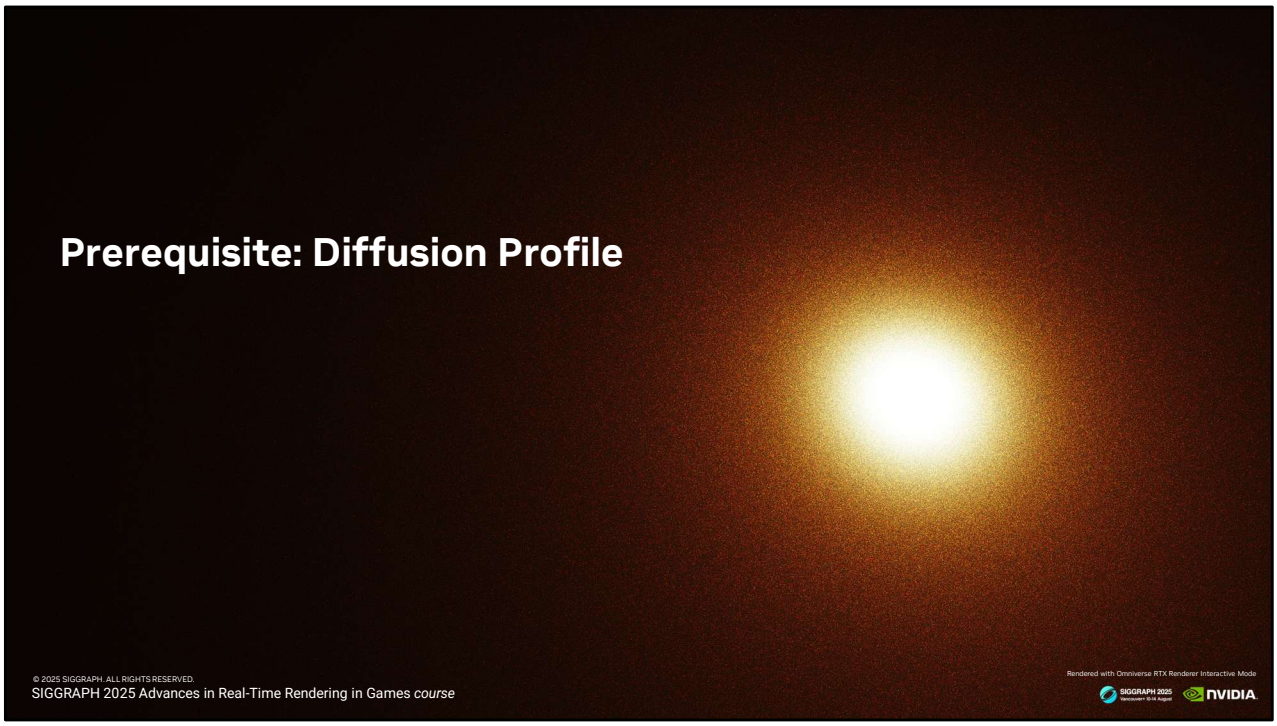
(single scattering albedo) $\alpha = \frac{1-s^2}{1-gs^2}$

$$\sigma_t = \frac{1}{\text{mfp}},\ \sigma_s = \frac{\alpha}{\text{mfp}}$$

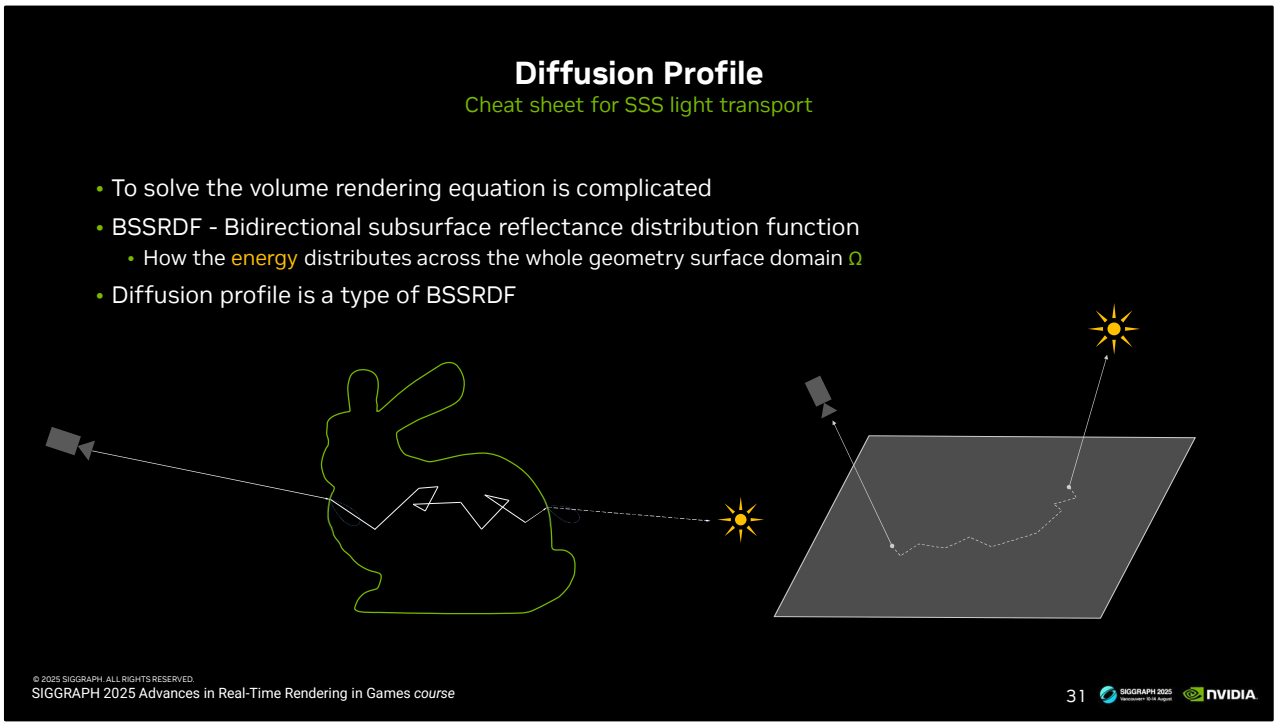
[Kulla 2017]

Here's how we solve this: we compute volume coefficients σ_t and σ_s by first calculating an intermediate value S from the color parameters, combining this with anisotropy g to get single scattering albedo, then dividing by the mean free path.

[Kulla 2017] Kulla, C. and Conty, A. Revisiting Physically Based Shading at Imageworks. 139.



Now, before I describe our hybrid approach, we need to understand diffusion profiles—the technique that's been widely used for real-time SSS approximation.



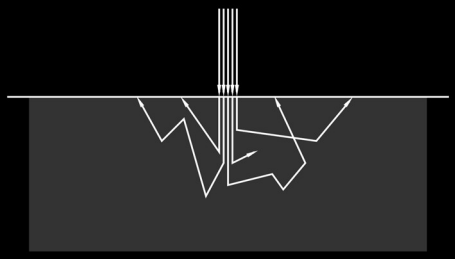
As we saw, full volume rendering is complex and slow. This is where the BSSRDF concept comes in—it generalizes BRDF for light entering at one surface point and exiting at another. Diffusion profiles are essentially precomputed BSSRDF functions that show how light spreads across a surface after entering the material.

NVIDIA

Diffusion Profile

Cheat sheet for SSS light transport

- Burley Diffusion profile
- 2D MC simulation of scene:
  - Semi-infinite, flat surface
  - Infinitely thick
- based on search light configuration
  - Shoot single light beam
  - perpendicular to surface
- implementation [Golubev 2018] [Golubev 2019] [Xie 2020]



[Burley 2016]

The widely used profile in game rendering is based on Burley 2015's work. I want to lay my emphasis on how the profile is built.

Burley diffusion profile is built based on Monte Carlo simulation and curve fitting under search light configuration.

imagine an idealized scenario: a semi-infinite, flat surface of a homogeneous material. "Semi-infinite" means the surface extends infinitely in the x–y directions and is infinitely thick deep into the object. This ensures we don't lose energy because light exit through the "back" of the object.

We shoot a light beam perpendicular to the surface. In the picture, it actually shows 4 beams which just represents 4 independent samples

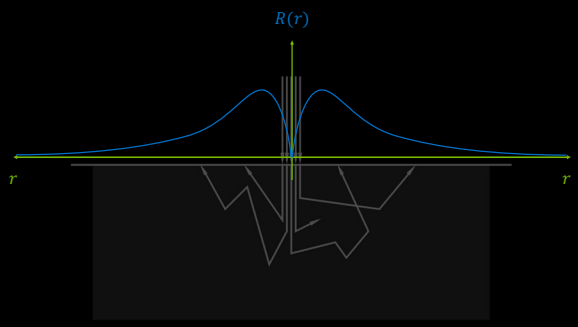
[Golubev 2018] Evgenii Golubev. 2018. Efficient screen-space subsurface scattering using Burley's normalized diffusion in real-time. Retrieved Aug 29, 2019 from http://advances.realtimerendering.com/s2018/Efficient%20screen%20space%20subsurface%20scattering%20Siggraph%202018.pdf
[Golubev 2019] Evgenii Golubev. 2019. Sampling Burley's Normalized Diffusion

Profiles. Retrieved Nov 17, 2019 from https://zero-radiance.github.io/post/sampling-diffusion/
[Xie 2020] Tiantian Xie, Marc Olano, Brian Karis, and Krzysztof Narkowicz. 2020. Real-time subsurface scattering with single pass variance-guided adaptive importance sampling. Proc. ACM Comput. Graph. Interact. Tech. 3, 1, Article 3 (Apr 2020), 21 pages. https://doi.org/10.1145/3384536

Diffusion Profile

Cheat sheet for SSS light transport

- 1D profile, symmetric across any direction

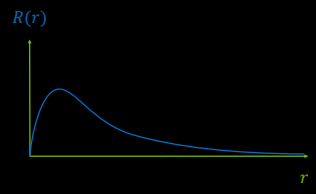


The Monte Carlo simulation measures outgoing light versus distance from the entry point, producing a diffuse reflectance profile R(r) for given material properties.

Diffusion Profile

Cheat sheet for SSS light transport

- 1D profile, symmetric across any direction

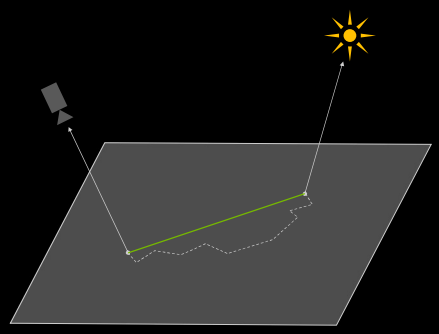


Since the surface is infinite and flat, this outgoing distribution has circular symmetry around the entry point, which gives us a clean 1D radial function.

Diffusion Profile

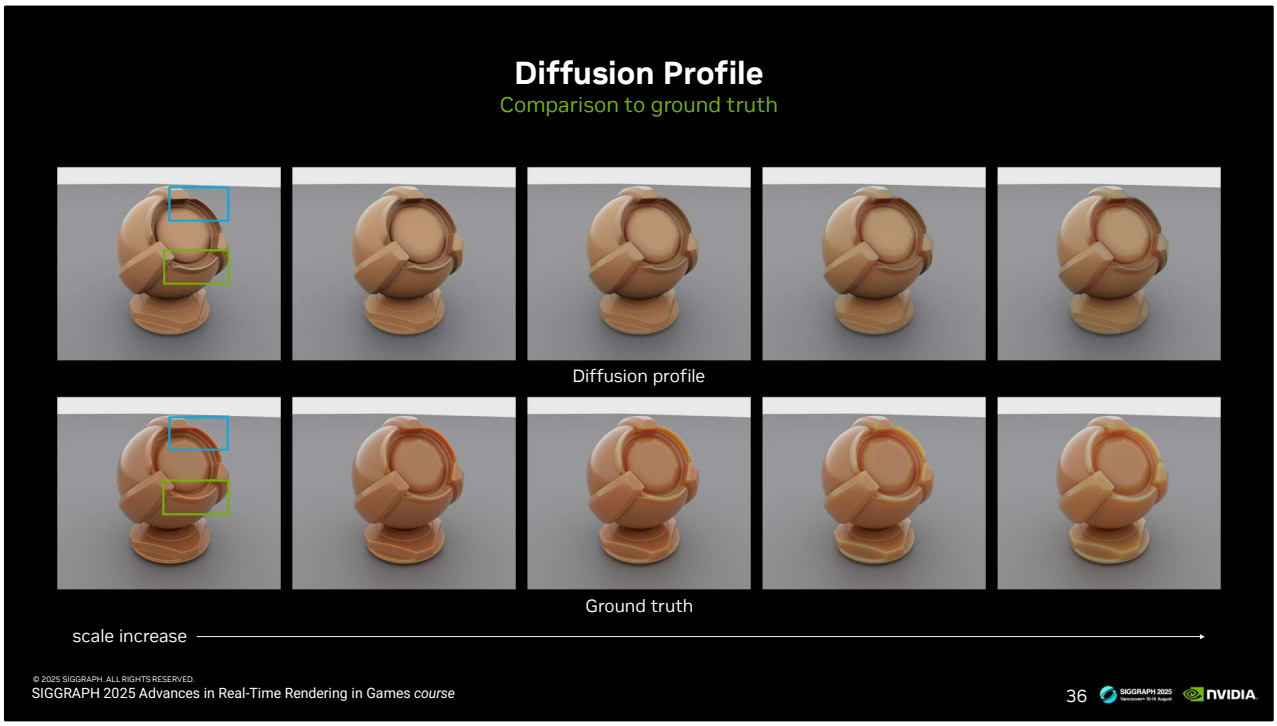
Why it works

- Transform the 3D, nested integral into a 2D surface integral
- A direct $R(r)$ "look up" by using r



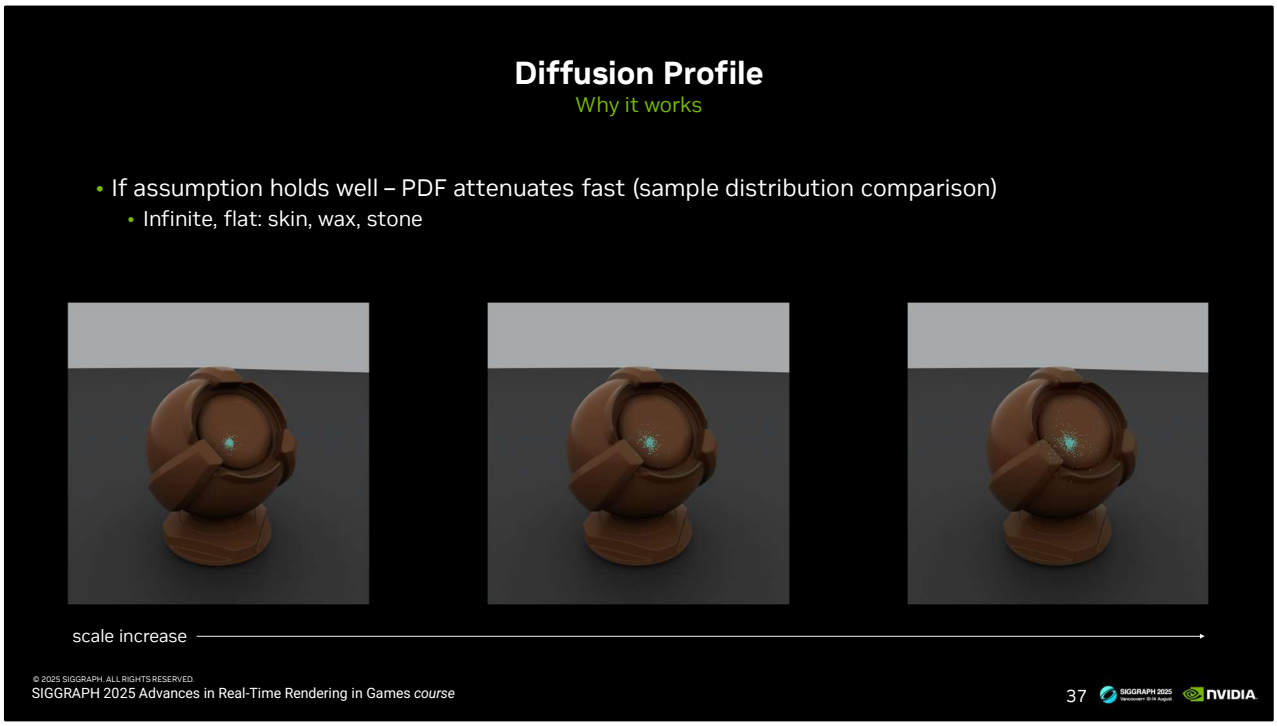
[Pbrt 2016]

This is powerful because diffusion simplifies 3D light transport into 2D surface integration. Instead of simulating volume scattering, we simply evaluate precomputed profiles over distance, directly returning the effects of multiple scattering.



When we compare against ground truth, we see that diffusion quality degrades significantly with increasing scattering scale, even for simple materials and geometry.

NVIDIA



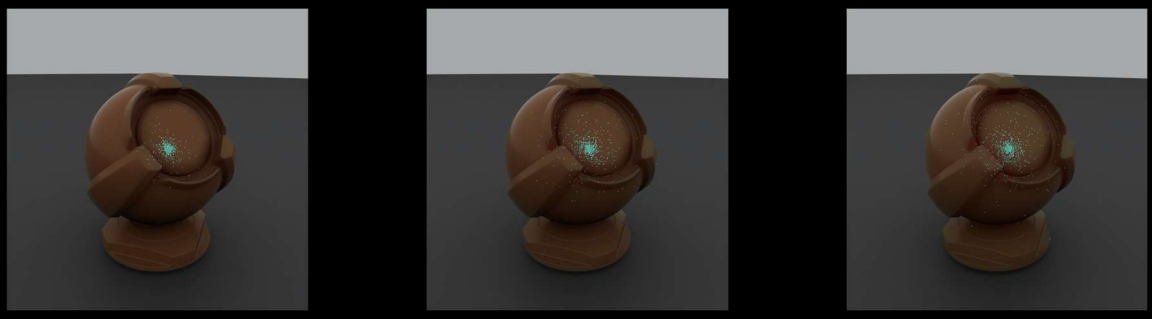
This is because diffusion works best when surfaces are locally flat and large compared to the scatter distance. But as you can see in these examples where scale increases, screen-space samples, which are shown in blue, spread out and accuracy decreases.

(scale 0.1, 0.2, 0.4)

Diffusion Profile

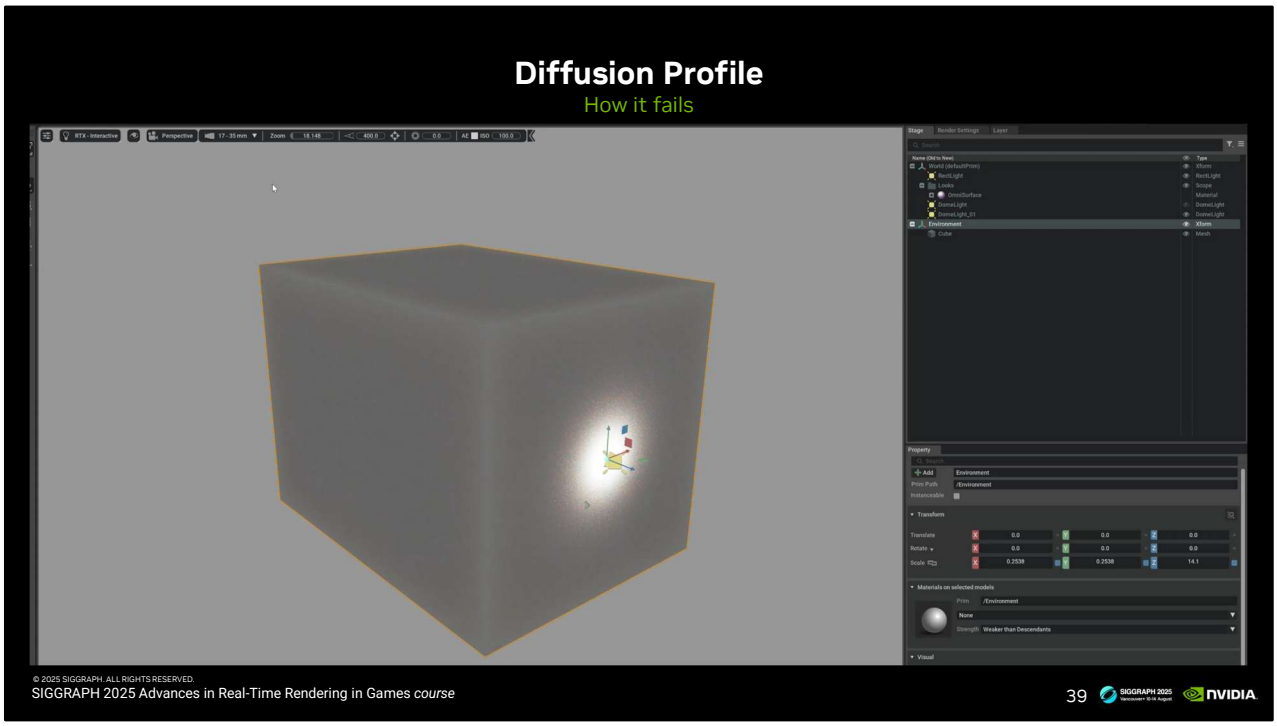
Why it works

- If assumption holds well – PDF attenuates fast (sample distribution comparison)
  - Infinite, flat: skin, wax, stone

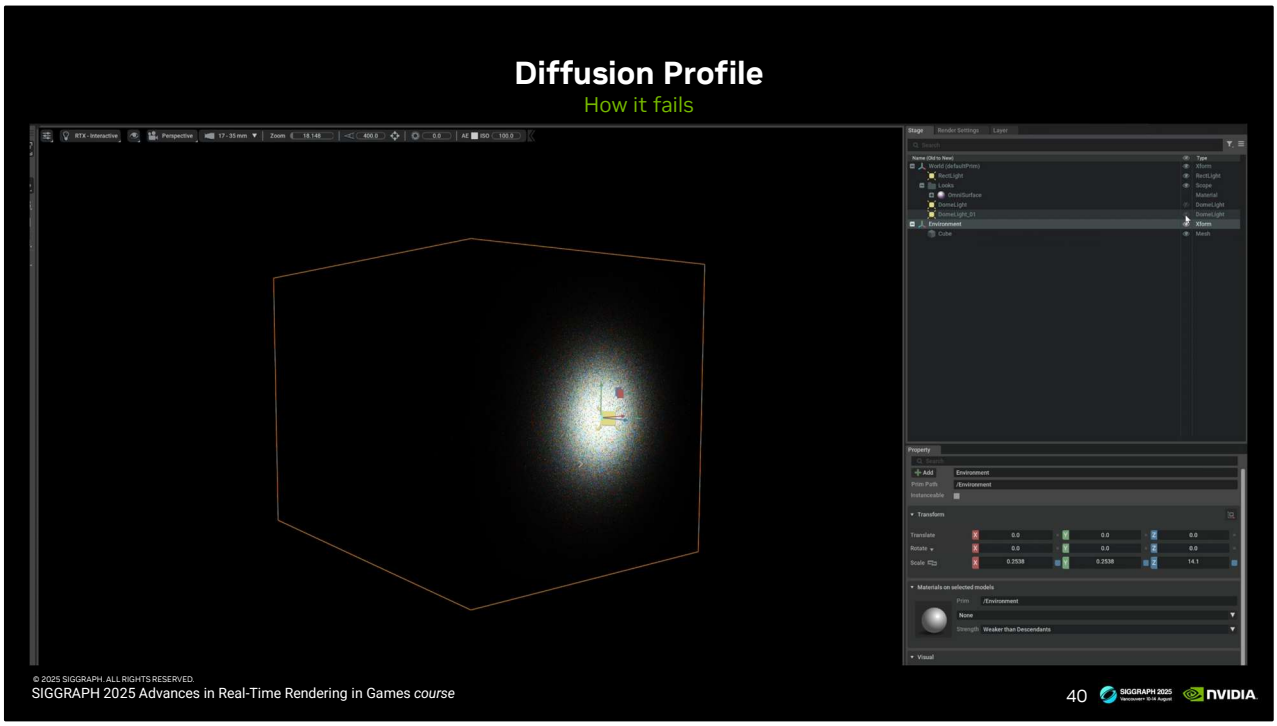


scale increase

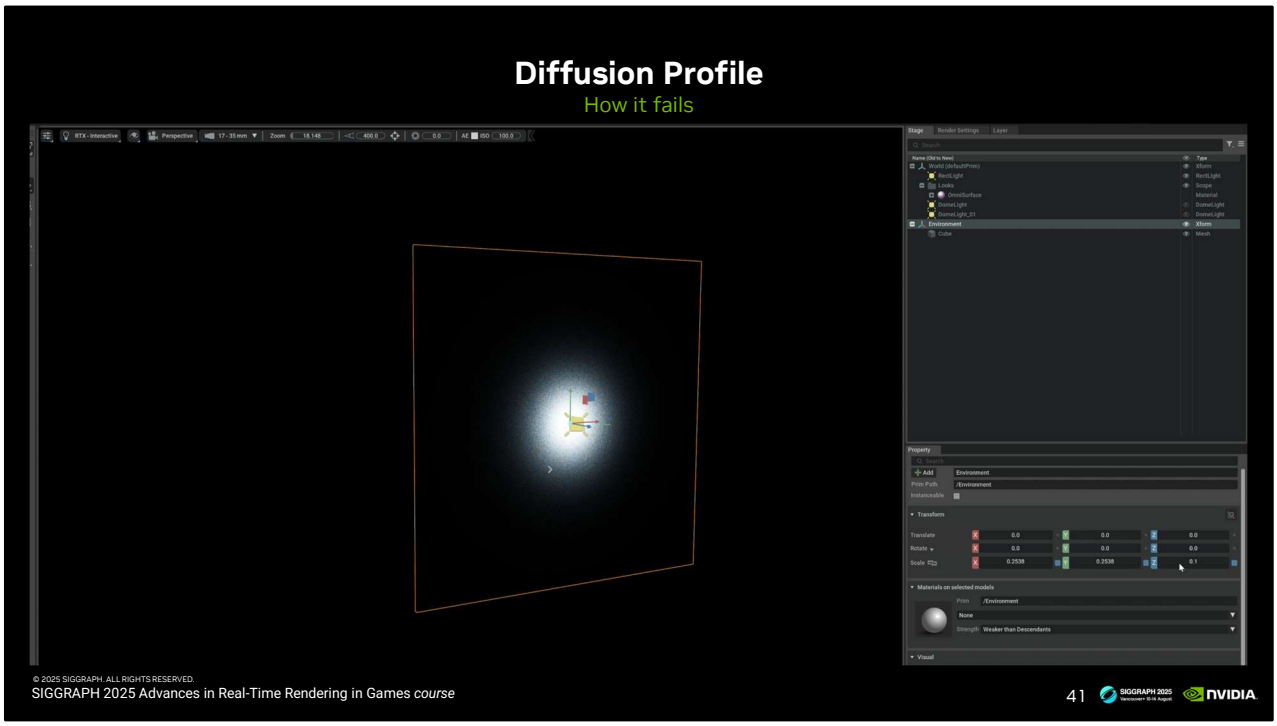
Eventually, samples scatter so far that they violate our assumptions and create bias. You get the same phenomenon when scaling models with identical materials.
(scale 0.4, 0.7, 1)



Let’s walk through this video. Like before, we have an SSS cube with a small rectangular light placed right against its surface. At the start, I leave the dome light on so you can get a better sense of the scene layout. You’ll see that once I switch to the real-time view with a diffusion profile, pay attention to the bright spot, the size, the brightness, it matches the ground truth pretty well.



Then I disable the dome light to isolate the light beam effect. They match well. As I shrink the cube down into a thin slab, you'll notice light begins to bleed through the back.



I flipped between real-time and ground truth, notice the real-time has bigger spot than ground truth. At that point, the diffusion profile breaks down. it no longer matches the ground truth, because now a lot of light is escaping out the far side. The assumption of semi-infinite thickness no longer holds. And once I thicken the slab, it matches better again.

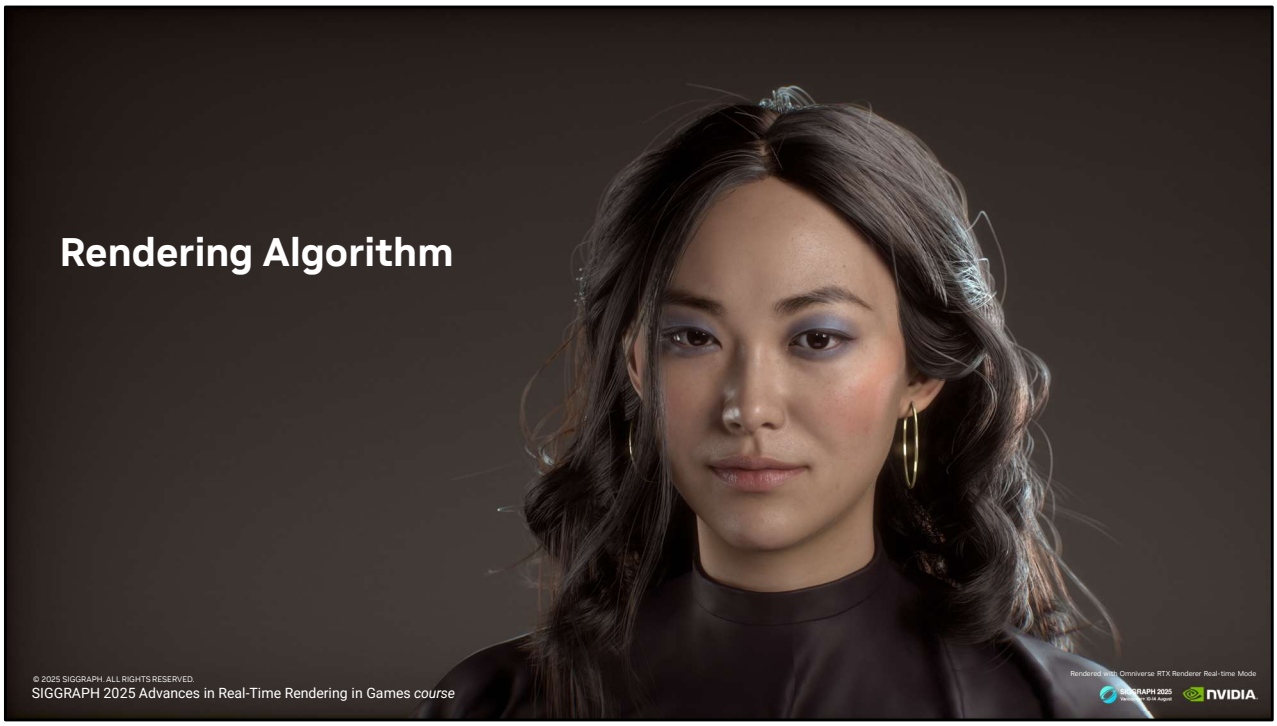
Originally, the cube was thick enough that even though it wasn't infinite, it behaved close enough. But once the shape thins out, that same dense material starts acting sparse. Diffusion keeps trying to scatter light as if it were still deep, and ends up overshooting the effect.

Diffusion Profile

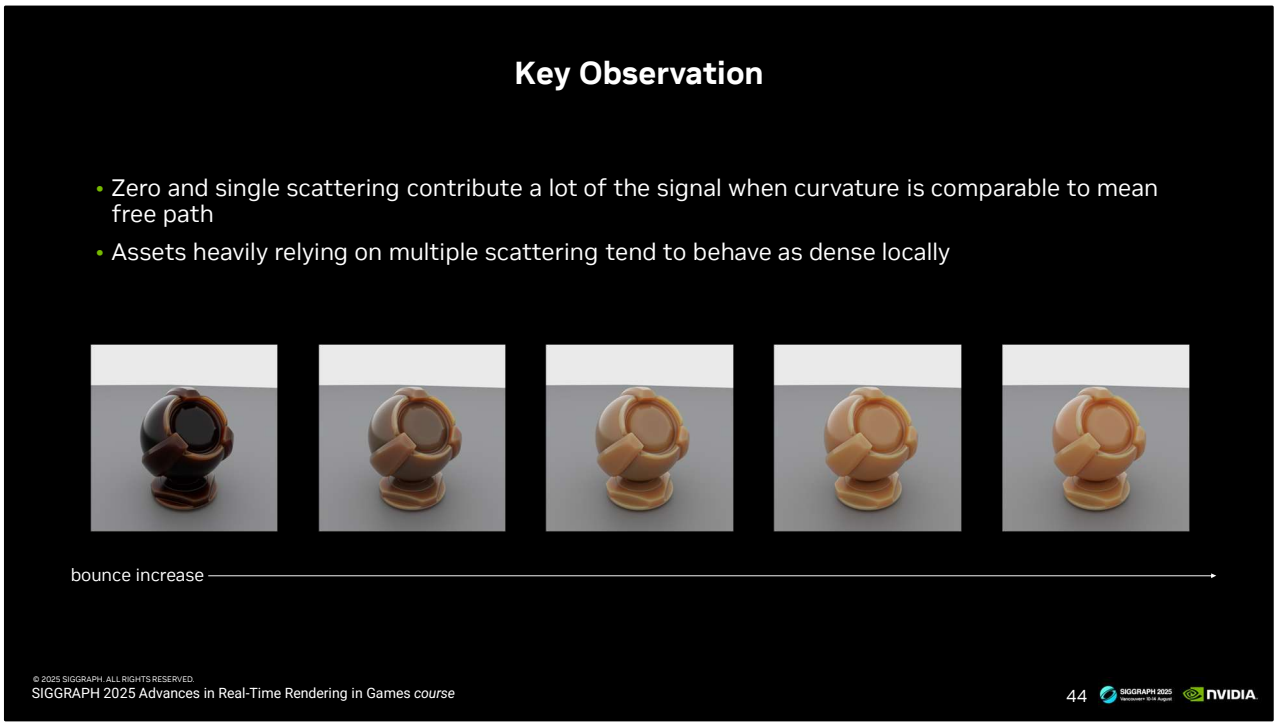
How it fails

- The infinite, flat assumption is relative to the mean free path
  - This Dense materials can deviate from a flat surface more than materials with thin geometry
  - Accurately predicting scattering in curved geometry is challenging

This highlights a core limitation: geometry isn't explicitly handled by diffusion. Things like thickness, concavities, or sharp curvature aren't part of the model—it has no awareness of the actual shape.



Understanding these diffusion limitations, let me now present our new approach, which is based on the observations we've made about both the strengths and weaknesses of current methods.



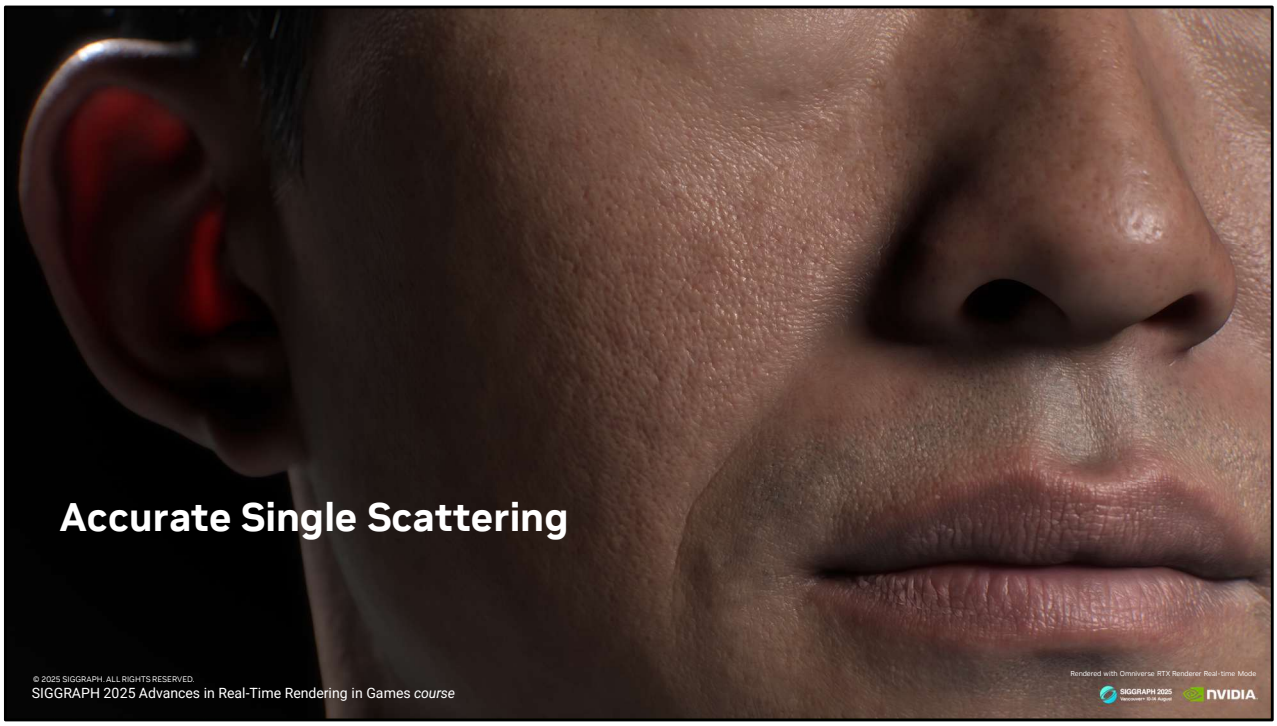
During development, we noticed something important about subsurface scattering in typical assets. Looking at these renders with different bounce counts, going from 0 to 1 bounce makes a huge visual difference—capturing most of the character. But adding more bounces often yields diminishing returns, just softening or saturating the look.

(bounce: 1, 2, 4, 8, 16)

Combined Approach

- Path trace zero and single scattering
- Diffusion approximation for higher order scattering

Our approach accurately captures zero and single scattering through ray tracing, giving us the detail and directionality of full random walks for these critical first events. For multi-scattering, we use diffusion approximation to add the soft, smooth lighting those higher bounces would produce. Tracing limited bounces is much cheaper than unknown bounce counts, while diffusion evaluation remains very fast.



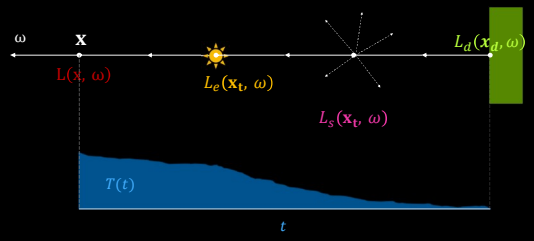
So let me focus now on the accurate single scattering part of our solution.

NVIDIA

Theory

- Revisit the volume rendering equation, allow the radiance beam or photon to split into fractions instead of doing Russian roulette as in the tracking methods [Fong 2017]
- The new equation has each of the terms in a separate integral

$$L(x,\omega) = \int_{t=0}^{d} T(t)\sigma_a(x_t)L_e(x_t,\omega)dt + \int_{t=0}^{d} T(t)\sigma_s(x_t)L_s(x_t,\omega)dt + T(d)L_d(x_d,\omega)$$



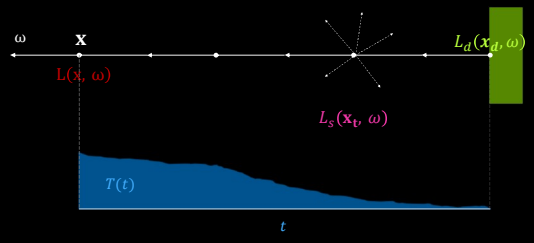
Applying this PDF approach to volume rendering creates separate integrals—one purely for scattering contributions, and one for unscattered transmission.

Theory

In Subsurface Scattering Context

- We can ignore emission for SSS

$$L(x,\omega) = \int_{t=0}^{d} \cancel{T(t)\sigma_a(x_t)L_e(x_t,\omega)}dt + \int_{t=0}^{d} T(t)\sigma_s(x_t)L_s(x_t,\omega)dt + T(d)L_d(x_d,\omega)$$



As I mentioned before, emission is modeled through the surface property, which means we can ignore the emission term within the volume transport.

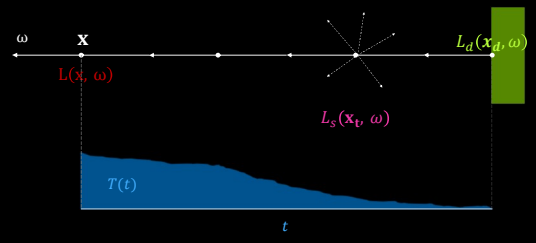
Theory

In Subsurface Scattering Context

- There are 2 components
  - Scattering term
  - Boundary term

Scattering Boundary

$$L(x,\omega) = \int_{t=0}^{d} T(t)\sigma_s(x_t)L_s(x_t,\omega)dt + T(d)L_d(x_d,\omega)$$



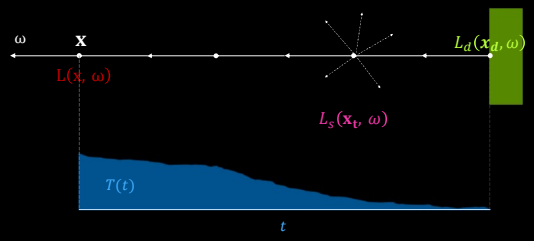
We label these as the Scattering term—which handles exactly one volume scattering event; And the Boundary term—which covers zero scattering where light just passes through. This separation clarifies our computation targets and dramatically simplifies the original equation. The boundary term is analytical, so we only need Monte Carlo for the scattering term.

Theory

In Subsurface Scattering Context

- Recall $T(t) = \exp\left(-\int_{s=0}^{t} \sigma_t(x_s) d\,s\right)$
- The key to sampling the integral is to find a pdf for $T(t)$
  - Hard for general volumes because density is not constant

$$L(x, \omega) = \int_{t=0}^{d} T(t)\sigma_s(x_t)L_s(x_t, \omega)dt + T(d)L_d(x_d, \omega)$$



The key challenge becomes sampling the scattering distance t with the proper distribution. For varying σ_t, this sampling would be quite complex.

Theory

In Subsurface Scattering Context

- The key to sampling the integral is to sample PDF - t'
  - Lucky for us, we can assume SSS being a homogeneous volume
  - Following Beer-Lambert law: $T(t) = \exp(-\sigma_t t)$

ξ Random number

Fortunately, since SSS assumes homogeneous volumes with constant σ, transmittance T(t) follows an exponential distribution according to Beer-Lambert law.

Theory

In Subsurface Scattering Context

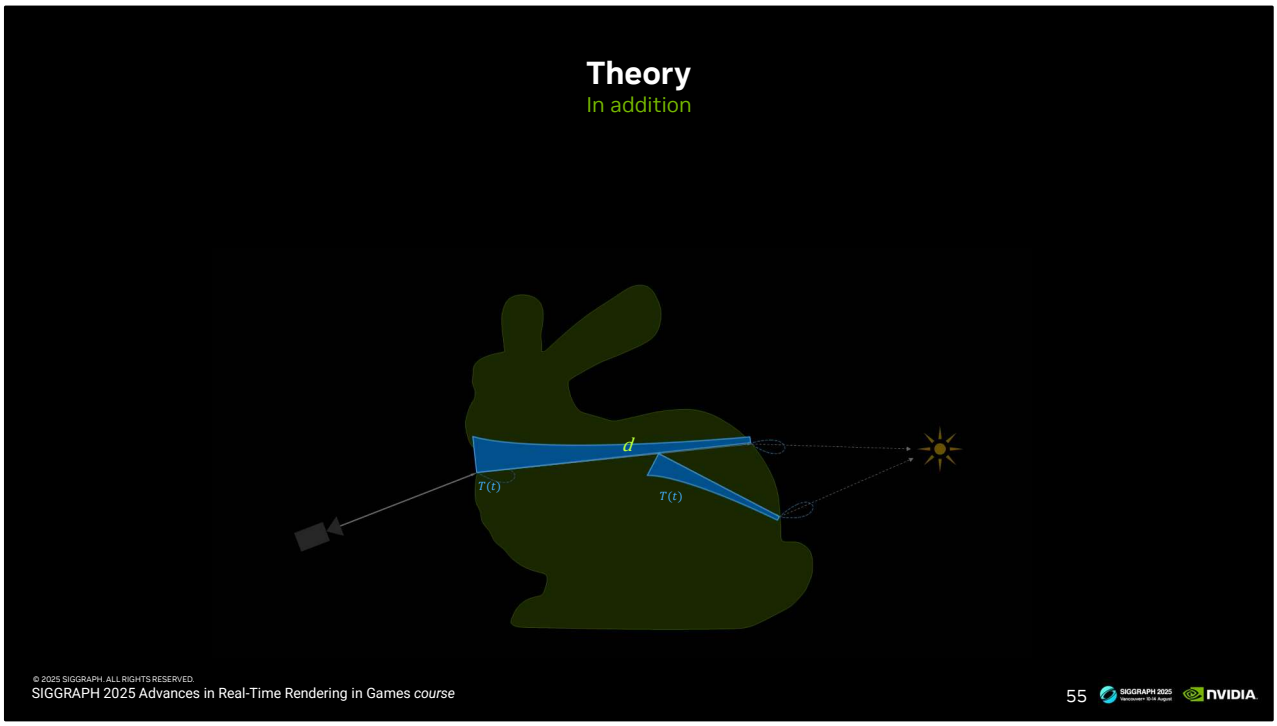
- The key to sampling the integral is to sample PDF - t'
  - Lucky for us, we can assume SSS being a homogeneous volume
  - Following Beer-Lambert law: $T(t) = \exp(-\sigma_t t)$
- Basic way: assuming $0 < t < \infty$ [Pbrt 2016]
  - Normalizing the integral: $p(t) = \sigma_t \exp(-\sigma_t t)$
  - Perfect sample $T(t)$ through inversion method: $t' = -\ln(1-\xi)/\sigma_t$

ξ Random number

[Fong 2017]

Using uniform random numbers between 0 and 1, we can normalize T(t) to create a sampling PDF. The inversion method gives us a simple logarithmic form.

[Pbrt 2016] Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. Physically Based Rendering: From Theory to Implementation (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.



Now, T(t) has this exponential distribution, but since t can theoretically reach infinity, T(t) doesn't reach zero at volume boundaries. This means our random sampling might fall outside the volume entirely, forcing us to discard samples.

Theory

In Subsurface Scattering Context

- The key to sampling the integral is to sample PDF - t'
  - Lucky for us, we can assume SSS being a homogeneous volume
  - Following Beer-Lambert law: $T(t) = \exp(-\sigma_t t)$
- Basic way: assuming $0 < t < \infty$ [Pbrt 2016]
  - Normalizing the integral: $p(t) = \sigma_t \exp(-\sigma_t t)$
  - Perfect sample $T(t)$ through inversion method: $t' = -\ln(1-\xi)/\sigma_t$
- Better way: assuming $0 < t \leq \mathrm{d}$ [Pbrt 2016]
  - Normalizing the integral over the $(0, \mathrm{d})$: $p(t) = \frac{\sigma_t \exp(-\sigma_t t)}{1-\exp(-\sigma_t d)}$
  - Perfect sample $T(t)$ through inversion method: $t' = \frac{-\ln(1-\xi(1-\exp(-\sigma_t d)))}{\sigma_t}$

ξ Random number

[Fong 2017]

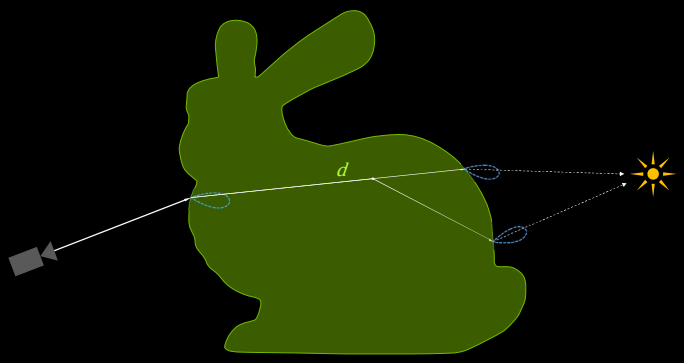
We can improve this by sampling the scatter distance while knowing the ray will exit at distance d. Constraining t between 0 and d reduces both variance and waste by re-normalizing the PDF appropriately.

[Pbrt 2016] Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. Physically Based Rendering: From Theory to Implementation (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Theory

In Subsurface Scattering Context

- Do not forget to account for interface: transmission BSDF
- Phase function: Henyey-Greenstein phase function, can be perfectly importance sampled

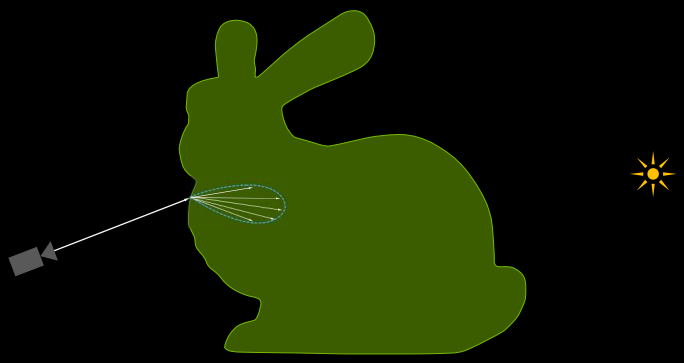


There are two key implementation details worth mentioning. First, we need transmission BSDF weights at surface entry and exit—this is often simplified by assuming diffuse transmission leads to SSS. Second, once we choose scattering points, we need to sample new directions using the phase function—we use the standard Henyey-Greenstein phase function which has an analytic sampling method.

Implementation Overview

Path Trace as Example

- Routine starts after a transmission event
  - Only if σ_s is non-zero
- Each refracted ray direction is determined by sampling transmission lobe

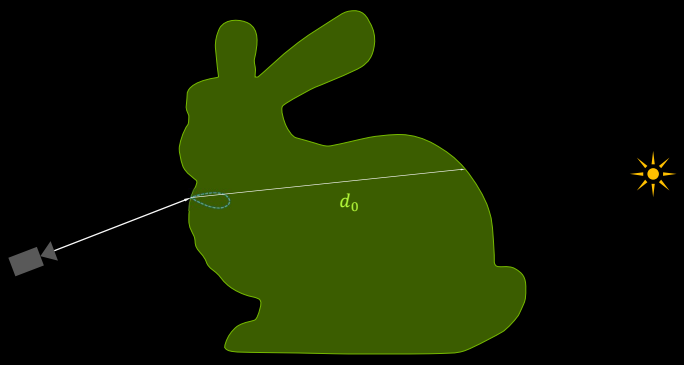


Here's how our algorithm works in practice. We handle rays that hit surfaces and decide to enter objects. The ray direction comes from sampling the transmission BSDF—this could be pure refraction for smooth dielectrics or diffuse transmission for rough translucent materials. Our algorithm isn't limited to any specific BSDF type.

Implementation Overview

Path Trace as Example

- Boundary term:
  - Cast this ray, hit the interface, evaluate d_0

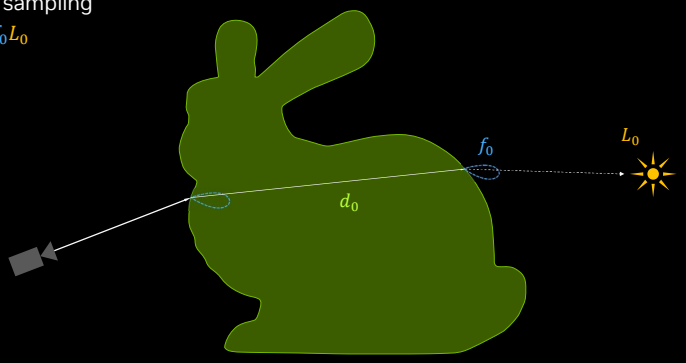


Next, we trace the ray through the interior until it hits the opposite surface at distance d. This is just a standard ray-scene intersection test continuing from the entry point.

Implementation Overview

Path Trace as Example

- Boundary term:
  - Cast this ray, hit the interface, evaluate d_0
  - Sample transmission BSDF again to exit volume
  - Light radiance sampling
  - Result $e^{-d_0 * \sigma_t} f_0 L_0$



At the exit point, we treat the interaction like any normal surface hit: we sample the transmission BSDF to determine how the ray leaves the surface and perform light radiance sampling at that exit point.

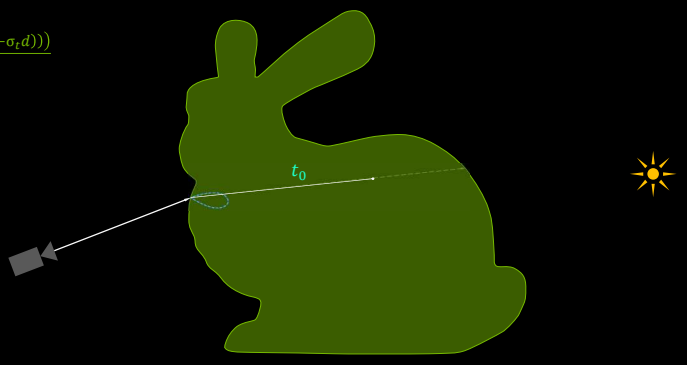
Implementation Overview

Path Trace as Example

- Single scattering term:
  - Sample distance t_0 with random number ξ_0

$$t' = \frac{-\ln(1-\xi(1-\exp(-\sigma_t d)))}{\sigma_t}$$

$$p(t) = \frac{\sigma_t \exp(-\sigma_t t)}{1-\exp(-\sigma_t d)}$$

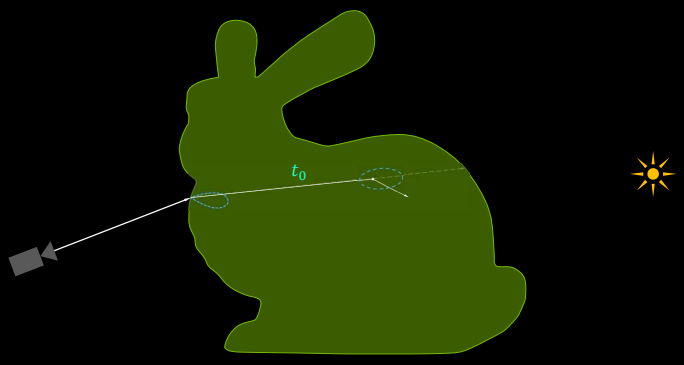


Now for the single scattering component. Knowing the exit distance d, we sample a scattering distance t_0 using fresh random numbers from our derived exponential distribution.

Implementation Overview

Path Trace as Example

- Single scattering term:
  - Sample distance t_0 with random number ξ_0
  - Sample phase function to refract the ray

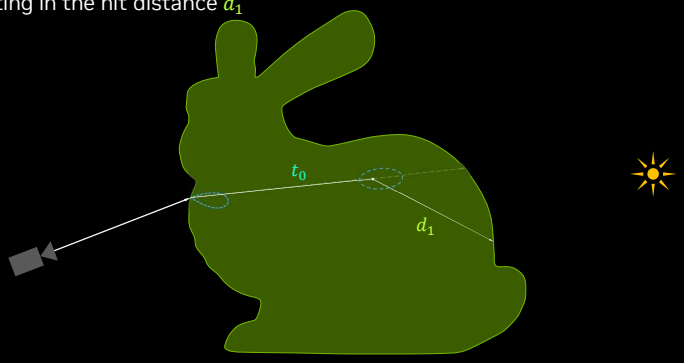


Then we sample the phase function to get a new direction for the scattered ray.

Implementation Overview

Path Trace as Example

- Single scattering term:
  - Sample distance t_0 with random number ξ_0
  - Sample phase function to refract the ray
  - Cast ray, resulting in the hit distance d_1

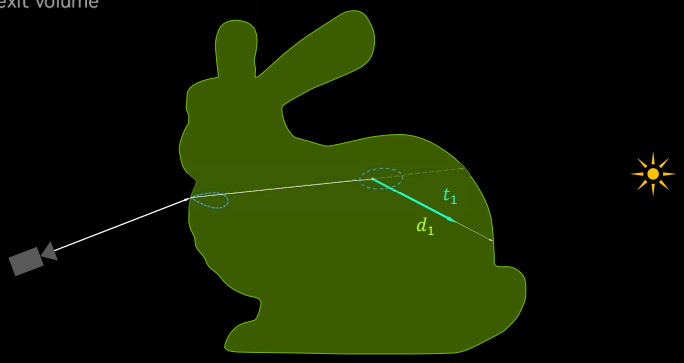


After scattering, we cast another ray from the scatter point in this new direction to find where it exits the object.

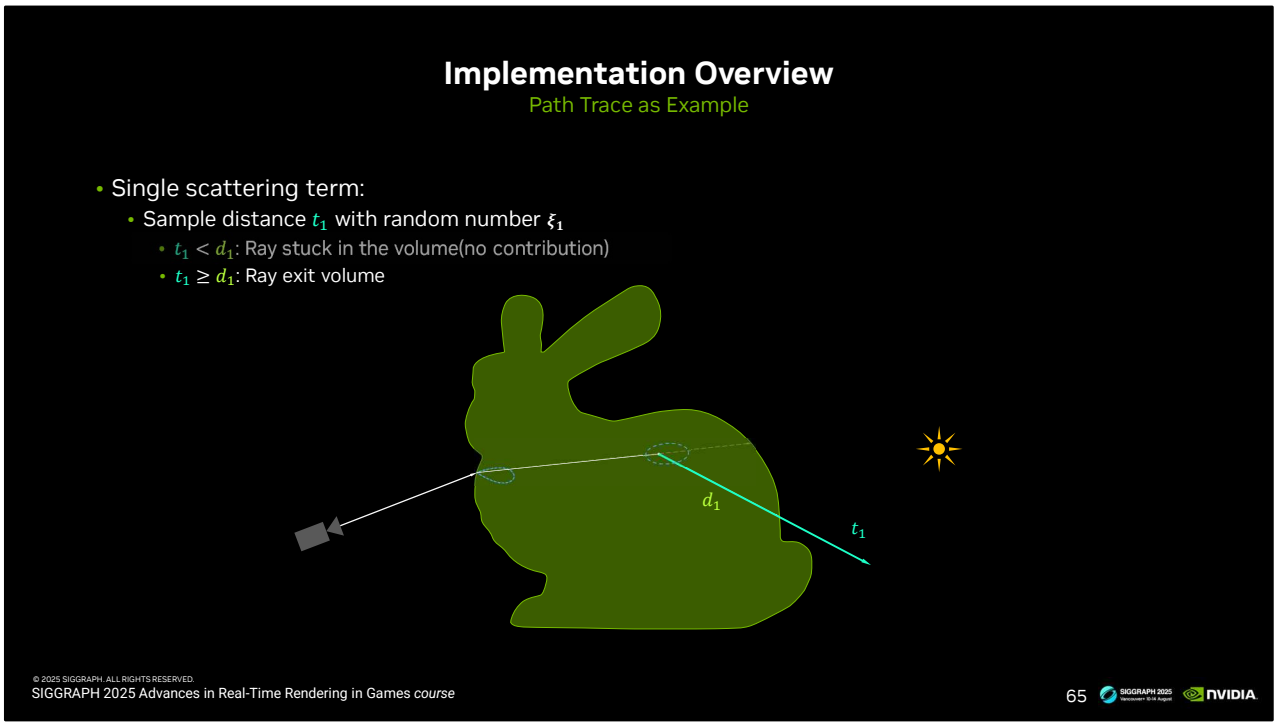
Implementation Overview

Path Trace as Example

- Single scattering term:
  - Sample distance t_1 with random number ξ_1
    - $t_1 < d_1$: Ray stuck in the volume(no contribution)
    - $t_1 \geq d_1$: Ray exit volume

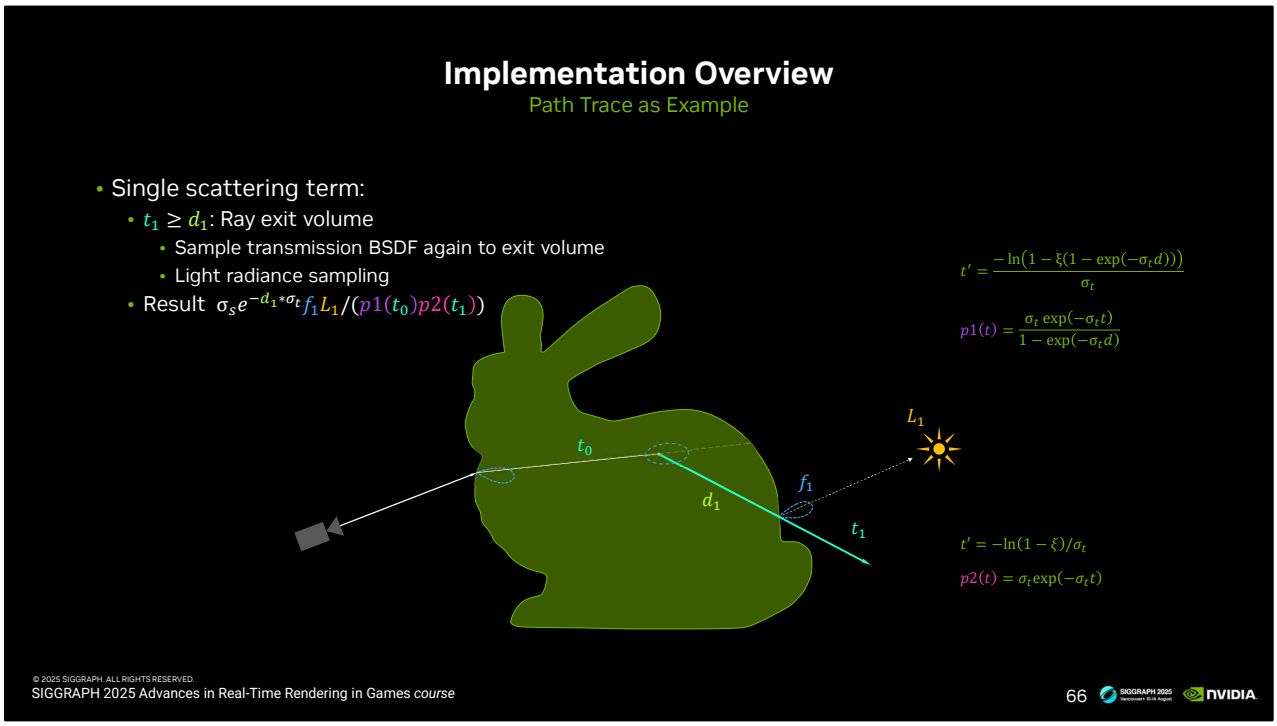


Our sampling logic handles two cases based on the sampled distance t. First case: t_1 is less than d_1, meaning the scatter point is inside the object—the photon scatters before reaching the far side. This ray would continue scattering within the volume, representing multiple scattering energy, which we ignore in this algorithm.



Second case: t_1 is greater than or equal to d_1, meaning the sampled scatter distance is beyond the exit boundary—essentially no scattering occurs before the photon leaves.

NVIDIA



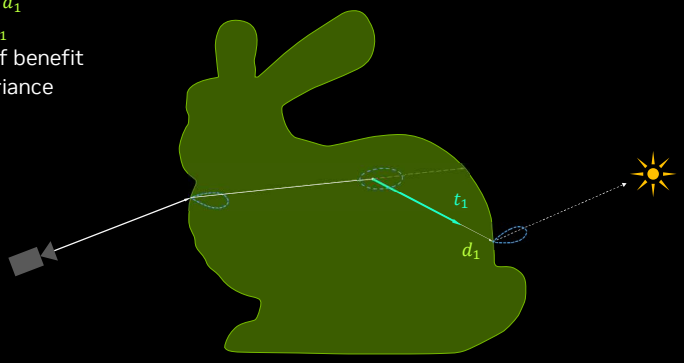
In this second case, we perform exit transmission BSDF sampling and light sampling for the scattered ray's exit point, gathering the single scattering energy contribution.
Check out the result term, because we got t0 and t1 both through sampling, you can see we need to apply both pdf. Then the exponential term which is the T(t), then finally the raidiance with bsdf weight f.

NVIDIA

Implementation Overview

Path Trace as Example

- Single scattering term:
  - Perform sample distance to determine t_1
  - Use t_1 to limit TMax for ray cast
    - Miss ➔ $t_1 < d_1$
    - Hit ➔ $t_1 \geq d_1$
  - Short ray - Perf benefit
  - Sample t_1 - variance



We use the basic sampling distribution here so that we can perform an adjusted routine instead.
Instead of casting ray first, we can sample the distance first, which would help to determine an optimal TMax for ray casting.

Unfortunately, since t_1 is sampled, the variance could increase.

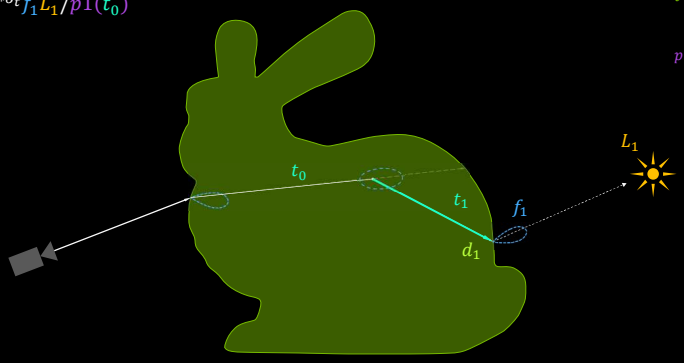
Implementation Overview

Path Trace as Example

- Single scattering term
  - Cast Ray with TMax = inf for d_1
  - Use exact d_1 to avoid sample distance
  - Result $\sigma_s e^{-d_1 * \sigma_t} f_1 L_1 / p1(t_0)$

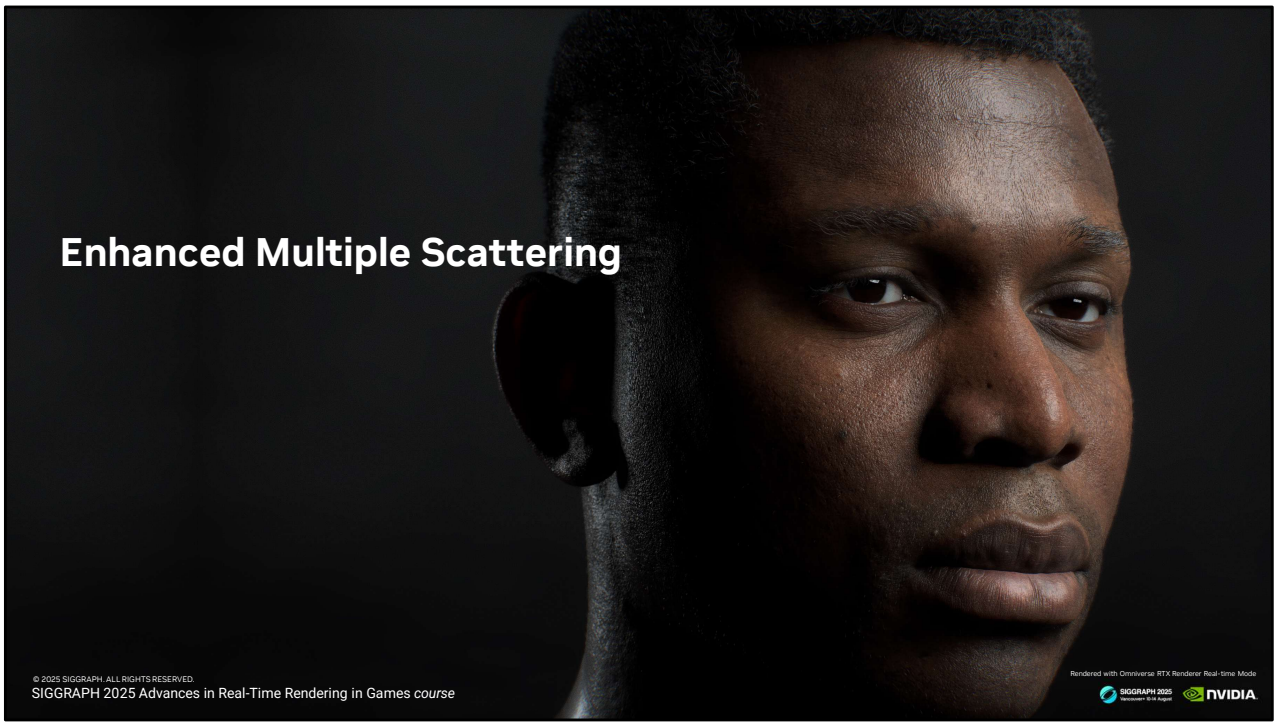
$$t' = \frac{-\ln(1 - \xi(1 - \exp(-\sigma_t d)))}{\sigma_t}$$

$$p1(t) = \frac{\sigma_t \exp(-\sigma_t t)}{1 - \exp(-\sigma_t d)}$$

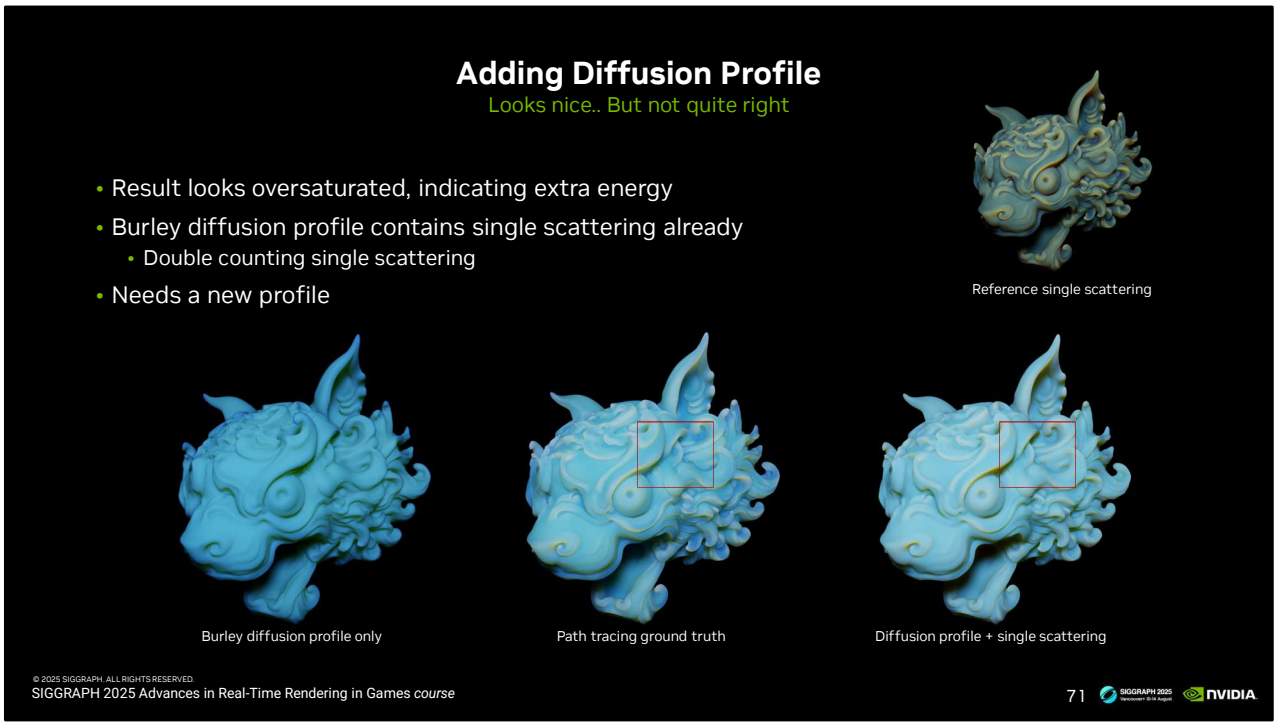


Althernatively, if reducing variance is more important, we can instead, cast a ray with infinite as TMax to solve d_1.
Then we can write our estimator without t_1.

So if you do everything correctly, here is what you would get. The image on the left side is rendered without any cache, so it almost look identical to the ground truth. On the other hand, you might see some subtle difference here and there comparing to the right picture, either way, the point is the single scattering should match very close to the ground truth.



Alright, so we've handled the first bounce with full fidelity. Now we need to add back the multiple scattering component—everything beyond that first scatter. This is where we bring in the diffusion profile approach.



Let’s continue with the same lion head example we saw. When we first tried adding diffusion profiles to our single-scattered results, something looked wrong—the images appeared over-saturated and too bright. The issue is that Burley's diffusion profile inherently includes single-scattering contributions since it captures all orders of scattering. So by adding our explicit first-bounce on top, we were effectively double-counting the single scattering component.

Single Scattering Diffusion Profile

- Given Burley's diffusion profile $R(r)$
- Energy conservation – $R_n(r) = R(r) - R_1(r)$
  - $R_1(r)$ – Single scattering profile
  - $R_n(r)$ – Multiple scattering (only) profile

With energy conservation, We know that single scattering plus all higher-order scattering should sum to the full profile. So to get a multiple-scattering-only profile, we can construct a single scattering profile and subtract it from the full diffusion profile.

Single Scattering Diffusion Profile

- The radiance after exact n scattering event can be found via Taylor series expansion [d'Eon 2022]
  - over single scattering albedo α, at $\alpha = 0$
- Rewrite the profile as a function of α

$$R_1(\alpha) = R(\alpha|1)$$

The mathematical foundation is that scattering events can be expanded as a Taylor series over single scattering albedo. Each scattering event multiplies by an albedo term, so n scatters corresponds to albedo raised to the nth power. For this expansion, we need to consider diffusion profiles as functions of alpha rather than just radius.

Single Scattering Diffusion Profile

- Given Burley's diffusion profile $R(r)$ [Burley 2016]
  - ℓ - mean free path $\ell = 1/\sigma_t$
  - r – radius, distance from origin
  - A – Surface albedo
  - α – Single scattering albedo

$$R(r) = As\frac{e^{-rs/\ell} + e^{-sr/(3\ell)}}{8\pi\ell r}$$

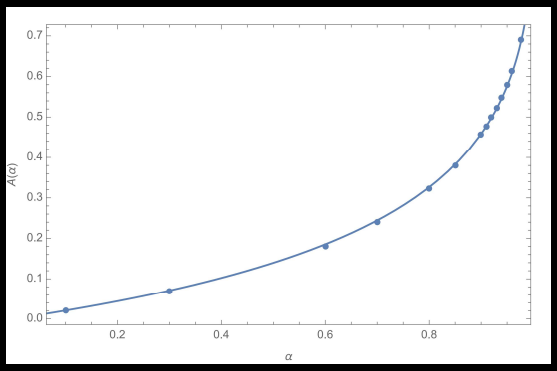
$$s(\alpha) = 1.85 - A + 7|A - 0.8|^3$$

We start with these formulations from Burley's original paper, making the necessary parameter substitutions. To make it easier to read, I highlighted the variable with color, you can consider everything else are just some constant values in this context.
The profile R(r) consists of capitalized A and s, and we would like to express both A and s in the form of alpha. So, this boils down to how to express capitalized A as form of alpha.

Single Scattering Diffusion Profile

- Run Monte Carlo simulation under a custom configuration [d'Eon 2022]
  - Assuming diffuse transmission
  - 1.4 IOR
  - Semi-infinite flat slab
- Curve fitting to approximate relationship $A(\alpha)$

$$A(\alpha) = \frac{1 - \sqrt{1 - \alpha}}{1.554\sqrt{1 - \alpha} + 1}$$



To complete this transformation, we run our own Monte Carlo simulation to find the relationship between surface albedo A and our alpha parameter.

Single Scattering Diffusion Profile

- Given Burley's diffusion profile $R(r)$
  - ℓ - mean free path $\ell = 1/\sigma_t$
  - r – variable
  - A – Surface albedo
  - α – Single scattering albedo

$$R(r) = As\frac{e^{-rs/\ell} + e^{-sr/(3\ell)}}{8\pi\ell r}$$

$$s(\alpha) = 1.85 - A + 7|A - 0.8|^3$$

$$A(\alpha) = \frac{1 - \sqrt{1-\alpha}}{1.554\sqrt{1-\alpha} + 1}$$

```
In[61]:= (* Define A[α] *)
         A[α_] := (1 - Sqrt[1 - α]) / (1.554 * Sqrt[1 - α] + 1)

         (* Define s[α] *)
         s[α_] := 1.85 - A[α] + 7 * Abs[A[α] - 0.8]^3

         (* Define R[α] as an expression involving symbolic r and l *)
         R[α_] := A[α] * s[α] * (Exp[-r * s[α] / l] + Exp[-s[α] * r / (3 l)]) / (8 * Pi * l * r)

         (* solve talyor series *)
         Normal[SeriesCoefficient[R[α], {α, 0, 1}, Assumptions → r > 0 && 0 < α < 1]]

Out[64]= (0.0423281 e^(-1.81133 r / l) (1. + e^(-3.62267 r / l))) / (r l)
```


With the additional formula, we can represent S as a function of alpha. Plugging everything into Mathematica produces this clean analytical result.

Copy the following code into mathematica
https://www.wolfram.com/language/index.php.en

```
(* Define A[\[Alpha]] *)
A[\[Alpha]_] := (1 - Sqrt[1 - \[Alpha]]) / (1.554*Sqrt[1 - \[Alpha]] + 1)

(* Define s[\[Alpha]] *)
s[\[Alpha]_] := 1.85 - A[\[Alpha]] + 7*Abs[A[\[Alpha]] - 0.8]^3

(* Define R[\[Alpha]] as an expression involving symbolic r and \[ScriptL] *)
R[\[Alpha]_] := A[\[Alpha]]*s[\[Alpha]]*(Exp[-r*s[\[Alpha]]/\[ScriptL]] + Exp[-
s[\[Alpha]]*r/(3 \[ScriptL])]) / (8*Pi*\[ScriptL]*r)

(* solve talyor series *)
Normal[SeriesCoefficient[R[\[Alpha]], {\[Alpha], 0, 1},Assumptions-
>r>0&&0<\[Alpha]<1]]
```

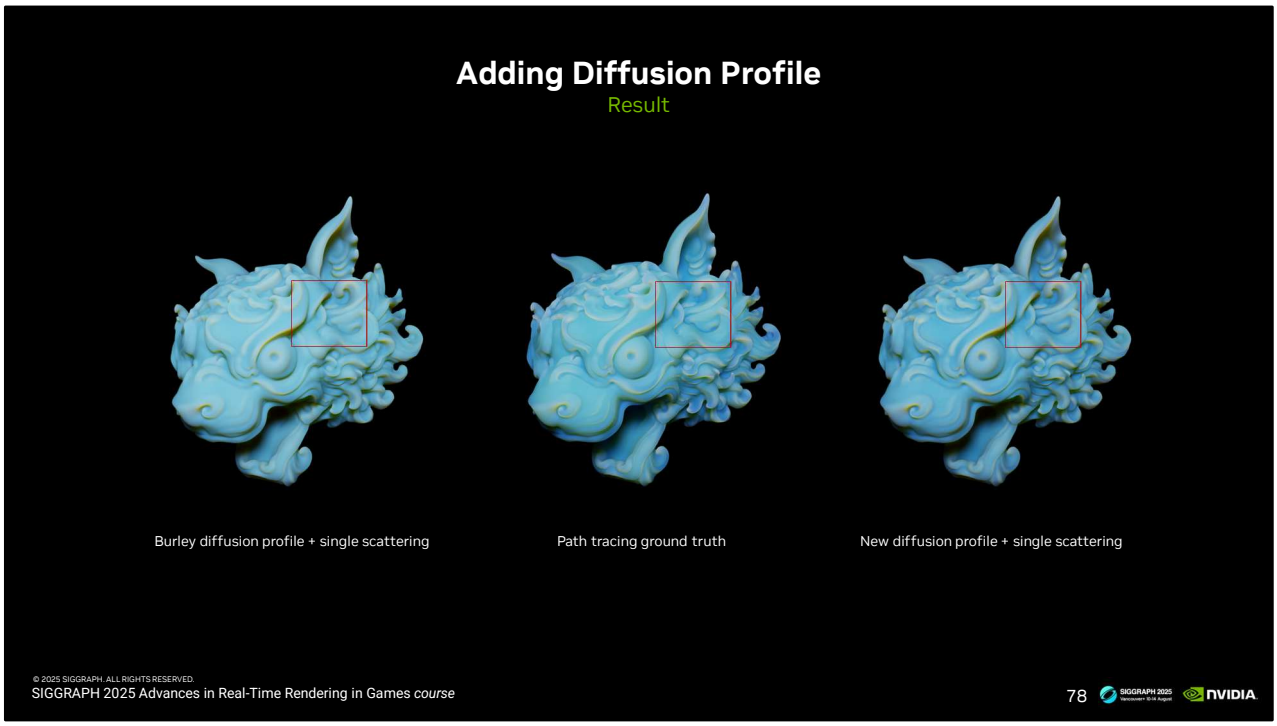
Single Scattering Diffusion Profile

- Given Burley's diffusion profile $R(r)$
- Energy conservation – $R_n(r) = R(r) - R_1(r)$
  - $R_1(r)$ – Single scattering profile
  - $R_n(r)$ – Multiple scattering (only) profile

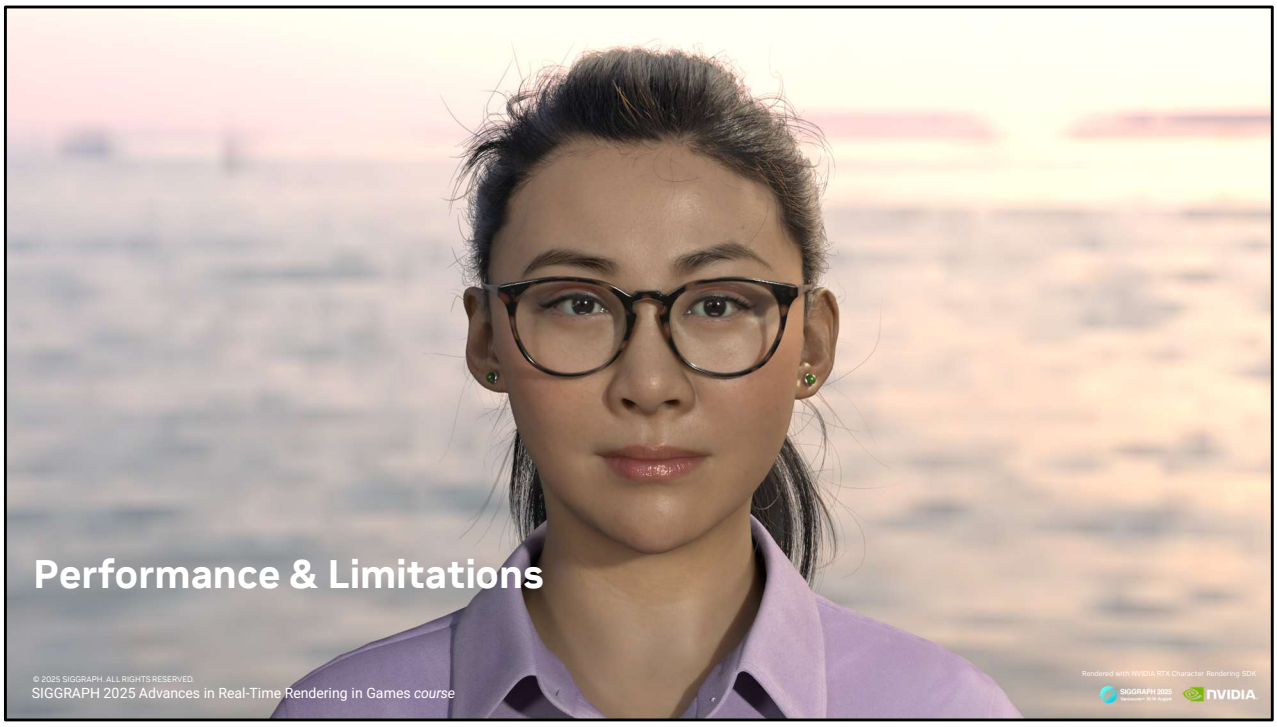
$$R(r) = As\frac{e^{-rs/\ell} + e^{-sr/(3\ell)}}{8\pi\ell r}$$

$$R_1(r) = \frac{0.0423281\left(e^{-5.434\ /\ell} + e^{-1.8113\ \ /\ell}\right)}{\ell r}$$

Now with our single scattering diffusion profile in hand, we subtract it from the original full profile to get our multiple scattering profile.



The results with this new diffusion profile show much better ground truth matching.

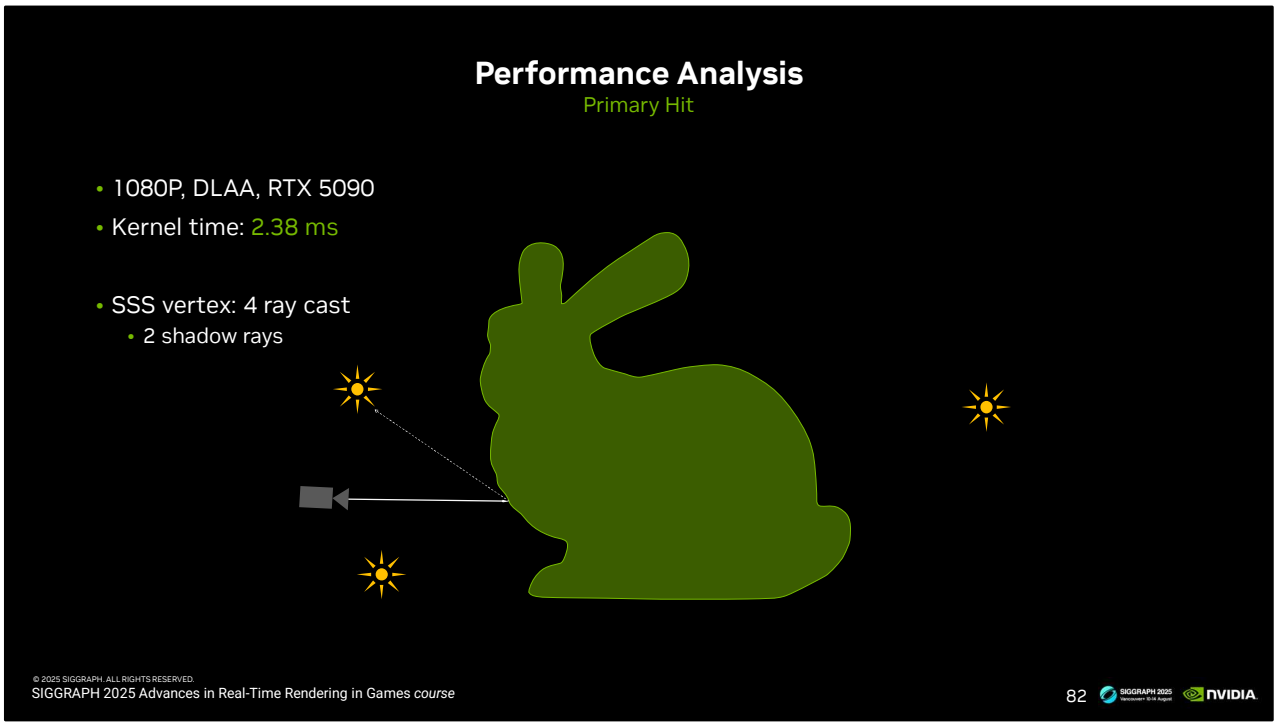


And that concludes our algorithm description. Now let's talk about performance.

Performance Analysis

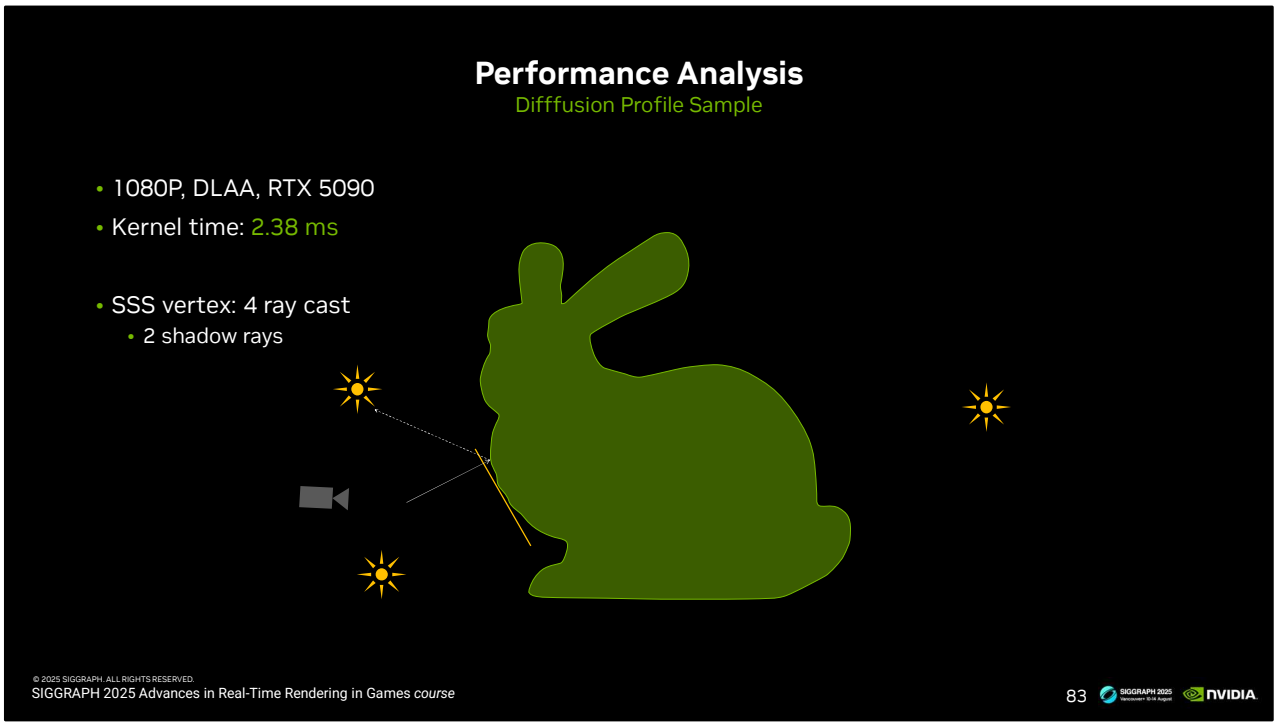
- 1080P, DLAA, RTX 5090
- Kernel time: 2.38 ms
- SSS vertex: 4 ray cast
  - 2 shadow rays
- 13.74M triangles
  - 7M groom

Our open-source sdk implements the whole algorithm within a real-time path tracer, meaning the frame rendered through a path trace kernel, and then we use a denoiser to denoise the final result.
The SSS vertices costs 4 rays per pixel, where 2 rays are shadow rays. This scene has almost 14 million triangles, and for this exact shot, the main rendering kernel costs 2.38ms.



I would like to explain the whole routine to give you a clear picture of the workload, so let's pretend our bunny is the lady we saw for a second...
We start from the camera and trace primary ray, upon primary hit, we perform next event estimation with a shadow ray. Then we sample bsdf events, if we manage to sample transmission event, SSS happens.

Also note that the dot line arrows all represent the shadow rays, which are rays marked as ACCEPT_FIRST_HIT_AND_END_SEARCH.



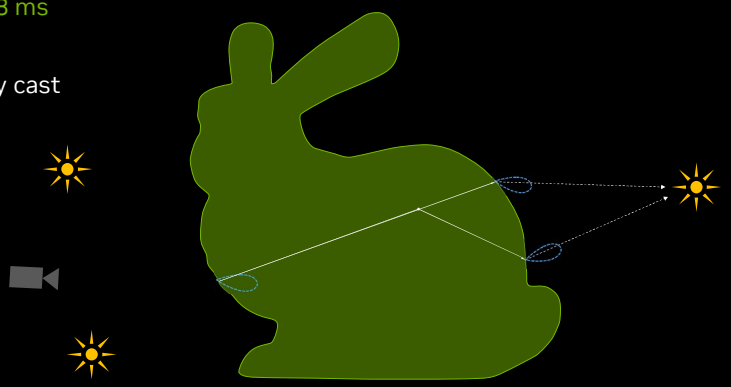
The kernel then sample the radius for diffusion profile and cast a ray towards geometry to retrieve a geometry sample. From that point, it performs an explicit light sample to complete the diffusion profile sample.

Performance Analysis

Single Scattering

- 1080P, DLAA, RTX 5090
- Kernel time: 2.38 ms

- SSS vertex: 4 ray cast
  - 2 shadow rays



After that, it goes through the single scattering routine we discussed before, which cost 2 regular rays and 2 shadow rays.

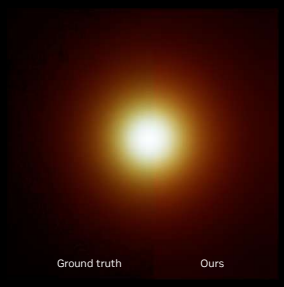
Performance Analysis

- 1080P, DLAA, RTX 5090
- Kernel time: 2.38 ms
- SSS vertex: 4 ray cast
  - 2 shadow rays
- 13.74M triangles
  - 7M groom

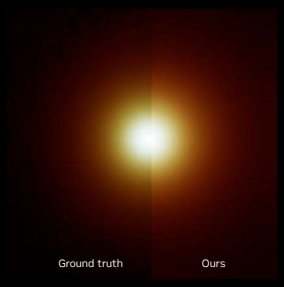
Some more details about the scene here. Although we terminate the path after SSS, the reflection GI can go up to 8 bounces, in this scene, this helps to render the hair.
Thanks to the path tracing nature, this implementation could handle indirect SSS event with refraction. For example, part of the face is behind the glasses. And also pay attention to those eyes here, the corneas are separate glass geometry layers, and right behind, we have the choroids which are modeled with a SSS material.

Limitation

- Still suffer from the limitation of diffusion profile
  - Ground truth color mismatch with complicated layered materials

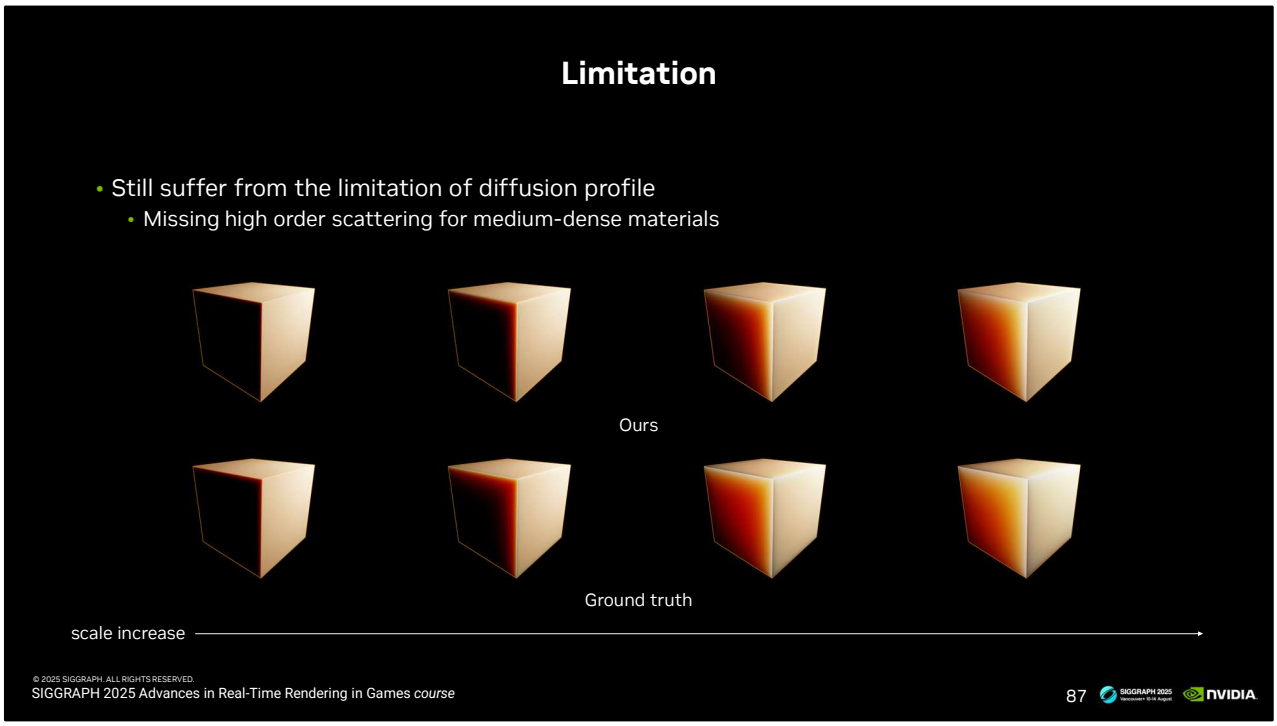


Diffuse Transmission



Diffuse Transmission
Specular reflection, 0.25 roughness

Unfortunately, we are still away from solving SSS. Although we are quite happy with the quality, we can still spot some major difference if we compare it against the random walk ground truth.
Because how diffusion profile is built on very specific material configuration, the color mismatch could happen with different material layer model. For example here, I added a reflection lobe, which causes the irradiance cannot be trivially sampled, and the weight of diffuse transmission becomes incoming direction dependent.



On the other hand, materials with medium range density relying on multiple scattering to have those very translucent appearance. The application of multiple scattering diffusion profile means we miss some of those energy. Notice in the pictures, the left one and the right one all matche closely while the middle ones lack some of the color bleed.

This brings us to a fundamental point: rendering is essentially a sampling problem. We need to sample light sources for correct irradiance and solve visibility terms, which requires understanding the underlying geometry. For SSS, this geometric understanding becomes even more critical.

A Sample Challenge

- Light sampling – (Ir)radiance estimation
  - Area light sampling
  - Many light sampling
  - Irradiance estimation
- Geometry Sampling
  - Cast rays, sample distance
  - How fast/accurately renderer can retrieve the geometry knowledge
    - The original integration contains a domain on geometry (parameterized by t)
- The better you can sample those terms, the better quality you get

While our algorithm flattens multiple bounce random walks into a consistent process with determined cost, the overall quality really scales with the capabilities of sampling both lighting and geometry. We have to answer many questions like: How do we sample area lights? How do we handle many lights? Do we have budget for hardware ray tracing? If not, what about screen space ray tracing?

(Ir)radiance Estimation

- Reuse any existing lighting techniques from main pipeline
  - LTC irradiance [Heitz 2016] + ratio shadow estimator [Heitz 2016]
  - Sampled lighting: sampled irradiance estimate
    - Handle area light, many light
    - 1 sample MC estimation: sampledRadiance * cosWeight / lightSamplePdf
  - Improve sample quality: e.g. ReSTIR

The algorithm itself does not specify to have to do any of those, it can be implemented to reuse whatever the renderer have and getting improved with the underlying sample quality.

When assuming a diffuse transmission bsdf, irradiance is the integral of radiance over hemisphere, which is widely used for diffusion profile.
For every sample of diffusion profile, we require 1 sample of irradiance from a target pixel.

Then recall what's described in the single scattering algorithm, we needs 2 radiance samples per path traced single scattering sample.

LTC provides direct formula of computing irradiance without accounting for visibility term, combining it with a ratio shadow estimator, it can be used as one approach to estimate irradiance.

Sample based lighting has becoming popular in the recent years to handle the many light sampling problem, in which case, we compute noise "radiance" samples. For every sample, we combine mechanisms to select a specific light from the scene, then optionally sample a point from the area light primitives.

This resulting a pdf for such light sample. Any engine does similar sample based lighting could estimate irradiance by doing 1 sample MC integration.

We will touch more on ReSTIR in the next slide, but for completeness, any further ways to improve the irradiance quality will be beneficial for implementation.

ReSTIR Recap

RIS sampling

- Importance Sampling: $\langle I \rangle = \frac{f(y)}{p(y)}$
- Resampled Importance Sampling:
  - Generate M candidate samples, $X_1, X_2, X_3 \ldots$
  - Pick Target function $\hat{p}$ (cheaply approximate f)
  - Evaluate resampling weight: $w_i = \frac{1}{m}\frac{\hat{p}(x_i)}{p(x_i)}$
- Select one sample Y according to w_i, compute $f(Y)$
- $\langle I \rangle_{\text{RIS}} = f(Y)\frac{1}{\hat{p}(Y)}\sum_{i=1}^{M} w_i$

Before we discuss the specific of ReSTIR, Let me quickly cover the basics. To sample an unknown distribution f with importance sampling, we can perform RIS—generate M samples through an approximate target function p_hat, then select one sample Y, and use Y to compute f weighted by the RIS weight.

ReSTIR Recap

ReSTIR - Spatiotemporal reservoir resampling

- Importance Sampling: $\langle I \rangle = \frac{f(y)}{p(y)}$
- Resampled Importance Sampling:
  - Generate M candidate samples, $X_1, X_2, X_3 \ldots$
  - Pick Target function $\hat{p}$ (cheaply approximate f)
  - Evaluate resampling weight: $w_i = \frac{1}{M} \frac{\hat{p}(x_i)}{p(x_i)}$
- Select one sample Y according to w_i, compute $f(Y)$
- $\langle I \rangle_{\text{RIS}} = f(Y) \frac{1}{\hat{p}(Y)} \sum_{i=1}^{M} w_i$

Put samples into a "reservoir"
Spatial/temporal reuse

-> ReSTIR

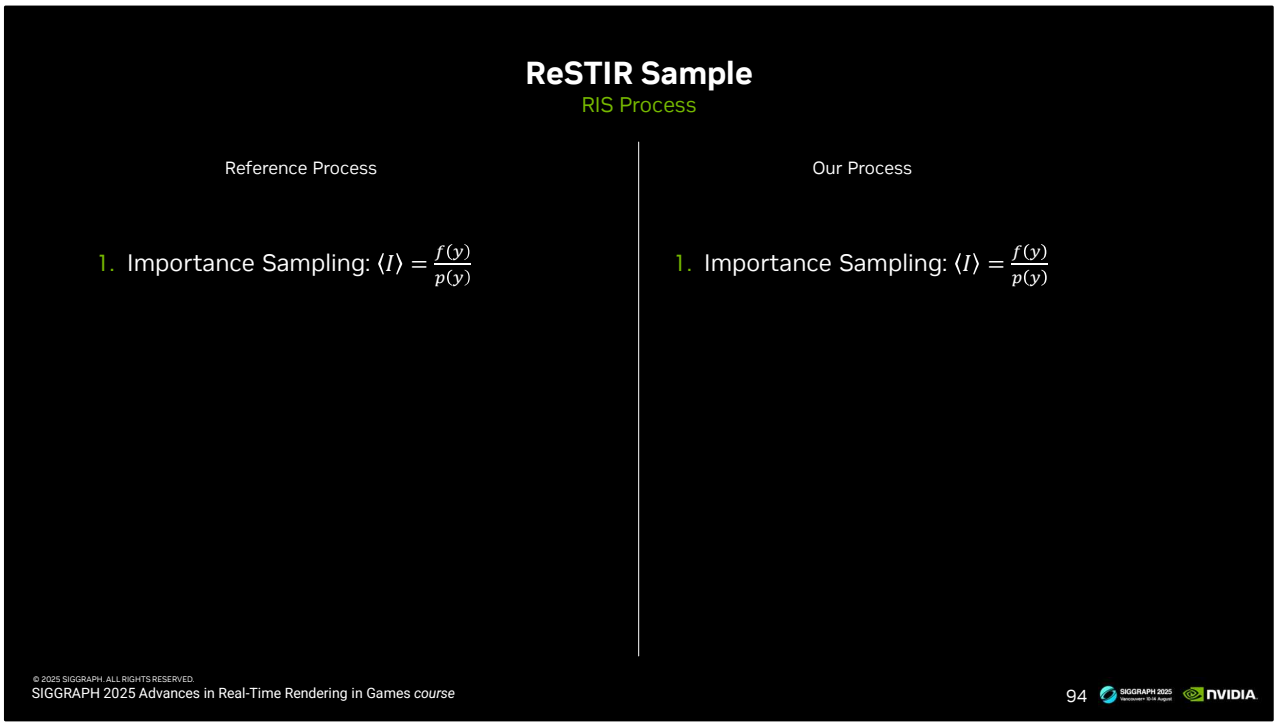
ReSTIR extends this RIS process by streaming samples into fixed-size reservoirs while reusing samples from spatial neighbors and previous frames.

ReSTIR Sample

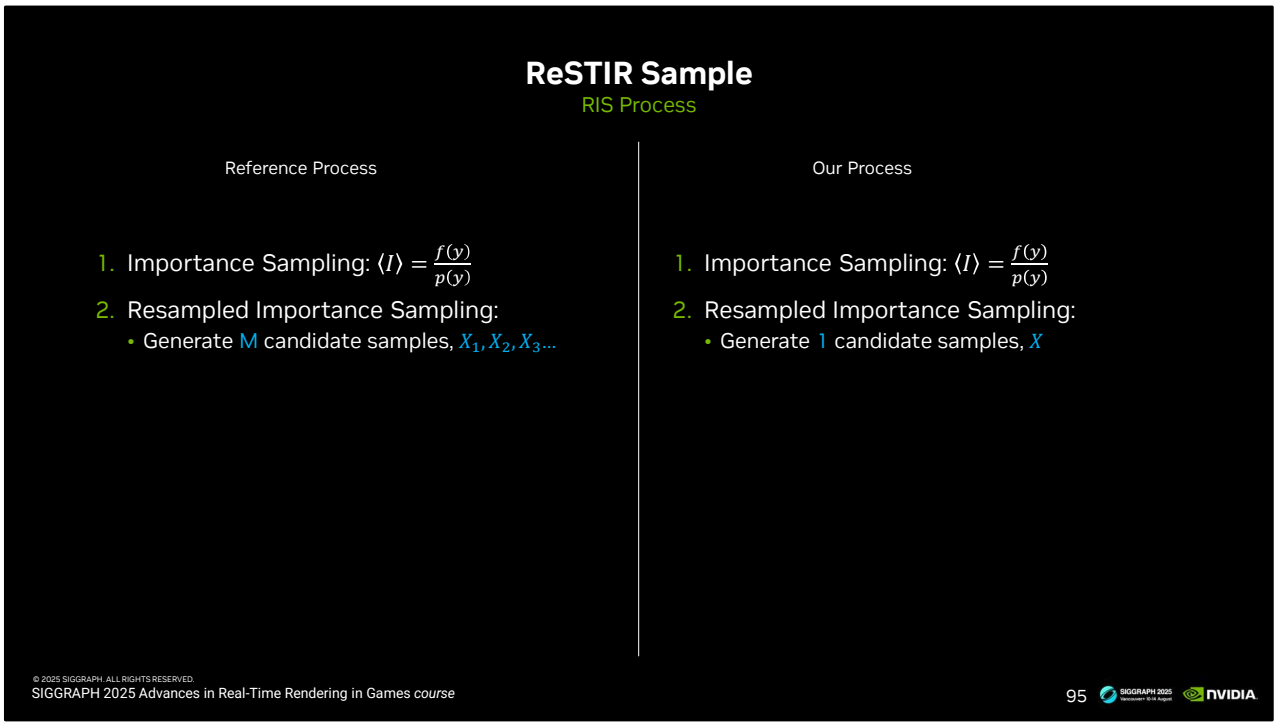
Key design

- Challenge of the algorithm: which direction to shoot ray?
- Goal
  - Minimal overhead – Every ray counts
  - Simple to implement
- Two dominant factors:
  - radiance strength (L)
  - Scattering distance (t)
- decide refraction direction upon entering volume

For subsurface scattering, the central challenge is determining optimal ray directions within volumes. Two factors guide this decision: radiance strength and scattering distance. We adopt a target function based on the boundary term, which is particularly beneficial because it requires no extra computation when sampling from our reservoirs.



Here's our process: Step one is the same, we have the same goal. specifically, f represents the boundary term. Y represents the sample direction



Step 2, first, we generate 1 sample, which means M equals to 1, where X represents the refraction directions.

ReSTIR Sample

RIS Process

Reference Process

1. Importance Sampling: $\langle I \rangle = \frac{f(y)}{p(y)}$
2. Resampled Importance Sampling:
   - Generate M candidate samples, $X_1, X_2, X_3 \ldots$
   - Pick Target function $\hat{p}$ (cheaply approximate f)

Our Process

1. Importance Sampling: $\langle I \rangle = \frac{f(y)}{p(y)}$
2. Resampled Importance Sampling:
   - Generate 1 candidate samples, X
   - Pick Target function $\hat{p} = f = \exp(-\sigma_t t)\, L$

Then we use the boundary term f, to be our p_hat.

ReSTIR Sample

RIS Process

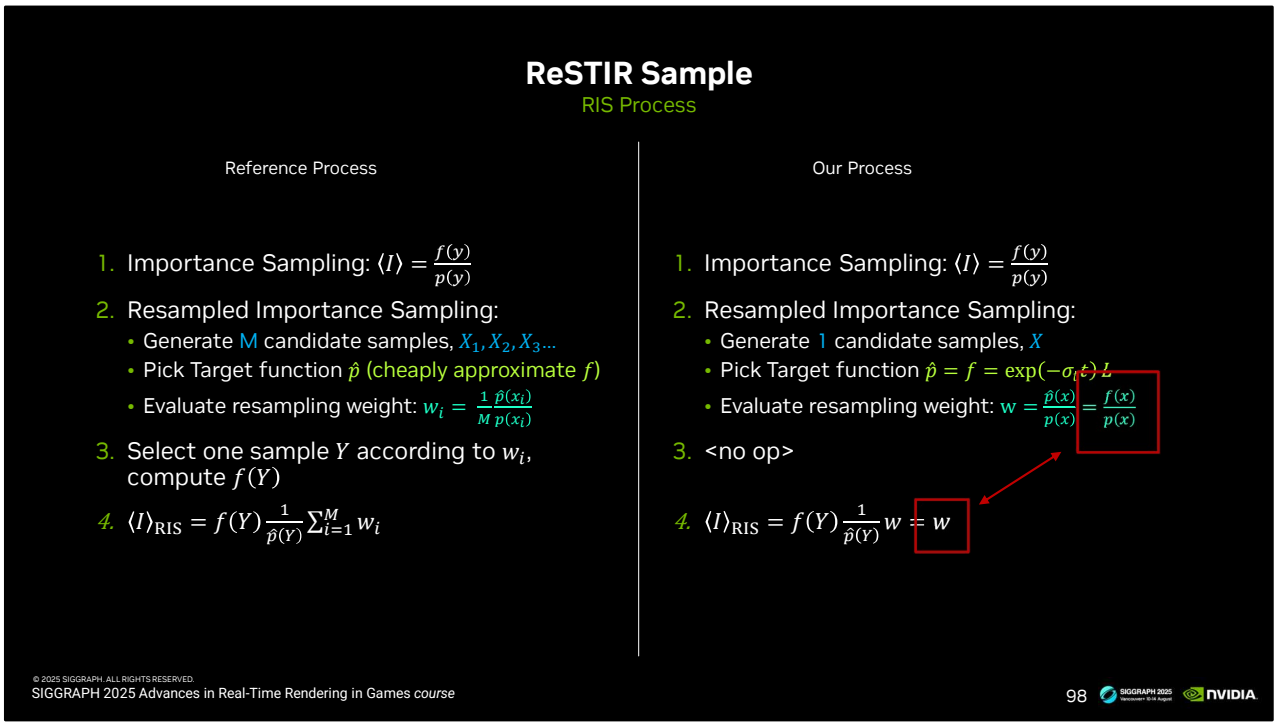
Reference Process

1. Importance Sampling: $\langle I \rangle = \frac{f(y)}{p(y)}$
2. Resampled Importance Sampling:
   - Generate M candidate samples, $X_1, X_2, X_3 \ldots$
   - Pick Target function $\hat{p}$ (cheaply approximate f)
   - Evaluate resampling weight: $w_i = \frac{1}{M} \frac{\hat{p}(x_i)}{p(x_i)}$

Our Process

1. Importance Sampling: $\langle I \rangle = \frac{f(y)}{p(y)}$
2. Resampled Importance Sampling:
   - Generate 1 candidate samples, X
   - Pick Target function $\hat{p} = f = \exp(-\sigma_t t)\, L$
   - Evaluate resampling weight: $w = \frac{\hat{p}(x)}{p(x)} = \frac{f(x)}{p(x)}$

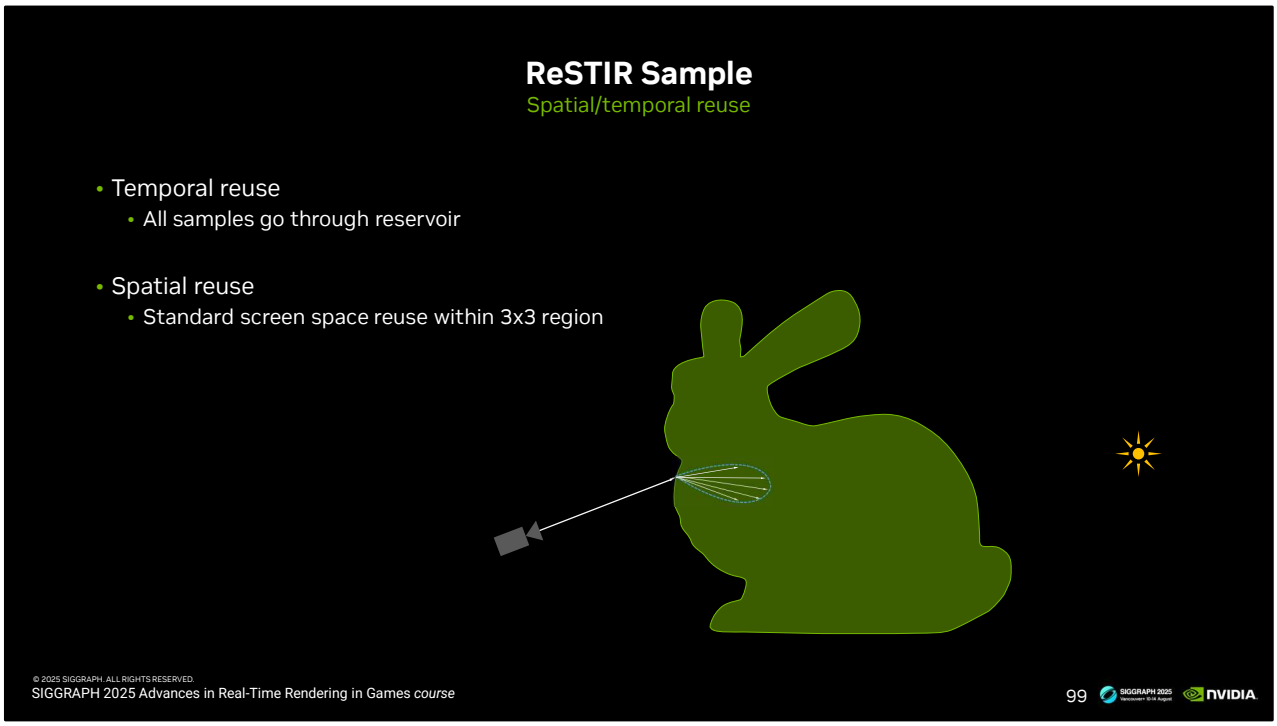
Then we plug in M equals to 1, p_hat equals to f. We get a very important formula, this is exactly what we want to solve.



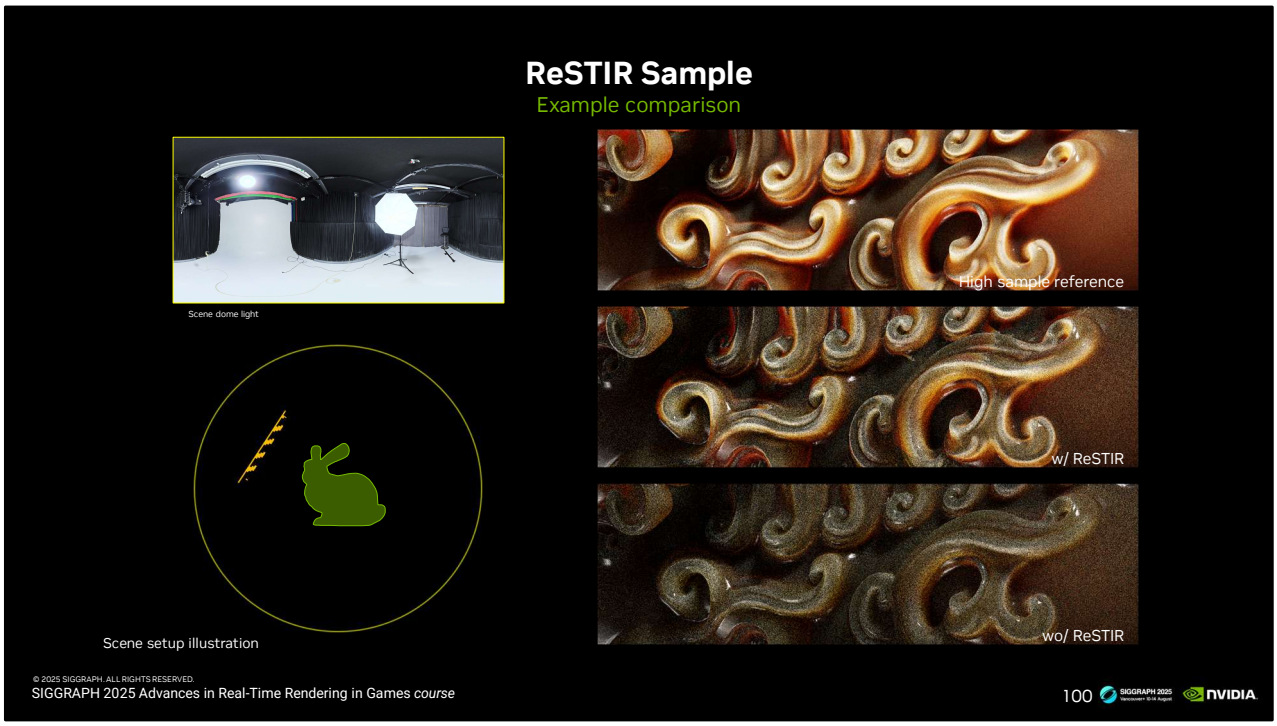
Then finally, let's look at step 4 first, we notice that the w is exactly what we need for the sample. This eliminate the process of using the direction to reevaluate f.

Let's go through the whole process all together.
First, X and Y represent refraction directions.
After entering the volume, we sample boundary terms. But instead of using such sample directly, we feed it into the reservoir, which is the step 2 here. We then read the reservoir weights as our sample results, which is step 4. This works elegantly because p_hat equals f as shown in step 2, so everything cancels out perfectly.

Without this approach, we'd need an extra ray for step 3 so that we can evaluate the geometry distance t.



This routine has a constant overhead of just 1 boundary term sample, costing 1 extra ray per frame. For example, for this bunny, we launch the pass, sample transmission BSDF once, cast the ray, compute boundary terms, feed the reservoir, then loop over 5 samples repeating this process when the setting requests 5 transmission bsdf samples. Every transmission BSDF sample effectively counts as 1 temporal reuse, and we perform standard spatial reuse in screen space with simple culling.



In this scene, we have an object illumnicated by a rectangular light and a texture mapped dome light.
The results speak for themselves—our specialized ReSTIR approach substantially reduces visual noise with only 1 sample and adds 1 ray constant overhead.

(Scene dome light texture from https://polyhaven.com/a/studio_small_03, under CC0 licence)

(Ir)radiance Estimation

- Reuse any existing lighting techniques from main pipeline
  - LTC irradiance [Heitz 2016] + ratio shadow estimator [Heitz 2016]
  - Sampled lighting: sampled irradiance estimate
    - Handle area lights, many lights
    - 1 sample MC estimation: sampledRadiance * cosWeight / lightSamplePdf
  - Improve sample quality: e.g. ReSTIR
- Caching:
  - Screen space
  - Probes based
  - Hash grid

Finally, for radiance estimation, we can deploy caching mechanisms for performance or quality improvements. For instance, we implement a back face irradiance screen space cache to capture irradiance distribution on the back side of geometry. This is especially helpful when drawing multiple samples, though in practice we only need 1 sample since DLSS can denoise the signal effectively.

Extra list

- Demodulate single-scattering albedo to preserve details
- How to sample distance based on float3 sigma_t (build channel CDF)
- Check out our open-source vendor agnostic implementation through NVIDIA RTX Character Rendering [Rtxcr 2025]

Access open-source implementation
https://github.com/NVIDIA-RTX/RTXCR

I have two bonus slides that explains some practical tricks. At the same time, make sure to check out our character rendering sdk for an open source reference implementation!

Demodulate Single-scattering albedo

1. Divide the signal by single scattering albedo
2. Feed the signal into denoiser
3. Multiply the denoised signal by single scattering albedo

- Very helpful for preserving the texture details
- Not necessary with DLSS-RR

This slide discuss a simple trick to preserve better texture details for denoised result. Before we adopted DLSS-RR, we used separate denoisers to to denoise individual signal. It really helped to preserve complicated texture details to denoise the sigal demodulated by the single scattering albedo, then multiply it back before it's fed to the DLSS.

RGB variation in σ_t

- σ_t is float3
- distance t is float. Example: $t' = -\ln(1-\xi)/\sigma_t$

- Simple strategy: randomly pick 1 channel.
- Better strategy: Select channel with a CDF.

```
const float3 sigmaTNormd = sigmaT * ssAlbedo / dot(sigmaT, ssAlbedo);
const float2 channelCDF = float2(sigmaTNormd.x, sigmaTNormd.x + sigmaTNormd.y);
int channel = 0;
float p = sampleUniformRng(rndSampler);
if(p >= channelCDF.x)
{
    if(p < channelCDF.y)
    {
        channel = 1;
    }
    else
    {
        channel = 2;
    }
}

const float sigma = sigmaT[channel];
const float3 pdf = dot(sigmaTNormd, sigmaT * exp(-sigmaT * t));
```


While sampling from diffusion profile or sampling distance, you would need a scale value for sigma. This requires you to either bias toward 1 channel, or have a routine to sample those channel too.
A good alternative is to apply hero tracking for chromatic RGB values – This implies you would need to apply an extra pdf term within your estimator.

Finally, we would like to thank all the colleagues and industry friends that helped us along this journey, which made this presentation possible. Also, we are grateful for the graphics community where we found abundance of information for the development. Finally, I also want to express my gratitude towards our amazing artists who made all those beautiful assets.

Reference

1. [Sketchfab 2025] Gummy bears - by Robert Kotsch (@robertkotsch), Sketchfab. Available at: https://sketchfab.com/3d-models/gummy-bears-4681d497333741ce8f87e3e8ba6ec095 (Accessed: 18 July 2025).
2. [Fong 2017] Julian Fong, Magnus Wrenninge, Christopher Kulla, and Ralf Habel. 2017. Production volume rendering: SIGGRAPH 2017 course. In ACM SIGGRAPH 2017 Courses (SIGGRAPH '17). Association for Computing Machinery, New York, NY, USA, Article 2, 1–79. https://doi.org/10.1145/3084873.3084907
3. [Pbrt 2016] Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. Physically Based Rendering: From Theory to Implementation (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
4. [Golubev 2018] Evgenii Golubev. 2018. Efficient screen-space subsurface scattering using Burley's normalized diffusion in real-time. Retrieved Aug 29, 2019 from http://advances.realtimerendering.com/s2018/Efficient%20screen%20space%20subsurface%20scattering%20Siggraph%202018.pdf
5. [Golubev 2019] Evgenii Golubev. 2019. Sampling Burley's Normalized Diffusion Profiles. Retrieved Nov 17, 2019 from https://zero-radiance.github.io/post/sampling-diffusion/
6. [Xie 2020] Tiantian Xie, Marc Olano, Brian Karis, and Krzysztof Narkowicz. 2020. Real-time subsurface scattering with single pass variance-guided adaptive importance sampling. Proc. ACM Comput. Graph. Interact. Tech. 3, 1, Article 3 (Apr 2020), 21 pages. https://doi.org/10.1145/3384538
7. [Burley 2016] Approximate Reflectance Profiles for Efficient Subsurface Scattering. Per H. Christensen, B Burley."An approximate reflectance profile for efficient subsurface scattering." ACM SIGGRAPH (2015): 1-1
8. [Kulla 2017] Kulla, C. and Conty, A. Revisiting Physically Based Shading at Imageworks. 139.
9. [Heitz 2016] Eric Heitz, Jonathan Dupuy, Stephen Hill, and David Neubelt. 2016. Real-time polygonal-light shading with linearly transformed cosines. ACM Trans. Graph. 35, 4, Article 41 (July 2016), 8 pages. https://doi.org/10.1145/2897824.2925895
10. [Heitz 2018] Eric Heitz, Stephen Hill, and Morgan McGuire. 2018. Combining analytic direct illumination and stochastic shadows. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '18). Association for Computing Machinery, New York, NY, USA, Article 2, 1–11. https://doi.org/10.1145/3190834.3190852
11. [d'Eon 2022] Eugene d'Eon 2022 A Hitchhiker's Guide to Multiple Scattering v0.3.2 graphics notation, page 106 https://eugenedeon.com/hitchhikers
12. [Rtxcr 2025] Nvidia-RTX/RTXCR: Nvidia RTX character rendering SDK, GitHub. Available at: https://github.com/NVIDIA-RTX/RTXCR (Accessed: 18 July 2025).

This concludes my presentation, thank you very much! Next, question time!