Path Tracing for Future Games (??!!?)

Matt Pharr, NVIDIA

It's always fun to talk in open problems—you don't have to have all the answers and get to ask provocative questions.

Today we'll dig into all sorts of things that I have no idea how to do.

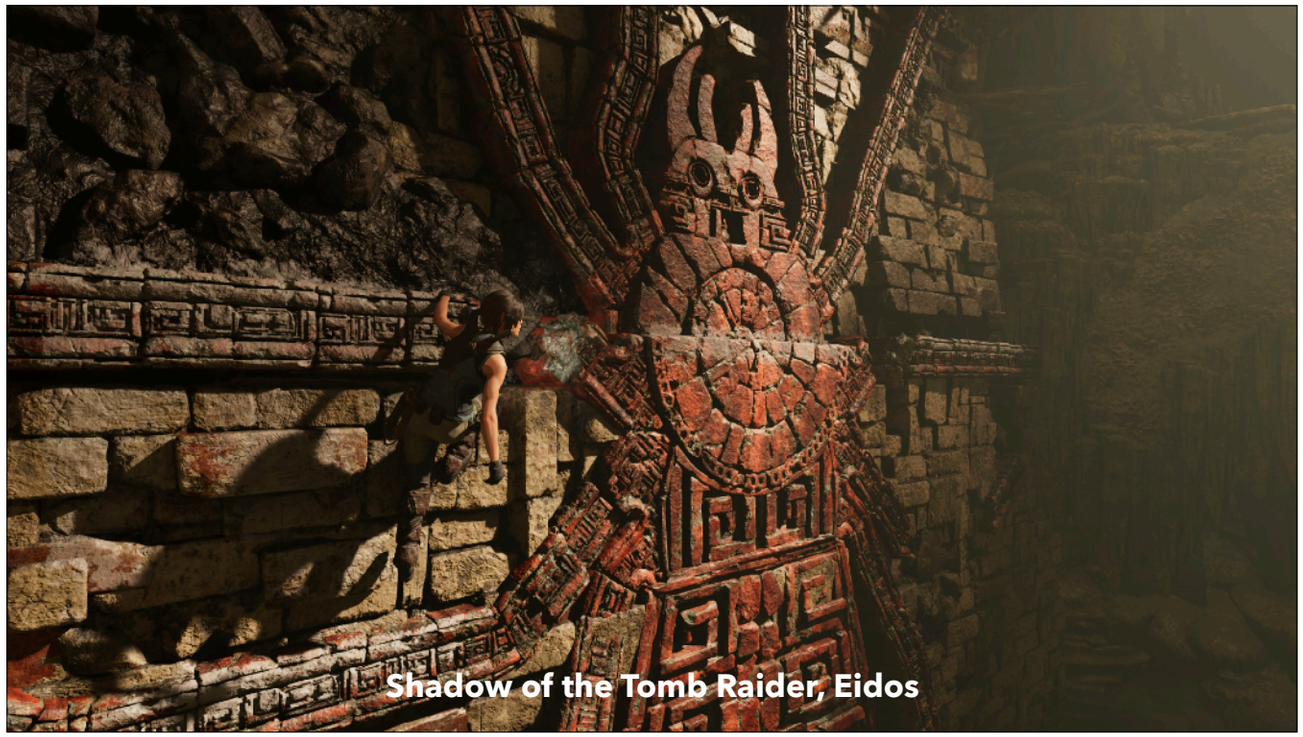
But they're also things that no one has an idea to do, but they're important—solving them can help get us to what would be a pretty amazing place—going all the way to path tracing for games.

It's not going to be easy, especially as the other talks in this session have shown, but it'd be pretty incredible to get there.

As everyone knows, ray tracing hardware is out there now.

People are already doing really neat things with it.

This image pair shows one of my favorite examples. This one is done with raster shadows…



…and this is done with ray tracing.

On one hand, it’s subtle. But if you flip back and forth between the images, it’s incredible what a difference it makes to have highly accurate shadows.

You get all these self-shadowing effects that are a big part of why rough surfaces look the way they do in real life.

It’s hard to get that level of shadow accuracy with raster, but it comes naturally from ray tracing.

More generally, now we’ve got a high-performance, accurate visibility primitive available for real-time.

And of course, there's also ray traced ambient occlusion and GI—dynamic lighting in dynamic scenes.

With ray tracing, you can do that more robustly than before: no need for baking or for screen-space light transport.


Metro Exodus, 4A Games

And everyone loves shiny things…
Battlefield V has got lots of nice specular ray traced effects.

These images are beautiful and super exciting, but they also illustrate individual ray traced effects added to a raster engine.
That's the right thing for most things shipping today, given the need to target hardware that doesn't support ray tracing well and given existing investments in raster engines.
But what about going further…

Path traced Quake 2 is a remarkable effort—it's gone all the way to path tracing, for everything.

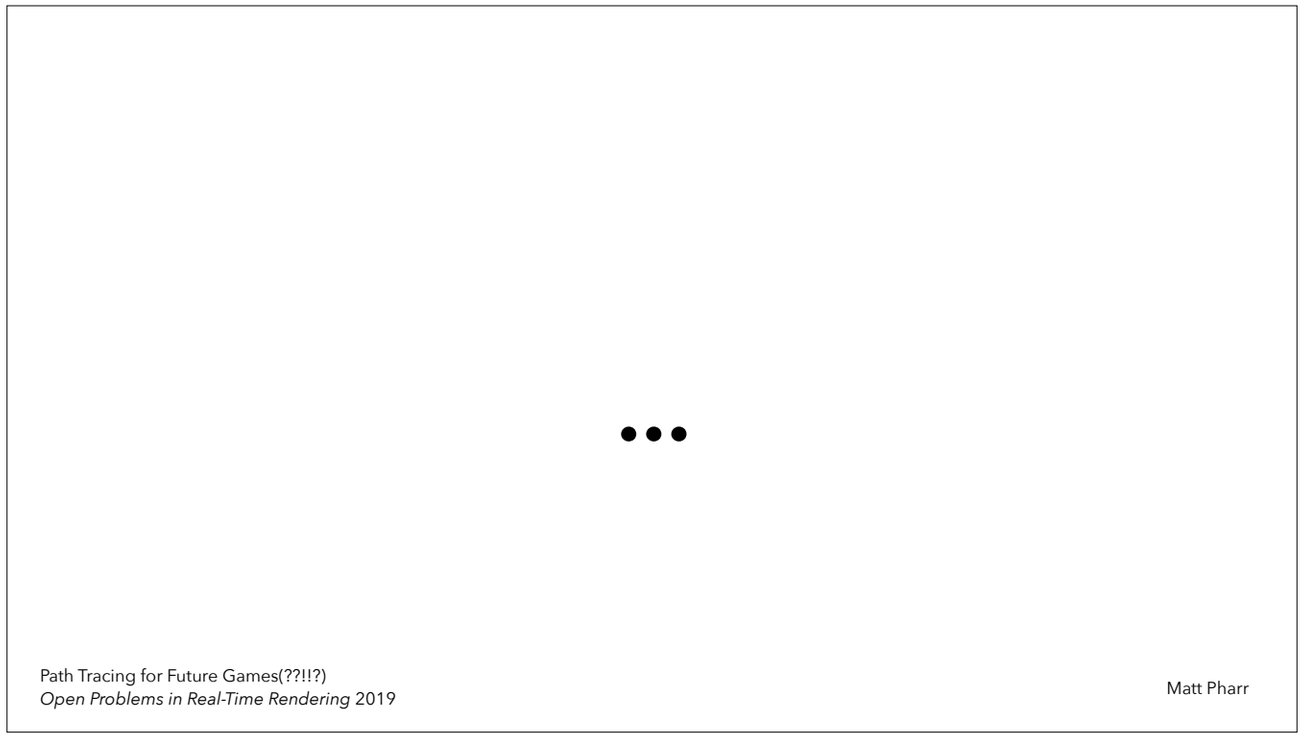
Sure, it's 20 year old content and not quite up to the complexity of modern AAA titles, but yet, wow, it looks amazing.

Ray tracing underlies all of the light transport in this image.

And it was done by a small team: the first iteration, Q2VPT by two super talented graduate students, and then a small team that took that and made Quake 2 RTX.

And then there's ray traced Minecraft, done by a single developer.
[And now, Feb 2020, there's Minecraft RTX.]
Again, there's one core algorithm that does everything: all BSDFs, all types of lighting—point, area, indirect…

I think that the small teams that have done these last two points to something really exciting: path tracing offers the hope of vastly simplifying the rendering engine.
Because everything is rendered with one core algorithm—path tracing—the engine is simpler, and thus, smaller teams can deliver visually compelling content.
That's an exciting future, as far as the variety of what we may see come out of it.



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

So we’ve seen lots of amazing stuff with ray tracing already.

Are we done?

[Beautiful War for the Planet of the Apes Image redacted]

We are not.

The offline rendering folks made this transition ~15 years ago and had to figure out a lot of stuff to make it work.

There's a lot to learn from them; some of it directly applicable, some of it not.

They have different constraints: long render times, future frames are known, artist in the loop.

They also have different goals: render until quality, not render until times. Average frame time constraint vs maximum frame time constraint.

As another aspirational example, here's a clip from a short film, Zero Day, by an artist who goes by "beeple".
It's path traced, off line.
Dynamic light sources, dynamic geometry, lots of glossy stuff—it's not easy to render.
It'd be amazing to get to the point of doing this sort of thing in real-time.

[Spolier-alert: in the following talk, Chris Wyman shows just that.]

As another aspirational example, here's a clip from a short film, Zero Day, by an artist who goes by "beeple".
It's path traced, off line.
Dynamic light sources, dynamic geometry, lots of glossy stuff—it's not easy to render.
It'd be amazing to get to the point of doing this sort of thing in real-time.

[Spolier-alert: in the following talk, Chris Wyman shows just that.]

Definitions (For Today)

- Ray tracing
  - Geometric algorithm to compute ray-based visibility
- Path tracing
  - Unified light transport algorithm based on random sampling (Monte Carlo integration)

There's been lots of ray tracing in games already.
Take existing raster engine, apply ray tracing for some visual effects.
It's super exciting stuff.
There are still a lot of problems to solve here—e.g. Natasha's GDC talk on ray tracing in the Unity engine, things like shadow map lookups for intersections outside of viewing frustum—we didn't have to worry about it before.
But that's not the focus of this talk.

Something similar happened in film rendering 15 years ago: at first incremental adoption of ray tracing, one effect at a time.
But then, fairly quickly, they made the jump and discarded raster entirely—all ray tracing.
Single light transport algorithm: hair and fur and volumetrics and surfaces and different types of lights, etc.
It turned out to be way simpler
And artists were more productive
And the visuals were better.
In "Open problems", we try to really push the boundaries, so I'm going to talk about whether games might make that leap, too, and what that might take.
Now, a lot of this applies to application of ray tracing to raster, too, so if you're not sold on games also making that transition, that's fine
…but there are some interesting data points…

Not Today

- Ray budgeting
  - Maintaining a fixed ray budget per frame
  - Scaling image quality with more / fewer / no rays
- Geometry
  - LOD for ray tracing
  - Particle systems
  - Hair and fur
  - Foliage and alpha-tested geometry
- Global illumination
  - Caustics
  - Artistic control of light transport
- Engine implications
  - Frustum culling, RIP
  - Data-driven programming: meet your scene graph
  - Streaming geometry / textures on demand
- Volumetric effects
  - Smoke
  - Emissive volumes
- Role of deep learning
  - What else beyond denoising?
- Outlier rejection
- Many others…

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

If I was repi, I'd have polled twitter and gotten an even wider range of stuff to call out of scope.

But there's enough to require a small font

Let's not worry about all of what's there.

Overview

- Path tracing primer:
  - Monte Carlo basics and direct illumination
  - Path tracing and indirect illumination
  - Sampling patterns
- Open problems:
  - Ray (in)coherence
  - Denoising
  - Data structures for visibility

Here's what the rest of the talk will be about.

First, some foundations, just to have a shared language and to all be on the same page.

For real-time, the key constraint is a limited number of rays per frame—you've got to make the most of them.

So after the foundations, we'll go into open problems, all subject to that constraint.

Monte Carlo and Diffuse Direct Lighting

Monte Carlo:
Integration via Random Sampling

$$\int_{[0,1)^n} f(x)\,\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

Here's the key idea behind Monte Carlo integration.

You can integrate more or less anything based on only point-wise evaluation of the function—that's it.

Any number of dimensions, n—no problem.

Here it's posed in terms of integrating over the unit n-dimensional hypercube, but you can map all of the rendering integrals we care about to that.

\int_{[0,1)^n} f(x) \, \mathrm{d}x \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)
x_i = (\xi_1, \xi_2, \ldots, \xi_n)
\xi_i \in [0,1)

Monte Carlo:
Integration via Random Sampling

$$\int_{[0,1)^n} f(x)\,\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

$$x_i = (\xi_1, \xi_2, \ldots, \xi_n)$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Here's the key idea behind Monte Carlo integration.

You can integrate more or less anything based on only point-wise evaluation of the function—that's it.

Any number of dimensions, n—no problem.

Here it's posed in terms of integrating over the unit n-dimensional hypercube, but you can map all of the rendering integrals we care about to that.

\int_{[0,1)^n} f(x) \, \mathrm{d}x \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)
x_i = (\xi_1, \xi_2, \ldots, \xi_n)
\xi_i \in [0,1)

Monte Carlo:
Integration via Random Sampling

$$\int_{[0,1)^n} f(x)\,\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

$$x_i = (\xi_1, \xi_2, \ldots, \xi_n)$$

$$\xi_i \in [0,1)$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Here's the key idea behind Monte Carlo integration.

You can integrate more or less anything based on only point-wise evaluation of the function—that's it.

Any number of dimensions, n—no problem.

Here it's posed in terms of integrating over the unit n-dimensional hypercube, but you can map all of the rendering integrals we care about to that.

```
\int_{[0,1)^n} f(x) \, \mathrm{d}x \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)
x_i = (\xi_1, \xi_2, \ldots, \xi_n)
\xi_i \in [0,1)
```

Monte Carlo Integration

$$\int f(x)\,\mathrm{d}x = E\left[\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right]$$

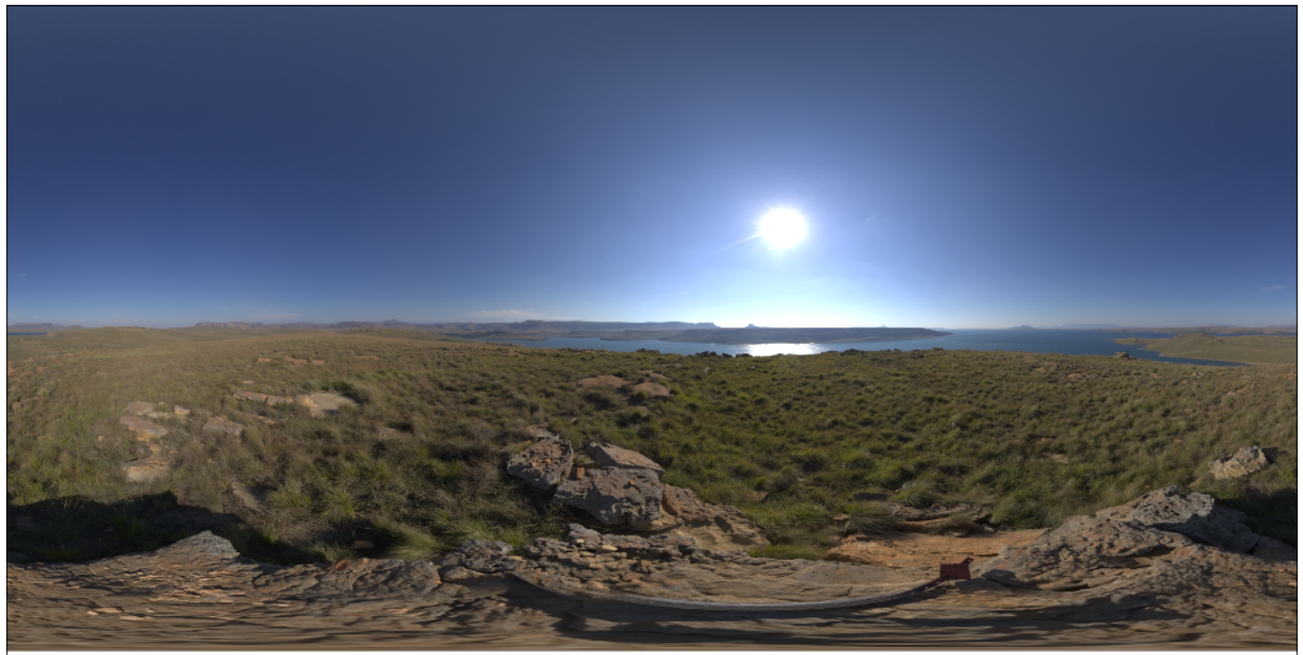
E **: expectation**

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And then the key property is that the expected value of this sum of random samples of the function f—-the integrand—is equal to the integral you care about.
On average, it's perfect.

\int f(x) \, \mathrm{d}x = E \left[\sum_{i=1}^{N} f(x_i) \right]



HDRI Haven: Qwantani

But there's "correct" and there's "efficient".

It may take a long time to get to the correct result.

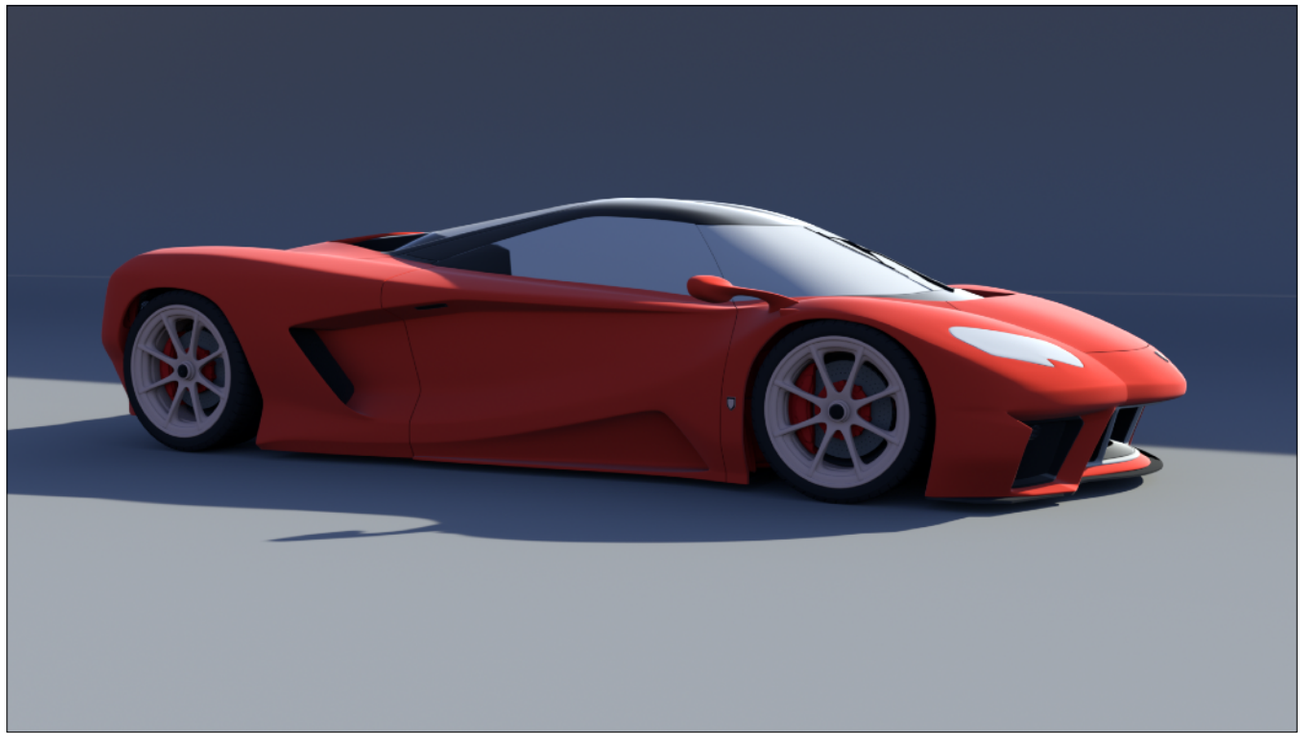
But how you sample the function—where you evaluate it—can make a big difference.

So let's dig into that.

Here's a nifty HDR environment map from HDRI Haven—outdoors, clear day, and a super bright sun.

The pixels with the sun are thousands of times brighter than the rest of them.

Pro-tip: the sun is bright.



And here's a converged image of a model illuminated by that environment map.

For now, all of the surfaces are diffuse.

Look at that remarkably sharp shadow—that's all from the sun being so much brighter than the rest of the environment.

HDR Environment Lighting

$$L(\mathrm{p}) = \int_\Omega L_\mathrm{i}(\omega)\, V(\mathrm{p}, \omega)\, K_\mathrm{d} \cos\theta \, \mathrm{d}\omega$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And here's the integral that describes the radiance at each point in that scene.

Because everything is diffuse, the BRDF is a constant.

So we're mostly just integrating the incident radiance, L_i times a binary visibility term.

L(\mathrm{p}) = \int_\Omega L_\mathrm{i}(\omega) \, V(\mathrm{p}, \omega) \, K_\mathrm{d} \, \cos \theta \, \mathrm{d}\omega

HDR Environment Lighting

Reflected light

$$L(\mathrm{p}) = \int_{\Omega} L_{\mathrm{i}}(\omega)\, V(\mathrm{p}, \omega)\, K_{\mathrm{d}} \cos\theta \, \mathrm{d}\omega$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And here's the integral that describes the radiance at each point in that scene.

Because everything is diffuse, the BRDF is a constant.

So we're mostly just integrating the incident radiance, L_i times a binary visibility term.

L(\mathrm{p}) = \int_\Omega L_\mathrm{i}(\omega) \, V(\mathrm{p}, \omega) \, K_\mathrm{d} \, \cos \theta \, \mathrm{d}\omega

HDR Environment Lighting

Reflected light

$$L(\mathrm{p}) = \int_\Omega L_\mathrm{i}(\omega)\, V(\mathrm{p}, \omega)\, K_\mathrm{d} \cos\theta \, \mathrm{d}\omega$$

Unoccluded light from environment

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And here's the integral that describes the radiance at each point in that scene.

Because everything is diffuse, the BRDF is a constant.

So we're mostly just integrating the incident radiance, L_i times a binary visibility term.

L(\mathrm{p}) = \int_\Omega L_\mathrm{i}(\omega) \, V(\mathrm{p}, \omega) \, K_\mathrm{d} \, \cos \theta \, \mathrm{d}\omega

HDR Environment Lighting

Reflected light

Visibility (ray traced)

$$L(\mathrm{p}) = \int_{\Omega} L_{\mathrm{i}}(\omega)\, V(\mathrm{p}, \omega)\, K_{\mathrm{d}} \cos\theta \, \mathrm{d}\omega$$

Unoccluded light from environment

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And here's the integral that describes the radiance at each point in that scene.

Because everything is diffuse, the BRDF is a constant.

So we're mostly just integrating the incident radiance, L_i times a binary visibility term.

L(\mathrm{p}) = \int_\Omega L_\mathrm{i}(\omega) \, V(\mathrm{p}, \omega) \, K_\mathrm{d} \, \cos \theta \, \mathrm{d}\omega

HDR Environment Lighting

Reflected light

Visibility (ray traced)

$$L(\mathrm{p}) = \int_\Omega L_\mathrm{i}(\omega)\, V(\mathrm{p}, \omega)\, K_\mathrm{d} \cos\theta \, \mathrm{d}\omega$$

Unoccluded light from environment

Diffuse reflection

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And here's the integral that describes the radiance at each point in that scene.

Because everything is diffuse, the BRDF is a constant.

So we're mostly just integrating the incident radiance, L_i times a binary visibility term.

L(\mathrm{p}) = \int_\Omega L_\mathrm{i}(\omega) \, V(\mathrm{p}, \omega) \, K_\mathrm{d} \, \cos \theta \, \mathrm{d}\omega

Environment Lighting Estimator

$$L(\mathrm{p}) = E\left[2\pi \frac{1}{N} \sum_{i=1}^{N} L_{\mathrm{i}}(\omega_i)\, V(\mathrm{p}, \omega_i)\, K_{\mathrm{d}}\, \cos\theta_i\right]$$

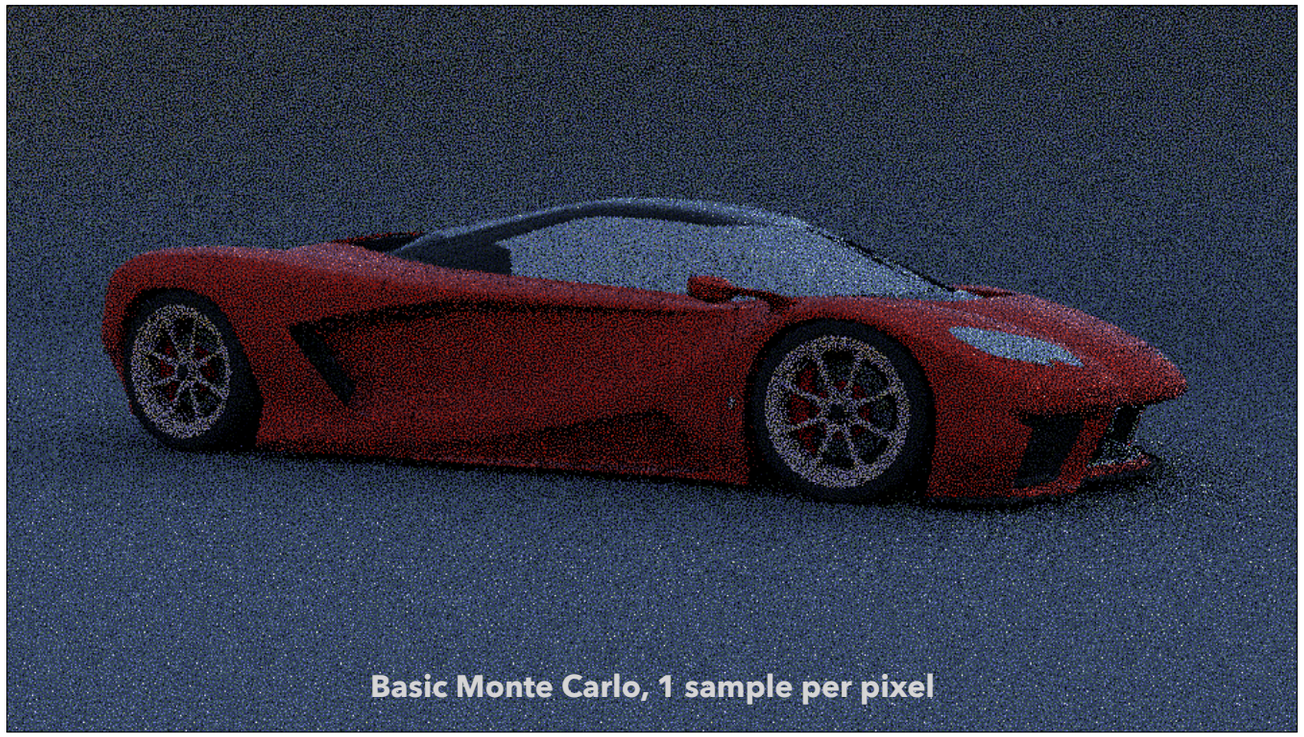
N **random directions** ω_i

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And here's the corresponding Monte Carlo estimator, just from turning the crank and applying the equation from a few slides back.
So in this case, it's just saying, choose N random directions w_i with equal probability and sum up the incident light times visibility times diffuse reflectance times cosine.
That 2pi factor shows up because we're integrating over the hemisphere, which has area 2pi.

L(\mathrm{p}) = E \left[2 \pi \frac{1}{N} \, \sum_{i=1}^N L_\mathrm{i}(\omega_i) \, V(\mathrm{p}, \omega_i) \, K_\mathrm{d} \, \cos \theta_i \right]

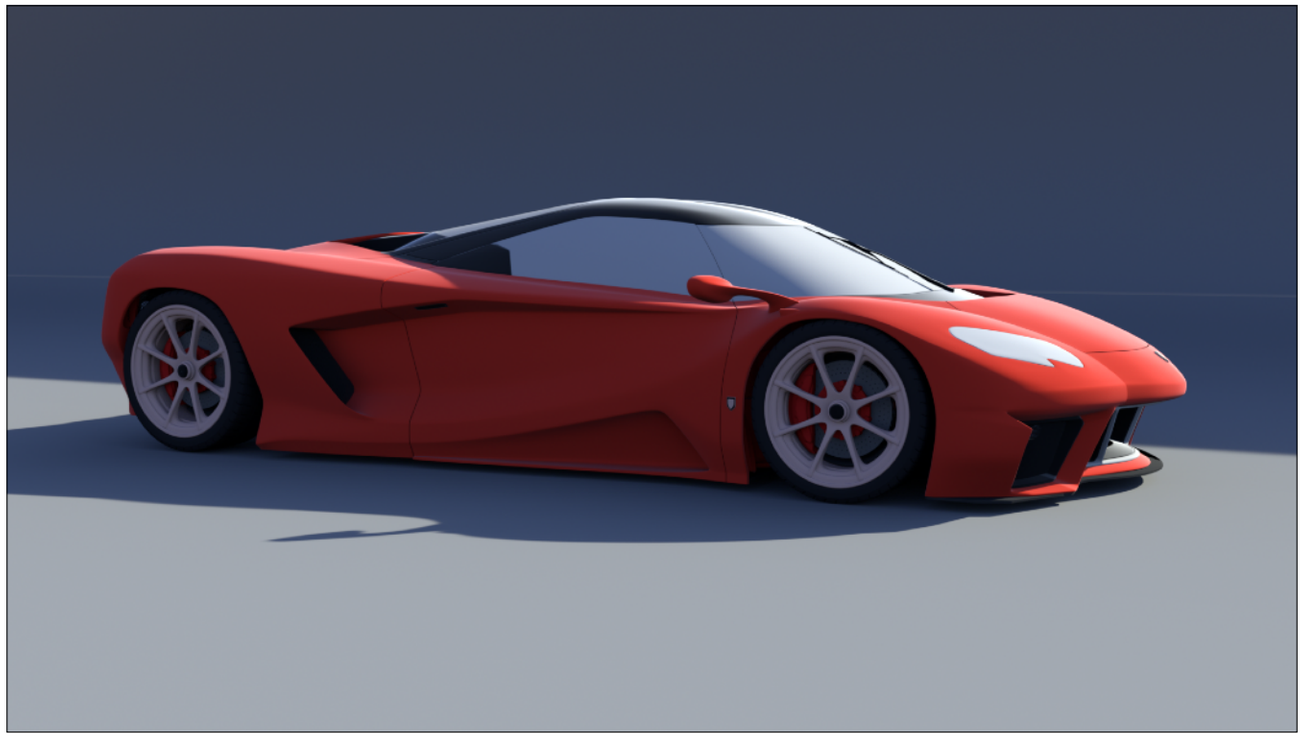


Ok, so if you do that, here's what you get.

This is one sample per pixel, using that Monte Carlo estimator.

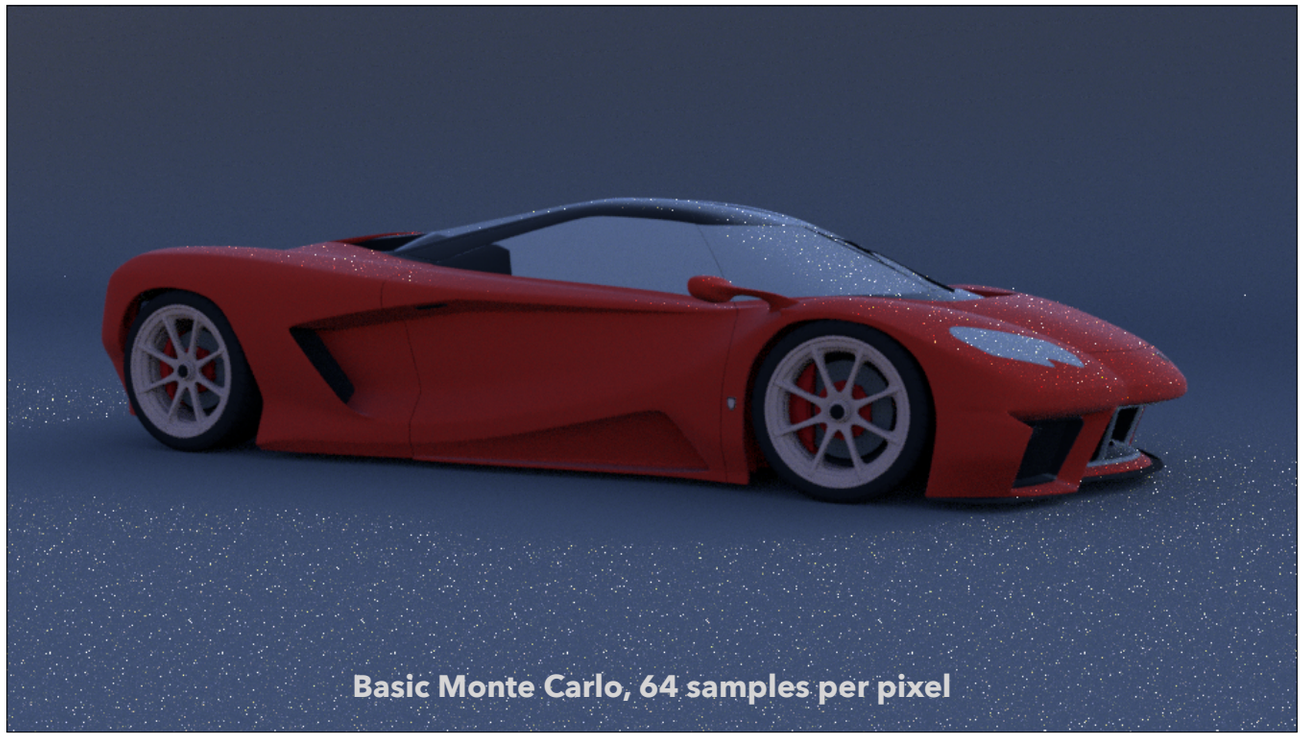
Um… What happened?

We basically completely lost the shadow from the sun.



Here’s the reference image again.

It’s “different”.



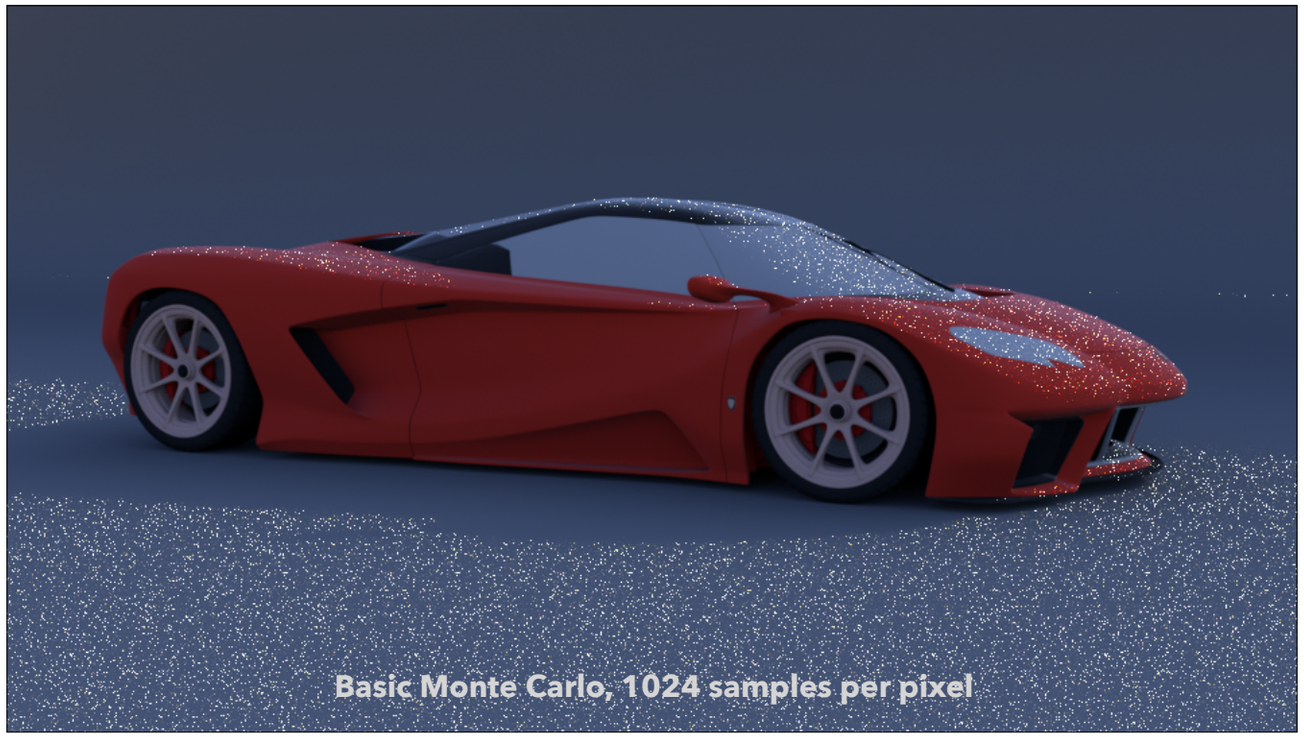
Basic Monte Carlo, 64 samples per pixel

So one sample (ray) per pixel isn't very many.

Let's take more—improve the accuracy.

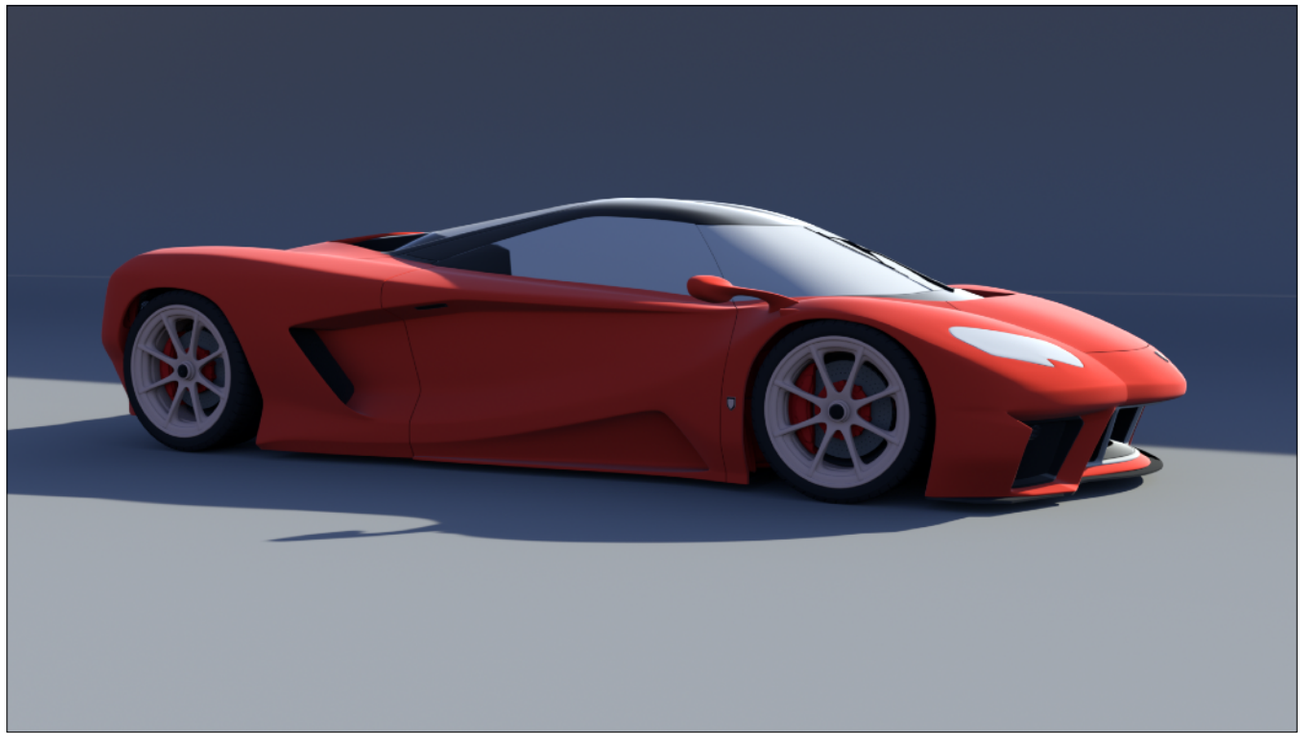
Here's what we get with 64 samples per pixel.

Not really all that much better…



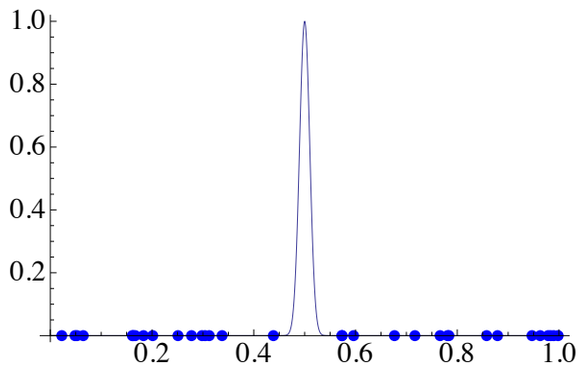
Here’s 1024 samples per pixel.

It’s way way too much for real-time and it still looks terrible.



We’re still a ways off..

Problem?



$$f(x) = \mathrm{e}^{-5000(x-\frac{1}{2})^2}$$

Estimate: 6.1418×10^{-12}

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Here's what's going on, illustrated with an example in 1D.

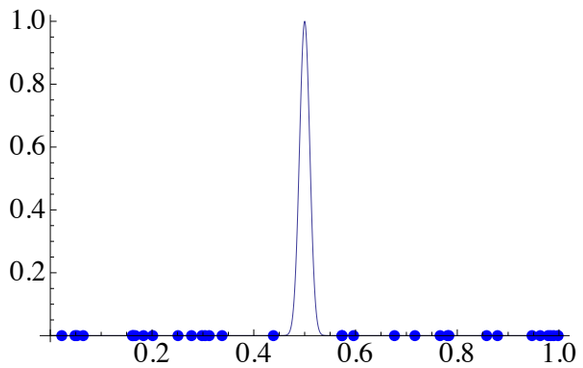
Say you have a really spiky function, like the one here—it's zero almost everywhere, but then spikes up in a small region of the domain.

If you sample uniformly—here between 0 and 1—then you may miss the spike entirely.

And if all of your samples are where the function is relatively small, then you have no hope of estimating its integral.

You need samples in that spike to be able to figure out what's going on with it—that's where the mass of the integral is.

Problem?



$$f(x) = \mathrm{e}^{-5000(x-\frac{1}{2})^2}$$

Estimate: 6.1418×10^{-12}

$$\int f(x)\mathrm{d}x = 0.025067\ldots$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Here's what's going on, illustrated with an example in 1D.

Say you have a really spiky function, like the one here—it's zero almost everywhere, but then spikes up in a small region of the domain.

If you sample uniformly—here between 0 and 1—then you may miss the spike entirely.

And if all of your samples are where the function is relatively small, then you have no hope of estimating its integral.

You need samples in that spike to be able to figure out what's going on with it—that's where the mass of the integral is.

Importance Sampling Estimator

$$\int f(x)\,\mathrm{d}x = E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}\right] \qquad x_i \sim p(x)$$

Use samples from a *non-uniform* distribution $p(x)$

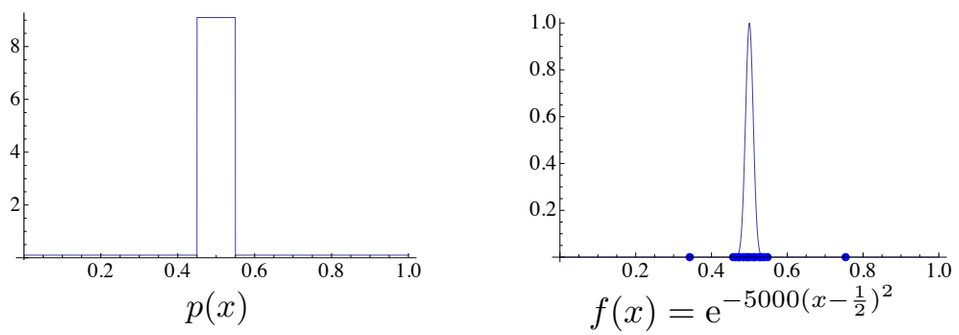
Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Another classic idea from Monte Carlo is importance sampling.
The idea is to distribute your samples non-uniformly—here, according to some probability density function p.
You try to place more samples where the function is "interesting" (aka has a large magnitude)
And the importance sampling estimator lets you do that.
We've now got a 1/p factor that balances that out.
Effectively, we're placing samples non-uniformly, and in regions where we're sampling more (p is high), the weight of those samples (1/p) is low.
We still compute an estimate of the integral in the end.
Now, you might think that doesn't make much difference—sure we sample more in some areas, but those samples count less.
Is that buying us anything?

\int f(x) \, \mathrm{d}x = E \left[\frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \right]

x_i \sim p(x)

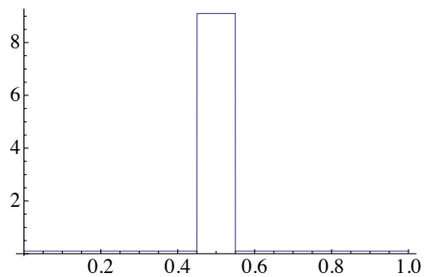


Let’s look at an example with our spiky function.

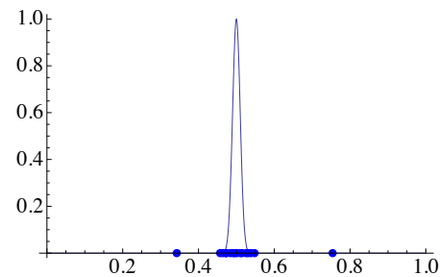
We’ll distribute samples according to the probability density on the left—more samples in the middle, fewer elsewhere.

And on the right is an example set of 16 samples drawn according to p.

We’ve got lots of them in the spiky part of the function.



$p(x)$



$f(x) = \mathrm{e}^{-5000(x-\frac{1}{2})^2}$

Estimates (N=16)

0.02502	**0.02612**	**0.02678**	**0.02678**
0.02490	**0.02534**	**0.02592**	**0.02443**

$$\int f(x)\mathrm{d}x = 0.025067\ldots$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And now suddenly, we can do a really good job of estimating the integral.

Here are a few independent examples of taking 16 samples according to p and using the importance sampling estimator.

We’re doing much better, coming much closer to the actual value of the integral.

Importance Sampling Diffuse Direct Lighting

$$L(\mathrm{p}) = E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{L_{\mathrm{i}}(\omega_i)\,V(\mathrm{p},\omega_i)\,K_{\mathrm{d}}\,\cos\theta_i}{p(\omega_i)}\right]$$

$$\downarrow$$

$$p(\omega) \propto L_i(\omega)$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

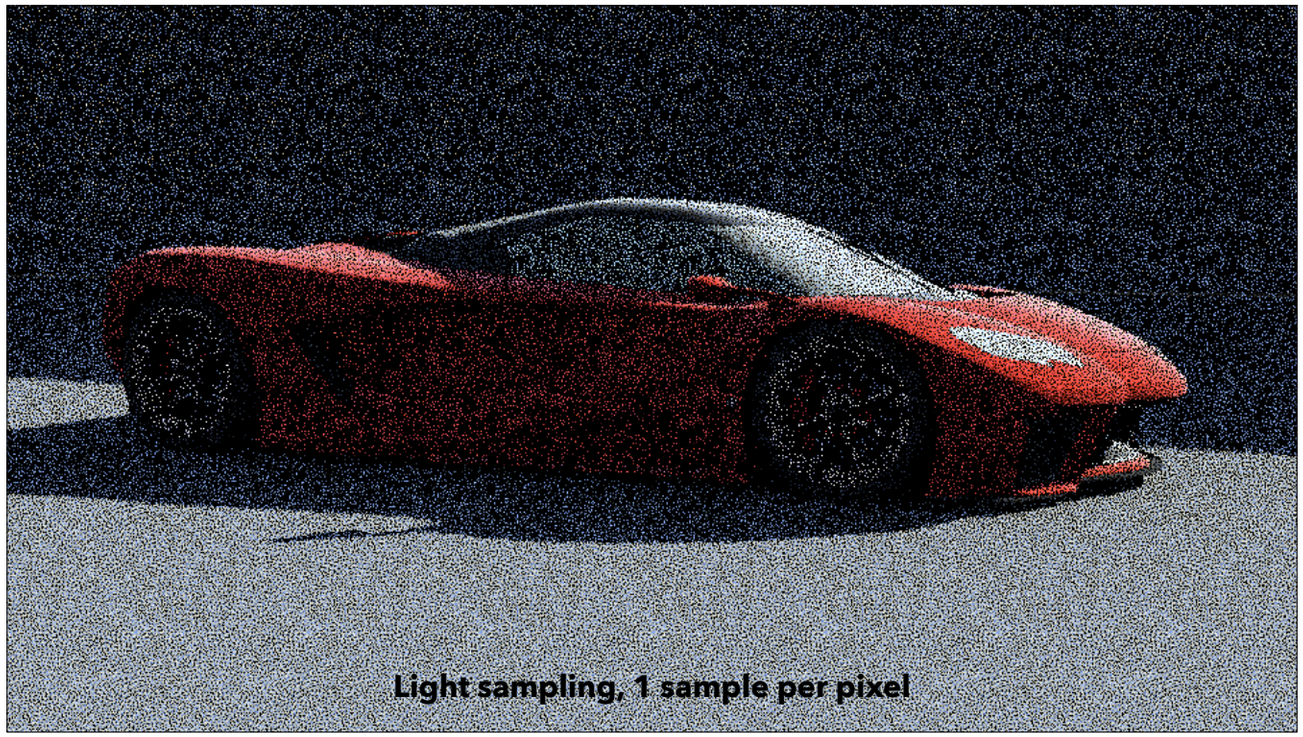
Matt Pharr

Now, let's go back to the happy place of rendering.
The HDR environment map defines the incident radiance function L_i, and it's just a table of values.
There are a few algorithms out there that make it possible to sample directions proportionally to the incident radiance function.
Let's try choosing samples according to the environment map—choose directions with probability proportional to the unshadowed radiance along them.

L(\mathrm{p}) = E \left[\frac{1}{N} \, \sum_{i=1}^N \frac{L_\mathrm{i}(\omega_i) \, V(\mathrm{p}, \omega_i) \, K_\mathrm{d} \, \cos \theta_i}{p(\omega_i)} \right]

p(\omega) \propto L_i(\omega) \, \longrightarrow \, \frac{L_i(\omega)}{p(\omega)} = c

L(\mathrm{p}) = c \, K_\mathrm{d} \, E \left[\frac{1}{N} \, \sum_{i=1}^N V(\mathrm{p}, \omega_i) \cos \theta_i \right]



Here’s what happens, with one ray per pixel.

It isn’t perfect, but hey, look—there’s that sharp shadow from the sun.

We’re picking that up, since the sun direction is sampled frequently.

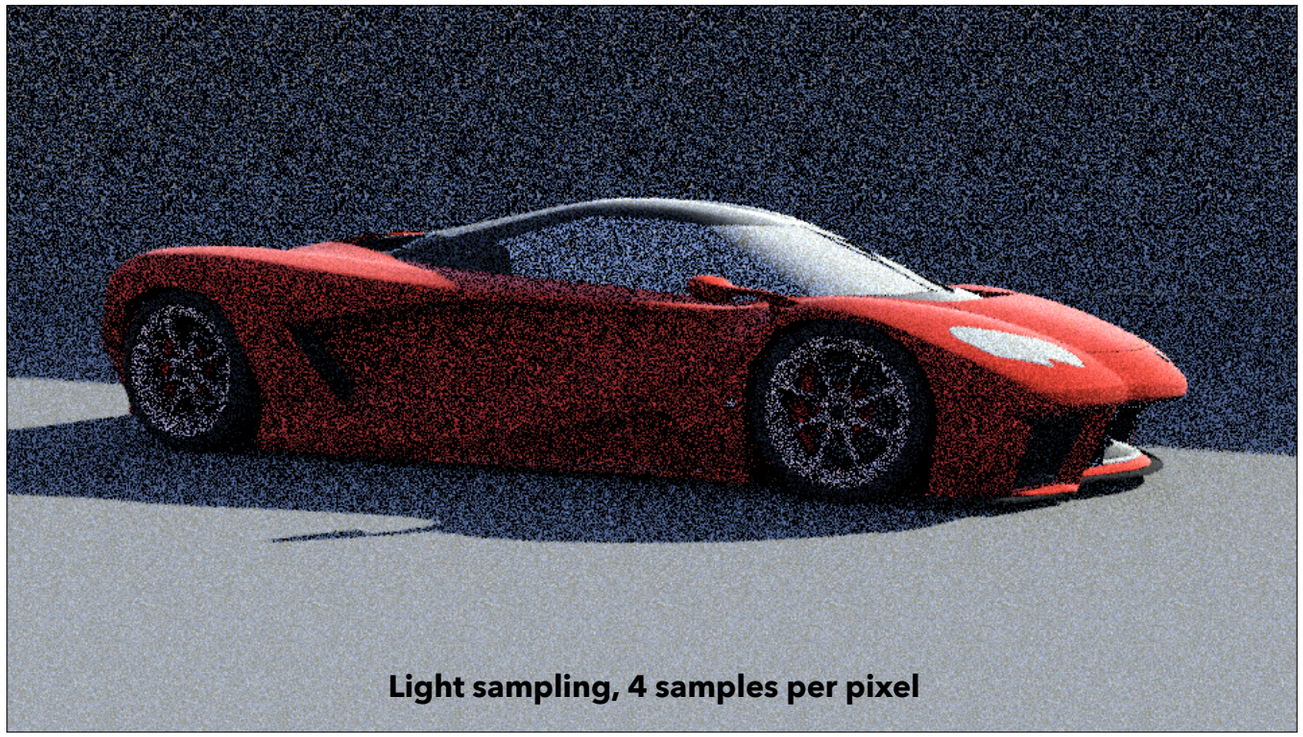
The key thing is, that ray tracing makes this possible.

Each pixel can choose the best direction for a ray to trace, independent of all of the others.

You can’t do that with rasterization.

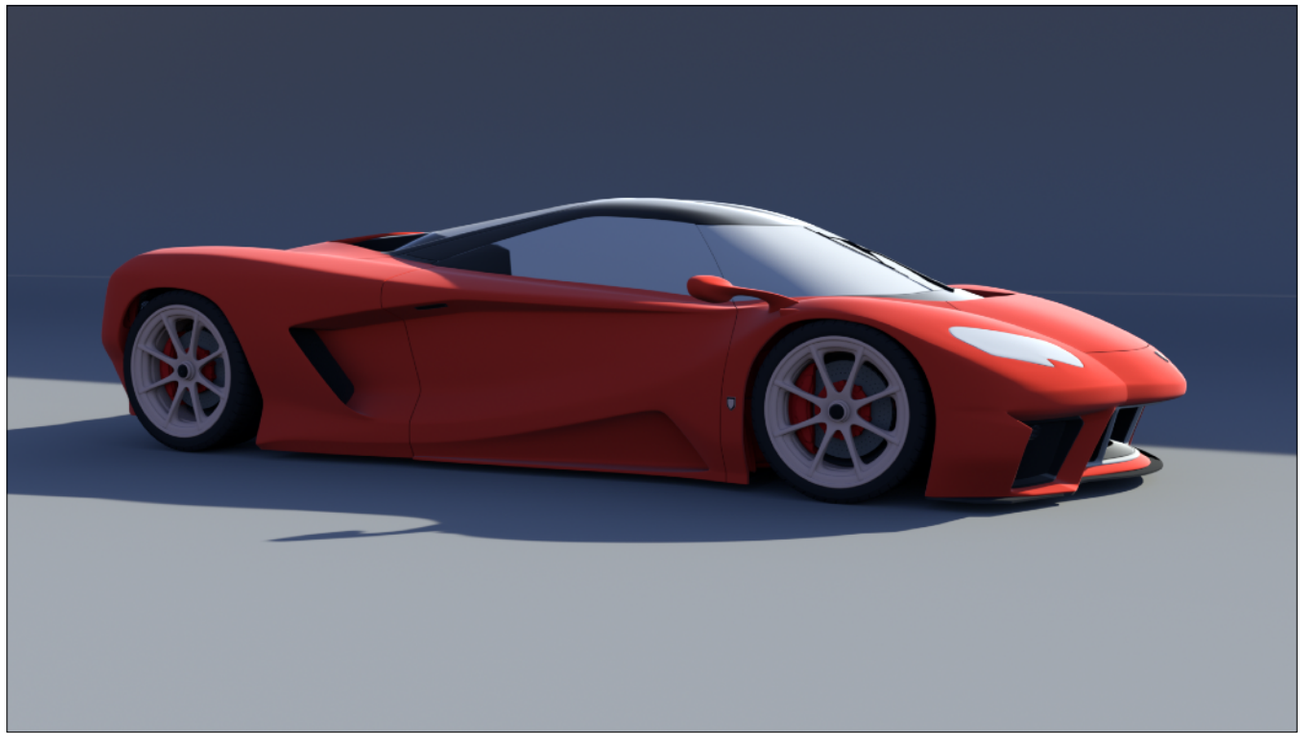
And therein is why ray tracing has the potential to be more efficient, overall than rasterization.

You can’t trace as many rays as you can rasterize fragments, but, you can go pretty far with the right ray versus a single direction imposed by rasterization.



Light sampling, 4 samples per pixel

With 4 rays per pixel, things are looking pretty good..



Not too far off from the reference image.

What Just Happened?

$$L(\mathrm{p}) = E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{L_{\mathrm{i}}(\omega_i)\,V(\mathrm{p},\omega_i)\,K_{\mathrm{d}}\,\cos\theta_i}{p(\omega_i)}\right]$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Let’s go back to the math to understand what happened.

We’re drawing samples w_i with probability proportional to the unshadowed radiance from the environment map.

In turn, when we consider the importance sampling estimator here, we have a ratio of L_i / p, which is a constant, which we’ll denote by c.

L(\mathrm{p}) = E \left[\frac{1}{N} \, \sum_{i=1}^N \frac{L_\mathrm{i}(\omega_i) \, V(\mathrm{p}, \omega_i) \, K_\mathrm{d} \, \cos \theta_i}{p(\omega_i)} \right]

p(\omega_i) \propto L_i(\omega_i) \longrightarrow \frac{L_i(\omega_i)}{p(\omega_i)}=c

L(\mathrm{p}) = c \, K_\mathrm{d} \, E \left[\frac{1}{N} \, \sum_{i=1}^N V(\mathrm{p}, \omega_i) \cos \theta_i \right]

What Just Happened?

$$L(\mathrm{p}) = E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{L_{\mathrm{i}}(\omega_i)\,V(\mathrm{p},\omega_i)\,K_{\mathrm{d}}\,\cos\theta_i}{p(\omega_i)}\right]$$

$$p(\omega_i) \propto L_i(\omega_i) \longrightarrow \frac{L_i(\omega_i)}{p(\omega_i)} = c$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Let’s go back to the math to understand what happened.

We’re drawing samples w_i with probability proportional to the unshadowed radiance from the environment map.

In turn, when we consider the importance sampling estimator here, we have a ratio of L_i / p, which is a constant, which we’ll denote by c.

L(\mathrm{p}) = E \left[\frac{1}{N} \, \sum_{i=1}^N \frac{L_\mathrm{i}(\omega_i) \, V(\mathrm{p}, \omega_i) \, K_\mathrm{d} \, \cos \theta_i}{p(\omega_i)} \right]

p(\omega_i) \propto L_i(\omega_i) \longrightarrow \frac{L_i(\omega_i)}{p(\omega_i)}=c

L(\mathrm{p}) = c \, K_\mathrm{d} \, E \left[\frac{1}{N} \, \sum_{i=1}^N V(\mathrm{p}, \omega_i) \cos \theta_i \right]

What Just Happened

$$L(\mathrm{p}) = c\,K_{\mathrm{d}}\,E\left[\frac{1}{N}\sum_{i=1}^{N} V(\mathrm{p},\omega_i)\cos\theta_i\right]$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

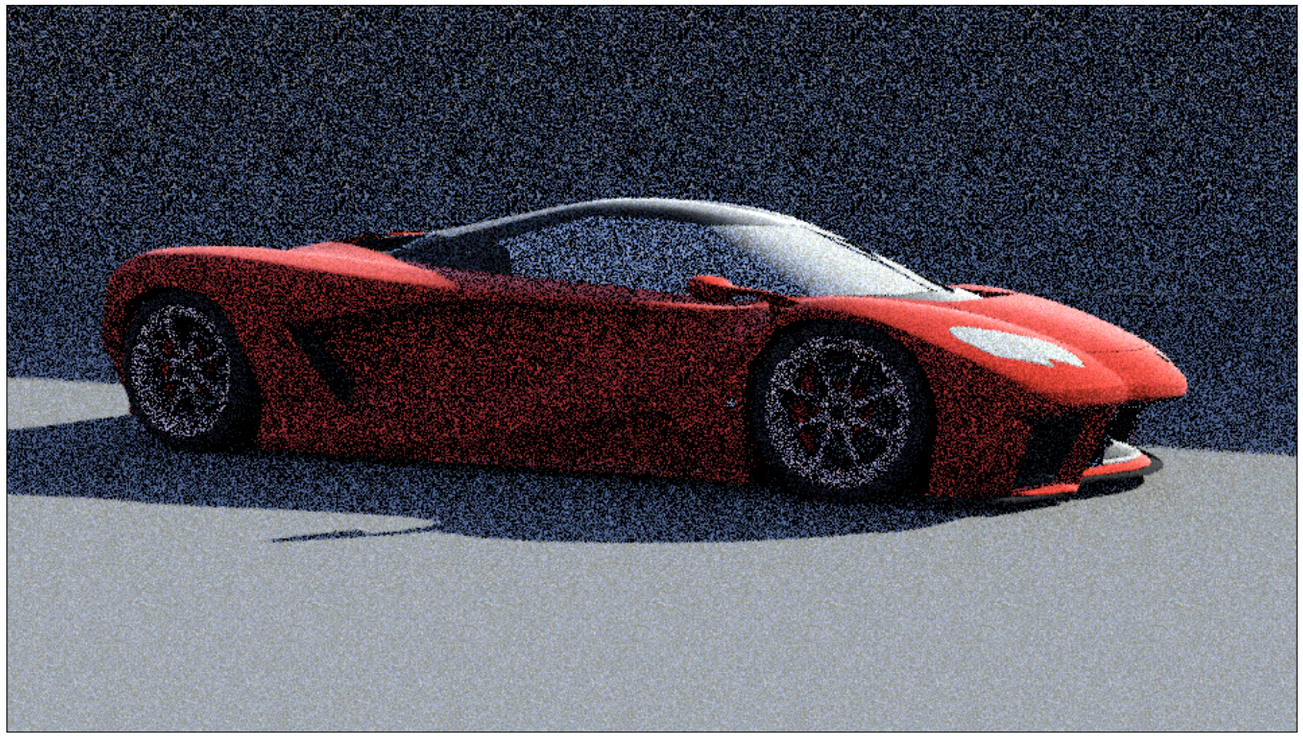
Matt Pharr

Cancel a few things out and pull constants out of the sum, and we have this, which is interesting.
There is a constant factor outside the sum, and then inside it, it's just the binary 0 or 1 visibility function and a little cosine.
So what this means is that there's limited variation in each term of the sum—they're all between 0 and 1.
And in turn, it gives a more accurate estimate.

L(\mathrm{p}) = E \left[\frac{1}{N} \, \sum_{i=1}^N \frac{L_\mathrm{i}(\omega_i) \, V(\mathrm{p}, \omega_i) \, K_\mathrm{d} \, \cos \theta_i}{p(\omega_i)} \right]

p(\omega_i) \propto L_i(\omega_i) \longrightarrow \frac{L_i(\omega_i)}{p(\omega_i)}=c

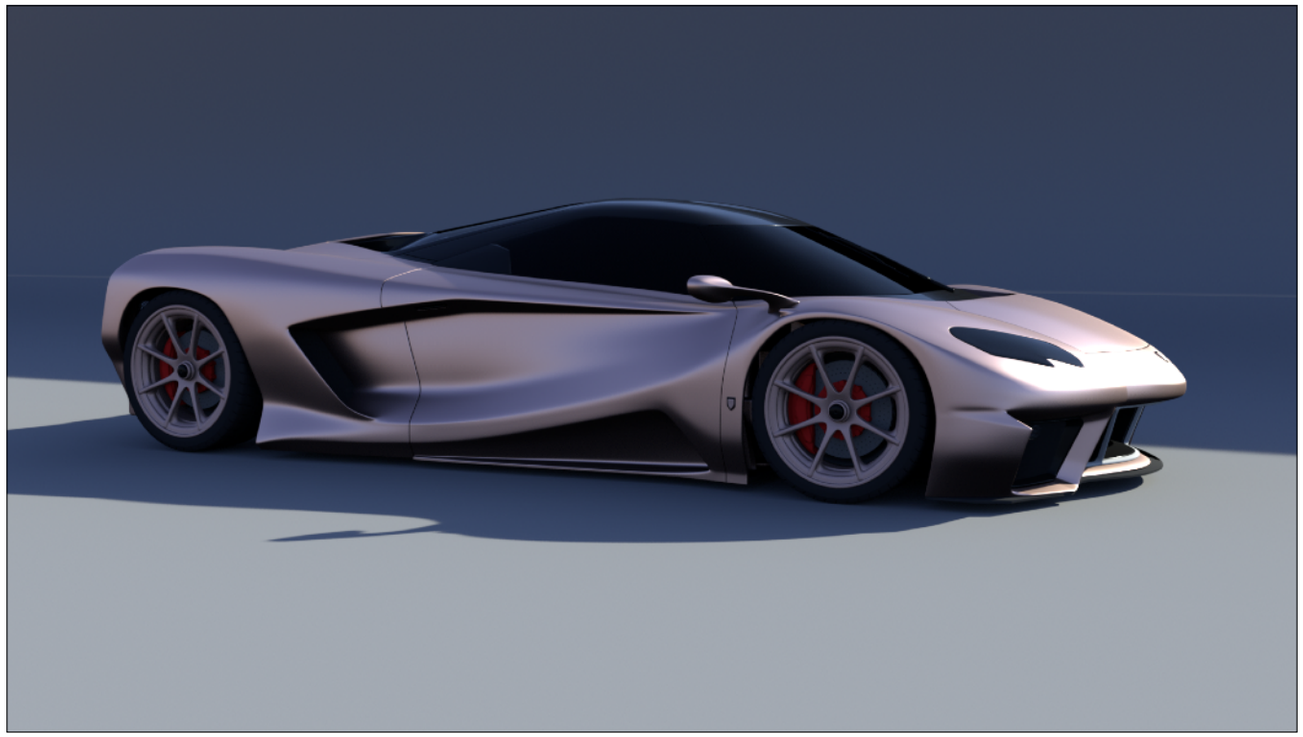
L(\mathrm{p}) = c \, K_\mathrm{d} \, E \left[\frac{1}{N} \, \sum_{i=1}^N V(\mathrm{p}, \omega_i) \cos \theta_i \right]



Here’s 4 samples per pixel again, where you can see that limited variation in the form of less noise.

Direct Lighting (General)

Diffuse surfaces are boring, so let’s bring in the BRDFs.



Here’s the model, now with a glossy BRDF for the car paint.

Here’s a reference rendering.

Direct Lighting Estimator

$$L(\mathrm{p}, \omega_o) = \int_\Omega L_i(\omega)\, V(\mathrm{p}, \omega)\, f_r(\omega \to \omega_o)\, \cos\theta \, \mathrm{d}\omega$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

On the top we have the direct lighting integral, which should be familiar.
And on the bottom is the importance sampled Monte Carlo estimator of it.
This is just applying the definition of it.
So let's sample according to the light source, like we did before—that worked well.

L(\mathrm{p}, \omega_o) = \int_\Omega L_i(\omega) \, V(\mathrm{p}, \omega) \, f_r(\omega \rightarrow \omega_o) \, \cos \theta \, \mathrm{d}\omega

L(\mathrm{p}, \omega_o) = E \left[\frac{1}{N} \sum_{i=1}^N \frac{L_i(\omega_i) \, V(\mathrm{p}, \omega_i) \, f_r(\omega_i \rightarrow \omega_o) \, \cos \theta_i}{p(\omega_i)} \right]

Direct Lighting Estimator

$$L(\mathrm{p}, \omega_o) = \int_\Omega L_i(\omega)\, V(\mathrm{p}, \omega)\, f_r(\omega \rightarrow \omega_o)\, \cos\theta \, \mathrm{d}\omega$$

$$L(\mathrm{p}, \omega_o) = E\left[\frac{1}{N}\sum_{i=1}^{N} \frac{L_i(\omega_i)\, V(\mathrm{p}, \omega_i)\, f_r(\omega_i \rightarrow \omega_o)\, \cos\theta_i}{p(\omega_i)}\right]$$

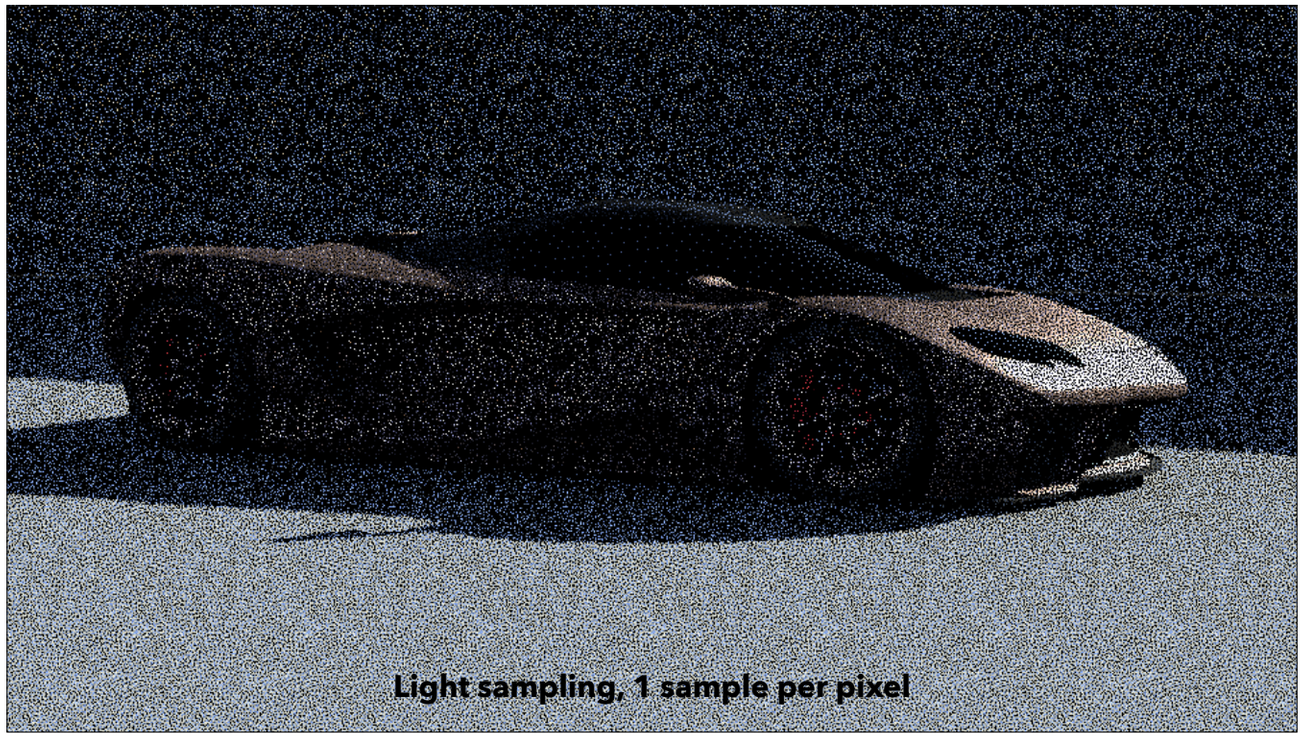
Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

On the top we have the direct lighting integral, which should be familiar.
And on the bottom is the importance sampled Monte Carlo estimator of it.
This is just applying the definition of it.
So let's sample according to the light source, like we did before—that worked well.

L(\mathrm{p}, \omega_o) = \int_\Omega L_i(\omega) \, V(\mathrm{p}, \omega) \, f_r(\omega \rightarrow \omega_o) \, \cos \theta \, \mathrm{d}\omega

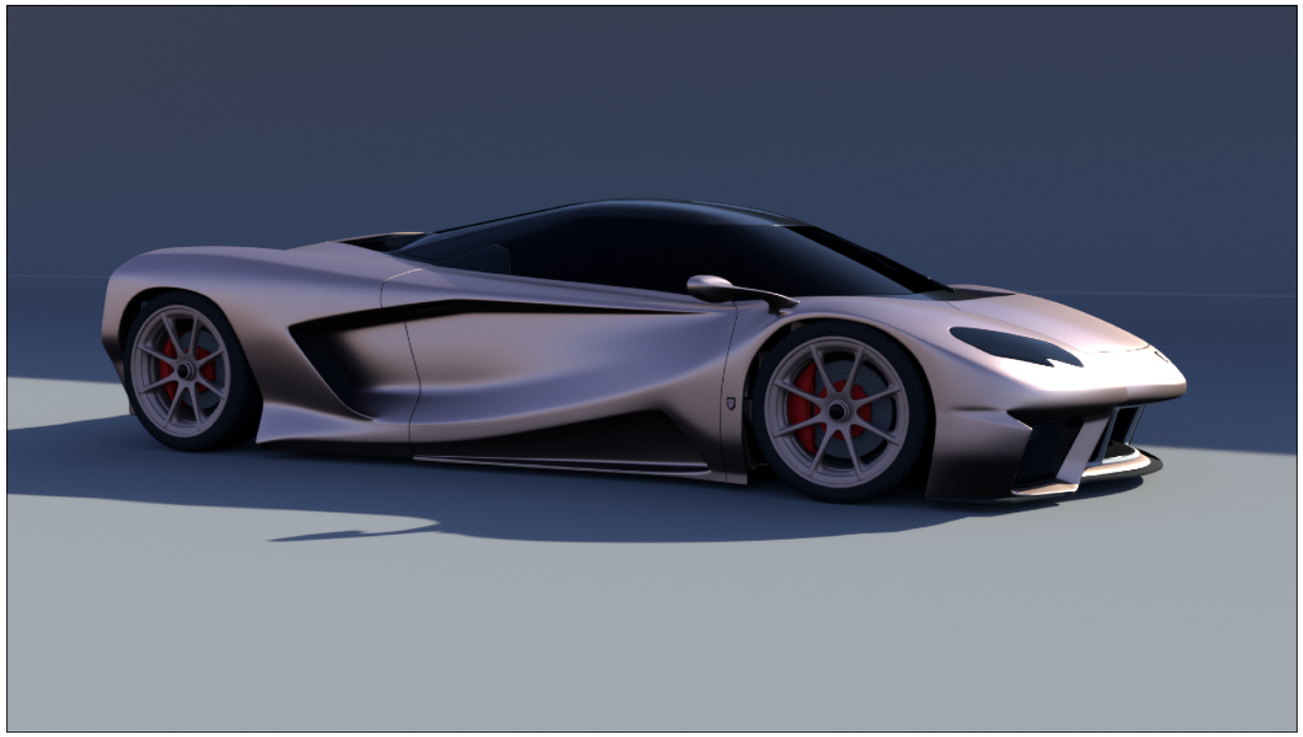
L(\mathrm{p}, \omega_o) = E \left[\frac{1}{N} \sum_{i=1}^N \frac{L_i(\omega_i) \, V(\mathrm{p}, \omega_i) \, f_r(\omega_i \rightarrow \omega_o) \, \cos \theta_i}{p(\omega_i)} \right]



…and it’s not pretty.

We got that sharp shadow from the sun on the ground.

But things are pretty bad on the car body, especially on the side that’s facing the viewer.



And here’s the reference image, for a painful reminder of how far off we are.

What Just Happened?

$$L(\mathrm{p}, \omega_o) = E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{L_i(\omega_i)\,V(\mathrm{p}, \omega_i)\,f_r(\omega_i \rightarrow \omega_o)\,\cos\theta_i}{p(\omega_i)}\right]$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

The math helps us understand what happened.

We sampled directions according to L_i, so again we have that constant factor c given by L_i / p,

L(\mathrm{p}, \omega_o) = c \, E \left[\frac{1}{N} \sum_{i=1}^N V(\mathrm{p}, \omega_i) \, f_r(\omega_i \rightarrow \omega_o) \, \cos \theta_i \right]

What Just Happened?

$$L(\mathrm{p}, \omega_o) = E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{L_i(\omega_i)\,V(\mathrm{p}, \omega_i)\,f_r(\omega_i \to \omega_o)\,\cos\theta_i}{p(\omega_i)}\right]$$

$$p(\omega_i) \propto L_i(\omega_i) \longrightarrow \frac{L_i(\omega_i)}{p(\omega_i)} = c$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

The math helps us understand what happened.

We sampled directions according to L_i, so again we have that constant factor c given by L_i / p,

L(\mathrm{p}, \omega_o) = c \, E \left[\frac{1}{N} \sum_{i=1}^N V(\mathrm{p}, \omega_i) \, f_r(\omega_i \rightarrow \omega_o) \, \cos \theta_i \right]

What Just Happened

$$L(\mathrm{p}, \omega_o) = c\, E\left[\frac{1}{N}\sum_{i=1}^{N} V(\mathrm{p}, \omega_i)\, f_r(\omega_i \rightarrow \omega_o)\, \cos\theta_i\right]$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Cancel things out and pull the constants out of the sum, and we've got this.

Like before, the visibility term and the cosine, but now there's the BRDF as well.

And what if that's spiky, like a highly glossy BRDF—like the car paint?

Well, the sampled directions w_i don't account for the BRDF as well, so we have that same problem we had in that original 1D example.

A small set of directions really matter, but we're not doing a good job of finding them.

L(\mathrm{p}, \omega_o) = c \, E \left[\frac{1}{N} \sum_{i=1}^N V(\mathrm{p}, \omega_i) \, f_r(\omega_i \rightarrow \omega_o) \, \cos \theta_i \right]

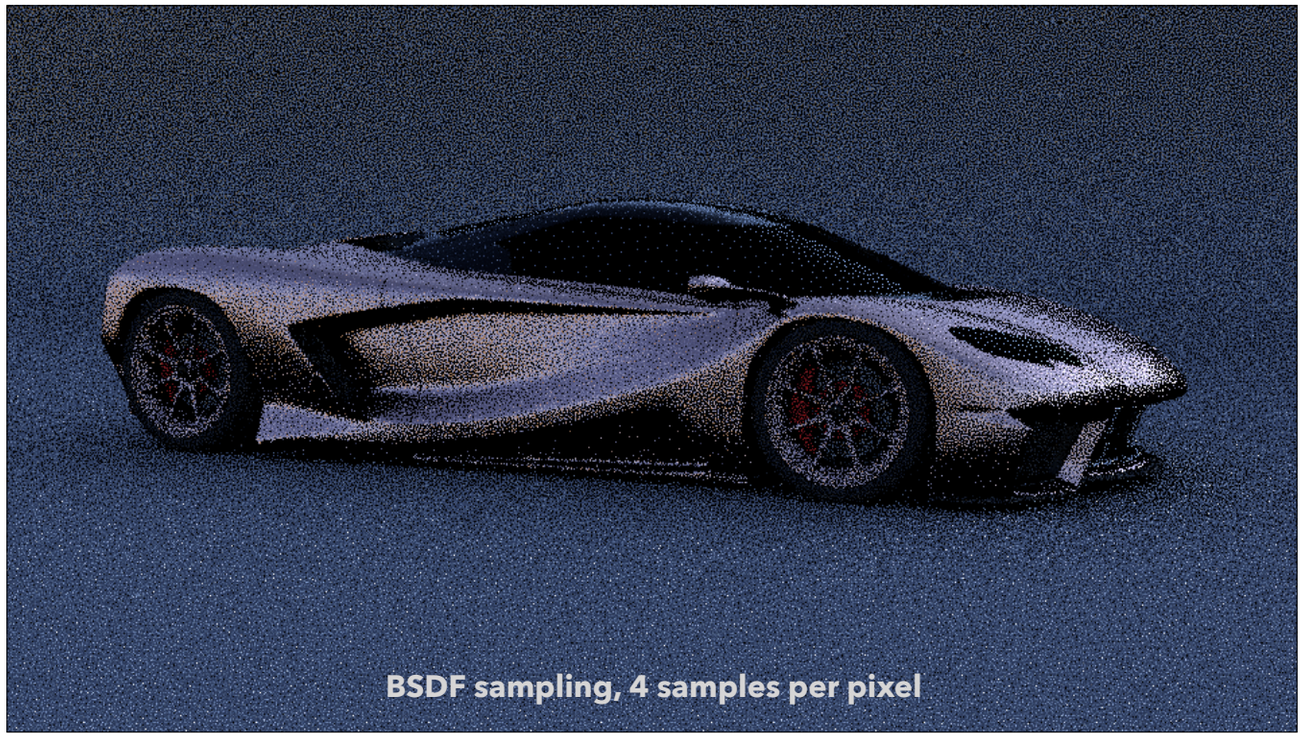
$$p(\omega) \propto f_r(\omega \rightarrow \omega_o)\,\textbf{?}$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Ok, fine. If the BRDF is so important, let's draw samples according to it.
There are lots of recipes for doing that, for all sorts of common BRDFs.

p(\omega) \propto f_r(\omega \rightarrow \omega_o)

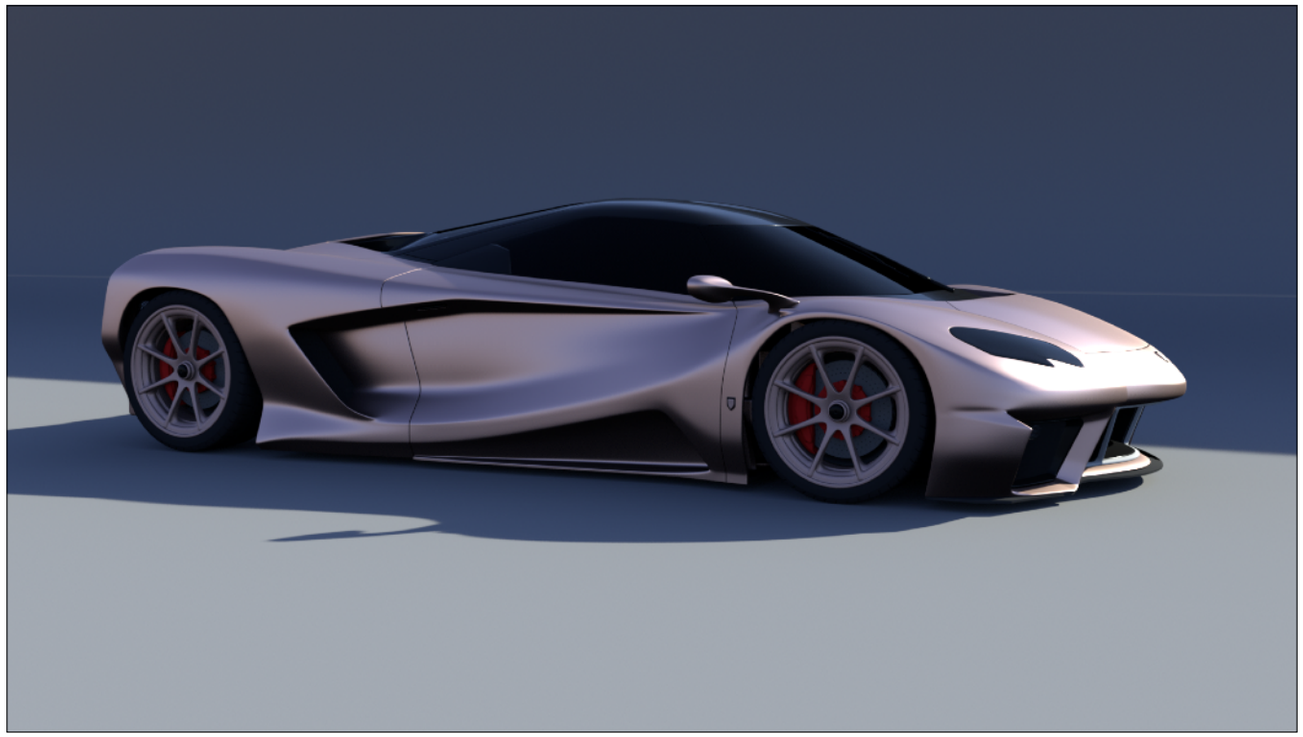


…and we get this, with 4 samples per pixel.

The car body is looking much better.

But we lost that sharp shadow from the sun.

Overall the ground is much worse.



And again the reference image, to mock us.

We'll switch back and forth between this and the BSDF sampling method image…

Take note of where BSDF sampling matches the reference closely—e.g. the inner part of the door.

In other parts, sometimes it's noisy, and sometimes it's completely bad, like the ground.

What Just Happened

$$L(\mathrm{p}, \omega_o) = c\, E \left[\frac{1}{N} \sum_{i=1}^{N} L_i(\omega_i)\, V(\mathrm{p}, \omega_i)\ \cos\theta_i \right]$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

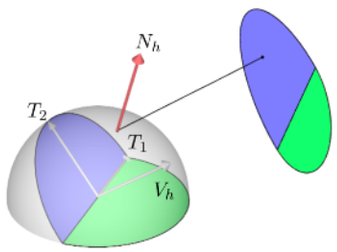
Matt Pharr

If we do the same trick of expressing the ratio of the sampled factor—the BRDF—and the PDF as a constant, c, we're left with this as the estimator.
As before, the part that's stochastically evaluated is in the square brackets.
So now we have the same problem as before: if L varies a lot, as it does here, and the sampling method isn't accounting for it, we have a spiky function left to integrate and get into the same old kinds of trouble.

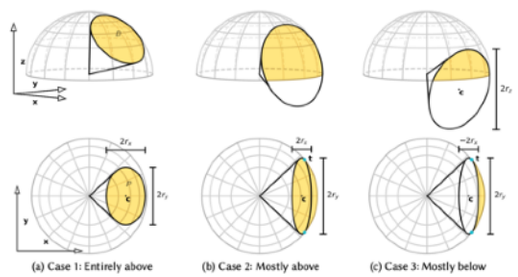
p(\omega) \propto f_r(\omega \rightarrow \omega_o) \, \longrightarrow \, \frac{f_r(\omega \rightarrow \omega_o)}{p(\omega)} = c

L(\mathrm{p}, \omega_o) = c \, E \left[\frac{1}{N} \sum_{i=1}^N L_i(\omega_i) \, V(\mathrm{p}, \omega_i) \, \cos \theta_i \right]

(Still) Open Problem:
BSDF and Light Sampling



***Sampling the GGX Distribution of Visible Normals*, Heitz 2018**

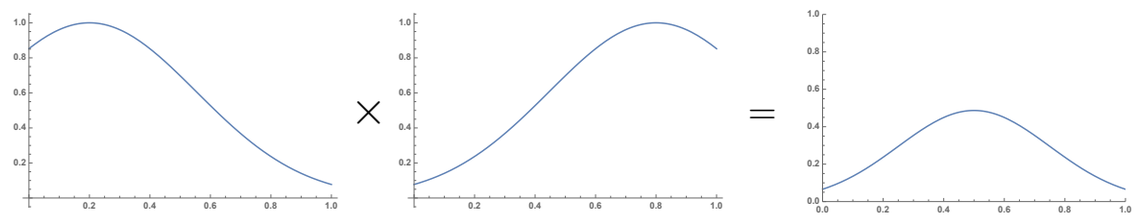


***Sampling Projected Spherical Caps in Real Time*, Peters and Dachsbacher 2019**

That brings us to our first open problem: sampling the product of BRDFs and lighting.

There are all sorts of algorithms out there to sample each factor individually—two cool recent ones are shown here—but it'd be even better to sample the product of lighting and BRDF together.

Product Sampling



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Here's another 1D illustration where we're looking at the product of two functions, shown here.

The product, on the right, looks nothing like either one: the peak is around 0.5.

If we sample one or both of the functions independently, we'll mostly be sampling around 0.2 or 0.8, respectively—we won't be concentrating samples around 0.5.

And that in turn, leads to Monte Carlo error..

Open Problem:
Light & BSDF Product Sampling

Want: $p(\omega) \propto f_r(\omega \to \omega_o)\, L_i(\omega) \longrightarrow \dfrac{f_r(\omega \to \omega_o)\, L_i(\omega)}{p(\omega)} = c$

So: $L(\mathrm{p}, \omega_o) = c\, E\left[\dfrac{1}{N}\sum_{i=1}^{N} V(\mathrm{p}, \omega_i)\, \cos\theta_i\right]$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

If we could sample the product, as shown here, then the estimator would just have that visibility variation.
That's the same nice case as we had with direct lighting with diffuse BRDFs.
There's been a lot of really nice research in this area, some of which is listed here.
But none of it is, IMHO, practical for real-time.
There are a lot of great ideas, but these methods also impose limitations like requiring tabularization of the BRDF, or have fairly significant computational overhead to prepare to generate samples.

```
p(\omega) \propto f_r(\omega \rightarrow \omega_o) \, L_i(\omega) \longrightarrow \,  \frac{f_r(\omega \rightarrow \omega_o) \, L_i(\omega)}{p(\omega)} = c

L(\mathrm{p}, \omega_o) = c \, E \left[ \frac{1}{N} \sum_{i=1}^N V(\mathrm{p}, \omega_i) \, \cos \theta_i \right]
```

Open Problem:
Light & BSDF Product Sampling

Want: $p(\omega) \propto f_r(\omega \to \omega_o)\, L_i(\omega) \longrightarrow \frac{f_r(\omega \to \omega_o)\, L_i(\omega)}{p(\omega)} = c$

So: $L(\mathrm{p}, \omega_o) = c\, E\left[\frac{1}{N} \sum_{i=1}^{N} V(\mathrm{p}, \omega_i)\, \cos\theta_i\right]$

***Wavelet importance sampling: efficiently evaluating products of complex functions*, Clarberg et al. 2005**
***Two stage importance sampling for direct lighting*, Cline et al. 2006**
***Efficient product sampling using hierarchical thresholding*, Rousselle et al. 2008**
***Practical product importance sampling for direct illumination*, Clarberg and Akenine-Möller 2008**
***Fast Product Importance Sampling of Environment Maps*, Conty Estevez and Lecocq 2018**

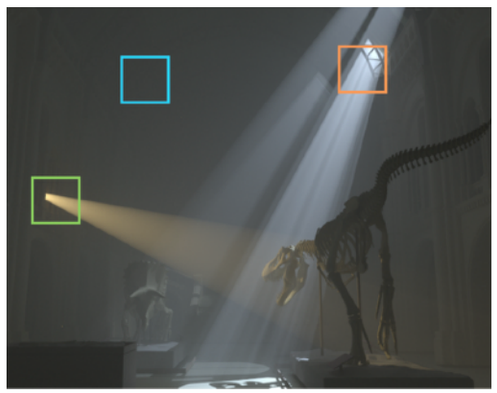
If we could sample the product, as shown here, then the estimator would just have that visibility variation.
That's the same nice case as we had with direct lighting with diffuse BRDFs.
There's been a lot of really nice research in this area, some of which is listed here.
But none of it is, IMHO, practical for real-time.
There are a lot of great ideas, but these methods also impose limitations like requiring tabularization of the BRDF, or have fairly significant computational overhead to prepare to generate samples.

p(\omega) \propto f_r(\omega \rightarrow \omega_o) \, L_i(\omega) \longrightarrow \, \frac{f_r(\omega \rightarrow \omega_o) \, L_i(\omega)}{p(\omega)} = c

L(\mathrm{p}, \omega_o) = c \, E \left[\frac{1}{N} \sum_{i=1}^N V(\mathrm{p}, \omega_i) \, \cos \theta_i \right]

Product Sampling: von-Mises Fisher Mixtures(?)

- Spherical distribution
- Directly supports convolution, product, and sampling
- Has been applied to product sampling for volumetrics



Volume Path Guiding Based on Zero-Variance Random Walk Theory
Herholz et al. 2019

So how might product sampling come to fruition for real-time?

There's a neat paper by Herholz et al that just came out that uses von-Mises Fisher distributions for product sampling for volumetrics.

They're a distribution that supports all of the sorts of things that are important for product sampling and evaluation.

They could potentially be applied to product sampling at surfaces, but the tricky part is fitting arbitrary BRDF lobes to this representation—that's not easy or efficient…

Product Sampling:
Revisit BSDF Models to Simplify?

- What *is* a BSDF?
  - Mostly: functional approximation to a statistical process
- Idea: base BSDF models on functions that provide capabilities that enable product sampling (cf. von-Mies Fisher)

Credit: Jasper Bekkers

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

So there's this impedance mismatch between the sorts of functions that BRDFs are represented by and the sorts of functions that are amenable to product sampling.
In one of those nice SIGGRAPH moments, I was having dinner with Jasper Bekkers last night and, naturally, we were talking about BRDF product sampling.
He had a really clever idea, which he was happy to let me include here.
Here it is: at the end of the day, the BRDF models we currently use are generally based on a set of reasonable but arbitrary assumptions.
We say, "let's assume the surface has a Gaussian distribution of microfacet normals, and let's assume that the microfacets are perfect mirrors, and let's ignore inter-reflection" (for example). And then that leads you to the Cook-Torrance BRDF with, say, the Beckmann NDF.
In practice, if those assumptions lead to a model that matches real-world materials well, then we use that model for rendering.
So why not try to find a set of parameterized functions that are plausible BRDFs, describe real-world materials well, and are set up to be amenable to product sampling with lights?
That'd be amazing if someone was able to figure it out…

(Bad) Alternative: Sum of Two Estimators

$$p_f(\omega) \propto f_r(\omega \rightarrow \omega_o) \qquad p_l(\omega) \propto L_i(\omega)$$

Lacking the ability to sample the product, you might think the best best thing is to sample both the BRDF and the light independently and then average the two estimators.
Best of both worlds, right?

p_f(\omega) \propto f_r(\omega \rightarrow \omega_o)

\omega_f \sim p_f(\omega)

L(\mathrm{p}, \omega_o) = &\frac{1}{2} c_f \, E \left[\frac{1}{N} \sum_{f=1}^N L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos \theta_f \right] + \\
&\frac{1}{2} c_l E \left[\frac{1}{N} \sum_{i=1}^N f_r(\omega_l \rightarrow \omega_o) \, V(\mathrm{p}, \omega_l) \, \cos \theta_l \right]

(Bad) Alternative: Sum of Two Estimators

$$p_f(\omega) \propto f_r(\omega \rightarrow \omega_o) \qquad p_l(\omega) \propto L_i(\omega)$$

$$\omega_f \sim p_f(\omega)$$

Lacking the ability to sample the product, you might think the best best thing is to sample both the BRDF and the light independently and then average the two estimators.
Best of both worlds, right?

p_f(\omega) \propto f_r(\omega \rightarrow \omega_o)

\omega_f \sim p_f(\omega)

L(\mathrm{p}, \omega_o) = &\frac{1}{2} c_f \, E \left[\frac{1}{N} \sum_{f=1}^N L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos \theta_f \right] + \\
&\frac{1}{2} c_l E \left[\frac{1}{N} \sum_{i=1}^N f_r(\omega_l \rightarrow \omega_o) \, V(\mathrm{p}, \omega_l) \, \cos \theta_l \right]

(Bad) Alternative: Sum of Two Estimators

$$p_f(\omega) \propto f_r(\omega \rightarrow \omega_o) \qquad p_l(\omega) \propto L_i(\omega)$$
$$\omega_f \sim p_f(\omega) \qquad \omega_l \sim p_l(\omega)$$

Lacking the ability to sample the product, you might think the best best thing is to sample both the BRDF and the light independently and then average the two estimators.
Best of both worlds, right?

p_f(\omega) \propto f_r(\omega \rightarrow \omega_o)

\omega_f \sim p_f(\omega)

L(\mathrm{p}, \omega_o) = &\frac{1}{2} c_f \, E \left[\frac{1}{N} \sum_{f=1}^N L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos \theta_f \right] + \\
&\frac{1}{2} c_l E \left[\frac{1}{N} \sum_{i=1}^N f_r(\omega_l \rightarrow \omega_o) \, V(\mathrm{p}, \omega_l) \, \cos \theta_l \right]

(Bad) Alternative: Sum of Two Estimators

$$p_f(\omega) \propto f_r(\omega \to \omega_o) \qquad p_l(\omega) \propto L_i(\omega)$$
$$\omega_f \sim p_f(\omega) \qquad \omega_l \sim p_l(\omega)$$

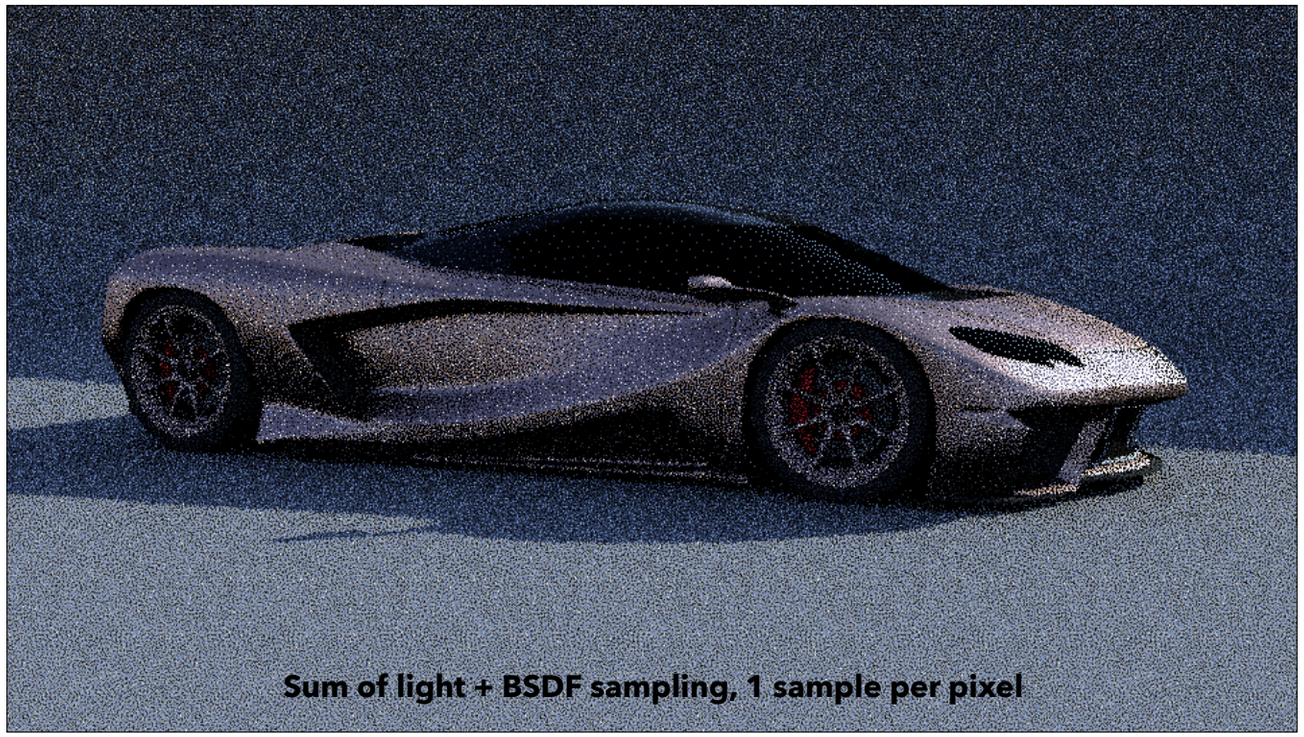
$$L(\mathrm{p}, \omega_o) = \frac{1}{2} c_f \, E \left[\frac{1}{N} \sum_{f=1}^{N} L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos\theta_f \right] + \frac{1}{2} c_l E \left[\frac{1}{N} \sum_{i=1}^{N} f_r(\omega_l \to \omega_o) \, V(\mathrm{p}, \omega_l) \, \cos\theta_l \right]$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Lacking the ability to sample the product, you might think the best best thing is to sample both the BRDF and the light independently and then average the two estimators.
Best of both worlds, right?

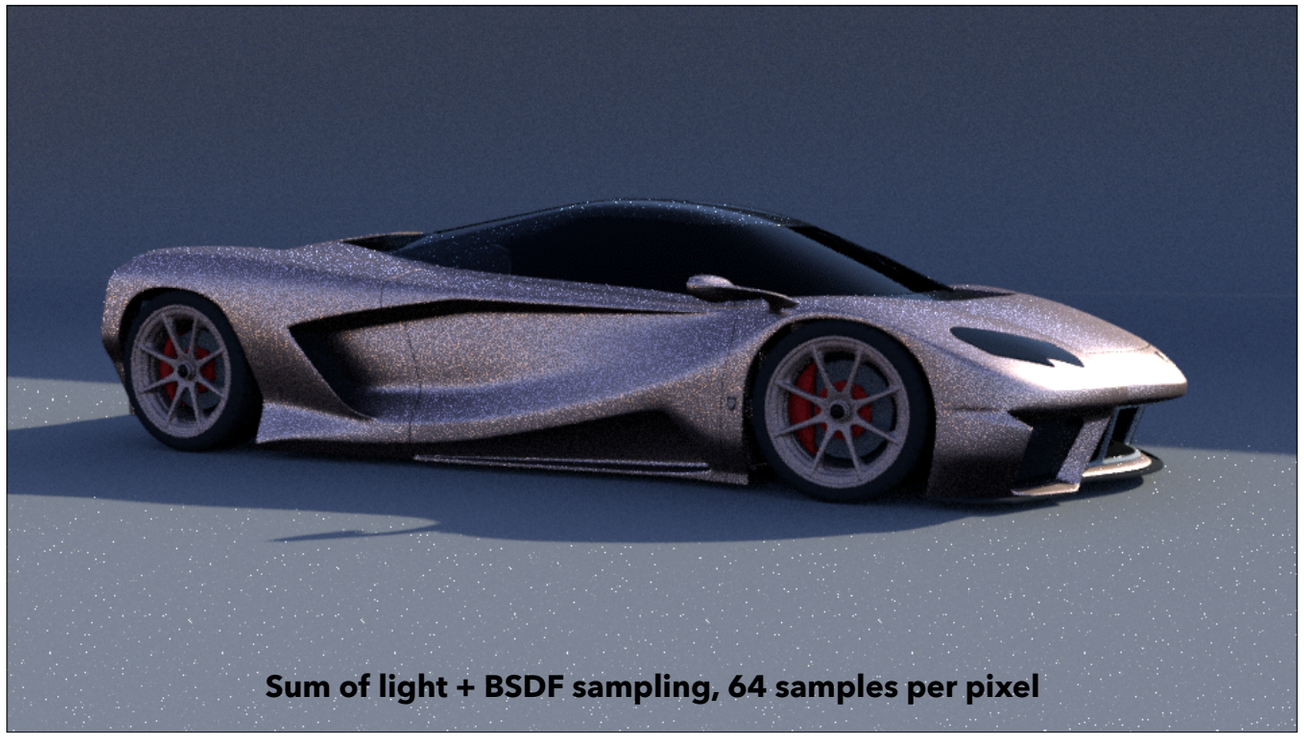
```
p_f(\omega) \propto f_r(\omega \rightarrow \omega_o)

\omega_f \sim p_f(\omega)

L(\mathrm{p}, \omega_o) = &\frac{1}{2} c_f \, E \left[ \frac{1}{N} \sum_{f=1}^N L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos \theta_f \right] + \\
&\frac{1}{2} c_l E \left[ \frac{1}{N} \sum_{i=1}^N f_r(\omega_l \rightarrow \omega_o) \, V(\mathrm{p}, \omega_l) \, \cos \theta_l \right]
```



At one sample per pixel, visually, it seems like we might be on to something.

We've got that sharp shadow from the sun and the car is looking much better than it did when we did light sampling only.

Let's take more samples to try to get the noise down…



And here's 64 samples.

Um…

It's better, but it's sort of worse, too.

We've got a whole bunch of bright "fireflies", and they're not going away.

In fact, it seems like we're getting more as we take more samples.

What's happening here is that when we for example sample the BRDF and choose some direction where the BRDF's value is low and so the probability function is small, now and again, we'll hit the sun.

We have an estimate with a big value in the numerator—the L_i factor is big, and a small factor in the denominator—the probability is small.

So a big thing divided by a small thing gives a really large value.

Good Alternative: Multiple Importance Sampling

$$p_f(\omega) \propto f_r(\omega \rightarrow \omega_o) \qquad p_l(\omega) \propto L_i(\omega)$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

The thing to do about this is to apply multiple importance sampling.
The idea is that you still sample both factors, but then you weight them in a way that accounts for both sampling techniques' probabilities of sampling a direction.
If one chooses a direction with low probability, the other sampling technique has a chance to weigh in and effectively say "that's actually an important direction as far as I'm concerned", and the samples are weighted in a way that eliminates those variance spikes.

L(\mathrm{p}, \omega_o) = & E \left[\frac{1}{N} \sum_{f=1}^N \frac{f_r(\omega_f \rightarrow \omega_o) L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos \theta_f}
{p_f(\omega_f) + p_l(\omega_l)} \right] + \\
& E \left[\frac{1}{N} \sum_{f=1}^N \frac{f_r(\omega_l \rightarrow \omega_o) L_i(\omega_l) \, V(\mathrm{p}, \omega_l) \, \cos \theta_l}{p_f(\omega_f) + p_l(\omega_l)} \right]

Good Alternative: Multiple Importance Sampling

$$p_f(\omega) \propto f_r(\omega \to \omega_o) \qquad p_l(\omega) \propto L_i(\omega)$$

$$\omega_f \sim p_f(\omega)$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

The thing to do about this is to apply multiple importance sampling.
The idea is that you still sample both factors, but then you weight them in a way that accounts for both sampling techniques' probabilities of sampling a direction.
If one chooses a direction with low probability, the other sampling technique has a chance to weigh in and effectively say "that's actually an important direction as far as I'm concerned", and the samples are weighted in a way that eliminates those variance spikes.

L(\mathrm{p}, \omega_o) = & E \left[\frac{1}{N} \sum_{f=1}^N \frac{f_r(\omega_f \rightarrow \omega_o) L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos \theta_f}{p_f(\omega_f) + p_l(\omega_l)} \right] + \\
& E \left[\frac{1}{N} \sum_{f=1}^N \frac{f_r(\omega_l \rightarrow \omega_o) L_i(\omega_l) \, V(\mathrm{p}, \omega_l) \, \cos \theta_l}{p_f(\omega_f) + p_l(\omega_l)} \right]

Good Alternative: Multiple Importance Sampling

$$p_f(\omega) \propto f_r(\omega \rightarrow \omega_o) \qquad p_l(\omega) \propto L_i(\omega)$$
$$\omega_f \sim p_f(\omega) \qquad \omega_l \sim p_l(\omega)$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

The thing to do about this is to apply multiple importance sampling.
The idea is that you still sample both factors, but then you weight them in a way that accounts for both sampling techniques' probabilities of sampling a direction.
If one chooses a direction with low probability, the other sampling technique has a chance to weigh in and effectively say "that's actually an important direction as far as I'm concerned", and the samples are weighted in a way that eliminates those variance spikes.

L(\mathrm{p}, \omega_o) = & E \left[\frac{1}{N} \sum_{f=1}^N \frac{f_r(\omega_f \rightarrow \omega_o) L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos \theta_f}{p_f(\omega_f) + p_l(\omega_l)} \right] + \\
& E \left[\frac{1}{N} \sum_{f=1}^N \frac{f_r(\omega_l \rightarrow \omega_o) L_i(\omega_l) \, V(\mathrm{p}, \omega_l) \, \cos \theta_l}{p_f(\omega_f) + p_l(\omega_l)} \right]

Good Alternative: Multiple Importance Sampling

$$p_f(\omega) \propto f_r(\omega \to \omega_o) \qquad p_l(\omega) \propto L_i(\omega)$$
$$\omega_f \sim p_f(\omega) \qquad \omega_l \sim p_l(\omega)$$

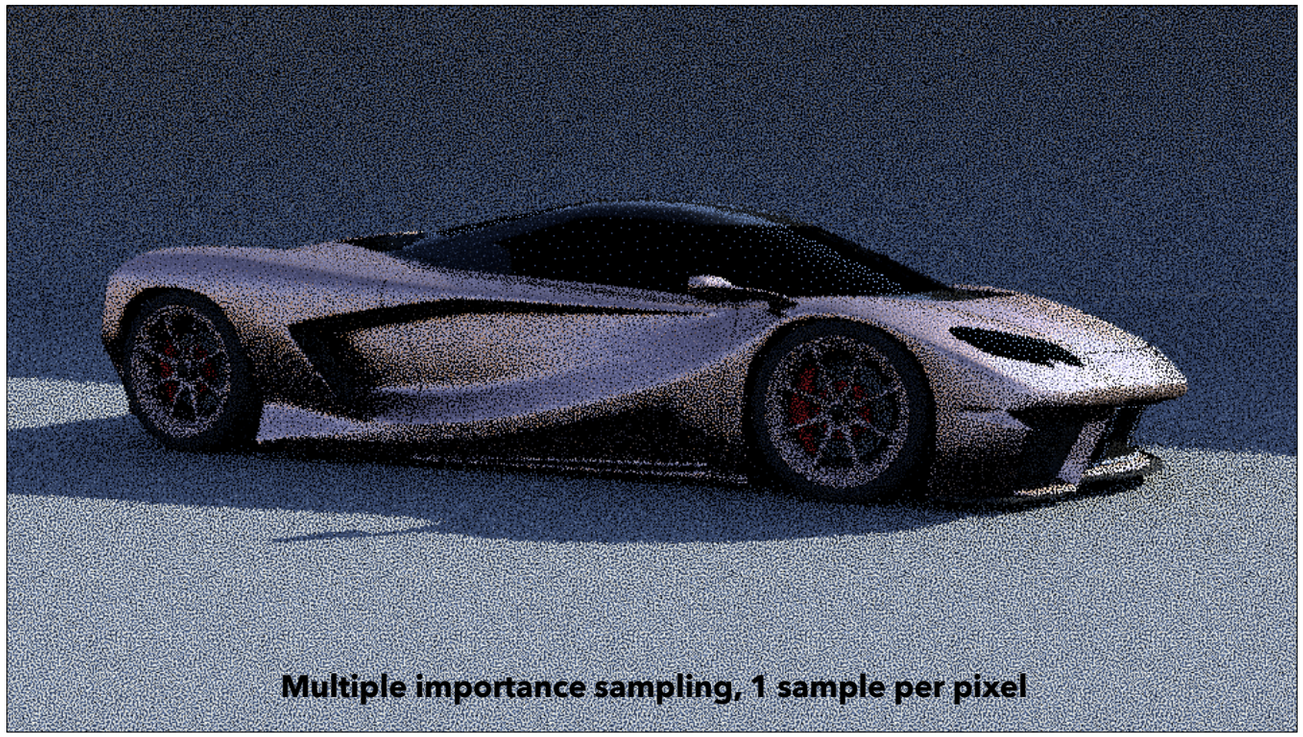
$$L(\mathrm{p}, \omega_o) = E\left[\frac{1}{N}\sum_{f=1}^{N}\frac{f_r(\omega_f \to \omega_o) L_i(\omega_f)\, V(\mathrm{p}, \omega_f)\, \cos\theta_f}{p_f(\omega_f) + p_l(\omega_l)}\right] + E\left[\frac{1}{N}\sum_{f=1}^{N}\frac{f_r(\omega_l \to \omega_o) L_i(\omega_l)\, V(\mathrm{p}, \omega_l)\, \cos\theta_l}{p_f(\omega_f) + p_l(\omega_l)}\right]$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

The thing to do about this is to apply multiple importance sampling.
The idea is that you still sample both factors, but then you weight them in a way that accounts for both sampling techniques' probabilities of sampling a direction.
If one chooses a direction with low probability, the other sampling technique has a chance to weigh in and effectively say "that's actually an important direction as far as I'm concerned", and the samples are weighted in a way that eliminates those variance spikes.

```
L(\mathrm{p}, \omega_o) = & E \left[ \frac{1}{N} \sum_{f=1}^N \frac{f_r(\omega_f \rightarrow \omega_o) L_i(\omega_f) \, V(\mathrm{p}, \omega_f) \, \cos \theta_f}{p_f(\omega_f) + p_l(\omega_l)} \right] + \\
&  E \left[ \frac{1}{N} \sum_{f=1}^N \frac{f_r(\omega_l \rightarrow \omega_o) L_i(\omega_l) \, V(\mathrm{p}, \omega_l) \, \cos \theta_l}{p_f(\omega_f) + p_l(\omega_l)} \right]
```

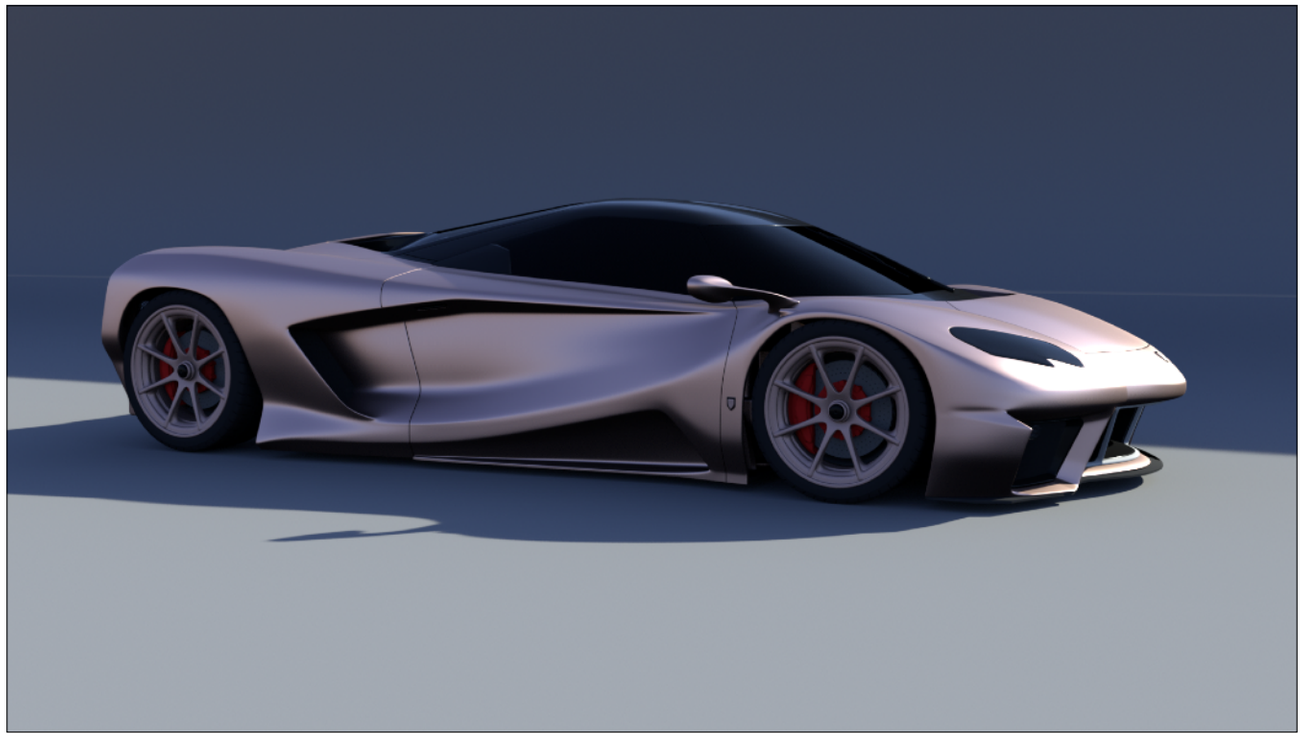


Here's multiple importance sampling with one sample per pixel.

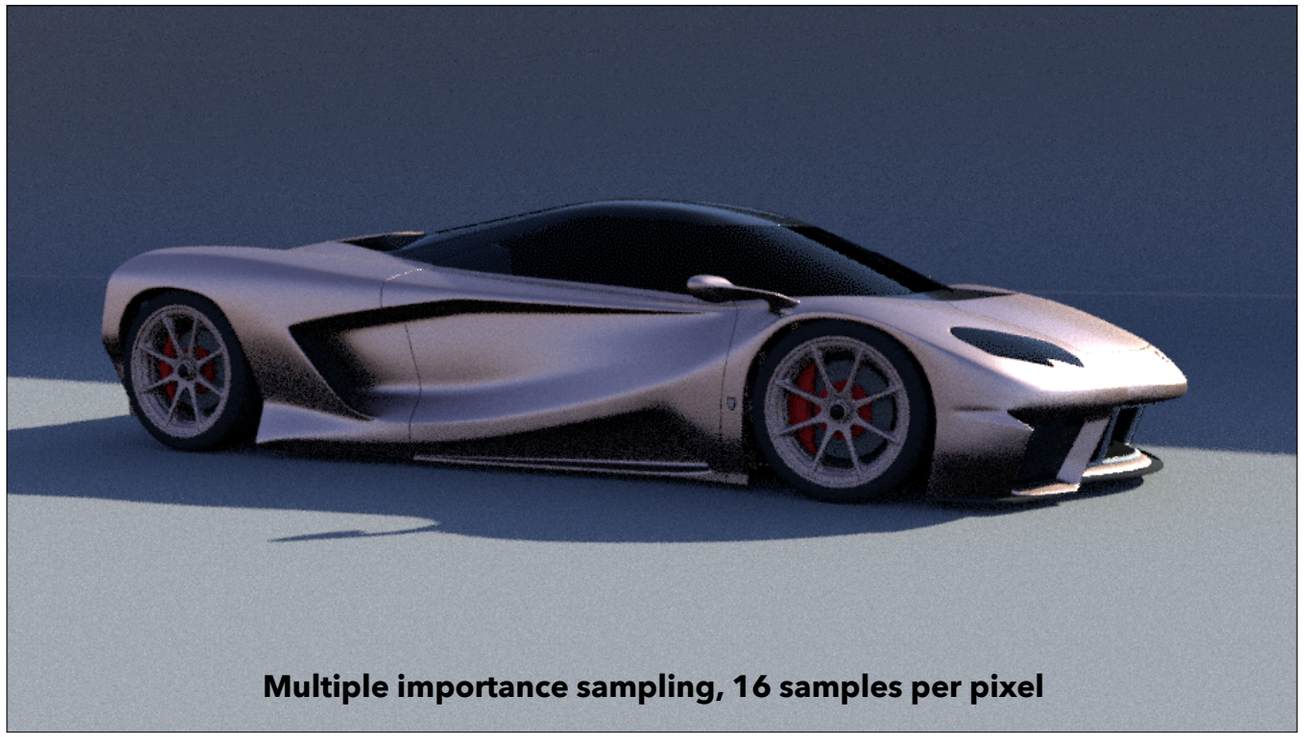
It's much better already.

There's till some noise in the car, but not everywhere.

It's good e.g. in that inner part of the door—where we saw that BSDF sampling happened to work well.

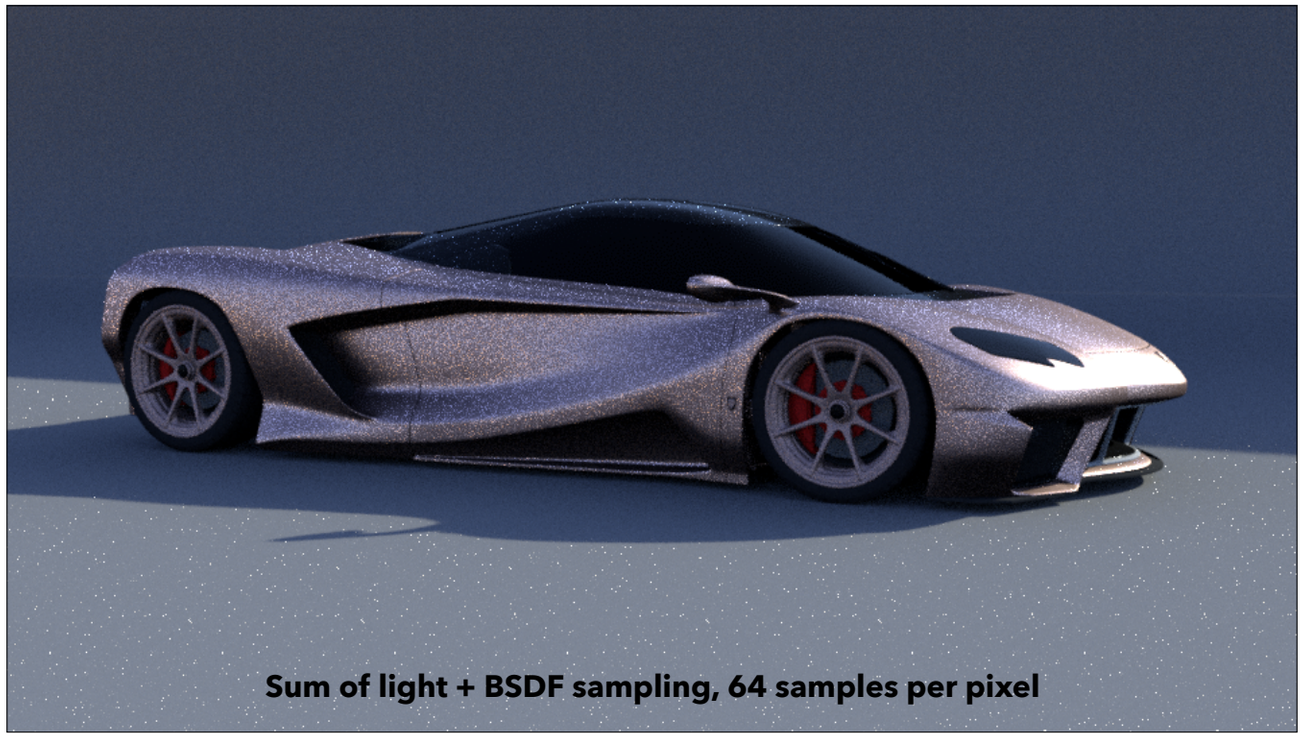


Compared to the reference, it’s not bad…



Multiple importance sampling, 16 samples per pixel

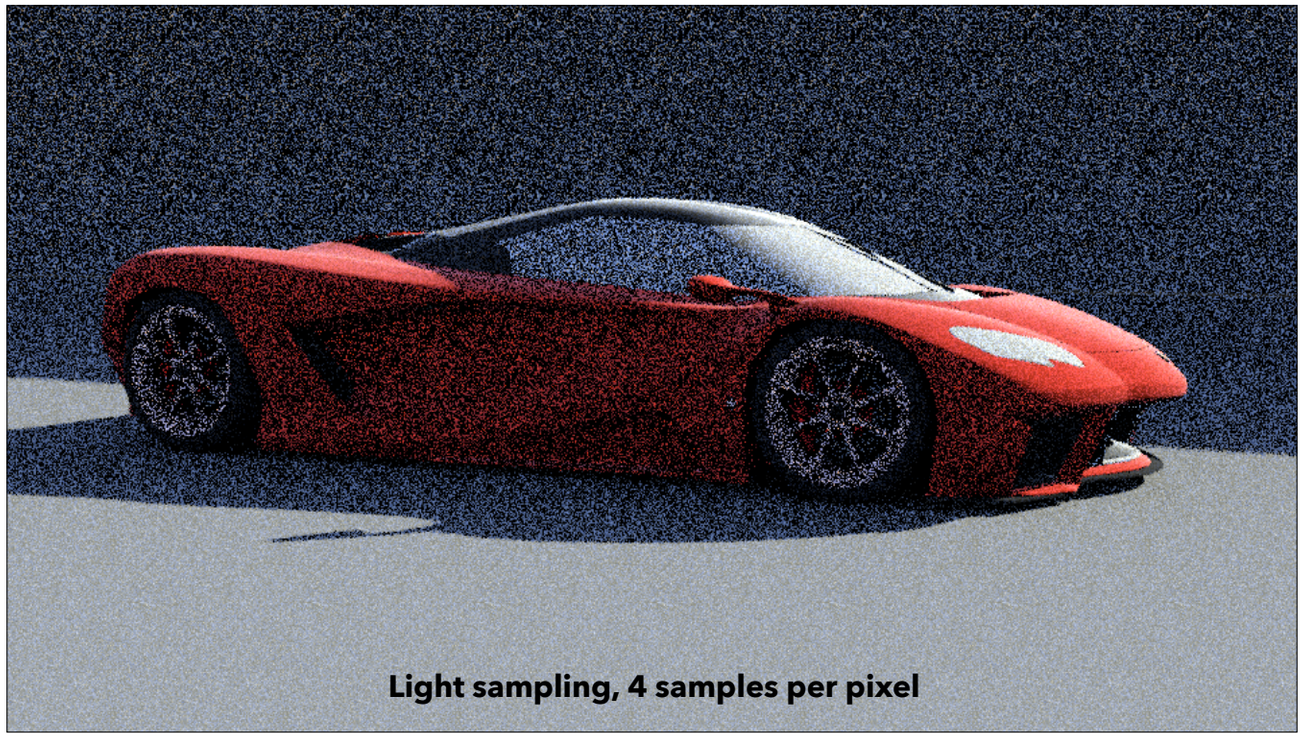
And with 16 samples per pixel, it’s looking pretty good.



Sum of light + BSDF sampling, 64 samples per pixel

For reference again, here is 64 samples per pixel, just adding the two sampling techniques, without using multiple importance sampling.

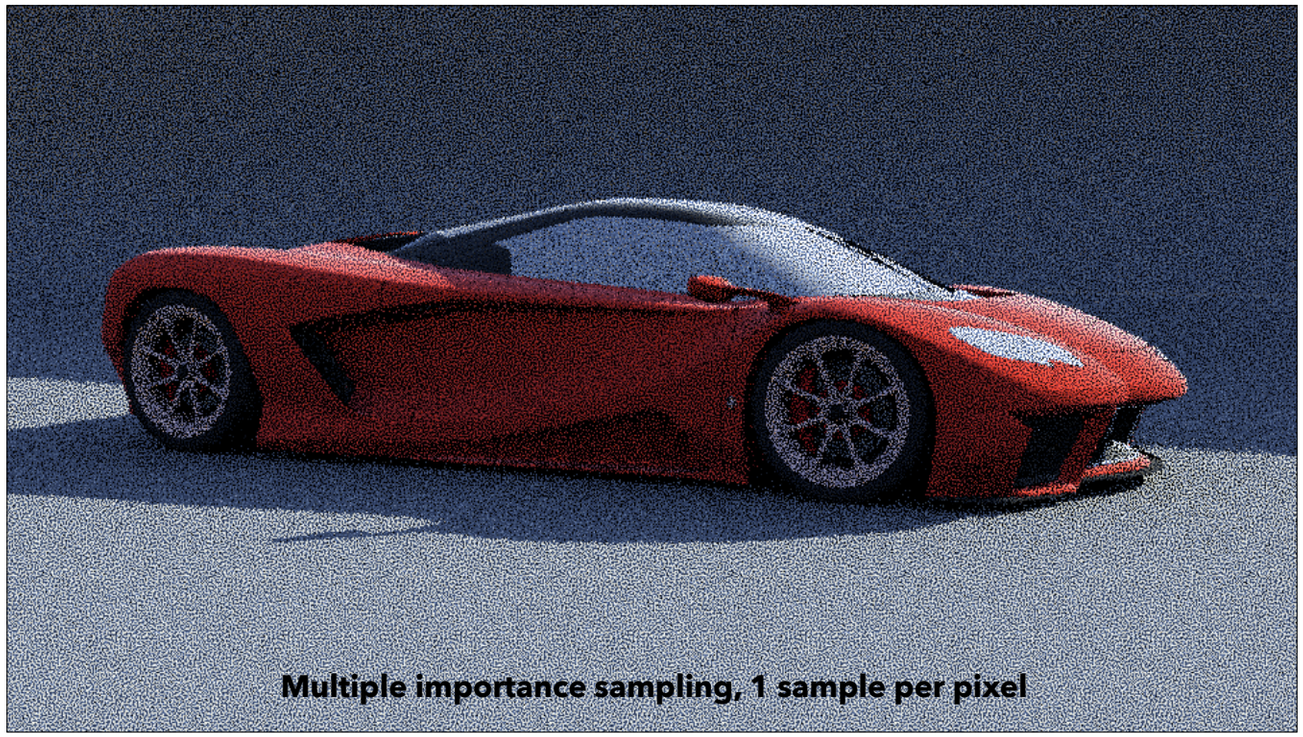
It’s much worse…



Light sampling, 4 samples per pixel

Multiple importance sampling even helps with that diffuse scene.

Here’s only sampling according to the light, with 4 samples per pixel.



And here's MIS, with one sample per pixel (but two rays).

The BRDF samples are just cosine-weighted hemispherical samples.

Mixing that in really helps especially in the regions of the image where the sun is shadowed.

There, many of the light samples will try to head for the sun, find themselves occluded, and contribute nothing.

The BRDF samples do a decent job of at least finding some light from the environment map.

Path Tracing

Ok, so even direct lighting with ray tracing offers plenty of challenges.

Let's move on to global illumination—using path tracing—which offers even more open problems.

Light Reflection at a Surface

$$L_o(\mathrm{p}, \omega_o) = L_e(\mathrm{p}, \omega_o) + \int_{H^2} L_i(\mathrm{p}, \omega_i)\, f_r(\mathrm{p}, \omega_i \to \omega_o)\, \cos\theta_i \, \mathrm{d}\omega_i$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Quick recap, here's the rendering equation.

It says that the reflected light at a point p in a direction w_o is given by directly emitted light—the first term on the right hand side—plus incident light reflected by the BRDF.

So the integral on the right is similar to the direct lighting integral, but it includes both direct and indirect illumination.

Light Reflection at a Surface

Reflected light

$$L_o(\mathrm{p}, \omega_o) = L_e(\mathrm{p}, \omega_o) + \int_{H^2} L_i(\mathrm{p}, \omega_i)\, f_r(\mathrm{p}, \omega_i \rightarrow \omega_o)\, \cos\theta_i \, \mathrm{d}\omega_i$$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Quick recap, here's the rendering equation.

It says that the reflected light at a point p in a direction w_o is given by directly emitted light—the first term on the right hand side—plus incident light reflected by the BRDF.

So the integral on the right is similar to the direct lighting integral, but it includes both direct and indirect illumination.

Light Reflection at a Surface

Reflected light

$$L_o(\mathrm{p}, \omega_o) = L_e(\mathrm{p}, \omega_o) + \int_{H^2} L_i(\mathrm{p}, \omega_i)\, f_r(\mathrm{p}, \omega_i \to \omega_o)\, \cos\theta_i \, \mathrm{d}\omega_i$$

Emitted light

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

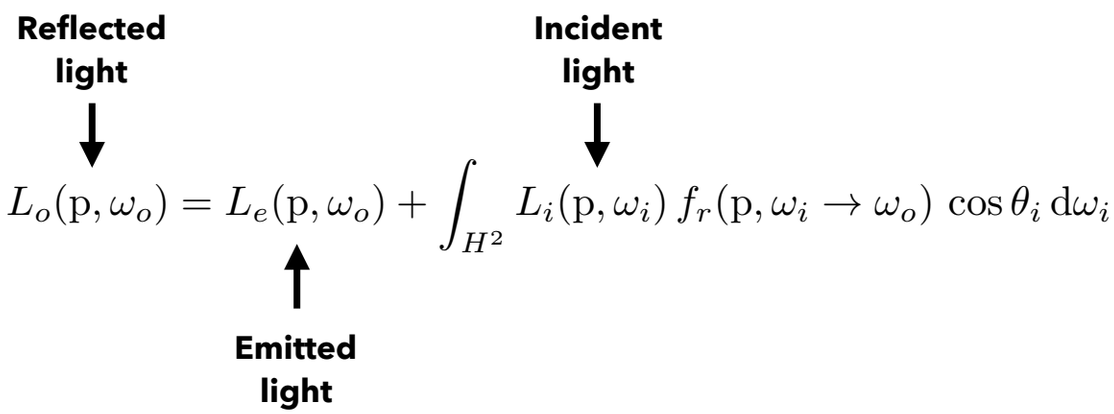
Matt Pharr

Quick recap, here's the rendering equation.

It says that the reflected light at a point p in a direction w_o is given by directly emitted light—the first term on the right hand side—plus incident light reflected by the BRDF.

So the integral on the right is similar to the direct lighting integral, but it includes both direct and indirect illumination.

Light Reflection at a Surface



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

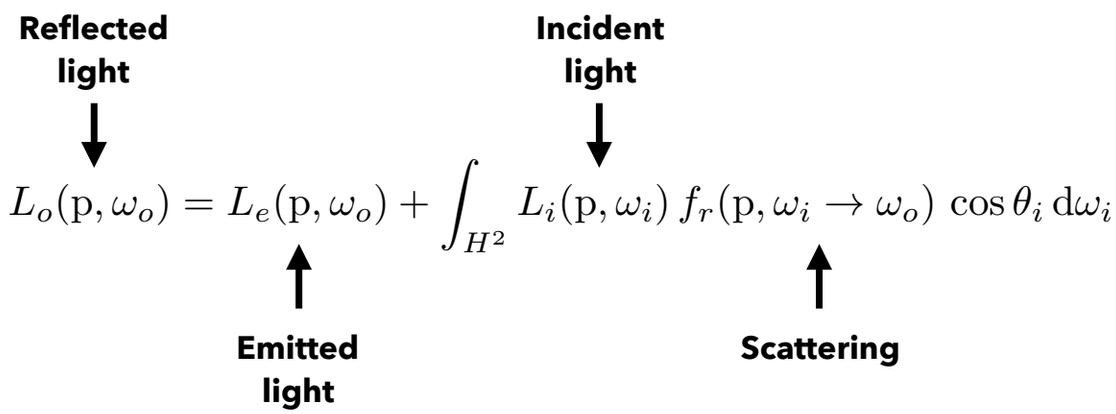
Matt Pharr

Quick recap, here's the rendering equation.

It says that the reflected light at a point p in a direction w_o is given by directly emitted light—the first term on the right hand side—plus incident light reflected by the BRDF.

So the integral on the right is similar to the direct lighting integral, but it includes both direct and indirect illumination.

Light Reflection at a Surface



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

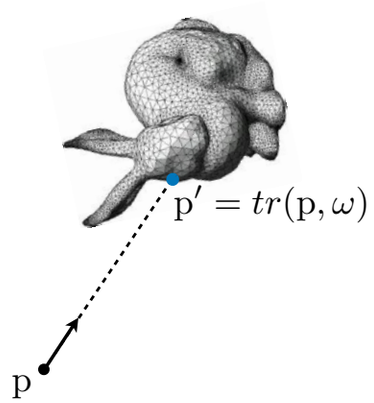
Quick recap, here's the rendering equation.

It says that the reflected light at a point p in a direction w_o is given by directly emitted light—the first term on the right hand side—plus incident light reflected by the BRDF.

So the integral on the right is similar to the direct lighting integral, but it includes both direct and indirect illumination.

Radiance Invariance Along Rays

$$L_i(\mathrm{p}, \omega_i) = L_o(tr(\mathrm{p}, \omega_i), -\omega_i)$$



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

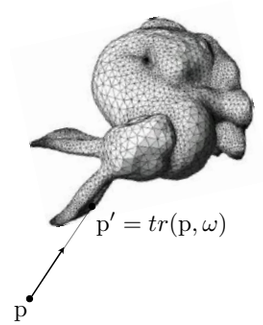
The key idea behind path tracing is to take advantage of the fact that the light carried by a ray between two mutually-visible points—the radiance—is unchanged along the ray (assuming that there's no participating media—that this is happening in a vacuum.)
That's close enough for many environments.
So what that says is that the incident radiance at a point p along a direction w_i is given by the radiance leaving the first visible surface along the ray from p in that direction.
We'll denote that p' and define tr as the ray tracing function that gives the first intersection point along the given ray.

L_o(\mathrm{p}, \omega_o) = L_e(\mathrm{p}, \omega_o) + \int_{H^2} f_r(\mathrm{p}, \omega_i \rightarrow \omega_o) \, L_o(tr(\mathrm{p}, \omega_i), -\omega_i) \, \cos \theta_i \, \mathrm{d}\omega_i

The Rendering Equation

Radiance invariance along rays:

$$L_i(\mathrm{p}, \omega_i) = L_o(tr(\mathrm{p}, \omega_i), -\omega_i)$$



$$L_o(\mathrm{p}, \omega_o) = L_e(\mathrm{p}, \omega_o) + \int_{H^2} L_o(tr(\mathrm{p}, \omega_i), -\omega_i)\, f_r(\mathrm{p}, \omega_i \rightarrow \omega_o)\, \cos\theta_i \, \mathrm{d}\omega_i$$

Light transport **Light scattering**

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And now what that means is that the incident radiance at p is given by the outgoing radiance at another point.

Notice what happened: the whole equation is written in terms of the outgoing radiance at points on surfaces in the scene.

That's pretty neat: there's this mutual coupling among all the points on surfaces in the scene.

And it's one equation with one unknown, which is generally a nice place to be.

L_o(\mathrm{p}, \omega_o) = L_e(\mathrm{p}, \omega_o) + \int_{H^2} f_r(\mathrm{p}, \omega_i \rightarrow \omega_o) \, L_o(tr(\mathrm{p}, \omega_i), -\omega_i) \, \cos \theta_i \, \mathrm{d}\omega_i

Path Tracing Estimator

- Sample incoming direction from some distribution (e.g. proportional to BRDF): $\omega_i \sim p(\omega)$
- Recursively evaluate path tracing function
- Estimator: $\dfrac{f_r(\omega_i \rightarrow \omega_o)\, L_o(tr(\mathrm{p}, \omega_i), -\omega_i)\, \cos\theta_i}{p(\omega_i)}$

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And we can apply the same old Monte Carlo trick to that integral: choose a direction from some distribution (generally based on the BRDF), and trace a recursive ray to get an estimate of the incident light.

Path Tracing

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

In practice, it's worthwhile to handle direct light and indirect light separately, so you get something like this.
You trace a ray from the camera (or rasterize a g-buffer for primary visibility—your choice), and then sample a light, trace a shadow ray to it, and then sample the BRDF to continue the path.

T(\bar{\mathrm{p}}) = \prod_j f_r(\mathrm{p}_j, \omega_{\mathrm{o}, j}, \omega_{\mathrm{i}, j}) \cos \theta_{\mathrm{i}, j}

L_i = T(\bar{\mathrm{p}}) L_e

Path Tracing

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

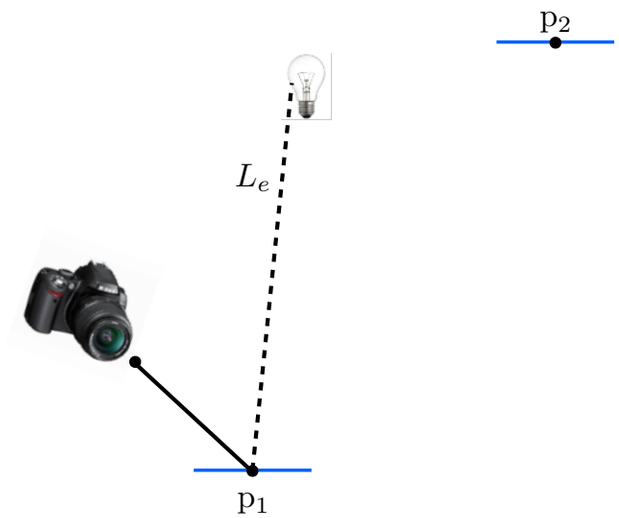
Matt Pharr

In practice, it's worthwhile to handle direct light and indirect light separately, so you get something like this.
You trace a ray from the camera (or rasterize a g-buffer for primary visibility—your choice), and then sample a light, trace a shadow ray to it, and then sample the BRDF to continue the path.

T(\bar{\mathrm{p}}) = \prod_j f_r(\mathrm{p}_j, \omega_{\mathrm{o}, j}, \omega_{\mathrm{i}, j}) \cos \theta_{\mathrm{i}, j}

L_i = T(\bar{\mathrm{p}}) L_e

Path Tracing



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

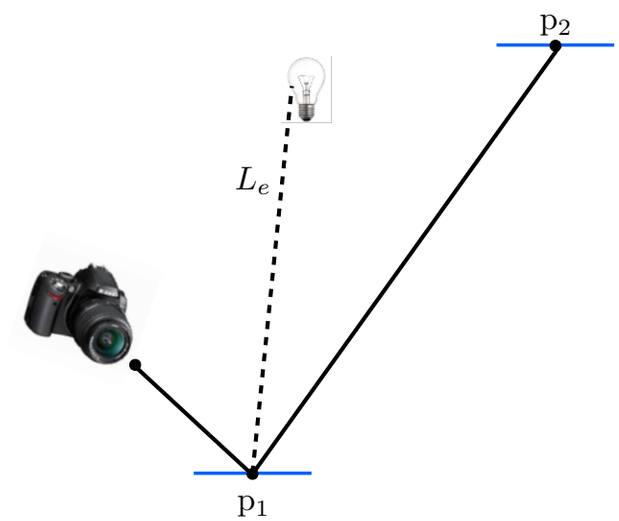
Matt Pharr

In practice, it's worthwhile to handle direct light and indirect light separately, so you get something like this.
You trace a ray from the camera (or rasterize a g-buffer for primary visibility—your choice), and then sample a light, trace a shadow ray to it, and then sample the BRDF to continue the path.

T(\bar{\mathrm{p}}) = \prod_j f_r(\mathrm{p}_j, \omega_{\mathrm{o}, j}, \omega_{\mathrm{i}, j}) \cos \theta_{\mathrm{i}, j}

L_i = T(\bar{\mathrm{p}}) L_e

Path Tracing



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

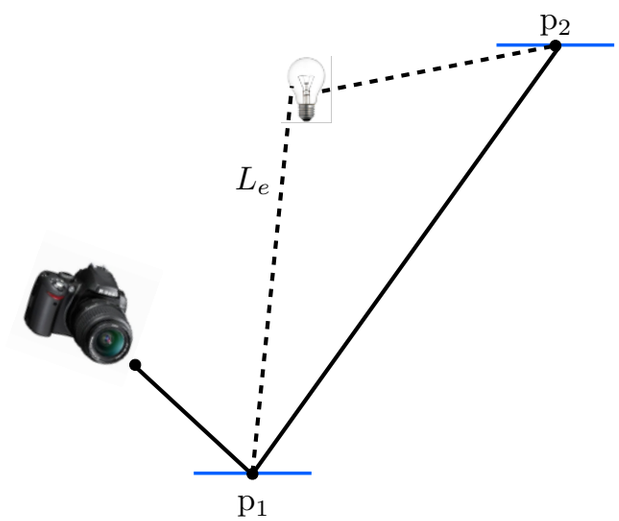
Matt Pharr

In practice, it's worthwhile to handle direct light and indirect light separately, so you get something like this.
You trace a ray from the camera (or rasterize a g-buffer for primary visibility—your choice), and then sample a light, trace a shadow ray to it, and then sample the BRDF to continue the path.

T(\bar{\mathrm{p}}) = \prod_j f_r(\mathrm{p}_j, \omega_{\mathrm{o}, j}, \omega_{\mathrm{i}, j}) \cos \theta_{\mathrm{i}, j}

L_i = T(\bar{\mathrm{p}}) L_e

Path Tracing



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

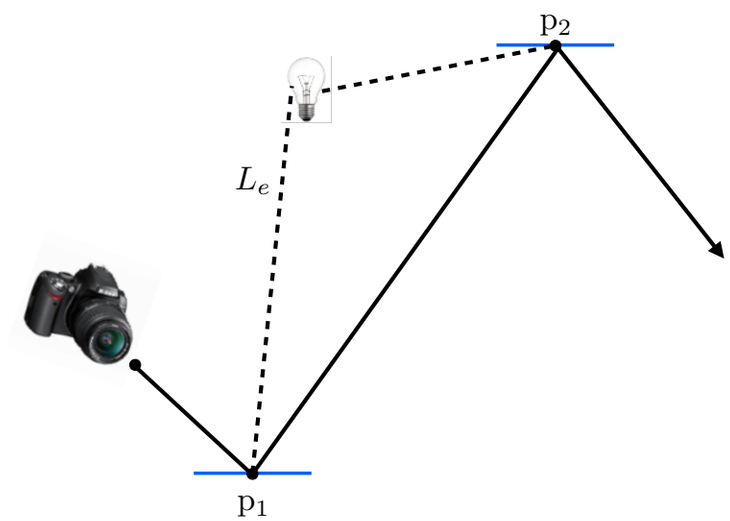
Matt Pharr

In practice, it's worthwhile to handle direct light and indirect light separately, so you get something like this.
You trace a ray from the camera (or rasterize a g-buffer for primary visibility—your choice), and then sample a light, trace a shadow ray to it, and then sample the BRDF to continue the path.

T(\bar{\mathrm{p}}) = \prod_j f_r(\mathrm{p}_j, \omega_{\mathrm{o}, j}, \omega_{\mathrm{i}, j}) \cos \theta_{\mathrm{i}, j}

L_i = T(\bar{\mathrm{p}}) L_e

Path Tracing



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

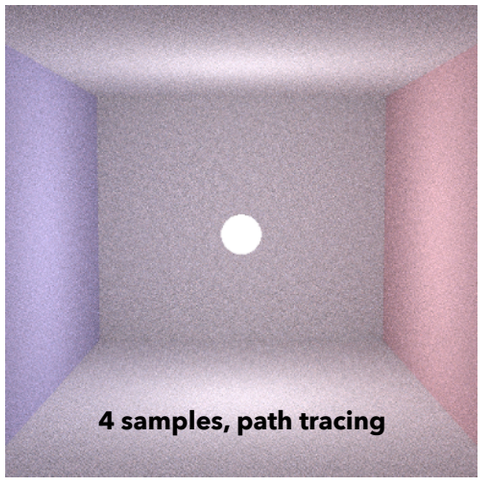
Matt Pharr

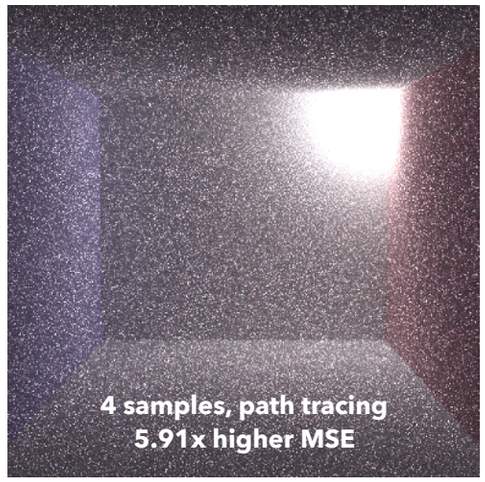
In practice, it's worthwhile to handle direct light and indirect light separately, so you get something like this.
You trace a ray from the camera (or rasterize a g-buffer for primary visibility—your choice), and then sample a light, trace a shadow ray to it, and then sample the BRDF to continue the path.

T(\bar{\mathrm{p}}) = \prod_j f_r(\mathrm{p}_j, \omega_{\mathrm{o}, j}, \omega_{\mathrm{i}, j}) \cos \theta_{\mathrm{i}, j}

L_i = T(\bar{\mathrm{p}}) L_e

Open Problem:
Sampling Tricky Indirect Lighting





Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And that often works well, but sometimes it doesn't.

So here we have the world's simplest scene: a box with a spherical light in it.

On the left, the light is in the center of the box, and on the right, it's in the corner.

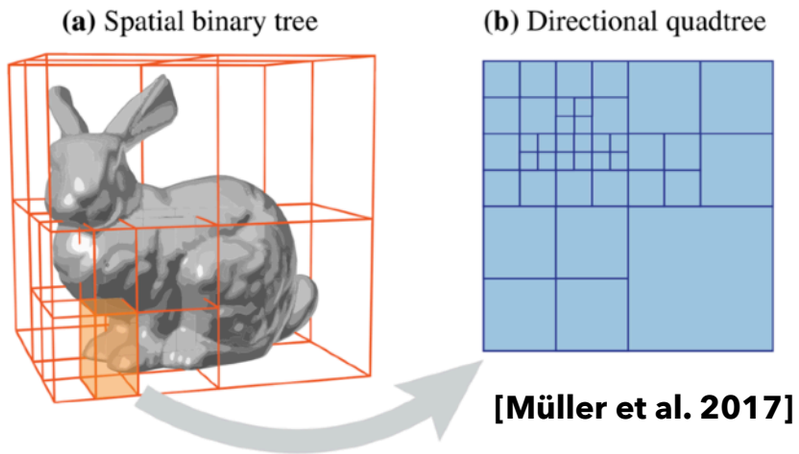
Just from moving the light, we have way more noise and way higher error.

The problem is that the indirect light is really concentrated—it's mostly coming from that corner, close to where the light is.

So at some point on the floor, for example, if we sample the BSDF to look for indirect lighting, we're more or less equally likely to go in all directions.

But it's like the environment map with the concentrated sun—you really want to head to the corner to find that indirect light.

Path Guiding



[Müller et al. 2017]

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

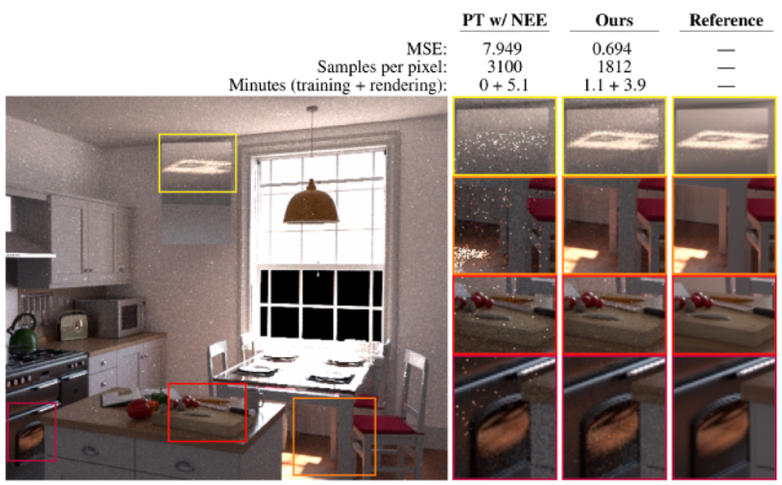
There's been a lot of work on path guiding to address this issue.

Müller et al's paper on practical path guiding is a really nice exemplar.

The idea is that as you trace rays, you record how much light you found from that point, in that direction.

Discretize space, discretize directions, and then use that to help sample future paths.

Path Guiding



[Müller et al. 2017]

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

It works really well, but the implementation is not GPU-friendly.

No one really knows how to do this effectively in real-time on massively parallel hardware: to learn from the results of previous rays, to build a data structure that helps inform the sampling of future rays.

But it can make a big difference and in particular helps make path tracing more robust—better able to just work in a wide variety of scenes.

There's a nice course on this topic coming up on Sunday.

Sampling Patterns

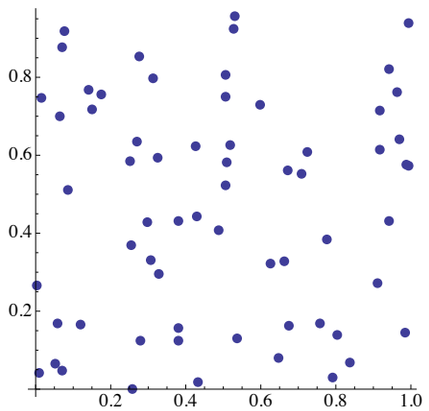
Next topic: sampling patterns.

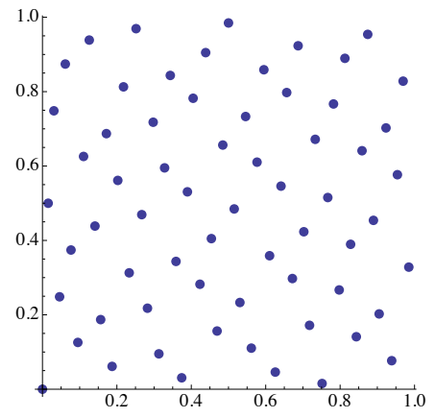
This is all about how do you generate the random (but not usually actually random) numbers you use for Monte Carlo integration.

This is another area where there's been lots of research, much of it focused to offline rendering.

There are lots of great ideas there, but they're not all clearly applicable to real-time.

Point-Sample Distributions





Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Say that you're choosing points on a light source using random numbers, mapping 2D values from the 2D unique square to points on the light.

You could use the points on the left or the points on the right.

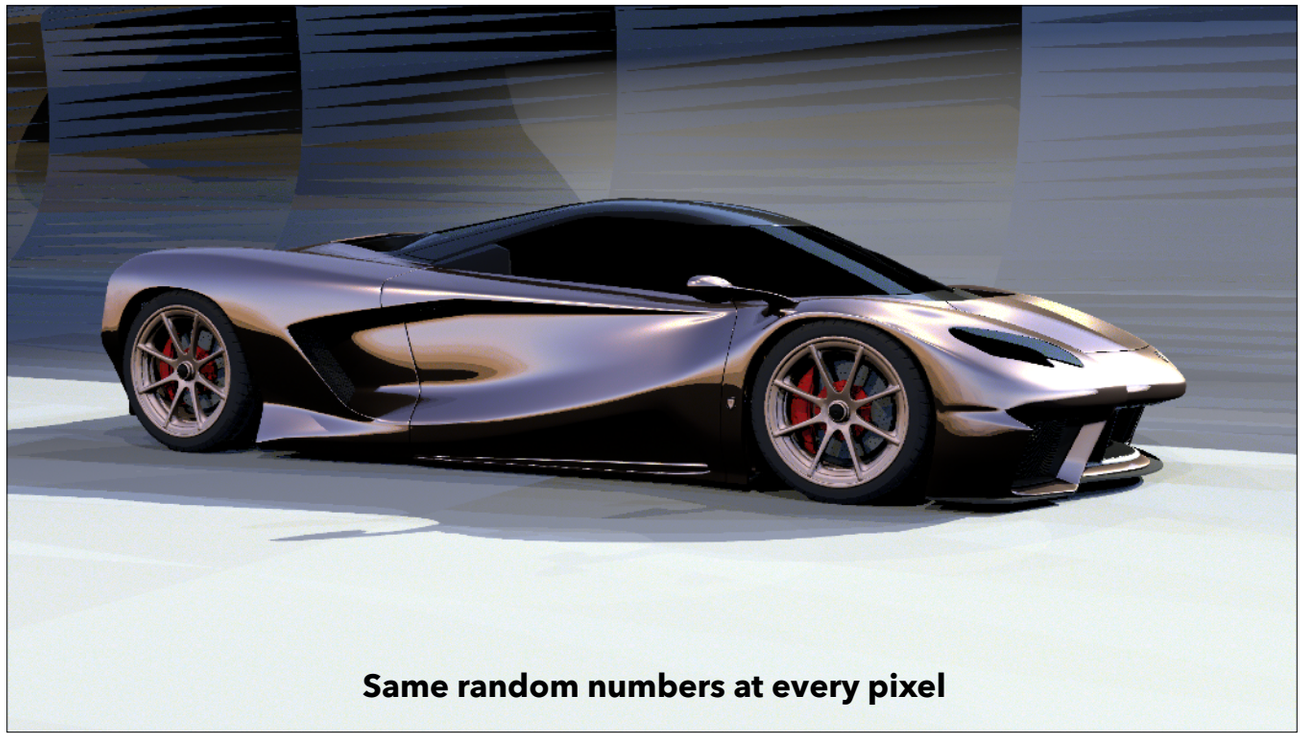
Which do you choose?

The right, right?

They're better distributed, less clumped together.

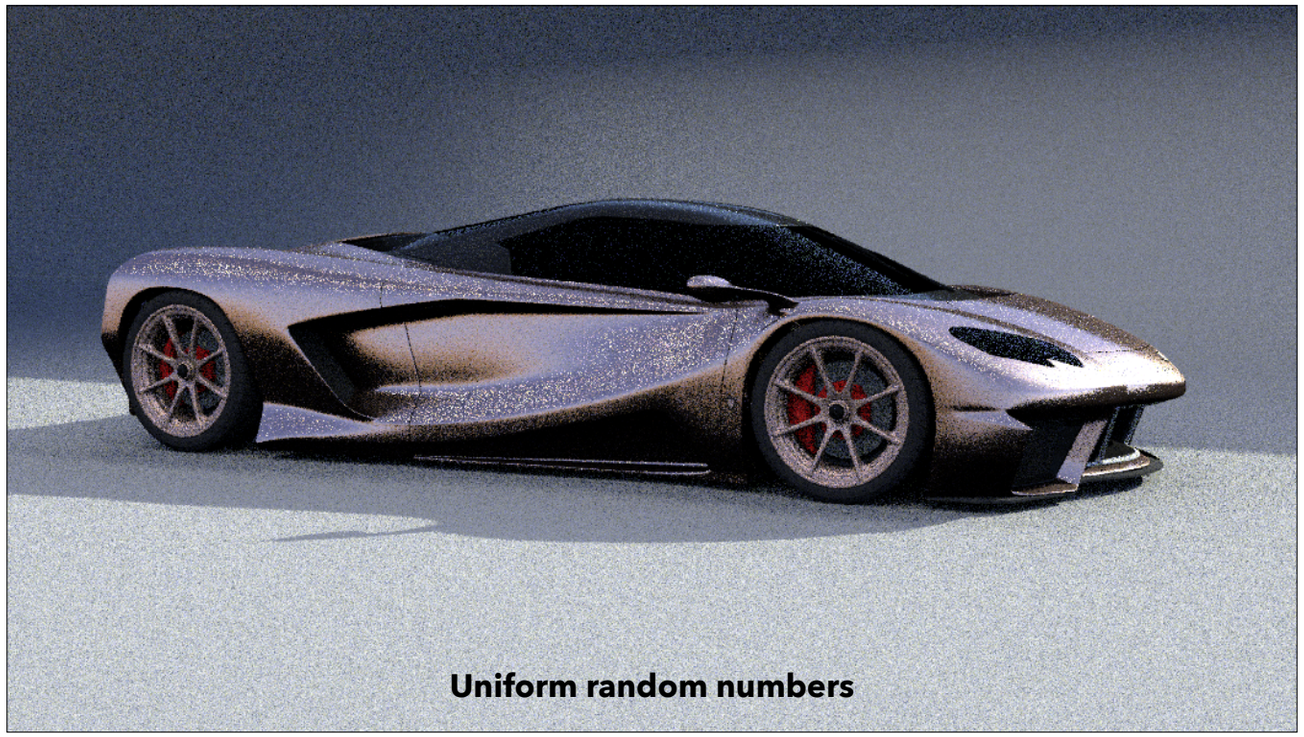
So you get a better distribution of points on the light, and it turns out that helps a lot for reducing error.

It's really powerful, since it's cheap to generate these points.



Same random numbers at every pixel

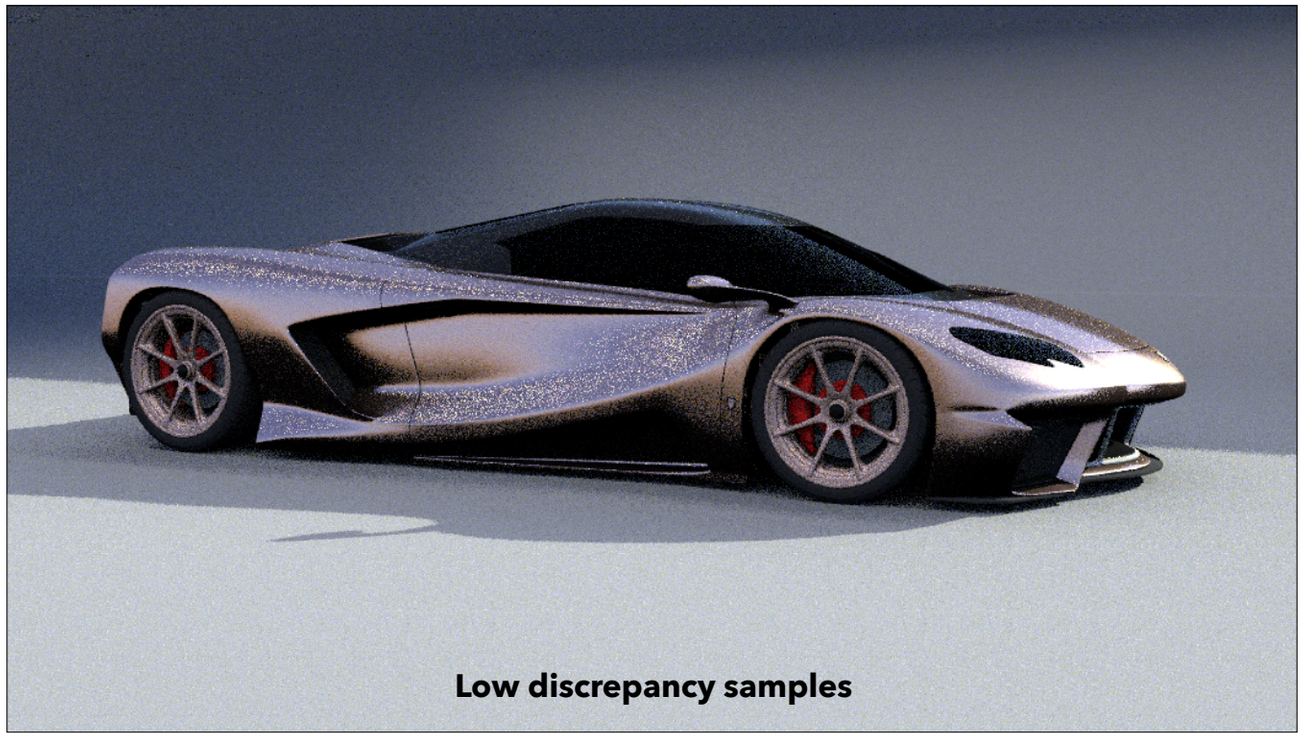
For starters, here's what happens if you use the exact same "random" numbers at each pixel, here again with our car with glossy BSDF and that HDR environment map.
This is 4 samples per pixel.
There's all this structure to the error in the image—Mach banding sorts of artifacts.
It's no good, and in particular you wouldn't want to throw this image at a denoiser; it wouldn't do well, in a sense because neighboring pixels don't offer any new information.



Here's that same image, still 4 samples per pixel, with uniform random numbers.

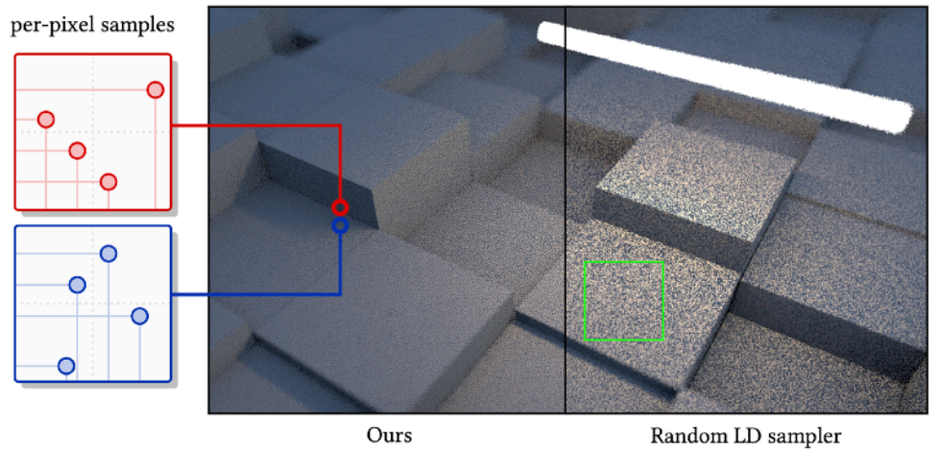
So each sample just picks 4 2D values, on its own, not worrying about which 2D values were chosen by the other samples.

Same number of rays, same amount of work, but visually, it's much better than using the same random numbers—all of that structure is broken up.



But here's 4spp with a low discrepancy sampling pattern—one that's more or less state of the art.

The same number of samples and the same number rays, but by being careful to distribute them well, you get this huge improvement in image quality, still for basically the same amount of work.



***A Low-Discrepancy Sampler that Distributes Monte Carlo Errors as a Blue Noise in Screen Space*, Heitz et al. 2019**

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

There's a cool talk by Eric Heitz and collaborators coming up on this topic later this week.

Here's one of their example images.

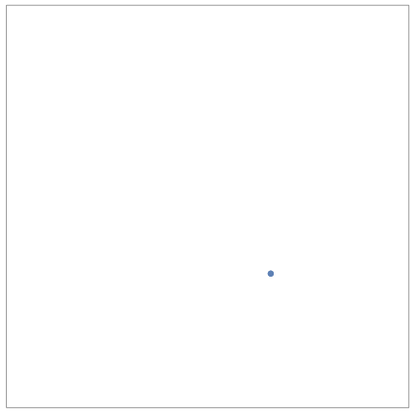
Both sides have the same number of samples (rays), and the crazy thing is, they both have about the same numerical error.

But they do a great job of decorrelating the error at nearby pixels.

In turn, the human visual system really likes that.

Because there aren't clumps of adjacent pixels that all suffer from the same error, we perceive the error to be much lower.

Sampling Well When Accumulating Frames



Progressive Sample Sequence

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Real-time rendering makes a lot of use of accumulating samples.

Think TAA for example: lots of reprojected previous frames' results contribute to the current frame.

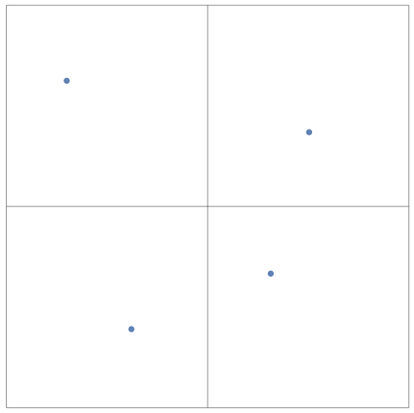
That interacts with sampling patterns.

Progressive sample sequences can be really useful for this: the idea is that as you include more and more samples in a pixel, you'd like them to be well distributed with respect to the samples you already took in previous frames.

Here's an example.

Here's the first sample: ok, fine, it's a point in the unit square.

Sampling Well When Accumulating Frames



Progressive Sample Sequence

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

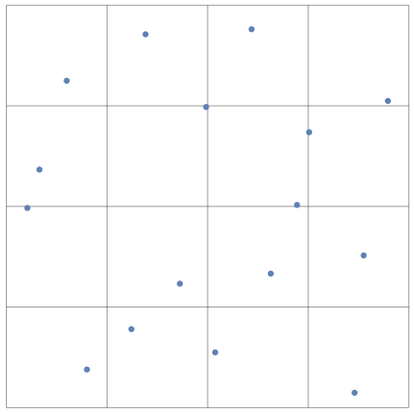
Matt Pharr

Now, intuitively, it'd be a good thing if the next three samples were all well distributed with respect to each other.

Here we've split the square into four strata and made sure that the next three samples fill in the empty 2x2 strata.

Incidentally, these points were generated using an algorithm that Per Christensen and collaborators described in a recent EGSR paper.

Sampling Well When Accumulating Frames



Progressive Sample Sequence

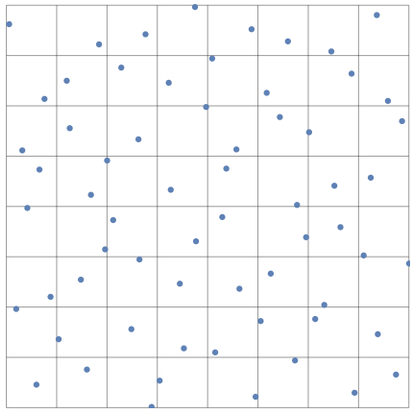
Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And you can continue this.

We've added 12 more samples to the 4 we had, going up to a 4x4 stratification, filling in the empty boxes.

Sampling Well When Accumulating Frames



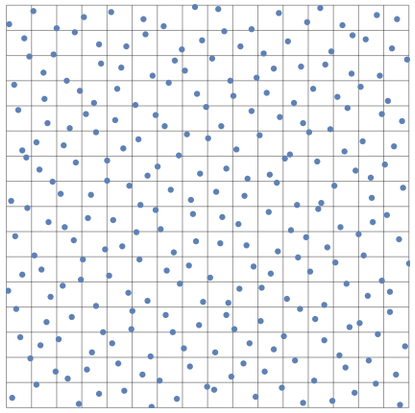
Progressive Sample Sequence

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

And so forth…

Sampling Well When Accumulating Frames

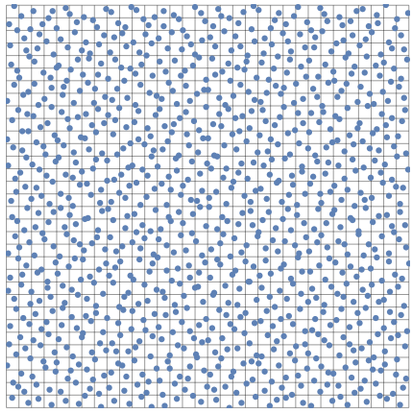


Progressive Sample Sequence

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Sampling Well When Accumulating Frames



Progressive Sample Sequence

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

So we've both got this really good set of sample points but we've also generated them progressively, sort of filling in the biggest holes as we go.

Open Problem:
Sampling Well Under Reprojection

Cool, so we can generate sets of samples that are distributed well w.r.t. each other as the sample count increases.

But there's a catch.

Say that we've generated these four samples on the left at the green pixel in the first frame.

Then, on the right, the camera has moved.

If we're doing a TAA or SVGF sort of thing, we reproject pixel values from the previous frame to the current frame.

But now if we want to generate more samples at the green pixel on the right, we'd like them to be well distributed with respect to the samples from the other pixels that were reprojected into it.

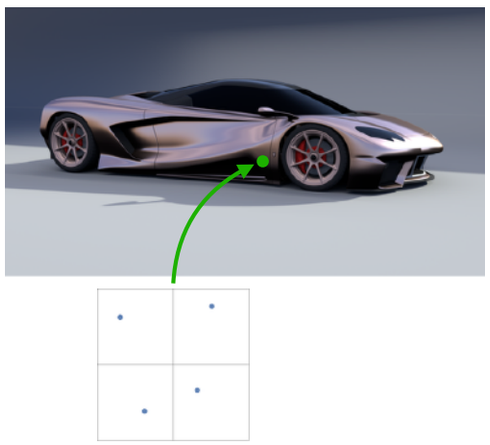
I don't know of a good way to do this.

So this means that, in the current world of retrojecting pixels, we can't enjoy all those benefits of well-distributed sample points.

We've seen how much they help, so it's sad to lose this.

I think there's a big open problem here: solving that problem.

Open Problem:
Sampling Well Under Reprojection



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Cool, so we can generate sets of samples that are distributed well w.r.t. each other as the sample count increases.

But there's a catch.

Say that we've generated these four samples on the left at the green pixel in the first frame.

Then, on the right, the camera has moved.

If we're doing a TAA or SVGF sort of thing, we reproject pixel values from the previous frame to the current frame.

But now if we want to generate more samples at the green pixel on the right, we'd like them to be well distributed with respect to the samples from the other pixels that were reprojected into it.

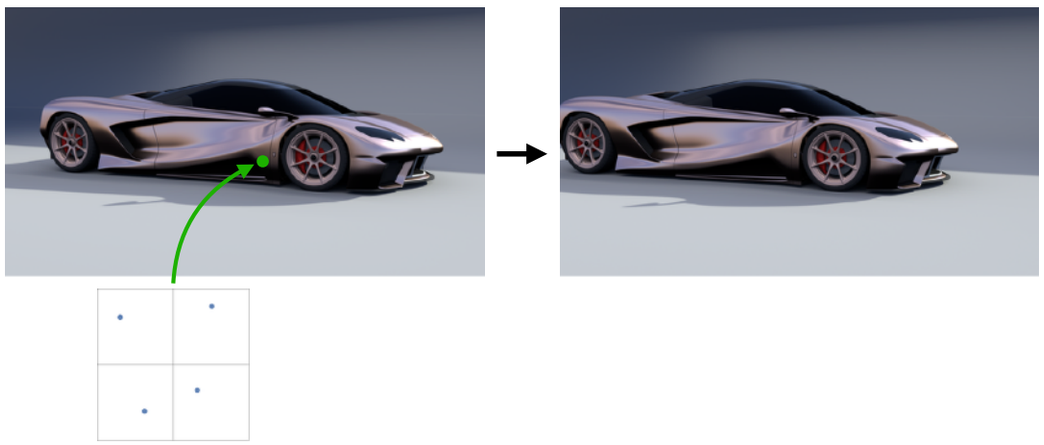
I don't know of a good way to do this.

So this means that, in the current world of retrojecting pixels, we can't enjoy all those benefits of well-distributed sample points.

We've seen how much they help, so it's sad to lose this.

I think there's a big open problem here: solving that problem.

Open Problem:
Sampling Well Under Reprojection



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Cool, so we can generate sets of samples that are distributed well w.r.t. each other as the sample count increases.

But there's a catch.

Say that we've generated these four samples on the left at the green pixel in the first frame.

Then, on the right, the camera has moved.

If we're doing a TAA or SVGF sort of thing, we reproject pixel values from the previous frame to the current frame.

But now if we want to generate more samples at the green pixel on the right, we'd like them to be well distributed with respect to the samples from the other pixels that were reprojected into it.

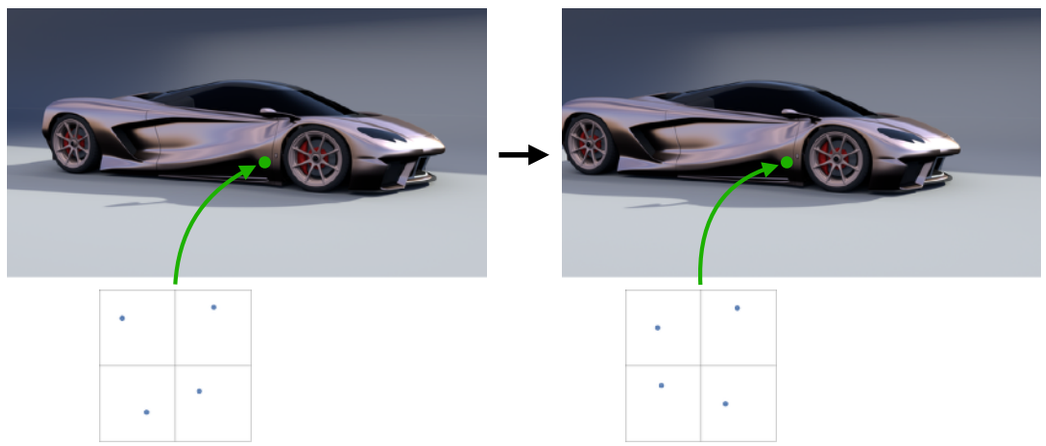
I don't know of a good way to do this.

So this means that, in the current world of retrojecting pixels, we can't enjoy all those benefits of well-distributed sample points.

We've seen how much they help, so it's sad to lose this.

I think there's a big open problem here: solving that problem.

Open Problem:
Sampling Well Under Reprojection



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Cool, so we can generate sets of samples that are distributed well w.r.t. each other as the sample count increases.

But there's a catch.

Say that we've generated these four samples on the left at the green pixel in the first frame.

Then, on the right, the camera has moved.

If we're doing a TAA or SVGF sort of thing, we reproject pixel values from the previous frame to the current frame.

But now if we want to generate more samples at the green pixel on the right, we'd like them to be well distributed with respect to the samples from the other pixels that were reprojected into it.

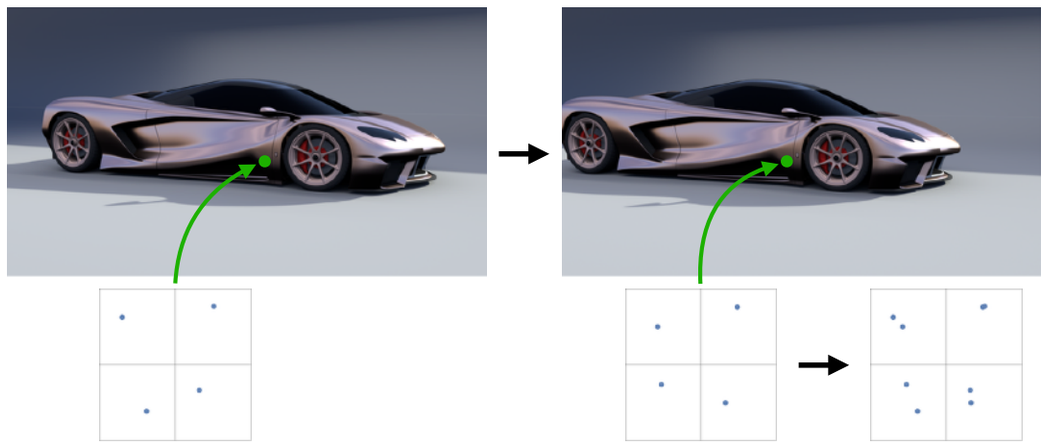
I don't know of a good way to do this.

So this means that, in the current world of retrojecting pixels, we can't enjoy all those benefits of well-distributed sample points.

We've seen how much they help, so it's sad to lose this.

I think there's a big open problem here: solving that problem.

Open Problem:
Sampling Well Under Reprojection



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

Cool, so we can generate sets of samples that are distributed well w.r.t. each other as the sample count increases.

But there's a catch.

Say that we've generated these four samples on the left at the green pixel in the first frame.

Then, on the right, the camera has moved.

If we're doing a TAA or SVGF sort of thing, we reproject pixel values from the previous frame to the current frame.

But now if we want to generate more samples at the green pixel on the right, we'd like them to be well distributed with respect to the samples from the other pixels that were reprojected into it.

I don't know of a good way to do this.

So this means that, in the current world of retrojecting pixels, we can't enjoy all those benefits of well-distributed sample points.

We've seen how much they help, so it's sad to lose this.

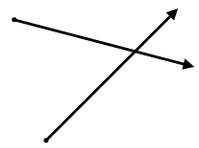
I think there's a big open problem here: solving that problem.

Coherence (and not)

Next topic: ray coherence, incoherence, and the tension between performance and information.

My Two Favorite Rays

Physics



Math

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

On one hand, two rays that are almost the same are cheaper to trace than two rays that are different.

That's just life: they're going to access similar parts of the scene, have better memory coherence, will require less bandwidth, and be cheaper in time and in energy.

Physics imposes that.

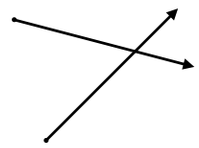
… but, we've seen that different rays have a lot of value. Remember that example of Eric Heitz et al's blue noise samples? They worked hard to decorrelate the rays at nearby pixels, and it looked way better.

And a denoiser is going to be happy if nearby pixels traced different rays—it has more information to work with.

But those rays are probably going to be more expensive to trace, no matter how hard IHVs work to make them not too expensive.

My Two Favorite Rays

Physics



Math
and the human visual system

On one hand, two rays that are almost the same are cheaper to trace than two rays that are different.

That's just life: they're going to access similar parts of the scene, have better memory coherence, will require less bandwidth, and be cheaper in time and in energy.

Physics imposes that.

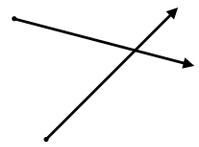
… but, we've seen that different rays have a lot of value. Remember that example of Eric Heitz et al's blue noise samples? They worked hard to decorrelate the rays at nearby pixels, and it looked way better.

And a denoiser is going to be happy if nearby pixels traced different rays—it has more information to work with.

But those rays are probably going to be more expensive to trace, no matter how hard IHVs work to make them not too expensive.

My Two Favorite Rays

Physics



Math
and the human visual system
and the denoiser…

On one hand, two rays that are almost the same are cheaper to trace than two rays that are different.

That's just life: they're going to access similar parts of the scene, have better memory coherence, will require less bandwidth, and be cheaper in time and in energy.

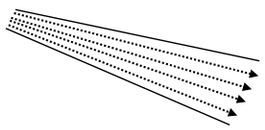
Physics imposes that.

… but, we've seen that different rays have a lot of value. Remember that example of Eric Heitz et al's blue noise samples? They worked hard to decorrelate the rays at nearby pixels, and it looked way better.

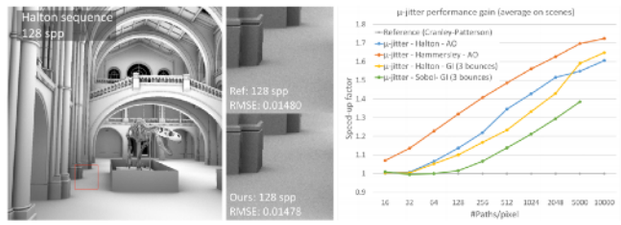
And a denoiser is going to be happy if nearby pixels traced different rays—it has more information to work with.

But those rays are probably going to be more expensive to trace, no matter how hard IHVs work to make them not too expensive.

Open Problem:
Finding (or Imposing) Coherence



Ray Frusta?



Cache-Friendly Micro-Jittered Sampling
Dufay et al. 2016

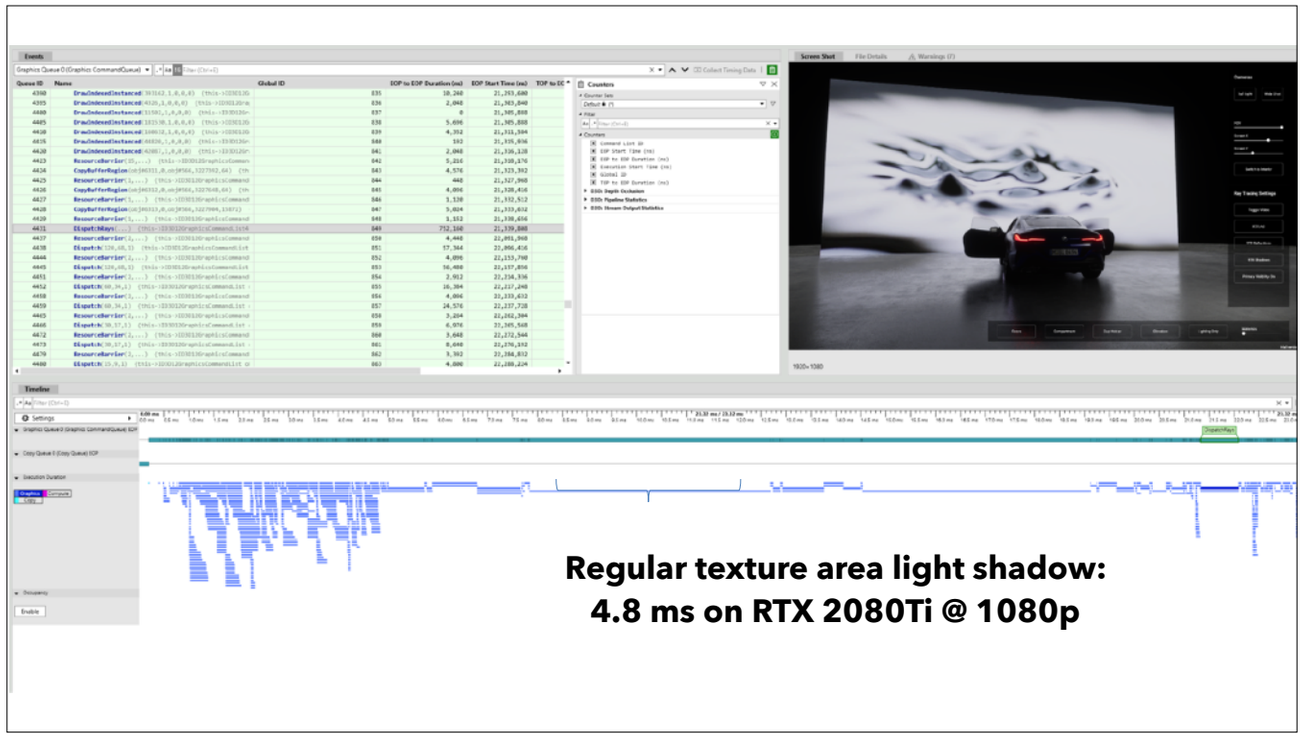
I think there's a big interesting question here.

Frusta / ray packets were a big deal in CPU ray tracing for a long time, until people generally realized that tracing fewer incoherent rays was more important.

But, architectures and algorithms have changed.

There's a tension here as well with everything we talked about before about focusing your effort on sampling what's most important—that, too, generally leads to incoherent rays.

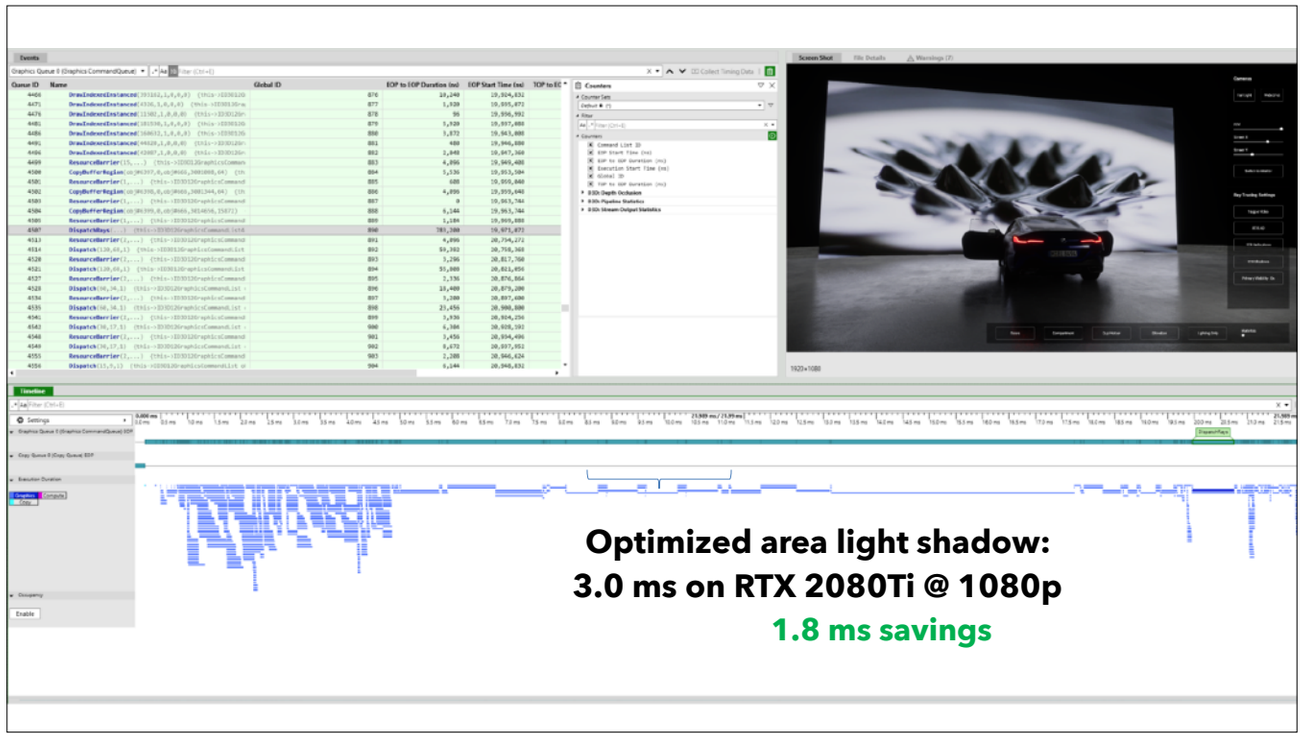
There was an interesting talk on this topic at SIGGRAPH a few years agony Dufay et al. about trying to balance these trade-offs.



And Natasha has an example of this in her GDC talk about ray tracing in Unity.

They have this scene with a big area light source—that screen—and are tracing 4 shadow rays at each pixel.

If you just take 4 samples and trace those rays, this ran in 4.8ms.



They tried splitting the light source into 4 quadrants and first traced the shadow rays to the first quadrant, then the second, and so on.

It was 1.6x faster! A 1.8ms improvement, down to 3ms per frame.

It’s a simple sorting, but it mattered a lot.

Similarly, BF V does a bit of ray sorting by direction, using an octahedral map.

Unity has also reported good results from doing that as well.

From the HW side, obviously IHVs want to make incoherent rays work well in all cases.

But often, developers have insight into structure of their rays, or have more flexibility in the order in which they’re generated.

It’s an interesting area.

Denoising

A few thoughts about denoising…

It’s been a huge part of making real-time ray tracing possible, given the relatively low numbers of rays that can be traced per pixel.

It’s definitely not yet a solved problem.

Open Problems:
Denoising and Temporal Reprojection

- Denoising: generally per-effect, per-light
  - Can a single denoiser handle everything?
- Temporal sample reuse:
  - You can't not do it… But can it be more rigorous?
  - Can we bound the error?

In practice, people are mostly using per-effect denoisers these days: one for reflections, one for AO, and one for each light for shadows.
That adds up to a lot of denoising passes, and sometimes it can be the bottleneck, more so than ray tracing.
It'd be fantastic to have a single denoiser that just worked for everything.
Another open problem comes in the form of specular and glossy reflections.
Denoisers these days depend on temporal reuse—reprojecting previous frames and using them to denoise the current frame.
That gets tricky with specular reflection: how do you compute a motion vector at a specular pixel.
Strictly speaking, there are multiple motion vectors involved, so it gets messy quickly—consider consider glass: light is reflected and transmitted, and may happen multiple times…

Open Problems: Denoising and Temporal Reprojection

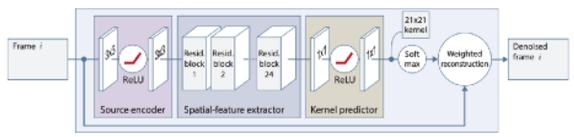
- Denoising: generally per-effect, per-light
  - Can a single denoiser handle everything?
- Temporal sample reuse:
  - You can't not do it… But can it be more rigorous?
  - Can we bound the error?

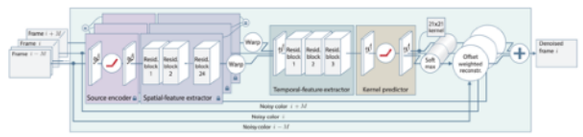
Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

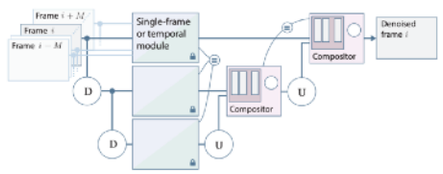
Matt Pharr

In practice, people are mostly using per-effect denoisers these days: one for reflections, one for AO, and one for each light for shadows.
That adds up to a lot of denoising passes, and sometimes it can be the bottleneck, more so than ray tracing.
It'd be fantastic to have a single denoiser that just worked for everything.
Another open problem comes in the form of specular and glossy reflections.
Denoisers these days depend on temporal reuse—reprojecting previous frames and using them to denoise the current frame.
That gets tricky with specular reflection: how do you compute a motion vector at a specular pixel.
Strictly speaking, there are multiple motion vectors involved, so it gets messy quickly—consider consider glass: light is reflected and transmitted, and may happen multiple times…

So You're Telling Me There's a Chance…







(10 spf)

Denoising with Kernel Prediction and Asymmetric Loss Functions
Vogels et al. 2018

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

The good news is that offline folks have shown that a single denoiser can work well; check out this SIGGRAPH paper from last year.

However, there's a bit of a gap to bridge.

First, they expect relatively high sample rates—think 64 samples per pixel.

And then they use fairly complex neural nets.

They denoise at 10 spf.

That's seconds per frame, so is about 600x off from real-time, if you spend the whole time denoising.

So call it 6000x off from practical for real time.

It'd be amazing to bridge that gap, though…

Visibility Data Structures

Ok, one last topic.

To me, one of the most exciting things about GPU ray tracing is the idea that for the first time since the start of real-time graphics hardware, hardware has provided a completely new visibility algorithm.

It's been rasterization (with all sorts of tweaks) throughout, but now the hardware gives us the capability of making high-performance incoherent visibility queries.

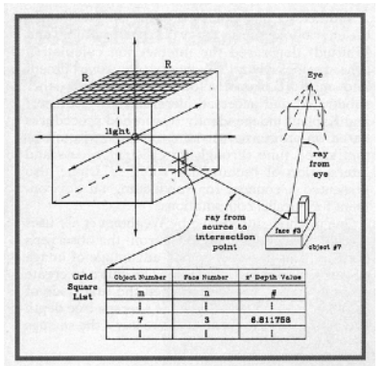
It's a huge change, and it opens up all sorts of interesting new things to look at.

Open Problem: Caching Visibility

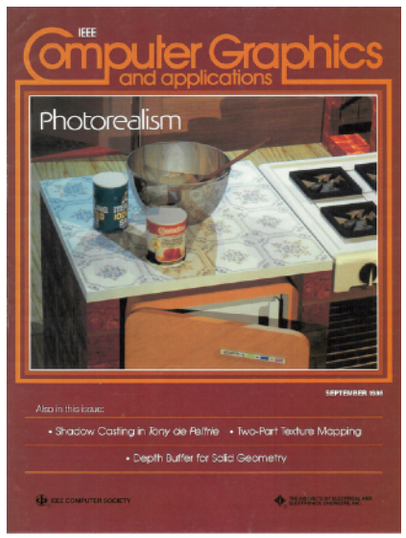
- Easy with the rasterizer: the hardware generates the data structure itself. (At least for starters)
- Ray tracing: anything's possible, but rays are tricky to sort and index (5/6D)
- Opportunity—low-res and adaptive structures:
  - e.g. DDGI [McGuire et al. 2019]: low-res directional probes of color, distance, distance^2.

Caching and resampling visibility results has been a huge part of raster-based graphics: think shadow maps and environment maps, and all that sort of stuff.
The hardware generates visibility information with the rasterizer in the form you ask it to (e.g. depth at each pixel), you hold on to it, and then you can re-use it, either sampling it or resampling it.
Now, a lot of that resampling (and the error that comes with it) isn't necessary if you can trace the actual ray you're interested in.
But you can't trace an infinite number of rays, so it's still worth holding on to visibility results if they're going to be useful again.
With ray tracing, it's not obvious how to do that, though—there's no simple 2D data structure like an image.
Rays are 5 dimensional things, and it's not obvious how to sort and index them.
One interesting recent example is McGuire et al's DDGI [now RTXGI], which has a camera-space adaptive grid of probes that store a low-res distance and squared distance buffer.

The Light Buffer



***The Light Buffer*,**
Haines and Greenberg, 1986



Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

One of the nice things about ray tracing is that it opens up the door to visibility data structures that sometimes give the answer but may also not have an answer—and in that case, you go ahead and trace a ray.

For the idea of it, there's a classic idea called the light buffer.

The idea is that for a point light source, you discretize a cube map of directions around it.

In each cube map pixel, you store one potentially-occluding triangle.

To compute shadows from the light, you first intersect against the triangle for the direction to the point being shaded.

If it's a hit, you know the light isn't visible; otherwise, you trace real ray against the scene geometry.

Note that this probably isn't a good idea with modern ray tracing hardware; it probably wouldn't be a win.

But the general idea is good food for thought, in thinking about ray tracing as a fall-back—are you able to peel away easy 'yes'es and easy 'no's in some manner and then trace rays only for the uncertain cases…

Path Space Hashing

- descriptors for selected vertices include

world space location x and optionally normal n, incident angle ω, and BRDF layer

- storing and loading data using hashed quantized descriptors
  - trade a larger hash table size for faster access (proportional to number of paths)
  - use a second hash of the descriptor instead of storing full keys
  - linear probing for collision resolution

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

Path Tracing for Future Games(??!!?)
Open Problems in Real-Time Rendering 2019

Matt Pharr

It's easy to organize rays around a point light source, but not so easy in general in 3D.

A neat idea that can be used to do this is "path space hashing".

The idea is to do a n-dimensional hash of attributes (ray origin, direction, local surface normal, whatever), where each dimension is quantized before hashing.

Thus, if you're storing rays—points and directions—then nearby points and directions hash to the same value.

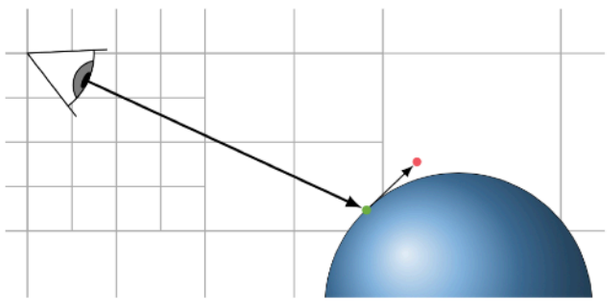
It's super efficient to do lookups in these hash tables, and they're GPU-friendly to update as well.

Path Space Hashing

- finding the hash table location i

$$l \leftarrow \text{level\_of\_detail}(|p_{\text{cam}} - x|)$$
$$x' \leftarrow x + \text{jitter}(n) \cdot \mathit{scale} \cdot 2^l$$
$$l' \leftarrow \text{level\_of\_detail}(|p_{\text{cam}} - x'|)$$
$$\tilde{x} \leftarrow \left\lfloor \frac{x'}{\mathit{scale} \cdot 2^{l'}} \right\rfloor$$
$$i \leftarrow \text{hash}(\tilde{x}, \ldots) \ \% \ \text{table\_size}$$
$$v \leftarrow \text{hash2}(\tilde{x}, n, \ldots)$$

for both averaging and querying



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

You can also do clever things like adapt the discretization based on distance.

Thus, you have a finer hash grid closer to the camera and so forth.

There's lots more to do in this area, but I think hashing is a promising direction for massively parallel hardware like GPUs.

Open Problem: Conservative Visibility

- Is there a ray tracing equivalent to conservative rasterization?
  - Efficient / dynamic data structure that takes a pair of points, reports “mutually visible”, “invisible”, or “not sure”.
  - Bonus? Guess + confidence level for “not sure”
  - Trace rays when you’re not sure

Ok, one last open problem.

As befitting the last one, this is the most hare-brained / hand-wavey.

With rasterization, we’ve now got the option to do conservative rasterization—that’s basically to be able to say for some frustum from some point, is a triangle visible in any of it, or, conversely, is a triangle visible in none of it.

So building on that, it’s possible to determine if particular frusta are fully occluded or unoccluded at some depth.

That’s very powerful and it’s a relatively small set of adjustments to the rasterization algorithm.

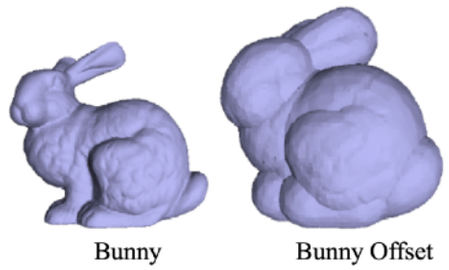
Ray tracing lets us make efficient arbitrary visibility queries.

Given that, wouldn’t it be nice to have a data structure that could sometimes answer arbitrary visibility queries and sometimes indicated it wasn’t sure?

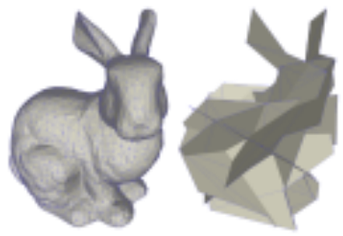
Say I’m in an office building and say I’m wondering if I can see a point in a room a few floors down—can I quickly determine that “no” and save the cost of tracing a ray to find out, and then save my rays for the trickier cases—can I see that point in that adjacent room?

There are a lot of interesting things to figure out along these lines, and not much work has been done.

Conservative Occluders



***Accurate Minkowski Sum Approximation of Polyhedral Models*, Varadhan and Manocha**



***Occluder Simplification using Planar Sections* Silvennoinen et al. 2014**

***Exploiting coherence of shadow rays*. Lukaszewski 2001**

A few papers to point at for inspiration on this front.

Amusingly, both of them targeted the Stanford bunny for their examples.

The Minowski sum idea is that if you compute an offset surface, a single ray can tell you if *any* ray to a set of directions (e.g. a light source) is occluded.

So, you can determine if a light is fully visible if a ray intersected against the Minowski-sum surface is unoccluded.

Related, the Silvennoinen paper is about computing simplified occluders: if a ray hits it, it's definitely occluded; if not, you need to consider the full geometry.

Another direction to consider here is voxel-based representations, but I'm more than over time, so we'll stop here.

Thanks!

Thanks everyone for your attention!

Next Up...

- Scaling Light Complexity in Games (Brian Karis)
- Scaling Light Complexity in Film Rendering (Patrick Kelly)
- From Theory to Fast: Scaling Game Path Tracing (Chris Wyman)