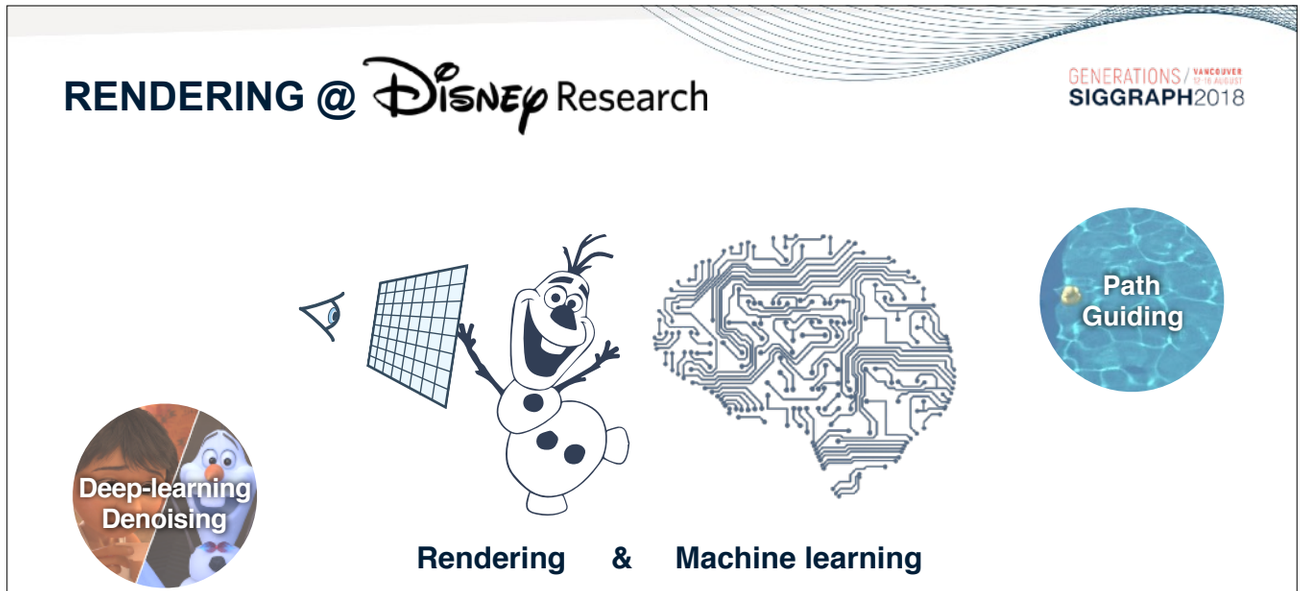
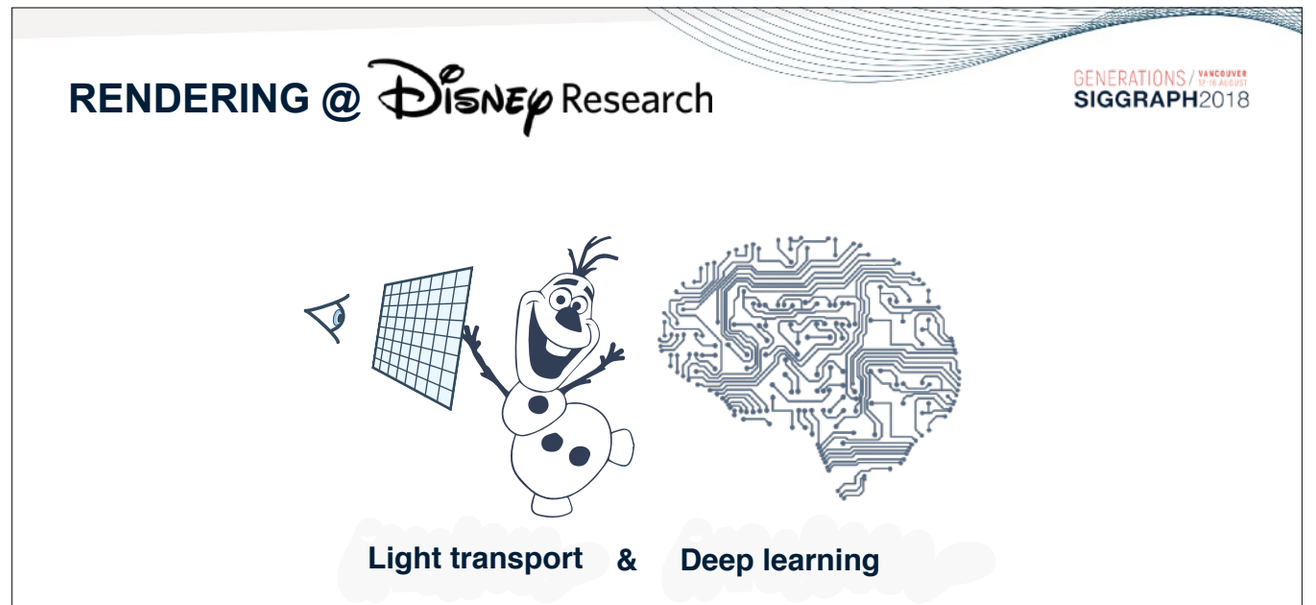

GENERATIONS / VANCOUVER 12-16 AUGUST
SIGGRAPH2018
DEEP LEARNING FOR LIGHT TRANSPORT SIMULATION
JAN NOVÁK, DISNEY RESEARCH



At Disney Research, we started looking at the cross-section of
rendering and machine learning about two years ago, asking ourselves if we can speedup certain computations in rendering using ML.

Yesterday, Thijs Vogels presented some of our endeavors into denoising with kernel-predicting CNNs [Vogels et al. 2018], last year at EGSR, we had a paper about data-driven path construction… path guiding.

While these do fall well within the course focus, this talk will be devoted to a slightly narrower topic: the cross-section of light-transport simulation with a subfield of machine learning, which is nowadays referred to as deep learning.



At Disney Research, we started looking at the cross-section of
rendering and machine learning about two years ago, asking ourselves if we can speedup certain computations in rendering using ML.

Yesterday, Thijs Vogels presented some of our endeavors into denoising with kernel-predicting CNNs [Vogels et al. 2018], last year at EGSR, we had a paper about data-driven path construction… path guiding.

While these do fall well within the course focus, this talk will be devoted to a slightly narrower topic: the cross-section of light-transport simulation with a subfield of machine learning, which is nowadays referred to as deep learning.

MOTIVATION

- Transport simulation is expensive
- Need to solve high-dimensional integration problems

Rendering equation

$$L(\mathbf{x},\mathbf{y}) = L_\mathrm{e}(\mathbf{x},\mathbf{y}) + \int f_\mathrm{r}(\mathbf{x},\mathbf{y},\mathbf{z}) G(\mathbf{x},\mathbf{y},\mathbf{z}) L(\mathbf{z},\mathbf{x}) \,\mathrm{d}\bar{\mathbf{z}}$$

Nasty recursion

Path-integral formulation

$$I = \int_{\mathcal{P}} L_\mathrm{e}(\mathbf{x}_0,\mathbf{x}_1) T(\bar{x}) W(\mathbf{x}_k,\mathbf{x}_{k-1}) \mathrm{d}\bar{x}$$

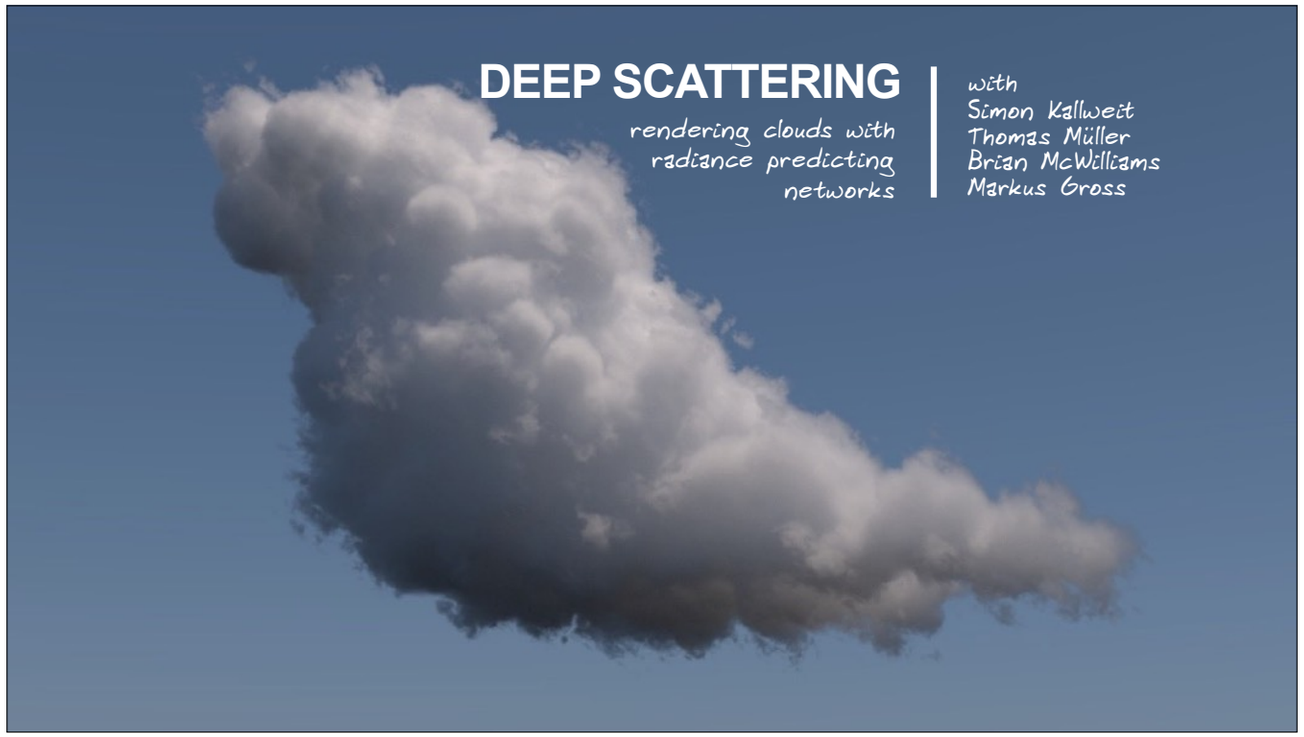
Nasty integration domain

The reason why we would like to apply deep learning to transport simulations are fairly obvious:
- simulation of light transport is computationally very expensive, and,
- in the general case, we are dealing with high-dimensional integration problems. It has been shown that the large modeling capacity of deep networks is quite useful in such situations.

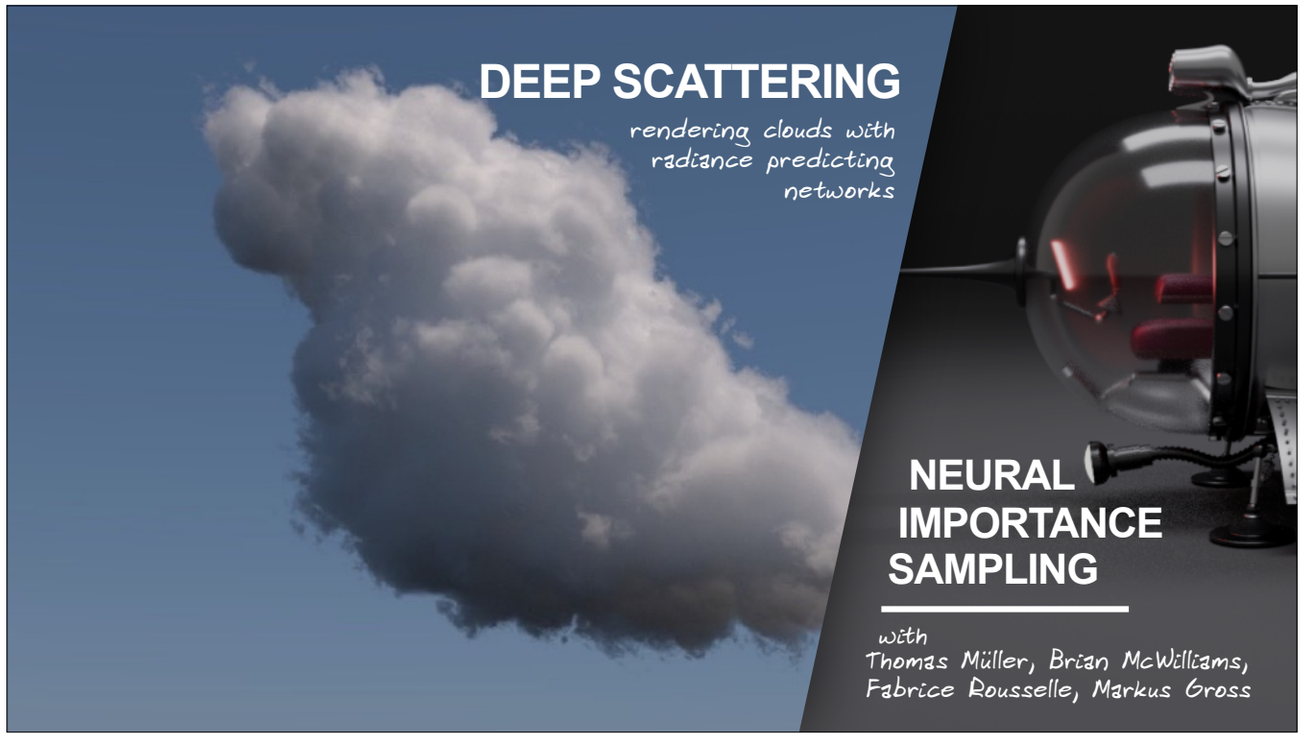
Independent of the formulation of light transport that we prefer to use, there is always something that challenges the computation. There is the nasty recursion in the rendering equation

If you prefer the path-integral formulation of light transport, the devil is in the innocent-looking P, which represents the high-dimensional path space that we integrate over.

Independent of the used formalism, estimating transport integrals, for instance by Monte Carlo integration, is challenging. I think it is worth considering every tool that we have available in the wider CS community to see if we could accelerate these computations somehow. Machine learning provides many such tools, and deep learning is one of them.



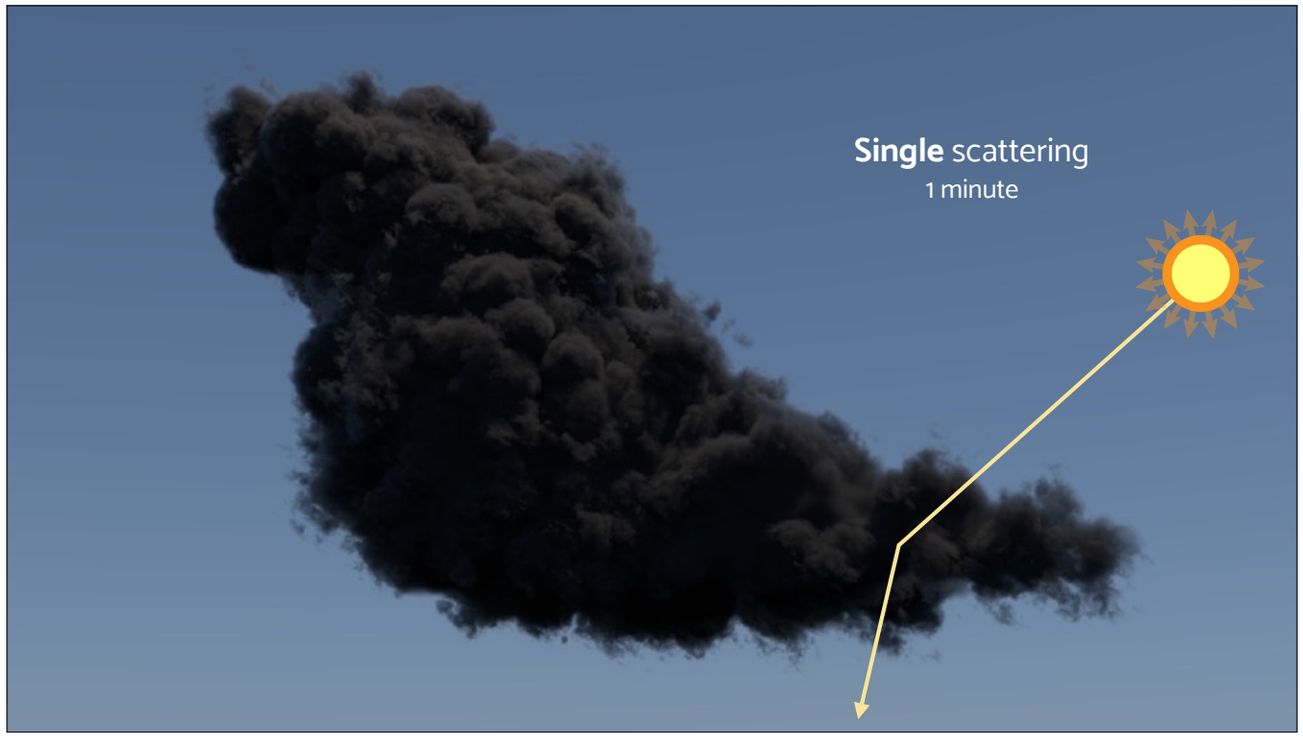
I will now focus on a specific kind of transport, namely multiple scattering in volumes, and describe our approach to approximating this transport using neural networks. Essentially, we will replace an expensive Monte Carlo estimator by a prediction obtained from a pre-trained neural network. That will be the first part of my talk.



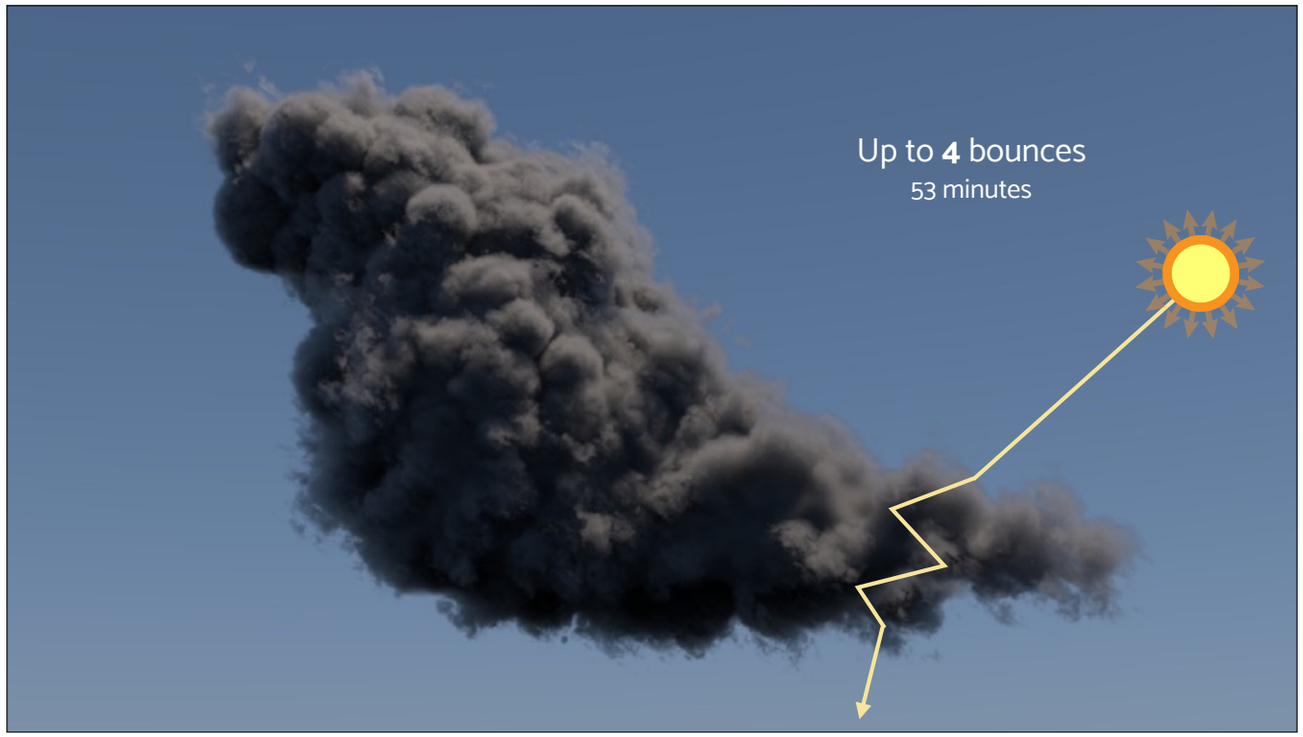
In the second part, we will briefly look at the challenges of using neural networks for importance sampling. So we will stay in the framework of Monte Carlo integration and we will use the networks to represent sampling densities.

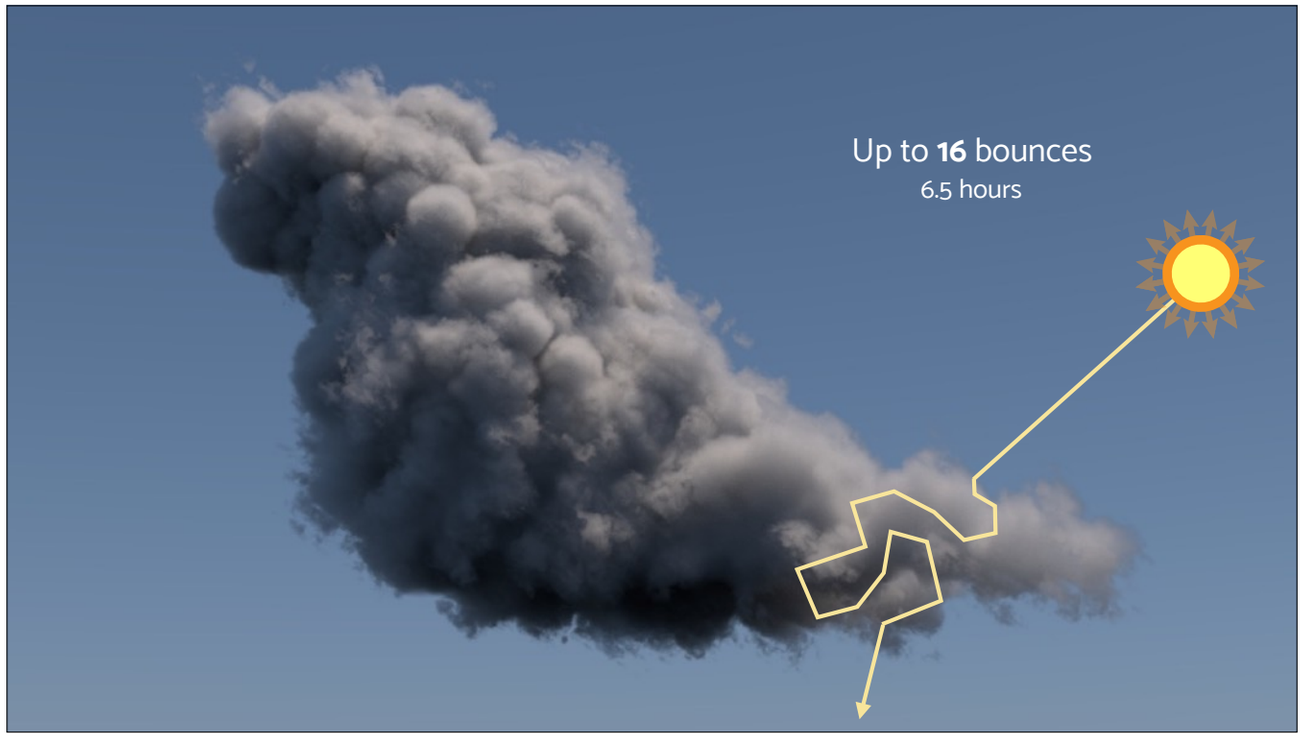
Our goal is to utilize neural networks to predict the amount of light anywhere inside an atmospheric cloud. You may wonder; “Why do we care about atmospheric clouds so much”. We had two very pragmatic reasons at Disney Research at one point:

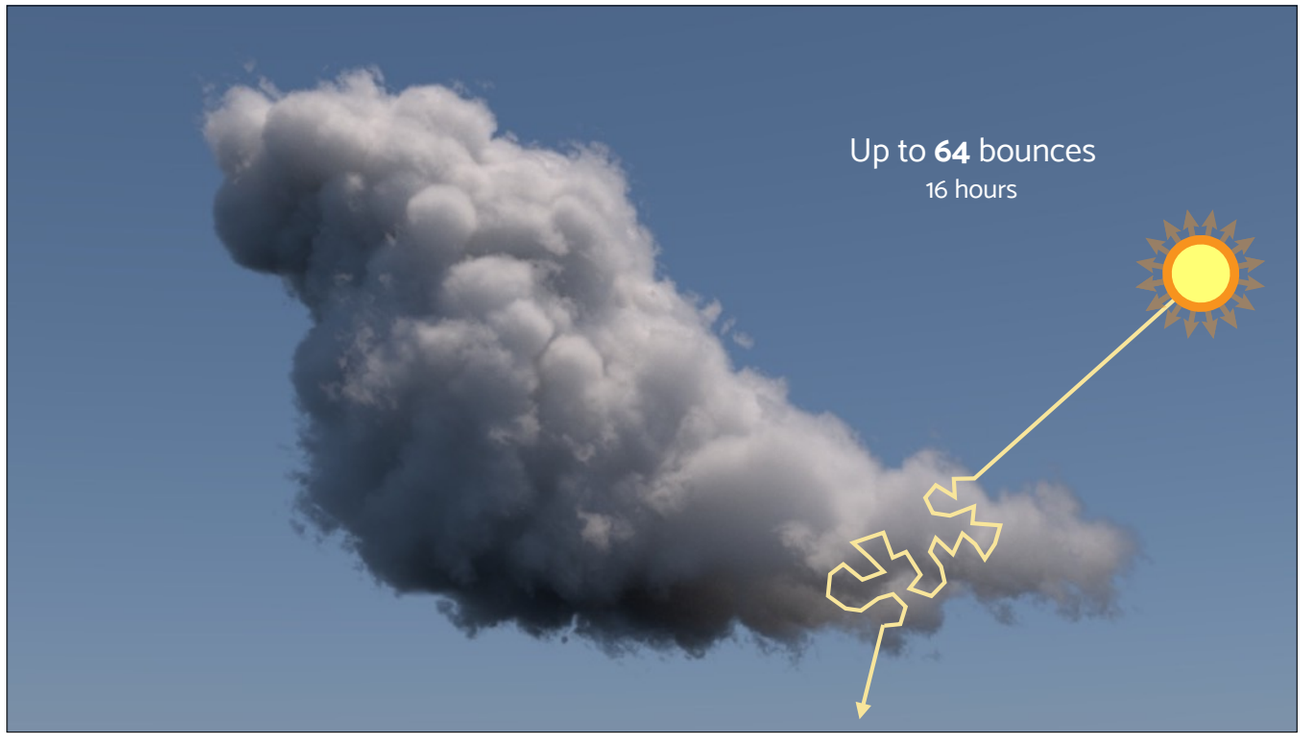
1) First, we wanted our artists to be able to sculpt and design clouds efficiently, ideally at interactive rates. This helps them be more productive in this specific case of modeling atmospheric clouds.
2) The second reason was even more pragmatic. We wanted to start experimenting with neural networks on a problem, where neural nets shall have easy time beating Monte Carlo estimation. Rendering clouds by synthesizing random walks is computationally extremely expensive. Furthermore, it shouldn’t be so hard for the network to learn their appearance. We really wanted to be in a good starting position here for our initial experiments with neural nets and their application to transport problems and leave the bigger challenges for later.



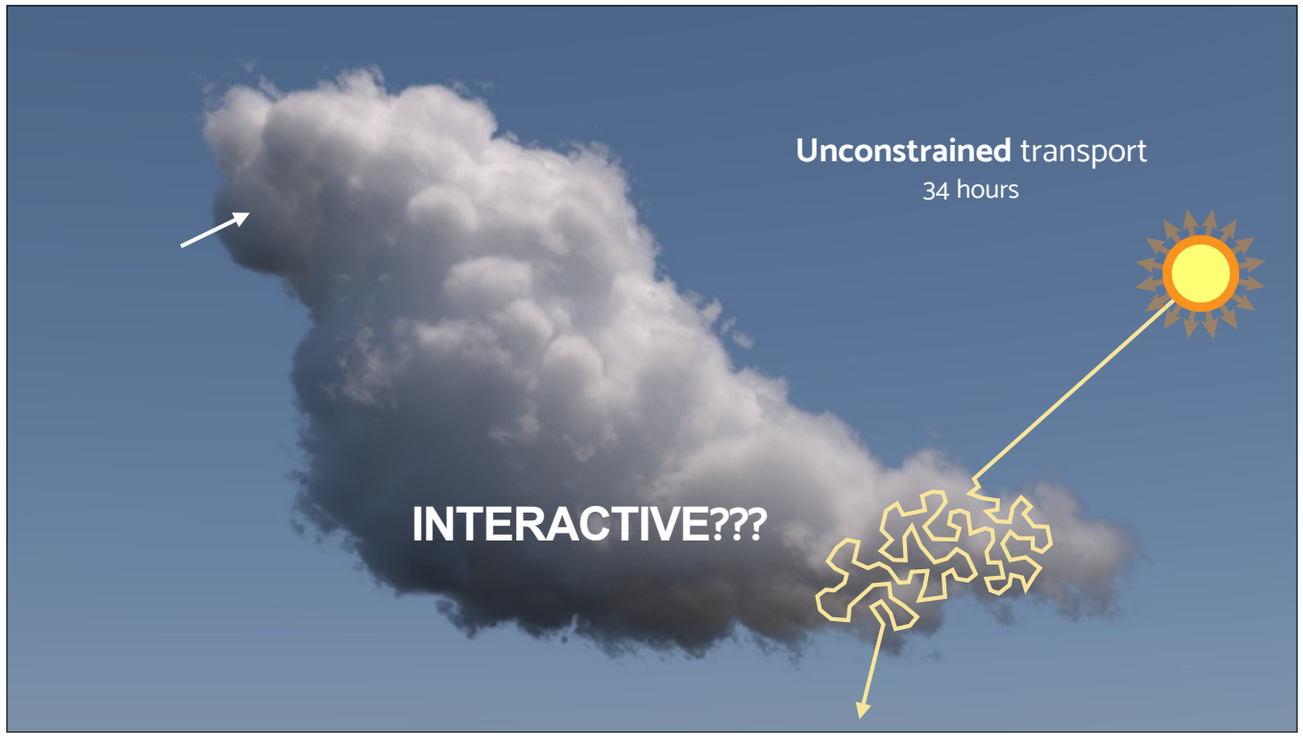
Here is a little illustration why transport in clouds is challenging…


Up to **4** bounces
53 minutes


Up to **16** bounces
6.5 hours



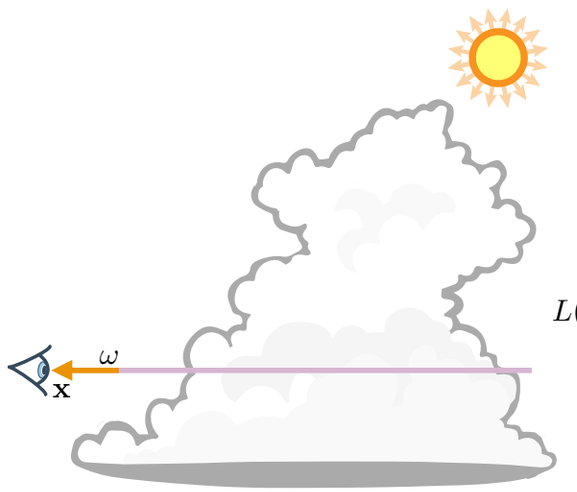
At 64 bounces, the image looks pretty good, but we are still missing a lot of energy.



In some of our experiments, the average number of bounces per single path was above 1000.
We currently cannot hope to achieve interactive render times with MC simulation.

And this could be concerning… especially in studios, where the artist may need to prepare many cloud assets for a movie or a computer game in different lighting conditions.

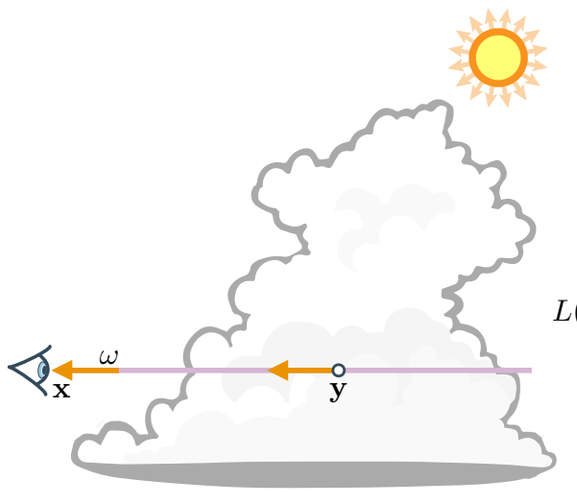
RENDERING CLOUDS



$$L(\mathbf{x},\omega) = \int_0^\infty p(\mathbf{y}|\mathbf{x}) L_s(\mathbf{y},\omega) \mathrm{d}y$$

Let’s now look at the problem from a slightly different perspective, through the equations that govern light transport.

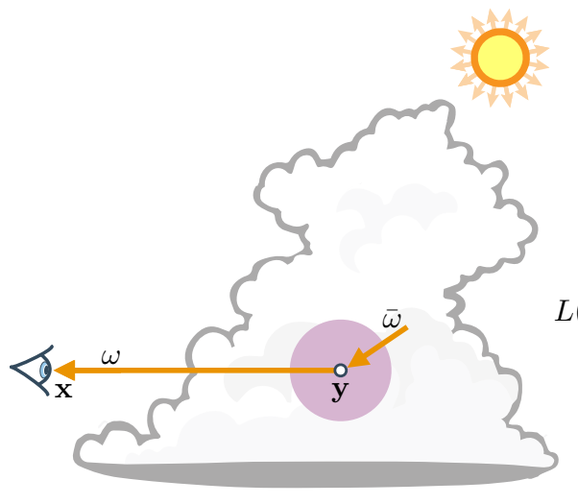
RENDERING CLOUDS



$$L(\mathbf{x},\omega) = \int_0^\infty p(\mathbf{y}|\mathbf{x}) L_s(\mathbf{y},\omega) \mathrm{d}y$$

At each point y, we need to compute the in-scattered radiance weighted by the probability density of a collision occurring at y, when traveling from point x. (The arrows here are pointing in the opposite direction: the direction of light travel).

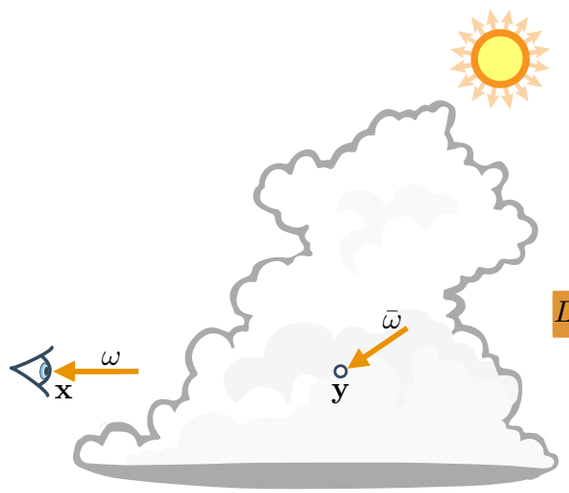
RENDERING CLOUDS



$$L(\mathbf{x},\omega) = \int_0^\infty p(\mathbf{y}|\mathbf{x})\,\alpha \int_S p(\bar{\omega}|\omega) L(\mathbf{y},\bar{\omega}) \mathrm{d}\bar{\omega}\,\mathrm{d}y$$

Computing the in-scattered radiance requires solving yet another integral. Here we see the main challenge formalized mathematically: the radiance function appears on both sides of the equation.

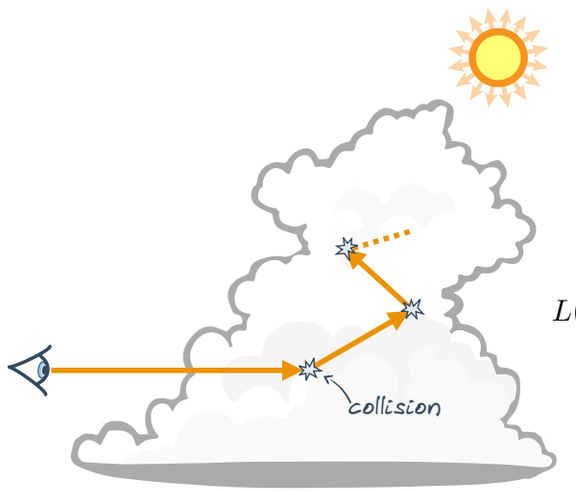
RENDERING CLOUDS



$$L(\mathbf{x},\omega) = \int_0^\infty p(\mathbf{y}|\mathbf{x})\,\alpha \int_S p(\bar{\omega}|\omega) L(\mathbf{y},\bar{\omega}) \mathrm{d}\bar{\omega}\,\mathrm{d}y$$

Computing the in-scattered radiance requires solving yet another integral. Here we see the main challenge formalized mathematically: the radiance function appears on both sides of the equation.

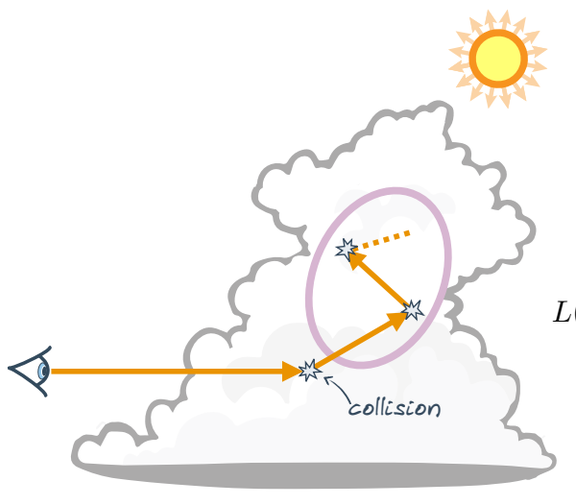
RENDERING CLOUDS



$$L(\mathbf{x},\omega) = \int_0^\infty p(\mathbf{y}|\mathbf{x})\,\alpha \int_S p(\bar{\omega}|\omega) L(\mathbf{y},\bar{\omega}) \mathrm{d}\bar{\omega}\,\mathrm{d}y$$

If we apply a one-sample Monte Carlo estimator, we end up alternating between distance and directional sampling constructing a path that can bounce hundreds of times thanks to the high albedo and high density of typical clouds.

RENDERING CLOUDS



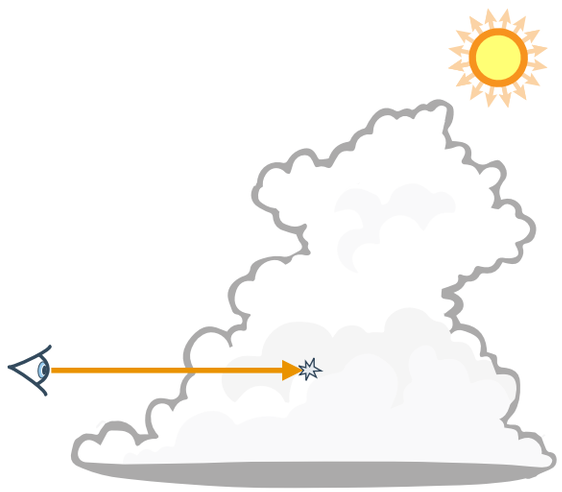
Hundreds of collisions per single path

$$L(\mathbf{x},\omega) = \int_0^\infty p(\mathbf{y}|\mathbf{x})\,\alpha \int_S p(\bar{\omega}|\omega) L(\mathbf{y},\bar{\omega}) \mathrm{d}\bar{\omega}\,\mathrm{d}y$$

Predict using a network

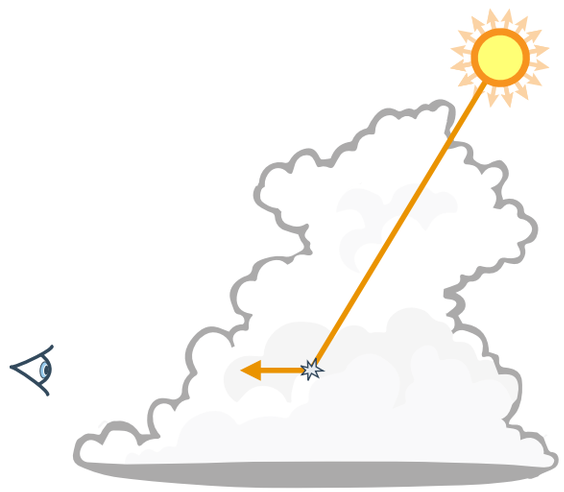
Our goal is to avoid creating these expensive path suffixes by predicting the expensive part of the simulation using a neural network.

RENDERING CLOUDS: OUR APPROACH



We split the transport simulation between MC estimation and neural network prediction.
We trace only the first collision…

RENDERING CLOUDS: OUR APPROACH



Scattered direct illumination
estimated using MC

…then we use Monte Carlo to estimate scattered direct illumination (single scattering). This is easy to do with MC and we want the single scattering to be accurate to capture effects such silverlining well.

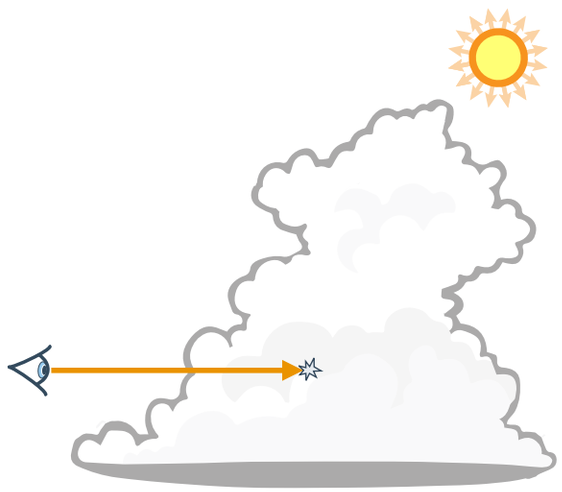
RENDERING CLOUDS: OUR APPROACH

Scattered direct illumination
estimated using MC

Scattered indirect illumination
predicted using NN

Multiple scattering, precisely the scattered indirect illumination, is predicted using a neural network.

RADIANCE PREDICTING MLP

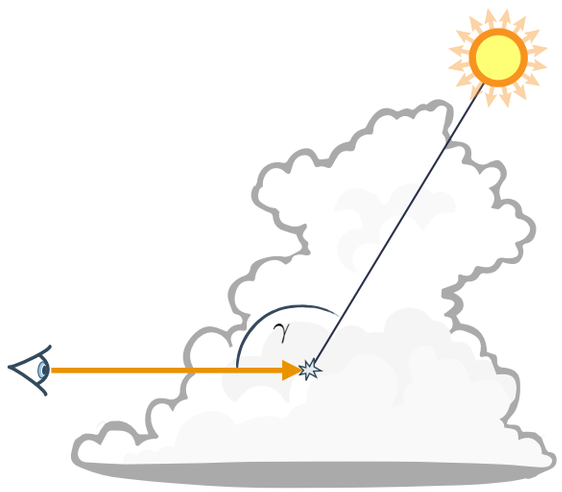


1) Parameterization of shading configuration

2) Feeding into the network

We use a standard, fully connected multi-layer perceptrons for the prediction.
The two main questions to resolve are:
1) How do we parameterize the shading configuration?
2) How do we feed this data into to the network?

RADIANCE PREDICTING MLP



1) Parameterization of shading configuration

2) Feeding into the network

Let’s look at the parameterization first.

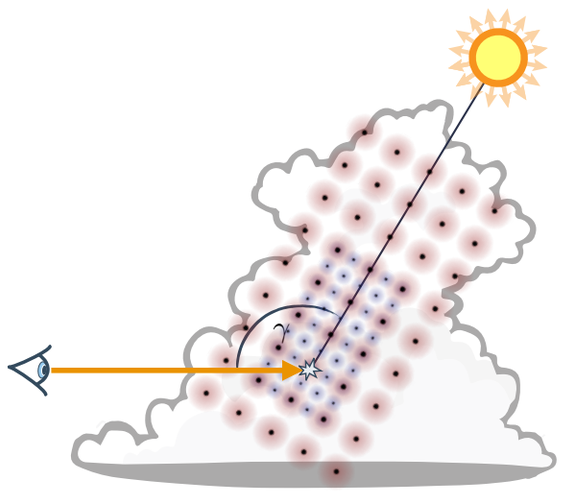
We somehow need to capture the geometric configuration between the eye ray and the sun. This is pretty simple, we can just compute this angle here.

The bigger challenge is efficiently approximating the entire cloud, or multiple clouds.
To that end, we simply lookup the density values of the cloud around the collision point. We use a rectangular stencil that has a few more point towards the light, as the cloud structure in that direction will impact the amount of light at the collision point the most.

In order to capture both local detail, but also remote parts of the cloud, we use an exponential hierarchy of these stencils. We use ten levels in total, where each level covers 8 times bigger volume than the previous level. We do this to make sure we cover even the remote parts of the cloud, and shadowing due to other clouds.

The stencils here are just for illustration, the smallest one has the shortest side 1 mean free path long.

RADIANCE PREDICTING MLP



1) Parameterization of shading configuration

2) Feeding into the network

Hierarchical rectangular stencil captures local details and overall shape

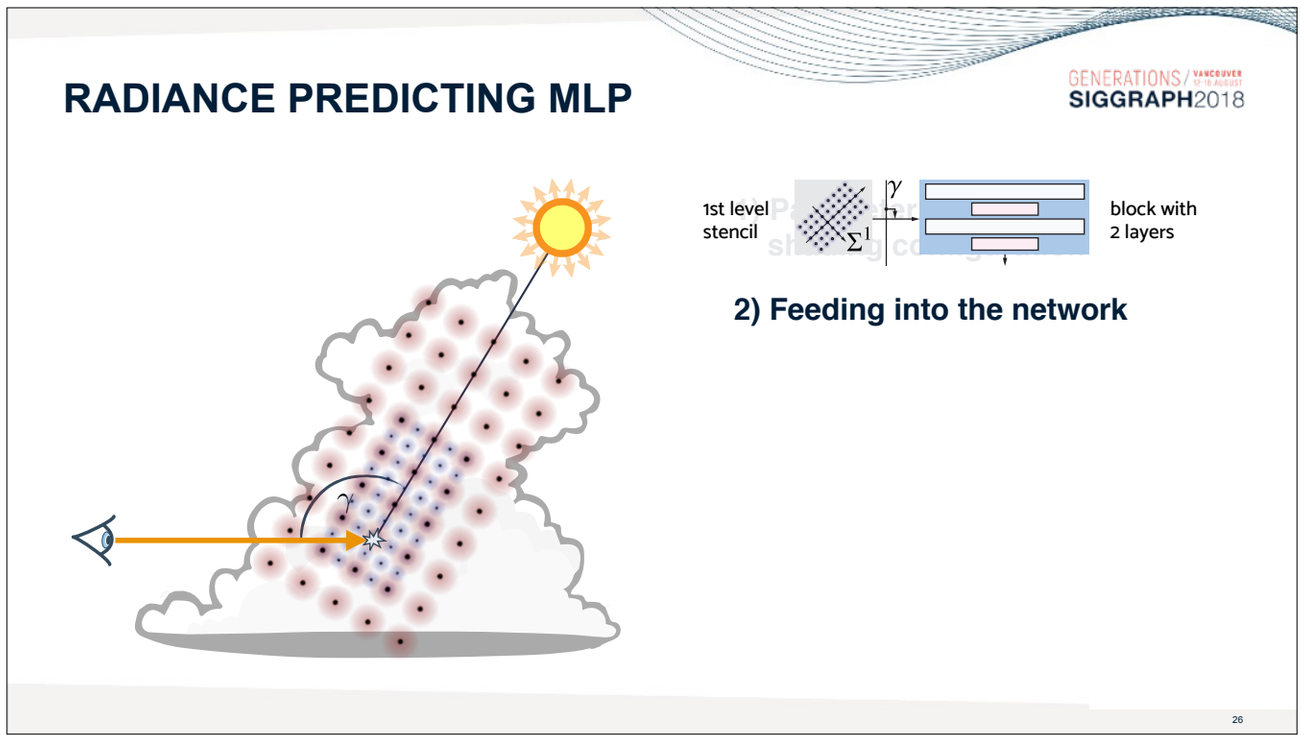
Let’s look at the parameterization first.

We somehow need to capture the geometric configuration between the eye ray and the sun. This is pretty simple, we can just compute this angle here.

The bigger challenge is efficiently approximating the entire cloud, or multiple clouds.
To that end, we simply lookup the density values of the cloud around the collision point. We use a rectangular stencil that has a few more point towards the light, as the cloud structure in that direction will impact the amount of light at the collision point the most.

In order to capture both local detail, but also remote parts of the cloud, we use an exponential hierarchy of these stencils. We use ten levels in total, where each level covers 8 times bigger volume than the previous level. We do this to make sure we cover even the remote parts of the cloud, and shadowing due to other clouds.

The stencils here are just for illustration, the smallest one has the shortest side 1 mean free path long.

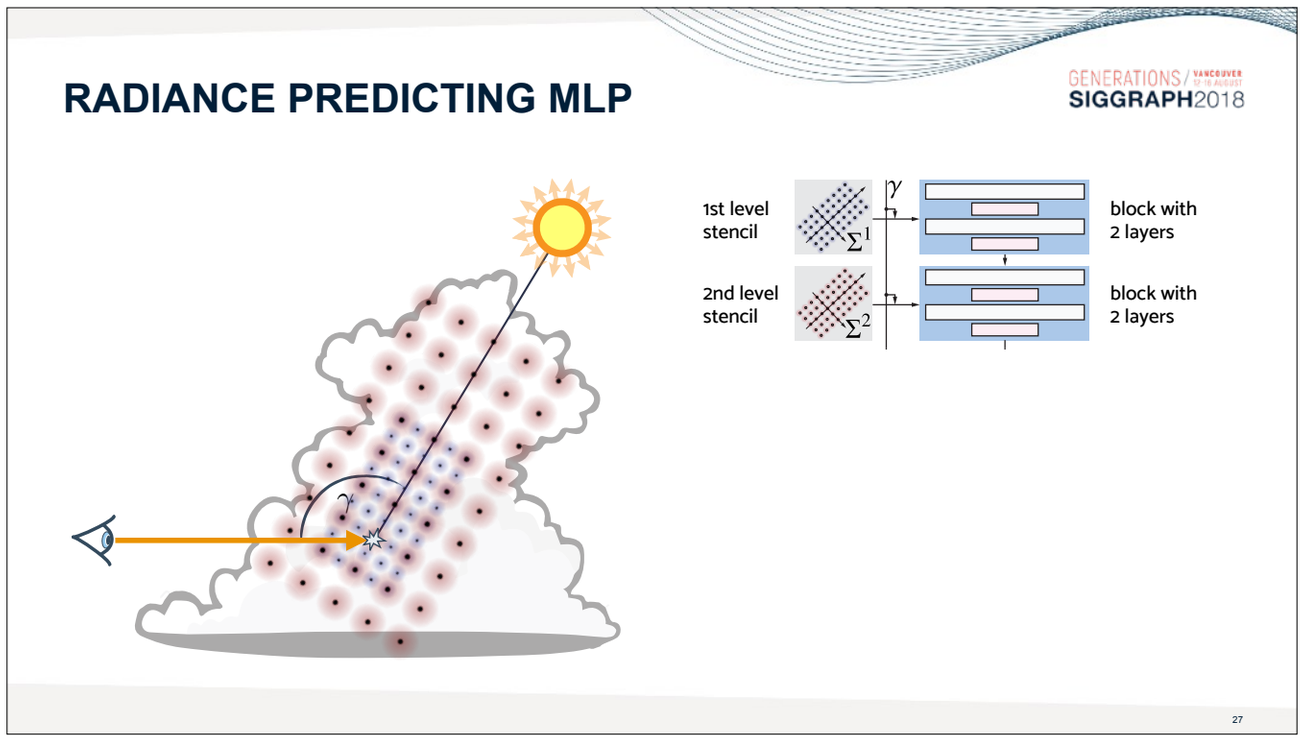


The second question is: “How do we input all these density values into the network?”

We first tried feeding all the stencils + the angle gamma, to the first layer, and it worked reasonably well. Nevertheless, we kept experimenting further and found that feeding the levels progressively, “level by layer”, significantly improves performance.

We take the smallest spatial scale, and feed it to a block of two layers. The output of this block is the routed to another block, which receives also the next level of the hierarchy, and this is repeated for all the 10 levels of the hierarchy.

This worked better than inputing all the density values to the first layer, which suggests that the network has difficulties extracting relations within and across different spatial scales and it benefits from being told about the scales explicitly.

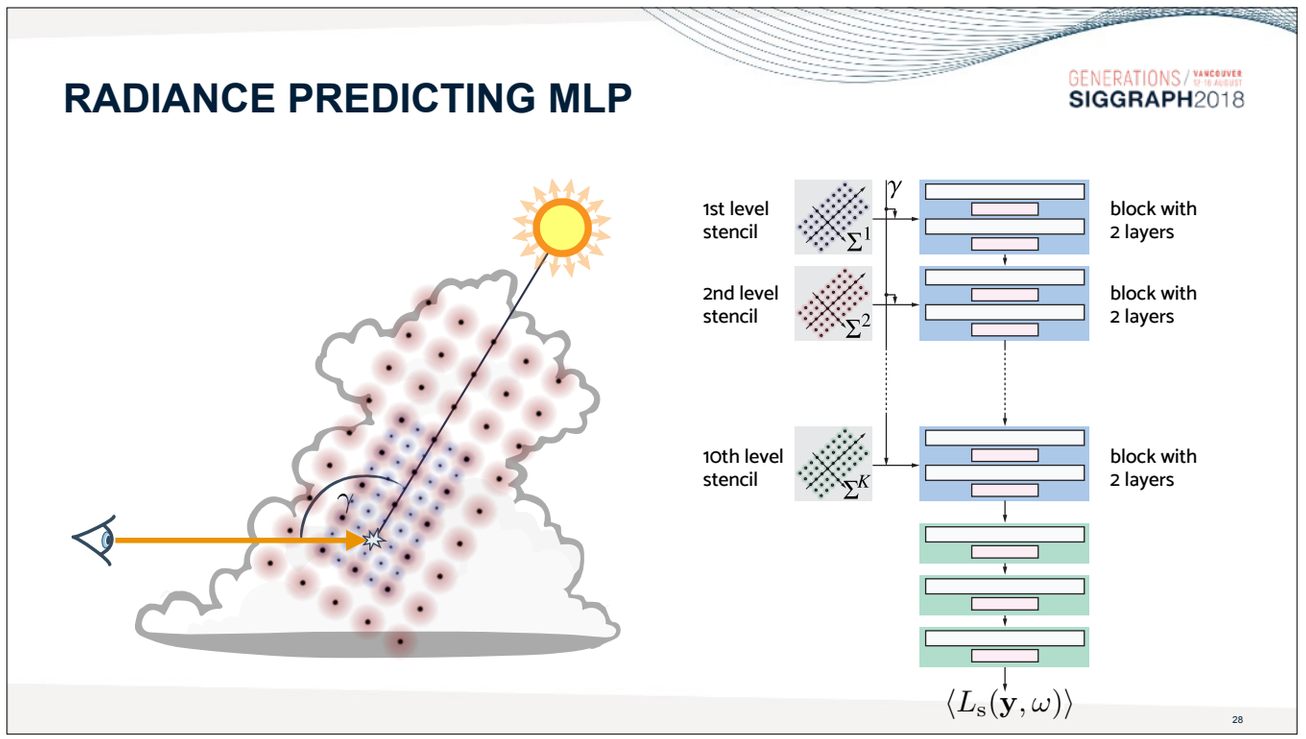


The second question is: “How do we input all these density values into the network?”

We first tried feeding all the stencils + the angle gamma, to the first layer, and it worked reasonably well. Nevertheless, we kept experimenting further and found that feeding the levels progressively, “level by layer”, significantly improves performance.

We take the smallest spatial scale, and feed it to a block of two layers. The output of this block is the routed to another block, which receives also the next level of the hierarchy, and this is repeated for all the 10 levels of the hierarchy.

This worked better than inputing all the density values to the first layer, which suggests that the network has difficulties extracting relations within and across different spatial scales and it benefits from being told about the scales explicitly.



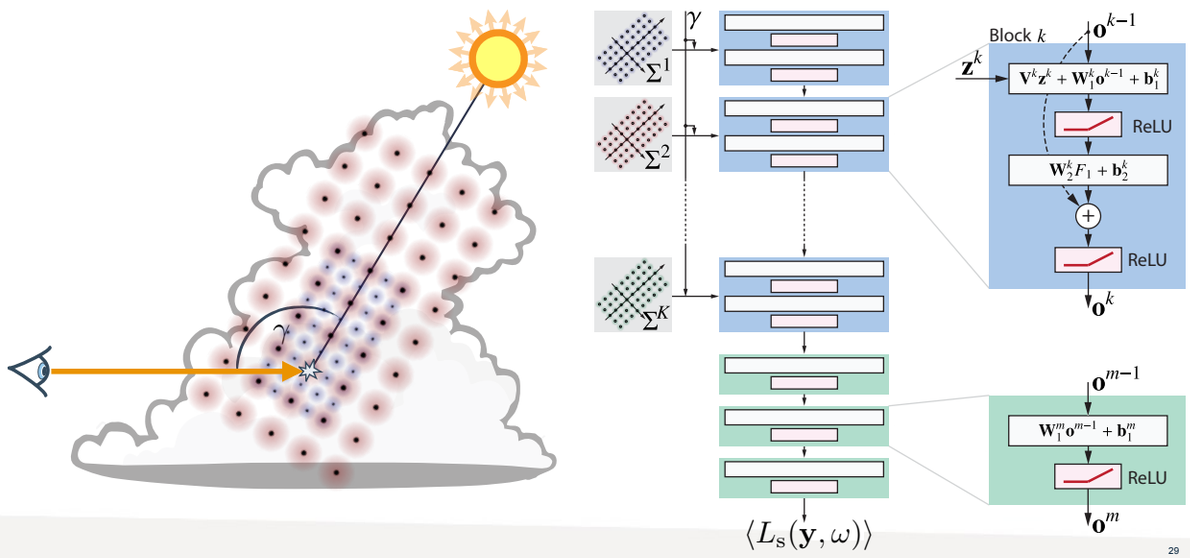
The second question is: “How do we input all these density values into the network?”

We first tried feeding all the stencils + the angle gamma, to the first layer, and it worked reasonably well. Nevertheless, we kept experimenting further and found that feeding the levels progressively, “level by layer”, significantly improves performance.

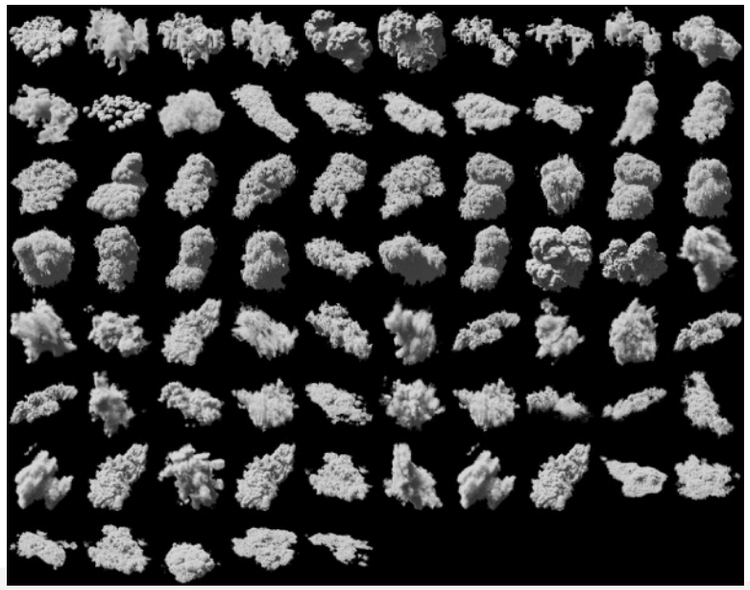
We take the smallest spatial scale, and feed it to a block of two layers. The output of this block is the routed to another block, which receives also the next level of the hierarchy, and this is repeated for all the 10 levels of the hierarchy.

This worked better than inputing all the density values to the first layer, which suggests that the network has difficulties extracting relations within and across different spatial scales and it benefits from being told about the scales explicitly.

RADIANCE PREDICTING MLP

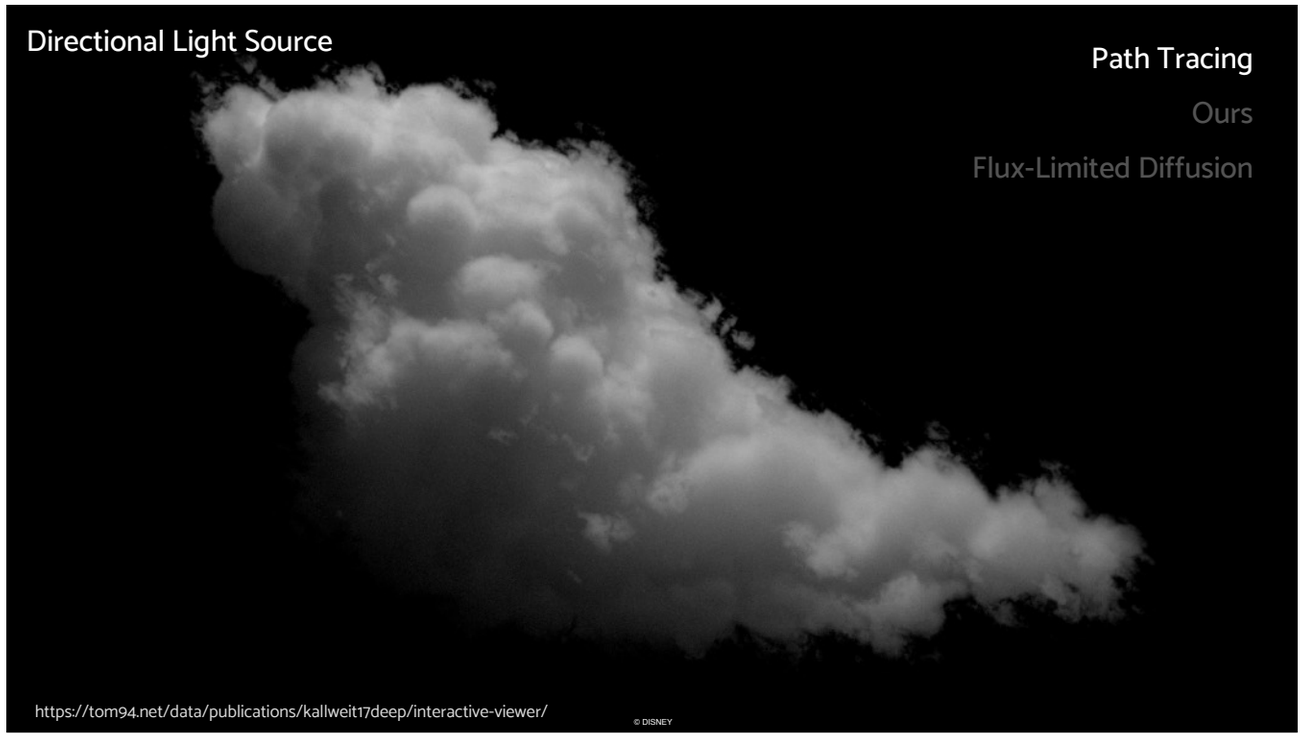


TRAINING DATA

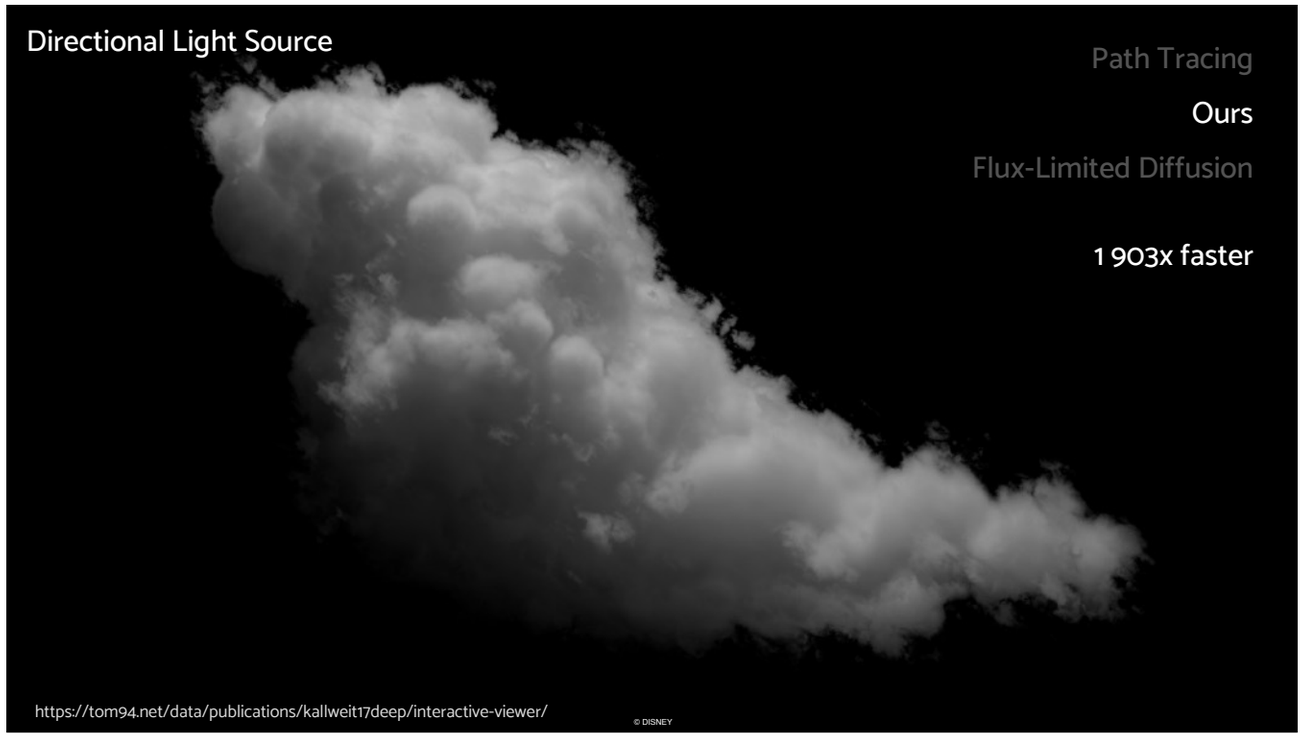


We trained the network using 80 clouds ([100-1200]^3 voxels) producing about 15 million data records—each represented by a high-quality MC estimate of multi-scattered radiance somewhere in the cloud.

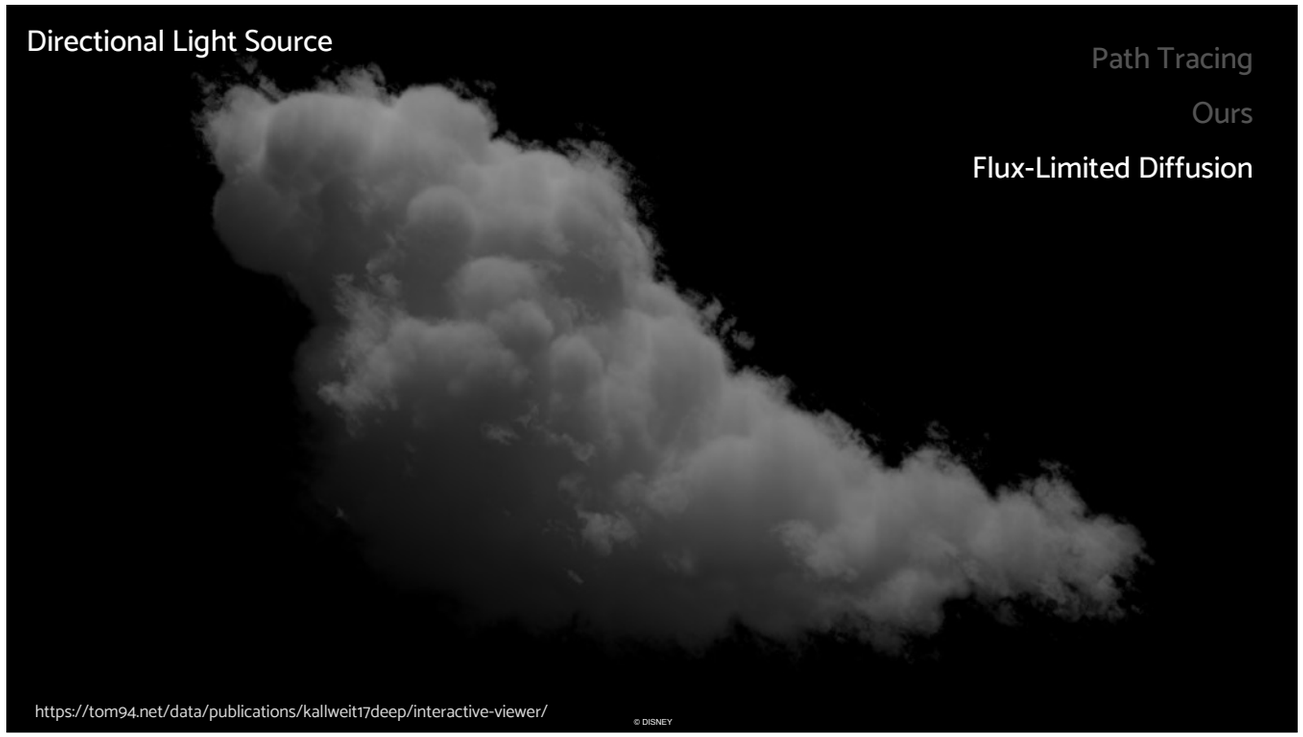
Each record is constructed using a volume with the following parameters randomized:
- cloud
- density scaling factor
- view position/direction
- light direction
- shading position (obtained by sampling the free path from outside “into” the cloud)

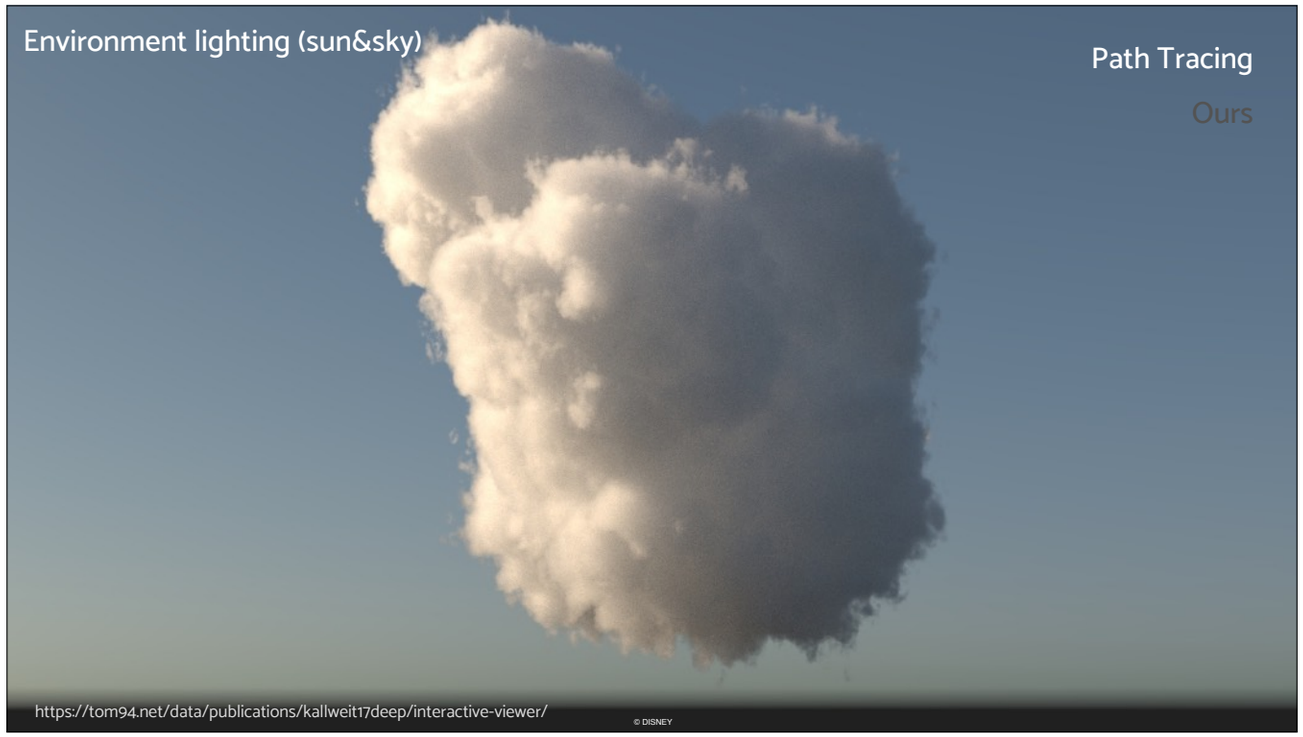


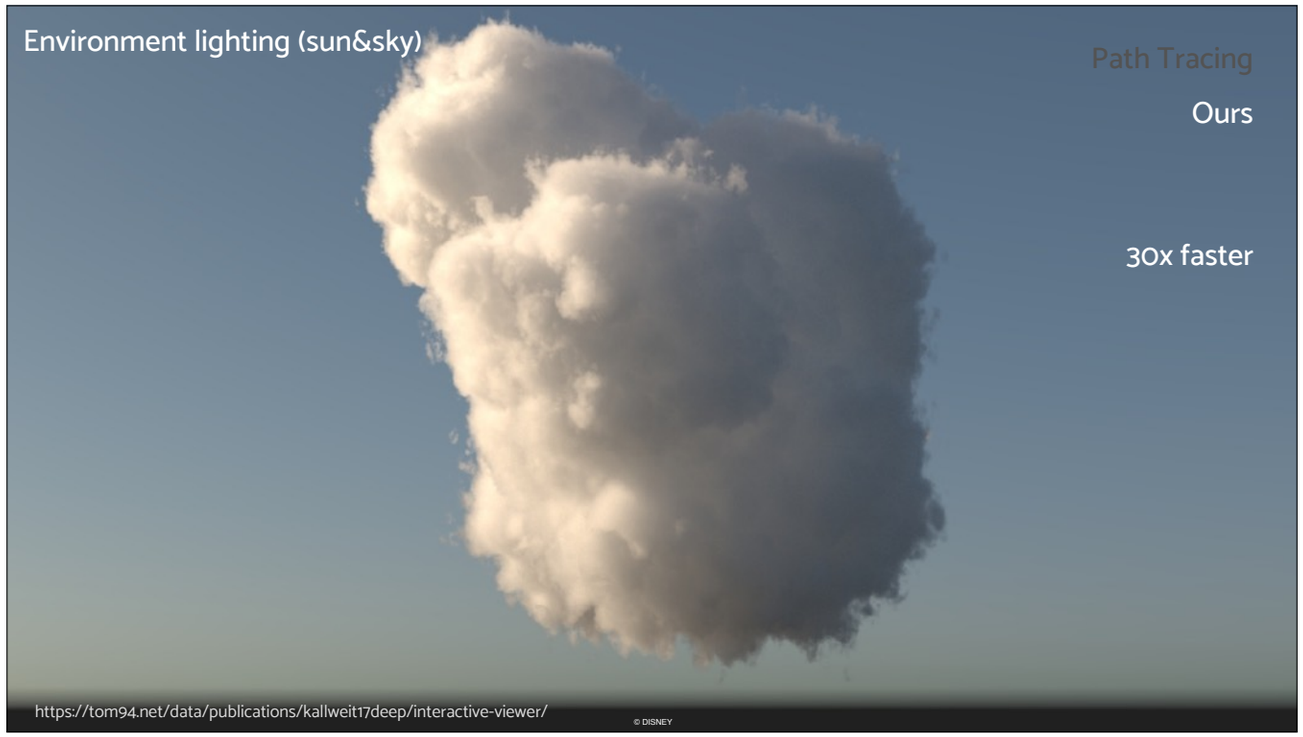
Here are the results…



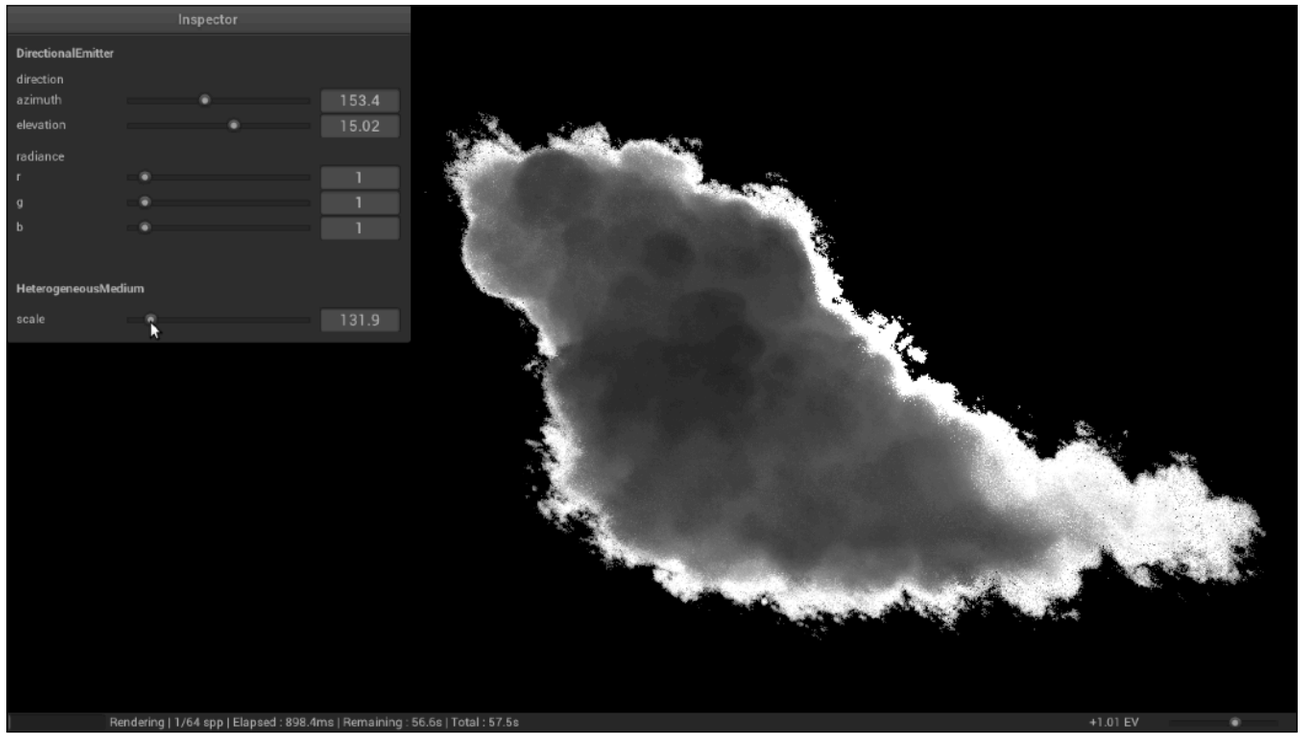
Using our method and a single GPU, the cloud can be rendered almost 2000 times faster than using a 48 thread CPU to get equal variance.


Directional Light Source
Path Tracing
Ours
Flux-Limited Diffusion
https://tom94.net/data/publications/kallweit17deep/interactive-viewer/
© DISNEY


Environment lighting (sun&sky)
Path Tracing
Ours
https://tom94.net/data/publications/kallweit17deep/interactive-viewer/
© DISNEY


Environment lighting (sun&sky)
Path Tracing
Ours
30x faster
https://tom94.net/data/publications/kallweit17deep/interactive-viewer/
© DISNEY


cloud model: https://disneyanimation.com/technology/datasets


Inspector
DirectionalEmitter
direction
azimuth
153.4
elevation
15.02
radiance
r
1
g
1
b
1
HeterogeneousMedium
scale
131.9
Rendering | 1/64 spp | Elapsed : 898.4ms | Remaining : 56.6s | Total : 57.5s
+1.01 EV

SIGGRAPH Asia 2017

Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks

SIMON KALLWEIT, Disney Research and ETH Zürich
THOMAS MÜLLER, Disney Research and ETH Zürich
BRIAN MCWILLIAMS, Disney Research
MARKUS GROSS, Disney Research and ETH Zürich
JAN NOVÁK, Disney Research

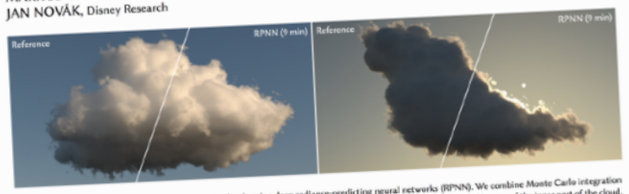


Fig. 1. We synthesize multi-scattered illumination in clouds using deep radiance-predicting neural networks (RPNN). We combine Monte Carlo integration with data-driven radiance predictions, accurately reproducing edge-darkening effects (left), silverlining (right), and the whiteness of the inner part of the cloud.

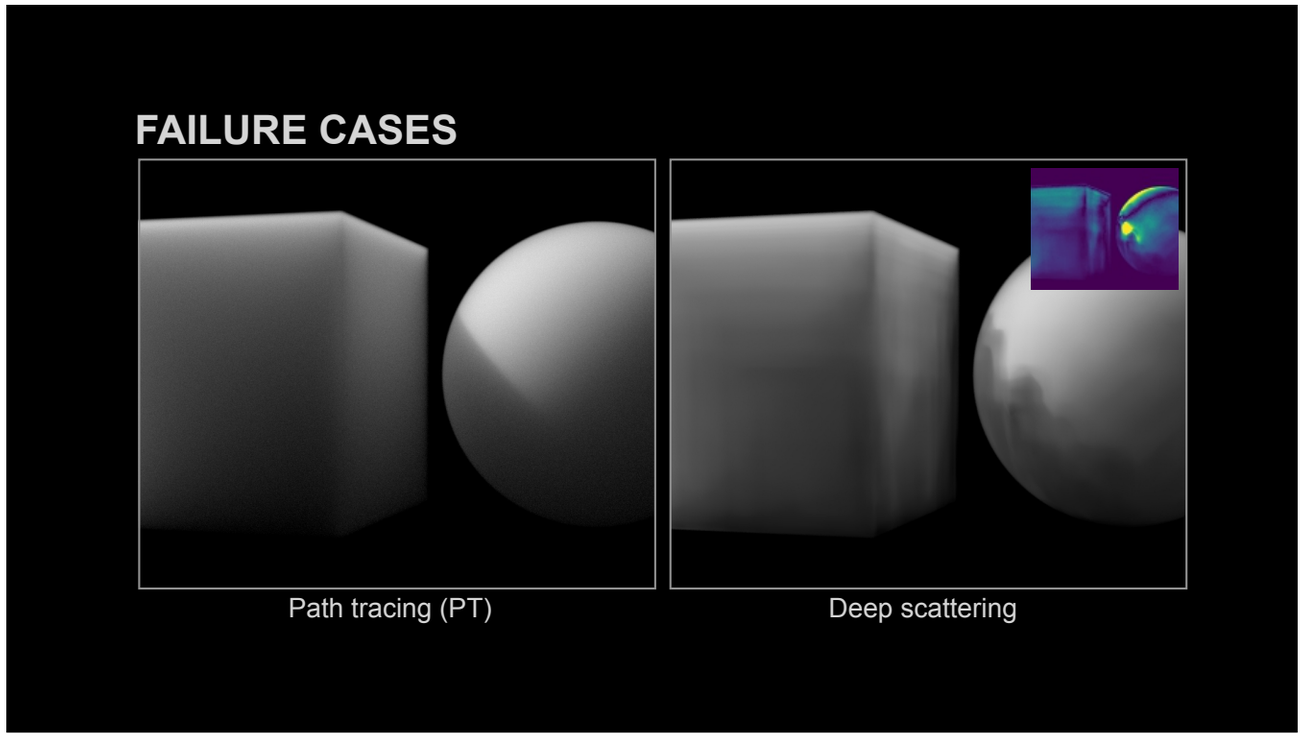
We present a technique for efficiently synthesizing images of atmospheric clouds using a combination of Monte Carlo integration and neural networks. The intricacies of Lorenz-Mie scattering and the high albedo of cloud-forming aerosols make rendering of clouds—e.g. the characteristic silverlining and the "whiteness" of the inner body—challenging for methods based solely on Monte Carlo integration or diffusion theory. We approach the problem differently. Instead of simulating all light transport during rendering, we pre-learn the spatial and directional distribution of radiant flux from tens of cloud exemplars. To render a new scene, we sample visible points of the cloud and, for each, extract a hierarchical 3D descriptor of the cloud geometry with respect to the shading location and the light source. The descriptor is input to a deep neural network that predicts the radiance function for each shading configuration. We make the key observation that

Additional Key Words and Phrases: Monte Carlo Rendering, Deep Learning, Machine Learning, Participating Media, Phase Functions, Clouds

ACM Reference format:
Simon Kallweit, Thomas Müller, Brian McWilliams, Markus Gross, and Jan Novák. 2017. Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks. *ACM Trans. Graph.* 36, 6, Article 231 (November 2017), 11 pages.
https://doi.org/10.1145/3130800.3130880

1 INTRODUCTION

Efficient and accurate rendering of high-albedo materials is a challenging problem, especially when the optical properties vary spatially. Heterogeneous densities of aerosols formed into visible struc-

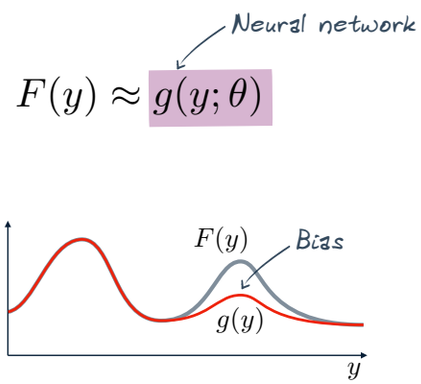


As any other method, our approach has some drawbacks. Our network is quite good at predicting radiance in clouds that it has not seen before. But if the volume looks very different, for instance the cube and the sphere here that do not have a fuzzy border, then the results can be fairly poor.

This shouldn’t be very surprising, as we have not trained the network on such volumes. All it has seen during training were fluffy cumulus and stratus clouds. We also did not parameterize the shading configuration to capture hard shadows well.

So the performance could be improved, for instance by choosing a more suitable parameterization, but it is still concerning that the approach does not generalize very well.

WHAT ARE ACCEPTABLE FAILURES/ERRORS?



I want to take a moment here to discuss the setting that we have used here and whether we could frame the problem slightly differently, so that when the network fails, it fails more gracefully.

Let's talk about the kinds of approximations we are introducing and what kinds of errors they manifest as. Say we have a function of two variables, one of which—the variable x—we are integrating out. In the deep scattering project, we decided to approximate the integral by a parametric function g—we used a neural network to that end.

As long as the approximation is accurate, the results were good.
In some situations, however, the predicted value deviated from the ground truth and the error showed up as darkening or brightening.

WHAT ARE ACCEPTABLE FAILURES/ERRORS?

Neural network

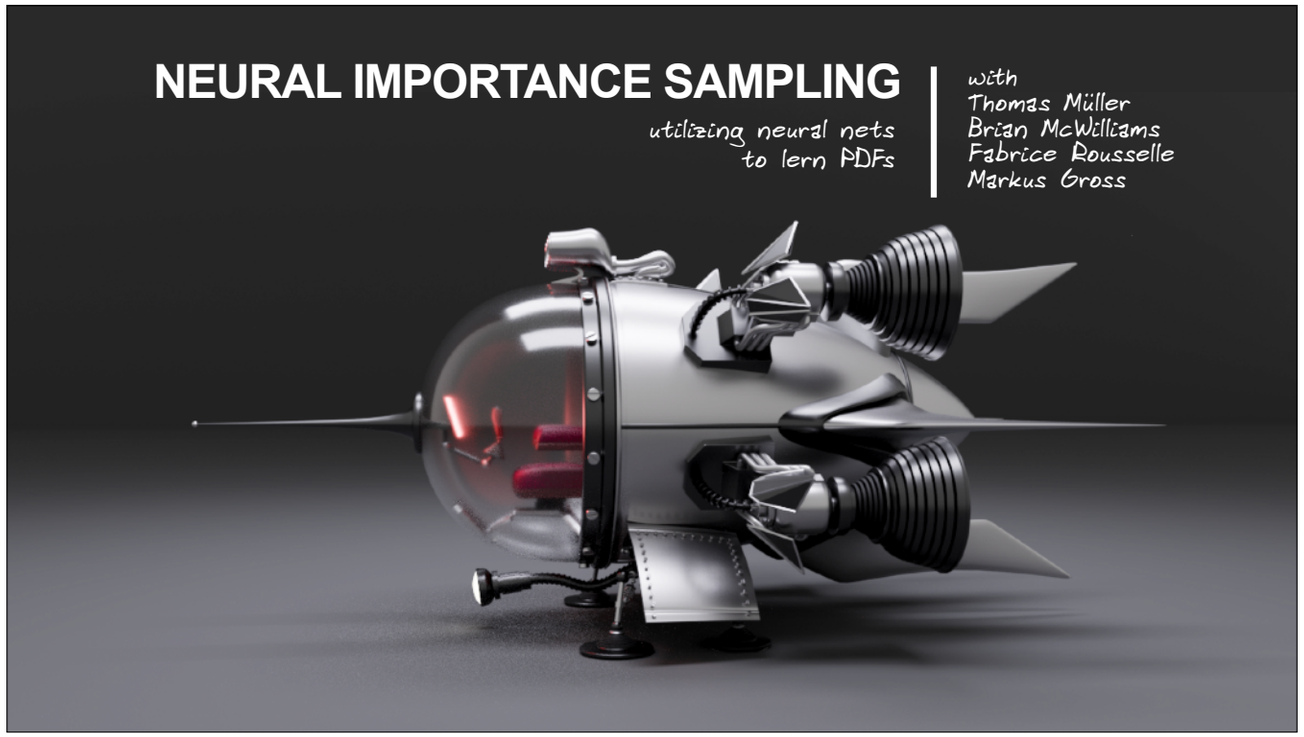
$$F(y) \approx g(y;\theta)$$

$$\langle F(y) \rangle = \frac{1}{N} \sum \frac{f(x_i, y)}{p(x_i;\theta)}$$

Neural network

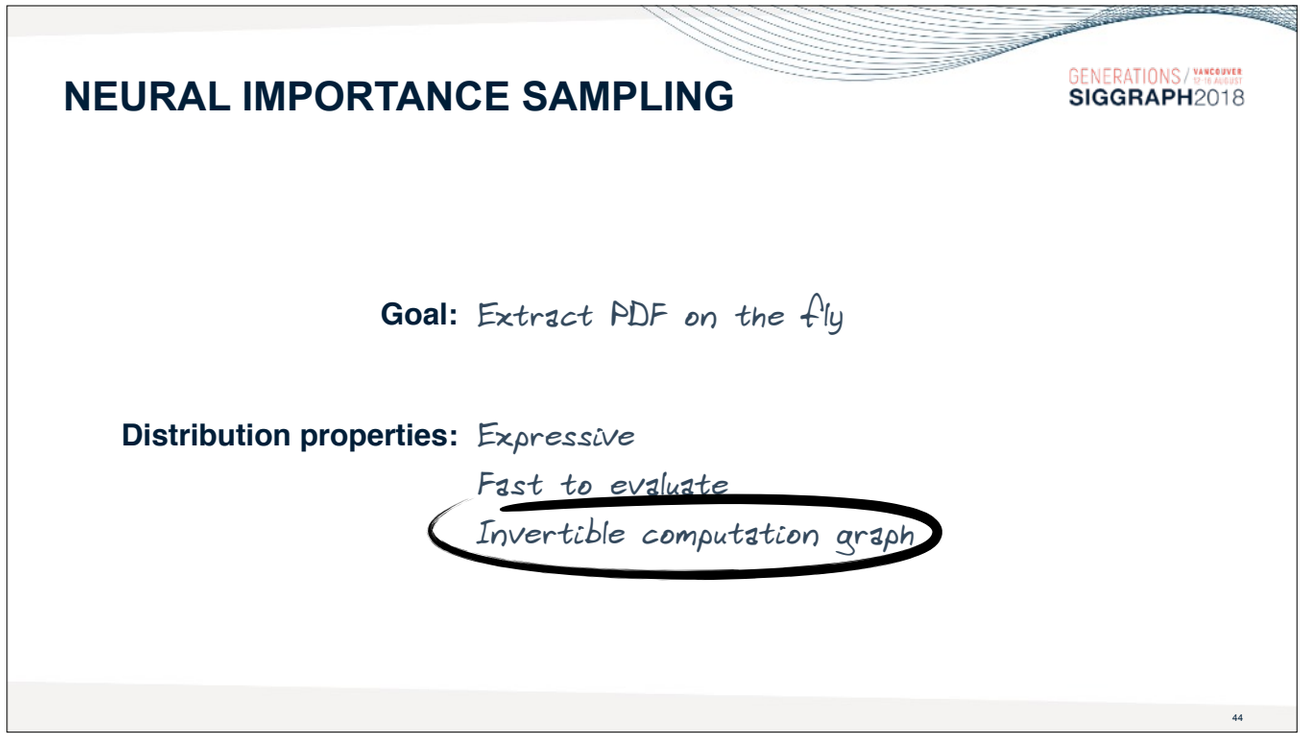
Now, I want to talk about a different approach to approximating F with the help of neural nets. In this approach, we leverage standard Monte Carlo estimation, and we use the network to only represent the sampling distribution. It will be responsible for merely generating samples, not any final estimates.

In this setting, a badly performing network will lead to noise rather than bias. This may be the preferred form of error in some applications. Furthermore, if the network performs well, the Monte Carlo estimator will have very low variance, so we are not really loosing any quality with this second setting. Now I said that the network will be used “only” for generating samples, but it is actually a much harder problem, which we started investigating almost two years ago…



…together with Thomas Müller and a few other folks at Disney Research.

This is still work in progress. We recently released a tech report to arxiv to share our early findings with the community. I will not go too much into the details here; I will actually not discuss the novel contributions that we propose—please see the paper if you are interested. I want to use the time here to rather introduce the problem that we are facing when using neural nets for importance sampling.



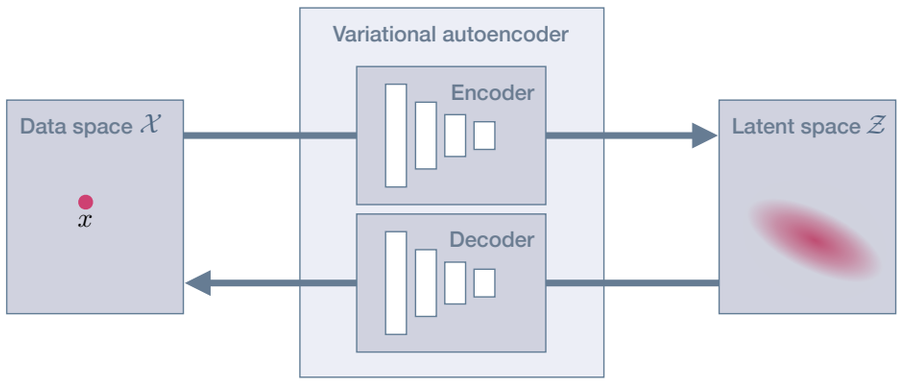
Our target application will be path guiding, much in the spirit of what Jaroslav talked about, but the problem statement and our execution are general enough to be applied to other integration problems as well.

Our goal is to train a network during rendering so that it can be used for generating well-placed samples. In order for the application to be successful, the sampling distribution needs to have certain properties:

- It should be expressive so that it can capture multimodal distributions; this should be doable given the high modeling capacity of neural networks.
- It should be fast to evaluate.
- But most importantly, the computation graph representing the distribution must be invertible, so that we can not only draw samples, but also exactly evaluate their probability density. This is mandatory to preserve unbiased estimation.

I will focus on the invertibility, which is the most interesting requirement here.

GENERATIVE PROBABILISTIC MODELING



$$p(x) = \int_{\mathcal{Z}} p(x|z;\theta)\, p(z)\, \mathrm{d}z$$

Exact evaluation is computationally intractable

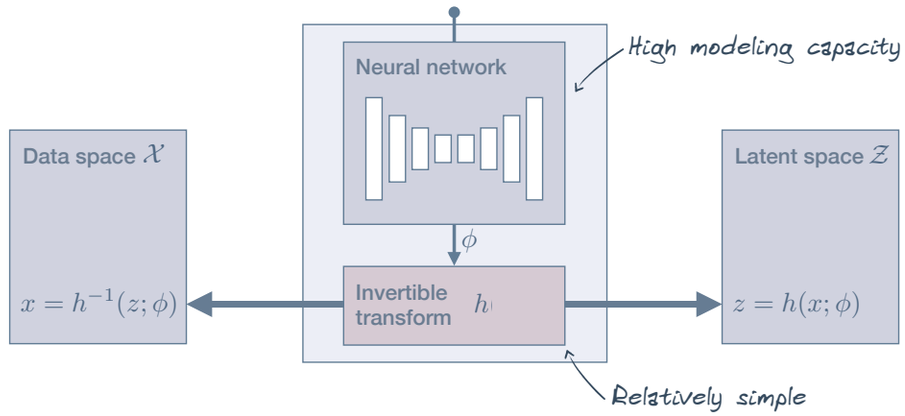
The setting here is the typical configuration of generative modeling: we have a number of observations in the data space whose distribution we want to model. Unfortunately it is difficult to do so directly in the data space. We therefore utilize a different space, the so-called latent space, which we define in such a way that allows easily drawing samples. The latent variables are typically distributed using standard distributions, e.g. uniform or normal distribution. The key ingredient in these approaches is the mapping between the data space and the latent space, which is learned from data, for instance variational autoencoders utilize two networks, encoder and decoder. But it could be any kind of networks.

The thing that is actually important here is how these networks relate points in the two spaces to each other. It is important because certain kinds of relations may disqualify the approach from being applicable to Monte Carlo integration.

For instance, in the case of a standard variational autoencoder, a sample x can be generated by
sampling the latent variables and passing the sample through the decoder to obtain x. But with most probabilistic models, the point x can actually be obtained using many configurations of latent variables. Therefore, to quantify the probability density of any x, we need to integrate over the latent space. This is computationally intractable in practice.

So not every generative approach can be used for importance sampling.

INVERTIBLE PARAMETRIC MAPPING [Dinh et al. 2014, 2016]



$$p(x) = p(z)\,|\det(\mathbf{J}_h)|$$

There is one particular probabilistic model, however, that suits our needs well. It was proposed by Laurent Dinh and colleagues, and the main idea is to relate the data and latent space using an invertible, differentiable mapping.

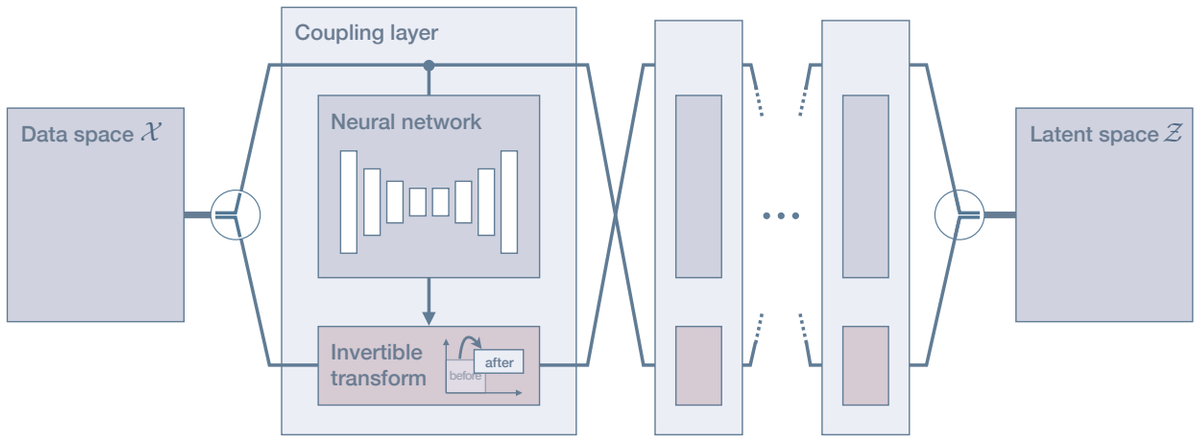
Dinh and colleagues propose to utilize a rather simple bijective transform, which is parameterized by the output of a neural network.

The high-capacity network can learn complex, non-linear relations, but the actual transformation “h” is kept rather simple, so that it can be easily inverted and differentiated.

Since the two spaces are related using bijection, there is no integration needed to obtain the probability density of x: it is simply given by the change-of-variables formula;
we take the density of the latent variable z and multiply it by the Jacobian determinant of the transform “h”.

COUPLING LAYERS

[Dinh et al. 2014, 2016]



The transform proposed by the authors is a simple translation and scaling, which is clearly not enough to capture the complex relations between the two spaces.
The authors thus propose to chain a number of them, referring to a single block of a neural network and the transform as the “coupling layer”.

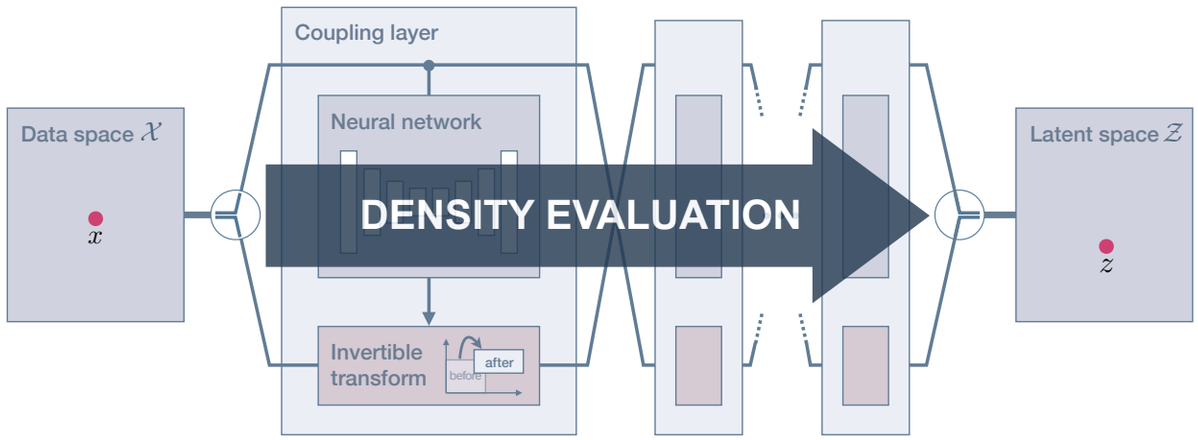
The chain of coupling layers still represents a bijective transform and the computation of the jacobian determinant is relatively cheap.

Once the networks are trained, you can take an existing data point, and evaluate the chain
from left to right to evaluate the probability density of x. In order to generate a new sample,
we pick a point in the latent space, and pass it from right to left to get the corresponding data-space sample.

COUPLING LAYERS

[Dinh et al. 2014, 2016]

GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018



The transform proposed by the authors is a simple translation and scaling, which is clearly not enough to capture the complex relations between the two spaces.
The authors thus propose to chain a number of them, referring to a single block of a neural network and the transform as the “coupling layer”.

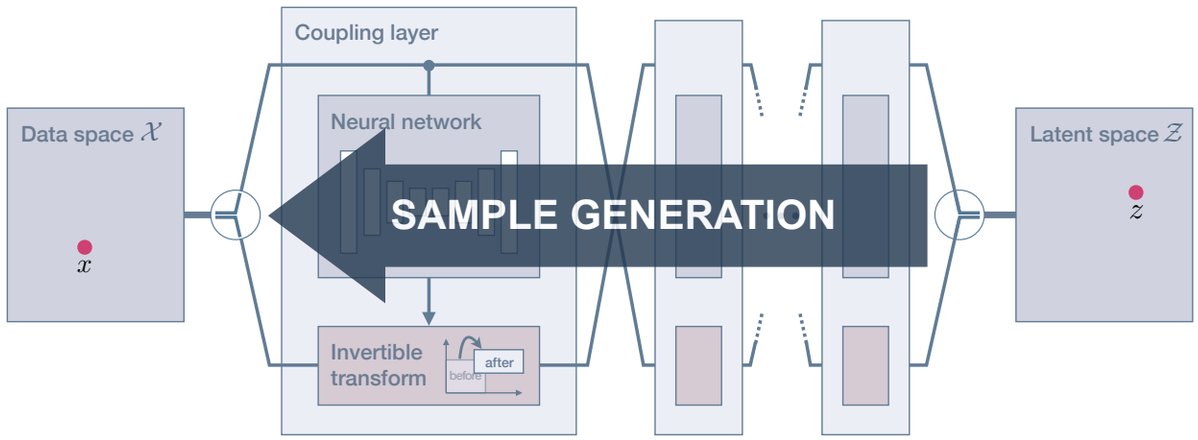
The chain of coupling layers still represents a bijective transform and the computation of the jacobian determinant is relatively cheap.

Once the networks are trained, you can take an existing data point, and evaluate the chain
from left to right to evaluate the probability density of x. In order to generate a new sample,
we pick a point in the latent space, and pass it from right to left to get the corresponding data-space sample.

COUPLING LAYERS

[Dinh et al. 2014, 2016]

GENERATIONS / VANCOUVER 12-16 AUGUST SIGGRAPH2018

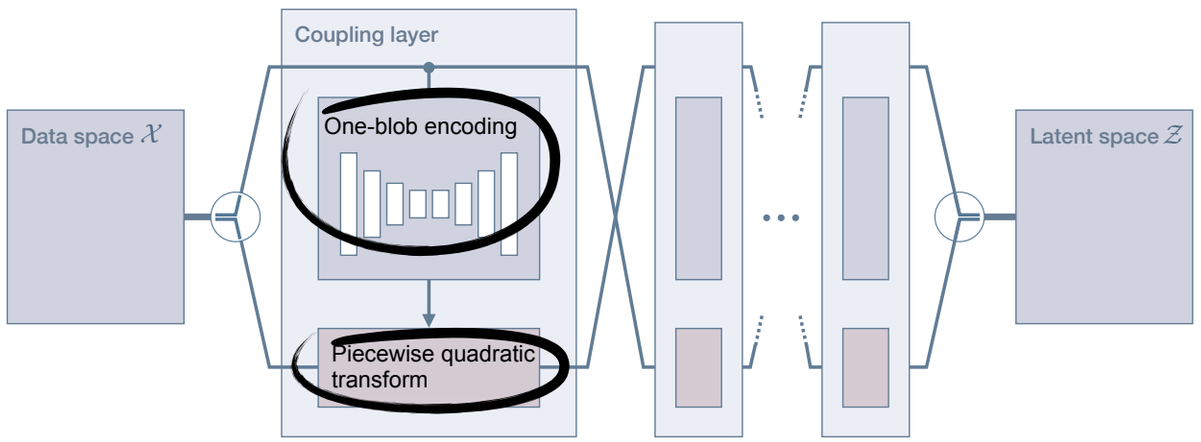


The transform proposed by the authors is a simple translation and scaling, which is clearly not enough to capture the complex relations between the two spaces.
The authors thus propose to chain a number of them, referring to a single block of a neural network and the transform as the “coupling layer”.

The chain of coupling layers still represents a bijective transform and the computation of the jacobian determinant is relatively cheap.

Once the networks are trained, you can take an existing data point, and evaluate the chain
from left to right to evaluate the probability density of x. In order to generate a new sample,
we pick a point in the latent space, and pass it from right to left to get the corresponding data-space sample.

OUR WORK

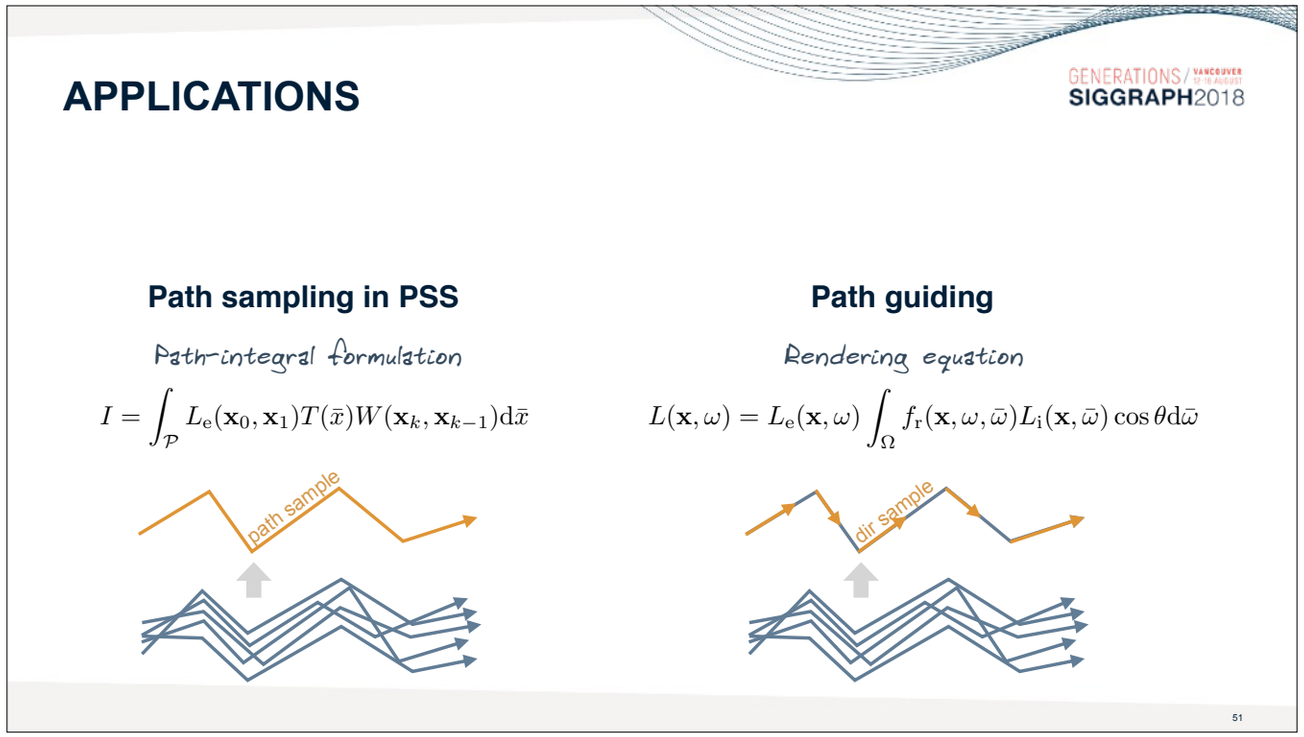


Application to Monte Carlo integration

We took this probabilistic model and proposed a number of improvements:

1) We propose piecewise polynomial transforms to improve the modeling power of each coupling layer. This allows reducing the number of coupling layers.
2) We introduce a one-blob encoding of the inputs to the network to further improve performance.
3) And we study the application to Monte Carlo integration, for instance, we show how to optimize the networks to minimize the variance of a Monte Carlo estimator that utilizes this approach for drawing samples. We show that this is actually equivalent to optimizing the networks using Chi2 divergence.

I will not go into details on any of these; they are well described in our arXiv report. Instead, I will show two applications to light transport.

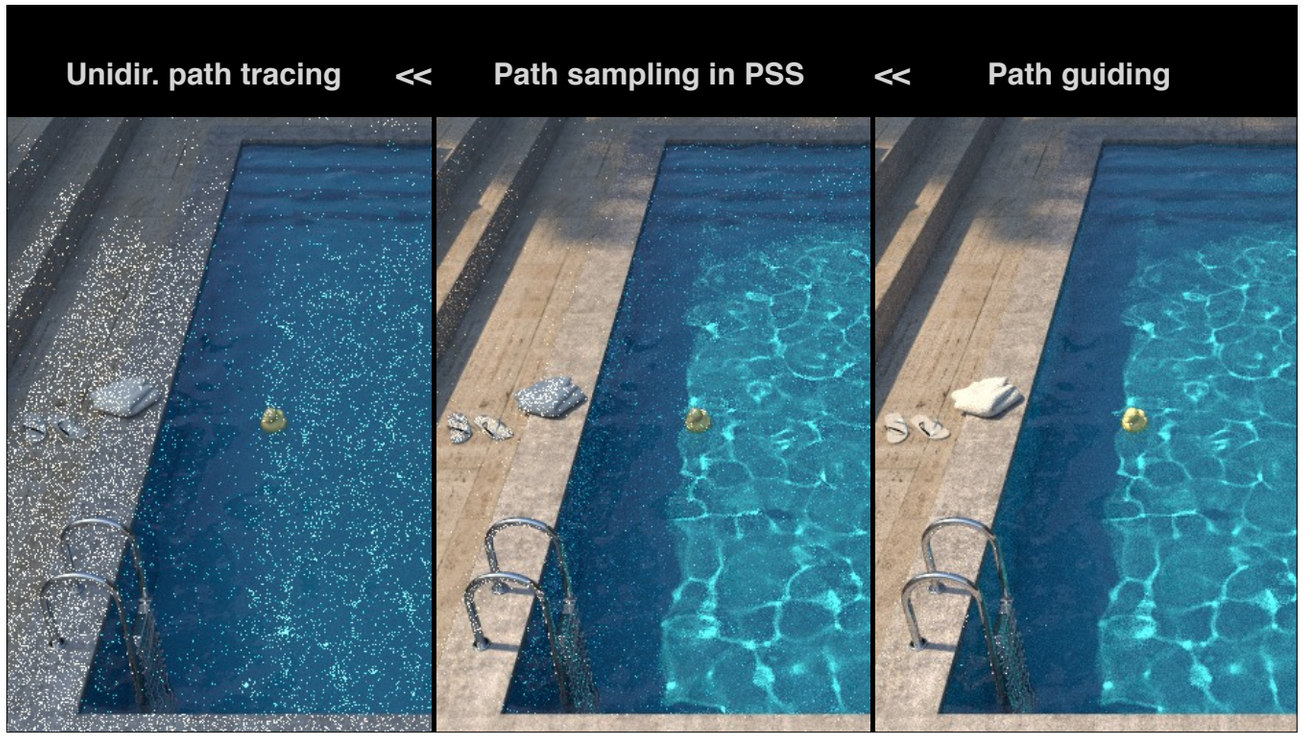


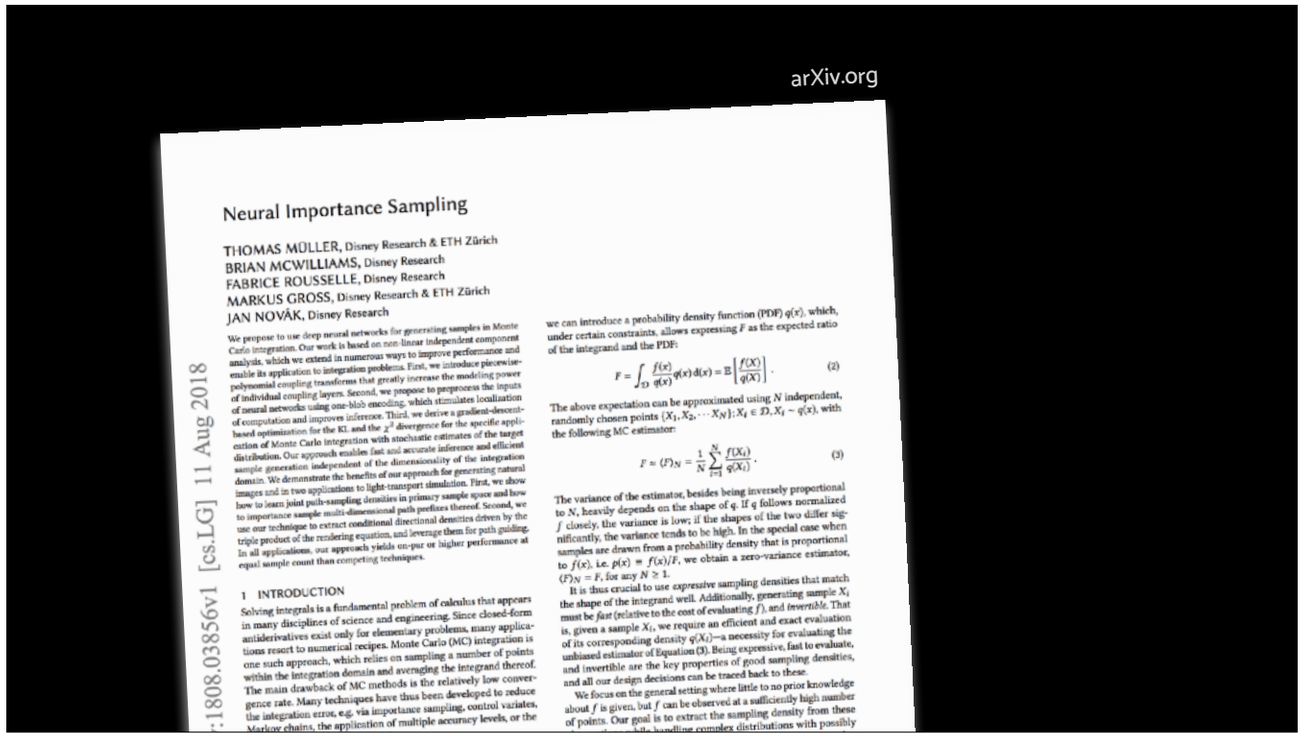
Since the framework is well suited for high-dimensional problems, we applied it to extracting path-sampling densities in the primary sample space. The integral that we are estimating is the primary-sample-space version of the path integral.

As we render, and as we construct entire paths, we learn a density proportional to their contributions in the primary sample space. A new path sample is then constructed by joint importance sampling in primary sample space using the learned distributions.

In the second application, we focus on the rendering equation, and learn a directional density that is proportional to the triple product of the BRDF, incident radiance, and the cosine term. The importance sampling is then applied on each bounce when constructing the path.

I will now show you one of the results we get.

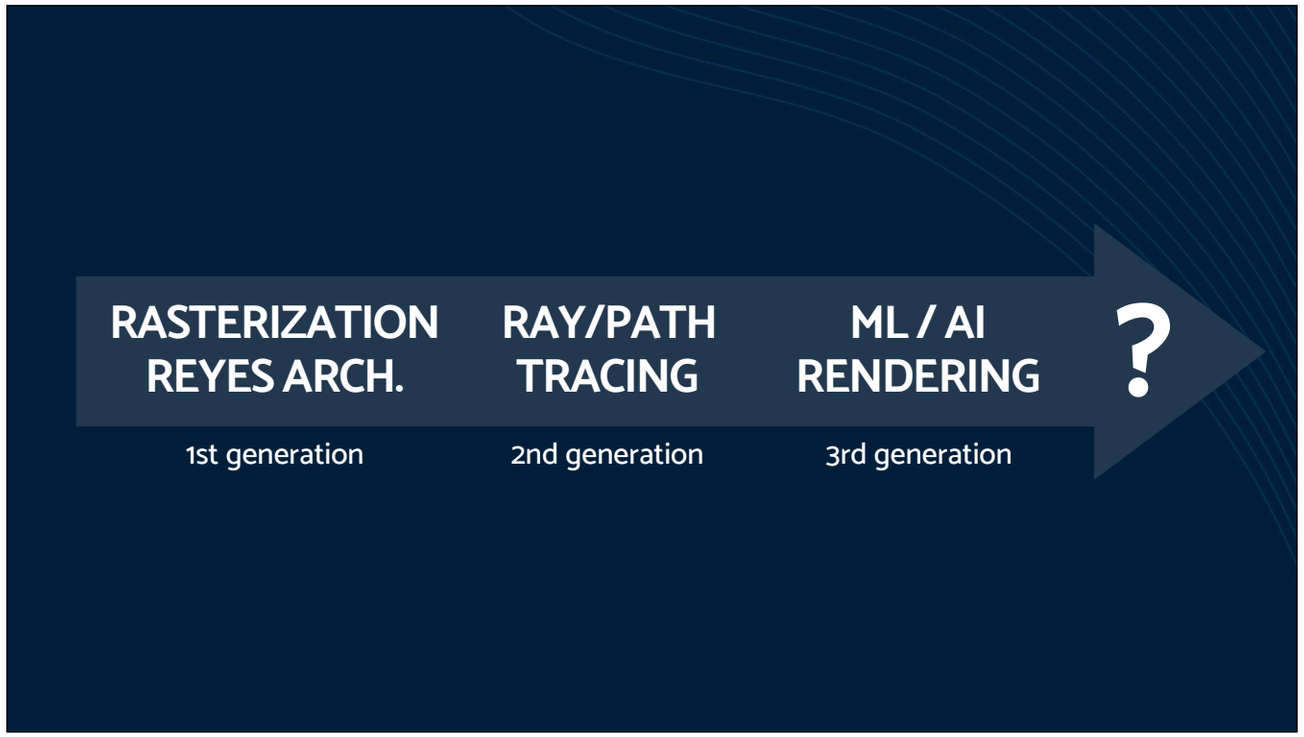

Unidir. path tracing
<<
Path sampling in PSS
<<
Path guiding



You can find many more results in the arXiv report, which is linked from our websites.

FUTURE?

In the rest of the talk, I want speculate a little bit about the future of rendering.



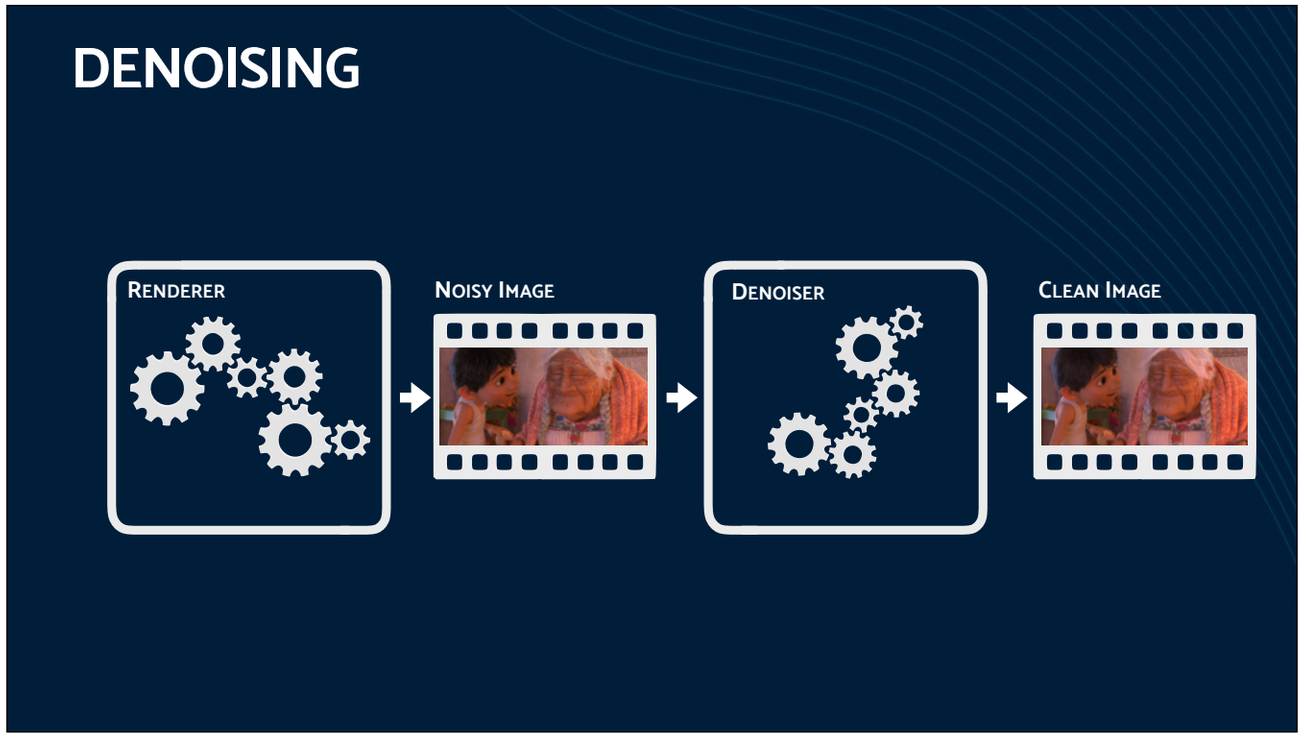
Broadly speaking, we can say that we had two major rendering approaches. The first generation was based on rasterization, interestingly, in both the offline and real-time rendering industries.

The offline industry switched to ray tracing, or path tracing a number of years ago. The same may now happen in real-time rendering.

It is worth asking, what is going to be the next major disruption? What is going to give us the next quantum leap in terms of rendering speed without sacrificing quality?

And it seems entirely plausible that the next generation of renderers could be heavily based on machine learning. That some form of ML will be at the very core of the renderer.

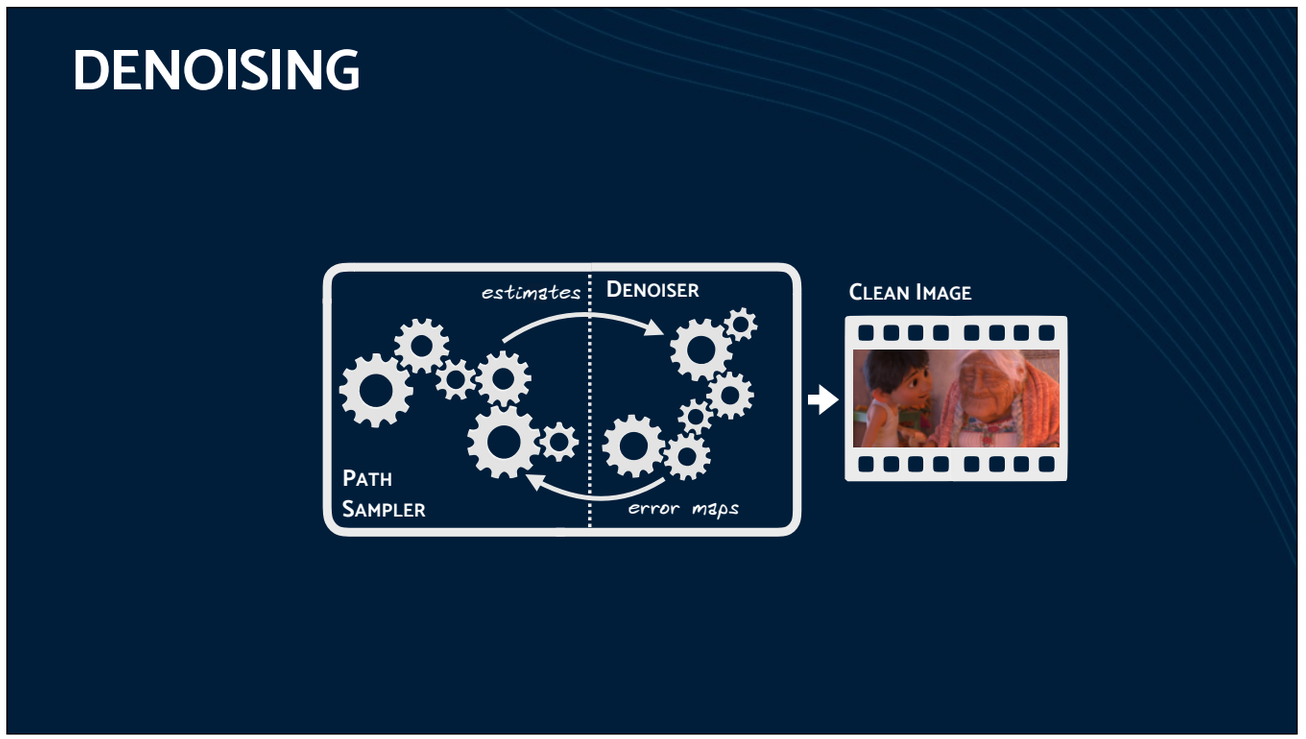
So let me speculate a bit how we could potentially get there.



Let’s look at the example of denoising.

Deep-learning-based denoisers have been shown to outperform traditional denoisers. Right now, the denoiser is executed as a post-processing filter, on images obtained by averaging color samples. There is really no need for the renderer and the denoiser to be two separate processes. The two should be integrated tightly and the renderer should pass individual samples directly to the denoiser. The denoiser can then in turn inform the renderer where to put more samples.

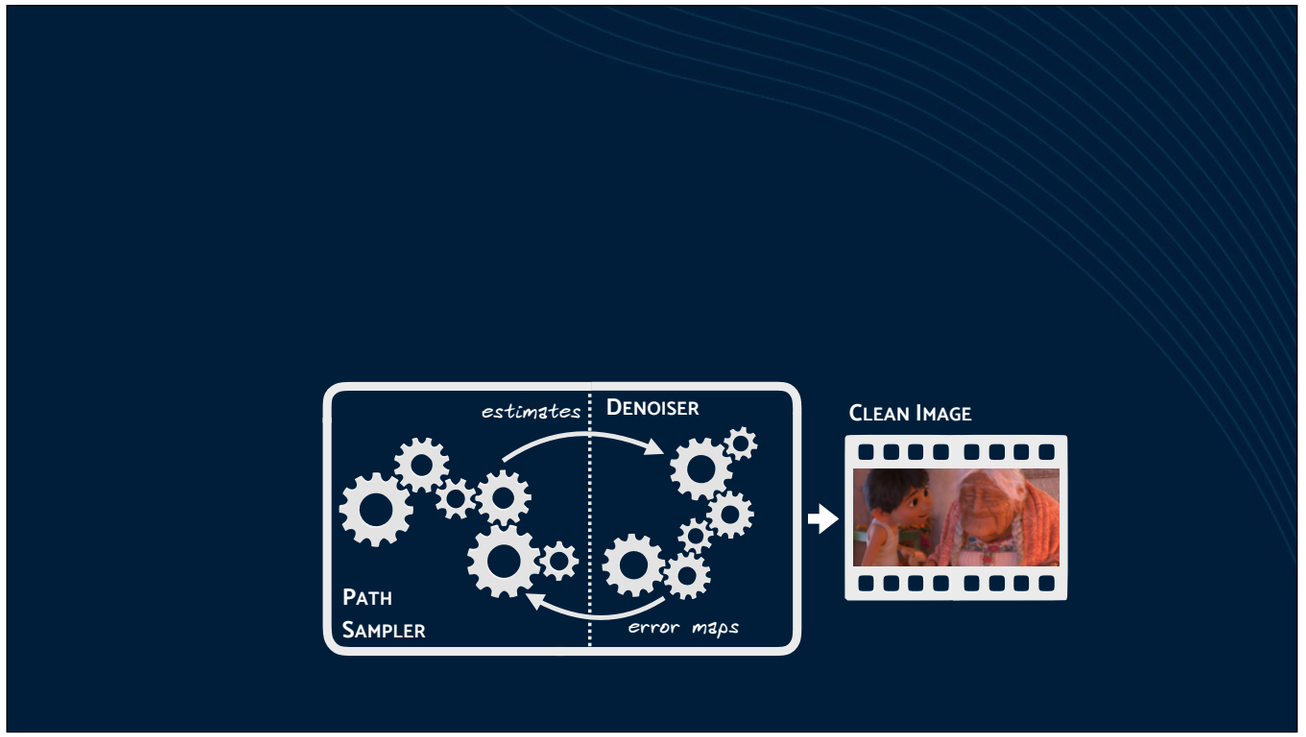
So this is one way how to integrate ML into the core: by making it responsible for final image reconstruction and adaptive sampling.

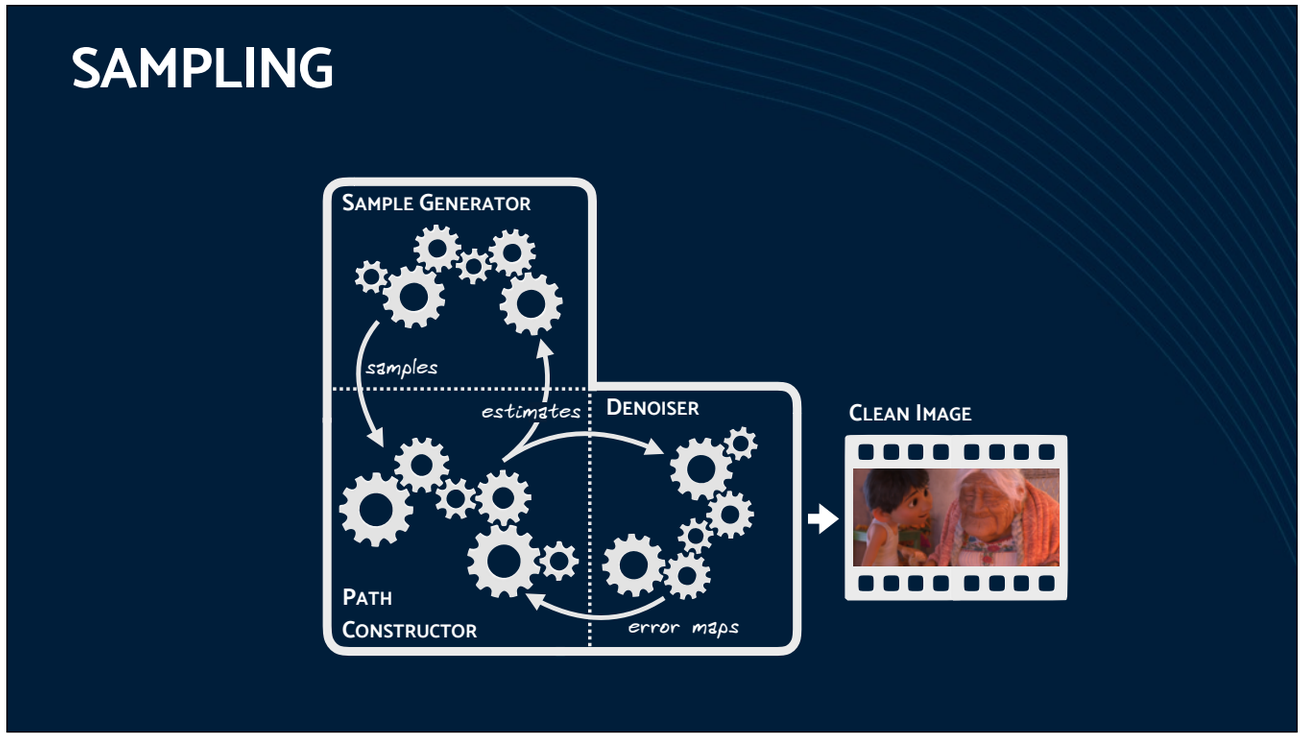


Let’s look at the example of denoising.

Deep-learning-based denoisers have been shown to outperform traditional denoisers. Right now, the denoiser is executed as a post-processing filter, on images obtained by averaging color samples. There is really no need for the renderer and the denoiser to be two separate processes. The two should be integrated tightly and the renderer should pass individual samples directly to the denoiser. The denoiser can then in turn inform the renderer where to put more samples.

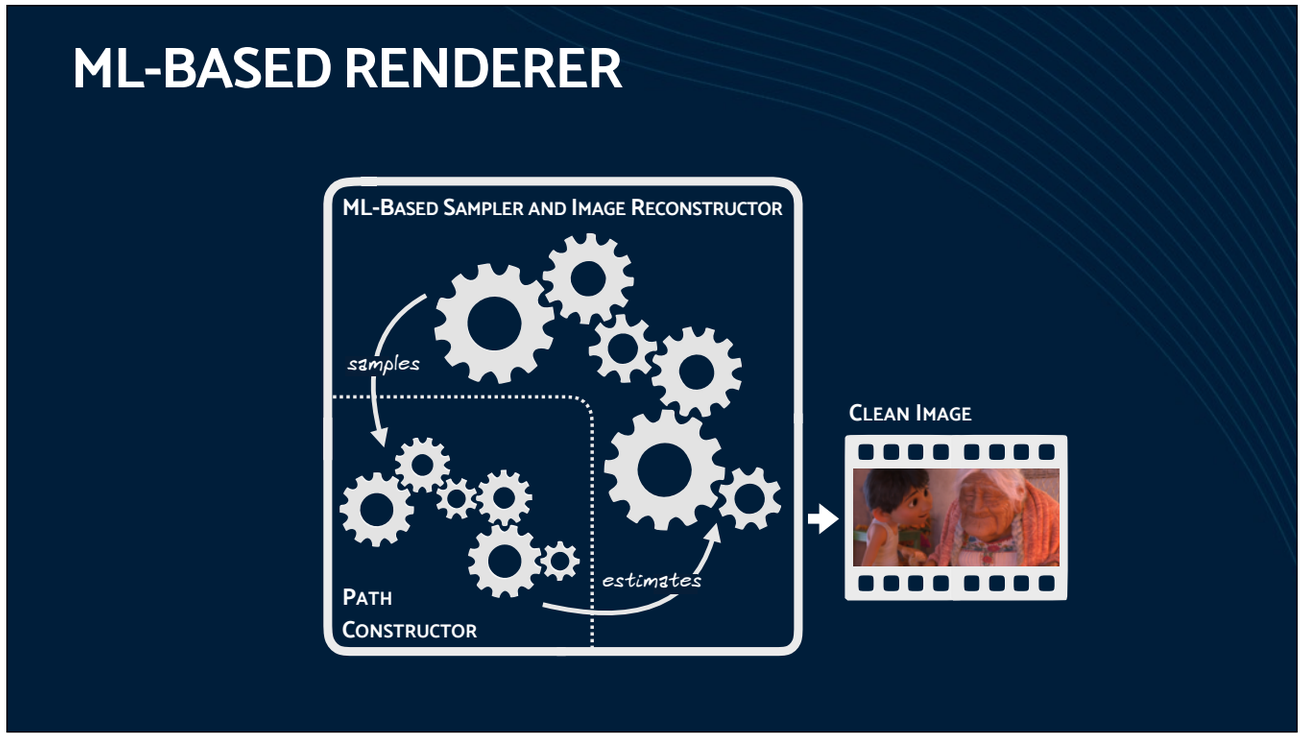
So this is one way how to integrate ML into the core: by making it responsible for final image reconstruction and adaptive sampling.


estimates
DENOISER
PATH
SAMPLER
error maps
CLEAN IMAGE



We have seen today a number of ML approaches for importance sampling.

This is another way how to enhance the renderer with ML: by having a mechanism to extract useful information from various estimates and guide the path construction in the form of providing of providing well-placed samples.



And there is really no reason why these two components should be separate.

In the future, we may see renderers consisting of a path-constructing core, which passes estimates to some learning mechanism that will be in charge of generating samples and reconstructing the clean image, or an entire animation sequence.

In general, there is too much redundancy in rendering to be ignored:
Pixels are similar, frames are similar, even movies are…. sometimes too similar.

The question is “how” and “in what form” to reuse the computation and I think this is an exciting area of research with a lot of potential.

Now I assume a good number of people in this room are very skeptical about machine learning, and probably even more so about deep learning.

And the skepticism is justified, it is really difficult to understand what is going on in these systems and even some very accomplished people in the ML community are concerned and describe machine learning as modern alchemy.

But the fact that we currently don't fully understand these things should not discourage us from looking at them. On the contrary, we should make every attempt to uncover the mysteries of the black box, to visualize its parts so that we can gain understanding, and promote papers that attempt this during reviews.

The black box works too well to be ignored.

So let's embrace this, let's analyze, visualize, and try to understand the box so that it feels a bit more like chemistry instead of alchemy.

Now I assume a good number of people in this room are very skeptical about machine learning, and probably even more so about deep learning.

And the skepticism is justified, it is really difficult to understand what is going on in these systems and even some very accomplished people in the ML community are concerned and describe machine learning as modern alchemy.

But the fact that we currently don’t fully understand these things should not discourage us from looking at them. On the contrary, we should make every attempt to uncover the mysteries of the black box, to visualize its parts so that we can gain understanding, and promote papers that attempt this during reviews.

The black box works too well to be ignored.

So let’s embrace this, let’s analyze, visualize, and try to understand the box so that it feels a bit more like chemistry instead of alchemy.

The last thing I want to mention is a convenience tool that we open-sourced recently. We work with a lot of EXR images when comparing different experiments and so we created a java-script viewer for them. The tool allows you to view EXRs in browsers, change the exposure, zoom, compare, and share the images easily with your co-workers. You can find more about it at jeri.io

ACKNOWLEDGEMENTS

Simon Kallweit

Thomas Müller

Fabrice Rousselle
Brian McWilliams
Thijs Vogels
Marios Papas
Markus Gross
Magnus Wrenninge

PIXAR ANIMATION STUDIOS

THANK YOU!

WALT DISNEY ANIMATION STUDIOS