Karlsruhe Institute of Technology

Real Time Neural Path Guiding

Bachelor Thesis of

Dominik Wüst

At the Department of Informatics
Computer Graphics Group

March 30, 2023

Reviewer:	Prof. Dr.-Ing. Carsten Dachsbacher
Second reviewer:	Prof. Dr. Hartmut Prautzsch
Advisor:	Mikhail Dereviannykh
Second advisor:	Dr. Johannes Schudeiske

Contents

Abstract

With modern GPUs supporting hardware-accelerated ray tracing, we do have the ability to generate path-traced images in realtime. Due to the time constraint, the amount of rays per pixel and frame is limited to a few. This leads to high variance in the Monte Carlo estimate for the Radiative Transfer Equation. In this work we discuss state of the art techniques for reducing variance of direct and indirect illumination with the main focus on rendering dynamic scenes in realtime.

We propose a novel path guiding technique to reduce variance by learning an estimate of the 5D incident radiance field within the scene. A multilayer perceptron is used to partition the spatial domain and a parameterized model is fitted to approximate the directional domain of this radiance field. During rendering this estimated radiance field can be used to guide the paths towards directions of high incident radiance. As a parameterized model we examine the use of von Mises-Fisher distributions and derive a loss function to train the multilayer perceptron in online fashion. We evaluate different strategies to collect training data and test our approach on static and dynamic scenes.

1. Introduction

Realistic images from virtual objects and environments are an important aspect of movies, games and advertisements. Nowadays, most movies and ads on TV do feature digitally created visual effects, not few are entirely digital. Video games are another example for the extensive use of virtual scenes. In a process called rendering an algorithm captures an image of the virtual scene. To do so we define a virtual camera located in the scene to represents our viewpoint. Techniques like rasterization do project the geometry of the scene onto a virtual screen in front of the camera. This approach is very efficient in terms of computational cost, so it is the current standard for most realtime applications such as video games, 3d modeling and animation software as well as most interactive applications such as VR. However, rasterization does not take into account how light interacts with surfaces and distributes within the scene. Rasterized images lack many features we expect from real images, making it obvious they were created digitally.

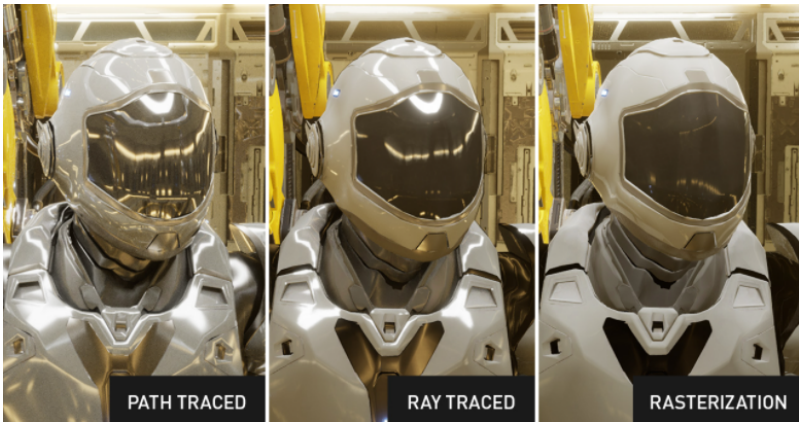


Figure 1.1.: Comparison of different rendering techniques. Specular reflections and indirect illumination are missing in the rasterized image. [Cau22]

For example, soft shadows, imperfect reflections or indirect illumination cannot be recreated by the rasterization algorithm and are missing in rasterized images. However, they are essential for rendering realistic images. To recreate such effects in a virtual scene we simulate light transport. We model light as a ray that is scattered in a random direction as it hits a surface. With this technique we create paths through our scene to connect the camera with a light source, we can then simulate the light transported along this path.

By simulating many such paths and accumulating the light at the camera we are able to render images like 1.2. This algorithm is called path tracing and will be discussed in detail in section 2.3.2. Path tracing has been used in movies and visual effects for a long time by now. But the technique is expensive to compute. The main reason is the need to create many light paths through the scene, in fact to account for all possible light transport we would require infinitely many light paths. Simulating fewer light paths results in noise within our image. The noise is caused by the random decisions when scattering at a surface and decreases as we simulate more light paths.

Figure 1.2.: BistroInterior[Lum17] scene rendered with the path tracing algorithm. On the left only one sample per pixel is computed. The resulting image is noisy. On the right we used 4096 samples per pixel, the noise is reduced significantly.

In section 2.3 we summarize state of the art techniques to reduce variance in a path tracer. We focus on realtime applications where the computation time for a frame is limited to less than 16 ms. In section 2.3.3 we start with techniques for improving direct illumination and discuss their limitations. This motivates the use of path guiding techniques which are able to improve indirect illumination as well. A variety of such techniques is used in offline rendering for movies. However, using them in interactive realtime applications posts additional challenges:

- **Dynamic scenes**: Offline rendering often assumes a static scene without moving objects and with a fixed camera position.[MGN17, HEV+16, VHH+19] These assumptions are not true in most interactive realtime applications.
- **Parallel construction**: For realtime use, most computation is done on a graphic processing unit (GPU). This hardware allows to compute many paths in parallel and is necessary to stay within the time limit for a frame. In contrast offline rendering is often done on the CPU. Algorithms and data structures for offline path guiding can not always take full advantage of the parallel computing capabilities of the GPU, f.e. due to synchronization.
- **Memory constraints**: Accessing the global memory on a GPU can be expensive, especially writing accesses often require synchronization of many parallel units. [MRNK21] Since path guiding is a data driven many techniques create large data structures with an iterative creation process. Memory access can then be a bottleneck on the GPU.

We discuss state of the art path guiding techniques with those challenges in mind in section 2.3.4. This will motivate our research in this area.

In chapter 4 we propose a novel path guiding technique which uses a multilayer perceptron (MLP) to learn the spatially varying radiance distribution in the scene. With this approach we avoid storing large data structures. The MLP we use is very compact. Similar to the recent approach of Müller et al. [MRNK21] we use online training of the MLP instead of pre-computation. This way, our approach can adapt to dynamic changes of the scene. A small set of training samples for the MLP is collected during the path tracing process.

To model the radiance distribution, we examine the use of the von Mises-Fisher (vMF) distribution. This distribution is very simple and consists of only 4 parameters, so it can be learned by a MLP with a small amount of training data. In the path tracer, we use the MLP to estimate the radiance distribution at the current shading point and sample our new scatter direction based on this estimate.

The training of our MLP will be explained in detail in section 4.4 and implementation details are listed in chapter 5. We compare our work to other state of the art techniques and evaluate the performance and cost on different test scenes in chapter 6. At the end, in chapter 7, we discuss our results and the limitations.

Before we dive into the world of light transport simulations, we will introduce the reader to the basics of artificial neural networks and probability theory, as these will be the foundation for the topics we introduced above.

2. Related Work

2.1. Artificial Neural Networks

2.1.1. Multilayer Perceptron

Artificial neural networks are a branch of artificial intelligence. In general these networks consist of interconnected nodes, also called neurons, to form a directional graph. In most applications this graph is structured into layers, each containing an arbitrary number of neurons. To use the neural network, an input and output layer is added and connected to the graph. In the case of feed-forward networks the input is propagated along the weighted edges from layer to layer in a directional manner until it reaches the output layer. There are other network architectures which utilize backward connections between the layers, these are called recurrent neural networks. In this work we focus on small feed-forward networks to be able to apply them to realtime applications. One such feed-forward network is the multilayer perceptron (MLP). As shown in figure 2.1 it consists of an input layer, an arbitrary number of hidden layers and an output layer. The input is propagated from left to right from layer to layer. Each neuron in a layer is connected to all neurons of the previous layer. No layers are skipped and no backward connections exist. Each neuron outputs a signal which is the sum of the inputs modified by a simple nonlinear activation function. The superposition of many nonlinear activation functions enables the MLP to approximate extremely non-linear functions. It has been shown that MLPs can approximate virtually any smooth, measurable function. In contrast to other statistical techniques it does not make assumptions about underlying data distribution. To approximate a function the weights on the edges can be tweaked. By adding more hidden layers or more neurons to a hidden layer, additional connections with weights are added. This gives the MLP more degrees of freedom to approximate more complex functions. The MLP shown in figure 2.1 represents a non-linear mapping between a 3-dimensional input vector and a 2-dimensional output vector.[GD98]

2.1.2. Training

The weights are not tuned by hand but instead learned from training data. The training data is a set of input vectors associated with the desired output vectors. To train the MLP, it is repeatedly presented with the training data and the weights are adjusted until the desired input-output mapping occurs. This kind of training is called supervised learning. During training, the output of the MLP for a given input vector may not equal the desired

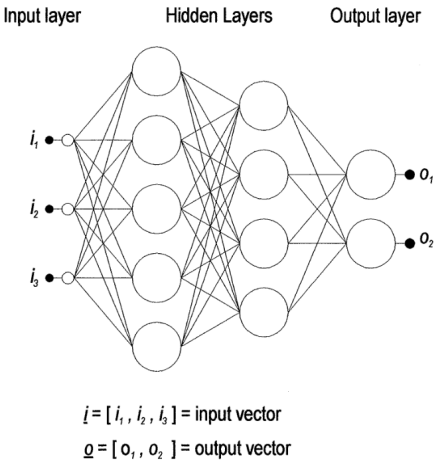


Figure 2.1.: Multilayer perceptron with 2 hidden layers. [GD98]

output. A loss function is defined as the difference between the actual and desired output. Based on this loss function the weights of the MLP are updated. The goal is to minimize the loss.

Backpropagation Algorithm

One method to minimize the loss function for an MLP is the backpropagation algorithm. It relies on gradient descent to reach a minimum on the error surface. The backpropagation algorithm works as follows:

1. initialize network weights with random values
2. present input vector from training data, calculate the network output
3. evaluate loss function for network output and desired output
4. propagate loss back through the network
5. adjust each weight according to the loss
6. repeat steps 2-6 with next input vector until overall error is small enough

For step 4, it is required to derive the loss with respect to each weight of the MLP. Due to the layered structure of the network the gradient of the loss function $\mathcal{L}$ can be computed efficiently using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial o_j} \cdot \frac{\partial o_j}{\partial \text{net}_j} \cdot \frac{\partial \text{net}_j}{\partial w_{ij}}$$

where $o_j = \varphi(\text{net}_j)$ is the j-th component of the output vector, φ is a differentiable activation function and $\text{net}_j = \sum_{i=1}^{n} x_i \cdot w_{ij}$ is the sum of the weighted inputs. [LH91]

As the name of the algorithm describes, we can propagate the loss back through the network by multiplying the partial derivatives. We need to compute the gradient of the loss w.r.t the output, the gradient of the activation function and the gradient of the weighted input sum. This pattern continues through the network. To use specialized activation functions or loss functions later on, we only compute the partial derivatives with respect to the input of the function, the gradients for the weights can be computed easily as a product.

Optimizer

In general, gradient descent is not guaranteed to reach the global minimum, as the gradient decent step might get stuck in local minima. Finding the global minimum of our loss would result in the smallest total loss possible for our training data. However, our training data is only a small subset of all possible inputs. In most applications we want the MLP to generalize so it can make good predictions for previously unseen data. This is a fundamental difference to remembering the training data, as it enables the neural network to make predictions about new data outside the training set. As shown in figure 2.2, reaching a global minimum (green line) on the training data (red and blue dots) does not result in good generalization of the underlying data distribution (black line). In most applications, outliers are present in the training data, f.e. due to measurement errors. In our case it will be noisy estimates as training data. So we do not want to reach the global minimum on our training data but instead achieve a general approximation that ignores some outliers.

To prevent the MLP from overfitting to these outliers, we use the Adam optimizer (Adaptive moment estimate). Adam is a first-order optimizer and relies on the backpropagation of the gradients. In contrast to stochastic gradient descent it uses running estimates of the first and second moments of the gradient to calculate the weight updates. The estimates of the first and second moments avoid reaching the global minimum quickly and help to escape local minima as well. This approach reduces the risk of overfitting while still minimizing the loss function. Adam provides a set of hyper-parameters to control the learning:

- α: the learning rate (also called step size) is the proportion that weights are updated. Smaller values result in slower learning.
- β_1: the exponential decay for the first moment estimate.
- β_2: the exponential decay for the second moment estimate.

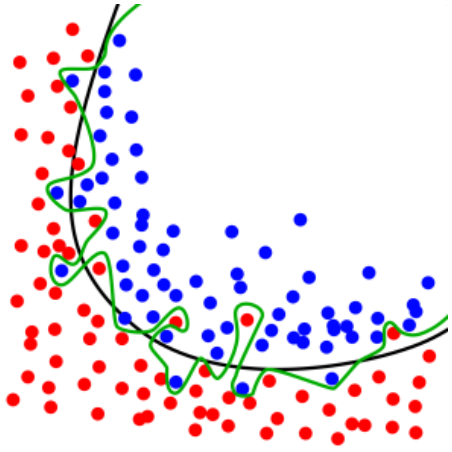


Figure 2.2.: Overfitting [Dra19]

Regularization

Regularization is an additional technique used to prevent overfitting. The idea is to limit the neural network to use only small values for the weights. This reduces the degrees of freedom it has when fitting to the underlying data distribution. To achieve this the sum

of all weights w_{ijk} of the network is added to the loss-function $\mathcal{L}$. Commonly used regularization techniques are either L1 regularization, where the absolute weights are summed, or L2 regularization, where the squared weights are used:

$$\mathcal{L}_{\text{L1reg}} = \mathcal{L} + \lambda \sum \mid w_{ijk} \mid \qquad \mathcal{L}_{\text{L2reg}} = \mathcal{L} + \lambda \sum w_{ijk}^2$$

For weights in the range $(-1, 1)$ L1 regularization adds a higher value to the loss whereas for weights outside this range the loss with L2 regularization will rise much faster due to the quadratic contribution. In both cases the optimizer will not converge to the global minimum of the loss on the training data but instead to a minimum where the weights are small as well. In practice this often results in better generalization. The regularization strength can be controlled with the parameter $\lambda \in [0, \infty)$. [Dra19]

Hardware Acceleration

The MLP implementation we use runs on the graphic processing unit (GPU). Due to the fully-connected architecture of the MLP, the computation for each layer consists of a vector-matrix-multiplication. We multiply the input vector or the output vector of the previous layer with the weight matrix for the current layer. Afterwards we apply the activation function element-wise to get the output of the layer. The element-wise application of the activation can be executed in parallel. The matrix multiplication can be run in parallel as well, latest GPUs do provide specialized hardware accelerated units for matrix multiplication. [MRNK21]

Furthermore we can train the network in batches. Instead of updating the weights after each backpropagation step, we accumulate the weight changes and apply them all at once after we processed a batch of training data. This is much more efficient as we can run multiple instances of the MLP in parallel without synchronizing them to increase training throughput by order of magnitude. Only when the batch has been processed the weight updates of the different instances have to be combined and the instances are synchronized.

2.2. Probability Theory

2.2.1. Probability Distribution

A random variable $X : \Omega \to S$ maps possible outcomes Ω of a random process to a measurable space S. In our case, this measurable space is a subset of real numbers or the real numbers themselves. Then the cumulative distribution function (CDF) of the random variable is defined as

$$F : \mathbb{R} \to [0, 1], \quad x \mapsto \mathbb{P}(X \leq x)$$

The CDF is non-decreasing with $\lim_{x\to\infty} = 1$ and $\lim_{x\to-\infty} = 0$. If a function f exists such that

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

We call f a probability density function (PDF) or density of F. This PDF is non-negative ($f(t) \geq 0, \forall t$), integrates to 1 ($\int_{-\infty}^{\infty} f(t)dt = 1$) and is not unique in general. Any function with those properties is a PDF for a probability distribution if we can integrate it. [DH06]

One important probability distribution is the uniform distribution. On the range $[0, 1)$ it is defined by the PDF:

$$f_{\text{uniform}}(t) = \begin{cases} 1, & \text{if } 0 \leq t < 1 \\ 0, & \text{otherwise} \end{cases}$$

This distribution is the foundation of random processes in computers, as most (pseudo) random number generators used in software draw random numbers which are uniformly distributed.

2.2.2. Inverse CDF Method

As discussed above we can generate uniformly distributed random numbers in a computer. However in many scenarios using the uniform distribution is not optimal and we want to generate random numbers with a different PDF. The inverse CDF method can be used to transform a uniformly distributed random variable such that it is distributed according to a given PDF f. We obtain the CDF F by integrating. As the CDF is non-decreasing, we can define its inverse as

$$F^{-1}(y) = \inf\{x \in \mathbb{R} : F(x) \geq y\}$$

If a random variable X is uniformly distributed on $[0, 1)$ then the random variable $Y := F^{-1}(X)$ has the density f. [MLM18]

This can be extended to multivariate densities as well. In our work we focus on the 2-dimensional case. Then the the density is $f(x, y)$ where $x \in [\underline{x}, \overline{x}), y \in [\underline{y}, \overline{y})$. We define the marginal densities

$$f_X(x) = \int_{\underline{y}}^{\overline{y}} f(x, y)\mathrm{d}y \qquad\qquad f_Y(y) = \int_{\underline{x}}^{\overline{x}} f(x, y)\mathrm{d}x$$

with marginal CDFs $F_X(x) = F(x, \overline{y})$ and $F_Y(y) = F(\overline{x}, y)$ as well as the conditional density

$$f(y|x) = \frac{f(x, y)}{f_X(x)}$$

with the conditional CDF $F(y|x) = F(x, y)/F_X(x)$. For the inverse of this conditional CDF to exist, the PDF has to be separable, so we are able to write it as $f(x, y) = g(x)\cdot h(y)$.

Now we use uniformly distributed random variables U, V and transform

$$U' := F_X^{-1}(U)$$
$$V' := F^{-1}(V|U')$$

The resulting random variable (U', V') is distributed according to our density $f(x, y)$. [MLM18]

2.2.3. Distributions on the Unit Sphere

Our work relies on random directions heavily. A direction in 3D space can be expressed as a point on the unit sphere. The 3-dimensional unit sphere is given as

$$\mathcal{S}^2 := \{x \in \mathbb{R}^3 :\| x \|_2= 1\}$$

Points on the unit sphere can be expressed in Cartesian coordinates $(x, y, z) \in \mathbb{R}^3$ or in spherical coordinates $(r, \theta, \varphi) \in [0, \infty) \times [0, \pi) \times [0, 2\pi)$. Since we are expressing points on the unit sphere, the distance to the origin is 1 for every such point. In case of spherical coordinates this means $r = 1$ and we can express our direction using only θ and φ. In many applications it can be useful to switch between the coordinate systems using

$$\Phi \begin{pmatrix} r \\ \theta \\ \varphi \end{pmatrix} = \begin{pmatrix} r\sin\theta\cos\varphi \\ r\sin\theta\sin\varphi \\ r\cos\theta \end{pmatrix}, \qquad \Phi^{-1} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sqrt{x^2+y^2+z^2} \\ \arccos\frac{z}{\sqrt{x^2+y^2+z^2}} \\ \mathrm{sgn}(y)\arccos\frac{x}{\sqrt{x^2+y^2}} \end{pmatrix} \tag{2.1}$$

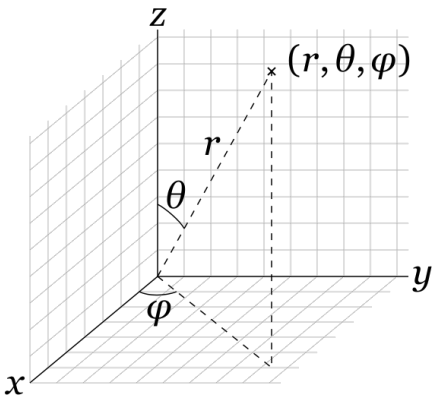


Figure 2.3.: Spherical coordinates [And22]

Uniform Distribution on the Unit Sphere

Generating uniformly distributed random directions is not as straightforward as one might expect. An intuitive approach is using 3 independent uniformly distributed random variables. This yields a 3D vector which is uniformly distributed within $[0,1)^3$ and can be scaled and translated to be within the $[-1,1)^3$. When normalizing this random vector x we end up with $x' = \frac{x}{\|x\|_2} \in \mathcal{S}^2$. This approach has multiple drawbacks, first of all it does require to drawing 3 random numbers. Also, if the random vector x is very close to the origin, the normalization $\|x\|_2$ can cause numerical issues. If we generate $(0,0,0)$ it would cause a division by zero, so in these cases we are required to generate even more random numbers. This can be costly and since we are going to need random directions a lot it is worth optimizing the sample generation.

Finding a PDF for a uniform distribution on the unit sphere is simple, as the only requirements are for it to be constant and normalized. For the unit sphere, we normalize by its surface area, which is 4π. Thus we use the following PDF:

$$f_{\text{uniform}}(\omega) = \frac{1}{4\pi}, \quad \omega \in \mathcal{S}^2$$

We can use the inverse CDF method to generate random directions according to this PDF. Note that directions ω are 3D-vectors, so the CDF for our PDF ends up with a 3-dimensional integral as well. To avoid this we transform our PDF to spherical coordinates using Φ (see equation 2.1). Note that the Jacobian determinant $|\det J|$ of the transformation is added according to the transformation rule: [MLM18]

$$p_{\text{uniform}}(\theta,\varphi) = f_{\text{uniform}}(\Phi(1,\theta,\varphi)) \cdot |\det J|$$

In case of the transformation to Cartesian coordinates, the Jacobian determinant is $|\det J| = |\sin\theta|$. This PDF is separable and we can use the inverse CDF method. We start with integrating the PDF to get the CDF:

$$P_{\text{uniform}}(\theta,\varphi) = \int_0^{\varphi}\int_0^{\theta} p_{\text{uniform}}(\theta',\varphi')d\theta' d\varphi' = \int_0^{\varphi}\int_0^{\theta} \frac{1}{4\pi}\sin\theta' d\theta' d\varphi' = \frac{(1-\cos\theta)\varphi}{4\pi}$$

Then we invert the marginal CDF and conditional CDF:

$$P_\theta(\theta) = P(\theta, 2\pi) = \frac{1-\cos\theta}{2} \Rightarrow P_\theta^{-1}(u) = \arccos(1-2u)$$

$$P(\varphi|\theta) = \frac{P(\theta,\varphi)}{P_\theta(\theta)} = \frac{\varphi}{2\pi} \Rightarrow P^{-1}(v|u) = 2\pi v$$

Finally, we use uniformly distributed random variables U, V and transform them to get the 2-dimensional random variable $(\arccos(1-2U), 2\pi V)$, which is uniformly distributed on the unit sphere. This vector is in spherical coordinates and can be converted to Cartesian coordinates using equation 2.1.

Von Mises-Fisher Distribution

Definition

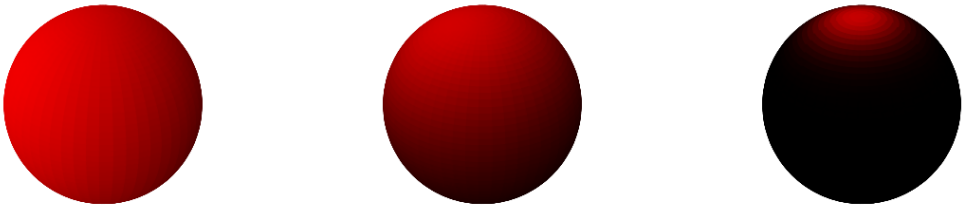
The von Mises-Fisher (vMF) distribution is a general-purpose distribution well suited for statistical representation of directional data. For an 3-dimensional random unit vector $\omega \in \mathcal{S}^2$ the PDF is given as

$$f_{\text{vMF}}(\omega; \mu, \kappa) = \frac{\kappa}{4\pi \sinh \kappa} e^{\kappa \mu^T \omega}$$

where $\kappa \geq 0$ is a concentration parameter and $\mu \in \mathcal{S}^2$ is the normalized mean direction. For $\kappa \to 0$ the vMF distribution approaches a uniform distribution on the unit sphere.

[Jak12] shows several numerical issues related to this formulation, f.e. $\sinh 100 = 1,34406 \cdot 10^{43}$, which overflows single precision floating point operations. They recommend using the following equivalent formulation for the density:

$$f_{\text{vMF}}(\omega; \mu, \kappa) = \begin{cases} \frac{1}{4\pi}, & \text{if } \kappa = 0 \\ \frac{\kappa}{2\pi(1-\exp(-2\kappa))} e^{\kappa \mu^T \omega - 1}, & \text{if } \kappa > 0 \end{cases} \tag{2.2}$$



$\kappa = 0.1$ $\kappa = 1$ $\kappa = 10$

Figure 2.4.: Density of the vMF distribution plotted with mean direction up ($\mu = (0,0,1)^T$) and different values for the concentration parameter.

Sampling

In our use case, it is important to generate independent samples which are distributed according to the vMF distribution. [Ulr84] use two random variables $V \in \mathbb{R}^2$ and $W_\kappa \in [-1, 1]$ to construct a vector

$$\omega_\kappa = (\sqrt{1 - W_\kappa^2} V, W_\kappa)^T$$

which is distributed according to the vMF distribution with fixed mean vector $\mu = (0,0,1)^T$ and arbitrary concentration κ. For this to work, V has to be uniformly distributed on the unit circle and W_κ has to be distributed according to the density

$$f(w) = \frac{\kappa}{2 \sinh \kappa} e^{\kappa w}$$

Again we use the inverse CDF method. First we integrate to get the CDF

$$F(w) = \int_{-1}^{w} f(w')dw' = \left[\frac{1}{2 \sinh \kappa} e^{\kappa w'} \right]_{w'=-1}^{w'=w} = \frac{1}{2 \sinh \kappa} (e^{\kappa w} - e^{-\kappa})$$

and invert it

$$F^{-1}(u) = \frac{\log(2u\sinh(\kappa) + e^{-\kappa})}{\kappa}$$

Due to the same reasons as the vMF density, this inverse suffers from numerical issues when using large concentration parameters. We thus use a more stable formulation: [Jak12]

$$F^{-1}(u) = 1 + \frac{\log(u + (1-u)e^{-2\kappa})}{\kappa}$$

Now we can get our random variable $W := F^{-1}(U)$ by transforming a uniformly distributed random variable U.

For V to be uniformly distributed on the unit circle, we use a similar approach as described in 2.2.3. Since we are dealing with a 2D vector this time we use polar coordinates $(r, \varphi) \in [0, \infty) \times [0, 2\pi)$. We are interested in points on the unit circle only, so the radius $r = 1$. We can sample the angle $\varphi \in [0, 2\pi)$ uniformly by scaling a random variable S with factor 2π. Then we transform the result back to Cartesian coordinates using

$$\Phi' \begin{pmatrix} r \\ \varphi \end{pmatrix} = \begin{pmatrix} r\cos\varphi \\ r\sin\varphi \end{pmatrix}$$

which results in $V = \Phi'(1, 2\pi S)^T = (\cos(2\pi S), \sin(2\pi S))^T$ for a uniformly distributed random variable S.

As mentioned above, with these two random variables V, W, we can generate a random vector ω_κ. However this one is distributed according to a vMF with mean direction $(0, 0, 1)^T =: u$. To generate samples for arbitrary mean directions $\mu = (\mu_1, \mu_2, \mu_3)^T$ we sample ω_κ as described and rotate it afterwards. We describe the rotation by a matrix $R \in \mathbb{R}^{3\times3}$ such that $Ru = \mu$.

To find such a rotation matrix R, we first search for a rotation axis a, which is orthogonal to both u and μ. Since u is constant we can use

$$\tilde{a} = (\mu_2, -\mu_1, 0)^T$$

This axis is obviously orthogonal to u and $\tilde{a}^T\mu = \mu_1\mu_2 - \mu_1\mu_2 = 0$, so it is orthogonal to μ as well. We normalize it and get $a = \tilde{a}/\|\tilde{a}\|_2$. Next we determine the rotation angle $\theta = \angle\mu, u$, which can be computed by taking the product of both vectors

$$\cos\theta = \mu^T u = \mu_3$$

Finally, we can compute our rotation matrix

$$R = (\cos\theta)I_3 + (\sin\theta)[a]_\times + (1 - \cos\theta)(a \otimes a)$$

where I_3 is the 3-dimensional identity matrix, $[a]_\times$ is the cross product matrix of a and $\otimes$ denotes the outer product. [HK22] With this rotation matrix in place, we can now generate random directions ω according to a vMF with arbitrary parameters μ, κ by generating ω_κ as mentioned above and rotating it like this: $\omega = R\omega_\kappa$.

Derivatives

In order to fit the distribution to our training data later on, we need the partial derivative of 2.2 with respect to both parameters $\mu \in \mathcal{S}^2$ and $\kappa \in [0, \infty)$:

$$\frac{\partial}{\partial\mu_i} f_{\text{vMF}}(\omega; \mu, \kappa) = \frac{\kappa^2\omega_i}{2\pi(1 - \exp(-2\kappa))} e^{\kappa\mu^T\omega - 1}, \qquad i = 1, 2, 3 \tag{2.3}$$

$$\frac{\partial}{\partial\kappa} f_{\text{vMF}}(\omega; \mu, \kappa) = \frac{1}{2\pi(1 - \exp(-2\kappa))} e^{\kappa\mu^T\omega - 1} \left(\kappa\mu^T\omega - \frac{2\kappa}{1 - \exp(2\kappa)} e^{-2\kappa} + 1\right) \tag{2.4}$$

2.2.4. Kullback-Leibler Divergence

Kullback-Leibler divergence (KL divergence) is a score to quantify the difference between two probability distributions P and Q. In our case we use continuous distributions with densities $p(x)$ and $q(x)$:

$$D_{\mathrm{KL}}(P \parallel Q) = \int_X p(x) \cdot \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x$$

It is not a metric since it is not symmetrical: $D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P)$. P therefore should be the ground-truth and Q the model.

2.2.5. Monte Carlo Integration

Concept

Monte Carlo (MC) methods in general rely on repeated random sampling to solve a problem. It can be used to solve integrals numerically. In light transport simulations, MC is the de facto standard to solve integrals. In contrast to other numerical approaches, it is independent of the dimension of the integral.

The expected value $\mathbb{E}[X]$ for a continuous random variable X with realizations x and probability density $p(x)$ is given as:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x \cdot p(x) \mathrm{d}x$$

If we repeat a random experiment N times, according to the law of large numbers, the average of the results will approach the expected value for large N:

$$\mathbb{E}[X] \approx \frac{1}{N} \sum_{i=1}^{N} x_i$$

Here x_i are the results of the experiment. The approximation will be better for larger N as the average approaches the expected value. For MC integration we combine both definitions for the expected value:

$$\mathbb{E}[g(x)] = \int g(x) p(x) \mathrm{d}x \approx \frac{1}{N} \sum_{i=1}^{N} g(x_i)$$

Now we use $g(x) = \frac{f(x)}{p(x)}$ and obtain the generic case for solving an integral with integrand f:

$$\mathbb{E}\left[\frac{f(x)}{p(x)}\right] = \underbrace{\int f(x) \mathrm{d}x}_{I} \approx \underbrace{\frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}}_{\hat{I}} \tag{2.5}$$

Note that the MC-estimator $\hat{I}$ for approximating the integral I is independent of the integral dimension.

Variance and Bias

We are approximating and thus we introduce some error. In case the error approaches 0 for large enough N, our estimator is unbiased and the error manifests itself as variance which decreases with more samples. In case of images we experience this variance as noise (see figure 1.2). If the error does not approach 0 however our estimator is biased, this happens if the mean of our random variable is not the integral we were solving. For a biased estimator, we also experience noise which reduces with larger N. However we converge to a wrong value. For images this means some areas are too dim or too bright. In figure 2.5, a numerical integration is performed using a MC estimator. As the estimate approaches the real value of the integral, this estimator is unbiased. Variance of the estimate is proportional to $1/N$.[MGN17]

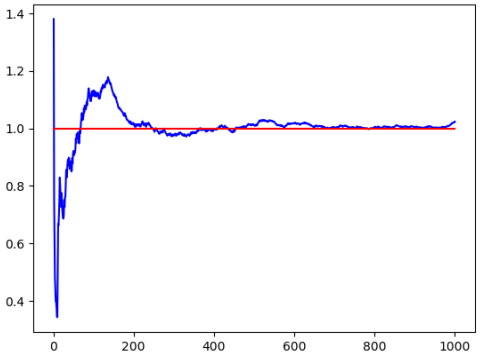


Figure 2.5.: Noisy MC estimate of an integral with increasing sample count N on the x axis. The analytic solution of the integral is 1 (red).

Importance Sampling

Furthermore the probability density $p(x)$ can be chosen arbitrarily if it is normalized and $f(x) > 0 \implies p(x) > 0$. So if we choose $p(x) = f(x)/\int f(y)\mathrm{d}y$ our MC estimator will calculate the exact result:

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{f(x_i)/\int f(y)\mathrm{d}y} = \frac{1}{N}\sum_{i=1}^{N}\left(\int f(y)\mathrm{d}y\right) = \int f(y)\mathrm{d}y = I$$

In practice it is not possible to use this probability density function as it requires the solution of the integral we want to solve in the first place. However it shows that it is optimal to distribute the samples proportional to the integrand [HIT+23]

$$p(x) \propto f(x)$$

This approach is called importance sampling and can reduce the variance of an MC estimator significantly.

2.3. Light Transport

2.3.1. Radiative Transfer Equation

The idea of light transport simulation is to compute the emission and scattering of light on different surfaces. To quantify light, we introduce the term radiance $L(x, \omega)$ to describe radiant power per area and solid angle, so its unit is $\frac{W}{m^2 sr}$. The radiance is the central unit when dealing with realistic light transport simulations. The radiative transport equation (RTE) is used to describe the radiance $L_o(x, \omega)$ leaving a surface point x in a direction ω: [Kaj86, MGN17]

$$L_o(x, \omega) = L_e(x, \omega) + \int_\Omega f_r(\omega, x, \omega_i) L_i(x, \omega_i) \cos\theta_i \mathrm{d}\omega_i \tag{2.6}$$

Here $L_e(x, \omega)$ is the radiance emitted by the surface in direction ω, f.e. on a light source. The integral describes the radiance that is reflected on the surface and is often called the reflection integral. $L_i(x, \omega_i)$ is the incident radiance at the surface point x from direction ω_i. This incident radiance is weighted with the angle $\cos\theta_i$ between the surface normal and the incoming direction ω_i. The bidirectional reflection distribution function (BRDF) $f_r(\omega, x, \omega_i)$ describes the surface properties such as color or reflectivity. So the multiplication with f_r turns incident radiance L_i from direction ω_i into radiance leaving the surface in direction ω. To take into account incident radiance from every direction, we integrate over the hemisphere Ω.

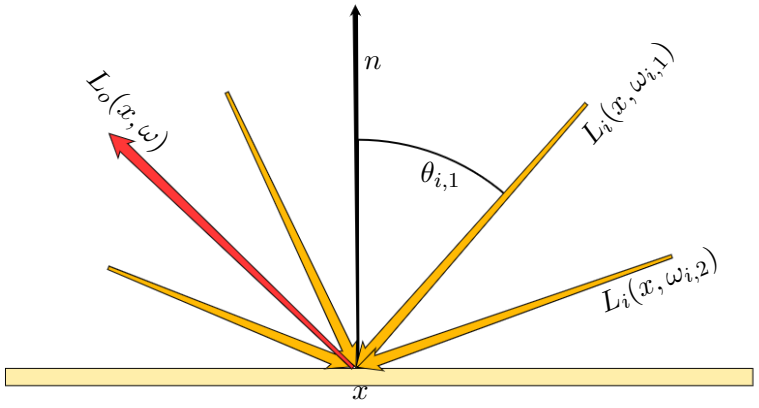


Figure 2.6.: Components of the RTE at a surface point x: outgoing radiance L_o in red, incident radiance L_i from different directions $\omega_{i,1}, \omega_{i,1}, ...$ in orange and the angle $\theta_{i,1}$ of an incoming direction with the surface normal n in black.

While the RTE is a compact representation of all the light transport within the scene, it is hard to compute due to the integration of L_i over the hemisphere. We cannot express L_i in a closed form which we can integrate analytically. However we can express incident radiance $L_i(x, \omega_i)$ in terms of outgoing radiance $L_o(y, \omega_i)$ from another point y:

$$L_o(x, \omega) = L_e(x, \omega) + \int_Y f_r(\omega, x, \omega_i) L_o(y, -\omega_i) g(x, y) \mathrm{d}y \tag{2.7}$$

where Y is the set of all surface points y in the scene, $g(x, y)$ is a geometry term which is zero if y is not visible from x and direction $\omega_i = \frac{y-x}{\|y-x\|}$ between the points. This formulation reveals the recursive nature of the RTE, which is the reason it is so difficult to solve. [Kaj86, Vea98]

This leads us to the path tracing algorithm that is widely used to approximate a solution of the RTE.

2.3.2. Path Tracing

First of all, we will model the light as a ray that travels in straight lines rather than a wave. Using modern raytracing hardware on the GPU, we can simulate many such rays in realtime. Furthermore we assume the surrounding media to be a vacuum, so no particles are blocking rays other than the surfaces of our virtual scene. To create a 2D image of our 3D virtual scene, we define a camera within the scene which represents our position and viewing direction. The idea of path tracing is to create transport paths $X = (x_0, x_1, ..., x_n)$ through the scene such that x_0 is the camera position, x_n is a point on a light source and $g(x_i, x_{i+1}) > 0, i = 0, .., n-1$. Such a path connects the camera with a light source and can be used to approximate the incident radiance at the camera. We call each point x_i a path vertex.

There are different techniques to create such a path, we will discuss some of them in this work. To ensure we get an estimate for every pixel of the image we are rendering, we start our paths at the camera. To estimate the color of a pixel we trace a ray through this pixel to find a surface point x_1. The direction of this ray is $-\omega$. We want to evaluate the RTE to get the radiance leaving that surface point towards our camera, so we are looking for $L_o(x_1, \omega)$.

We can evaluate $L_e(x_1, \omega)$ right away as this is part of the material of the surface. As mentioned above, the hemispheric reflection integral can not be evaluated in close form. We rely on Monte Carlo integration as described in section 2.2.5 and approximate the integral using a sum of N samples [MGN17]

$$\int_\Omega f_r(\omega, x_1, \omega_i) L_i(x_1, \omega_i) \cos\theta_i \mathrm{d}\omega_i \approx \frac{1}{N} \sum_{j=1}^{N} \frac{f_r(\omega, x_1, \omega_j) L_i(x_1, \omega_j) \cos\theta_j}{p(\omega_j)}$$

where directions $\omega_1, ...\omega_N$ are randomly sampled and distributed according to the density p. Now we can evaluate both the cosine as well as the BRDF f_R.

To evaluate $L_i(\omega, x_1, \omega_j)$ we trace another ray from x_1 in direction ω_j to find the next surface point x_2 where again we want to evaluate $L_o(x_2, \omega_j)$. So we repeat the above steps and use MC integration again. With this approach we can create paths of arbitrary length. Depending on the path, length we will be able to simulate more complex indirect illumination. However, in realtime applications, it is common to use a fixed path length to reduce computational cost.

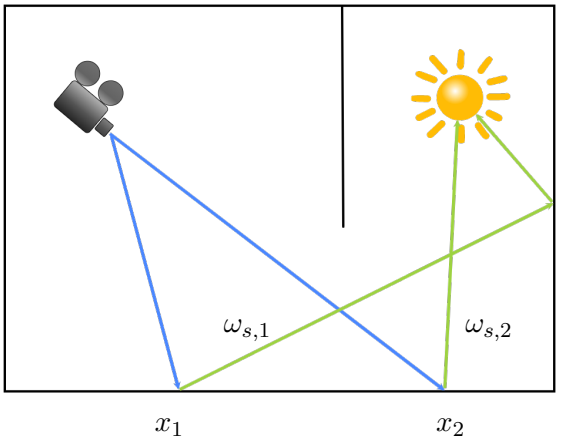


Figure 2.7.: Two possible paths generated during path tracing. $\omega_{s,i}$ is a random scatter direction at this surface point.

2.3.3. Importance Sampling

Unidirectional path tracing, as described above, is both simple to implement and flexible, while enabling complex light transport simulations. Due to the randomness introduced by the MC estimator at each path vertex, the resulting images are noisy. As described in section 2.2.5 this variance can be reduced by using more samples N for the estimate. The variance of the estimate is proportional to $1/N$ [MGN17], so this is costly and in realtime applications only few samples per pixel are possible.

An alternative approach is importance sampling. As described in section 2.2.5, we can reduce the variance of the MC estimator by choosing the probability density p such that it is correlated with the integrand of the reflection integral. So in our case $p \propto f_r(\omega, x, \omega_i) L_i(x, \omega_i) \cos\theta_i$. However, in practice, this can be very hard to achieve due to the complexity of the different components.[BWP+20] Instead we can focus on importance sampling the components of the integrand individually and combine the results later on. For example the cosine lobe can be importance sampled easily as we have an analytical representation that we can use as a PDF for sampling.

BRDF Importance Sampling

The bidirectional reflection distribution function (BRDF) $f_r(\omega, x, \omega_i)$ is one component of the reflection integral. It describes the material properties of the surface and reflects incident radiance to outgoing radiance in direction ω. Many BRDF functions are based on a closed-form representation which is designed with importance sampling in mind, so a PDF can be derived analytically. For example, in microfacet-based BRDFs the distribution of the microfacet normals can be importance sampled [KE09, Hd14]. There are techniques to importance sample BRDF models based on measured data as well f.e. by fitting a gaussian mixture to the data [KE09, HEV+16]. Importance sampling the BRDF is particularly effective for secular or glossy materials such as metal, mirrors or paint. This is because these materials have a strong correlation between incoming direction and outgoing direction.

Next Event Estimation

The incident radiance L_i is difficult to importance sample as we don't have an analytical closed-form equation for it. We can simplify the task by focusing on direct illumination only. Recall the reflection integral from equation 2.7:

$$\int_Y f_r(\omega, x, \omega_i) L_o(y, \omega_i) g(x, y) \mathrm{d}y$$

where Y is the set of all surface points in the scene. When considering the direct illumination of a point x, then we can reduce this set to E, which contains all emissive surface points. This subset is significantly smaller in most virtual scenes. L_o simplifies to L_e as we do not account for the indirect light so we do not evaluate the reflection integral recursively. We are left with

$$\int_E f_r(\omega, x, \omega_i) L_e(y, \omega_i) g(x, y) \mathrm{d}y$$

where $\omega_i = \frac{y-x}{\|y-x\|}$. So we can evaluate direct illumination of x with an MC estimator which samples random surface points y on emissive surfaces:

$$\frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\omega, x, \omega_i) L_e(y, \omega_i) g(x, y)}{p(y)}$$

Note that we have to evaluate the geometry term $g(x, y)$ as we have to check if the point on the light source is visible from the surface point x. This requires tracing a ray from the x to the point y on the light source. This technique is called light sampling or next event estimation (NEE).

In case our scene has many emissive surfaces, we can apply importance sampling for this estimator as well, common techniques use bounding volume hierarchies (BVH) over the emissive surfaces.[MPC19, MRNK21, HIT+23] Such a BVH consists of nested bounding boxes for emissive geometry and organizes them in a tree structure for efficient traversal. An emissive surface can be sampled by traversing the tree and randomly choosing a branch based on the weight assigned to this branch during construction. The PDF of the sample is stored implicitly in the weights of the tree and can be computed during traversal. In offline rendering such BVHs are pre-computed, however in realtime applications with dynamic scenes a static pre-computed BVH is not ideal. Moreau et al.[MPC19] propose a GPU-optimized algorithm for updating BVHs in realtime applications.

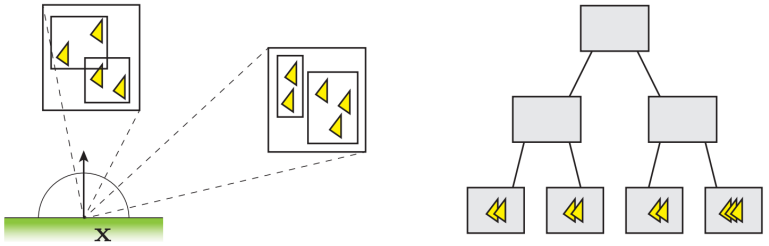


Figure 2.8.: 2D visualization of a bounding volume hierarchy. Light sources are stored in bounding volumes (left). A tree of bounding volumes is constructed (right) and can be traversed at a surface point x to sample a light. [MPC19]

ReSTIR

Bitterli et al.[BWP+20] propose a spatiotemporal reservoir resampling algorithm called ReSTIR to sample direct illumination for the first path vertex x_p. This vertex is directly visible from the camera and was sampled by tracing a path through the pixel p. For every pixel p a set of M samples $(y_{p,1}, .., y_{p,M})$ from emissive surfaces are selected. They are stored in a reservoir together with a weight for every sample. Only samples that are visible from the surface point x_p are added to the reservoir for pixel p. During rendering a sample is drawn from the reservoir based on the weight and the PDF can be computed as well. The power of this approach lies in (re)using the reservoirs from neighboring pixels and previous frames. This increases the number of samples with large contributions to p dramatically with little computational overhead. This comes at the cost of bias as the visibility for samples y' from neighboring reservoirs or from older reservoirs has not been checked during sample collection. To remove the bias the geometry term $g(x_p, y')$ has to be evaluated for every y' which was taken from an old or neighboring reservoir. Evaluating the geometry term requires tracing a ray, so removing the bias introduces significant computational cost. [BWP+20] It should be mentioned that this technique is limited to generating light samples for the first path vertex only. In contrast, NEE with a BVH can be used at any path vertex. So the ReSTIR algorithm can not be used to improve indirect illumination in this form.

Multiple Importance Sampling

As we discussed different importance sampling techniques above, we found that each has its benefits and limitations f.e. NEE does reduce noise for the estimate of direct illumination

significantly but performs poorly when rendering highly specular materials as it does not account for the BRDF.

This motivates the combination of multiple MC estimators. Veach [Vea98] introduces multiple importance sampling (MIS) to combine n sampling strategies with densities $p_i(x), i = 1, .., n$ by adding a weight $w_i(x)$ for each sampling strategy. Then we can compute the weighted sum of the estimators $f(x)/p_i(x)$:

$$\hat{I}_{MIS} = \sum_{i=1}^{n} \frac{1}{N_i} \sum_{j=1}^{N_i} w(x_{i,j}) \frac{f(x_{i,j})}{p_i(x_{i,j})}$$

Note that the weights are dependent on the sample. This way the contribution of each estimator can be adjusted depending on the sample. For the resulting MIS estimator $\hat{I}_{MIS}$ to be unbiased the weights need to be chosen such that

$$\sum_{i=1}^{n} w_i(x) = 1 \qquad \forall x : f(x) \neq 0$$
$$w_i(x) = 0 \qquad \forall x : p_i(x) = 0$$

A well known example of such weights is given by the power heuristic: [Vea98]

$$\hat{w}_i(x) = \frac{N_i \cdot p_i^{\beta}(x)}{\sum_{k=1}^{n} N_k \cdot p_k^{\beta}(x)}$$

In our tests, we use MIS with the power heuristic and $\beta = 2$.

2.3.4. Path Guiding

Previously mentioned techniques are very good at generating short transport paths with high contribution to the pixel color. However they do not generate complex transport paths for indirect illumination efficiently. To render f.e. caustics with a conventional uni-directional path tracer we have to rely on brute-force MC estimation with many samples. This is because neither NEE nor BRDF importance sampling is good at constructing such paths. The idea of path guiding is to iteratively learn to construct difficult paths. [MGN17]

We will discuss current path guiding techniques and their limitations. As our work is focused on real time rendering of dynamic scenes, we will focus on these aspects.

Practical Path Guiding

Müller et al. [MGN17] present a path guiding technique using SD-trees. They partition the 3D scene using a binary tree. For each leaf of this tree they construct a quad-tree to partition the 2D directional domain. Since the resulting tree structure partitions both the spatial and directional domain, they call it SD-tree. The advantage of such trees is their ability to adapt to the scene and provide high-resolution partitioning only where needed. They use a pre-computation step to fit this tree to the incident radiance field of the scene using path tracing. They start with BRDF importance sampling only to construct a first estimate L_i^1 of the incident radiance field. For the next iteration they rely on MIS to combine BRDF importance sampling and importance sampling of L_i^1. With this approach, they improve their estimate L_i^k iteratively. Müller et al. show that sampling from the previous estimate during training can accelerate the convergence of the estimate drastically when compared to naive MC estimation.[MGN17] Due to the pre-processing step for learning the radiance distribution, this technique is limited to the use in static scenes.

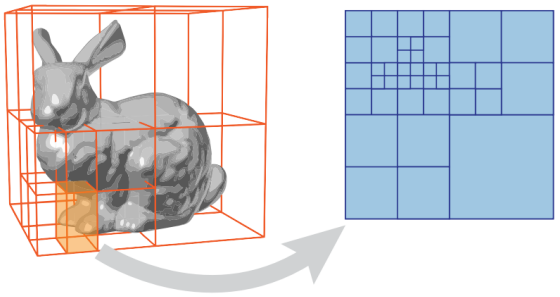


Figure 2.9.: Spatio-directional partitioning with a binary tree (left) for the spatial domain and a quadtree (right) for the directional domain. [MGN17]

While data structures such as SD-trees can be stored and traversed efficiently and in parallel on a GPU, their parallel construction is much more expensive. This is partially due to the high latency of memory access [HIT+23] on the GPU as well as the need for synchronization during construction. [OLK+21] Huang et al. [HIT+23] propose to use a voxel grid instead of the spatial binary tree. Only the directional octree for each voxel has to be constructed and updated in this case. Since this can be done in parallel, performance on a GPU is increased significantly.[HIT+23] This enables them to replace the pre-computation step with online training. However in their work they do not discuss the use of this technique in real time applications, so further research is required to adopt this approach for realtime applications and dynamic scenes. Additionally the resolution of this grid is limited as it has to cover the entire 3D space of the scene.

Product Importance Sampling

Herholz et al. [HEV+16] present a technique to combine BRDF importance sampling with path guiding. While we can use MIS to combine the contribution from paths created with either of the two techniques in a near ideal way, it does not help to decide which technique to use for creating the path in the first place. When sampling a scatter direction at a shading point we have to decide whether to sample according to the BRDF or according to the PDF we obtain from path guiding. As mentioned above, path guiding is used to learn an estimate of the incident radiance L_i in the first place and in many cases does not take the BRDF into account.

To get a combined PDF from the BRDF f_r and the incident radiance L_i, Herholz et al. propose to represent both using the same parameterized model:

$$p_r(\omega, x, \omega_i) \propto f_r(\omega, x, \omega_i)$$
$$p_L(x, \omega_i) \propto L_i(x, \omega_i) \cos\theta_i$$

This model has to chosen such that the product

$$p_\otimes = p_r(\omega, x, \omega_i) \otimes p_L(x, \omega_i)$$

can be calculated analytically.

In their work they used a Gaussian Mixture Model (GMM) for this. Fitting the GMM p_r to the BRDF can be done in a pre-computing step for every material in the scene. This is also possible in realtime applications if the material properties don't change.

For the guiding distribution p_L they used a pre-processing step to build an adaptive spatial cache. As we discussed before, such a pre-computation step is only possible for static scenes and thus not an option in many realtime applications.

During rendering the product $p_\otimes$ needs to calculated on the fly for each shading point, which can be used to sample an outgoing direction. Depending on the accuracy of the estimates p_r and p_L the the resulting product is a good approximation of the whole integrand $f_r(\omega, x, \omega_i)L_i(x, \omega_i)\cos\theta_i$.

Realtime Path Guiding based on PMM

Derevyannykh [Der21] proposed to use a parametric mixture model (PMM), which is composed of a 2D Gaussian distribution $\mathcal{N}$ and the BRDF distribution f_r

$$D(\omega_i) = \lambda \cdot \mathcal{N}(M^{-1}(\omega_i), \mu, \Sigma) + (1 - \lambda) \cdot f_r(\omega, x, \omega_i)$$

where μ, Σ are the mean vector and covariance matrix of the Gaussian distribution, λ is a mixing coefficient and M is an area-preserving mapping from 2D unit square to 3D hemisphere.

The BRDF is included into the mixture to support importance sampling on specular or glossy surfaces. The Gaussian distribution is used to importance sample $L_i(x, \omega_i)\cos\theta_i$. The mixing coefficient is learned as well.

The PMM parameters are stored in screen space, this allows for very fast access during rendering and training. This way a large or complex spatial data structure is avoided. In contrast to a spatial data structure, where multiple neighboring shading points might end up in the same partition and thus have to use the same guiding distribution, such a screen space approach stores a PMM for every shading point individually.

The parameters of the PMM are initialized such that λ is close to 0, this way in the beginning the algorithm relies on BRDF sampling only. Using expectation maximization (EM), the parameters are updated iteratively after every frame. The training data for this is collected during path tracing. For each pixel p a ray is traced from the camera through the pixel to find the visible surface x_p. Then the PMM for this pixel is used to sample a scatter direction $\omega_{s,p}$ using either the Gaussian or the BRDF part. The incident radiance $L_{i,p}(x_p, \omega_{s,p})$ is estimated with path tracing. For subsequent training of the PMM, the incident radiance $L_{i,p}$ as well as the next path vertex y_p, which was generated during the path tracing, are stored in a screen space texture. Based on this data EM can be used to update the PMM parameters. Since this is done in screen space the path vertices $y_{p'}$ from neighboring pixels can be used as well. The incident radiance contribution of $y_{p'}$ to x_p can be approximated similarly to ReSTIR [BWP+20] (see 2.3.3). However, since this approximation is not used for rendering but only for fitting the PMM, introducing bias here does not introduce bias to the final image, it only affects the quality of the PMM. So for performance reasons the author suggests to use the biased approximation for training without an additional visibility check.

An important aspect of screen space techniques like ReSTIR from Bitterli et al. [BWP+20] or Derevyannykhs Realtime PG with PMMs [Der21] is reprojecting the pixels as the camera moves. This is necessary as the per pixel information correlates with a surface point in the scene. If the camera moves or rotates, this particular surface point might be at a different position in screen space (or outside the screen entirely). In case of bad reprojection or no reprojection at all, a PMM is used which correlates with a (very) different surface point. In that case the importance sampling might be worse than naive MC and the training is slow too. [Der21]

This approach is designed for realtime applications and it is able to adapt to dynamic scenes in realtime. Movement of the camera can be compensated by reprojection, however this can lead to higher variance in some areas where the reprojection failed. Since the PMMs are stored in screen space for better performance and reuse in neighboring pixels this technique is limited to guiding 2-bounce lighting efficiently.

2.3.5. Radiance Caching

Müller et al. [MRNK21] proposed a neural radiance caching (NRC) technique for higher-order indirect illumination. They rely on a MLP to learn the scattered radiance, which is the solution of the reflection integral:

$$L_s(x,\omega) = \int_\Omega f_r(\omega, x, \omega_i) L_i(x, \omega_i) \mathrm{d}\omega_i$$

Conventional radiance caching techniques deal with problems similar to the ones we discussed in the path guiding section (see section 2.3.4). They also rely on a spatial data structure to store cache records and access them efficiently. Instead of a guiding distribution they store the scattered radiance after evaluating the reflection integral with path tracing. During rendering the paths are terminated early (usually after a few bounces) and the stored radiance from the cache is used to estimate the scattered radiance. This technique has two big disadvantages:

- **Bias**: most of the time there is no cache record for the shading point itself, so a nearby cache record is used. This introduces bias as we now converge to a wrong value no matter how many samples we use in the MC estimator.
- **Cache invalidation**: In dynamic scenes the cached radiance is invalid if something moves. Using an old guiding distribution in path guiding leads to high variance whereas using an old radiance record from the cache results in bias. It is expensive to determine which records are still valid because the original path that created them needs to be traced again.

Müller et al. [MRNK21] use a MLP to learn the scattered radiance and train it in online fashion. To do so they collect training samples during the path tracing step. Afterwards they optimize their cache. By using a very aggressive learning rate they minimize the bias due to cache invalidation. The MLP re-learns a decent approximation of the new radiance in less then 10 frames. Additionally their approach does not need a large data structure for the entire scene. Due to the fast learning of the cache it can adapt to previously unseen parts of the scene quickly.

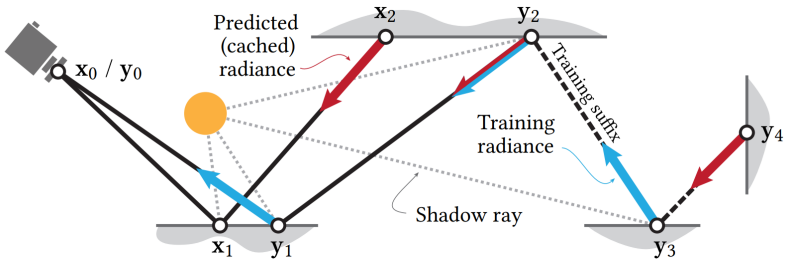


Figure 2.10.: Use and training of the neural radiance cache. Short paths (x_0, x_1, x_2) are terminated and the cache is used to estimate the scattered radiance at that point (red arrows). Some paths are extended to train the cache in online fashion.[MRNK21]

As shown in figure 2.10 the NRC is able to predict the scattered radiance for a surface point in world space and an outgoing direction ω. As the neural network prediction is not noisy, the resulting estimate of the scattered radiance is noise free as well. They use this to improve the quality of their training samples by terminating the training paths into the cache as well ((y_2, y_3, y_4) in figure 2.10).

2.4. Discussion

As shown in the previous sections there are numerous techniques to reduce noise in a path tracer. The topic has been in research for offline rendering for a long time now. Adapting these techniques to realtime applications is difficult for a number of reasons:

- data structures are not optimized for parallel construction on a GPU [MGN17, VHH+19, HEV+16]
- data structures are large to span the scene and need high resolution for optimal results [HIT+23]
- often pre-computation is necessary under the assumption a scene would be static [HEV+16]

Some attempts were made to overcome these challenges with screen space techniques. [Der21] Storing data per-pixel removes the large or complex data structures and enables high resolution since every shading point has its own guiding distribution and can reuse data from old frames or nearby pixels. However such techniques are limited as well. They can be used for guiding at the first path vertex only. Additionally they require a reprojection if the camera is moving which can cause artifacts and high variance.

An alternative approach to path guiding is radiance caching. Müller et al. [MRNK21] proposed a realtime technique using an online trained MLP. This approach shows the potential of neural networks to partition 3D space with a high resolution. They achieve this through fast adaption of the MLP to the scene. This approach works both to adapt to previously unseen parts of the scene as well as for dynamic geometry.

Based on these observations we want to motivate the use of MLPs for path guiding as radiance caching and path guiding share many problems related to partitioning the spatial domain.

3. Concurrent Work

3.1. Neural Path Guiding with NASG.

Before we derive our neural path guiding technique in detail we want to mention the research of Huang et al. [HIT+23] which was in work concurrently to ours. They proposed to importance sample the full integrand of the reflection integral. They use a mixture of normalized anisotropic spherical Gaussian distributions (NASG) to approximate the integrand

$$q(\omega_i, \gamma) \propto f_r(\omega, x, \omega_i) L_i(x, \omega_i) \cos\theta_i$$

where q is the NASG mixture with parameters γ. Instead of using a spatial data structure like an SD-tree [MGN17] or illumination cache [HEV+16] they rely on training a MLP to predict the NASG parameters. As described by Müller et al. [MRNK21] this approach relies one using relatively cheap computation instead of storing large data structures with expensive memory access for traversal.

Huang et al. use a MLP with 4 hidden layers and 128 neurons in each layer. They train the MLP in online fashion and combine the resulting estimator with NEE using multiple importance sampling. With this approach they avoid using pre-processing as NEE dominates in the beginning of the training and the MIS weights for their technique increase as the estimated NASG mixture approximates the integrand more closely.

They learn a mixing parameter c as well to decide whether to sample from the NASG mixture q or from the BRDF f_r during rendering:

$$\hat{q}(\omega_i, \gamma) = c \cdot q(\omega_i, \gamma) + (1 - c) \cdot f_r(\omega, x, \omega_i)$$

This is an important aspect for training, as it allows their MLP to ignore the NASG parameters in areas where the BRDF is going to be used anyway. Then the MLP can focus on areas where the incident radiance L_i dominates and approximate with better precision. To avoid falling in a local optimum with mixing parameter $c = 0$ everywhere they constructed a loss function based on KL-divergence (see 2.2.4)

$$\mathcal{L}(p(\omega_i), q(\omega_i, \gamma)) = e D_{\mathrm{KL}}(p(\omega_i) \parallel \hat{q}(\omega_i, \gamma)) + (1 - e) D_{\mathrm{KL}}(p(\omega_i) \parallel q(\omega_i, \gamma)) \tag{3.1}$$

with a fixed ratio $e = 0.2$. This loss is optimized for the use with MIS and thus their approach can be combined with NEE and BRDF sampling.

During training they rely on using both NEE and BRDF importance sampling to improve the quality of their training data. Note that the viewing direction ω is passed to the MLP since it has to account for the BRDF as well. This requires additional learning when the camera is moving even if the scene itself does not contain dynamic elements.

In contrast to our approach their work is not focused on realtime use. Since they published their results when our work was in the final stages we were not able to make an in-depth comparison.

4. Real Time Neural Path Guiding

4.1. Concept

As mentioned in 2.3.4, path guiding aims to reduce variance of the path tracer by learning to create complex transport paths. To do so, path guiding techniques iteratively learn an approximation of the 5D incident radiance $L_i(x, \omega_i)$. Since we are not accounting for the BRDF, which is the only component of the integrand that is dependent on the viewing direction ω, our approach is independent of the camera position. This is a big advantage for use in realtime applications, as the camera is highly dynamic in most scenarios.

For our approach, we will not make any assumptions about the scene and its dynamic, but instead rely on online learning. We will update our approximation of the incident radiance field continuously after each frame. This way, our approximation will improve over time and adapt to dynamic changes in the scene.

We use a MLP to predict the parameters of a probability distribution for path guiding. We collect the data for training when path tracing a frame. Based on this data, we train the MLP to improve the approximation of L_i. We then use the MLP for guiding the next frame. Due to this iterative learning, we hope to avoid pre-training the MLP or do any other pre-processing steps. We call this MLP our *guiding model*. An overview of our technique is shown in figure 4.1.

In the following sections we discuss the probability distribution we use for guiding as well as the architecture, training and collection of training data for the *guiding model*.

4.2. Guiding Distribution

As discussed in 2.2.5 the PDF used for importance sampling in our MC estimator should be proportional to the integrand of the integral we try to solve. In our approach, this is the scattering integral of the RTE. However, we limit ourselves to importance sample the incident radiance L_i, so our model will be independent of the viewing direction. Since we are integrating over the hemisphere, we need to normalize our integrand for this domain as well before approximating:

$$\frac{1}{\int_\Omega L_i(x, \omega_i) \mathrm{d}\omega_i} L_i(x, \omega) = \frac{1}{A(x)} L_i(x, \omega) =: f(x, \omega) \tag{4.1}$$

Here f is the guiding distribution that we want to approximate in order to importance sample incident radiance. $A(x)$ is the normalization factor for the radiance distribution. We discuss this factor in detail in section 4.5.

We use the vMF distribution $f_{\text{vMF}}(x, \omega)$ to approximate $f(x, \omega)$ at each surface point x. The vMF distribution can be evaluated efficiently and numerically stable. As described in section 2.2.3, we can generate random samples, which are distributed according to vMF. It is represented by two parameters $\mu(x)$ and $\kappa(x)$, which we can tune to approximate $f(x, \omega)$ as best as possible.

Having an explicit probability distribution for sampling the scatter direction is an important aspect as this allows us to evaluate the probability density function (PDF) for the scatter direction. This is a necessity for combining our neural path guiding with other techniques using multiple importance sampling (MIS). As explained in 2.3.2 the PDF is required to calculate the MIS weights for combining the contributions from different techniques. Other approaches, such as ReSTIR, require extra computation to obtain the (correct) PDF of the sample.

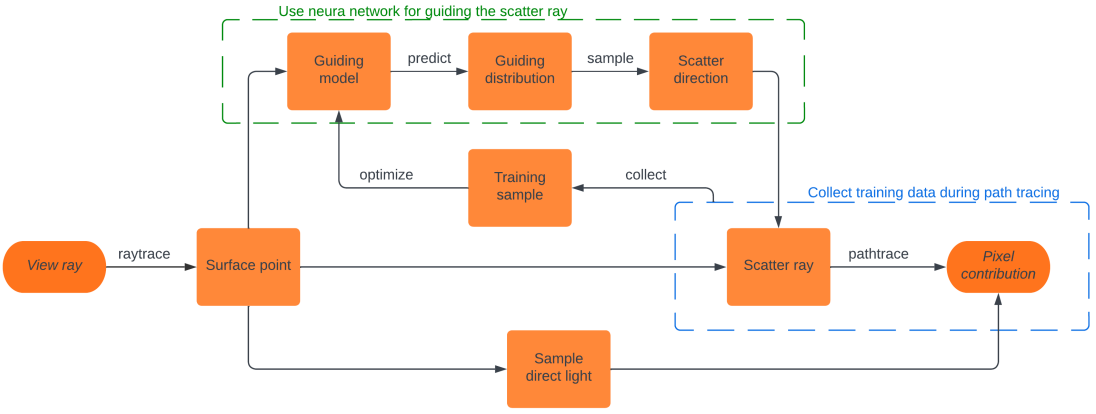


Figure 4.1.: Overview of our NPG approach for each frame. In the green part our guiding model is used for guiding the scatter-ray. The path is then finished using path tracing. Sampling the direct light is optional and can be included in the pixel contribution using MIS.

4.3. Guiding Model

4.3.1. Architecture

The *guiding model* is used to learn a guiding distribution for every world space position $x \in \mathbb{R}^3$. So the input to our guiding model is a 3-dimensional vector. As we use a vMF distribution for guiding, the output has to be a 4-dimensional vector (μ, κ) with the parameters $\mu \in [-1, 1]^3$, $\kappa \in [0, \infty)$.

In general, different types of neural networks or other models could be used to learn the parameters for the guiding distributions. In our case, we are focused on realtime use, so we rely on small and simple MLPs. In our tests, we used fully-fused MLPs because they are composed of a matrix multiplication and the application of an element-wise activation function. These operations can be executed very efficiently on current generation GPUs as they feature large amounts of stream processing units and dedicated tensor cores for operations like matrix multiplication.

We use 6 hidden layers with 64 neurons each. Additionally, the MLP consists of the input layer for the shading point and the output layer with the vMF parameters. These will be discussed in sections 4.3.2 and 4.3.3.

We tested Sigmoid and ReLU activations for the hidden layers. Since the ReLU activation is very simple, it is much more efficient to compute. In our experiments, Sigmoid converged much slower as well, so we use ReLU.

For training the MLP, we use the Adam optimizer as described in 2.1.2. We configure it as follows:

- learning rate $\alpha = 0.0001$
- exponential decay for the first moment estimate $\beta_1 = 0.9$
- exponential decay for the second moment estimate $\beta_2 = 0.99$

We use a constant learning rate and disabled both absolute and relative decay of the learning rate, because we do want our model to continuously learn and adapt to changes in the scene. The learning rate is chosen as high as possible for fast adaption.

4.3.2. Input

Normalizing the Position

As our approach aims to learn a guiding distribution for every surface point in the scene, the input to our *guiding model* is a 3-dimensional vector representing the world space coordinates of the surface point. We use the dimensions of the bounding box to transform the world space position to the range $[0, 1]^3$. For small test scenes such as CornellBox, this normalization helps to keep the weights of the neural network small as there is no large offset or scaling factor to learn. For larger scenes, this normalization does not work well and further research will be needed to improve this.

Additional Inputs

Using only the position as input to the MLP does not represent the radiance distribution well [MRNK21]. Thus we provide additional information, which correlates with the radiance distribution, to the MLP. In section 2.3 we discussed similar approaches. Müller et a. [MRNK21] used the surface normal, diffuse and specular reflectance and surface roughness as additional inputs to their radiance cache. The surface roughness and reflectance correlate with the BRDF, but not with the incident radiance at this point. Since we do not approximate the BRDF, we did not use these as additional inputs. We use the surface normal as a second input and evaluate the performance and cost.

Figure 4.2 visualizes the improvements from using the surface normal as an additional input. Due to the boxes being illuminated from two opposite directions, the radiance distribution can be fundamentally different on nearby surface points, such as on different sides of the left box (blue cutout). Without information about the surface normal, the *guiding model* fails to separate the spatial domain near the edge of the box. This results in bad guiding and thus very high variance. Near the edges, our guiding performed worse than plain path tracing.

When adding the surface normal as input, the radiance distribution along such an edge can easily be separated as the surface normals differ significantly. In our experiments, the MSE was reduced by order of magnitude when using the surface normal.

For areas and scenes with little to no change in the surface normal, f.e. the green wall (red cutout), the additional input does not help separate the spatial domain. In our

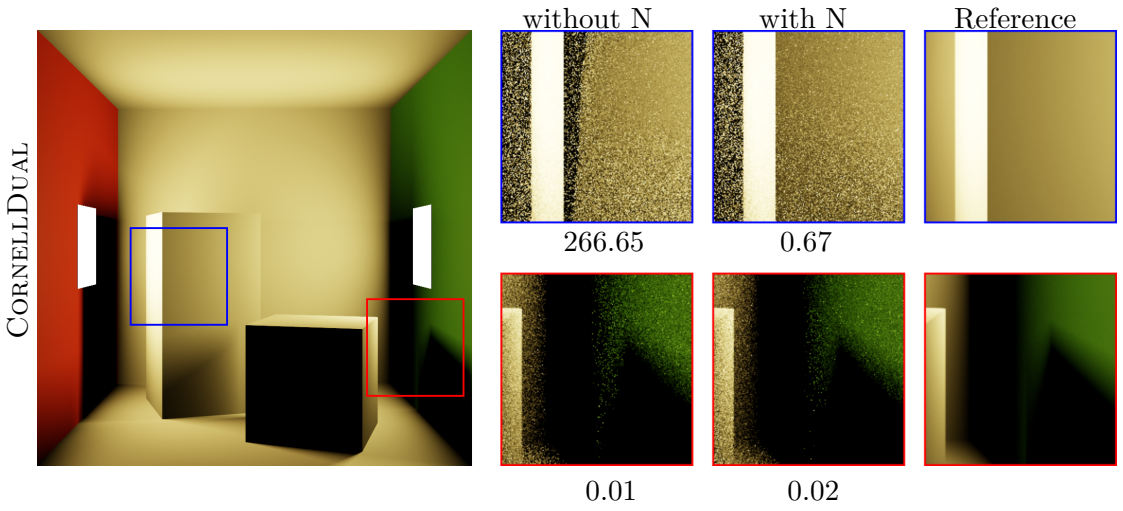


Figure 4.2.: Comparison of the rendered image when using position only or position and surface normal N as input to the *guiding model*. For both renders we used 16 spp and let the MLP converge to stable state. We report the MSE for each cutout individually.

experiments, it did not cause higher variance or error either. Since adding the additional input only affects the first layer of the MLP, this does not change computational cost much. However, the required memory bandwidth doubles as we have to send both a 3-dimensional vector for position and surface normal to the neural network. On average, this resulted in 4.4 FPS less in our tests.

As shown in 4.2, the added surface normal reduces the MSE significantly in scenes with multiple light sources. We also evaluated the difference for scenes with only one light source, for this we chose the CornellBox and CornellIndirect scenes. To our surprise the MSE on these scenes was higher when using the surface normal as input in all our experiments. This could be an instance of overfitting due to the noisy samples, where the MLP tries to find a correlation between surface normal and radiance distribution, although in these scenes, the surface normal does not provide additional information.

Since the computational cost differs by few a frames and we see significant improvements for scenes with more than one light source, we do recommend using the surface normal as an input for the *guiding model* in general use cases.

Scene	Input	FPS	MSE
CornellBox	position	109	0.009
	position + normal	102	0.013
CornellDual	position	105	5.78
	position + normal	101	0.14
CornellIndirect	position	80	2.28
	position + normal	78	2.41
Average	position	98	2.69
	position + normal	93.6	0.85

Table 4.1.: Effect of using the surface normal as input to the *guiding model* on framerate (FPS) and mean squared error (MSE). Our test system uses an AMD Ryzen 5 7600X processor, 32 GB RAM and a RTX 3070. We performed each test 3 times and reported the average FPS and MSE.

4.3.3. Output

The activation function in the output layer often differs from the activations used in hidden layers since the output of this activation will be the final prediction of the neural network. This can be used to map each output to a desired range or normalize the output vector, f.e. to use as a probability distribution. In case of our *guiding model*, the outputs are the parameters of the vMF distribution. The 4-dimensional output vector (μ, κ) consists of $\mu \in [-1, 1]^3$ and $\kappa \in [0, \infty)$.

Output Activation for Mean Direction

Since $\mu = (\mu_1, \mu_2, \mu_3)$ will be used as the mean direction of our vMF distribution, we use a translated Sigmoid activation to map each component to the desired range individually:

$$\varphi(\mu_i) = \frac{2}{1 + e^{-\mu_i}} - 1$$

Note that the resulting vector μ is not normalized. We have to normalize it before sampling later on.

Output Activation for Concentration

For the concentration parameter κ, we have to ensure it is non-negative. Furthermore, the range of this value is very large. For $\kappa \to 0$ our vMF approaches a uniform distribution, which will be needed in evenly lit areas with no dominant direction of incident radiance. For precise guiding, f.e. towards a small light source, it is important to support values much larger than 100 to concentrate most of the probability mass at the mean vector. We tried to use a quadratic mapping for κ, this ensures non-negativity. However, our MLP struggled to predict large values. In our experiments, we found an exponential activation increases the range of κ significantly and also ensures non-negativity.

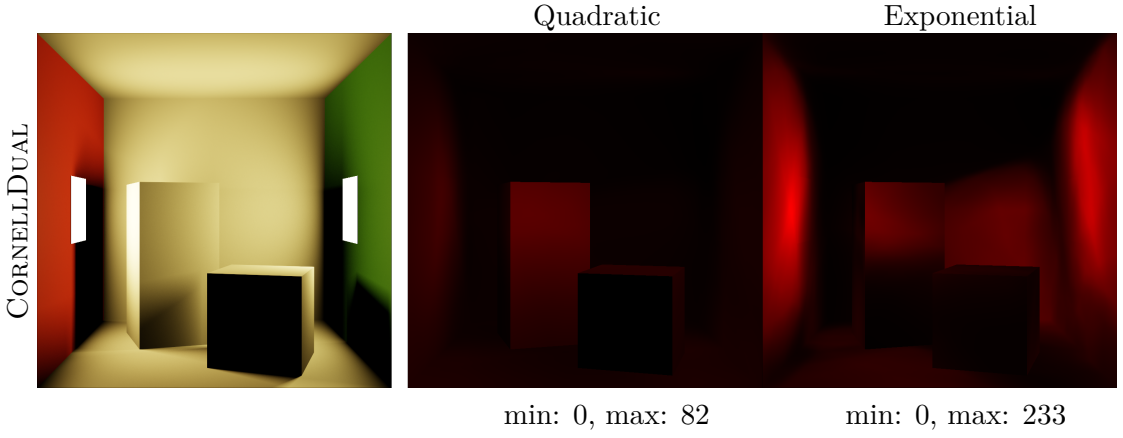


Figure 4.3.: Predicted concentration parameter κ visualized with quadratic and exponential activation function after 20 seconds of training. Only direct illumination is computed.

As shown in figure 4.3 using an exponential activation for κ results in a much larger output range. For visualization and testing, we modified the CornellBox scene. Instead of illuminating the room with one area light at the ceiling, we placed two light sources on opposite walls. Thus we call this scene CornellDual. With direct illumination only, this scene has areas that are lit by one light source only, f.e. the left wall is lit by the

right light source only. These areas are a perfect case for our vMF distribution because most incident radiance is coming from one major direction so we fit our mean vector to this direction and use a large κ to concentrate the distribution in this direction. This can be seen on the left and right wall with both activation functions. However with the exponential activation the network is able to output much larger values which results in lower noise when rendering.

The ceiling and parts of the rear wall, on the other hand, are illuminated by both area lights. Since these are placed in opposite directions, the vMF distribution is a bad choice to represent this radiance distribution. The best fit requires a small value for κ, which results in a near uniform distribution. This way, none of the two lights is neglected. For our network, it is important to be able to predict very small values for κ as well. Throughout all our test scenes, both the quadratic and the exponential activation were able to output values close to 0.

4.4. Training the Neural Network

4.4.1. vMF Loss

The *guiding model* is supposed to learn parameters for a vMF distribution $f_{\text{vMF}}(\omega; \mu, \kappa)$. This distribution should approximate the incident radiance distribution $L(x, \omega)$, which is dependent on the spatial domain. Our parameters $\mu(x)$ and $\kappa(x)$ are thus dependent on the spatial domain as well. As discussed previously, the position x in world space coordinates is the input to the *guiding model* and the parameters $\mu(x)$ and $\kappa(x)$ are the output. To train the MLP, we will sample random positions x and fit our vMF distribution to the incident radiance distribution at this point. By repeating this training for many random points, the MLP can learn general correlations between world space positions and vMF parameters. This allows the MLP to predict vMF parameters for positions it has not seen during training.

As discussed in 2.1.2, we need a loss function to compare the MLP prediction with the desired output. In our case, the outputs are the vMF parameters. However, the target is the incident radiance distribution which we cannot evaluate in closed form. We thus have to compare the vMF distribution with the incident radiance distribution and obtain a distance between the two. We use KL-divergence for this. As discussed in 2.2.4, KL-divergence is designed to measure the difference between two probability distributions. By design our vMF distribution is a probability distribution (2.2.3). However the incident radiance distribution is not as it is not normalized. We normalize it using a normalization constant $A(x) = \int_\Omega L(x, \omega_i) \mathrm{d}\omega_i$. This way we obtain the normalized radiance distribution

$$f_L(x, \omega) = \frac{1}{A(x)} \cdot L(x, \omega)$$

As we can verify the radiance distribution is indeed normalized by integrating over the hemisphere Ω:

$$\int_\Omega f_L(x, \omega) \mathrm{d}\omega = \frac{1}{A(x)} \int_\Omega L(x, \omega) \mathrm{d}\omega = \frac{1}{\int_\Omega L(x, \omega_i) \mathrm{d}\omega_i} \int_\Omega L(x, \omega) \mathrm{d}\omega = 1$$

We have to ensure that $A(x) > 0$, but this is a reasonable condition as guiding at positions with no incident radiance at all is infeasible. We thus add a small ϵ to our normalization constant $A(x)$ so we avoid numerical issues in dark areas.

For better reading, we will simplify the notation for the vMF and normalized radiance distributions by making the spatial dependencies implicit:

$$\begin{aligned} f_{\text{vMF}}(\omega) &:= f_{\text{vMF}}(\omega; \mu(x), \kappa(x)) \\ f_L(\omega) &:= f_L(x, \omega) \\ A &:= A(x) \end{aligned}$$

Kullback-Leibler divergence

We can now use KL-divergence to obtain a distance between our predicted vMF distribution f_{vMF} and our target radiance distribution f_L:

$$D_{\text{KL}}(f_L \parallel f_{\text{vMF}}) = \int_\Omega f_L(\omega) \cdot \log\left(\frac{f_L(\omega)}{f_{\text{vMF}}(\omega)}\right) \mathrm{d}\omega = \int_\Omega f_L(\omega) \cdot \log\left(\frac{L(x,\omega)}{f_{\text{vMF}}(\omega) \cdot A}\right) \mathrm{d}\omega$$

To solve this integral in an efficient way, we use MC integration (2.2.5): we approximate the integral with a sum of randomly drawn samples. In our case we use just one sample ω_s and compute the incident radiance $L(x, \omega_s)$ for this sample to obtain our training target:

$$\begin{aligned} D_{\text{KL}}(f_L(\omega_s) \parallel f_{\text{vMF}}(\omega_s)) &= \frac{f_L(\omega_s)}{p(\omega_s)} \cdot \log\left(\frac{L(x,\omega_s)}{f_{\text{vMF}}(\omega_s) \cdot A}\right) \\ &= \frac{f_L(\omega_s)}{p(\omega_s)} \cdot (\log(\underbrace{L(x,\omega_s)}_{\text{target}}) - \log(\underbrace{f_{\text{vMF}}(\omega_s) \cdot A}_{\text{prediction}})) \end{aligned}$$

Here $p(\omega_s)$ is the probability density of the distribution used to sample ω_s in the first place. We discuss the process of generating training data in section 4.6.

To sum this up, we need to select a random position x in world space for training. We then sample a random scatter direction ω_s and evaluate $L(x, \omega_s)$ to obtain the target. We pass the position x to the MLP and predict the parameters μ and κ for the vMF distribution. To compute the distance between our predicted vMF and the target distribution, we use KL-divergence. To evaluate this, we need to pass the scatter direction ω_s, the radiance estimate $L(x, \omega_s)$, the normalization factor A and the probability density $p(\omega_s)$ to the MLP.

Relative loss

As discussed before, MC integration with few samples results in high variance. Since we are approximating the incident radiance for a single direction ω_s, our training target $L(x, \omega_s)$ is very noisy. For training a MLP based on such training data a robust loss function to compensate for the noise is needed. Using KL-divergence as is for computing the training loss and gradients did not work as the MLP was not able to learn from the noisy targets. Huang et al. [HIT+23] encountered similar issues when using the KL-divergence as loss function directly. We decided to use relative $\mathcal{L}^2$ loss to train the MLP as it provides unbiased gradient estimates for noisy training data [MRNK21]. For scaling the loss we use the network prediction instead of the sampled radiance distribution because it is less noisy.

$$\mathcal{L}^2_{\text{vMF}}(f_{\text{vMF}}(\omega_s; \mu, \kappa), f_L(\omega_s)) = \frac{D_{\text{KL}}(f_L(\omega_s) \parallel f_{\text{vMF}}(\omega_s; \mu, \kappa))^2}{\text{sg}(f_{\text{vMF}}(\omega_s; \mu, \kappa) \cdot A)^2 + \epsilon}$$

The operator $\text{sg}(\cdot)$ denotes this argument should be treated as a constant when optimizing the loss. This is necessary as otherwise the MLP would learn to maximize $f_{\text{vMF}}(\omega_s; \mu, \kappa)$ which means learning the probability distribution $p(\omega_s)$ we used for generating our training data. We add a small ϵ for numerical stability. In our tests, we use $\epsilon = 0.02$.

Training targets with little radiance

In general our training targets $L(x, \omega_s)$ are in range $[0, \infty)$, with large values being the most interesting ones as we want to guide our paths in directions with high radiance. However our training targets can also transport little radiance, f.e. in dark areas. Thus the target value $L(x, \omega_s)$ can be very small. When computing KL-divergence for small target values we end up with a large negative loss. The reason for this issue is the logarithm present in KL-divergence, because

$$\log(L(x, \omega_s)) \to -\infty \qquad \text{if} \quad L(x, \omega_s) \to 0$$

This issue affects both the training target $L(x, \omega)$ and the predicted vMF, however due to the noise in the training target they do not cancel out. We tried adding a small ϵ to the target and prediction. However this did not resolve the issue. We thus removed the logarithm for prediction and target from our final loss function.

Training targets with high radiance

Another source for noise in our loss is the multiplication with the normalized radiance distribution $f_L(\omega_s)$ in the KL-divergence. As this value can be very large if a path with low probability reaches a bright light source, such samples (fireflies) dominate the training and lead to stability issues. We compensated this by adding $f_L(\omega_s)$ to the scaling factor of our loss function. This way $f_L(\omega_s)$ cancels out and we end up with the following vMF loss:

$$\mathcal{L}^2_{\text{vMF}}(f_{\text{vMF}}(\omega_s; \mu, \kappa), f_L(\omega_s)) = \frac{(L(x, \omega_s) - f_{\text{vMF}}(\omega_s; \mu, \kappa) \cdot A)^2}{p(\omega_s)^2 \cdot \text{sg}(f_{\text{vMF}}(\omega_s; \mu, \kappa) \cdot A)^2 + \epsilon} \tag{4.2}$$

4.4.2. Regularization

We found regularization to be a useful tool to counter overflowing weights and other issues related to noisy training data. L2 regularization adds the sum of the squared MLP weights to the loss:

$$\mathcal{L}^2_{\text{regularized}} = \mathcal{L}^2_{\text{vMF}} + \lambda \cdot \sum w_i^2$$

So for every weight w_i of the MLP, the gradient will contain $2\lambda w_i$ and thus the weights are minimized as well. The parameter λ affects the scale of the regularization with no regularization for $\lambda = 0$. Choosing $\lambda > 0$ results in lower values for the weights and avoids numerical problems. This comes at the cost of underfitting, because at some point the contribution of the regularization to the gradient overpowers the contribution from the vMF-loss. We choose $\lambda = 0.2$.

4.4.3. Temporal Stability

The noise within our training data as well as the random distribution of training samples causes visual artifacts such as flickering in the output. This is most notable when the *giuding model* has learned to predict large values for the concentration parameter κ. One technique to reduce such temporal artifacts used by [MRNK21] is to apply an exponential moving average (EMA) to the network weights:

$$\overline{W}_t := \frac{1-\alpha}{1-\alpha^t} \cdot W_t + \alpha \cdot (1-\alpha^{t-1}) \cdot \overline{W}_{t-1}$$

Here W_t is the set of weights produced by the t-th training step and for evaluating the MLP we create a second set of weights $\overline{W}_t$. The exponentially reducing contribution of older weights dampens the temporal artifacts. High frequency flickering is compensated and only low frequency artifacts remain. The α parameter controls the exponential decay. We use $\alpha = 0.9$ in our experiments. It is important to note that these moving average weights $\overline{W}$ are used for the prediction step only, during training the MLP uses the learned weights W. This way, the visible output of the MLP is improved without affecting the training at all.

4.5. Normalized Training Targets

4.5.1. Learning the Normalization Factor

As seen above, for training the guiding model we need to normalize our training sample $L(x, \omega)$ to obtain a PDF. The normalization factor is $A(x) = \int_\Omega L(x, \omega_i) \mathrm{d}\omega_i$. This, however requires a solution of the RTE itself, so its what we want to compute in the first place. Without normalization of the training samples, the MLP does focus on learning the brightest areas as these have a much higher contribution to the loss. To counter this and learn guiding distributions for darker areas as well, we need an approximation of A. We use our MLP-approach to learn this normalization factor $A(x)$.

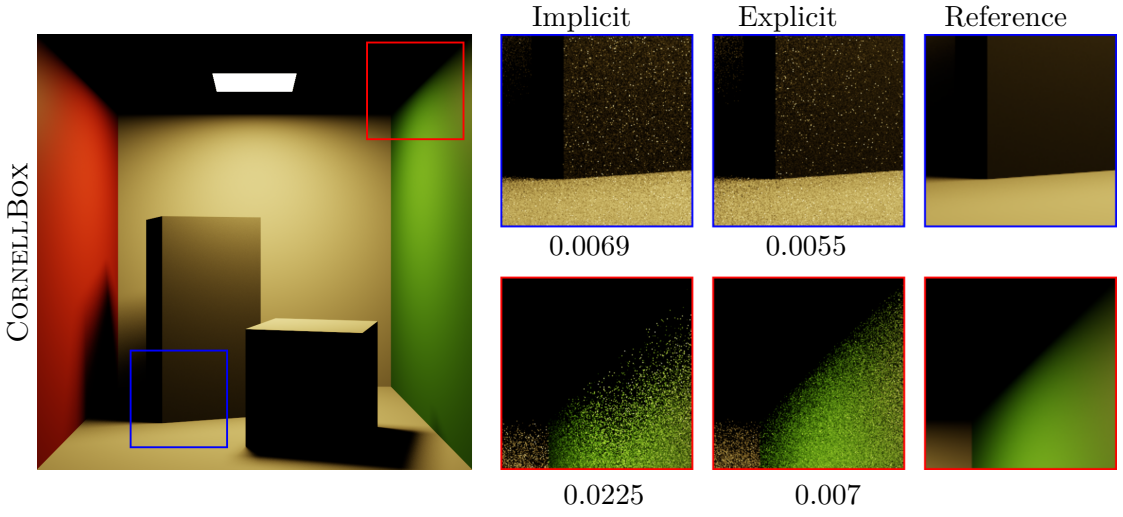


Figure 4.4.: Comparison of learning the normalization factor implicitly in the *guiding model* or using an explicit *normalization model*. For both renders we used 16 spp and let the MLPs converge to stable state. We report the MSE for each cutout individually. It should be noted that our explicit implementation was around 20% slower than the implicit approach.

Implicit

We tried including this task in the *guiding model* so it predicts the normalization factor implicitly with the vMF parameters. This is a simple addition as we only add one more output to the guiding model and derive the vMF loss with respect to the normalization factor A for it to be learned. The computational overhead is minimal as this change only affects the output layer.

One downside of this approach is the tight connection between the normalization factor and guiding distribution in one MLP. If the predicted normalization factor is poorly approximated, then the target for learning the guiding distribution suffers from this approximation error as well. This can lead to bad training for the guiding distributions as well as stability issues.

Explicit

Our vMF loss is designed for robust learning of the vMF parameters, but it is not specialized for learning the normalization factor. This leads to non-optimal learning of this factor. We propose to use a second independent MLP to learn the normalization factor. This way, we can choose a loss function independent from the vMF loss and tune the MLP specifically for this task.

This additional *normalization model* comes at significant cost as we need to predict the normalization factor for each training sample before training the *guiding model*. Furthermore, we need to train the *normalization model* as well.

As shown in figure 4.4, this additional computational effort can improve the learned guiding distributions significantly. As mentioned before, the effect of normalization is most notable in darker areas. As we can see in the red cutout, the implicitly learned normalization factor does not represent the incident radiance in that area well. This results in a bad guiding distribution as the training target for the *guiding model* is not normalized correctly.

In the following sections, we discuss the details of the *normalization model* used to create the results shown in figure 4.4.

4.5.2. Explicit Normalization Model

Architecture

In contrast to the implicit approach, with an extra model for learning the normalization factor, we have the ability to choose a different architecture for the normalization model. For performance reasons we use only 4 hidden layers in this model. We use 64 neurons per layer and ReLU activation as well.

We decided to stick with the same input scheme we use for the *guiding model*, so we input the world space position and the shading point to the MLP.

Similar to the concentration parameter of the vMF which is an output of the *guiding model* (see section 4.3.3) the normalization factor can span a large range of values. We compared using a linear and an exponential activation function. Again the exponential was able to produce higher output values while still being able to output values close to zero. Additionally this output activation does not produce negative values which is very important for the calculation of the vMF loss.

Loss

Similar to the work of Müller et al. [MRNK21] we used a relative $\mathcal{L}^2$ loss which works well for learning a scalar value even if the target is noisy:

$$\mathcal{L}^2(A(x), L(x, \omega_s)) = \frac{(A(x) - L(x, \omega_s))^2}{p(\omega_s)^2 \cdot A(x)^2}$$

where $p(\omega_s)$ is the density of the distribution our scatter direction ω_s was sampled from.

Optimizer

For training the MLP we use the Adam optimizer as described in 2.1.2. We configured it as follows:

- learning rate $\alpha = 0.001$
- exponential decay for the first moment estimate $\beta_1 = 0.9$
- exponential decay for the second moment estimate $\beta_2 = 0.999$

We do apply the exponential moving average (see section 4.4.3) for this model as well. In our experiments, regularization was not needed to generalize well. This might be due to the smaller network architecture.

4.6. Training Data

The core idea of our approach is to learn a guiding distribution during rendering. This is the task of our *guiding model*. To do so the MLP requires data to learn from. In our case we do not use pre-computation for generating this data but instead collect it during rendering of the frame. This way the computational overhead is limited to the actual training and evaluation of the MLP without the need for computing additional training data.

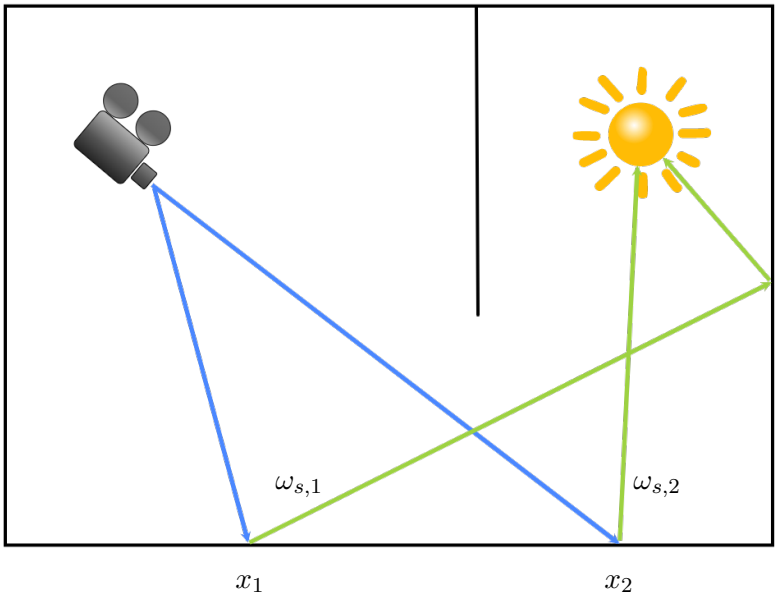


Figure 4.5.: Two possible paths generated during path tracing. The green part of each path can be used for training. x_i is surface point and will be input to the *guiding model*. $\omega_{s,i}$ is a random scatter direction at this surface point. We use path tracing to estimate the incident radiance $L(x_i, omega_{s_i})$ for the particular surface point and scatter direction. Our MLP can learn both direct illumination (path 2) and indirect illumination (path 1).

4.6.1. Input

In our experiments, we used the *guiding model* to predict distributions for directly visible surfaces only. As these points are all contained within the view frustum of the camera, we focus our training on this area as well.

We uniformly select pixels in screen space to be used for training. We start by tracing a ray through each pixel to find the visible surface point related to the pixel. As discussed in section 4.1, we can use this surface point as input to our *guiding model* and obtain a guiding distribution. We need to evaluate the MLP for training as well, so we store the world space coordinates of the surface point $x \in \mathbb{R}^3$ to use as training input. We also store the surface normal $n \in \mathcal{S}^2$ since it is used as additional input parameter to the MLP.

4.6.2. Target for the Guiding Model

We also need a training target for each training input. As discussed in section 4.4.1 our vMF loss requires the scatter direction $\omega_s \in \mathcal{S}^2$, the incident radiance estimate $L(x, \omega_s) \in [0, \infty)$ for the scatter direction, the normalization factor $A(x) \in [0, \infty)$ and the probability density $p(\omega_s) \in (0, \infty]$ of our scatter direction.

To obtain the normalization factor, we use our *normalization model*. The scatter direction ω_s can be selected randomly. It is, however, important to know the probability density $p(\omega_s)$. For now we rely on uniform sampling of the hemisphere, but other distributions for the scatter direction will be discussed in section 4.6.5. The last part missing for our training target is the incident radiance $L(x, \omega_s)$ for our scatter direction. As the incident radiance is needed for computing the pixel contribution of this path, it is computed during the rendering process anyways, so no extra effort is needed. As discussed before the incident radiance obtained from path tracing is a noisy approximation. After the frame is rendered we have all components needed for training the *guiding model* with the vMF loss.

4.6.3. Target for the Explicit Normalization Model

As the normalization model has to learn the normalization factor, training data is required for it as well. As it relies on the same input, we can reuse the training input from the *guiding model*. Additionally, an estimate of the incident radiance $L(x, \omega_s)$ at this shading point and the PDF $p(\omega_s)$ of the scatter direction ω_s is needed. We can just reuse the data from the *guiding model* for this as well.

4.6.4. Batch Size

As mentioned before, we uniformly select pixels in screen space to collect training data from. We use 6% of the pixels in screen space for this, so on a 1080p Monitor this results in about 124000 training samples per frame. However, this number can be significantly lower because not every pixel correlates with a shading point, f.e. if the environment map is visible at a pixel. We do not resample in that case but instead use fewer training samples.

We split the data into 4 batches for efficient training on the GPU.

4.6.5. Guided Training

As mentioned before noisy training data can cause many problems such as overfitting, bad learning and instability of the output. As we are computing the training targets in online fashion right before training, we have some control over the quality of the targets. As mentioned above, the scatter direction ω_s of the training sample is selected randomly and we can choose its probability distribution. We can use the *guiding model* to guide the scatter direction of the training sample as well. As we do use the guiding distribution for rendering anyway, no extra computation is required. We use the guiding distribution to sample our scatter direction for training. This is a common approach in many path guiding techniques (see 2.3.4).

However, this approach has some disadvantages as well: Our training is focused on what we have learned already, so most of our scatter rays are guided in directions of high incident

radiance. Other directions are very unlikely to be explored. This can be a problem if our approximation of the radiance distribution is bad, f.e. because we did not discover another direction of high incident radiance jet. When using our guiding distribution for training as well, it is less likely we will ever discover such an important part of the radiance distribution. This is especially important as we do not make any assumptions about the dynamics of the scene, so the radiance distribution can change at any time, f.e. if a light source is moved.

To make use of the improved training targets obtained from our guiding distribution and still be able to discover new directions and changes in the radiance distribution, we use a mixture of both by randomly choosing whether to use uniform hemispherical distribution or our guiding distribution. We control the ratio of both with the parameter $\tau \in [0, 1]$. In our implementation, this parameter is called exploration-ratio and for $\tau = 1$ all scatter directions for training are drawn from the guiding distribution, while for $\tau = 0$ all are drawn uniformly. We used $\tau = 0.5$.

4.7. Indirect Illumination

When using the *guiding model* to learn direct illumination, we are only interested in sampling directions that lead to the light source. However, this is only a part of the light transport, f.e. shadows are completely black and caustics are missing. Our approach is not limited to guiding the direct illumination, so we can train our *guiding model* for indirect illumination as well. This is especially important as techniques like NEE or ReSTIR are limited to direct illumination.

All information that is present in our training data can be learned by our MLP, including directions of high indirect radiance. However, most of the time, direct illumination contributes much more to the pixel color. This is because at every surface point on our path we multiply the radiance with the BRDF of this surface. A realistic BRDF has to obey physical laws such as energy conservation: the outgoing radiance cannot be larger than the incident radiance. Along the path, parts of the energy emitted by the light source are scattered away or absorbed by the surfaces. For our training, this results in a problem as the direct samples dominate due to their higher contribution. On directly lit surfaces the MLP does approximate the incident radiance from direct illumination without accounting for the indirect contributions.

Even in scenes with only a few directly illuminated surfaces, these surfaces dominate the training. Our *normalization model* and the relative L2 loss are intended to normalize the training samples and loss to eliminate this problem. However, this normalization factor is approximated as well.

We added an option to filter the training data while it is collected in the tracing step. If the scatter ray for a training sample hits an emissive surface right away it will not be used for training. This way only paths without direct light contribution to the shading point remain in the training set. Since we are using the same training data for both the *guiding model* and normalization model, no additional changes are needed to focus the guiding on indirect illumination only.

Now the direct illumination is ignored by our MLP. We need to use some other technique like NEE or ReSTIR to sample the direct light. Then we combine this contribution with the indirect contribution we obtain from our path guiding using MIS.

4.8. Product Importance Sampling

Our approach does not account for the BRDF at all. If we draw all samples according to our guiding distribution, the BRDF will not be importance sampled at all. This results in high variance when sampling specular or glossy surfaces such as glass, metal or paint.

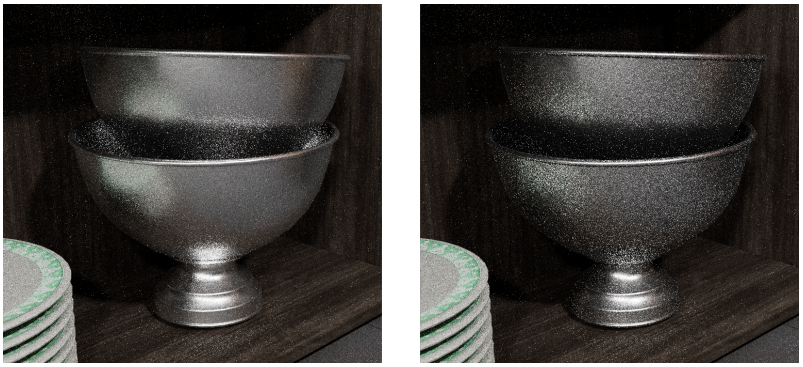


Figure 4.6.: A metal surface rendered with (left) and without (right) BRDF importance sampling.

We propose to use product importance sampling as described in section 2.3.4. In contrast to the work of Herholz et al.[HEV+16] we use the vMF distribution instead of Gaussian mixtures. As stated by Jacob [Jak12] the product of two vMFs can be closely approximated by another vMF. This can be done analytically, so we can use product importance sampling to obtain a guiding distribution that is much closer to the integrand of the RTE. To do so we have to fit a vMF distribution to each material in the scene. If the material does not change we can rely on pre-computation for this. The parameters μ and κ for the vMF can be estimated using the expectation maximization (EM) algorithm. This approach can be extended to mixtures of vMF lobes if necessary.[HG14]

5. Implementation

We implemented our Neural Path Guiding (NPG) technique in a realtime path tracing framework. For this we used Nvidia Falcor [KCK+22]. Our implementation is a modification of their Megakernel-Pathtracer to which we added the sampling from our guiding distribution as well as the collection of the training data.

For implementing, training and evaluating the MLP we rely on Müller's Tiny CUDA Neural Networks implementation [M̈21] which is used in the Neural Radiance Cache [MRNK21] as well.

Due to the time constraint we did not implement the product importance sampling from section 4.8.

5.1. Falcor

For our implementation, we used Falcor 4.4, which contains both the source code for the Megakernel-Pathtracer and additional render passes for evaluating MSE and FLIP in realtime. We did use the CornellBox scene, which is part of the framework as a basis and modified it to our needs.

This version of Falcor (specifically their implementation of the Megakernel-Pathtracer) contains a bug related to dielectric surfaces such as glass. Such surfaces are rendered black. We were not able to test our technique with transmissive surfaces because of this bug.

The latest version Falcor 5.2 contains an implementation of the ReSTIR algorithm (see section 2.3.3) called RTXDI, which we used for comparison.

5.2. Tiny CUDA Neural Networks

The Tiny CUDA Neural Networks framework contains a flexible, high-performance implementation for a MLP which fits in shared memory of the CUDA streaming processors. This way, no access to the global GPU memory is needed when evaluating the MLP other than reading the network input and writing the network output back. [MRNK21] This implementation relies on modern hardware, we used an RTX 3070 GPU for this.

6. Results

6.1. Limitations of the Guiding Distribution

As we discussed in section 2.2.3, the vMF distribution is an exponential class distribution on the unit sphere with a mean direction and concentration in this direction. We use it to approximate the incident radiance distribution at a shading point. This approximation is reasonably well if the incident radiance at the shading point is concentrated in one direction, f.e. if there is only one direct light source illuminating the shading point and no strong indirect illumination such as caustics. It can also fit uniform incident light from any direction, because for small concentration values the vMF approaches the uniform distribution. However in that case no guiding is needed anyways, as sampling of the BRDF in that case will result in the best importance sampling.

However, the vMF is limited to concentration in only one direction. In case the incident radiance is concentrated in multiple different directions this results in a bad fit. As shown in figure 6.1 we end up with a near uniform distribution which means almost no guiding.

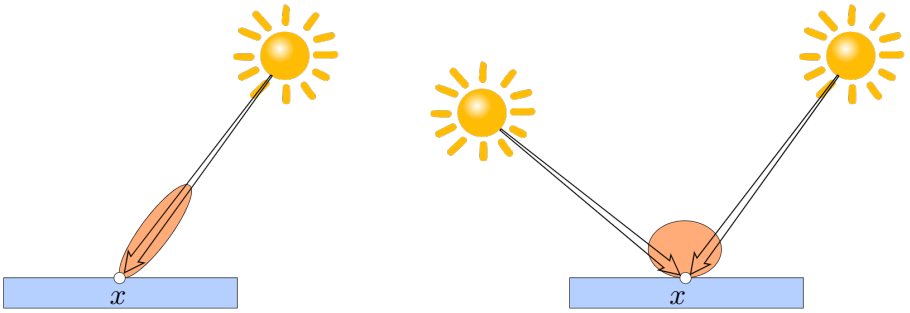


Figure 6.1.: The vMF distribution can approximate the incident radiance distribution at a shading point x well if most radiance is concentrated in one direction (left) or if the shading point is illuminated evenly from all directions (in that case no guiding is needed). However if the incident radiance is concentrated in multiple (f.e. 2) directions the vMF is a bad fit as the best it can do is a nearly uniform distribution (right).

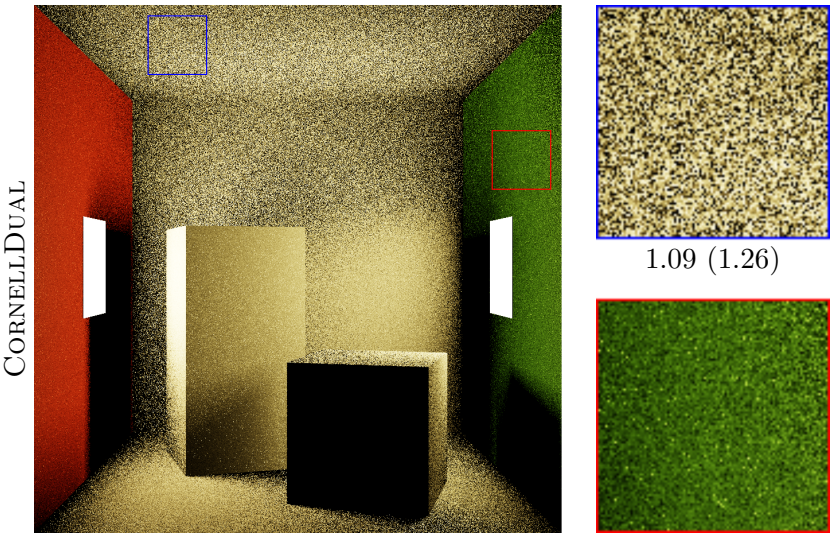


Figure 6.2.: Render of the CornellDual scene with two area lights in opposite directions. On surfaces which are illuminated by both light sources (blue) variance of the MC estimator remains high. The red box shows a surface that is illuminated by one light source only so the predicted vMF can approximate the incident radiance well. The image is rendered with 16 spp.

6.2. Computational Cost

Our Neural Path Guiding approach is intended for realtime applications. Hence we have strict time constraints for predicting the guiding distribution. To render 60 frames per second, we have a total of 16.6 ms per frame. In this time, we have to evaluate the *guiding model* at least once per pixel for guiding at the first visible surface. It is possible to use our model at the second surface bounce and any later one as well, since the model is not limited to screen space. Due to the computational cost, we did not attempt this and limited our research to guiding on the surface directly visible from the camera. We have to keep in mind that we do not only query the model for every pixel, but also perform the path tracing and shading, which includes evaluating the BRDF for every surface along the path. Furthermore we train our model for every frame. This is necessary to adapt to changes in the scene.

To evaluate the computational cost we measured the time for rendering a frame on different scenes:

- CornellBox: simple test scene with one area light source and few geometry, this yields minimal cost for the path tracer. We compute 2 surface bounces.
- VeachBidir: simple geometry with two light sources, different materials such as glass and metal. We compute 2 surface bounces. [Bit16]
- BistroInterior: complex scene with more than 1 million triangles, multiple light sources and a variety of materials.[Lum17] This makes the path tracing step expensive, so we compute direct illumination only.

Tracing the paths through the scene will become more expensive as we add more geometry. The underlying acceleration structures for the triangles get larger, we experience cache or memory overflows more often and finding the closest intersection for each ray with the scene geometry requires more ray-triangle intersection tests. While this affects the time

needed for performing the path tracing, it does not affect the performance of our model. This is because we gather our training data during path tracing the image, so no additional paths need to be traced. The model itself has a fixed number of neurons and layers and we are evaluating it once per pixel, so it is dependent on the resolution of the screen, but not on the scene geometry. This means both evaluation and training of our model are independent of the scene geometry.

To verify this statement we used Nvidia Nsight Systems to determine the contribution of each component to the total frame time. Our test system uses an AMD Ryzen 5 7600X processor, 32 GB RAM and a RTX 3070. For our Neural Path Guiding (NPG), we measured the time to render a frame, including guiding and training. As a reference, we rendered the same scene with a forward path tracer. For this, we used Falcor's Megakernel Path Tracer (MKPT) with NEE enabled.

Predicting the guiding distribution is the main task of our approach. We need a guiding distribution for every pixel, so on a 1080p screen our *guiding model* will be evaluated more than 2 million times per frame. The MLP is evaluated in batches, this makes it fast enough for realtime applications. We measured the time required to predict the guiding distribution for all pixels.

Training the guiding model requires a normalization factor. We thus report the time required to *predict the normalization factor* for every training sample using the *normalization model*. As we use a learning based approach for normalization we have to train the *normalization model* as well. In table 6.1, we report the time to *train all*, which consists of predicting the normalization factor and the training of both MLPs. Currently more than 50% of our training time is spent on predicting and learning the normalization factor. For future optimization, is is important to know, which components require the most time.

	CornellBox	VeachBidir	BistroInterior	Average
predict guiding	1,942 ms	1,759 ms	1,786 ms	1,829 ms
train all	4,962 ms	4,515 ms	4,645 ms	4,707 ms
predict normalization	1,454 ms	1,318 ms	1,363 ms	1,378 ms
train guiding	2,006 ms	1,828 ms	1,864 ms	1,899 ms
train normalization	1,502 ms	1,369 ms	1,418 ms	1,430 ms
render frame NPG	13,42 ms	14,95 ms	15,01 ms	14,46 ms
render frame MKPT	3,51 ms	3,48 ms	9,12 ms	5,37 ms

Table 6.1.: Breakdown of the computational cost of our neural path guiding implementation.

Our NPG implementation is very expensive compared to a state of the art path tracer. This is most noticeable for simple scenes such as CornellBox where the path tracing itself requires less than 4 ms and our guiding adds almost 10 ms. This cost is not negligible, even on complex scenes such as BistroInterior. While the total rendering time per frame increases with the scene complexity, the time for predicting the guiding distribution and training the models stays the same. This is an important aspect as it proves our approach scales to complex scenes without additional computational cost.

6.3. Learning Direct Illumination

6.3.1. Scope and Limitations

In this section, we take a close look at the performance of our NPG technique when learning direct illumination. This is considerably easier and is an important first step. For this part we disable NEE intentionally as one of our main goals is to learn from noisy input data. This is necessary as path guiding techniques aim to learn complex indirect illumination which can not be sampled efficiently with NEE. Thus the samples to learn such indirect illumination are noisy as well.

We decided to use 16 samples per pixel (spp) in our renders for better visualization. Without NEE, images at 1 spp are very noisy.

Our technique uses the bounding box of the scene to normalize training data. For small scenes, this is a good normalization However, on larger scenes, where we only see a small part, this does not normalize the data but just maps it to a very small range. Because of this our testing is limited to small scenes as we did not focus on normalizing the positional input yet.

6.3.2. Convergence Speed

Our approach aims for use in realtime applications and with dynamic scenes. Pre-computing the radiance distribution for a scene is not possible as it might change over time. We analyze the convergence of our estimated radiance distribution. To do so we train our *guiding model* for 600 frames and measure the mean square error (MSE) as well as the FLIP score. For the training, we disable both NEE and indirect illumination. Note that this is the best case scenario for approximating with a vMF distribution as there is only one light source and without indirect illumination all incident radiance is within a small solid angle. On scenes with multiple light sources the vMF can not approximate the incident radiance distribution well and there is not much training progress to report. For reference we rendered the scene using a forward path tracer with NEE and BRDF importance sampling and 4096 samples per pixel (spp).

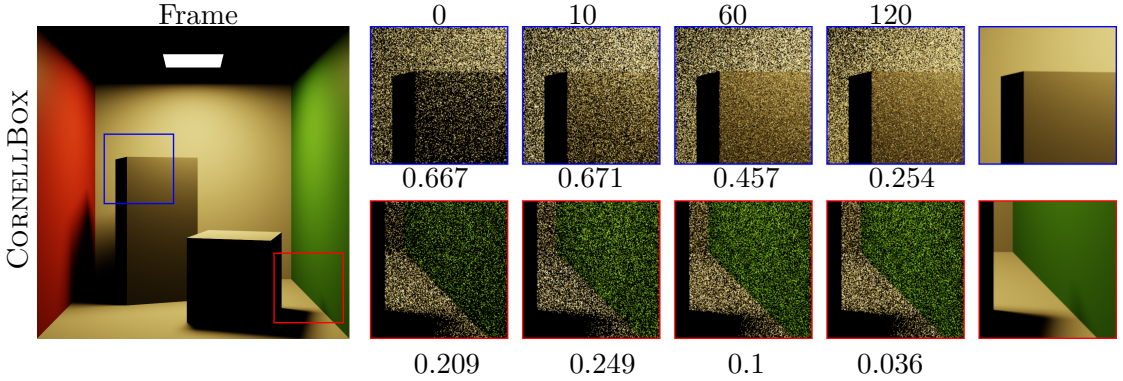


Figure 6.3.: Improvements after N frames of training. We report the MSE for each cutout. The estimate converges faster at surfaces close to the light source (blue cutout).

We report a rendered image after 0, 10, 60 and 120 frames of training. For the training process we used 1 spp, however for better visualization we used 16 spp for the images in figure 6.3. We report the MSE for each cutout section individually. We can see visibly less noise after 60 frames of training. This is most notable at surfaces close to the light source

(blue box). At these points it is more likely to sample the light source during training and thus the radiance distribution can be learned faster. Note that the MSE for the red cutout is significantly lower, this is because this section of the image is darker and thus its mean is lower as well.

In figure 6.4 both MSE and FLIP are plotted for the full duration of our test with the number of frames on the x-axis. For these plots we report the error across the full image. In the MSE plot we can see quick convergence after the first 15 frames. Within the first 120 frames there are notable jumps. These occur when the *guiding model* discovers a new part of the radiance distribution. Recall our vMF loss function 4.2 which is scaled by the MLP prediction. In case a previously unknown direction with high incident radiance is discovered, the resulting difference of target and prediction is large. However the scaling factor is very small. This results in a large loss value and also large gradients which dominate the training until the prediction is more accurate again. These jumps are not visible in the FLIP metric because it is focused on measuring high frequency noise by comparing it to the previously rendered image. As these jumps when discovering a new feature are limited by the learning rate they are comparably low frequency and are not captured by the FLIP metric because of that.

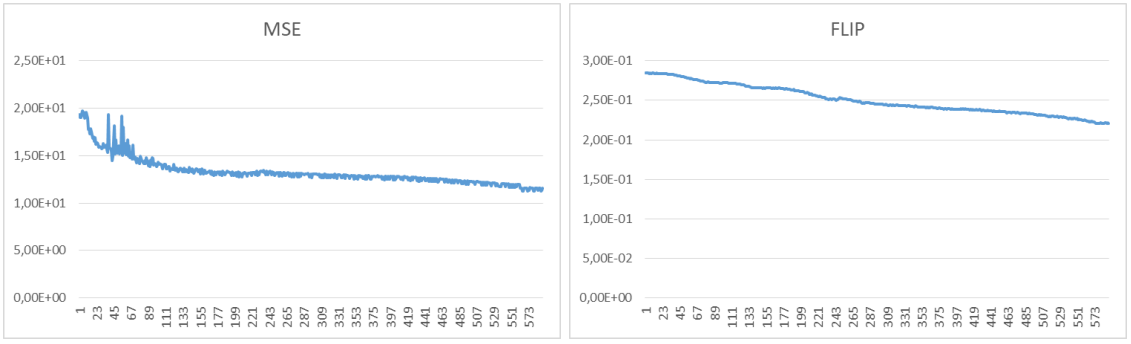


Figure 6.4.: Mean square error (MSE) and FLIP score during the training process. We see fast initial improvement but the MLP is capable of improving the estimate over an extended period of time. The quick initial convergence is important as this also means our approach can adapt to changes of the scene quickly.

6.3.3. Comparison

To evaluate the quality of our estimated radiance distribution, we compare our technique with a conventional forward path tracer (PT) without NEE and with a state of the art technique for direct illumination. We use Nvidia's RTXDI implementation of the ReSTIR algorithm (see section 2.3.3). This implementation is publicly available in Falcor 5.2, so we use a different version of the framework to render these images. RTXDI is heavily optimized whereas our approach is not. We thus rely on an equal sample count for comparison. We assume that a heavily optimized implementation of our approach would be slower than RTXDI by order of magnitude.

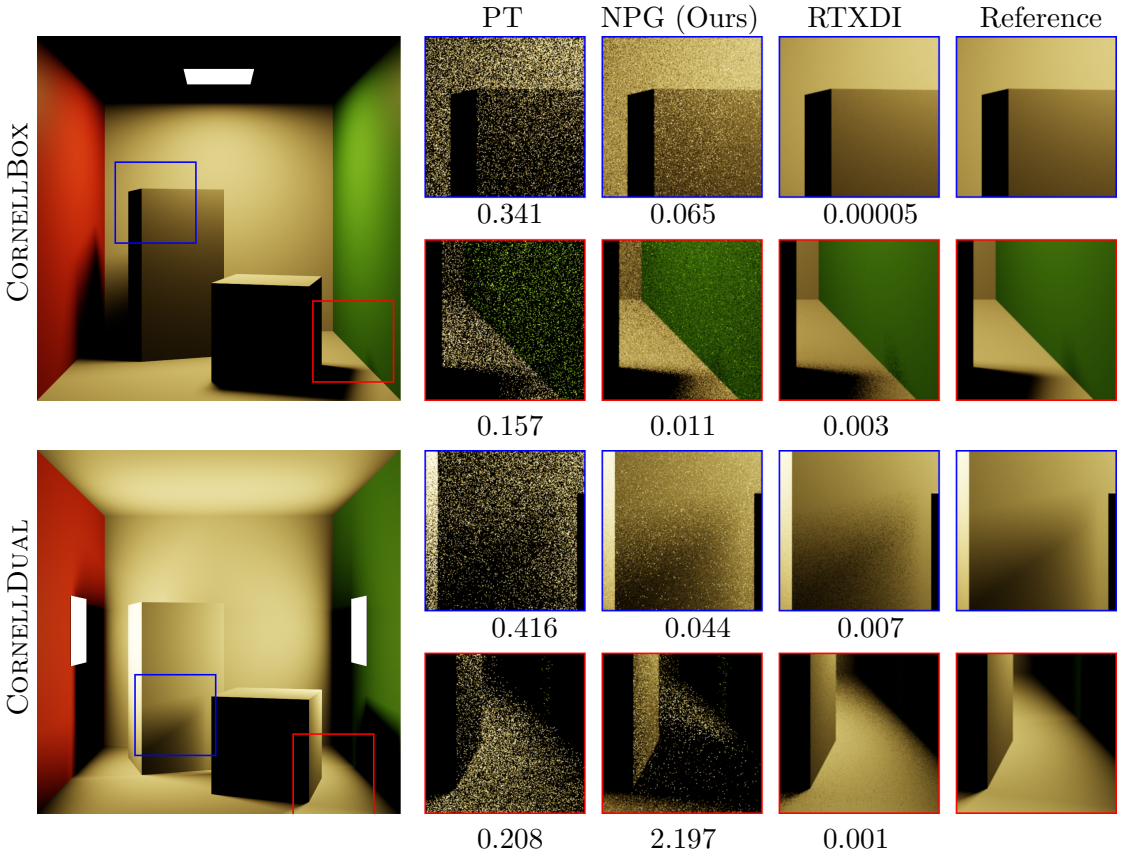


Figure 6.5.: We compare our NPG to plain path tracing without NEE and to RTXDI which is the current state of the art for realtime direct illumination. We report the MSE for each cutout individually. All images were rendered with 16 spp and equal time for initial convergence was given to all techniques. It should be noted that initial convergence of RTXDI is faster than ours by an order of magnitude.

6.4. Learning Indirect Illumination

6.4.1. Scope and Limitations

In this section, we analyze the performance of our NPG technique for indirect illumination. We use scenes with a lot of indirect lighting and explicitly do not guide the direct illumination (see section 4.7). This restriction is necessary because our normalization is only an estimate and if the intensity within the scene varies a lot, the bright spots dominate the training. As described in section 4.1 our technique is used for guiding on the first shading point (first path vertex), but not for subsequent ones. We do not compare to RTXDI in this section as it can not improve indirect illumination. Instead we rely on NEE which can be used at any path vertex to compute the direct illumination at that point. NEE is enabled for both the PT and our NPG.

6.4.2. Comparison

For testing indirect illumination we modified the CornellBox scene. By moving the area light outside the box and adding a window, only a small section is illuminated directly. This is a best case scenario for the *guiding model* as there is one bright spot to guide towards.

For comparison we also used VeachBidir which features two light sources, so our vMF might have problems fitting to those conditions. Additionally this scene contains specular materials such as the lamp or the (glass) egg on the table. Note that Falcor 4.4 contains a bug when rendering transmissive objects, where only the reflection is calculated but not the transmission. For rendering this scene we did disable caustic rendering as this introduces a lot of noise and with the Falcor path tracer we were not able to render a reference image with reasonably low noise. This scene was designed to show the limits of forward path tracing in the first place. For the reference images we used 8k spp. The renders were captured with 16 spp.

Our NPG technique performs better than regular path tracing on diffuse surfaces such as the walls in both scenes. However since we did not implement the product importance sampling, the BRDF is not sampled at all. On VeachBidir the highlights on the highly specular (glass) egg are missing because of this. The forward path tracer on the other hand can sample the BRDF.

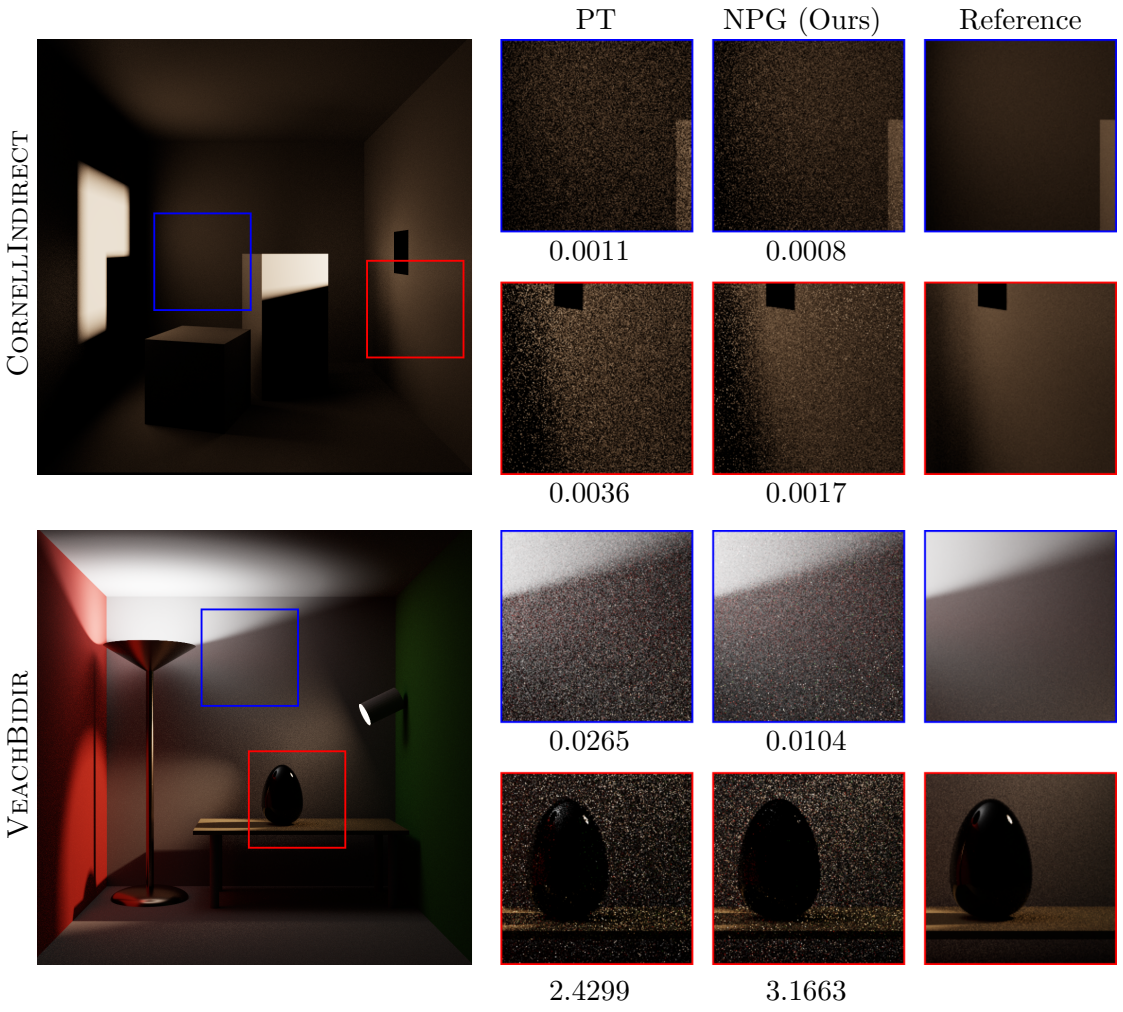


Figure 6.6.: We compare our NPG to plain path tracing with NEE and BSDF importance sampling. For CornellIndirect single-scattered indirect illumination is computed. On VeachBidir paths with two surface bounces were computed to enable the highlight in the reflective egg (red box).

7. Discussion and Future Work

Inspired by the neural radiance caching technique from Müller et al. [MRNK21] we have explored the use of MLPs for realtime path guiding. In this field it is common to build large and complex data structures to store guiding distributions for the entire scene. We propose to learn an estimate of the 5D radiance field of a virtual scene. Using online learning it is not necessary to learn this radiance field for the entire scene but only for the area visible to the camera. The method we proposed is capable of guiding both direct and indirect illumination and can adapt to dynamic changes in the scene. Online training of neural networks in realtime applications is made possible by modern hardware. This is a new research area with few attempts made so far, so many open research questions remain.

Performance

Our results show that the MLP is able to learn both direct and indirect illumination from noisy data collected during path tracing. In the case of direct illumination our technique is outperformed by state of the art techniques such as ReSTIR or NEE. Both can be used in realtime applications and with dynamic scenes. These techniques sample points on emissive geometry and don't require large shared memory on the GPU and specialized hardware for evaluating a MLP in realtime.

In the case of indirect illumination there are fewer alternatives and the ones operating in screen space are limited to guiding the scatter direction on the first path vertex. Path guiding techniques such as ours operate in world space and could be used for guiding multiple-scattered indirect illumination as well. However in realtime scenarios the amount of compute per frame is limited and thus paths have to be relatively short. So for now the focus should be on improving single-scattered indirect illumination. Both world space and screen space techniques are capable of learning this.

Normalizing the Training Targets

Most of the computational cost of our technique lies within the training. This is due to the need for normalizing the training targets. The normalization factor is expensive to compute and we have to rely on an approximation. While our explicit *normalization model* yields high quality results, the computational overhead is significant. Our alternative approach uses the *guiding model* to learn the normalization factor as well and has only little

additional computational cost. Future research could focus on optimizing this approach to avoid using a second MLP.

Huang et al. [HIT+23] proposed a loss function based on KL-divergence as well. In combination with a moment-based optimizer such as Adam, it is not necessary to compute the normalization factor at all. We did some initial tests with this approach where the learning worked well in the beginning. However our naive implementation of their loss with the vMF distribution suffers from stability issues and the learning fails completely in many cases. Further research is needed to use this loss function with vMF distributions.

Noisy Training Data

Using noisy data as training targets turns out to be a challenge. There are many cases where our MLP is not able to learn from the noisy training targets at all or suffers from overflowing weights. We propose to use a relative loss function and regularization to make the training more robust to noise. Additionally we try to use our current estimate of the radiance distribution for generating training targets with less noise for the next training step.

This approach was discussed by Huang et al. [HIT+23] as well. They reduced the noise in their training targets using NEE. However they reported that their technique suffered from stability issues when rendering caustics. Since caustics are a prominent example of light paths that can not be sampled efficiently using NEE, noisy training data appears to be a general problem when using neural networks for rendering.

For the use of neural path guiding in difficult lighting conditions, it is crucial to find more robust ways to train the neural networks, as in MC methods, noise is inevitable.

8. Appendix

A. vMF Loss Gradient

To train the MLP with a gradient based optimizer such as Adam (see 2.1.2) we have to calculate the gradient of our loss function 4.2. The gradient is composed by the partial derivatives of our loss function. We want to optimize the parameters of our vMF so we derive with respect to the mean vector $\mu = (\mu_1, \mu_2, \mu_3)^T$ and the concentration parameter κ. For the implicit normalization approach A is part of the MLP output and thus we need the partial derivative with respect to A as well:

$$\frac{\partial}{\partial \mu_i}\mathcal{L}^2_{\text{vMF}}(f_{\text{vMF}}(\omega_s;\mu,\kappa), f_L(\omega_s)) = \frac{-2\cdot(L(x,\omega_s) - f_{\text{vMF}}(\omega_s;\mu,\kappa)\cdot A)}{pdf(\omega_s)^2\cdot \text{sg}(f_{\text{vMF}}(\omega_s;\mu,\kappa)\cdot A)^2 + \epsilon}\cdot\frac{\partial}{\partial \mu_i} f_{\text{vMF}}(\omega_s;\mu,\kappa)\cdot A$$

$$\frac{\partial}{\partial \kappa}\mathcal{L}^2_{\text{vMF}}(f_{\text{vMF}}(\omega_s;\mu,\kappa), f_L(\omega_s)) = \frac{-2\cdot(L(x,\omega_s) - f_{\text{vMF}}(\omega_s;\mu,\kappa)\cdot A)}{pdf(\omega_s)^2\cdot \text{sg}(f_{\text{vMF}}(\omega_s;\mu,\kappa)\cdot A)^2 + \epsilon}\cdot\frac{\partial}{\partial \kappa} f_{\text{vMF}}(\omega_s;\mu,\kappa)\cdot A$$

$$\frac{\partial}{\partial A}\mathcal{L}^2_{\text{vMF}}(f_{\text{vMF}}(\omega_s;\mu,\kappa), f_L(\omega_s)) = \frac{-2\cdot(L(x,\omega_s) - f_{\text{vMF}}(\omega_s;\mu,\kappa)\cdot A)}{pdf(\omega_s)^2\cdot \text{sg}(f_{\text{vMF}}(\omega_s;\mu,\kappa)\cdot A)^2 + \epsilon}\cdot f_{\text{vMF}}(\omega_s;\mu,\kappa)$$

Bibliography

[And22] Andeggs, "3dspherical," 2022, [Online; accessed March 30, 2023]. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=7478049

[Bit16] B. Bitterli, "Rendering resources," 2016, https://benedikt-bitterli.me/resources/.

[BWP+20] B. Bitterli, C. Wyman, M. Pharr, P. Shirley, A. Lefohn, and W. Jarosz, "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting," *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, vol. 39, no. 4, Jul. 2020.

[Cau22] B. Caulfield. (2022) What is path tracing? [Online]. Available: https://blogs.nvidia.com/blog/2022/03/23/what-is-path-tracing/

[Der21] M. Derevyannykh, "Real-time path-guiding based on parametric mixture models," 2021. [Online]. Available: https://arxiv.org/abs/2112.09728

[DH06] H. Dehling and B. Haupt, *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Springer-Verlag, 2006.

[Dra19] R. Draelos. (2019) Glass box medicine. [Online]. Available: https://glassboxmedicine.com/2019/06/08/regularization-for-neural-networks-with-framingham-case-study/

[GD98] M. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric Environment*, vol. 32, no. 14, pp. 2627–2636, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1352231097004470

[Hd14] E. Heitz and E. d'Eon, "Importance sampling microfacet-based bsdfs using the distribution of visible normals," in *Computer Graphics Forum*, vol. 33, no. 4. Wiley Online Library, 2014, pp. 103–112.

[HEV+16] S. Herholz, O. Elek, J. Vorba, H. Lensch, and J. Křivánek, "Product importance sampling for light transport path guiding," *Computer Graphics Forum*, vol. 35, no. 4, pp. 67–77, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12950

[HG14] K. Hornik and B. Grün, "movmf: An r package for fitting mixtures of von mises-fisher distributions." *Journal of Statistical Software*, vol. 58, no. 10, pp. 1 – 31, 2014.

[HIT+23] J. Huang, A. Iizuka, H. Tanaka, T. Komura, and Y. Kitamura, "Online neural path guiding with normalized anisotropic spherical gaussians," *arXiv preprint arXiv:2303.08064*, 2023.

[HK22] R. K. Hota and C. S. Kumar, "Derivation of the rotation matrix for an axis-angle rotation based on an intuitive interpretation of the rotation matrix," in *Machines, Mechanism and Robotics: Proceedings of iNaCoMM 2019*. Springer, 2022, pp. 939–945.

[Jak12] W. Jakob, "Numerically stable sampling of the von mises-fisher distribution on s^2 (and other tricks)," *Interactive Geometry Lab, ETH Zürich, Tech. Rep*, p. 6, 2012.

[Kaj86] J. T. Kajiya, "The rendering equation," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, p. 143–150, aug 1986. [Online]. Available: https://doi.org/10.1145/15886.15902

[KCK+22] S. Kallweit, P. Clarberg, C. Kolb, T. Davidovič, K.-H. Yao, T. Foley, Y. He, L. Wu, L. Chen, T. Akenine-Möller, C. Wyman, C. Crassin, and N. Benty, "The Falcor rendering framework," 8 2022, https://github.com/NVIDIAGameWorks/Falcor. [Online]. Available: https://github.com/NVIDIAGameWorks/Falcor

[KE09] M. Kurt and D. Edwards, "A survey of brdf models for computer graphics," *SIGGRAPH Comput. Graph.*, vol. 43, no. 2, may 2009. [Online]. Available: https://doi.org/10.1145/1629216.1629222

[LH91] H. Leung and S. Haykin, "The complex backpropagation algorithm," *IEEE Transactions on signal processing*, vol. 39, no. 9, pp. 2101–2104, 1991.

[Lum17] A. Lumberyard, "Amazon lumberyard bistro, open research content archive (orca)," July 2017, `http://developer.nvidia.com/orca/amazon-lumberyard-bistro`. [Online]. Available: http://developer.nvidia.com/orca/amazon-lumberyard-bistro

[MÖ21] T. Müller, "tiny-cuda-nn," 4 2021. [Online]. Available: https://github.com/NVlabs/tiny-cuda-nn

[MGN17] T. Müller, M. Gross, and J. Novák, "Practical path guiding for efficient light-transport simulation," *Computer Graphics Forum*, vol. 36, no. 4, pp. 91–100, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13227

[MLM18] L. Martino, D. Luengo, and J. Míguez, *Independent random sampling methods*. Springer, 2018.

[MPC19] P. Moreau, M. Pharr, and P. Clarberg, "Dynamic many-light sampling for real-time ray tracing." in *High Performance Graphics (Short Papers)*, 2019, pp. 21–26.

[MRNK21] T. Müller, F. Rousselle, J. Nová k, and A. Keller, "Real-time neural radiance caching for path tracing," *ACM Transactions on Graphics*, vol. 40, no. 4, pp. 1–16, 08 2021. [Online]. Available: https://doi.org/10.1145%2F3450626.3459812

[OLK+21] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni, "Restir gi: Path resampling for real-time path tracing," *Computer Graphics Forum*, vol. 40, no. 8, pp. 17–29, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14378

[Ulr84] G. Ulrich, "Computer generation of distributions on the m-sphere," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 33, no. 2, pp. 158–163, 1984.

[Vea98] E. Veach, *Robust Monte Carlo methods for light transport simulation*. Stanford University, 1998.

[VHH+19] J. Vorba, J. Hanika, S. Herholz, T. Müller, J. Křivánek, and A. Keller, "Path guiding in production," in *ACM SIGGRAPH 2019 Courses*, ser. SIGGRAPH '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3305366.3328091

Declaration of originality

I hereby declare that I have composed this paper by myself and without any assistance other than the sources given in my list of works cited. This paper has not been submitted in the past or is currently being submitted to any other examination institution. It has not been published. All direct quotes as well as indirect quotes which in phrasing or original idea have been taken from a different text (written or otherwise) have been marked as such clearly and in each single instance under a precise specification of the source. I have observed the current version of the KIT statutes for ensuring good scientific practice and I am aware that any false claim made here results in failing the examination.

Karlsruhe, March 30, 2023 (Dominik Wüst)