Metappearance: Meta-Learning for Visual Appearance Reproduction

MICHAEL FISCHER, University College London, United Kingdom
TOBIAS RITSCHEL, University College London, United Kingdom

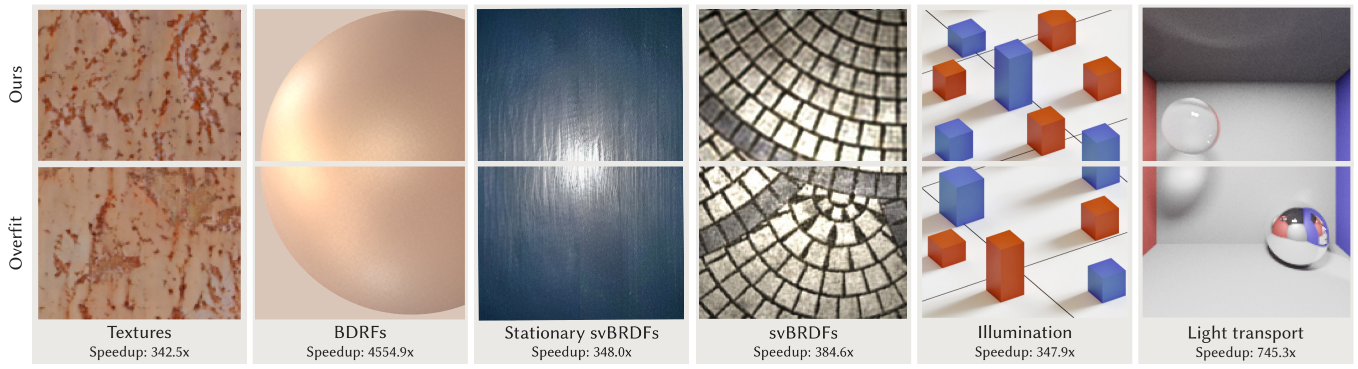


Fig. 1. We propose meta-learning for a wide range of appearance reproduction tasks. Given as few as 10 optimization steps, our method (top in each subfigure) achieves quality comparable to overfit-approaches (bottom in each subfigure) that take orders of magnitude more training iterations.

There currently exist two main approaches to reproducing visual appearance using Machine Learning (ML): The first is training models that generalize over different instances of a problem, e.g., different images of a dataset. As one-shot approaches, these offer fast inference, but often fall short in quality. The second approach does not train models that generalize across tasks, but rather over-fit a single instance of a problem, e.g., a flash image of a material. These methods offer high quality, but take long to train. We suggest to combine both techniques end-to-end using meta-learning: We over-fit onto a single problem instance in an inner loop, while also learning how to do so efficiently in an outer-loop across many exemplars. To this end, we derive the required formalism that allows applying meta-learning to a wide range of visual appearance reproduction problems: textures, Bi-directional Reflectance Distribution Functions (BRDFs), spatially-varying BRDFs (svBRDFs), illumination or the entire light transport of a scene. The effects of meta-learning parameters on several different aspects of visual appearance are analyzed in our framework, and specific guidance for different tasks is provided. Metappearance enables visual quality that is similar to over-fit approaches in only a fraction of their runtime while keeping the adaptivity of general models.

Additional Key Words and Phrases: Visual Appearance; Deep Learning; Meta-Learning; BRDFs; svBRDFs; Light Transport.

1 INTRODUCTION

Reproduction of visual appearance [Dorsey et al. 2010] is a key part of Computer Graphics that has achieved new levels of simplicity, speed and accuracy thanks to recent developments in Machine learning (ML). The classic use of ML for appearance reproduction was to capture light or materials from very little input, sometimes only single images [Deschaintre et al. 2018; Georgoulis et al. 2017], without access to ground truth maps. Approaches that are capable thereof usually train for a long time on large datasets and achieve impressive levels of generalization, often due to Convolutional neural networks (CNNs) that recognize patterns in the data. Unfortunately, this generality comes at the price of not matching the target precisely: we might get a great looking Bi-directional Reflectance Distribution Function (BRDF) or spatially-varying BRDF (svBRDF) from a single image, but it might not exactly match the input.

More recently, a second line of research has evolved, where no attempt is made to generalize over a large dataset, and, instead, non-linear optimization and differentiable rendering are used to explain visual appearance in input images [Gatys et al. 2015; Mildenhall et al. 2020; Rainer et al. 2019]. These methods minutely match the reference, but need many input observations, take long to train and can be slow to execute. Typically, such approaches use point-operations, e.g., Multi-layered perceptrons (MLPs), rather than CNNs.

A first step to combine these two training paradigms was introduced by adapting the output of a model from the general class in a second, non-end-to-end step, the so-called fine-tuning or post-refinement stage [Deschaintre et al. 2020; Gao et al. 2019; Guo et al. 2020; Henzler et al. 2021]. Approaches that use fine-tuning usually run an additional number of gradient steps (in the order of magnitude 10^3) towards a specific target, which greatly improves reconstruction quality, but inflates runtime to the order of minutes, whereas feed-forward CNNs operate in milliseconds.

A dilemma materializes: Should one rather make a user wait in order to provide them with high quality output, or would it be better to provide fast, interactive results that might be of inferior quality? Both solutions are unsatisfactory, which is why in this work, we aim to diminish this quality-speed-gap and provide quality that is a) close to model-overfitting or fine-tuning, and b) available at interactive runtimes, close to those of general feed-forward networks.

We achieve this by harnessing the power of *meta-learning*: building on the MAML algorithm [Finn et al. 2017] from the machine learning community, our framework Metappearance uses two nested optimization loops, where the outer loop is sequentially presented

Authors' addresses: Michael Fischer, University College London, United Kingdom, m.fischer@cs.ucl.ac.uk; Tobias Ritschel, University College London, United Kingdom, t.ritschel@ucl.ac.uk.

with all exemplars in a (training) dataset. For each exemplar, the inner loop is then tasked with over-fitting a model onto this specific exemplar. Characteristically, the inner loop operates under the constraint of a very limited number of available gradient descent steps, typically around 10 only. Metappearance hence learns to efficiently drive the inner optimization towards a specific target, but still is able to exploit coherency and priors in the data due to knowledge gathered in the outer loop.

In this work, we present a framework that formalizes the application of meta-learning to the task of visual appearance reproduction. Importantly, we do not propose new visual appearance methods or new loss functions, nor do we compare methods or analyze their properties. In fact, quite the contrary is true: we keep the methods the same, but instead propose a different way of training them. By comparing our approach against "traditional" training paradigms, we show which types of applications can benefit from meta-learning and explore the implications on performance and quality. We validate that Metappearance outperforms mere general inference followed by fine-tuning through ablation- and convergence-studies. Additionally, we, for the first time in the graphics literature, make the connection between meta-learning, model compression and data efficiency. We show that Metappearance speeds up faithful appearance reproduction by several orders of magnitude, while keeping all desirable properties of the respective base approaches and similar visual quality.

In summary, our contributions are

- Metappearance[1], a model that adapts to new, unseen visual appearance tasks in only a few steps of gradient descent.
- Optimizing for a fast and accurate optimizer of this model.
- Instances of this model that accurately match texture, BRDFs, svBRDFs, illumination, or light transport orders of magnitude faster than strong baselines, at comparable quality, and
- An analysis of our method's properties, its convergence and its behaviour under ablation.

2 PREVIOUS WORK

2.1 Visual Appearance

We consider visual appearance reproduction, the task of generating plausible and accurate visual patterns across all positions and orientations from evidence captured for some angles and locations.

Ignoring angle and considering an exemplar's statistics, we would talk about appearance as *texture* [Efros and Leung 1999; Gatys et al. 2015; Julesz 1975]. When angle matters, we would call this Bi-directional Reflectance Distribution Function (BRDF) [Guarnera et al. 2016; Ngan et al. 2005] and when both space and orientation are considered, spatially-varying BRDF (svBRDF) [Dana and Wang 2004]. Guarnera et al. [2016] summarize these approaches. Textures [Gatys et al. 2015; Ulyanov et al. 2016], BRDFs [Georgoulis et al. 2017] and svBRDFs [Deschaintre et al. 2018] have all been acquired and represented by means of ML, for which Tewari et al. [2020] provide a survey. We defer discussion of the specific existing solutions for all those sub-problems to Sec. 4.1.

[1]Our source code will be available at https://github.com/mfischer-ucl/metappearance.

2.2 Learning

More important to our problem is how the different methods are trained, i.e., optimized, given either the information of a single instance or an entire set of exemplars (Tab. 1).

Table 1. Different ways to optimize for visual appearance reproduction.

	General	Fast	Accurate	Compact
`General`	✓	✓	×	×
`Overfit`	×	×	✓	✓
`Finetune`	×	×	✓	×
`Meta` (ours)	✓	✓	✓	✓

General Learning. A typical paradigm is to collect a training dataset, say, 2D images, to curate them with appearance supervision, e.g., BRDF parameters, and to learn a mapping from the image to those parameters, for example through a CNN [Georgoulis et al. 2017]. Often, such methods create a latent space. While it is a strength that this space will mostly contain valid exemplars, it comes at the expense of a bottleneck, reducing specific details. In simple words, a 100-dimensional latent space can make sure every latent code is a grass texture, but it cannot represent the exact location of 200 grass blades in an image. Examples of such approaches include work by Henzler et al. [2020] (texture), Georgoulis et al. [2017] (BRDF) or Deschaintre et al. [2018]; Gao et al. [2019]; Guo et al. [2020]; Kuznetsov et al. [2019] (svBRDF) or Bako et al. [2019]; Huo et al. [2020]; Zhu et al. [2020, 2021] (light paths). These methods generalize well to new data, but do not *exactly* match the test-time input, and hence are *general* and *fast*, but not *accurate*, as per the taxonomy established in Tab. 1. Moreover, they often require an encoder- and decoder branch, which makes them not *compact*. We call these **`General`** and formalize them in Sec. 3.2.

Over-fit Optimization. A second, more classical approach is to not seek generalization, but to fit a model to samples of a specific problem instance. This technique has seen a recent increase in popularity due to the emergence of coordinate-based neural representations, and often is used in conjunction with MLPs. Examples are numerous and include most works related to Neural radiance fields (NeRF) [Mildenhall et al. 2020] as well as others for texture [Kuznetsov et al. 2021], BRDFs [Sztrajman et al. 2021], svBRDFs [Zhang et al. 2021] or the entire light transport [Müller et al. 2019; Zheng and Zwicker 2019]. We call these methods **`Overfit`** and define them in Sec. 3.3. Most **`Overfit`** approaches are *accurate* and *compact* (they usually do not require an encoder, as they do not need to generalize), but neither *fast* to train nor *general*.

Fine-tuning. A combination of above approaches is sometimes used, where first a general network is trained and then, when the target instance is known, is optimized a second time [Deschaintre et al. 2020; Henzler et al. 2021]. Some have employed optimization in latent space [Tan and Mayrovouniotis 1995] while keeping the rest of the network fixed [Gao et al. 2019; Guo et al. 2020; Kang et al. 2018, 2019], or in pixel space after a user-adjusted number of iterations, aiming to fit the target perfectly. We here name these **`Finetune`** and define them in detail in Sec. 3.4. Approaches that use fine-tuning or post-optimization usually are *accurate*, but neither *fast* (post-refinement usually happens at non-interactive runtimes) nor *general* (once the model is fine-tuned, it cannot be used for general inference anymore). Most fine-tuning models are *compact*, as it is usually enough to store the fine-tuned decoder and the corresponding latent code ([Henzler et al. 2021]), although this is not always the case ([Deschaintre et al. 2020].

Hyper- and meta-learning. Hyper-networks produce weights of another network [Ha et al. 2016]. This has been applied to appearance [Bi et al. 2021; Maximov et al. 2019], BRDFs [Sztrajman et al. 2021] and NeRF-like representations [Sitzmann et al. 2019]. Meta-learning, instead, does not directly produce the parameters of another network, but guides the optimization that drives the inner learning. This optimization is often based on gradient descent, so the outer optimization produces setting such as start values and step sizes. Sometimes, the gradient rule itself is learned [Adler and Öktem 2018; Ravi and Larochelle 2016]. Applications of meta-learning were proposed for geometry [Sitzmann et al. 2020], super-resolution [Hu et al. 2019] and animation [Wang et al. 2021], layered depth images [Flynn et al. 2019], as well as for NeRF by Bergman et al. [2021] and Tancik et al. [2021].

Approaches that use meta-learning are *fast* and *general* by construction, as they can run inference on new, unseen samples in only a few gradient steps. As we will show in this work, meta-learning for visual appearance reproduction is also *accurate*, as its output is close to overfit- or fine-tuning quality. Moreover, meta-learning enables *compact*ness, as the model initialization and optimization themselves are learned, and hence do not need to rely on latent codes produced by, e.g., bulky encoder networks. We would not be aware of work attempting to model visual appearance using meta-learning, as we set out to do in Sec. 3.5.

3 OUR APPROACH

After introducing the problem we solve (Sec. 3.1), we provide a common formalization of three previous solutions (Sec. 3.2, 3.3 and 3.4), and finally introduce Metappearance (Sec. 3.5).

3.1 Problem statement

We now discuss representing visual appearance, its parametrization, and finally its optimization.

Representation. We represent visual appearance as $L_\theta(\mathbf{x}|I)$, a radiance function of a positional-directional coordinate $\mathbf{x}$, conditioned on input I and parametrized by the tunable vector θ. The coordinate $\mathbf{x}$ can be two-, three- or higher-dimensional and might be positional, directional or both. The condition I varies per application and could be a single image, sparse measurements or light paths.

Parametrization. Parameterizing L_θ by θ is possible in a large number of ways, for instance through a plain, pixel-based RGB image, spatial data-structures, or a more implicit representation, like a CNN or an MLP, and the parametrization might make use of hard-coded, rendering-like operations. For now, we deliberately do not specify this further and only require L to be differentiable w.r.t. the parameters θ. Supplemental Tab. 2 will give examples for instances of this model which we will evaluate in our experiments.

Optimization. Let us assume a scalar function $\text{Loss}(\theta, T)$ that is low if θ explains the data T well and high otherwise. We will specify different ways to define this loss, leading to different approaches of reproducing visual appearance. Let us further assume that we have access to an optimizer function $\textsc{Learn}(\theta_0, \alpha, T, \text{Loss})$ which performs Gradient descent (GD) that starts at θ_0 to change parameter θ with stepsize α so as to minimize the loss Loss. This procedure is given in Alg. 1, where n_l is the number of GD iterations.

Algorithm 1 Classic learning: The function `grad(·)` differentiates its first argument (an expression) with respect to the second.

```
1: procedure $\textsc{Learn}(\theta_0, \alpha, T, \text{Loss})$
2:     $\theta = \theta_0$
3:     for $i \in \{1, \dots, n_\text{l}\}$ do
4:         $\theta$ -= $\alpha \cdot$ grad(Loss($\theta$,$T$), $\theta$)
5:     end for
6:     return $\theta$
7: end procedure
```

Combining Loss and Learn leads to different methods. In classic learning, the start parameters and the optimization step size both are hyper-parameters that need to be chosen by the user. We will see that **Meta**-learning chooses these optimally through optimization.

3.2 General

General methods, that attempt to map a condition I directly to appearance, use the loss described in Alg. 2:

$$\text{Loss}_{\texttt{General}}(\theta, T) = \mathbb{E}_{i \in T}[\Delta(\,L_\theta(\mathbf{x}_i|I_i), L_i\,)] \tag{1}$$

where $\Delta(\cdot,\cdot)$ here, and in the following, can refer to any norm.

Algorithm 2 Classic loss. The function `sample(·)` takes a set as an argument and returns a random index into that set.

```
1: procedure Loss($\theta$, $T$)
2:     $cost = 0$
3:     for $i \in \{1, \dots, b\}$ do
4:         $j$ = sample($T$)
5:         $cost$ += $\Delta(L_\theta(\mathbf{x}_j|I_j), L_j)$
6:     end for
7:     return $cost/b$
8: end procedure
```

Visual appearance problems usually are ambiguous: One I_i can typically be explained by more than one parameter vector θ. Over the course of training, a **General** optimization sees many different conditions I_i and hence can build priors about what solutions are more likely than others. These priors are then used to generalize to new conditions under new angles and positions, e.g., a new 2D photo of a sphere that can then provide reflectance for new 3D angles and positions. However, the encoding of these priors that then handle variations over I must be performed under the constraint of a finite budget of parameters. In typical applications, this results in more or less subtle forms of smoothing: a generated BRDF does not quite resemble the BRDF the input specifies, some spatial details are lost in svBRDFs, etc. We will show examples of this in Sec. 4.1.

3.3 Over-fitting

Differently, in over-fitting, the loss is

$$\text{Loss}_{\texttt{Overfit}}(\theta, T) = \mathbb{E}_{i \in T}[\Delta(\, L_\theta(\mathbf{x}_i|I), L_i\,)] \quad (2)$$

where, importantly, I is constant and does not depend on i. This task is comparatively easy, as the network only has to deal with one specific input. Consequently, results are often of higher quality than in the **General** setting. However, the optimization now lacks the synoptic approach that sees all instances and can use this "bigger picture" to build priors and make do with fewer information in lower time. Typically, over-fitting approaches need many iterations to train, take from minutes to hours to converge, and often require further regularization, e.g., by physical constraints, to avoid overfitting to specific training positions and directions.

3.4 Fine-tuning

Both overfitting and generalization can be combined in a trivial way: First run a general method on the input, second, optimize the output so that it resembles the input even more. Fine-tuning usually starts from initial parameters θ_0, that have been trained across many inputs (e.g., the converged state of a general model, cf. Sec. 3.2), and then optimizes these for a fixed target, as in over-fitting (Sec. 3.3), with a fixed step size α_F. This means to compute

$$\textsc{Learn}(\textsc{Learn}(\theta_0, \alpha_\text{G}, T, \text{Loss}_{\texttt{General}}), \alpha_\text{F}, T, \text{Loss}_{\texttt{Overfit}}). \quad (3)$$

While the general step can be re-used across several inputs, the subsequent fine-tuning (essentially, over-fitting) optimization must be repeated for each new input. **Finetune** is faster than **Overfit**, as only the inner optimization needs to be executed for inference, while the outer step is a feed-forward network execution, and sometimes is accelerated further by increasing the learning rate α_F. Still, optimization usually takes in the order of minutes, i.e., it is slow compared to a single feed-forward execution of the general network that typically would take milliseconds. Moreover, the solution might diverge from the priors that informed the first step. By jointly training over both the general projection and the fine-tuning stage, we overcome these issues in quality and speed, as explained next.

3.5 Meta-learning

A general model's training is agnostic to the fact that later fine-tuning iterations will be used to further improve the results. This drives the **General** projection step towards learning unnecessarily detailed representations while missing other important features and over-smoothing the space (cf. Sec. 3.2). The **General** step hence will try to incorporate features that fine-tuning might include anyway, and subsequently disregard other, more general elements that the fine-tuning operator might miss.

If we do not know how to trade those properties, could we instead learn how to do that? Could we learn how to perform an optimization optimally? To do that, we need i) a domain to optimize over, ii) to understand what is "optimal", and iii) an actual algorithm. We will now look into these aspects.

As our optimization domain, we consider meta learning of both the initial solution θ_0 as well as a per-parameter step size α. Both the initialization and the step sizes are fixed between tasks and stay constant at test time. We stack θ_0 and α into a *meta parameter* vector,

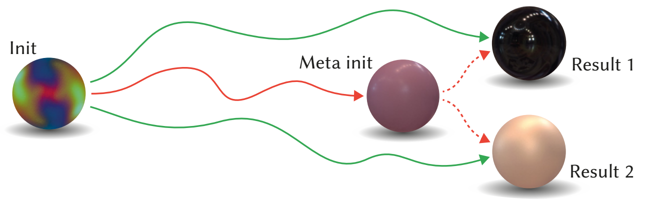


Fig. 2. Learning the init: Trajectories for **Meta** and **Overfit** for the example task of BRDF representation. The dotted line denotes inner optimization. Note how the dotted trajectories for **Meta** are shorter, i.e., faster learning.

denoted as $\phi = \{\theta_0, \alpha\}$. Meta learning can then be formalized[2] as a new loss (Alg. 3):

$$\text{Loss}_{\texttt{Meta}}(\phi, \mathcal{T}) = \mathbb{E}_{i \in \mathcal{T}}[\text{Loss}_{\texttt{Overfit}}(\textsc{Learn}(\phi, T_i, \text{Loss}_{\texttt{Overfit}}), T_i)]. \quad (4)$$

The first thing to note is that the loss is defined on meta-parameters ϕ and that it calls Learn with these, to quantify how suitable they are for an inner learner. Second, it samples from the space of all tasks $\mathcal{T}$ (e.g., multiple BRDFs), not from a single task. The sampled task T is the same for meta-train and meta-test; the same for the call to Learn and to Loss. Because `sample` inside the loss function is randomized, different positions and directions are used for meta-test and meta-train. Doing so, parameters that generalize across positions and directions inside one task are advantaged.

To actually perform meta-learning, we

$$\textsc{Learn}(\phi_0, \alpha_\text{M}, \mathcal{T}, \text{Loss}_{\texttt{Meta}}),$$

i.e., perform common learning with an advanced loss and a meta-initialization, ϕ_0, as well as a meta step-size α_M.

Algorithm 3 Meta-learning involves a loss that depends on the hyper-parameters of calling the function Learn on the actual task.

```
1: procedure MetaLoss($\phi$, $\mathcal{T}$)
2:     $cost = 0$
3:     for $i \in \{1, \ldots, b\}$ do
4:         $j$ = sample ($\mathcal{T}$)
5:         $cost$ += Loss(Learn($\phi$, $\mathcal{T}_j$, Loss), $\mathcal{T}_j$))
6:     end for
7:     return $cost/b$
8: end procedure
```

By encouraging network parametrizations that enable few-step convergence on unseen samples, meta-learning optimizes over optimization itself. More specifically, in our scenario, the inner optimizer learns to over-fit to the appearance of one exemplar. The outer optimizer then changes the inner optimizer's start parameter values, so that the next inner-loop execution will achieve improved results and do so much quicker. Fig. 2 illustrates this idea with two very basic tasks. As with other losses, the metaloss is computed across a batch, i.e., ϕ_0 and α_M are updated with information averaged across multiple optimizations (the `for` loop in Line 3).

Fig. 3 illustrates the purpose of learning the step size. As explained in Fig. 2, meta-learning will change the init from A to a suitable

[2]In a slight abuse of notation, as Learn takes four parameters, while it is called with three here, where the first is a tuple holding the first two arguments, init and stepsize.

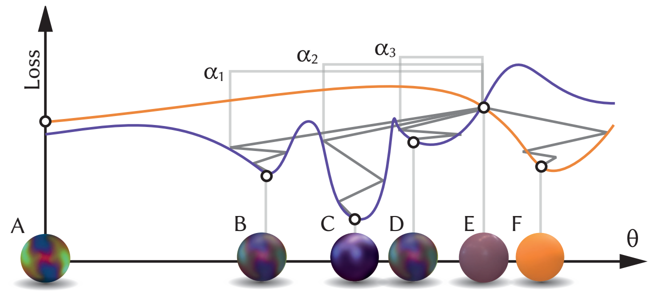


Fig. 3. Learning the step size: The orange and violet curve show the loss (vertical) for different parameters θ (horizontal) for two BRDF tasks. The gray α-intervals denote three alternative step sizes. The zig-zags are the convergence paths for specific choices of step size. Please see the text for discussion.

position E. When choosing the step size right, (α_2 for this init) the optimizer will converge to the correct BRDFs, here C and F. With a step size too small, α_3, or a step size too large, α_1, we converge to less suitable results (D or B, respectively) for the violet task. As the above considerations might be different in higher dimensions, we parameterize the step size as a vector instead of a scalar, which allows anisotropic gradient steps [Li et al. 2017]. During meta-inference, i.e., when using the meta-trained model to quickly infer a result for a new, unseen sample provided by a user, the step-sizes are fixed, and only the model weights are changed.

Jointly learning the model initialization and the corresponding step sizes combines the quality of over-fitting with the ability to build priors of general approaches. In practice, the inner training loop takes several orders of magnitude fewer iterations than common over-fitting and is up to two orders of magnitudes faster than fine-tuning, which enables execution at interactive rates: in most applications, our inference time is less than 1 second.

Implementation. Our implementation follows the Model-agnostic meta-learning (MAML) framework proposed by Finn et al. [2017]. As the name suggests, the meta-learner is agnostic to the inner network used, which makes the approach flexible and well-suited for our different application scenarios. We learn our per-parameter stepsize (cf. Fig. 3) according to the approach presented in Meta-SGD [Li et al. 2017]. For details of the different meta-learning algorithms and tools used, please see Supplemental Sec. 1.

4 EVALUATION

We have introduced a framework for using meta-learning for visual appearance reproduction, but how well does it compare to more traditional training approaches? To answer this, we will now demonstrate the effectiveness of Metappearance on a variety of different applications. We will now introduce those, including notes on previous work and the architecture (Sec. 4.1), then outline the evaluation protocols (Sec. 4.2), and, finally, report qualitative and quantitative results (Sec. 4.3).

4.1 Applications

We consider six increasingly complex applications (Supplemental Tab. 2): i) RGB textures, ii) BRDFs, iii) stationary and iv) non-stationary svBRDF maps from flash images, v) illumination maps from RGB images with normals and finally vi) the entire light transport in a scene.

Neither the tasks addressed nor architectures used are novel; the contribution lies in the way they are trained. We re-iterate that it is hence not our goal to compare different *approaches* (e.g., CNN vs. MLP for BRDF encoding), but rather compare different methods of *training* a specific approach. We will detail each application next.

4.1.1 Textures. In a TEXTURE, RGB appearance varies over space, but has uniform visual feature statistics [Portilla and Simoncelli 2000]. Gatys et al. [2015] optimized for a finite image in pixel space such that its VGG activation statistics match the exemplar, a solution that would be **`Overfit`** in our taxonomy. Later, Ulyanov et al. [2016] trained a single CNN to perform this task feed-forward. Huang and Belongie [2017] have shown how control over (instance) normalization can produce new textures corresponding to a **`General`** solution in the logic of this work. Henzler et al. [2020] show how to do this conditioned on an input image, optionally involving a step of **`Finetune`**. For a comprehensive survey, we refer the reader to Raad et al. [2018].

These methods are exemplary for the spectrum we challenge: either they take long to learn and fit the input exactly, or they are fast and only approximate the input. We study a design based on Ulyanov et al. [2016] and Henzler et al. [2020] as per TEXTURE in Supplemental Tab. 2. For the exact network and training setup, please confer Supplemental Sec. 2.1.

4.1.2 BRDFs. While the RGB textures varied in space, but not in angle, we now look into visual appearance varying with angle, but not over space, the classic BRDF representation task. We use a network the learn the BRDF responses for given light- and view-directions. Our experiments follow Sztrajman et al. [2021] and Hu et al. [2020], who both use networks combined with custom parametrizations to encode the MERL [Matusik 2003] BRDF database. Details on related work and the architectures used are found in Supplemental Sec. 2.2.

4.1.3 Stationary svBRDFs. The next-higher level of complexity are stationary spatially varying BRDFs (SVBRDFSTAT) that combine spatial and angular variation of reflectance, as also surveyed by Guarnera et al. [2016]. The theme recurs: optimization is slow but matches the target well, while feed-forward networks are fast, but often do not reproduce the target.

Specifically, we study estimating stationary svBRDFs from flash images, pioneered by Aittala et al. [2016], denoted SVBRDFSTAT in Supplemental Tab. 2. We look at a design using a noise-conditioned encoder-decoder, as demonstrated in Henzler et al. [2021]. We show re-lit results, parameter maps and all network details and training routines in Supplemental Sec. 2.3.

4.1.4 Non-stationary svBRDFs. Besides stationary svBRDFs, we look into estimating non-stationary ones (SVBRDFNONSTAT), also from flash images. This task was explored by Deschaintre et al. [2018] as well as Guo et al. [2020] and Gao et al. [2019] before. They all combine learning with fine-tuning in different ways. While Deschaintre et al. [2020] use additional information and upsampling, Gao et al. [2019] and Guo et al. [2020] optimize first in a latent space, and later in the pixel space given only the target flash image.

We adapt the architecture from Deschaintre et al. [2018], an encoder-decoder with a re-rendering loss, trained supervised under L_1 on synthetic flash images, svBRDFNonStat from Supplemental Tab. 2. For testing, both the reference as well as the inferred results are rendered from a set of novel view and light directions and compared.

4.1.5 Illumination. While the previous applications have looked into different forms of reflectance, another important application for visual appearance is estimating Illumination. To study the relation to meta-learning, we consider the task of representing natural spherical illumination itself as a Neural network (NN). In particular, we consider an encoder-decoder that takes as input a diffuse shaded Low-dynamic range (LDR) image of a sphere and outputs the High-dynamic range (HDR) environment map. Training data is rendered using the Laval HDR environment map dataset [Gardner et al. 2017] to illuminate spheres of random materials. For evaluation, we render a second scene under the reference- as well as the inferred illumination and compare both results. Details are found in Supplemental Sec. 2.5.

4.1.6 Light transport. The ultimate explanation for visual appearance is the light transport in a scene itself [Veach 1998]. To this day, robust handling of all forms of light transport remains a challenge [Keller et al. 2020]. One important building block that recently received a lot of attention is the *guidance* of paths [Herholz et al. 2016; Lafortune and Willems 1995; Müller et al. 2017, 2019; Vorba et al. 2014; Zheng and Zwicker 2019; Zhu et al. 2021], where previous paths build a model (parametric or NN-based) that is then used to steer path generation towards relevant paths that reduce variance more effectively. As this strategy involves optimization, it can also be meta-learned, so as to transfering understanding of light transport across scenes.

To do so, we study the architecture of Zheng and Zwicker [2019], which relies on a normalizing flow [Rezende and Mohamed 2015] to learn a map from the unit hypercube to Primary Sample Space (PSS), such that path density matches the one of the target scene. In other words, the normalizing flow learns a scene-dependent PSS warp that is then applied to random PSS samples. Once trained, the model can be used to generate new paths as well as their probability. For details on architecture and training, please see Supplemental Sec. 2.6.

While previously these methods were applied to a single scene, we consider an entire set of scenes. To study this, we use a Cornell-box like configuration that is populated by a random number of between 1 and 5 spheres of random diffuse, glass or metal materials in random positions, a mirror that is randomly placed at a sidewall, and an area light positioned randomly on the ceiling. To quantify success, we measure the Negative Log-Likelihood (NLL) across the test-set, as is done in [Zheng and Zwicker 2019], as using image-based metrics to quantify training error would require rendering the entire test-set per training epoch and method, which is computationally intractable. If the model matches the scene-dependent radiance distribution well, i.e., it correctly adapts to the light transport within the scene, the resulting NLL will be low. We report image-space metrics with the trained models for an equal number of rays in Tab. 3.

4.2 Methodology

Protocol. We compare our method against the approaches listed in Sec. 3. The protocol is as follows: Let $\mathcal{I}$ denote the entire training data set, e.g., all BRDF samples in the MERL database. For `Overfit`, we sample a single input I from $\mathcal{I}$ and train a network on I, and only I, until convergence. `General` denotes a network that has been trained on all elements in $\mathcal{I}$ and then is conditioned on the particular I we want inference for. `Finetune` applies a pre-defined number of n fine-tuning steps to the output of `General` to further improve the result for a particular I. Finally, our proposed `Meta`-learning is trained on all tasks in $\mathcal{I}$, but under the constraint of being given only a fixed budget of n_l gradient descent steps, with $n_l \ll n$. At inference time, a model conditioned on a particular I can then quickly be instantiated by updating the initial meta-model parameters n_l times.

Timing. All experiments report inference time, i.e., the time elapsed between first presenting an input to the network and receiving its final output. We refrain from reporting render- or path-tracing times, as these do not change across methods. Please note that for `Meta`, inference time is different from (meta-) training time: meta-inference is quick, as we only need to perform a small number of gradient steps and do not need to calculate costly higher-order derivatives. We report this figure, as this is what a user would experience when presenting new, unseen inputs to one of our meta-trained applications. During meta-training, however, the opposite is true: we often need to backpropagate though the gradient operator itself, and do so for many examples. This leads to long meta-training times: `Meta` trains roughly twice as long as `General`. For a more detailed listing of training times, cf. Supplemental Tab. 1. Note that for inference, the speed and memory consumption of the deployed NN is unaffected by meta-learning.

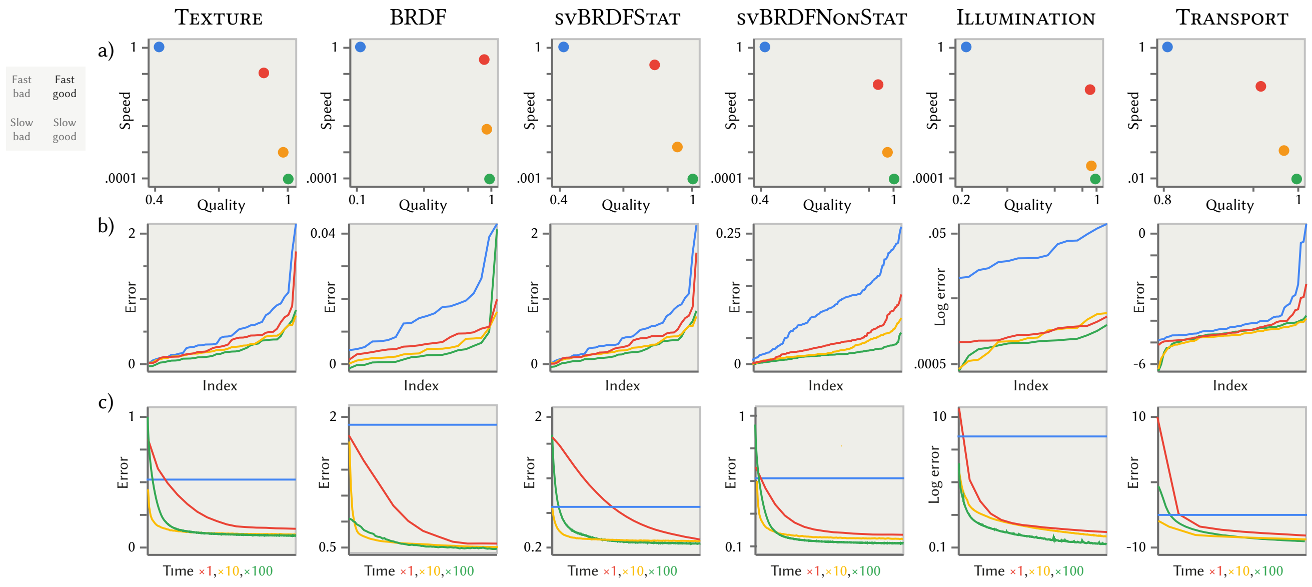
Metrics. We evaluate each method on unseen input from the test set, with the particular evaluation metric depending on the application (column "Metric" in Supplemental Tab. 2). In particular, meta-learning does not "cheat" by disclosing any test data during training; the split is the same as in conventional training. This means that `Meta` is presented with entirely new tasks (e.g., a completely unseen BRDF) instead of just withheld samples from a previously processed task (Supplemental Fig. 1 visualizes this).

4.3 Results

We summarize quantitative results in Tab. 2. We consistently show lower compute time than `Finetune` and `Overfit` at only slightly reduced quality. The speed-quality plots (Tab. 2, a) show that our method is not just a compromise between the speed of `General` and the quality of `Overfit`, but instead is located much closer to the ideal range (top right corner) than all other methods. We will now discuss each application's results in turn.

4.3.1 Textures. Tab. 2, b shows distribution of error across the texture task for all exemplars. We see that while both `General` and `Meta` struggle more with some specific (not necessarily the same) problem cases, the result is not dominated by outliers. The progress

Table 2. Quantitative results for all methods on all applications. The first row of plots shows the quality-speed continuum spanned by the four methods. The ideal range (fast inference *and* high quality) is in the top right corner. The second row shows test-set error, individually sorted for each method. The third row shows convergence plots at inference time. Note that very different time-scales are plotted on the same horizontal range, so comparison can only be made in shape, not between fixed values at any point in time.



	Texture Error	Texture Time	BRDF Error	BRDF Time	svBRDFStat Error	svBRDFStat Time	svBRDFNonStat Error	svBRDFNonStat Time	Illumination Error	Illumination Time	Transport Error	Transport Time
`General`	0.522	0.022	1.892	0.005	0.436	0.201	0.540	0.040	6.536	0.002	-3.457	0.040
`Overfit`	0.183	212.421	0.631	141.201	0.229	516.515	0.099	182.292	0.168	133.583	-4.120	362.210
`Finetune`	0.201	21.210	0.654	9.972	0.256	103.675	0.141	57.221	0.313	26.812	-4.030	42.070
`Meta`	0.252	0.619	0.720	0.031	0.311	1.484	0.197	0.474	0.352	0.384	-3.970	0.486

of training is seen in Tab. 2, c, where `Meta` achieves a quality more similar to `Finetune` than to `General`, but in a fraction of the time.

Fig. 4 shows qualitative results that further confirm these quantitative findings. `General` often projects the unseen textures into the latent space only approximately, which results in subtle but noticeable differences in features, color and scale. `Finetune` and `Overfit` both almost perfectly replicate the original, as both methods conduct a complete optimization run on the current sample. Our `Meta`-method faithfully replicates all textures with only minor differences in style. For more results, please cf. the supplemental.

4.3.2 BRDFs. Our `Meta`-method achieves high fidelity reproduction results across all BRDFs in the MERL-database, as quantified in Tab. 2. When rendered under Paul Debevec's St. Peter's Basilica illumination, `Meta` achieves Structural Similarity (SSIM) values of ≥ 0.95 on 99% of all materials. For a visual comparison of the reproduction results of the different approaches, cf. Fig. 5: Our method picks up fine nuances in the BRDF correctly, and we consistently show large improvements over `General`. In some cases, `Meta` even outperform methods `Finetune` and `Overfit`, which both have a time budget several orders of magnitudes larger than our method.

4.3.3 Stationary svBRDFs. We detail quantitative results in Tab. 2 and show qualitative results in Fig. 6. This application again confirms our previous findings: `General` is fast, but fails to match the input accurately. This becomes evident in Fig. 6, where `General` broadly matches the target, but is missing fine details and has slightly tinted colors (top and bottom row) or washed-out highlights (middle rows). Both `Finetune` and `Overfit` match the target well and pick up subtle details such as the wood grain and shading cues correctly, but take long to converge.

Our `Meta`-method achieves similar visual quality in a fraction of their runtime and manages to produce correct svBRDF maps in just over a second. We believe that a reason for this is the combination of strong priors built during meta-training (cf. Fig. 11, left column) and learning how to adapt them optimally. While the quality-speed improvement is still significant, the gain from using `Meta` here is comparatively small, as seen from the position of the red-dot in the quality-speed continuum.

4.3.4 Non-stationary svBDRFs. Qualitative results for non-stationary svBRDFs are shown in Fig. 7. We see that `General` is producing highlights not present in optimization-based training schemes, including ours. Out of those optimization-based methods, ours is several orders of magnitude faster, as seen in the numbers and the speed-quality plot in Tab. 2, column "svBRDFNonStat". Both `Finetune` and `Overfit` perform well, although the visual comparison in Fig. 7 shows `Finetune` to perform slightly better. We presume that this

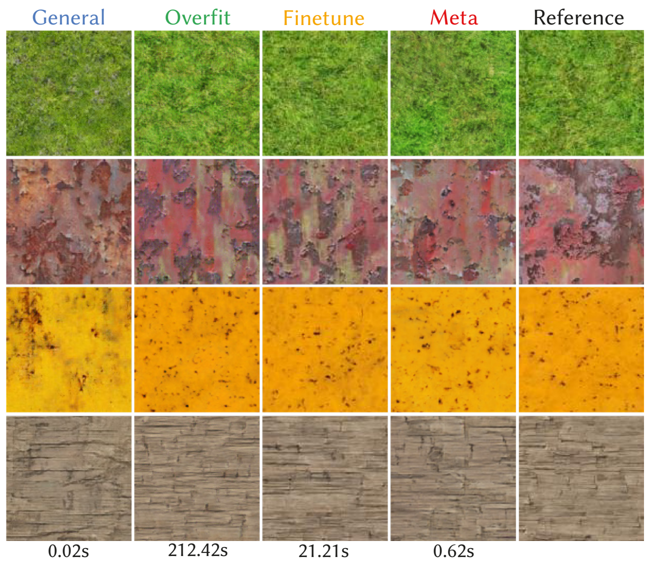


Fig. 4. Results across the test set for the Texture application. Every result is conditioned on a random process, so not meant to be compared pixel-by-pixel. We report inference time to the respective result.

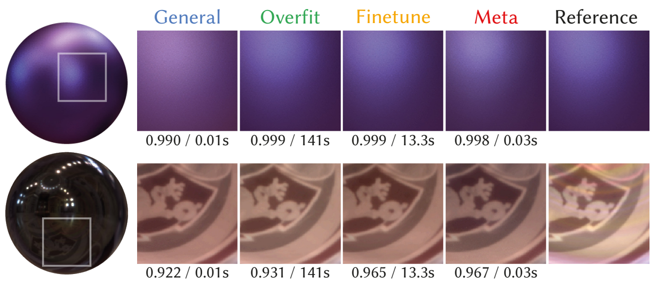


Fig. 5. Rendered results for unseen BRDFs from the test-set, trained with the different methods. The insets quantify SSIM and inference time.

is due to the fact that `Finetune` benefits from the priors developed by `General` (recall, fine-tuning starts at the output of the general model), whereas `Overfit` starts the training from scratch. In heavily ill-posed tasks like svBRDF estimation, the importance of solid priors has been shown to be of great importance for the optimization (cf. [Gao et al. 2019; Guo et al. 2020]). This is also part of the explanation for `Meta`'s success on this task, as it can build priors over the dataset in the outer loop *and* perform a quick overfit-optimization in the inner loop without diverging too far.

4.3.5 Illumination. We present results for our Illumination task in Fig. 8, where we render the inferred envmaps on a scene with a specular and diffuse object. We compare all methods against a reference image of that same scene rendered under the groundtruth illumination. We note how `General` is able to place sharp shadows (indicating it handles HDR well) but does not manage to exactly match the intensity. Similarly, reflections look plausible, but do not match the reference. Optimization-based methods meet this requirement, but only our meta-trained approach is orders of magnitude faster and achieves comparable quality. This is confirmed by the quality-speed plots in Tab. 2, where `Meta` is far-right, indicating that quality is very close to the full optimization-based methods. Tab. 2, c shows, that `Meta` converges even faster than for other tasks (the red curve is more concave). From the distribution in Tab. 2, b we see that the classic optimization-based methods have no problematic outliers, a property retained by `Meta`.

4.3.6 Light transport. We show the outcome for Transport in Fig. 9. Recall that Transport uses a resampling of a scene's radiance distribution to learn a model that is used for importance sampling that particular scene. To compare the effectiveness of each approach, we render a novel scene (unobserved during training for `General` and during meta-training for `Meta`) using samples generated by the respective importance-sampling model. We further include an

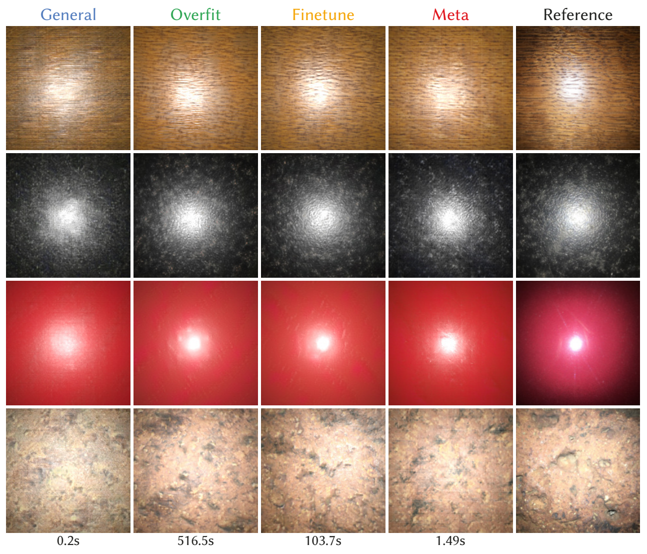


Fig. 6. Results across the svBRDFStat test-set. Note that every result is a realization of a random process, so not meant to be compared pixel-by-pixel. For re-lit renderings and shading maps, cf. the supplemental.

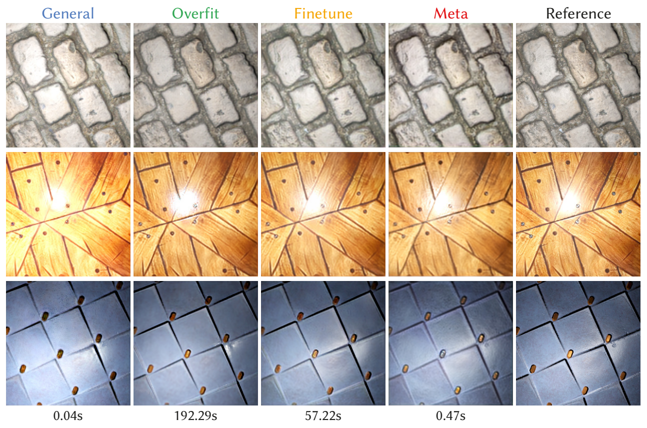


Fig. 7. Relighting results across the test-set for the svBRDFNonStat task. We render the resulting parameter maps under a different view- and light angle. For more results and the parameter maps, cf. the supplemental.

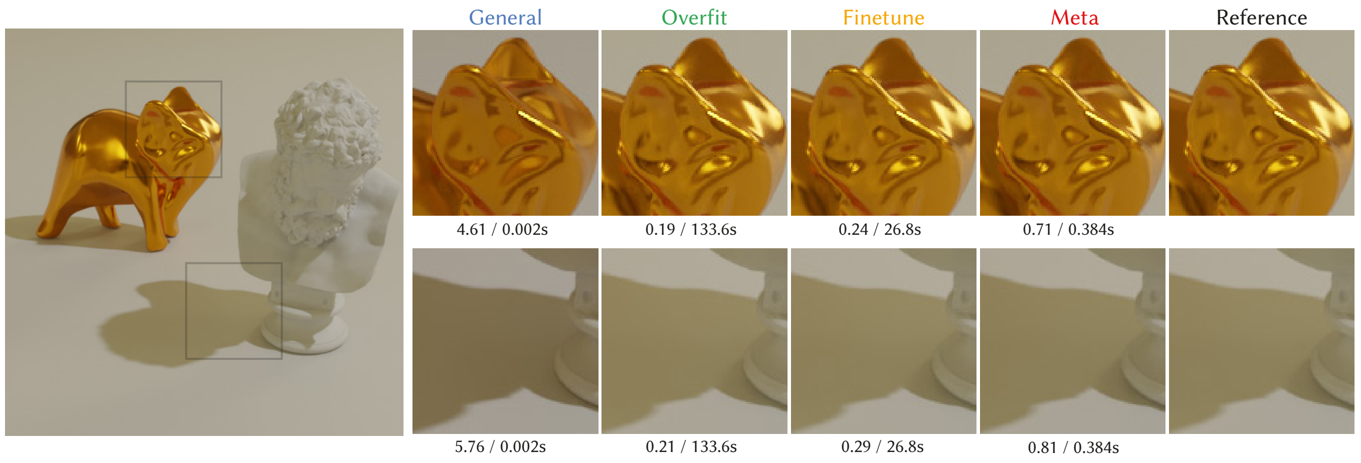


Fig. 8. Results for an unseen instance from the Illumination test-set, inferred from a single RGB image (not shown) and used to render a novel scene (left). Quality of illumination is most revealed in reflections (top) and cast shadow (bottom). The sharpness and shape of the shadows is very indicative of the high dynamic range of the regressed envmap. We report MAE $\times 10^2$ and inference time. For direct visualization of the envmaps, cf. the supplemental.

additional baseline, **Regular**, for this application. **Regular** uses importance-sampling for the geometric term and randomly samples outgoing paths from the hemisphere oriented around the normal. All subfigures are rendered with the same number of samples (4096) and very similar compute time, as querying the importance model can be parallelized on the GPU and hence is fast compared to the tracing of rays (milliseconds vs. minutes). As elaborated earlier (cf. Sec. 4.1.6), we report the time between model instantiation and the final training step (i.e., when it is ready to produce light path samples) as this is the part that the choice of training scheme can influence, whereas the subsequent path-tracing time is approximately invariant[3] to the origin of the samples. For a comparison of ray-generation and -tracing times, cf. Supplemental Tab. 4.

Quantitative results are seen in Tab. 2, column Transport: **Meta** is again closest to the top-right corner, indicating that it can combine the quality of **Overfit** and the speed of **General**. The qualitative outcome (Fig. 9) indicates that all methods successfully reduce variance w.r.t. the **Regular** baseline. Again, **General** performs slightly below the iterative optimization-based methods, and again, **Meta** is orders of magnitude faster. To quantify how well the instantiated models perform in image space, we rendered the entire test-set with light paths produced by the respective importance model and a fixed sampling budget of 1024 rays per pixel. We display common metrics calculated on these renderings in Tab. 3. The results confirm the qualitative inspection in Fig. 9 and show that **Meta** again is much closer to **Overfit** and **Finetune** than to the general or regular baseline. For details on the sampling and rendering operations, we again refer to Supplemental Sec. 2.6.

Please note that we explicitly refrain from discussing which method of importance sampling or path guiding is most appropriate for practical applications and re-iterate that we compare ways of *training* approaches instead of directly comparing the performance of different approaches. We here introduce meta-learning to the importance-sampling and rendering community as a first proof of concept and show that a meta-importance-sampler can generalize across a distribution of Cornell box-like scenes with practical benefits. To our knowledge, this is the first application of meta-learning in rendering, and the first presentation of meta-learned normalizing flows.

[3]We write approximately as the PSS samples created by all *trained* methods are created in Python and must be passed to the C++ renderer, which incurs a time overhead that the **Regular** baseline does not suffer. However, this is in the order of milliseconds, and hence can be neglected.

Table 3. Mean absolute percentage error and Structural Dissimilarity (DSSIM) across the Transport test-set. Lower is better for both metrics.

	Regular	**General**	**Overfit**	**Finetune**	**Meta**
MAPE	0.516	0.471	0.405	0.409	0.422
DSSIM	0.546	0.479	0.418	0.425	0.430

5 ANALYSIS

As the previous section has shown, **Meta** achieves similar quality than optimization-based approaches that take orders of magnitude more training time. To analyze the inner workings of Metappearance, we will next discuss a range of further properties that can be deduced from our experiments: We will ablate our learned components (Sec. 5.1), look at convergence of the inner and outer loop (Sec. 5.2), explain how Metappearance can be interpreted as model compression (Sec. 5.3.1) and finally discuss how **Meta** can make do with much fewer input observations (Sec. 5.3.2).

5.1 Ablations

We meta-learn two hyper-parameters: step size and inititalization, but which of them actually contributes to the success? Fig. 10 looks into this question. In column a), we display the output of **General**. In column b), we take this as starting point and use our learned step size to perform $n_1 = 20$ "smart" gradient steps towards the reference. Evidently, this leads to inferior results, as the general model has been trained with a fixed, global learning rate, and hence does not

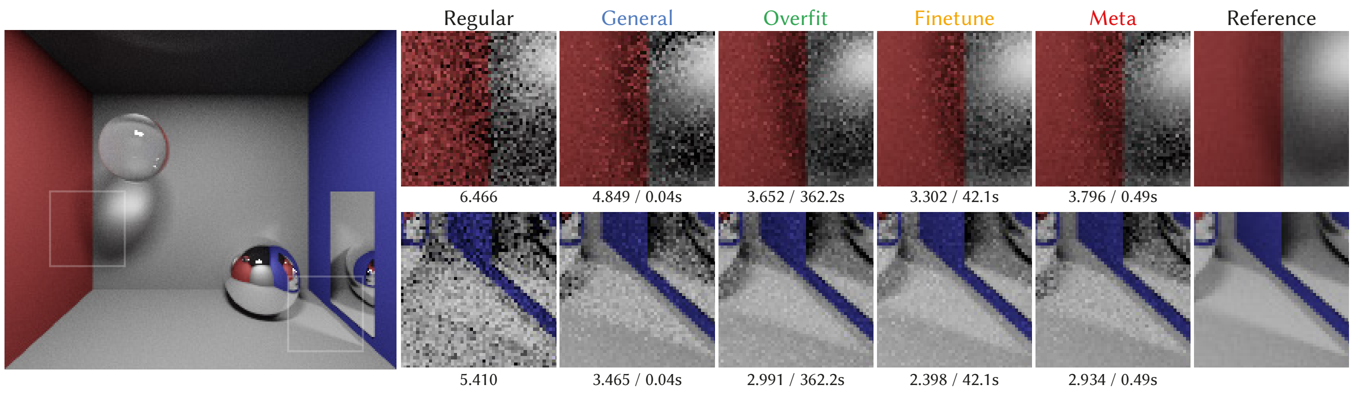


Fig. 9. Results for an unseen test-scene for the Transport application. The left image is rendered with 240,000spp to guarantee a noise-free reference. The images to the right are produced by rendering the reference scene with PSS samples produced by our different approaches (equal number of samples, i.e., equal rendering time). We report symmetric mean absolute percentage error (SMAPE) and the respective model inference time.

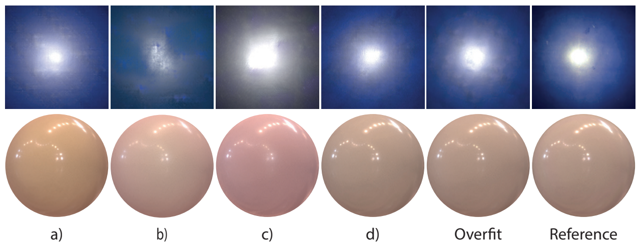


Fig. 10. We compare the influence of a learned learning rate (column b) and initialization (column c). Cf. the main text for details.

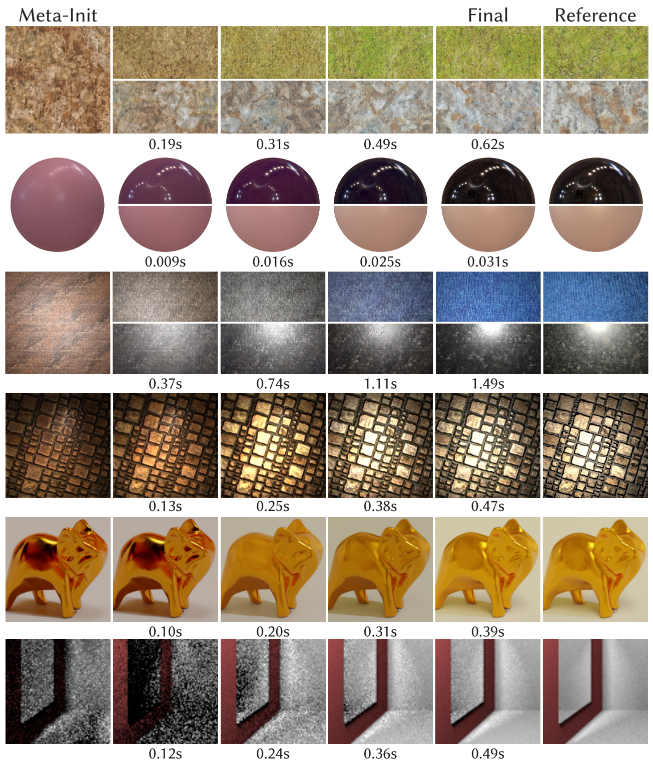


Fig. 11. Convergence on unseen test-tasks from the meta-init (left) to different targets (right) for our different applications. We report wall-clock inference time and results after approx. 25%, 50% and 75% (columns 2 - 5) of the inner-loop steps. Note that the init itself is a plausible instance and enables the optimization to "branch" to specific, very different goals. The result for Transport is an equal-spp rendering with the inferred importance model. Note how the noise clears with more meta-iterations although the sample count stays the same.

know how to account for a per-parameter learning rate. In column c), we use our meta-learned initialization to perform $n_l = 20$ steps of conventional Adam optimization (learning rate multiplied by 10 for faster convergence) towards the reference. This again leads to poor results, as our learned initialization normally is adapted through large, non-uniform gradient steps. During meta-training, the initialization was hence moved to a region of the objective space that is approximately equally well-suited for all tasks, but not necessarily easy to navigate with uniform gradient steps. Column d) finally shows the output of our Meta-method, where learned initialization and step size are used in combination. Evidently, this outperforms all alternative configurations.

This decay in reproduction quality shows that meta-learning really combines the best of both worlds: By optimizing for optimization directly, the outer optimizer can discover gradient paths that lead to local minima by not only moving the network weights (as would General), but also the step-size with which these weights are updated for a certain number of iterations (as in Finetune).

It is tempting to argument that optimizing over optimization itself, e.g., learning the step size, automates time-consuming hyper-parameter searches. While this is true to a certain extent, one still must choose the *meta*-hyperparameters, e.g., outer loop learning rate, etc. All our experiments use very similar hyper-parameters (cf. Supplemental Sec. 1) that were not particularly tuned, but this might be different in different applications or designs (cf. [Antoniou et al. 2018]).

5.2 Convergence

In Fig. 11, we show the convergence of our meta-learned initialization (leftmost column) towards different targets. All intermediate outputs show realistic appearance, and even the meta-initialization could pass as a problem instance (e.g., a texture) on its own. Throughout optimization, our method does not introduce unwanted artifacts, even for the ambiguous single-image svBRDF estimation task.

For maximal quality, we can fine-tune a converged Meta model for an additional number of training steps. We explicitly refrained from doing so in our main experiments, as this defies the purpose of Metappearance (achieving high quality *without* fine-tuning). However, Meta will converge to a loss difference of less than 1 % relative to Overfit in only 65 additional training steps (less than 5 seconds on every application). Related, General will not improve from further general training and has been trained to saturation already.

However, it is not our objective to claim superior quality for infinite time and compute resources. Instead, let us consider what often is the case in applications that involve user interaction: operating under the constraint of a finite time budget. Imagine, for instance, an architect wanting to quickly add a real-world svBRDF to his 3D model; imagine an app instantaneously adding a customized deep visual appearance model to a Tik-Tok video. The results of these applications must not only be highly accurate, but also available within split seconds to keep the user engaged, and no other method comes close to ours in this quality-speed trade-off. We hence claim highest quality under the constraint of limited time, and additionally investigate an equal-time comparison between Finetune and Meta, where both approaches are allowed to perform the same number of gradient steps n_I that would normally be used during meta-inference. This effectively is a very quick fine-tuning session, which is why we refer to it as "QuickFinetune", or **QuickFT**[4]. **QuickFT** uses Adam, for which we again increase the learning rate for faster convergence, as we have done for almost all applications in our experiments (the **QuickFT** config used here is the same as in Supplemental Tab. 1), while Meta uses both its learned init and the learned stepsize.

As Tab. 4 shows, **QuickFT** already offers great gains over the general method on some applications. However, it is outperformed by Meta on all applications. For example-specific visualizations of this experiment, including equal-time comparisons to Overfit, cf. Supplemental Fig. 6. This confirms that meta-learning is more than mere general inference followed by fine-tuning and that optimizing over the optimization procedure itself (recall, we learn how to overfit a sample efficiently) really finds a non-trivial optimizer.

5.3 Compression and Efficiency

5.3.1 Storage. Our Meta-method can further be interpreted as a compression scheme. Consider a rendering or 3D-modeling application, e.g., Blender, that loads a pre-trained model's weights to create new, diverse textures for surfaces. With methods Overfit and Finetune, such an application would have to store an entire set of weights per texture to be generated (note that, in such a scenario, storing the fine-tuned decoder of method Finetune is sufficient), and then is restricted to synthesizing the pre-learned texture exemplars. One could alternatively store the heavy-weight general model, but then would either have to forego accurate high-quality synthesis or fine-tune the result, which is unsatisfying and time-consuming, respectively. With our proposed Meta-method, it is sufficient to store two sets of weights only (the model's weights, and the per-parameter learning rate) to achieve high-quality, diverse texture synthesis in interactive runtime, i.e., in less than a second.

[4]To be able to fine-tune, we need to run General once. We do not deduct this runtime from **QuickFT**'s time budget, as this makes the comparison stronger and also usually is a rather fast operation.

Table 4. Average error across the respective application's test-set for our QuickFinetune-experiment. For the metrics reported, please cf. Supplemental Tab. 2. For convenience, we repeat results for methods General and Meta from Tab. 2.

	n_I	General	QuickFT	Meta
Texture	15	0.522	0.285	0.252
BRDF	10	1.892	1.346	0.720
svBRDFStat	20	0.436	0.351	0.311
svBRDFNonStat	15	0.540	0.289	0.197
Illumination	15	6.536	5.240	0.352
Transport	8	-3.457	-3.551	-3.970

Similar arguments can be made for all the applications we have presented in this work. Let w denote the number of disk space required to store the model's weights in methods Overfit and Finetune (we omit General from this comparison as we are concerned with high-quality results only). The total amount of storage required for the efficient and exact synthesis of m textures then equals wm, i.e., one set of weights per exemplar. Our Meta-method achieves similar-quality results with *constant* storage requirement $2w$ (weights and per-parameter learning rate), and is not limited to pre-trained or fine-tuned texture exemplars but instead can quickly infer new, unseen exemplars with high fidelity. The compression factor our method achieves hence is $2m^{-1}$, which, in the case of our exemplar texture application with 500 exemplars, equals $1 : 250$.

We would furthermore like to point out that compression also is an inherent property of using neural networks on certain problem instances. An NBRDF (a BRDF encoded in a network, cf. [Sztrajman et al. 2021]), for instance, has a significantly smaller memory footprint than regular BRDFs (28 kB vs. 34.2 MB in the case of our Meta-NBRDF), so we additionally compress the original BRDF with a factor of approx. $1 : 2000$. However, we do not claim credit for this as a property of our method, but rather of the base approaches we meta-train. In fact, quite the contrary is true – our method requires double the storage of a single network (model weights and per-parameter learning rate). However, we believe Meta's ability to quickly converge to unseen tasks and the resulting, aforementioned compression arguments to outweigh this moderate increase.

5.3.2 Sample Efficiency. For methods that consume single data points, i.e., methods that use an MLP, there is a further compression argument that can be made. To do so, we would like to draw attention to the way Meta is trained.

Recall that in meta-learning, the inner loop performs a fixed (small) number of gradient descent steps towards the reference. Naturally, as with most recent optimization algorithms, Meta uses *stochastic* gradient descent, i.e., the inner-loop gradients are not calculated per-sample or across all samples, but rather over a randomly selected subset of all available samples. Evidently, as Meta only performs n_l inner loop steps, only n_l batches of size b are sampled.

Naturally, this process repeats many times during meta-training and hence eventually samples all data in a task (e.g., all samples in a BRDF). However, this is not the case during meta-inference: Recall that for meta-inference, we merely run n_l gradient steps on a new, unseen task. Evidently, this leads to Meta seeing only $n_l \times b$ samples of a task, while all its competitors have access to *all* the samples in a task – in the cases of Overfit and Finetune, even repeatedly. Nonetheless, Meta delivers quality that is very close to its competitors that have seen all data.

This is no surprise: The ability to make do with scarce data is a core property of meta-learning and has been amply explored in previous works [Al-Shedivat et al. 2021; Finn et al. 2017]. In Metappearance, this property could be useful in a number of ways: Imagine, for instance, a client-server architecture, where the server stores large amounts of data (e.g., a large set of scene-dependent Transport radiance distributions), and the client stores the neural representation that will be trained on samples of this data. In order to train a model on a specific radiance distribution, the server would have to transfer *all* of its samples to the client (let us ignore the considerable time cost of training or fine-tuning a network on this data for a second). However, following the above argument, we only need to transfer $n_l \times b$ samples to infer a converged instance of Meta, for which the inference time can *really* be neglected.

In summary, this higher efficiency leads to bandwidth savings of approx. 72.2% for the Transport application (a full dataset is 288,000 samples, Meta consumes only $8 \times 10{,}000 = 80{,}000$ samples). For the case of BRDF encoding, the resulting reduction in needed data transmission is even more drastic: Meta takes $n_l = 10$ inner loop steps with a batchsize of $b = 512$ and hence consumes 5,120 samples, whereas a full MERL BRDF, as is needed by all other methods, consists of $180 \times 90 \times 90 \approx 1.46 \times 10^6$ samples. Meta hence achieves a bandwidth saving of 99.6%.

6 CONCLUSION

We have used meta-learning for efficient and accurate appearance-reproduction on a variety of increasingly complex applications. Our model, Metappearance, provides users with results that qualitatively compare well to other training schemes which take orders of magnitude more training iterations or data. We have shown that Metappearance generalizes not only across problem instances of a similar nature, e.g., our variety of Cornell-box scenes, but can also be applied across applications. In terms of implementation effort, the additional code relative to a solution that already uses an existing optimization is small. In fact, as we have shown in Sec. 3.5, re-phrasing the loss function is sufficient.

The main point of our experimentation is that while we cannot yet have both perfect speed and perfect quality, we, in several cases, improve substantially over a mere trade-off between the two, as seen from the red dot in Tab. 2, a), which has moved much closer to the ideal top-right spot, where visual appearance reproduction aims to be. Directions for future research could include the application of Metappearance to even more complex light-transport algorithms or its extension to meta-learning the objective function or the sampling pattern itself, which would enable even higher accuracy for visual appearance reproduction.

ACKNOWLEDGMENTS

This work was supported by Meta Reality Labs, Grant Nr. 5034015. We also acknowledge Gilles Rainer, Alejandro Sztrajman and Philipp Henzler for proofreading.

REFERENCES

Jonas Adler and Ozan Öktem. 2018. Learned primal-dual reconstruction. *IEEE transactions on medical imaging* 37, 6 (2018).

Miika Aittala, Timo Aila, and Jaakko Lehtinen. 2016. Reflectance modeling by neural texture synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH)* 35, 4 (2016).

Maruan Al-Shedivat, Liam Li, Eric Xing, and Ameet Talwalkar. 2021. On data efficiency of meta-learning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1369–1377.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. 2018. How to train your MAML. *arXiv:1810.09502* (2018).

Steve Bako, Mark Meyer, Tony DeRose, and Pradeep Sen. 2019. Offline deep importance sampling for Monte Carlo path tracing. In *Comp. Graph. Forum (Proc. EGSR)*, Vol. 38.

Alexander William Bergman, Petr Kellnhofer, and Gordon Wetzstein. 2021. Fast training of neural lumigraph representations using meta learning. In *NeurIPS*.

Sai Bi, Stephen Lombardi, Shunsuke Saito, Tomas Simon, Shih-En Wei, Kevyn Mcphail, Ravi Ramamoorthi, Yaser Sheikh, and Jason Saragih. 2021. Deep relightable appearance models for animatable faces. *ACM Trans. Graph.* 40, 4 (2021).

Kristin J Dana and Jing Wang. 2004. Device for convenient measurement of spatially varying bidirectional reflectance. *JOSA A* 21, 1 (2004).

Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. 2018. Single-image svbrdf capture with a rendering-aware deep network. *ACM Trans. Graph. (Proc. SIGGRAPH)* 37, 4 (2018).

Valentin Deschaintre, George Drettakis, and Adrien Bousseau. 2020. Guided Fine-Tuning for Large-Scale Material Transfer. In *Comp. Graph. Forum*, Vol. 39.

Julie Dorsey, Holly Rushmeier, and François Sillion. 2010. *Digital modeling of material appearance.*

Alexei A Efros and Thomas K Leung. 1999. Texture synthesis by non-parametric sampling. In *ICCV*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. 2019. Deepview: View synthesis with learned gradient descent. In *CVPR*.

Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. 2019. Deep inverse rendering for high-resolution SVBRDF estimation from an arbitrary number of images. *ACM Trans. Graph.* 38, 4 (2019).

Marc-André Gardner, Kalyan Sunkavalli, Ersin Yumer, Xiaohui Shen, Emiliano Gambaretto, Christian Gagné, and Jean-François Lalonde. 2017. Learning to predict indoor illumination from a single image. *arXiv preprint arXiv:1704.00090* (2017).

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style. *arXiv:1508.06576* (2015).

Stamatios Georgoulis, Konstantinos Rematas, Tobias Ritschel, Efstratios Gavves, Mario Fritz, Luc Van Gool, and Tinne Tuytelaars. 2017. Reflectance and natural illumination from single-material specular objects using deep learning. *IEEE PAMI* 40, 8 (2017).

Darya Guarnera, Giuseppe Claudio Guarnera, Abhijeet Ghosh, Cornelia Denk, and Mashhuda Glencross. 2016. BRDF representation and acquisition. 35, 2 (2016).

Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. 2020. MaterialGAN: reflectance capture using a generative svBRDF model. *arXiv:2010.00114* (2020).

David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv:1609.09106* (2016).

Philipp Henzler, Valentin Deschaintre, Niloy J Mitra, and Tobias Ritschel. 2021. Generative Modelling of BRDF Textures from Flash Images. *ACM Trans Graph (Proc SIGGRAPH Asia)* 40, 5 (2021).

Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. 2020. Learning a neural 3d texture space from 2d exemplars. In *CVPR*.

Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. 2016. Product importance sampling for light transport path guiding. 35, 4 (2016).

Bingyang Hu, Jie Guo, Yanjun Chen, Mengtian Li, and Yanwen Guo. 2020. DeepBRDF: A Deep Representation for Manipulating Measured BRDF. 39, 2 (2020).

Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Tieniu Tan, and Jian Sun. 2019. Meta-SR: A magnification-arbitrary network for super-resolution. In *CVPR*.

Xun Huang and Serge Belongie. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*.

Yuchi Huo, Rui Wang, Ruzahng Zheng, Hualin Xu, Hujun Bao, and Sung-Eui Yoon. 2020. Adaptive incident radiance field sampling and reconstruction using deep reinforcement learning. *ACM Trans. Graph.* 39, 1 (2020), 1–17.

Bela Julesz. 1975. Experiments in the visual perception of texture. *Scientific American* 232, 4 (1975).

Kaizhang Kang, Zimin Chen, Jiaping Wang, Kun Zhou, and Hongzhi Wu. 2018. Efficient reflectance capture using an autoencoder. *ACM Trans. Graph.* 37, 4 (2018).

Kaizhang Kang, Cihui Xie, Chengan He, Mingqi Yi, Minyi Gu, Zimin Chen, Kun Zhou, and Hongzhi Wu. 2019. Learning efficient illumination multiplexing for joint capture of reflectance and shape. *ACM Trans. Graph.* 38, 6 (2019), 165–1.

Alexander Keller, Pascal Grittmann, Jiří Vorba, Iliyan Georgiev, Martin Šik, Eugene d'Eon, Pascal Gautron, Petr Vévoda, and Ivo Kondapaneni. 2020. Advances in Monte Carlo Rendering: The Legacy of Jaroslav Křivánek. In *SIGGRAPH Courses*. Article 3.

Alexandr Kuznetsov, Milos Hasan, Zexiang Xu, Ling-Qi Yan, Bruce Walter, Nima Khademi Kalantari, Steve Marschner, and Ravi Ramamoorthi. 2019. Learning generative models for rendering specular microgeometry. *ACM Trans. Graph.* 38, 6 (2019).

Alexandr Kuznetsov, Krishna Mullia, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. 2021. NeuMIP: multi-resolution neural materials. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (2021).

Eric P Lafortune and Yves D Willems. 1995. A 5D tree to reduce the variance of Monte Carlo ray tracing. In *EGSR*. 11–20.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. 2017. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv:1707.09835* (2017).

Wojciech Matusik. 2003. *A data-driven reflectance model.* Ph.D. Dissertation. Massachusetts Institute of Technology.

Maxim Maximov, Laura Leal-Taixé, Mario Fritz, and Tobias Ritschel. 2019. Deep appearance maps. In *ICCV*.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*.

Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical path guiding for efficient light-transport simulation. In *Comp. Graph. Forum*, Vol. 36.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural importance sampling. *ACM Trans. Graph. (Proc, SIGGRAPH)* 38, 5 (2019).

Addy Ngan, Frédo Durand, and Wojciech Matusik. 2005. Experimental Analysis of BRDF Models. *Rendering Techniques* (2005).

Javier Portilla and Eero P Simoncelli. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int J Computer Vision* 40, 1 (2000).

Lara Raad, Axel Davy, Agnès Desolneux, and Jean-Michel Morel. 2018. A survey of exemplar-based texture synthesis. *Annals of Mathematical Sciences and Applications* 3, 1 (2018).

Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. 2019. Neural BTF compression and interpolation. In *Comp. Graph. Forum*, Vol. 38.

Sachin Ravi and Hugo Larochelle. 2016. Optimization as a model for few-shot learning. (2016).

Danilo Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *ICML*.

Vincent Sitzmann, Eric R Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. 2020. MetaSDF: Meta-learning signed distance functions. *arXiv:2006.09662* (2020).

Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *arXiv:1906.01618* (2019).

Alejandro Sztrajman, Gilles Rainer, Tobias Ritschel, and Tim Weyrich. 2021. Neural BRDF Representation and Importance Sampling. In *Comp. Graph. Forum*, Vol. 40.

Shufeng Tan and Michael L Mayrovouniotis. 1995. Reducing data dimensionality through optimizing neural network inputs. *AIChE Journal* 41, 6 (1995).

Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. 2021. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*.

Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. 2020. State of the art on neural rendering. In *Comp. Graph. Forum*, Vol. 39.

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. 2016. Texture networks: Feed-forward synthesis of textures and stylized images.. In *ICML*.

Eric Veach. 1998. *Robust Monte Carlo methods for light transport simulation.* Stanford University.

Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-line learning of parametric mixture models for light transport simulation. *ACM Trans. Graph.* 33, 4 (2014).

Shaofei Wang, Marko Mihajlovic, Qianli Ma, Andreas Geiger, and Siyu Tang. 2021. MetaAvatar: Learning Animatable Clothed Human Models from Few Depth Images. *arXiv:2106.11944* (2021).

Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. 2021. NeRFactor: Neural Factorization of Shape and Reflectance Under an Unknown Illumination. *arXiv:2106.01970* (2021).

Quan Zheng and Matthias Zwicker. 2019. Learning to importance sample in primary sample space. 38, 2 (2019).

Shilin Zhu, Zexiang Xu, Henrik Wann Jensen, Hao Su, and Ravi Ramamoorthi. 2020. Deep kernel density estimation for photon mapping. In *Comp. Graph. Forum. (proc. EGSR)*, Vol. 39.

Shilin Zhu, Zexiang Xu, Tiancheng Sun, Alexandr Kuznetsov, Mark Meyer, Henrik Wann Jensen, Hao Su, and Ravi Ramamoorthi. 2021. Photon-Driven Neural Reconstruction for Path Guiding. *ACM Trans. Graph.* 41, 1 (2021).