Metappearance: Meta-Learning for Visual Appearance Reproduction - Supplemental

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

1 META-LEARNING

As slightly different variants of meta-learning are used for our different applications, we here detail their differences. We use different variants since, in order to compute the meta-gradients, one must backpropagate through backpropagation itself, which is a very compute-intensive process, as higher-order gradients (more specifically, the Hessian-vector product) must be calculated throughout the computation graph. To alleviate the computational burden this imposes, several Model-agnostic meta-learning (MAML)-variants that use first-order gradient approximations have been proposed. One of those algorithms that finds use in this work is First-Order MAML (FOMAML) [Finn et al. 2017], which approximates higher-order gradients by replacing the Hessian with the identity-matrix and hence updates the meta-objective with the most recent inner-loop gradient, with significant savings on GPU memory and compute time. In practice, this means that, while MAML directly optimizes over the single gradient steps that are taken to reach a solution, FOMAML approximates this high-dimensional gradient trajectory with the local gradient of its last vertex. FOMAML has been shown to produce results close to MAML on certain applications [Finn et al. 2017; Nichol et al. 2018], which is commonly attributed to the fact that ReLU networks behave almost linear in high-dimensional spaces [Goodfellow et al. 2014], which in turn implies that their derivatives do not carry much second-order gradient information and can be omitted without severe performance penalties. We also experimented with Reptile [Nichol et al. 2018], but observed no performance improvements. For the exact algorithm setup, please confer the following application subsections.

Note that using MAML is only compute- and memory-intensive during the meta-training phase: for inference, we run a mere gradient descent on the model parameters, and no additional overhead is incurred. We observed performance increase across all tasks and applications when also meta-learning the inner-loop learning rate, as proposed by Li et al. [2017]. We implement our meta-networks and competitors in PyTorch [Paszke et al. 2019] and Torchmeta [Deleu et al. 2019].

Fig. 1 shows the data splits used during each method's training. In `General`, test and train are split into disjoint sets, and the network generalizes over entire instances of the data. For `Overfit` and `Finetune`, the samples in a problem instance are split into disjoint sets, e.g., train- and test-angles of a Bi-directional Reflectance Distribution Function (BRDF). For `Meta`, we distinguish between a meta-train and a meta-test set (in the meta-learning literature, these are also called context- and target-set). During completion, the inner-loop samples from the meta-train-set, while its final performance *after* completion — on which the meta-gradients are calculated — is evaluated on data from the meta-test-set. However, this is *not* the data used to evaluate the final model at inference time. Instead, the same withheld test set as in `General` is used. We would like to emphasize that neither training nor meta-training sees test-data, ever, and we report all our experimental results exclusively on test data.

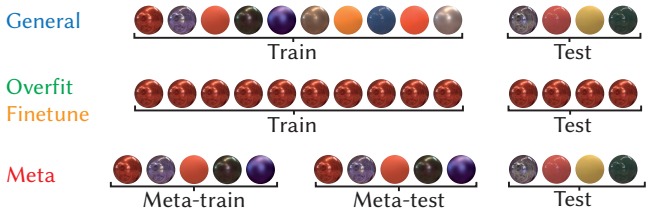

Fig. 1. Splitting of test and train for all four training types

2 NETWORKS AND IMPLEMENTATION DETAILS

We summarize the exact hyperparameters and algorithm setups in Tab. 1. The following sections shortly elaborate on the choice of models and approaches we use for the respective applications.

2.1 Textures

We use a U-Net [Ronneberger et al. 2015] Convolutional neural network (CNN) with residual skip connections and AdaIN blocks [Huang and Belongie 2017]. As in [Gatys et al. 2015], we optimize for the mean absolute error of VGG matrices between the reference exemplar and the network's output. The architecture is similar to Henzler et al. [2021] and we refer to their publication for the exact network details. We use this architecture as our method `Overfit` and empirically determined that 1000 training iterations are sufficient to replicate most textures faithfully.

Method `General` preprends a ResNet-based encoder (ResNet-50, [He et al. 2016]) to the aforementioned U-Net in order to project the input image into a latent space, from where the previously mentioned U-Net decoder, conditioned on the latent code $z \in \mathbb{R}^{64}$, reconstructs the exemplar's features.

Finetuning has been applied in recent publications to steer the output of a general model towards more accurate representations [Deschaintre et al. 2020; Guo et al. 2020; Henzler et al. 2021]. Note that, for fine-tuning, we increase the learning rate by a factor of 10, as in Henzler et al. [2021], to accelerate convergence, which makes `Finetune` a strong baseline.

For our `Meta`-method, we found a higher number of inner-loop steps to outperform the gains from second-order gradients, and hence use FOMAML with $k = 15$ inner-loop steps.

To show that we compare our meta-method against the best-configured competitors, we ran several experiments to empirically determine the best-suited optimizer and learning-rate setting. The

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.

Table 1. Algorithm setup and meta hyper-parameters for our experiments and the approaches we compare against.

	Textures	BRDF	svBRDFStat	svBRDFNonStat	Illumination	Transport
Publication	Henzler et al. [2021]	Sztrajman et al. [2021]	Henzler et al. [2021]	Deschaintre et al. [2018]	Georgoulis et al. [2017]	Zheng and Zwicker [2019]
Type	CNN	MLP	CNN	CNN	CNN	MLP
Rendering	—	Mitsuba Jakob [2010]	Cook-Torrance	Cook-Torrance	Blender	PBRT style
Input	RGB Image	MERL	Flash Image	RGB Image	RGB Image	PSS samples
Data	500 textures	Matusik [2003]	Henzler et al. [2021]	Deschaintre et al. [2018]	Gardner et al. [2017]	500 Cornell box scenes
Meta Algorithm	FOMAML	MAML	FOMAML	FOMAML	MAML	MAML
Gradient Order	First	Higher	First	First	Higher	Higher
Cosine Annealing	Yes	No	Yes	No	No	No
Meta-SGD	Yes	Yes	Yes	Yes	Yes	Yes
Meta-SGD Init.	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}
Meta-Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Meta-Optim. LR	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-5}	1×10^{-5}	1×10^{-4}
Weight Decay	—	1×10^{-6}	—	1×10^{-6}	1×10^{-6}	1×10^{-6}
Inner-Loop Steps	15	10	20	15	15	8
Meta-Batchsize	5	1	3	1	1	1
Meta Train Time	~ 2 days	~ 3 days	~ 4 days	~ 4 days	~ 2 days	~ 4 days
Meta Train Epo.	100,000	7.6×10^{6}	80,000	200,000	220,000	180,000
Time Meta-forw.	0.022	0.00117	0.0309	0.0158	0.01233	0.0296
Time Meta Step	0.06185	0.00309	0.0742s	0.0316	0.02566	0.0592
Time Meta-Inf.	0.6185	0.0309	1.484s	0.474	0.384	0.486
Iterations Overfit	1,000	83,000	5,000	2,000	5,000	8,600
Itera. Finetune	100	1,000	1,000	500	1,000	1,000
Batchsize	4	512	4	8	8	2,000
Latent Space Dim.	64	10	64	512	512	10
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Learning Rate	1×10^{-4}	5×10^{-4}	1×10^{-4}	1×10^{-5}	5×10^{-5}	1×10^{-4}
LR × Finetune	10	1	10	10	20	10
Weight Decay	1×10^{-5}	—	1×10^{-5}	—	—	—
Training Loss	L1 VGG	Log. MAE	Henzler et al. [2021]	Deschaintre et al. [2018]	MSE	FW KL Divergence

Table 2. Overview of all appearance reproduction applications we consider. Below, "ED" denotes encoder-decoder, "NF" is normalizing flow, "PN" is a Point-Net.

Application	Input I	Output L_θ	Domain $\mathbf{x}_i$	Architecture θ	Metric	Source
Texture	RGB image	Infinite RGB map	2D Pos	ED CNN+Noise	VGG Gram L_1	Ulyanov et al. [2016]
BRDF	Angle pair	RGB reflectance	4D Dir	CNN+MLP	L_1	Sztrajman et al. [2021]
svBRDFStat	Flash image	Infinite BRDF map	2D Pos, 4D Dir	ED CNN+Noise	VGG Gram L_1	Henzler et al. [2021]
svBRDFNonStat	Flash image	Finite BRDF map	2D Pos, 4D Dir	ED CNN	Re-render L_1	Deschaintre et al. [2018]
Illumination	RGBN image	RGB HDR Envmap	2D Dir	ED CNN	Re-render L_1	Georgoulis et al. [2017]
Transport	PSS sample	Light path + prob.	n-D Pos/Dir	PN MLP+NF	NLL	Zheng and Zwicker [2019]

results of these experiments are depicted in Tab. 2, while we show results on unseen test-data in Fig. 2.

2.2 BRDFs

Much work was devoted to create, efficiently compress, interpolate and re-sample (spaces of) BRDFs; for a general survey we refer to Guarnera et al. [2016]. Many applications revolve around representing a full BRDF in high quality from a small set of measurements. When these are taken in a suitable pattern [Nielsen et al. 2015], a linear basis found through Principal component analysis (PCA) can be used as an encoding. Recently, Rainer et al. [2019] showed that NN-based encoding of radiance data is a viable alternative to traditional, PCA-based methods. Moreover, Hu et al. [2020] and Rainer et al. [2020] encode multiple BRDFs and Bidirectional texture functions (BTFs) in a single network, respectively. Finally, Sztrajman et al. [2021] show that a compact two-layer Multi-layered perceptron (MLP) can learn the mapping between angular measurements and RGB reflectance and is able to faithfully reproduce BRDFs.

Table 3. Different learning-rate and optimizer comparisons for the baseline methods we compare against. We choose the best-performing optimizer, respectively.

	0.1	0.01	0.001	1×10^{-4}	1×10^{-5}
`Finetune`, Adam	—	0.246	**0.205**	0.209	0.309
`Overfit`, Adam	—	—	0.188	**0.183**	0.598
`Finetune`, SGD	0.643	0.327	0.289	0.416	—
`Overfit`, SGD	0.287	0.259	0.515	—	—

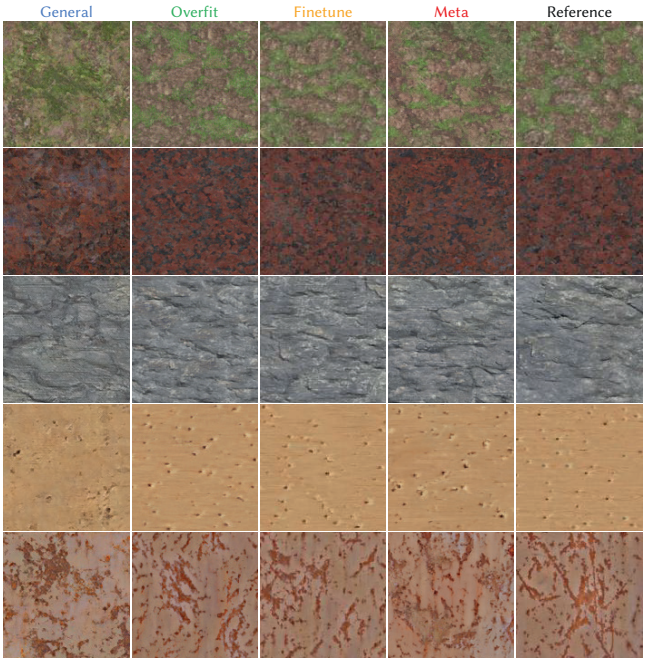

Fig. 2. Inference results for unseen test textures from the classes Grass, Marble, Rock, Wood, Rust (top to bottom).

To tackle the task of BRDF reconstruction, we use a simple two-layer MLP with 21 neurons per hidden layer, as described by Sztrajman et al. [2021], as method `Overfit`. The network receives randomly sample batches of light- and view-direction in Rusinkiewicz [1998] parametrization and then outputs the logarithmic RGB reflectance for these query directions. We train our competitor `Overfit` with the official codebase from [Sztrajman et al. 2021] with one network per MERL-material.

For `General`, we take inspiration from [Hu et al. 2020] and create a latent space of BRDFs by training a encoder CNN (for the exact details of the architecture we refer to their publication) that consumes the entire BRDF measurement of $180 \times 90 \times 90$ RGB triplets at once. For the encoder, we use the official code from Hu et al. [2020], which was kindly provided by the authors. To enable a fair comparison between all methods, we use the two-layer MLP of Sztrajman et al. [2021] as a decoder. For `Finetune`, we fine-tune the output

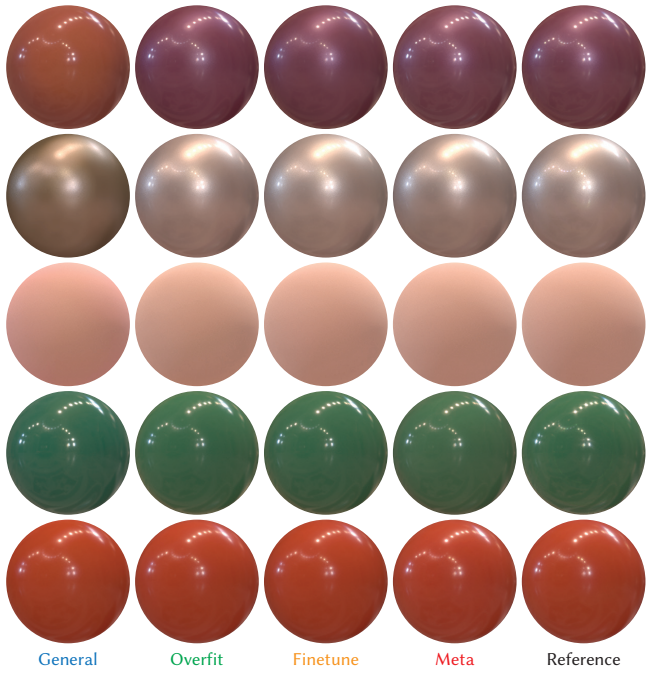

Fig. 3. Inference results for different test BRDFS from the MERL database.

of `General` for 1000 iterations, which amounts to seeing each measurement in the data twice. We found that increasing the learning rate had no visible benefits and hence did not modify it.

Our `Meta`-method uses the same base architecture as `Overfit` and [Sztrajman et al. 2021], a two-layer MLP with 21 neurons per hidden layer. We observed that incorporating second-order gradients into the optimization leads to more consistent results across all tasks and therefore use MAML with $k = 10$ inner-loop steps, which is made possible by the lightweight architecture (675 parameters in total). While using a meta-batchsize improved generalization for the previous task, we found that averaging the meta-gradients over batches here leads to a decrease in result quality and hence use a meta-batchsize of one. We use the classic 80%-20% train-test split for training, both for the MERL materials and the angular measurements within a MERL material, and show results on unseen test BRDFs in Fig. 3.

2.3 Stationary svBRDFs

Classically, spatially-varying BRDF (svBRDF) were acquired by optimization [Lensch et al. 2003] involving appropriate priors [Dror et al. 2001; Lombardi and Nishino 2012; Nam et al. 2018], optimization in a neural representation [Aittala et al. 2016; Liu et al. 2017] and finally methods that solve the task using a feed-forward network [Henzler et al. 2021]. Optimization can be performed in the pixel- [Aittala et al. 2016], or Neural network (NN) basis [Henzler et al. 2021]. We again use an encoder-decoder approach to solve this problem, similar to the one motivated by Henzler et al. [2021]. Their method uses an encoder-decoder architecture, where the encoder first projects a flash-illuminated image of a material into a latent space. The latent code is then used to condition the decoder,

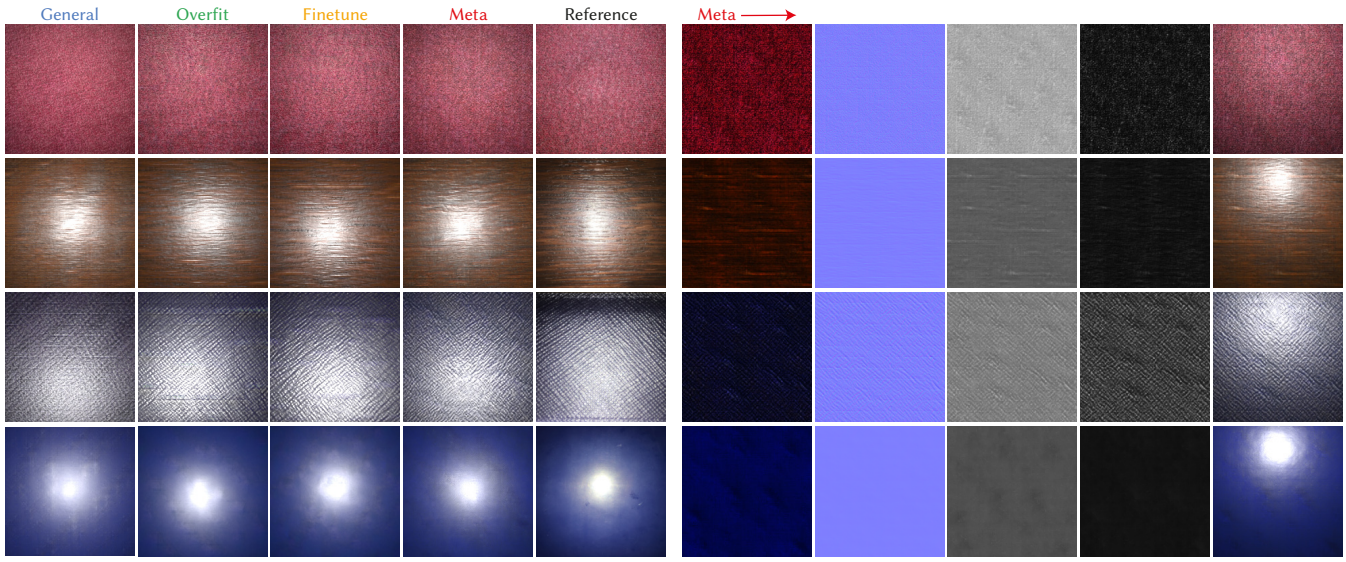

Fig. 4. Results for unseen test svBRDFs. On the right, we show the parameter maps produced by **Meta**: diffuse albedo, normals, roughness and specular albedo. We provide a top-light illuminated rendering with those maps in the rightmost column. We do not show the parameter maps of the other approaches for brevity and refer to [Henzler et al. 2021]. Note that all maps must be free of baked shading, as they are stationary by construction.

which is additionally provided with noise (for details, we refer to the publication) to then infer the svBRDF parameters. More specifically, the decoder outputs parameter maps of the Cook-Torrance [1982] reflectance model, i.e., diffuse and specular albedo, roughness and height, which is differentiated to a normal map. The generated svBRDF maps are then rendered by a differentiable renderer, assuming collocated camera and light, flat geometry and stationarity, as elaborated on by Aittala et al. [2016].

We implement method **General** with the public code provided by Henzler et al. [2021], without fine-tuning, and train until convergence. For **Finetune**, we follow the method proposed in [Henzler et al. 2021] and fine-tune the learned general decoder for 1000 epochs, with the learning rate increased by a factor of 10. For **Overfit**, we train the decoder of **General** only, as there is no need to span a latent space when overfitting only a single exemplar. We additionally let the model learn the rotation parameters of the flash highlight, which, surprisingly, does not always put the flash in the center. We attribute this to the fact that minimizing the error between gram matrices, which is one of the main parts of the training loss, matches exemplar statistics globally and discards information about relative spatial layout. Note that the ratio of fine-tuning to over-fitting is 20%, which is remarkable.

Due to GPU memory constraints, our **Meta**-method for this application uses FOMAML and limits training to the decoder, with $k = 20$ gradient steps and a meta-batchsize of 3. We observed slightly improved performance when using latent codes generated by a pre-trained encoder over constant latents, which can be interpreted as a data pre-processing step. We also experimented with lower amounts of inner-loop steps, but the results were not convincing. We attribute this to the ambiguity and under-constrained nature of the problem of estimating svBRDFs from a single image: While it may be relatively simple for a model to quickly learn RGB colors or statistics alone, inversely solving the rendering operation for the svBRDF-maps that created said colors or statistics is a much harder task. We note that it is not uncommon to use a large number of inner-loop steps when meta-learning on ambiguous problems: Tancik et al. [2021] use 64 steps on a Reptile-model to meta-learn an MLP that approximates neural radiance fields.

We show inference results for unseen test svBRDFs in Fig. 4. We further show the parameter maps produced by our method and confirm that they are free of baked shading with a re-lighting. This can also be proven by construction, as only non-stationary parameter maps can bake non-stationary shading into albedo. Stationary maps (like the ones used here) can, by construction, not bake-in non-stationary shading, as explained by Aittala et al. [2016] and Henzler et al. [2021]. Proving this, we have re-lit our results following the protocol in [Henzler et al. 2021] and achieved relit-errors (VGG Gram L1, lower is better) of 0.25 / 0.10 / 0.12 / 0.16 for **General**, **Overfit**, **Finetune**, **Meta**, respectively.

2.4 Non-Stationary svBRDFs

To estimate non-stationary shading parameter maps, we use the approaches presented by Deschaintre et al. [2018]. For the results produced by method **General**, we run their publicly available model implementation. For all other methods, we use a PyTorch-port of their original Tensorflow implementation. All methods use their proposed rendering-loss, which is crucial for accurate reconstruction. For **Overfit**, we run the network for 2000 iterations on randomly created scene configurations (as in the original publication, we use 3 diffuse and 6 specular scenes and aim to re-create all other settings as closely as possible). Method **Finetune** starts from **General**'s parameter maps and refines these, in equal fashion, for another 500 iterations. The ratio for fine-tuning to over-fitting thus is 25%, which

is remarkable. We have experimented with different learning rates and, similar to our previous experiments (e.g., Texture) found a learning rate multiplied by factor 10 to achieve fastest convergence. Even higher learning rates, e.g., 1×10^{-3}, lead the network to diverge and produce uniformly colored shading maps only. Lower learning rates, e.g., 5×10^{-5} require lots of training iterations (roughly 50% of `Overfit`) to achieve satisfactory performance, which somewhat defies the purpose of fine-tuning. `Meta` uses FOMAML as meta-learning algorithm, as the sheer size of the network (`General` has over 80 million trainable parameters) makes computing higher-order gradients through several inner-loop steps computationally intractable. We use $k = 15$ steps for the reported results. We alternatively experimented with running a lower-resolution version of the training process (128×128px) and a higher number of MAML inner-loop steps, but found this to de-stabilize training. For all methods, we train and evaluate on the publicly available data from Deschaintre et al. [2018].

2.5 Illumination

Prior work has predicted parametric or general illumination using optimization [Li et al. 2020; Park et al. 2020; Wei et al. 2020], particularly indoors [Gardner et al. 2019; Garon et al. 2019; Song and Funkhouser 2019; Weber et al. 2018] but also outdoors [Hold-Geoffroy et al. 2019], and even as a volume [Srinivasan et al. 2020; Wang et al. 2021].

In order to meta-learn this task, we train on the Laval HDR Dataset [Gardner et al. 2017], which provides a wide variety of illumination conditions. As the high spatial resolution of the provided envmaps quickly makes computation intractable, we use a down-sampled version of the dataset at 32×64 pixels. Our architecture for encoding illumination is inspired by Rematas et al. [2016] and similarly uses a U-Net-like encoder-decoder architecture with skip connections. Our encoder takes as input a 128×128 RGBN rendering of a sphere illuminated with the respective environment map. The down-branch extracts the image information through a cascade of 3×3 convolutions, all followed by ReLU activation and Batch normalization (BN). As in Gao et al. [2019] and Rematas et al. [2016], we restrict the use of BN to the encoder part and find this to achieve higher output fidelity than applying BN on the full architecture. To avoid checkerboard artefacts, we use bilinear upsampling in the decoder branch, followed by a zero-padded 3×3 ReLU-convolution, until we finally arrive at the original spatial resolution of 128×128, which we spatial-pool to the desired envmap resolution of 32×64. Additionally, we found it beneficial to let the network operate in log-space and also append a positional-encoded (6 encoding functions) coordinate grid to each input. We use this architecture for all methods.

Similarly to the previous applications, we train `Overfit` on one problem instance only, which it overfits in roughly 5,000 iterations. While over-fitting for more iterations is entirely possible and will result in a slight performance improvement, we found the performance-gain per increased time to diminish significantly after 5,000 iterations and empirically chose to stop the training then. Similarly, `Finetune` needs around 1,000 iterations to fully nudge the output of `General` to convergence. We experimented with several learning rate configurations for the fine-tuning experiment and chose to use the best-performing optimizer (Adam, learning rate 1×10^{-3}). Note that, as in previous experiments, we again increase the fine-tuning learning rate, which makes `Finetune` a strong baseline (please also cf. Tab. 1). For `Meta`, we run MAML with $k = 15$ inner-loop steps. Interestingly, this application required us to change the outer-loop learning rate for `Meta` from 1×10^{-4}, as in previous experiments, to 1×10^{-5}, to stabilize training. We attribute this to the high dynamic range of the envmaps and the consequently high values of the gradients. The Meta-SGD init of 0.001 did not require changing.

Table 4. Timing measurements for the Transport application in seconds.

	`Regular`	`General`	`Overfit`	`Finetune`	`Meta`
Prepare Rays	0.0s	4.1s	4.1s	4.1s	4.1s
Model Inference	0.0s	0.04s	362.2s	42.1s	0.5s
Trace Image	164.0s	164.0s	164.0s	164.0s	164.0s
Total	164.0s	168.1s	530.3s	210.2s	168.6s

2.6 Transport

To (meta-) learn the light transport in a scene, we use the method presented by Zheng and Zwicker [2019], where a normalizing flow is used to warp the renderer's Primary Sample Space (PSS) in order to produce PSS samples that reduce rendering variance. In this scenario, the normalizing flow can be interpreted as an importance model that learns to produce samples proportional to the scene-dependent radiance, and can then, once training is finished, be sampled from.

We re-implement the normalizing flow architecture described in Zheng and Zwicker [2019], which is a variant of RealNVP [Dinh et al. 2016]. We use 8 coupling layers, each of which consists of two residual-MLPs with 40 neurons per layer and batch normalization. As in Zheng and Zwicker [2019], we pre-train the model to achieve the identity warp and use the resulting weights as initialization for all further experiments to speed up convergence (note that this pre-training cost is excluded from all timings we report). Moreover, we found it beneficial to include an additional ActNorm layer [Kingma and Dhariwal 2018] and use it for all methods. The network is trained with batchsize 2000. To keep computation tractable, we limit PSS warping to $m = 2$, i.e., our model learns importance distributions for the first two bounces. For all subsequent bounces, we continue with randomly sampled rays. This is an established technique (cf. [Müller et al. 2019; Zheng and Zwicker 2019]), as later bounces contribute less to the final render and hence are not as amenable to importance sampling.

For our experiments, we use a PBRT-style renderer written in C++ and integrated into Python via pybind11. At inference time, the renderer *consumes* PSS coordinates created by the trained normalizing flow, whereas, for the data creation process, it is also able to *return* the PSS coordinates that have been used to render the scene. To generate the data for the corresponding PSS warps, we create a set of Cornell-box-like scenes (Fig. 8), with a random number

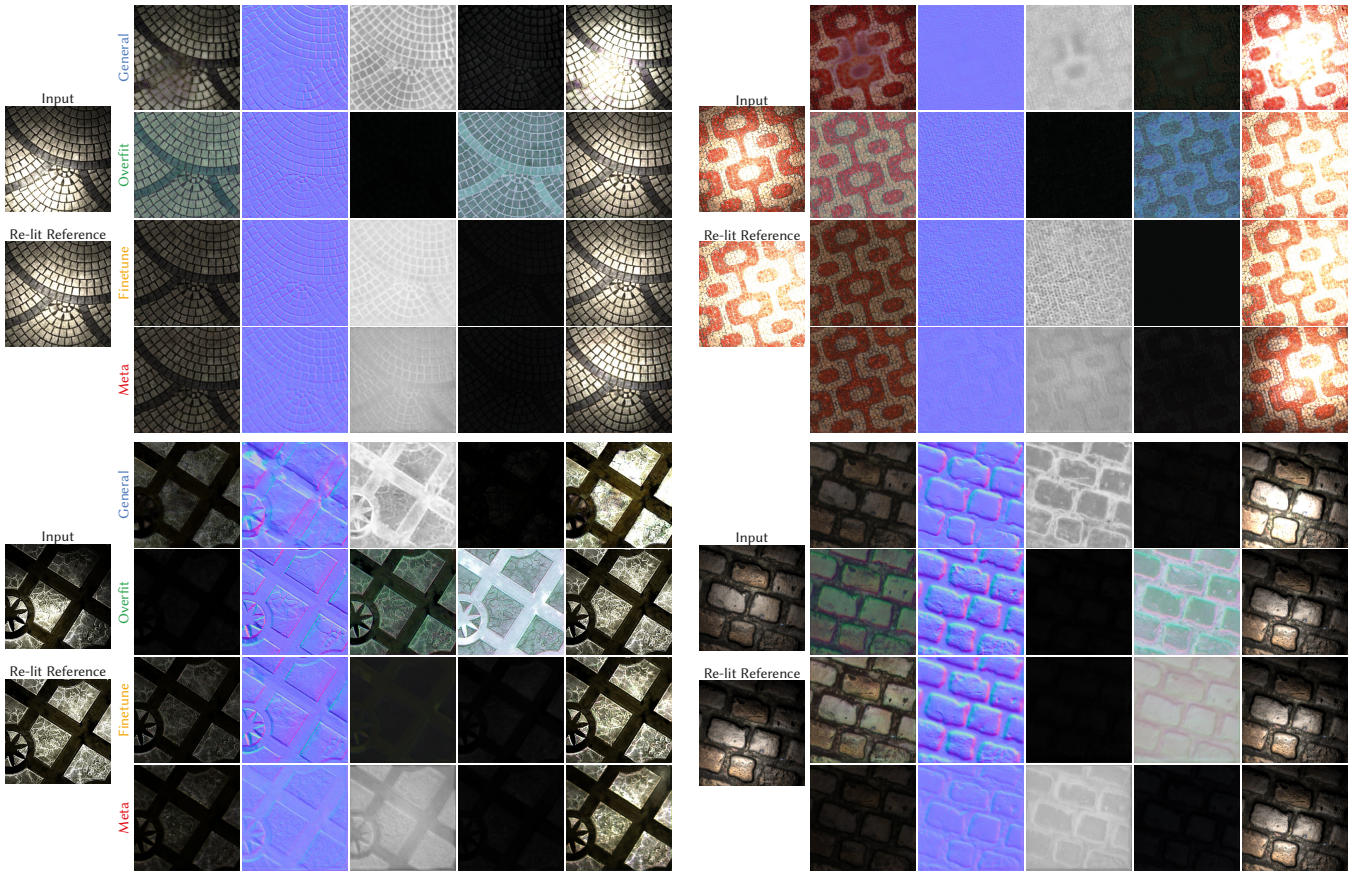

Fig. 5. Inference results for svBRDFs from [Deschaintre et al. 2018]. We show the shading parameter maps diffuse albedo, normals, roughness and specular albedo and a re-lighting. Note how `Overfit` outputs a high specular albedo and a reduced diffuse albedo, something that (mostly) doesn't occur in other methods. This is because `Overfit` cannot benefit from priors over the reflectance data and hence only adjusts the maps to fit the rendering.

of spheres (between 1 and 5) with random materials (diffuse color, reflective metal, refractive glass) and a random top-light configuration. We trace all these scenes at resolution 120×120 pixels (cf. Tab. 4 for timing) to ease computation, but ask the reader to note that path-tracing with the final flow model can be carried out at any resolution. We store all paths traversed during rendering and subsequently re-sample those that carry high throughput to achieve approx. 20 *epp* (examples-per-pixel, a SPP metric that originates from omitting the pixel-filter and is effectively the average spp, cf. [Zheng and Zwicker 2019]), which leads to a 6-dimensional dataset per scene (recall that PSS dimensionality is $2(m + 1)$) that consists of $120 \times 120 \times 20$ samples.

For `Overfit`, we adhere to the training guidelines published in Zheng and Zwicker [2019] and train each network for 60 epochs (this corresponds to approx. 8,600 gradient steps). As usual, overfitting produces one network instantiation per scene. For `General`, we want to be able to train a network that generalizes across scenes. To allow this, we prepend the previously discussed flow model with a PointNet (PN) [Qi et al. 2017] -like encoder that consumes the *already resampled* dataset. Using the resampled PSS coordinates as input effectively allows the encoder to focus on encoding and outsources the task of deciding which samples are important to a preprocessing stage that is equivalent for all methods. Our PN encoder uses three linear layers with 64, 128 and 512 neurons, respectively, batch normalization and pReLU activation units, and outputs a latent code on which we then condition the flow by concatenation. As in our previous experiments, `Finetune` again starts from the output of `General` and re-fines the estimated density for a total of 1,000 gradient steps. `Meta` is trained with eight MAML inner-loop steps and consumes batches of size 10,000. We found the higher batchsize necessary to stabilize meta-training with a higher number of inner-loop steps. Note that even after all inner-loop steps have been completed, `Meta` still has seen much fewer data samples (`Meta`: $8 \times 10{,}000 = 80{,}000$) than its competitors, that are presented with the entire dataset of approx. 288,000 samples.

REFERENCES

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

Robert L Cook and Kenneth E. Torrance. 1982. A reflectance model for computer graphics. *ACM Trans. Graph.* 1, 1 (1982).

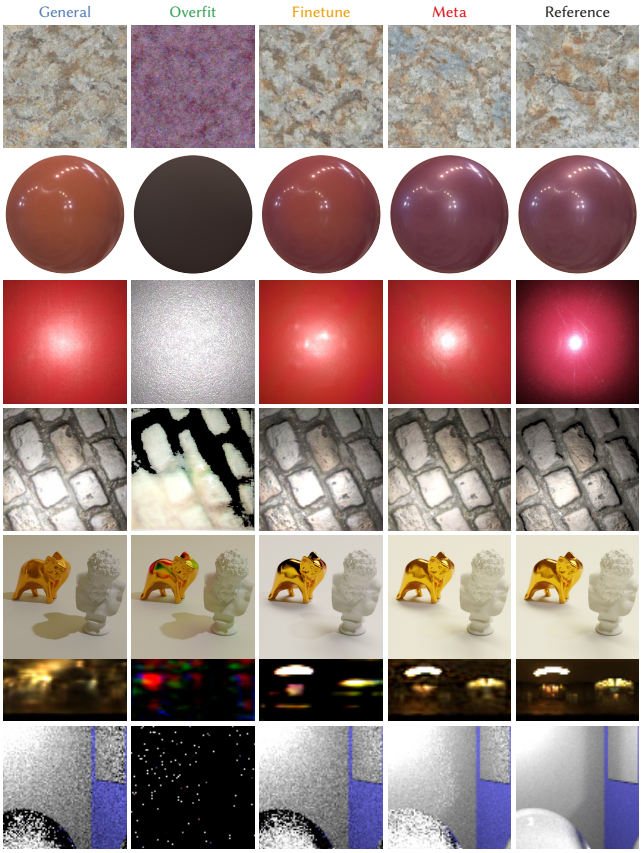

Fig. 6. Equal-time comparisons on unseen data from the test set for each application (top to bottom row: Texture, BRDF, svBRDF, svBRDFnonstat, Illumination including the regressed envmap, Transport). Methods **Overfit** and **Finetune** are ran with the same amount of gradient steps **Meta** uses, i.e., the same wall-clock time. As **Overfit** starts the training from scratch, it cannot move from the (random) model initialization in such a low number of gradient steps, which is why its output is unsatisfactory. **General** does not change during inference. **Finetune** moves the output of **General** towards the reference, but cannot achieve good quality in such few optimization steps. **Meta** encodes the reference best across all applications.

Tristan Deleu, Tobias Würfl, Mandana Samiei, Joseph Paul Cohen, and Yoshua Bengio. 2019. Torchmeta: A Meta-Learning library for PyTorch. arXiv:1909.06576

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.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2016. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803* (2016).

Ron O Dror, Edward H Adelson, and Alan S Willsky. 2001. Recognition of surface reflectance properties from a single image under unknown real-world illumination. *CVPR* (2001).

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

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, Yannick Hold-Geoffroy, Kalyan Sunkavalli, Christian Gagné, and Jean-François Lalonde. 2019. Deep parametric indoor lighting estimation. In *PICCV*.

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).

Mathieu Garon, Kalyan Sunkavalli, Sunil Hadap, Nathan Carr, and Jean-François Lalonde. 2019. Fast spatially-varying indoor lighting estimation. In *CVPR*.

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, Mario Fritz, Tinne Tuytelaars, and Luc Van Gool. 2017. What is around the camera?. In *ICCV*.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

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).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

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).

Yannick Hold-Geoffroy, Akshaya Athawale, and Jean-François Lalonde. 2019. Deep sky modeling for single image outdoor lighting estimation. In *CVPR*.

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

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

Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

Durk P Kingma and Prafulla Dhariwal. 2018. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems* 31 (2018).

Hendrik PA Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. 2003. Image-based reconstruction of spatial appearance and geometric detail. *ACM Trans. Graph.* 22, 2 (2003).

Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. 2020. Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image. In *CVPR*.

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

Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. 2017. Material editing using a physically based rendering network. In *CVPR*.

Stephen Lombardi and Ko Nishino. 2012. Reflectance and natural illumination from a single image. In *ECCV*. Springer.

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

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

Giljoo Nam, Joo Ho Lee, Diego Gutierrez, and Min H Kim. 2018. Practical svbrdf acquisition of 3d objects with unstructured flash photography. *ACM Trans. Graph.* 37, 6 (2018).

Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. *arXiv:1803.02999* (2018).

Jannik Boll Nielsen, Henrik Wann Jensen, and Ravi Ramamoorthi. 2015. On optimal, minimal BRDF sampling for reflectance acquisition. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 34, 6 (2015).

Jeong Joon Park, Aleksander Holynski, and Steven M Seitz. 2020. Seeing the world in a bag of chips. In *CVPR*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.

Gilles Rainer, Abhijeet Ghosh, Wenzel Jakob, and Tim Weyrich. 2020. Unified neural encoding of BTFs. 39, 2 (2020).

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

Konstantinos Rematas, Tobias Ritschel, Mario Fritz, Efstratios Gavves, and Tinne Tuytelaars. 2016. Deep reflectance maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4508–4516.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*.

Szymon M Rusinkiewicz. 1998. A new change of variables for efficient BRDF representation. In *EGWR*.

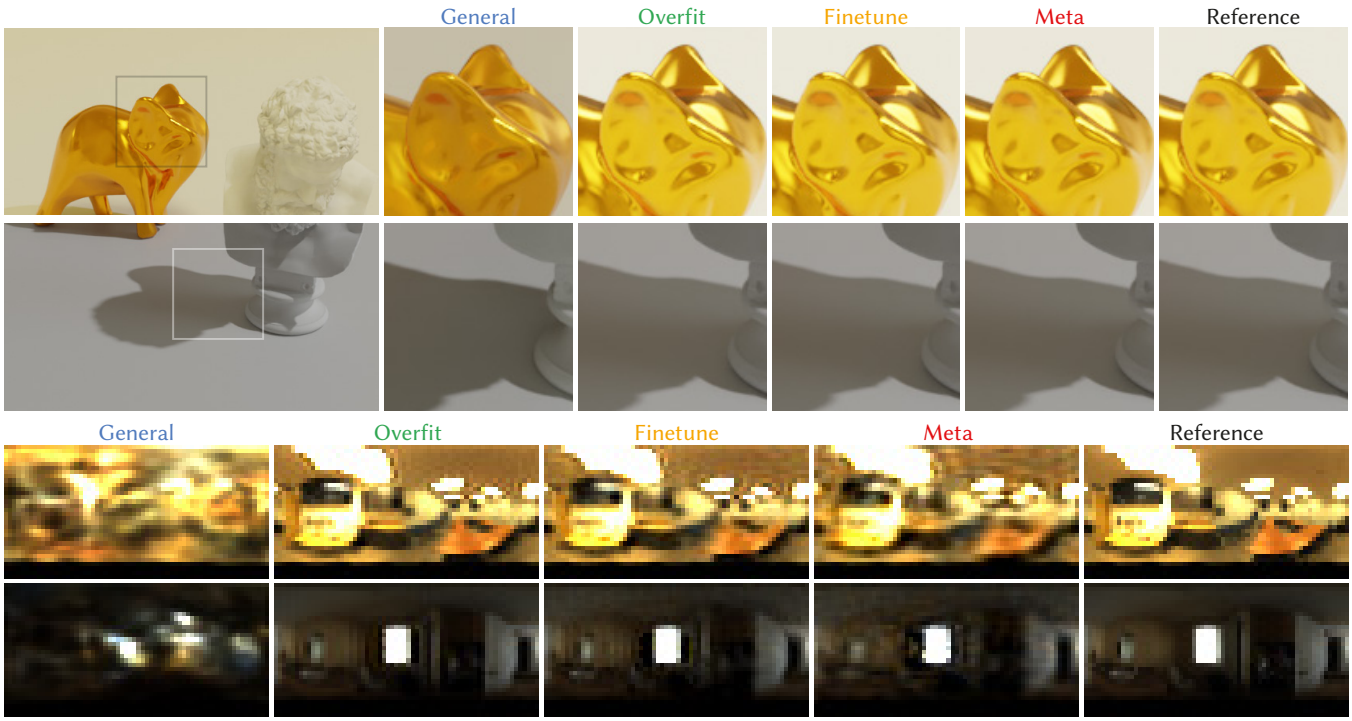

Fig. 7. Relighting results with unseen environment maps from the test-set. The top row shows re-renderings illuminated by the different method's outputs, which we display in the bottom row. Note how all methods achieve crisp, hard shadows, indicating that the high dynamic range of the illumination is matched well, but only optimization-based methods can regress fine nuances, such as the shading gradient in the bottom inset.

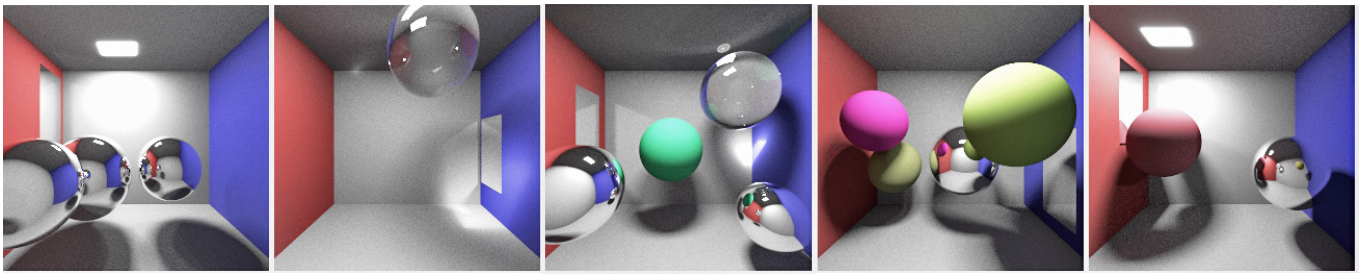

Fig. 8. Random samples from the Cornell box distribution.

Shuran Song and Thomas Funkhouser. 2019. Neural illumination: Lighting prediction for indoor environments. In *CVPR*.

Pratul P Srinivasan, Ben Mildenhall, Matthew Tancik, Jonathan T Barron, Richard Tucker, and Noah Snavely. 2020. Lighthouse: Predicting lighting volumes for spatially-coherent illumination. In *CVPR*.

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

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*.

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

Zian Wang, Jonah Philion, Sanja Fidler, and Jan Kautz. 2021. Learning indoor inverse rendering with 3d spatially-varying lighting. In *CVPR*.

Henrique Weber, Donald Prévost, and Jean-François Lalonde. 2018. Learning to estimate indoor lighting from 3d objects. In *3DV*.

Xin Wei, Guojun Chen, Yue Dong, Stephen Lin, and Xin Tong. 2020. Object-based illumination estimation with rendering-aware neural networks. In *ECCV*.

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