Learning Generative Models for Rendering Specular Microgeometry

ALEXANDR KUZNETSOV, University of California, San Diego
MILOŠ HAŠAN, Adobe Research
ZEXIANG XU, University of California, San Diego
LING-QI YAN, University of California, Santa Barbara
BRUCE WALTER, Cornell University
NIMA KHADEMI KALANTARI, Texas A&M University
STEVE MARSCHNER, Cornell University
RAVI RAMAMOORTHI, University of California, San Diego

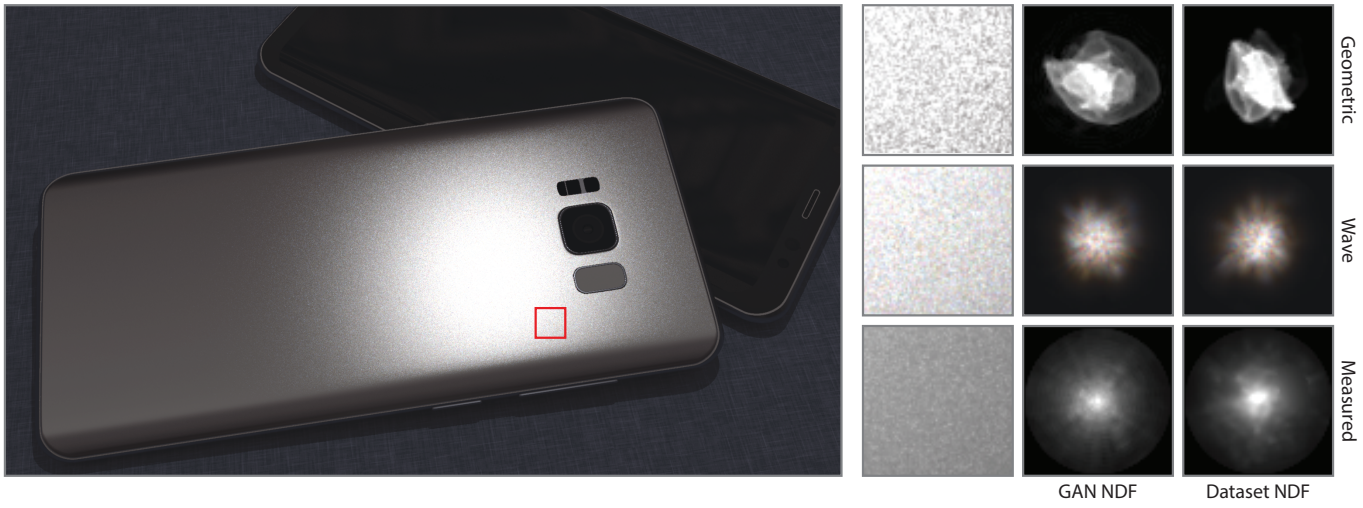


Fig. 1. Our technique renders specular microstructure by *learning* high-frequency angular patterns (which we term generalized NDFs) from synthetic or measured examples. We utilize conditional generative adversarial networks (GANs). Left: the result with a GAN trained on a dataset synthetically generated using a wave optics model. Right: different looks can be achieved by training from different datasets. The top two rows show materials based on synthetic data (using geometric optics and wave optics) and the bottom row is an example trained with measurements of a real surface. The right two columns show example NDF images from the dataset and the generator, to illustrate how the generator mimics the distribution of training data.

Rendering specular material appearance is a core problem of computer graphics. While smooth analytical material models are widely used, the high-frequency structure of real specular highlights requires considering discrete, finite microgeometry. Instead of explicit modeling and simulation of the surface microstructure (which was explored in previous work), we propose a novel direction: learning the high-frequency directional patterns from synthetic or measured examples, by training a generative adversarial network (GAN). A key challenge in applying GAN synthesis to spatially varying BRDFs is evaluating the reflectance for a single location and direction without the cost of evaluating the whole hemisphere. We resolve this using a novel method for partial evaluation of the generator network. We are also able to control large-scale spatial texture using a conditional GAN approach. The benefits of our approach include the ability to synthesize spatially large results without repetition, support for learning from measured data, and evaluation performance independent of the complexity of the dataset synthesis or measurement.

CCS Concepts: • **Computing methodologies → Rendering**.

Additional Key Words and Phrases: specular surface rendering, glints, material appearance, wave optics

ACM Reference Format:
Alexandr Kuznetsov, Miloš Hašan, 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, Article 225 (November 2019), 14 pages. https://doi.org/10.1145/3355089.3356525

Authors' addresses: Alexandr Kuznetsov, University of California, San Diego, a1kuznet@eng.ucsd.edu; Miloš Hašan, Adobe Research, milos.hasan@gmail.com; Zexiang Xu, University of California, San Diego, zexiangxu@cs.ucsd.edu; Ling-Qi Yan, University of California, Santa Barbara, lingqi@cs.ucsb.edu; Bruce Walter, Cornell University, bruce.walter@cornell.edu; Nima Khademi Kalantari, Texas A&M University, nimak@tamu.edu; Steve Marschner, Cornell University, srm@cs.cornell.edu; Ravi Ramamoorthi, University of California, San Diego, ravir@cs.ucsd.edu.

0730-0301/2019/11-ART225 $15.00
https://doi.org/10.1145/3355089.3356525

1 INTRODUCTION

When simulating the appearance of specular highlights, the widely used material models are analytic, based on smooth averages over unresolved statistical microgeometry. However, recent progress has shown that utilizing discrete, finite sub-pixel microgeometry is able to reproduce the high-frequency directional structure of real specular highlights. Previous work explored using various microstructure types and either geometric or wave optics as the underlying physical model. Most of these existing methods are based on explicit modeling and simulation of the microstructure, given either by heightfields or normal maps [Yan et al. 2016, 2018], or by focusing on a specific effect such as reflective flakes or scratches [Jakob et al. 2014; Raymond et al. 2016; Werner et al. 2017]. These methods are either expensive in terms of performance and storage, or limited in their supported set of appearances.

Our idea is to instead *learn* the required high-frequency directional patterns from synthetic or measured examples. Generative learning models, such as GANs [Goodfellow et al. 2014; Radford et al. 2015], are able to synthesize high-resolution, plausible results in various domains, including images, audio, and video. Our paper introduces GANs to appearance modeling: we learn a model from training data that captures the complex directional distributions induced by specular microstructure (Section 5). The functions we learn resemble smooth microfacet NDFs but are not necessarily proper distributions, can be functions of vectors other than the half vector, and can even contain RGB colors; we term these learned directional distributions *generalized normal distribution functions (GNDFs)*. At runtime, we evaluate the trained model, rather than computing the result using explicit physical simulation like in previous work. This means we can use much smaller textures compared to full microgeometry, or even eliminate textures entirely.

Larger-scale structures that induce correlations between pixels (e.g. scratches on metals or weave patterns on fabrics) pose a challenge for basic GANs, which simply generate samples resembling the training set without any addditional control. We solve this by defining a low-dimensional feature vector that has the right correlation, making it an additional input to a conditional GAN (cGAN) [Mirza and Osindero 2014]. This feature vector can itself be learned using an autoencoder network (Section 5.4).

One challenge is that evaluating the full network for synthesizing the entire GNDF at a given pixel is wasteful. Instead, at each surface location, we are only interested in evaluating a single GNDF value, to obtain the reflectance for a specific light direction. We address this efficiency issue by introducing partial evaluation of convolutional neural networks. Given a small range of interest in the generator output (say, a single element or a small block), we find corresponding ranges in the internal layers of the network that influence the values of interest, and compute only these values (Section 6 and Figure 8).

An advantage of generative models is that they can be trained on any data, real or synthetic, geometric or wave optics. We demonstrate results trained on synthetic BRDF data computed using both geometric and wave optics, and on data captured using a spherical gantry. The key novel contributions of our approach are thus:

- The first method to use GANs and cGANs (or deep learning in general) to render complex sub-pixel specular microstructure.
- A novel partial evaluation algorithm, allowing for point or range queries of generator network output.

We make several additional technical contributions, including the reformulation of wave-optics glints [Yan et al. 2018] in a GNDF framework and computing the wave GNDFs efficiently using FFTs, mapping fiber-based cloth models into the GNDF framework, and hole filling in measured data during training. Our solution has the following advantages over previous work:

- Evaluation performance and rendering algorithm complexity are independent of the training dataset. This allows for expensive synthesis or measurement methods without slowing down the rendering.
- Ability to synthesize results without obvious spatial repetition, which is difficult to achieve for previous methods working from explicit microstructures. In our cGAN results, the feature texture can have much lower spatial resolution than the microstructure itself; therefore, it can cover a much larger spatial area with the same storage. Note that our non-conditional results are naturally infinite and non-repeating.
- Ability to learn the model from a measured dataset. While measuring appearance data is not the primary goal of our paper, we include one example dataset to illustrate this possibility. Using measured data for rendering directly is theoretically possible, but impractical due to massive storage and artifacts due to tiling and holes; our solution is much more convenient.
- The storage requirement of the trained generator network is just over 1 megabyte (~300,000 weights using 32-bit floats), regardless of the size of the microstructure description or the training dataset, which can be arbitrarily large. The network thus becomes a convenient material representation and exchange format; the rendering system using it does not need to know about the advanced optics models, measurement or training techniques that went into producing it.

Figure 1 shows a result generated with our method; please refer to the supplementary video for animated results. We demonstrate the generality of our approach by showing a diverse set of results including several isotropic glinty metallic materials (synthesized using geometric and wave optics, and measured), an anisotropic brushed metal, as well as scratched ceramic and fabric examples (the latter two using a cGAN controlled with feature textures).

We believe our work is a step towards appearance models that achieve or surpass the quality of explicit microstructure rendering, without the run-time algorithmic complexity and storage cost.

2 RELATED WORK

In this section, we review previous work on rendering detailed specular microgeometry, followed by related work on material capture, generative models in machine learning, and more.

Explicit heightfields. Several previous methods target spatially-varying fine-scale details and "glints." Yan et al. [2014; 2016] presented *glint integrators* for rendering surfaces defined by explicit high-resolution heightfields (or normal maps), under geometric

optics. These approaches are successful at simulating very high-resolution, spatially varying glinty behavior. The key idea is to extend the NDF from standard microfacet models [Cook and Torrance 1982; Walter et al. 2007] to a patch-based $\mathcal{P}$-NDF, essentially replacing the large-area average for the whole surface by a unique BRDF per given small patch of the surface. Lately, Yan et al. proposed an approach that addresses the same problem using wave optics models instead [2018]. Our method succeeds in replicating the results of these methods with much lower storage requirements and (in most cases) faster performance, but also allows a broader range of input data (measured GNDFs, fabric microgeometry).

Scratches and flakes. Raymond et al. [2016] represents the surface as a collection of one-dimensional scratches over a smooth BRDF. Werner et al. [2017] take a similar approach based on wave optics, rendering surfaces with collections of randomly oriented scratches using a Harvey-Shack-based model. In each of these approaches, the solution for a single scratch can be found analytically or precomputed. Jakob et al. [2014] also simulated glinty surfaces but used a statistical distribution of tiny mirror-like flakes rather than an explicit surface. The locations of the flakes are defined implicitly through a procedural hierarchy in position-normal space. However, this algorithm only generates a specific kind of microstructure similar to metallic paint flakes.

Material capture. A broad area of computer graphics focuses on acquisition of materials from physical measurements; here we only mention recent work that uses deep learning. Aittala et al. [2016] captured stationary spatially-varying SVBRDFs from a single flash-lit photograph using a neural texture descriptor. Further work [Deschaintre et al. 2018; Li et al. 2017, 2018] has been able to capture some non-stationary SVBRDFs with an end-to-end deep convolutional architecture. However, only parameters of smooth BRDF models are captured with a single estimate per pixel at relatively low resolution, unlike our focus on complex specular microstructure *within a pixel*.

Microgeometry capture. Dong et al. [2015] acquired the surface microgeometry of real metallic surfaces using interferometry, and applied wave optics theory to successfully predict their smooth, large-area average BRDFs. We considered using their captured microgeometry data, but found we would need larger spatial area to provide sufficiently varied training data for our method. Note that Dong et al. do not render directly from the measured microgeometry. Instead, they generate textures from their sparse data by relying on two simplifying assumptions: the NDFs fit a simple analytic model (ellipsoid NDF, essentially a generalization of GGX) and the spatial patterns are 1D separable products. However, these assumptions do not hold for our datasets and our method does not use them.

Another line of previous research focuses on capturing microgeometry explicitly using visible light [Graham et al. 2013; Nagano et al. 2015; Nam et al. 2016]. Unlike this prior work, we do not try to explicitly reconstruct the microgeometry of the surface, but instead measure a slice of the local BRDF which is then used to infer the GNDF. As such we need more light positions but less spatial resolution. However, microgeometry scanned using these methods could be used as input to our GNDF synthesis.

Generative models. Generative adversarial networks (GANs) [Goodfellow et al. 2014] have become widely used to synthesize plausible results in various domains, including image [Radford et al. 2015], video [Tulyakov et al. 2018], and audio [Donahue et al. 2018]. A GAN typically consists of two competing networks; a generator and a discriminator. The generator is trained to produce results that are indistinguishable from the real data, while the discriminator learns to identify them.

Extensive research has been conducted to improve the quality of the synthesized results through better network architectures (e.g., DCGAN [Radford et al. 2015]), more powerful loss functions [Arjovsky et al. 2017], and training strategies [Karras et al. 2018]. Because of these developments, GANs have been used to handle a variety of applications such as image-to-image translation [Wang et al. 2018], image inpainting [Iizuka et al. 2017], and super-resolution [Ledig et al. 2017]. In this paper, we utilize GANs for appearance modeling, a rather different application. We use a network architecture similar to DCGAN [Radford et al. 2015] and introduce point-wise and range evaluations to improve the efficiency. To our knowledge, none of the previous work on GANs or related generative models considered the problem of point-wise and/or range evaluation of the models, but this operation is critical in our application.

Other related work. Recent work on neural BTF compression [Rainer et al. 2019] uses an autoencoder framework to compress BTFs, which is a different but related appearance representation to our GNDFs. However, they do not focus on detailed glinty microgeometry rendering. We also use autoencoders to control the feature vectors in our cGAN; we find the decoder produces smooth blurry results compared to GANs, so it is likely that a GAN framework like ours is required to generate high-frequency results in the directional domain. Galerne et al. [2012] proposed a system to synthesize Gabor noise from examples; this is in a sense a generative model as well, and could be considered as an alternative to GANs in our framework.

3 OVERVIEW

Several existing methods for rendering specular microstrucure can be seen as using a microfacet BRDF with modified (generalized) normal distribution functions (GNDFs). The high-level idea of our approach is to generate examples of these GNDFs, learn a GAN (or cGAN) model to synthesize GNDFs indistinguishable from the ones in the dataset, and design an efficient way to use the generator at rendering time to produce final pixel values.

3.1 Our material model

Recall the standard microfacet BRDF [Cook and Torrance 1982; Walter et al. 2007]:

$$f_r(\boldsymbol{i}, \boldsymbol{o}) = \frac{F(\boldsymbol{i}\cdot\boldsymbol{h})\, G(\boldsymbol{i}, \boldsymbol{o})\, D(\tilde{\boldsymbol{h}})}{4\,(\boldsymbol{i}\cdot\mathbf{n})\,(\boldsymbol{o}\cdot\mathbf{n})} \tag{1}$$

Definitions of all symbols can be found in Table 1. We use $\boldsymbol{i}$, $\boldsymbol{o}$, $\boldsymbol{h}$, and $\mathbf{n}$ to refer to input, output, half vector, and macrosurface normal, respectively. We use $\bar{\boldsymbol{h}}$ to denote the projected half vector (dropping the z-coordinate of the unit vector $\boldsymbol{h}$). The key term that determines

Table 1. Notation used in the paper.

symbol	domain	definition
$\mathcal{H}$		hemisphere
$\mathcal{D}$		unit disk (projected hemisphere)
$\mathbf{n}$	$\mathcal{H}$	macrosurface normal
$\boldsymbol{i}$	$\mathcal{H}$	incoming direction
$\boldsymbol{o}$	$\mathcal{H}$	outgoing direction
$\boldsymbol{\psi}$	$\mathbb{R}^3$	sum vector, $\boldsymbol{\psi} = \boldsymbol{i} + \boldsymbol{o}$
$\boldsymbol{h}$	$\mathcal{H}$	half vector, $\boldsymbol{h} = \boldsymbol{\psi}/\|\boldsymbol{\psi}\|$
$\bar{\boldsymbol{h}}$	$\mathcal{D}$	projected half vector (drop 3rd coord.)
$\bar{\boldsymbol{\psi}}$	$\mathbb{R}^2$	projected sum vector (drop 3rd coord.)
$\mathbf{u}$	$\mathbb{R}^2$	surface position
$\boldsymbol{d}$	$\mathcal{D}$ or $\mathbb{R}^2$	projected direction (either $\bar{\boldsymbol{h}}$ or $\bar{\boldsymbol{\psi}}$)
z	$\mathbb{R}^{100}$	latent random vector (generator input)
$n(\mathbf{u})$	$\mathbb{R}^2 \to \mathcal{D}$	microsurface normal function
$G_p(\mathbf{u})$	$\mathbb{R}^2 \to \mathbb{R}$	pixel footprint Gaussian
$G_c(\boldsymbol{s})$	$\mathbb{R}^2 \to \mathbb{R}$	coherence kernel Gaussian
$D(\bar{\boldsymbol{h}})$	$\mathcal{D} \to \mathbb{R}$	normal distribution function (NDF)
$D^*(\mathbf{u}, \boldsymbol{d})$	$\mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$	generalized NDF
$F(\boldsymbol{i} \cdot \boldsymbol{h})$	$\mathbb{R} \to \mathbb{R}$	Fresnel term (accurate or Schlick)
$G(\boldsymbol{i}, \boldsymbol{o})$	$\mathcal{H} \times \mathcal{H} \to \mathbb{R}$	shadowing/masking term
$G^*(\boldsymbol{i}, \boldsymbol{o})$	$\mathcal{H} \times \mathcal{H} \to \mathbb{R}$	extra term ($\frac{1}{4}(\boldsymbol{\psi} \cdot \mathbf{n})^2$ for wave optics, $G(\boldsymbol{i}, \boldsymbol{o})$ for geometric optics)

the shape of the specular highlight is the normal distribution function (NDF) $D(\bar{\boldsymbol{h}})$, giving the probability distribution of microfacet normals describing the surface. The NDF is indeed a pdf on the projected hemisphere $\mathcal{D}$ (it integrates to 1 on this domain, but would need an additional cosine term on $\mathcal{H}$).

Most previous work on specular glints can be thought of as replacing the smooth, global NDF $D(\bar{\boldsymbol{h}})$ by a different function $D^*(\mathbf{u}, \bar{\boldsymbol{h}})$ or $D^*(\mathbf{u}, \bar{\boldsymbol{\psi}})$, where $\boldsymbol{\psi} = \boldsymbol{i} + \boldsymbol{o}$ and $\bar{\boldsymbol{\psi}}$ is the projected $\boldsymbol{\psi}$; $\mathbf{u}$ is a 2D spatial location. In this paper we will therefore consider spatially varying BRDFs in the following microfacet-like form:

$$f_r(\mathbf{u}, \boldsymbol{i}, \boldsymbol{o}) = \frac{F(\boldsymbol{i} \cdot \boldsymbol{h})\, G^*(\boldsymbol{i}, \boldsymbol{o})\, D^*(\mathbf{u}, \boldsymbol{d})}{4\, (\boldsymbol{i} \cdot \mathbf{n})\, (\boldsymbol{o} \cdot \mathbf{n})} \tag{2}$$

The main difference from equation (1) is in the NDF term, which we replaced by a generalized NDF $D^*(\mathbf{u}, \boldsymbol{d})$. For brevity we use a $\boldsymbol{d}$ parameter in the definition, which stands for any projected direction, $\bar{\boldsymbol{h}}$ or $\bar{\boldsymbol{\psi}}$. We also introduce an extra term $G^*(\boldsymbol{i}, \boldsymbol{o})$, which is defined differently for geometric optics and wave optics models (see Table 1 and next section). Therefore, this single definition encapsulates both the geometric optics and wave optics models used in this paper.

The function $D^*(\mathbf{u}, \boldsymbol{d})$ depends on the surface location $\mathbf{u}$ and typically contains high directional frequencies. In a slight abuse of terminology, we still call these functions generalized NDFs, even though they are not necessarily distributions of normals anymore, and may even contain color variation. In other words, we use the term GNDFs in a broad sense, to mean directional functions that resemble (or fit into) the NDF component of the microfacet model. (In previous work, the term $\mathcal{P}$-NDFs is sometimes used.)

3.2 Learning generalized NDFs

While several previous methods compute values of the generalized NDF precisely from an explicit surface heightfield or normal map, we take a different approach: we learn a GAN that generates these complex GNDF distributions, by providing a training set. We define a *GNDF image* to be the discretized directional variation of $D^*(\mathbf{u}, \boldsymbol{d})$, for a fixed $\mathbf{u}$, discretizing the projected directions $\boldsymbol{d}$ in the plane. The training set is simply a collection of such GNDF images for different surface locations $\mathbf{u}$.

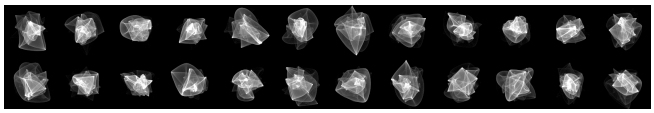


Fig. 2. Examples of geometric optics GNDF images, computed by binning.

The surface microstrucure types we are considering in this paper have very limited spatial coherence (i.e., small patches that are tens of microns apart exhibit largely unrelated microstrucure). This allows us to design an approach consisting of the following steps:

(1) Generate a training dataset of example GNDF images, either synthetically (by covering a given heightfield texture or fiber-lever fabric model with footprints and computing the corresponding GNDFs) or by measurement on a spherical gantry. The number of training images in our experiments is between 65 and 75 thousand. For isotropic microstructures (where the microstructure is rotationally invariant), the dataset can be augmented by random rotation.
(2) Learn a GAN that produces GNDF images indistinguishable from the ones in the dataset, parameterized by a latent random vector $\boldsymbol{z}$. We use a network structure inspired by DCGAN, with transposed convolution operations used in the generator.
(3) For examples with large-scale spatial structure (scratched ceramic and fabric), we use an additional feature texture defining $\boldsymbol{f}$ at each spatial location, which is produced by autoencoder learning and texture synthesis.
(4) To be able to evaluate the BRDF from equation (2), we need to define the corresponding $D^*(\mathbf{u}, \boldsymbol{d})$. This is done by either assigning a different latent vector $\boldsymbol{z}$ to each pixel, or by defining a grid of latent vectors in texture space, interpolating latent vectors within the grid cells (the first method is faster while the second is more general). For a given $\mathbf{u}$, we thus find the appropriate $\boldsymbol{z}$ (and optionally $\boldsymbol{f}$) and run the generator network to produce the corresponding GNDF image, which gives the directional variation as a function of $\boldsymbol{d}$ (i.e., $\bar{\boldsymbol{\psi}}$ or $\bar{\boldsymbol{h}}$). This GNDF is then plugged into the full BRDF equation (2).

3.3 Partial evaluation of generator networks

As described, the above procedure has one highly inefficient step: for a single query, it runs the entire generator network to compute the full GNDF image, only to query a single location $\boldsymbol{d}$ in it and throw away the rest. A key contribution of our work is to enable partial evaluation of generator networks, computing only the portion of the network that affects the desired point or range query.

The next three sections will cover the above steps in detail: data generation/measurement, GAN training, and final model evaluation.

4 DATA GENERATION AND MEASUREMENT

The input to our GAN training procedure is a dataset of example GNDF images. We produce this dataset either synthetically from a given heightfield (bump map) texture, fiber-lever fabric model, or by measurement using a spherical gantry. Figure 3 illustrates the shapes of some of our dataset GNDFs, as well as the corresponding generated GNDFs (discussed in Section 5).

4.1 Synthetic GNDF generation

We start with a high-resolution input heightfield texture. The heightfield is then covered by a uniform grid of footprints. To compute the GNDFs corresponding to these footprints, we explore two different models: using geometric optics [Yan et al. 2014] and using wave optics [Yan et al. 2018].

4.1.1 Geometric optics GNDFs. Yan et al. [2014] consider an entire patch (footprint) of a surface heightfield at once, and consider the actual distribution of surface normals across this patch. More precisely, let $\mathcal{D}$ be the unit disk (projected hemisphere). A normal map is defined as a function $n : \mathbb{R}^2 \to \mathcal{D}$ from points $\mathbf{u}$ in texture space to normals $n(\mathbf{u})$ on the unit disk. This normal map can be easily computed from the gradient of the heightfield, using bicubic interpolation. Define a Gaussian footprint $G_p(\mathbf{u})$ with standard deviation σ_p. Let X be a random variable distributed according to this Gaussian. The GNDF for this footprint can then be defined as the probability distribution on $\mathcal{D}$ of the random variable $n(X)$.

We compute the geometric optics GNDFs by a binning (histogram) approach: we generate samples of the random variable X, and bin the resulting values of $n(X)$. The noise in this approach is significantly reduced by stratifying the sampling of X. We find that one million samples per GNDF are sufficient for our purposes. When binning, we further jitter the value $n(X)$ by a Gaussian whose standard deviation matches half a pixel size in the GNDF image, to alleviate box filtering artifacts from the bins; this is equivalent to the intrinsic roughness defined by Yan et al. [2014]). Examples of GNDF images computed in this way are shown in Figure 2.

4.1.2 Wave optics GNDFs. More recently, a wave optics method for rendering glints was introduced [Yan et al. 2018]. This paper proposes multiple forms of the spatially varying wave-optics BRDF. Here we will use the "reciprocal original Harvey-Shack" (R-OHS) model, which can be written as follows:

$$f_r(\mathbf{u}, \boldsymbol{i}, \boldsymbol{o}) = \frac{(\boldsymbol{\psi} \cdot \mathbf{n})^2 F(\boldsymbol{i} \cdot \boldsymbol{h})}{4 A_c \lambda^2 \, (\boldsymbol{i} \cdot \mathbf{n}) \, (\boldsymbol{o} \cdot \mathbf{n})} \left| \int_{\mathbb{R}^2} R_{\mathbf{u}}(\boldsymbol{s}) \, e^{-i \frac{2\pi}{\lambda} (\bar{\boldsymbol{\psi}} \cdot \boldsymbol{s})} d\boldsymbol{s} \right|^2, \quad (3)$$

where λ is the wavelength, and

$$R_{\mathbf{u}}(\boldsymbol{s}) = G_c(\boldsymbol{s}) \, e^{-i \frac{4\pi}{\lambda} h(\boldsymbol{s})} \quad (4)$$

is the reflection function weighted by the coherence kernel $G_c(\boldsymbol{s})$. This coherence kernel is centered at $\mathbf{u}$ and has a standard deviation σ_c (typically 5-10 microns). $A_c = \int_{\mathbb{R}^2} G_c(\boldsymbol{s})^2 \, d\boldsymbol{s}$ is a normalizing constant. Please refer to Yan et al. [2018] for a more detailed derivation and explanation of the BRDF model and the coherence kernel concept.

While this formulation does not explicitly use the concept of normals and their distributions, we find that it is useful to rewrite it in the form (2) matching a microfacet BRDF with a modified "wave optics GNDF". This can be done as follows:

$$f_r(\mathbf{u}, \boldsymbol{i}, \boldsymbol{o}) = \frac{F(\boldsymbol{i} \cdot \boldsymbol{h}) \, G^*(\boldsymbol{i}, \boldsymbol{o}) \, D^*(\mathbf{u}, \bar{\boldsymbol{\psi}})}{4 \, (\boldsymbol{i} \cdot \mathbf{n}) \, (\boldsymbol{o} \cdot \mathbf{n})}, \quad (5)$$

where

$$D^*(\mathbf{u}, \bar{\boldsymbol{\psi}}) = \frac{4}{A_c \lambda^2} \left| \int_{\mathbb{R}^2} R_{\mathbf{u}}(\boldsymbol{s}) \, e^{-i \frac{2\pi}{\lambda} (\bar{\boldsymbol{\psi}} \cdot \boldsymbol{s})} d\boldsymbol{s} \right|^2. \quad (6)$$

and $G^*(\boldsymbol{i}, \boldsymbol{o}) = \frac{1}{4} (\boldsymbol{\psi} \cdot \mathbf{n})^2$. Defining the normalization this way makes the extra term $G^*(\boldsymbol{i}, \boldsymbol{o})$ bounded by 1, and the wave GNDF has a range of values comparable to the geometric GNDF of the same microstructure. A similar rewrite was proposed by Dong et al. [2015] in the context of the Kirchhoff wave optics model.

Looking at the definition of this GNDF, we find it is (ignoring the multiplicative terms in the front) essentially the squared absolute value of the Fourier transform of the weighted reflection function $R_{\mathbf{u}}(\boldsymbol{s})$, queried at point $\bar{\boldsymbol{\psi}}/\lambda$. This suggests we can use the fast Fourier transform to compute the GNDF image efficiently for each coherence footprint. This computation is deterministic; it does not use random sampling, unlike our geometric optics binning method.

The above GNDF definition depends on the wavelength λ. A spectral version can be constructed simply by evaluating several wavelengths (each using a separate FFT) and converting into RGB space. We find that 8 spectral samples are sufficient for this purpose.

Furthermore, the pixel footprint is typically larger than the coherence kernel. We resolve this by averaging several GNDF images computed as above, to obtain GNDF images matching the target pixel size. Note, this is not equivalent to simply enlarging the coherence kernel, as the coherence integral is in the complex domain (within the squared absolute value operator), while the pixel averaging happens in the real domain. Figure 4 shows examples of wave GNDFs; the top of the figure shows the coherence kernel GNDFs, while the bottom shows the averaged pixel footprint GNDFs.

4.2 Fabric GNDFs

Modeling cloth at the fiber level is a growing area of research with remarkable recent progress [Leaf et al. 2018; Zhao et al. 2016]; the explicit fiber representation can be used for high-fidelity rendering, but the cost is prohibitive in most applications. The situation is different in our case, where the cost is incurred only in dataset creation, not in training or rendering. We extend our approach to render fabrics, by synthesizing and learning GNDFs from a basket weave cloth pattern simulated at the fiber level. We shade the fibers using the hair BSDF model of Chiang et al. [2016]. The GNDF data synthesis is accomplished by covering the cloth area by 256^2 footprints, and computing the corresponding GNDFs by sampling scattering paths with random incoming direction through the fabric. We bin the resulting path throughput at the half vector given by the path incoming and outgoing direction.

The true BRDF of this material is not in the form required by our equation 2, so this approach is an approximation to the true BRDF. In terms of the half-vector and difference vector parameterization of BRDFs [Rusinkiewicz 1998], we are ignoring the difference vector dimension (i.e. averaging over it), keeping only the half vector dimension. However, this approximation is quite effective for single fiber reflection, and still produces a reasonable appearance for

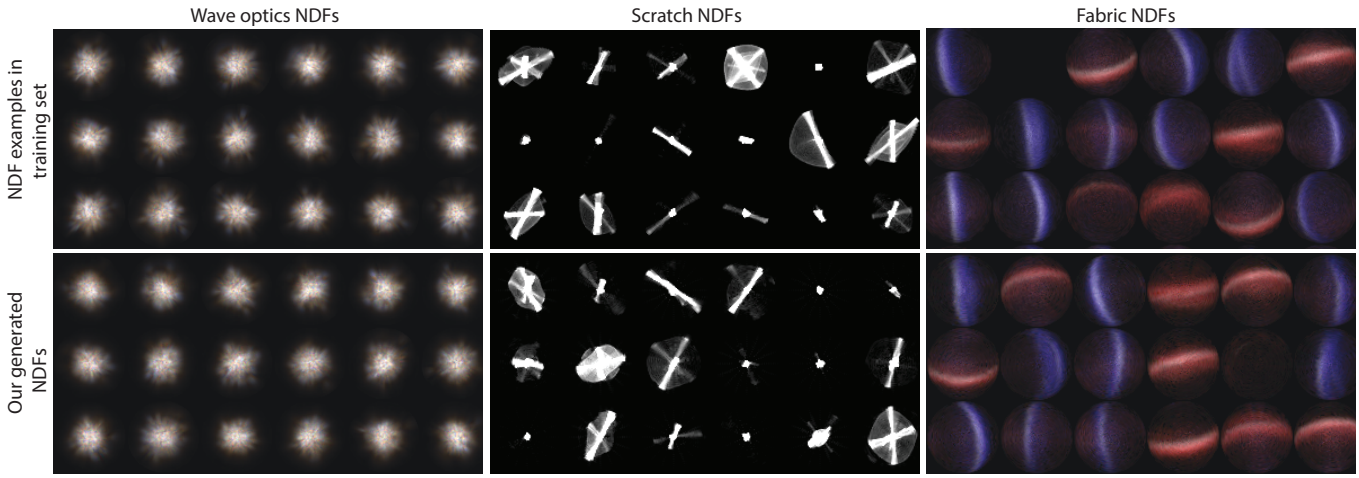


Fig. 3. Comparison of ground truth (dataset) GNDFs and GNDFs generated by our method. Our method not only accurately captures the general distributions of GNDFs for each different material, but also generates each individual GNDF with high quality and detail.

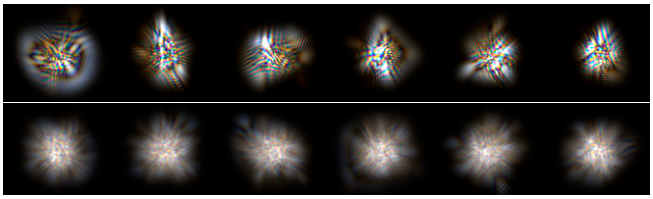


Fig. 4. Examples of wave optics GNDFs. Top: GNDF images corresponding to coherence kernels. Bottom: GNDFs corresponding to pixel footprints; these are essentially weighted averages of the images above.

low-order scattering (we are limiting the maximum depth to 5 fiber interactions).

4.3 Measurement of GNDFs

For GNDF measurement, we used a spherical gantry consisting of a camera and a light source mounted on robotically controlled arms, automatically calibrated before each run using computer vision techniques. The camera and sample were kept fixed; we only moved the light to measure a variety of incoming directions.

The sample used is a steel Q-Panel with matte finish, manufactured by Q-Lab Corporation. The camera is a Canon 70D with a 100 mm macro lens configured to keep a fixed focus and aperture. It is 1 meter away from the sample and rotated 15 degrees from vertical, so we can observe GNDF peaks. The setup is shown in Figure 5 (top). For each measurement, we combine multiple exposures to create an HDR image. The images were downsized by a factor of 2 to reduce noise and converted to grayscale (as there was little color variation for this particular sample). The resulting pixel spacing corresponds to roughly 70 microns on the sample surface.

The light source is a 19 mm diameter white LED at a distance of 60cm from the sample. We took measurements at 4681 light positions covering the hemisphere above the sample except for grazing angles and a small retro-reflection region where the light source would obstruct the camera's view. The light source intensity is calibrated by observing a reference material within the field of view.

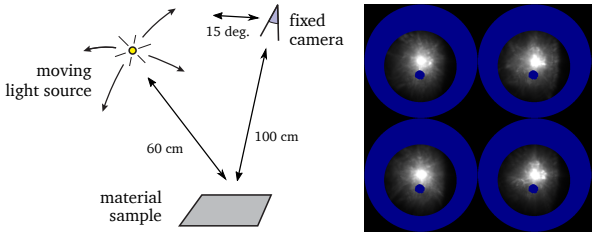


Fig. 5. Left: Our measurement setup, with a camera 15 degrees off vertical direction, and a moving light. Right: Examples of measured GNDFs, with blue color indicating missing data (grazing angles or occlusion).

For each pixel and light position, the measured values are converted to BRDF values. Assuming a microfacet model, we further convert these into GNDF estimates, approximating the shadowing/masking term as 1 and the Fresnel term as constant over the region of interest (as grazing angles are excluded). For each pixel, the measured values are interpolated over the hemisphere of directions using a kernel with a 3 degree radius. Thus, each visible pixel of the sample gives rise to a single measured GNDF image. Several examples are shown in Figure 5 (bottom).

5 MODEL AND TRAINING

Here we describe the network structure of our GAN, and the training procedure. Our implementation uses the PyTorch framework.

5.1 Data transformations

We apply a log transformation $\log(1 + x)$ to the data, as the original GNDF values have fairly high dynamic range. This is commonly used by recent techniques that deal with HDR data [Eilertsen et al. 2017; Zhang and Lalonde 2017] as it makes the training process better behaved. We keep all training data positive; we currently do not center the values at zero.

We further apply a polar transformation to the data. We find that the training performance is significantly improved, because GNDF images have a broadly circular structure and the convolution can reuse the features. For GNDFs with rotationally symmetric statistics,

a rotation becomes horizontal translation: a natural setup for a convolutional network. This ensures that our learned convolutional kernels treat the GNDF values at the same distance from the center equally.

5.2 Network structure

Our generator and discriminator networks broadly follow DCGAN [Radford et al. 2015], and are visualized in Figure 6. The generator starts from a latent vector of length 100, followed by five transposed convolution layers. Increasing the length of the latent vector is possible, but has little impact on quality or performance. The transposed convolutions have a kernel size of 4×4 and a stride of 2. Unlike DCGAN, which uses between 64 and 1024 internal channels, the number of channels of our internal layers is always 64. The output layer has 1 or 3 channels. The discriminator is symmetric to the generator, using standard convolutions instead of transposed ones.

The generator output (in the polar domain) has a 64×64 resolution; this is sufficent to achieve high directional frequencies (and resulting high temporal frequencies with slow light movement) in our results. Larger resolutions should be possible, if very high directional frequencies are desired.

Due to our use of the polar domain, we introduce *horizontally wrapped* convolution operators (both in the generator and discriminator). In the polar domain, the columns of our images correspond to angles and the rows correspond to radii. As we would like our result to be continuous in angles, we need the convolution and transposed convolution operators to wrap around horizontally across the image edges; this applies to internal as well as output layers. For this purpose, we introduce two custom neural network modules, `Conv2dHWrap` and `ConvTranspose2dHWrap`; their implementation is in PyTorch and does not require new C++ primitives.

Some further differences from DCGAN include omitting the batch normalization operations in the generator and the tanh function at the end of the generator. We use leaky ReLU in both generator and discriminator.

5.3 Training process

We use noisy labels similarly to what is proposed by Salimans et al. [2016]; this appears to improve stability in the beginning of the training process.

We utilize random rotation to augment our training data, for the materials that are isotropic; their statistics are rotationally invariant, so the distribution of rotated GNDF images looks the same as the original distribution. In polar coordinates, this random rotation becomes particularly easy, simply requiring a horizontal pixel shift with wrap-around. (This does not apply to brushed, scratched and fabric examples, as they are not rotationally invariant.)

For training with RGB examples (as needed for the spectral wave optics and fabric models), we find that while it is possible to train the networks with RGB examples directly, the convergence is notably worse than with grayscale GNDF images. We found that we can achieve an improved convergence for RGB datasets by starting the training process with grayscale versions of the images (say over the first 10,000 batches), and gradually blending in the color (say over the next 10,000 batches or so).

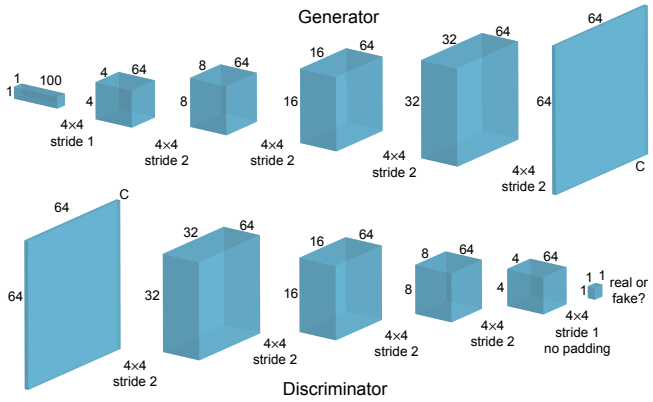


Fig. 6. The network structure of our GAN.

We further compute the mean of the dataset GNDF images, as well as the generated GNDF mean. We find that these match fairly closely in their overall scaling, so the GAN succeeds in generating GNDF values with the correct integrals. However, we find that there is a slight directional pattern in the generated mean. Therefore, during evaluation, we correct the remaining generator bias by multiplying the output by the ratio of dataset and generated means. Note, this slight mismatch between dataset and generated mean occurs for traditional GAN applications as well, but is not a problem for them, because generated images are used individually and not as a big collection at once. Once the training is done, the discriminator is discarded and we generate the results using only the generator.

5.4 cGAN and feature vectors

For microstructures with large-scale texture variation (scratches or fabric yarns), the spatial correlations between pixels are important to the appearance. We extend our model to account for this correlation by using a cGAN to make the generated GNDF distribution depend on a low-dimensional feature vector $\boldsymbol{f}$. The feature vector varies according to a texture to efficiently model the visually important spatial structure of the material. One could, in theory, design such feature vectors by hand (for example, for the scratched ceramic example it could be a dominant scratch direction and footprint coverage). We however find it more powerful and general to learn the feature space itself, through an autoencoder architecture.

Our encoder and decoder shapes are virtually identical to the discriminator and generator, respectively. The difference is that the encoder outputs the feature vector $\boldsymbol{f}$ (using 9 dimensions in our results) and the decoder takes it as input, attempting to reconstruct the encoder input. We train the autoencoder with the same GNDF training set, using an L1 loss. The decoder thus has a similar role as the generator; however its outputs tend to be blurry and lack detail, which the adversarially-trained generator produces much more effectively.

This allows us to compute a feature vector for each GNDF in the training set, giving rise to a 256^2 feature texture. In the case of scratched ceramic and fabric, this texture is too small to produce non-repeating results, and needs to be extended; we currently achieve that by a patch-based texture synthesis [Efros and Freeman 2001]. Figure 7 demonstrates the idea: an initial scratched heightfield is

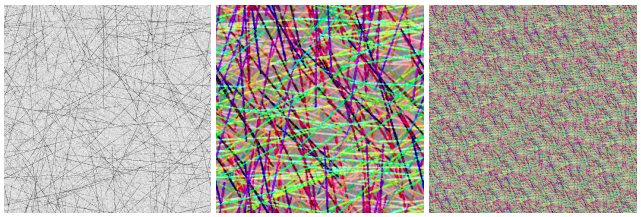


Fig. 7. Left: initial scratched heightfield. Middle: feature vector texture computed using our autoencoder from the GNDFs (showing 3 out of 9 channels as RGB colors). Right: the feature texture extended by texture synthesis.

used to create a GNDF training set, resulting in a feature vector texture, which is finally extended by texture synthesis.

Using the feature $\boldsymbol{f}$ in the generator is straightforward; we simply append it to the latent random vector $\boldsymbol{z}$. For the discriminator, we pass $\boldsymbol{f}$ through the decoder first and use the resulting image as an additional discriminator input.

In summary, we are using a two-pass learning system. First, we train an autoencoder and fix its weights. Then, we use the resulting fixed feature vectors (textures) to train a cGAN. After training the autoencoder for a particular material, we evaluate the encoder for each of the 256x256 NDF images in the input data, producing a 9-dimensional feature vector for each NDF. The resulting 9-dimensional feature vectors form a 256x256x9 texture. The feature textures can furthermore be extended through synthesis, to cover much larger areas without repetition.

We opted for training separate models for each material because we want to keep the networks as small as possible. A more general network would require more weights to represent all required variations. That said, in the future it may be possible to train a single general (larger) cGAN for multiple materials, conditioned on the material type.

5.5 Hole filling in measured data

In measured GNDFs, some positions could not be captured due to obstruction of the camera by the light, resulting in holes. As a first step, we mask the holes in captured GNDF images by using the average value at that direction. Moreover, we randomly rotate captured GNDF images, so the hole is not in the same spot. For our generator, we apply the same mask to the output, but randomly rotate it. That way, the generator does not need to generate a hole for the discriminator. The discriminator therefore sees the hole in different positions for both measured and generated data, and the generator learns to generate complete GNDF images without holes.

6 MODEL EVALUATION

In this section, we describe the evaluation of the full BRDF model (2), using the trained generator networks. A key component of this section is our partial network evaluation; this is critical for efficency of the whole solution. To evaluate the BRDF, the only non-trivial component is the evaluation of the GNDF; all other terms (Fresnel, normalization, etc.) can be incorporated easily.

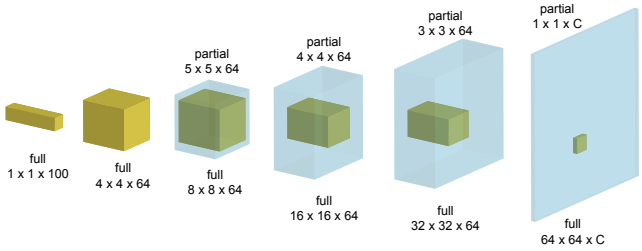


Fig. 8. Partial evaluation of the generator network. The full layers are shown in light blue, while the portions required by partial evaluation are shown in yellow. In this example, we are only interested in evaluating index (34, 42) in the output layer. Using the range bounding approach, we find that within the third, fourth and fifth layer, we need to evaluate ranges (blocks) $[2, 6]\times[3, 7]$, $[7, 10] \times [9, 12]$, and $[16, 18] \times [20, 22]$, respectively.

6.1 Pixel sampling for GNDF evaluation

There are two possible ways to render the pixels of the image. The first option is to assign one latent vector z per pixel of the appropriate material. This approach is convenient due to its simplicity and performance, and has the additional advantage of supporting random rotation of isotropic GNDFs. (Note: this result rotation at rendering time is distinct from the rotation of the input data during training, which is always valid for isotropic microstructures.) Random rotation gives a slightly improved result, further equalizing the angular statistics. This simple per-pixel approach requires the network to be trained for a given pixel footprint size, and will only work optimally with a fixed camera, and on objects rendered at the right distance.

Alternatively, and more generally, latent vectors can be defined on the vertices of a rectangular texel grid and smoothly interpolated within texels, which leads to smoothly changing resulting GNDF queries, due to the continuity of the generator network as a function of z. We find that interpolating the latent vectors gives a reasonable "morphed" GNDF. This is similar to previous GAN approaches for e.g. faces, where latent interpolation gives rise to usually valid intermediate faces [Karras et al. 2018]. This morphed GNDF is not equivalent to a bilinear blend of the initial GNDFs, but it is similarly acceptable as an approximation to the unknown intermediate GNDF.

For choosing the spacing in the UV-domain, we generally follow the convention of choosing a Gaussian footprint with standard deviation σ_p (typically a few microns), and making the spacing (grid cell side-length) for synthetic data generation equal to σ_p, thus getting some overlap of footprints. In effect, each training dataset is captured at a particular spatial scale and the object UV-mapping should preserve this. Latent vectors are then defined on vertices of the same grid. These heuristics may indeed not be optimal; previous work also does not provide answers about optimal or automatic footprint/scale selection, and this problem remains open.

For the brushed metal example, we define the latent vectors on the vertices of a texel grid. Each grid cell is anisotropic (rectangular), with a side ratio of 1:30. The latent vectors are smoothly interpolated within the grid cells, which interpolates GNDF images in a smooth non-linear way, resulting in the horizontally elongated features of brushed metal. Care must be taken to normalize the interpolated latent vectors in order to have the same norm distribution as original vectors.

6.2 Partial generator evaluation

To describe our partial evaluation approach, we first review the definition of standard and transposed convolution, then describe range bounding for transposed convolution, and finally discuss how to extend the range bounding idea to support full-featured generator networks.

6.2.1 Review of standard and transposed convolution. For simplicity, let's assume single-channel input and output images. A standard 1-dimensional convolution is a "gather"-type operation, where each output value is a linear combination of k input values, weighted by a kernel W. Similarly, a 2-dimensional convolution computes each output value as a weighted linear combination of the corresponding $k \times k$ values of the input.

More precisely, let $I(i, j)$ and $O(i, j)$ be the input and output images, respectively. Let $W(p, q)$ be the $k \times k$ convolution kernel. Let s be the *stride* of the convolution. The default stride of 1 achieves a one-to-one mapping between the image resolutions, while a stride of 2 causes the output resolution to be half the input resolution. This convolution can be written as:

$$O(i, j) = \sum_{p=0}^{k-1} \sum_{q=0}^{k-1} W(p, q)\, I(si + p, sj + q). \tag{7}$$

For simplicity, we assume that accessing the input image beyond its bounds simply returns zero; other boundary conditions (such as wrap-around) can be easily handled as well.

A transposed convolution is defined as the multiplication by the transpose of the sparse matrix that describes the (linear) convolution operation. Therefore, a transposed convolution can be seen as a "scatter" instead of a "gather" operation. This is most easily expressed using the following pseudocode:

(1) initialize $O(i, j)$ to 0
(2) for each input image pixel (i, j):
    - $O(si : si + k - 1, sj : sj + k - 1)$ += $I(i, j)\, W(:, :)$

That is, the entire kernel W is splatted into a $k \times k$ block of the output, weighted by the input pixel value $I(i, j)$. Note that in a transposed convolution, a stride of 2 will double the resolution of the output, compared to the input.

6.2.2 Range bounding for partial evaluation. Consider a transposed convolution operation with kernel size $k \times k$ and stride s. The key question that needs to be answered to enable partial evaluation is: given a block of interest in the output image, which block in the input image can influence the values in the output block?

Let the output block of interest be $B_o = [i_1, i_2] \times [j_1, j_2]$, where the ranges are assumed to include their endpoints. We would like to find the input block $B_i = [i'_1, i'_2] \times [j'_1, j'_2]$ such that values in B_i affect values in B_o. After some analysis, we find that

$$i'_1 = \left\lfloor \frac{i_1 - k + s}{s} \right\rfloor \qquad i'_2 = \left\lfloor \frac{i_2}{s} \right\rfloor \tag{8}$$

and similarly

$$j'_1 = \left\lfloor \frac{j_1 - k + s}{s} \right\rfloor \qquad j'_2 = \left\lfloor \frac{j_2}{s} \right\rfloor \tag{9}$$

If the computed input block goes beyond the input image dimensions, we simply clamp it. Therefore, for any desired block in the output image, the above formulas let us find the corresponding block of the input image, over which we need to iterate, to guarantee coverage of the desired output block.

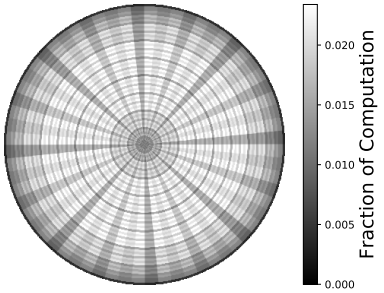


Fig. 9. Computational cost of partial evaluation, for a 2×2 range query, visualized as a fraction of the cost of doing full evaluation for different points on a GNDF. The image represents a circular domain in the projected hemisphere, though we use polar coordinates for the actual evaluation. The average cost of partial evaluation is just 1.75% compared to full evaluation. The variation in cost is due to the transposed convolution interacting with the padding/wraparound near the edges. We typically evaluate a 2×2 region instead of just 1×1, since we use bilinear interpolation in GNDF lookups.

6.2.3 Partial evaluation of full generator network. The above simplified situation needs to be extended to support the full set of features used in our generator network. This entails:

(1) Supporting multiple channels in the input and output images. This simply requires more computation when splatting into a given output pixel index.
(2) Supporting bias and leaky ReLU operations: these are per-element operations and can be handled easily.
(3) Supporting multiple layers. This is achieved by recursively propagating the block query backwards through the network. Boundary conditions (padding) need to be accounted for in the propagation.
(4) Supporting horizontal wraparound. This is slightly more technically involved, and requires blocks that cross the image boundary. First, we compute the ranges assuming no horizontal wraparound and the size of the input being infinite. Then, we clamp the horizontal ranges to their respective wrap sizes.

In summary, we find that this partial evaluation approach can compute any desired values of the output layer with no numerical error, while saving between 97.7% and 99.7% of the computation, depending on output size and query location. See Figure 9 for a visualization of the computational cost of partial evaluation. In practice, we implement the partial evaluation in C++ inside CPU rendering code (a Mitsuba BSDF plugin). The actual wall clock speedup of partial evaluation over full evaluation (using the same CPU code) is 150x. Our smaller memory footprint/caching enables an even greater (3x) speedup beyond that expected solely from the number of operations performed.

6.3 Discussion of alternatives

An alternative idea to our partial evaluation is to learn a network that takes the value of $\boldsymbol{d}$ as input, in addition to $\boldsymbol{z}$ (and $\boldsymbol{f}$), and directly

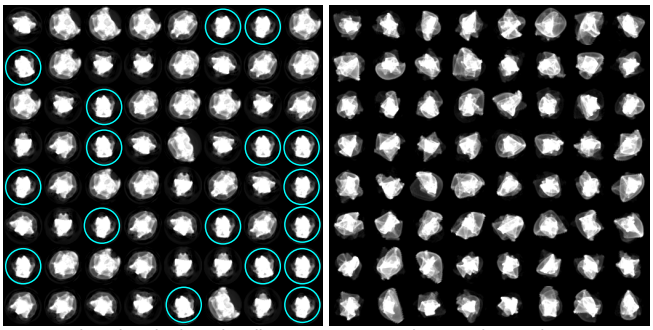


Fig. 10. Left: The alternative query-based method, taking the query vector $\boldsymbol{d}$ as input. In our experiments, this method exhibits mode collapse, where the same examples are repeated with minor variations; we mark one such case with circles but there are clearly others. Right: our partial generator evaluation has no such issue.

outputs the final value of the GNDF at that direction. We could still use an adversarial training scheme, evaluating the network for all $\boldsymbol{d}$ and passing the concatenated result to the discriminator as usual.

We have tried this idea in several different versions, but could not obtain good results. We experimented with different network architectures and different $\boldsymbol{d}$ parameterizations. They all generated results that are notably less accurate than with our partial evaluation approach. The results also suffered from mode collapse, so that the learned distribution failed to reflect the full variation of the training data; this is obvious from Figure 10.

Another serious issue with this approach is that training becomes very expensive. If we want to generate a single GNDF, we need to evaluate 64x64 = 4096 individual networks. Doing that "in parallel" does not help, as essentially none of the computation is shared. The backpropagation becomes slow and memory intensive.

Yet another alternative we considered is to evaluate appearance matching in the final renderings, as opposed to the GNDF domain. However, to obtain temporal glint coherence with light (or camera) movement, we would need to use videos instead of images, and the computational challenges are formidable.

7 IMPLEMENTATION DETAILS

Here we discuss more details about training dataset generation and including our method within full global illumination simulations.

7.1 Training data generation

All GANs are trained on datasets of size 65 and 75 thousand GNDF images. The synthetic datasets are obtained by uniformly covering the heightfield textures (or fiber-level cloth model) by 256^2 or 274^2 footprints and computing the corresponding GNDF images. The measured dataset is similarly obtained by taking a 256^2 window from the camera view.

7.2 Combining with other light paths

To render full global illumination, we need to combine the result of our technique with other light paths; there are several ways this can be done. Typically, the high-frequency directional effects are not visible in indirect and environment illumination. We therefore take the approach of computing the direct illumination on the glinty surface in a first pass. We compute other light paths (indirect and environment illumination) in separate passes, approximating the glinty material by a standard smooth microfacet BRDF with a matching roughness. The roughness of the smooth (average) GNDF for indirect/environment lighting can be obtained from the mean GNDF image of the dataset. Various other sampling schemes are also possible, but this approach currently gives the best performance for our example scenes.

Table 2. Comparison of our rendering time (our evaluation component only, excluding global illumination) versus the corresponding previous method for different material types.

Scene	Type	Our Time	Prev. Time
Macbook	Geom.	4.5s	2.0s
Macbook	Wave	5.9s	234s
Plate	Geom.	12.3s	45.8s
Phone	Geom.	6.4s	3.3s
Cloth	Fabric	10.1s	n/a

8 RESULTS AND DISCUSSION

In this section, we show the behavior of our approach on six different learned materials, showcased on five different scenes. We discuss the performance and storage of our method in comparison to previous work, and conclude with a discussion of limitations and future work. The performance details are given in Table 2.

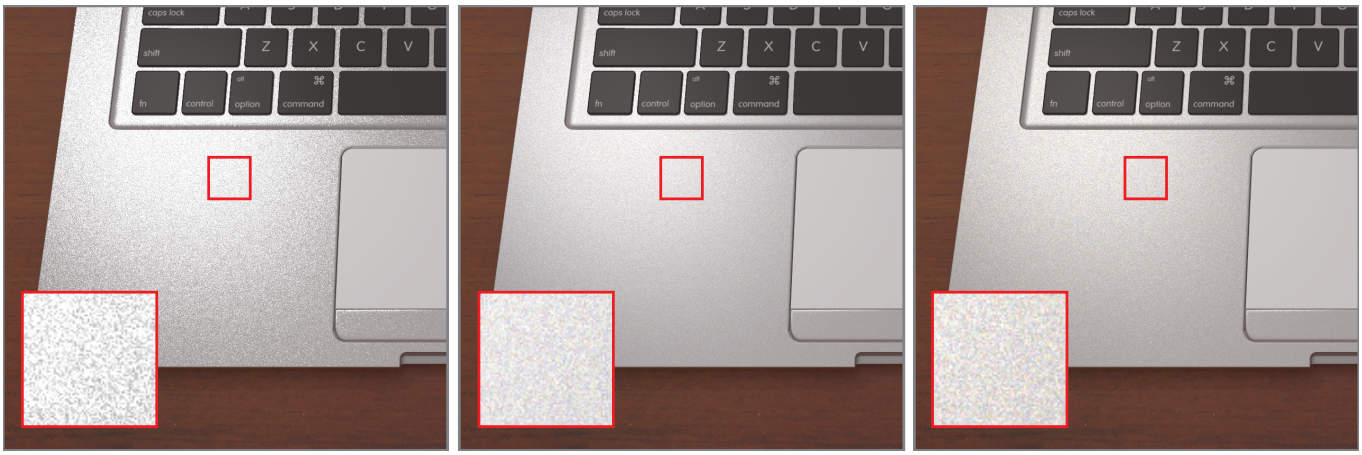
We implemented our rendering (including partial cGAN evaluation) in C++ as a CPU Mitsuba plugin. We run on an Intel 8-core i7-7820X machine, using an with NVIDIA 2080Ti GPU for training. The GAN training uses the PyTorch framework and takes around 3 hours per material. The synthetic dataset generation can take from 1 to 6 hours depending on the type of material. Currently our data synthesis is CPU-based, so the GPU was used only for training.

8.1 Rendered images

Our geometric optics result for an isotropic noise heightfield can be seen in Figure 11 (left), as well as in Figure 12 (left). In Figure 12 (right), we also provide a comparison to a result computed using the method of Yan et al. [2016]. Our method can achieve similar spatial and temporal patterns. While we can exceed the performance of the highly optimized previous work [Yan et al. 2016] only in some cases, our storage requirements are much lower. We also show a brushed metal material in Figure 14. This is using geometric optics and an anisotropic spacing of latent vectors in texture space, and also demonstrating a moving geometry.

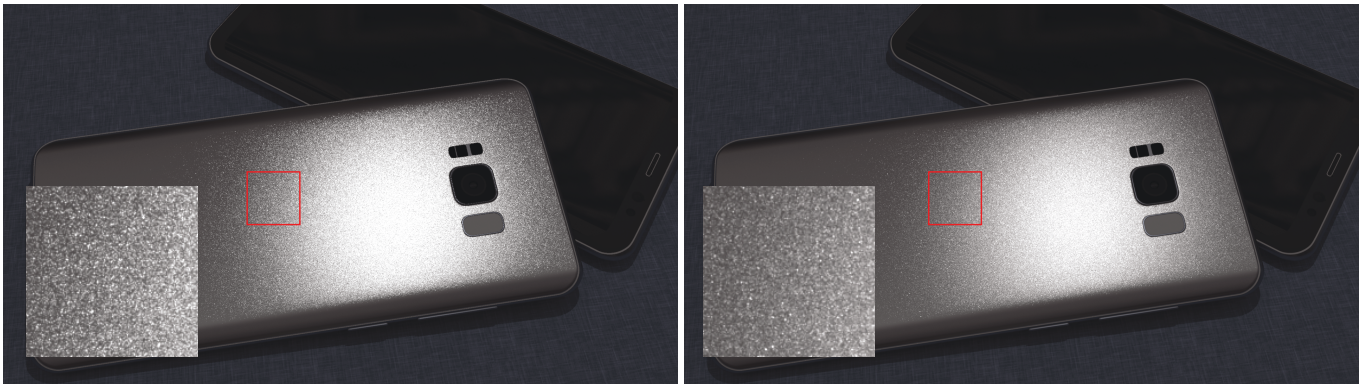
Our wave optics result can be seen in Figure 11 (middle), as well as in the teaser (Figure 1). A comparison with the wave optics method of Yan et al. [2018] is provided in Figure 11 (right). Again, our result is quite close in its spatial and temporal behavior, as well as the subtle color effects. The previous wave optics method is quite expensive, and our method has significantly faster performance, as well as lower storage requirements.

A result rendered with the measured GNDF dataset is shown in Figure 13 and compared to the naive approach of using the measured data directly. Our method is much more practical, not requiring the massive data storage, avoiding tiling artifacts, as well as learning to fill the holes. While it may well be possible to fix the missing



a) Our geometric GAN b) Our wave optics GAN c) [Yan et al. 2018] wave optics

Fig. 11. Laptop. From left to right: our geometric optics GAN, our wave optics GAN and Yan et al.'s [2018] wave optics method.



a) Our geometric GAN b) [Yan et al. 2016] geometric optics

Fig. 12. Phone. Comparison of our geometric optics GAN to Yan et al.'s [2016] explicit method for geometric optics.

a) Directly using the measured GNDFs a) Our GAN learned from the measured GNDFs

Fig. 13. The left image shows the naive approach of using the measured data for rendering directly. This is clearly not viable, and illustrates the need for some texture synthesis and hole filling in order to use this data effectively. Our GAN result learned from the measured GNDF dataset (right) shows closely matching highlight size and intensity, and similar glint appearance, but fixes the issues with holes and the visible tiling pattern.

Fig. 14. Tumbler. This scene demonstrates our anisotropic brushed metal extension.

data hole and render good images directly, without requiring our full framework, the drawback of having to store a large number of NDFs would remain. The fact that our method handles the storage and missing data issues cleanly is a notable advantage.

Figure 15 shows a scratched ceramic example using our cGAN technique. We show a comparison between [Yan et al. 2016] (left) and our method using a feature texture describing the same area as the original heightfield (middle). The images match fairly well, while our rendering is about 3 times faster. However, both images show a clear pattern repetition issue, as the original microstructure covers too little area. The image on the right shows our method rendered with a much larger feature texture, synthesized using image quilting [Efros and Freeman 2001], with a greatly reduced pattern repetition problem. This can be done at no performance cost, and with minimal storage increase for the larger feature texture.

Figure 16 shows a fabric example, with GNDFs constructed from fiber-level microgeometry, also using our cGAN technique to achieve seamless texturing. As above, we used image quilting to extend the feature texture. This example shows that our method can be applied to microgeometry types that are not expressible as heightfields. Finally, figure 17 shows six additional material variations, with different microgeometry leading to alternative GNDFs and feature maps, all of which can be handled by our framework.

We strongly encourage the reader to view the temporal behavior of these examples in the supplementary video, as static images can convey only a partial impression of our technique's capabilities.

8.2 Performance and storage comparison

Our method can reproduce the results of previous glints methods, but has multiple advantages over them. Since we pretrain our network, rendering time is independent of the complexity of optical simulation and the patch size covered by the pixel; this is not the case for previous methods. For example, in the case of Yan et al. [2018] wave optics, the simulation is very computationally expensive. Moreover, they use a fixed coherence kernel σ_c; where the pixel is several times larger than the coherence kernel (like in the laptop example), their method needs to be rendered with a higher sample count to cover the pixel. Our method can train the network with GNDF images already integrated for the pixel size. Rendering the laptop scene using the previous wave optics method took 234 seconds, while our method took just 5.9 seconds (both for the glints component only, ignoring global illumination).

Another advantage of our method is its low storage requirement. The size of our network is small (under 1.3 MB) and independent of the heightfield or training data size. In addition, because we do partial evaluation, we just need under 20 KB to evaluate it. On the other hand, the previous methods can require several GB to store the associated acceleration data structures, even for a relatively small heightfield of size 4k×4k. Moreover, in scenes containing multiple different glinty objects, the storage cost of previous methods quickly becomes prohibitive, while with our method we can easily have multiple networks for different materials.

8.3 Discussion and future work

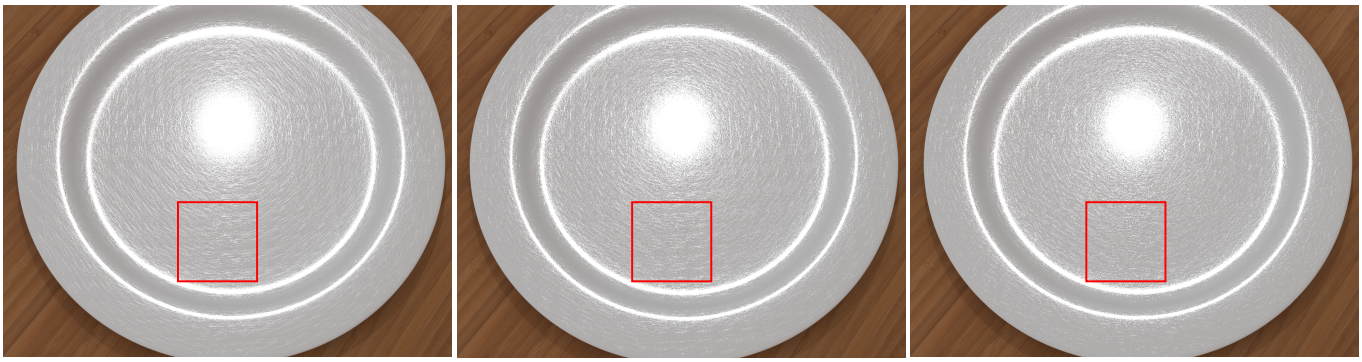
Comparing specular microgeometry renderings to ground truth is itself an open problem due to the stochastic nature of patterns, dependence on texture sampling and filtering, etc. However, by looking at GNDFs directly, we can see that our method can faithfully reproduce the overall appearance of the GNDFs in the synthetic or measured input data (Figure 3). In this sense, our BRDF evaluation is close to ground truth.

Currently, our generator is trained at a single resolution (footprint size). This works fine for rendering animations where objects are at a largely fixed distance from the camera; we can support some level of zoom-out but efficiency will eventually be lost. We are interested in extending the framework to learn the right material appearance across several different resolutions (footprint sizes).

The generative model could likely become more compact and even faster, by further exploring how small a network can still learn visually plausible models. The appearance training data can be produced from a broader range of microstructures, by ever more elaborate optical simulations, or by further exploration of measurement setups. Finally, it is possible that our approach could be extended beyond heightfield and fabric microstructures, to materials such as snow, foam, packed crystals, and more.

9 CONCLUSION

Rendering glinty highlights on stochastic surfaces provides an important improvement in the realism of images, especially when surfaces are viewed up close. Previous work has shown how to do this with reasonable efficiency, but improvements in the reflection model, progressing from flat mirror flakes through smooth surfaces under geometric optics to diffraction models for arbitrary heightfields, have come with progressively higher computational requirements. At the same time, there is no obvious way to set these methods up to match measurements of a specific real material. The



a) [Yan et al. 2016] geometric optics b) Our conditional GAN c) Our conditional GAN with texture synthesis

Fig. 15. Plate. From left to right: Yan et al.'s [2016] geometric optics method, our method on the same microgeometry (using a corresponding small feature texture), and our method using a much larger feature texture produced by texture synthesis. Note the repeating pattern due to small original microgeometry (for example, within the red square, but not limited to it) is addressed by our texture synthesis.

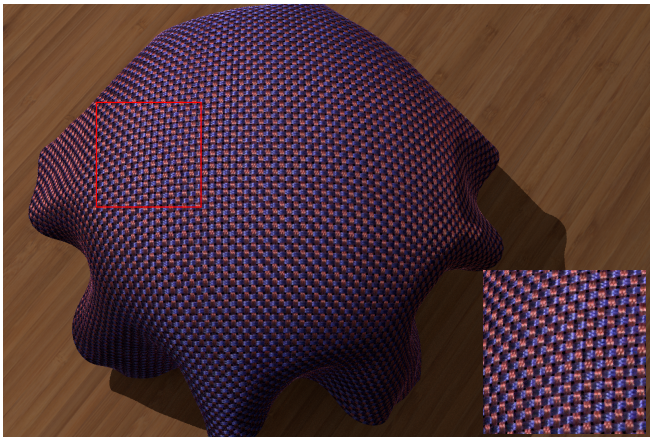


Fig. 16. Result generated with our fabric GNDF dataset.

method we have presented here shows a path forward by taking advantage of the stochastic nature of the surface. By using our GAN to capture the statistical variation of the generalized NDFs that determine surface appearance, we decouple the run-time computation from the source of the appearance data. This makes the cost of the optical computations inconsequential, while also making it equally easy to train the model on measured data. Finally, the storage required by the trained generator network is small, turning the network into a convenient material exchange format.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants 1703957 and 1704540, an Adobe Fellowship, the Ronald L. Graham chair, and the UC San Diego Center for Visual Computing. We thank Sitian Chen for generating fiber-level fabric microstructures. We also thank sriniwasjha for the phone model .

REFERENCES

Miika Aittala, Timo Aila, and Jaakko Lehtinen. 2016. Reflectance Modeling by Neural Texture Synthesis. *ACM Trans. Graph.* 35, 4, Article 65 (2016), 65:1–65:13 pages.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 214–223. http://proceedings.mlr.press/v70/arjovsky17a.html

Matt Jen-Yuan Chiang, Benedikt Bitterli, Chuck Tappan, and Brent Burley. 2016. A Practical and Controllable Hair and Fur Model for Production Path Tracing. *Computer Graphics Forum* 35, 2 (2016), 275–283.

R. L. Cook and K. E. Torrance. 1982. A Reflectance Model for Computer Graphics. *ACM Trans. Graph.* 1, 1 (1982), 7–24.

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.* 37, 4, Article 128 (July 2018), 128:1–128:15 pages.

Chris Donahue, Julian McAuley, and Miller Puckette. 2018. Synthesizing Audio with Generative Adversarial Networks. *CoRR* abs/1802.04208 (2018). arXiv:1802.04208 http://arxiv.org/abs/1802.04208

Zhao Dong, Bruce Walter, Steve Marschner, and Donald P. Greenberg. 2015. Predicting Appearance from Measured Microgeometry of Metal Surfaces. *ACM Trans. Graph.* 35, 1, Article 9 (2015), 13 pages. https://doi.org/10.1145/2815618

Alexei A. Efros and William T. Freeman. 2001. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. 341–346.

Gabriel Eilertsen, Joel Kronander, Gyorgy Denes, Rafał K. Mantiuk, and Jonas Unger. 2017. HDR Image Reconstruction from a Single Exposure Using Deep CNNs. *ACM Trans. Graph.* 36, 6, Article 178 (Nov. 2017), 15 pages.

Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. 2012. Gabor Noise by Example. *ACM Trans. Graph.* 31, 4, Article 73 (July 2012), 9 pages.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*. 2672–2680.

Paul Graham, Borom Tunwattanapong, Jay Busch, Xueming Yu, Andrew Jones, Paul Debevec, and Abhijeet Ghosh. 2013. Measurement-Based Synthesis of Facial Microgeometry. *Computer Graphics Forum* 32, 2pt3 (2013), 335–344.

Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. 2017. Globally and Locally Consistent Image Completion. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017)* 36, 4, Article 107 (2017), 107:1–107:14 pages.

Wenzel Jakob, Miloš Hašan, Ling-Qi Yan, Jason Lawrence, Ravi Ramamoorthi, and Steve Marschner. 2014. Discrete Stochastic Microfacet Models. *ACM Trans. Graph.* 33, 4 (2014).

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hk99zCeAb

Jonathan Leaf, Rundong Wu, Eston Schweickart, Doug L. James, and Steve Marschner. 2018. Interactive Design of Periodic Yarn-level Cloth Patterns. *ACM Trans. Graph.* 37, 6, Article 202 (Dec. 2018), 15 pages.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. 2017. Modeling Surface Appearance from a Single Photograph Using Self-augmented Convolutional Neural Networks. *ACM Trans. Graph.* 36, 4, Article 45 (July 2017), 11 pages.

Zhengqin Li, Kalyan Sunkavalli, and Manmohan Chandraker. 2018. Materials for Masses: SVBRDF Acquisition with a Single Mobile Phone Image. In *Computer Vision*

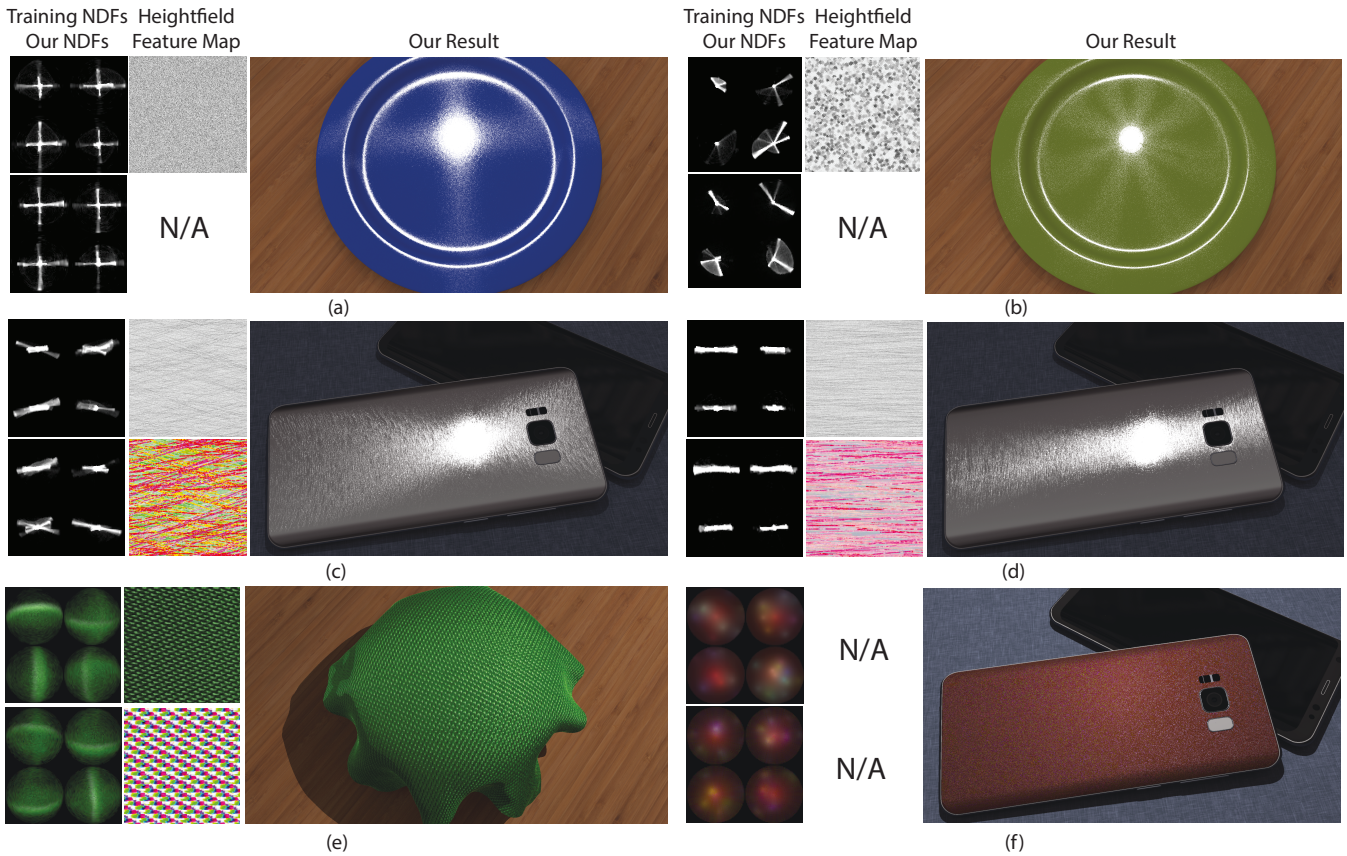


Fig. 17. Our method can be used to generate a wide variety of materials, represented as different GNDFs and (optionally) feature textures. From left to right: (a) A plate with a smoothed random box pattern as an underlying heightfield, causing the normals to be roughly axis-aligned. (b) A plate with 8-pointed stars as an underlying heightfield. As a result, it has normals roughly aligned with the star edges, causing 8 glinty rays out of the center of the highlight. (c) A phone with scratches with random directions limited to a 30° range. (d) Similar to (c), but scratch directions limited to a 10° range. (e) An alternative fabric weave pattern (plaine weave instead of basket weave). (f) A phone with a synthetic GNDF made with randomly placed colored Gaussian flakes.

- *ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part III (Lecture Notes in Computer Science)*, Vol. 11207. 74–90.

Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. *CoRR* abs/1411.1784 (2014). http://arxiv.org/abs/1411.1784

Koki Nagano, Graham Fyffe, Oleg Alexander, Jernej Barbiç, Hao Li, Abhijeet Ghosh, and Paul Debevec. 2015. Skin Microstructure Deformation with Displacement Map Convolution. *ACM Trans. Graph.* 34, 4, Article 109 (July 2015), 10 pages.

Giljoo Nam, Joo Ho Lee, Hongzhi Wu, Diego Gutierrez, and Min H. Kim. 2016. Simultaneous Acquisition of Microscale Reflectance and Normals. *ACM Trans. Graph.* 35, 6, Article 185 (Nov. 2016), 11 pages.

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR* abs/1511.06434 (2015). arXiv:1511.06434 http://arxiv.org/abs/1511.06434

Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. 2019. Neural BTF Compression and Interpolation. *Computer Graphics Forum* (2019). https://doi.org/10.1111/cgf.13633

Boris Raymond, Gael Guennebaud, and Pascal Barla. 2016. Multi-Scale Rendering of Scratched Materials using a Structured SV-BRDF Model. *ACM Transactions on Graphics* (July 2016). https://doi.org/10.1145/2897824.2925945

Szymon Rusinkiewicz. 1998. A New Change of Variables for Efficient BRDF Representation.. In *Rendering Techniques (Eurographics)*. Springer, 11–22.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. In *Advances in neural information processing systems*. 2234–2242.

Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. 2018. MoCoGAN: Decomposing Motion and Content for Video Generation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. 2007. Microfacet Models for Refraction Through Rough Surfaces *(EGSR 07)*. 195–206.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. 2018. High-Resolution Image Synthesis and Semantic Manipulation With Conditional GANs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sebastian Werner, Zdravko Velinov, Wenzel Jakob, and Matthias B. Hullin. 2017. Scratch Iridescence: Wave-optical Rendering of Diffractive Surface Structure. *ACM Trans. Graph.* 36, 6, Article 207 (2017), 14 pages. https://doi.org/10.1145/3130800.3130840

Ling-Qi Yan, Miloš Hašan, Wenzel Jakob, Jason Lawrence, Steve Marschner, and Ravi Ramamoorthi. 2014. Rendering Glints on High-resolution Normal-mapped Specular Surfaces. *ACM Trans. Graph.* 33, 4, Article 116 (2014), 9 pages. https://doi.org/10.1145/2601097.2601155

Ling-Qi Yan, Miloš Hašan, Steve Marschner, and Ravi Ramamoorthi. 2016. Position-normal Distributions for Efficient Rendering of Specular Microstructure. *ACM Trans. Graph.* 35, 4, Article 56 (2016), 9 pages. https://doi.org/10.1145/2897824.2925915

Ling-Qi Yan, Miloš Hašan, Bruce Walter, Steve Marschner, and Ravi Ramamoorthi. 2018. Rendering Specular Microgeometry with Wave Optics. *ACM Trans. Graph.* 37, 4 (2018).

Jinsong Zhang and Jean-Francois Lalonde. 2017. Learning High Dynamic Range From Outdoor Panoramas. In *The IEEE International Conference on Computer Vision (ICCV)*.

Shuang Zhao, Fujun Luan, and Kavita Bala. 2016. Fitting Procedural Yarn Models for Realistic Cloth Rendering. *ACM Trans. Graph.* 35, 4, Article 51 (July 2016), 11 pages.