Position-Normal Distributions for Efficient Rendering of Specular Microstructure

Ling-Qi Yan[1] Miloš Hašan[2] Steve Marschner[3] Ravi Ramamoorthi[4]
[1]University of California, Berkeley [2]Autodesk [3]Cornell University [4]University of California, San Diego

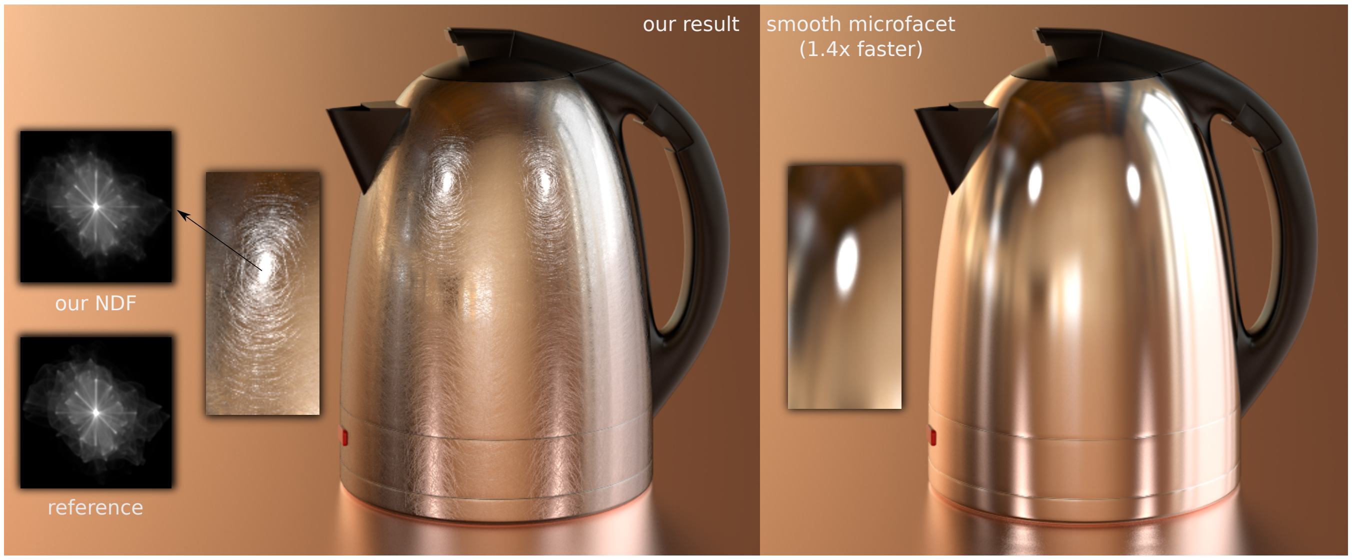


Figure 1: *A scratched stainless steel kettle rendered with our method (left). The kettle is lit by small area lights and an environment map, with surface microstructure modeled using a high-resolution normal map. Our method uses millions of 4D Gaussians to fit the position-normal distribution induced by the normal map; this lets us approximate the normal distribution function of a given pixel almost as accurately as Yan et al. [2014], but our evaluation is two orders of magnitude faster. Moreover, our technique can integrate area and environment lighting, and multiple importance sampling, which was not practical with Yan et al. [2014]. Our rendering takes only 1.4× longer than a standard microfacet BRDF rendering (right).*

Abstract

Specular BRDF rendering traditionally approximates surface microstructure using a smooth normal distribution, but this ignores glinty effects, easily observable in the real world. While modeling the actual surface microstructure is possible, the resulting rendering problem is prohibitively expensive. Recently, Yan et al. [2014] and Jakob et al. [2014] made progress on this problem, but their approaches are still expensive and lack full generality in their material and illumination support. We introduce an efficient and general method that can be easily integrated in a standard rendering system. We treat a specular surface as a four-dimensional *position-normal distribution*, and fit this distribution using millions of 4D Gaussians, which we call *elements*. This leads to closed-form solutions to the required BRDF evaluation and sampling queries, enabling the first practical solution to rendering specular microstructure.

Keywords: rendering, specular highlights, glints, surface microstructure, normal distribution function

Concepts: •**Computing methodologies** → **Rendering; Reflectance modeling;**

SIGGRAPH 2016, July 24-28, 2016, Anaheim, CA
ISBN: 978-1-4503-ABCD-E/16/07

1 Introduction

Sharp lighting exposes a large amount of structure in real-world specular highlights, especially in close-ups (like in Figure 1, left). However, the conventional microfacet BRDF models the detailed surface micro-structure using a smooth normal distribution function (NDF). While this leads to good results under distant views and smooth illumination, the high-frequency glint structure is lost (Figure 1, right). It is certainly possible to *model* explicit surface microstructure using normal mapping, but the resulting *rendering* problem is challenging for straightforward Monte Carlo approaches, because uniform pixel sampling misses the tiny, strong highlights.

Recently, there has been progress on resolving the full detail of real-world specular surfaces, in the work of Yan et al. [2014] and Jakob et al. [2014]. While the results demonstrated in their papers are compelling, these methods still have drawbacks. Yan et al. [2014] computes an accurate solution for a surface defined by an explicit high-resolution normal map. To achieve this, however, a fine tessellation of the normal map into triangular elements is necessary, and the contribution of a single element to the result requires a rather involved computation. Jakob et al. [2014] defines distributions and orientations of specular point scatterers implicitly; this has the advantage of low storage requirements, but only works for random piece-wise flat surfaces, and loses the ability to represent specific desired surfaces with local curvature. Neither method is fast or easy to integrate in a rendering system, which limits their practical impact.

DOI: http://doi.acm.org/10.1145/9999997.9999999

The traditional definition of a normal map is a function mapping a surface position to a normal. We instead map a surface position to a narrow Gaussian lobe of normals, thus thinking of a specular surface as a *position-normal distribution*: a function in the four-dimensional cross-product space of surface positions and normals. This function can be approximated as a mixture of millions of 4-dimensional Gaussians, which we call *elements*. (Their number is linear in the normal map size, typically 1-4 elements per texel.) Intuitively, each element represents a "fuzzy" patch of the surface, either flat (if the Gaussian is axis-aligned) or curved (for a general Gaussian). This works for both piece-wise constant elements (common in metallic paint) and smooth heightfields (which can be used to model most other surfaces, including bumps, brushes, and scratches). The 4D Gaussian primitive, combined with a Gaussian query representing a desired surface footprint and half-vector, leads to an efficient closed-form solution. Despite using millions of elements, a single NDF query will only depend on a handful of elements, which our acceleration hierarchy can efficiently isolate.

Our BRDF evaluation is over $100\times$ times faster than Yan et al. [2014]. This speed-up has significant benefits: it enables our method to be used as a standard BRDF in a Monte Carlo renderer, which is convenient from an engineering perspective. As a consequence, we can now use the BRDF with multiple importance sampling, and transparently handle all effects supported in Monte Carlo frameworks, such as illumination from environment maps and area lights (both used in Figure 1). While in theory, the method of Yan et al. [2014] could also be used as a standard BRDF in this fashion, the $100\times$ slow-down in evaluation performance would limit the practicality of that approach.

2 Related work

Normal maps are a well-known technique for adding detail to specular surfaces. The standard approach to compute direct illumination on a normal-mapped specular surface is to evaluate the normal at the hit point, and use a low-roughness BRDF to shade it. However, when truly high resolution maps are used in conjunction with sharp light sources, both Yan et al. [2014] and Jakob et al. [2014] make a clear case that this naive approach cannot be effective.

Normal map filtering techniques can deliver artifact-free renderings by approximating a pixel's NDF by a single lobe [Toksvig 2005; Olano and Baker 2010; Dupuy et al. 2013]. These approaches use a mip-map hierarchy on top of the normal map, together with a way to interpolate surface roughness in a linear way, such that increasingly large patches of a complex surface are approximated as flat surfaces with higher roughness. However, none of these methods can capture glinty appearance, because the sharp features of the true NDFs are lost when approximating them by a single smooth lobe.

The closest previous work to our approach, Yan et al. [2014] considers the actual, unsimplified NDF of a surface patch $\mathcal{P}$ seen through a single pixel. This $\mathcal{P}$-NDF can be easily sampled: simply choose a point on the patch and take its normal. However, for efficient direct illumination, the $\mathcal{P}$-NDF needs to be *evaluated* for a given half-vector, and a sampling procedure is of little help. They introduce an algorithm that evaluates $\mathcal{P}$-NDFs by turning the problem into integration over the normal map, finely discretized into triangles. This is accurate but quite slow. To get around the performance issue, a two-pass approach was used, which only supported direct illumination from point lights. In contrast, our the better performance of our method enables direct integration into standard, single-pass Monte Carlo rendering, with only minor slow-down (Figure 1).

Jakob et al. [2014] addresses the problem of glinty surfaces using a stochastic approach. They model the surface as a procedural random collection of specular flakes. The set of flakes is not given, but instead implicit in the construction of the hierarchical evaluation algorithm. Therefore, they can count the particles contributing to a particular illumination calculation without actually generating them, which is memory-efficient, and retains performance for distant views. On the other hand, the framework limits the kinds of surfaces expressible; brushed, scratched, and smooth bumpy surfaces cannot be represented. The effects achievable are essentially a subset of our method's capabilities, though we do require more storage (up to 16 million elements for a 2048^2 normal map).

A mixture of a small number of Gaussian (more precisely von-Mises-Fisher) lobes is used by Han et al. [2007]. This is related, but their work targets real-time effects under smooth illumination, which is a different end of the spectrum from our focus on sharp glints. Our mixture can contain millions of Gaussians, and a single $\mathcal{P}$-NDF can still have hundreds of Gaussians contributing to it, though a single half-vector query will only depend on a few.

Dong et al. [2015] acquire surface microgeometry of real brushed metals and develop a material model to fit and render this data. This is an interesting approach that could be combined with our work, to construct our position-normal distributions based on real data. Many other works in graphics explored applying small or large Gaussian mixtures to approximate other objects, e.g. environment maps [Tsai and Shih 2006; Xu et al. 2013] and volumetric distributions [Jakob et al. 2011].

symbol	**domain**	**definition**
$\perp$		invalid normal
$\mathcal{D}_\perp$		extended unit disk
$\mathbf{s} = (s, t)$	$\mathcal{D}$	normal (on proj. hemisphere)
$\mathbf{u} = (u, v)$	$\mathbb{R}^2$	texture space parameters
$n(\mathbf{u})$	$\mathbb{R}^2 \to \mathcal{D}$	normal map function
$\mathbf{J}(\mathbf{u})$	$\mathbb{R}^2 \to \mathbb{R}^{2\times 2}$	Jacobian of $n(\mathbf{u})$
$\mathcal{P}$		surface patch (pixel footprint)
$G_p(\mathbf{u})$	$\mathbb{R}^2 \to \mathbb{R}$	pixel Gaussian of patch $\mathcal{P}$
$\mathbf{u_0}$	$\mathbb{R}^2$	center of pixel Gaussian
$G_r(\mathbf{s})$	$\mathbb{R}^2 \to \mathbb{R}$	intrinsic roughness Gaussian
$D_{\mathcal{P}}(\mathbf{s})$	$\mathcal{D} \to \mathbb{R}$	normal dist. function for patch $\mathcal{P}$
h	$\mathbb{R}$	step size for seeding Gaussians
σ_h	$\mathbb{R}$	std. deviation of seed Gaussians
σ_r	$\mathbb{R}$	intrinsic roughness
$\mathcal{N}(u, s)$	$\mathbb{R}^4 \to \mathbb{R}$	4D mixture (position-normal dist.)
G_i	$\mathbb{R}^4 \to \mathbb{R}$	i-th 4D Gaussian in the mixture
m	integer	number of Gaussians

Table 1: *Notation used in the paper.*

3 $\mathcal{P}$-NDF theory and notation

First we will introduce several concepts from Yan et al. [2014], together with notation that we will use in the rest of the paper; Table 1 summarizes the symbols.

To render glints correctly, the reflectance over whole patches $\mathcal{P}$ of the surface needs to be considered as a $\mathcal{P}$-BRDF (function of $\mathcal{P}$ in addition to incoming and outgoing directions). The naive Monte Carlo rendering approach would instead attempt to compute this implicitly by pixel sampling, which fails because the highlights form a tiny fraction of the pixel, and the resulting estimate has unusably large variance.

In the microfacet framework, BRDF evaluation for an incoming direction turns into NDF evaluation for a half-vector (modulo a few terms such as cosines, Fresnel and shadowing/masking). In the case of a patch $\mathcal{P}$, we can evaluate the $\mathcal{P}$-BRDF by turning it into evaluation of the $\mathcal{P}$-NDF: the distribution of surface normals over the relevant surface patch $\mathcal{P}$. Most of the difficulty lies in the $\mathcal{P}$-NDF evaluation; the other terms can be included easily. (We currently approximate shadowing-masking by the Smith term for a Beckmann NDF of matching overall roughness. We include the Fresnel term for metals or dielectrics as appropriate.)

Our method requires a mapping of normals into a planar domain. We choose to represent normals as 2D vectors on the projected hemisphere (unit disk). More precisely, like Yan et al. [2014], we define the *extended unit disk* as the union of the unit disk with a special symbol $\perp$ for an invalid normal. This definition lets us convolve normal distributions on the unit disk with Gaussians, handling the tiny non-zero probability of invalid normals outside of the unit disk. Our method could potentially use other planar representations of normals, e.g. the domain of gradients [Olano and Baker 2010; Heitz 2014]. However, like Yan et al. [2014] we prefer the extended unit disk, as it assigns lowest measure to areas of the hemisphere near the horizon, which are least important for our purposes. Furthermore, an NDF is directly a pdf on our domain, i.e. it integrates to 1 without needing additional terms in the integrand.

A *normal map* can be defined as a function $n : \mathbb{R}^2 \to \mathcal{D}_\perp$ from points $\mathbf{u} = (u, v)$ in texture space to normals $\mathbf{s} = (s, t)$ on the extended unit disk, representing unit vectors $(s, t, \sqrt{1 - s^2 - t^2})$. The Jacobian of $n(\mathbf{u})$, denoted $\mathbf{J}(\mathbf{u})$, is a measure of local surface curvature, and comes up regularly in analyzing smooth specular surfaces.

The *footprint* $\mathcal{P}$ can be defined as a 2D Gaussian G_p in the uv-parameterization of the normal map. Its size and orientation can be computed by any common method used for texture mip-mapping, e.g. by ray differentials [Igehy 1999]. We currently set its standard deviation σ_p to 1/16th of the projected pixel radius; for 1024 stratified samples per pixel, this provides enough prefiltering that the strata become smooth.

Like Yan et al. [2014], we also treat the surface as having a tiny amount of *intrinsic roughness*, which prevents singularities (infinitely bright highlights), arising on perfectly specular surfaces when $\det \mathbf{J}(\mathbf{u}) = 0$. This is achieved by thinking of each normal as a 2D Gaussian G_r with a very small standard deviation.

Using these concepts, the $\mathcal{P}$-NDF for a footprint $\mathcal{P}$ can be defined as the probability distribution (on $\mathcal{D}$) of a random variable $n(X)$, defined by sampling the footprint Gaussian G_p, evaluating the normal at the sampled location, and perturbing the result by a sample from the intrinsic roughness Gaussian. (The perturbation is taken on $\mathcal{D}$, the projected hemisphere. This is slightly unusual, but works well and simplifies the framework considerably.) While sampling $\mathcal{P}$-NDFs is straightforward (directly from the above definition), evaluation for a given normal (half-vector) is more involved, and it is the core challenge of both Yan et al. [2014] and our method.

A key design decision of our approach is the representation of the high-resolution specular surface detail. We propose to represent the normals of the surface, with intrinsic roughness applied, as a 4-dimensional *position-normal distribution*. This distribution can be accurately approximated as a mixture of millions of 4-dimensional Gaussians (on the order of the number of texels). We will denote these Gaussians as *elements*. In a sense, our approach has some properties of both Jakob et al. [2014] (thinking of the surface as a collection of discrete reflectors) and Yan et al. [2014] (ability to handle explicit smooth or non-smooth surfaces).

Below, we first explain this idea in flatland (where the Gaussians are 2-dimensional), and the next section will transfer our intuitions to the full 4D case.

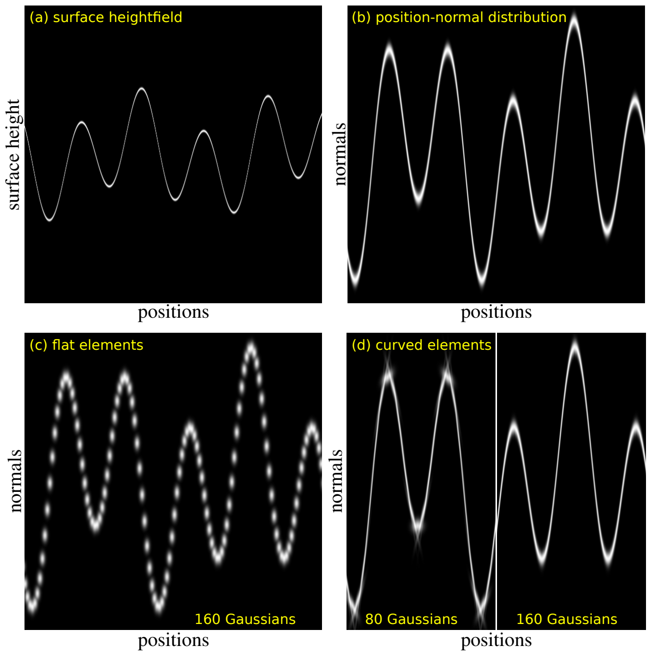


Figure 2: *A given flatland surface heightfield (a) is converted into a position-normal distribution, by taking the graph of its normal map function and blurring it vertically with the intrinsic roughness Gaussian (b). Approximating this distribution by axis-aligned Gaussians or "flat elements" (c) would clearly need much higher sampling to be accurate. In contrast, general Gaussians or "curved elements" (d) enable a much better approximation. Some individual Gaussians are still noticeable with 80 Gaussians but they disappear with 160.*

4 Mathematical framework in flatland

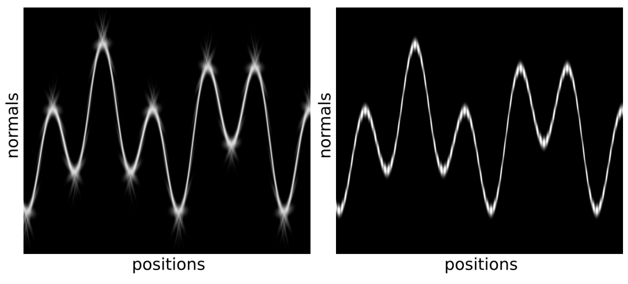
In flatland, the domain analogous to the unit disk (projected hemisphere) is simply the interval $[-1, 1]$. Yan et al. [2014] demonstrate that $\mathcal{P}$-NDF evaluation in flatland, with intrinsic roughness, can be written as:

$$\begin{aligned} D_{\mathcal{P}}(s) &= \int_{-1}^{1} G_r(s - s') \int_{-\infty}^{\infty} G_p(u)\delta(n(u) - s')\,\mathrm{d}u\,\mathrm{d}s' \\ &= \int_{-\infty}^{\infty} G_p(u) G_r(n(u) - s)\,\mathrm{d}u, \end{aligned} \qquad (1)$$

where $G_p(u)$ defines the footprint and s the normal (half-vector) of interest. G_r is the intrinsic roughness Gaussian with zero mean. The reasoning is: first define the pdf of the random variable $n(X)$, where X is a random variable whose pdf is $G_p(u)$. Second, convolve by intrinsic roughness G_r, and finally simplify. Note that both G_r and G_p are normalized (they integrate to 1).

Yan et al. [2014] proceed to evaluate the integral by assuming $n(u)$ is piecewise linear, which works but leads to harder integrals without closed form solutions. We instead precompute an approximation to the whole second factor, $G_r(n(u)-s)$. This function is what we propose to be our surface representation, the position-normal distribution $\mathcal{N}(u, s)$. The top row of Figure 2 demonstrates this: (a) shows a flatland surface and (b) the position-normal distribution of this surface. Note that normalization of $\mathcal{N}$ over u is not needed; it is sufficient that G_r is normalized over s.

Next, we propose to approximate $\mathcal{N}$ as a sum of 2D Gaussians in u and s, with scaling coefficients c_i, means $\mathbf{x}_i$ and covariance



(a) overshoot (b) clumping

Figure 3: *If the sampling rate is insufficient, or an overly large σ_h is used, this leads to overshoot (a): the Gaussians do not follow the distribution closely enough. In contrast, an overly small σ_h used for the seed Gaussians leads to a clumping artifact (b). Both problems can be avoided by using a sufficiently small step size h and setting $\sigma_h = h/\sqrt{8 \log 2}$.*

matrices $\mathbf{\Sigma}_i^{-1}$:

$$\mathcal{N}(u,s) = G_r(n(u) - s) \approx \sum_{i=1}^{m} G_i(u,s), \tag{2}$$

where

$$G_i(u,s) = c_i \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \mathbf{x}_i)\right) \tag{3}$$

and $\mathbf{x} = (u,s)^T$. A key advantage of this representation is that the $\mathcal{P}$-NDF query (for a given $\mathcal{P}$ and s) can be written as:

$$D_{\mathcal{P}}(s) = \int_{-\infty}^{\infty} G_p(u)\mathcal{N}(u,s)\,\mathrm{d}u \approx \sum_{i=1}^{m} \int_{-\infty}^{\infty} G_p(u) G_i(u,s)\,\mathrm{d}u. \tag{4}$$

Conveniently, the integral inside the sum has an efficient closed-form solution: as s is fixed, this is a product of two 1D Gaussians, which is just another Gaussian, easily integrable on the infinite domain. Fortunately, this convenience will carry over to the full 4D case. This is in contrast to Yan et al. [2014], whose approach leads to much harder finite-domain integrals.

4.1 Creating a Gaussian mixture

A key question is how to create the Gaussian mixture approximation to $\mathcal{N}(u,s)$. It would be interesting to consider hierarchical EM techniques [Verbeek et al. 2006; Jakob et al. 2011], but typical applications of EM start from discrete samples. In our problem, the distribution is available explicitly. Specifically, we have access to the Jacobian of the normal map, letting us tightly align anisotropic Gaussians to surface curvature. EM would also introduce randomness, losing the guarantee that animated normal data will remain temporally coherent. Therefore we use a more direct approach specialized for our application.

Flat and curved elements. When creating the Gaussian mixture representing a surface heightfield, we can think about approximating the *heightfield* as locally first-order (flat), or we can approximate the *normal map* as locally first-order, essentially making the heightfield approximation second-order (curved). In the following, we will explore and compare both approaches. We will define *flat elements* as ones given by axis-aligned Gaussians (i.e. ones with a diagonal covariance matrix); intuitively, these represent flat surface patches with a constant normal. We also define *curved elements* to be given by general Gaussians; these represent patches with approximately first-order local normal variation, thus a second-order variation of the surface heightfield.

Seed points. As a first step, we distribute m seed points u_i in the normal map domain. Each of these will be converted into an element by considering the value, and optionally the derivative, of the normal map function at u_i. The simplest approach to choose the seed locations is uniform sampling with step h; the ideal value of h will depend on the frequency content of the normal map function.

Next, we need to choose the standard deviation σ_h of the Gaussians in the normal map domain. Consider two Gaussians with standard deviation of σ_h whose centers are h apart; we would like them to decay to half of their peak value exactly at the midpoint between them. Setting $\sigma_h = h/\sqrt{8 \log 2}$ achieves this, and works well in practice.

Flat elements. To convert a normal map into flat elements, we are approximating the normal $n(u)$ as locally constant; that is, the surface is assumed locally *flat* near u_i. The i-th Gaussian we are creating can be written as:

$$G_i(u,s) = c_i \underbrace{\exp\left(-\frac{(u-u_i)^2}{2\sigma_h^2}\right)}_{\text{position band}} \underbrace{\exp\left(-\frac{(s-n(u_i))^2}{2\sigma_r^2}\right)}_{\text{normal band}}. \tag{5}$$

This results in a 2D Gaussian, whose inverse covariance matrix is diagonal, with values $1/\sigma_h^2$ and $1/\sigma_r^2$ on the diagonal. We can see the above 2D Gaussian as a product of a Gaussian band in positions around the sampling point u_i, and a Gaussian band in normals around the value $n(u_i)$. The resulting approximation can be seen in Figure 2(c): we can see how axis-aligned Gaussians are suboptimal for representing the position-normal distribution of a smooth surface.

Curved elements. For curved elements, we are intuitively trying to make the Gaussian locally align with the position-normal distribution, by approximating the normal map function $n(u)$ as linear (first-order) throughout the element. To achieve this, we replace the above normal band around $n(u_i)$ with a "sheared" band, which follows the first-order expansion $n(u) \approx n(u_i) + n'(u_i)(u - u_i)$. This leads to the following definition:

$$G_i(u,s) = c_i \exp\left(-\frac{(u-u_i)^2}{2\sigma_h^2}\right) \exp\left(-\frac{(s-n(u_i)-n'(u_i)(u-u_i))^2}{2\sigma_r^2}\right). \tag{6}$$

By using the shorthand notation $\delta u = u - u_i$, $\delta s = s - n(u_i)$ and $n' = n'(u_i)$, we can write this as:

$$G_i(u,s) = c_i \exp\left(-\frac{\delta u^2}{2\sigma_h^2}\right) \exp\left(-\frac{n'^2 \delta u^2 - 2n' \delta u \delta s + \delta s^2}{2\sigma_r^2}\right). \tag{7}$$

This lets us write the 2×2 inverse covariance matrix of this Gaussian as follows:

$$\mathbf{\Sigma}_i^{-1} = \frac{1}{\sigma_h^2}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{\sigma_r^2}\begin{pmatrix} n'^2 & -n' \\ -n' & 1 \end{pmatrix}. \tag{8}$$

The result can be seen in Figure 2(d): the position-normal distribution is approximated much better, with subtle artifacts still visible with 80 elements, but barely any error with 160 elements.

The seed size σ_h and the sampling step h need to be chosen well, otherwise approximation artifacts will result (Figure 3). The seed size can be fixed by following our σ_h heuristic, but the ideal step size will depend on the normal map frequency content. For well-sampled normal maps without excessive high frequencies, we found setting h to half a texel was sufficient for accurate results. More discussion of step size (and the resulting storage requirements) can be found in the results section.

5 Full algorithm in 4D

In this section, we will transfer the flatland derivations to 4D. Afterwards, to turn the framework into a full algorithm, we will discuss importance sampling and acceleration of the algorithm using a 4D hierarchy.

5.1 Framework and mixture construction in 4D

For a 2D normal map, we can similarly define the position-normal distribution, and approximate it as a mixture of Gaussians:

$$\mathcal{N}(\mathbf{u},\mathbf{s}) = G_r(\mathbf{n}(\mathbf{u}) - \mathbf{s}) \approx \sum_{i=1}^{m} G_i(\mathbf{u},\mathbf{s}), \tag{9}$$

where the G_i are 4-dimensional Gaussians in positions and normals. They can be represented as triples $(c_i, \mathbf{x}_i, \mathbf{\Sigma}_i)$ of a scaling coefficient, center and a 4×4 covariance matrix:

$$G_i(\mathbf{x}) = c_i \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^T \mathbf{\Sigma}_i^{-1}(\mathbf{x}-\mathbf{x}_i)\right), \tag{10}$$

where $\mathbf{x} = (\mathbf{u},\mathbf{s})^T$ is a 4D column vector. The constant c_i should be chosen such that the integral of G_i equals the area of the normal map it represents, e.g. 1/4 of a texel's area, when using 4 elements per texel. Again, we can evaluate the $\mathcal{P}$-NDF efficiently:

$$D_{\mathcal{P}}(\mathbf{s}) = \int_{\mathbb{R}^2} G_p(\mathbf{u})\mathcal{N}(\mathbf{u},\mathbf{s})\,\mathrm{d}\mathbf{u} \approx \sum_{i=1}^{m}\int_{\mathbb{R}^2} G_p(\mathbf{u}) G_i(\mathbf{u},\mathbf{s})\,\mathrm{d}\mathbf{u}, \tag{11}$$

where the latter integrand collapses to a 2D Gaussian, leading to a closed form solution. Please refer to the Appendix for details.

Like in flatland, we can create the Gaussian mixture by distributing seed points in texture space, and turning each seed point into either a flat or curved element, as detailed below.

Flat 4D elements. To convert a 2D normal map into flat elements, we define $\mathbf{s}_i = n(\mathbf{u}_i)$, $\delta\mathbf{u} = \mathbf{u} - \mathbf{u}_i$ and $\delta\mathbf{s} = \mathbf{s} - \mathbf{s}_i$. Extending the flatland idea to a 4D Gaussian is straightforward:

$$G_i(\mathbf{u},\mathbf{s}) = c_i \exp\left(-\frac{\|\delta\mathbf{u}\|^2}{2\sigma_h^2}\right)\exp\left(-\frac{\|\delta\mathbf{s}\|^2}{2\sigma_r^2}\right). \tag{12}$$

This covariance matrix can be easily seen to be diagonal: $\mathbf{\Sigma}_i^{-1} = \mathrm{diag}(\sigma_h^2, \sigma_h^2, \sigma_r^2, \sigma_r^2)^{-1}$.

As an aside, a mixture representing a collection of flat elements (as in metallic paint) does not necessarily need to be sampled from an underlying normal map. It can also be created directly: we can use Poisson sampling [Dunbar and Humphreys 2006] to distribute seed Gaussians, and assign to each a random constant normal (e.g. drawn from the Beckmann distribution). This is used for our metallic paint flakes approximation.

Curved 4D elements. Here we are approximating the normal function $n(\mathbf{u})$ as linear: $n(\mathbf{u}) \approx \mathbf{s}_i + \mathbf{J}\delta\mathbf{u}$, where $\mathbf{J}$ is the Jacobian of n at $\mathbf{u}_i$. The Gaussian then becomes:

$$G_i(\mathbf{u},\mathbf{s}) = c_i \exp\left(-\frac{\|\delta\mathbf{u}\|^2}{2\sigma_h^2}\right)\exp\left(-\frac{\|\delta\mathbf{s} - J\delta\mathbf{u}\|^2}{2\sigma_r^2}\right). \tag{13}$$

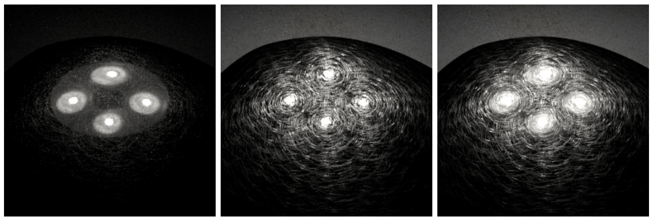

(a) sampling (b) evaluation (c) combined

Figure 4: *Our material model can be used inside a standard BRDF sampling/evaluation framework with multiple importance sampling. BRDF sampling alone (left) captures the reflection of the light through the flat areas of the map, but is suboptimal for rendering the scratches. Light sampling (middle), using our fast $\mathcal{P}$-NDF evaluation under the hood, captures illumination from the high-intensity parts of the HDR light texture onto the scratches. The combined result (c) has the benefits of both estimators.*

By expanding $\|\delta\mathbf{s} - J\delta\mathbf{u}\|^2$, we find that the 4×4 inverse covariance matrix of this Gaussian can be written (using block notation) as:

$$\mathbf{\Sigma}_i^{-1} = \frac{1}{\sigma_h^2}\begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} + \frac{1}{\sigma_r^2}\begin{pmatrix} \mathbf{J}^T\mathbf{J} & -\mathbf{J}^T \\ -\mathbf{J} & \mathbf{I} \end{pmatrix}. \tag{14}$$

5.2 Importance sampling

The above sections dealt with $\mathcal{P}$-NDF evaluation, which is the more challenging part, but for integration in Monte Carlo rendering, we also need to be able to sample $\mathcal{P}$-NDFs on a given footprint $\mathcal{P}$.

We could sample the underlying normal map from which the Gaussian mixture is generated, and perturb the sampled normal by the intrinsic roughness. This is the same approach we use for generating a ground truth $\mathcal{P}$-NDF. An alternative way is to sample from the Gaussian mixture directly, picking an element proportional to its contribution to the footprint, then picking a normal from that element. The advantage of the first is speed and a simple implementation. However, it only works for mixtures that have been constructed from a normal map, which need not be true in general (our metallic flakes in the top of Figure 8 are such an example). Furthermore, the latter method samples a distribution that matches our evaluation exactly, instead of just approximately. Therefore, we use the latter option in our results.

Having the ability to evaluate and sample the $\mathcal{P}$-NDF enables us to fully integrate the algorithm into a multiple importance-sampling framework [Veach 1997], and support various sources of illumination: environment maps and area lights, including ones with HDR emission textures. In theory, the method of Yan et al. [2014] can also be integrated into an MIS framework, but their $\mathcal{P}$-NDF evaluation is so much slower than sampling that this would have limited practicality. Figure 4 shows a scratched surface rendered under a textured area light. We separate the two estimators (sampling and evaluation); the components include the weights, so that the sum of the separated images is the combined image. The separation we observe supports natural intuition: the reflection of the light through the flat parts of the scratched map is easy to render by BRDF sampling (a), but shading the scratches requires sampling the high-intensity spots on the lightsource and evaluating the $\mathcal{P}$-BRDF and thus the $\mathcal{P}$-NDF in that direction (b). The combined image (c) has the benefits of both.

5.3 Acceleration hierarchy

The number m of 4D Gaussian elements in our mixture will be in the millions: a typical sampling step h is about half texel to one texel, which for a 2048^2 normal map will produce 16 million elements. Clearly, only a tiny number of these Gaussians will have a non-negligible contribution to a given query footprint $G_p(\mathbf{u})$ and a given query half-vector $\mathbf{s}$. This problem is identical to one faced by Jakob et al. [2014] and Yan et al. [2014], except the primitives they are bounding are point reflectors and triangles, respectively. They address this issue by 4D bounding-volume hierarchies over the (u, v, s, t)-space, and we follow a similar approach with some improvements.

A straightforward approach would be to create a 4D bounding box for each Gaussian element (treating their contribution as negligible beyond some distance, e.g. using a 3σ rule), and build a hierarchy on these bounding boxes in a top-down manner, using median splits. A $\mathcal{P}$-NDF evaluation query in this hierarchy is given by a rectangle in (u, v)-space, bounding the footprint Gaussian $G_p(\mathbf{u})$, and a point in (s, t) space, specifying the half-vector of interest. The contributing Gaussians can be found by a top-down traversal, pruning bounding boxes with no intersection with the query.

We found this natural approach works, but its performance can be enhanced by two additional ideas. First, we can trade off storage for performance: instead of having a single hierarchy, we subdivide (s, t)-space into a few hundred cells (sub-domains), and build a much smaller hierarchy per cell. Our query is point-wise in (s, t), so each query can be immediately answered by a single one of these smaller sub-hierarchies, and can therefore be faster. Of course, this is at the expense of extra storage, since an element will commonly occur in multiple sub-hierarchies.

An additional speedup can be achieved by stopping the build process earlier (at about 5 Gaussians per leaf node), since our closed-form solution per Gaussian is fast, and at some level becomes less of a bottleneck than traversal operations.

Finally, since our $\mathcal{P}$-NDF sampling is done using the element approximation, instead of sampling the underlying normal map, we need to accelerate the sampling operation. Here we just need to sample the footprint Gaussian, obtaining a (u, v) pair, and then quickly find all elements that contribute to this pair. We simply use a separate 2D hierarchy over the (u, v) domain to answer this query.

6 Results

We implemented the algorithm in C++. For rendering final images, we integrated our $\mathcal{P}$-NDF evaluation and sampling code in the Mitsuba framework [Jakob 2010].

6.1 Correctness and performance

First, we compare the $\mathcal{P}$-NDFs computed using our method to Yan et al. [2014]; their evaluation is accurate and can be treated as ground truth. Figure 5 (left images) shows a good match between our $\mathcal{P}$-NDF evaluation and theirs, while demonstrating a 130× speedup. One may think that the speed-up is because of overly fine subdivision used by Yan et al. (32 triangles per texel). However, it turns out that decreasing the subdivision rate of their method significantly reduces the quality and is still 12× slower than our method.

In addition to comparing $\mathcal{P}$-NDFs, we also directly compare the ground truth position-normal distribution $\mathcal{N}(\mathbf{u}, \mathbf{s})$ to our approximation using Gaussian mixtures. This can be seen in Figure 5 (right images). Since these are 4D distributions, we visualize them by slicing along a small segment of a normal map row, and integrating along the t-component of the normal. This shows that our approximation matches the ground truth well, if the right sampling step is chosen.

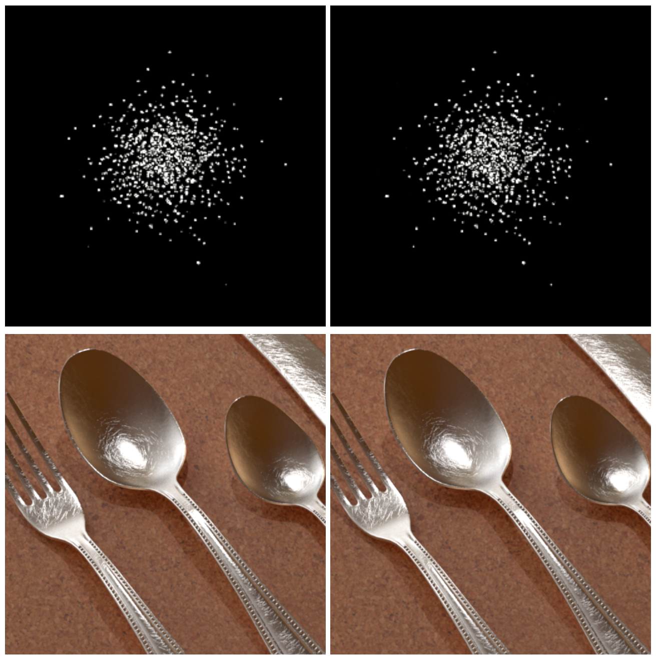


Figure 6: *Rendering comparison to Yan et al. [2014] (with full quality settings), demonstrating that a close match in the $\mathcal{P}$-NDF values translates to a close match in renderings. We separate the specular component of the image and use an identical sampling pattern for a direct comparison. Top: direct specular component on a simple curved surface, showing a match down to specific glints. Bottom: cutlery scene with additional light transport added. We achieve a speedup of about 32× in rendering the specular component.*

We also compare to Yan et al. [2014] on an actual rendering in Figure 6. Here, for a direct comparison, we separate the specular component and use a point light like their method; we also use the same sampling pattern. We get closely matching results, with a speedup of 32× in computing the direct specular component (this is less than in Figure 5, because other operations become important, notably tracing eye and shadow rays).

6.2 Effect of sampling rate on quality

Of course, a key issue is whether the step size h delivers sufficient accuracy, and this depends on the specific normal map, and on the desired intrinsic roughness σ_r. In our experience, setting h to half a texel size delivers accurate results for $\sigma_r = 0.005$, for typical normal maps that do not contain excessive fine noise. We use these settings in our final image and video results.

Figure 7 shows the result of varying the step size h, setting it to 0.5, 1, and 2 texels. This corresponds to 4, 1, and 0.25 elements per texel, respectively. With 4 elements per texel, both the image and the $\mathcal{P}$-NDF visualization show no artifacts. With one element per texel, some error can be seen in the $\mathcal{P}$-NDF visualization but the rendering itself is barely degraded. With one element per 4 texels, the $\mathcal{P}$-NDF image shows obvious overshoot and individual Gaus-

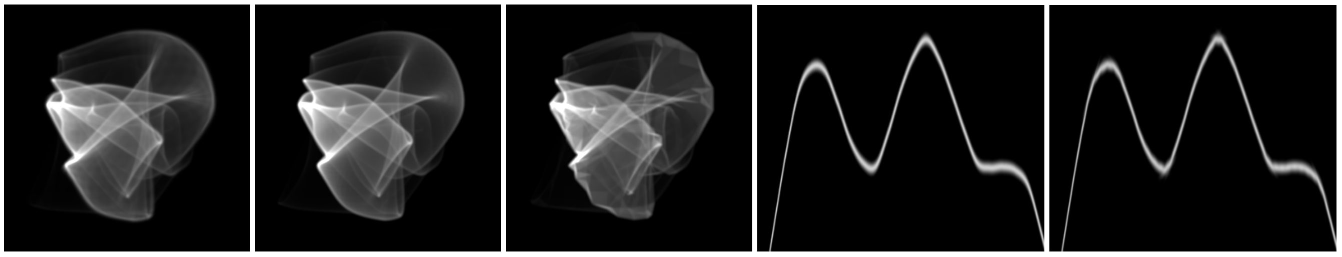

(a) 24 sec
Our method
(b) 3114 sec
[Yan 2014], 32 Δ/texel
(c) 300 sec
[Yan 2014], 2 Δ/texel
(d) Ground truth
Slice of $\mathcal{N}(\mathbf{u}, \mathbf{s})$ (full 4D)
(e) Our approximation
Slice of $\mathcal{N}(\mathbf{u}, \mathbf{s})$ (full 4D)

Figure 5: Left 3 images: *$\mathcal{P}$-NDF images evaluated in single-threaded C++; the timings are for evaluating the visualizations themselves, not final renderings. Our method (a) using Gaussian mixtures is about $130\times$ faster than Yan et al. [2014] when the normal map is discretized using 32 triangles per texel, as used in their paper (b). The match is very good, because our Gaussian mixture fits the position-normal distribution well. Our method is, moreover, still more than $12\times$ faster than their approach if using only 2 triangles per texel (c), and has higher quality.* **Right 2 images:** *Accuracy of fitting the actual 4D position-normal distribution $\mathcal{N}(\mathbf{u}, \mathbf{s})$ with our Gaussian mixture. The full distributions are four-dimensional and cannot be visualized directly. Therefore we slice the space by taking a small segment of a normal map scanline, and show only the s-component of normals, integrating along the t-component. Note how our approximation is accurate, with minimal overshoot.*

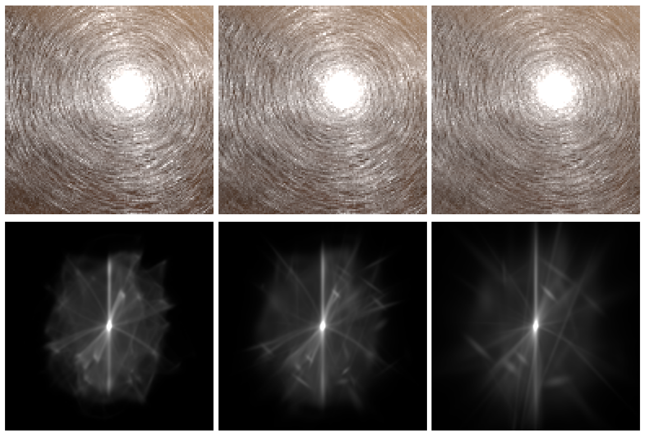

(a) $h = 0.5$ texel (b) $h = 1$ texel (c) $h = 2$ texels

Figure 7: *Comparison of different sampling steps h. When setting h to half a texel (a), i.e. 4 elements per texel, neither the image nor the $\mathcal{P}$-NDF visualization show approximation artifacts; this setting is what we use in our final results. When $h = 1$ texel (b), i.e. one element per texel, the rendering is only slightly degraded but some individual Gaussians become visible in the $\mathcal{P}$-NDF visualization (b). With $h = 2$ texels (c), i.e. one element per 4 texels, the rendered highlight starts showing signs of lower contrast, while the $\mathcal{P}$-NDF image clearly shows overshoot and individual Gaussians.*

sians; this causes directional blurring, whose effect on the rendering is some loss of specular contrast.

6.3 Effect of element sampling rate on quality

Of course, a key issue is whether the step size h delivers sufficient accuracy, and this depends on the specific normal map, and on the desired intrinsic roughness σ_r. In our experience, setting h to half a texel size delivers accurate results for $\sigma_r = 0.005$, for typical normal maps that do not contain excessive fine noise. We use these settings in our final image and video results.

Figure 7 shows the result of varying the step size h, setting it to 0.5, 1, and 2 texels. This corresponds to 4, 1, and 0.25 elements per texel, respectively. With 4 elements per texel, both the image and the $\mathcal{P}$-NDF visualization are accurate. With one element per texel, some error can be seen in the $\mathcal{P}$-NDF visualization but the rendering itself is barely degraded. With one element per 4 texels, the $\mathcal{P}$-NDF image shows obvious overshoot and individual Gaussians; this causes directional blurring, whose effect on the rendering is some loss of specular contrast.

6.4 Final renderings

We also illustrate our method's capabilities on final image renderings, shown in Figure 1 and 8. Results of this kind would not be easily achievable by the method of Jakob et al. [2014] (due to lack of support for smooth explicit normal maps), or Yan et al. [2014] (due to lack of support for high-frequency environment map and area lighting). Please make sure to watch the temporal versions of these results in the included video.

Kettle. This scene (Figure 1) demonstrates a glinty scratched stainless steel material similar to one shown previously by Yan et al. [2014]. However, note that our rendering is lit by area and environment lighting without multiple passes or other special handling. In fact, our performance is only $1.4\times$ slower than the same scene rendered with a traditional microfacet BRDF with no glinty behavior (2.65 vs. 1.88 min).

Car door. This scene is shown in Figure 8, top. The material combines two different glinty effects. A top coating is modeled by a scratched normal map, represented using curved elements. The bottom layer of metallic flakes embedded in the paint under it is represented using flat elements and has been created directly by Poisson sampling, without the need for a normal map. Lighting is from an environment map combined with a point light.

Wood floor and leather sofa. This scene is shown in Figure 8, bottom. It shows a leather sofa on a wood floor. It is demonstrating an additional useful feature of combining a macro-scale bump map (handled using the standard approach of shading frame perturbation) with a micro-scale normal map handled using our $\mathcal{P}$-BRDF approach, similar to [Zirr and Kaplanyan 2016]. This lets our method render materials that have interesting structure at multiple scales, which would require immensely large normal maps to represent using a naive single-map approach.

Time and storage. All images are rendered at 1280×720, with 1024 samples per pixel (1600 for leather sofa). These sampling rates were used to achieve low Monte Carlo noise in the videos; good still images can be produced with fewer samples. The timings on a 36-thread Amazon EC2 machine (c4.8xlarge) were 2.6 min (kettle), 6.8 min (car door) and 7.6 min (leather sofa).

The storage requirements depend on normal map size and sampling rate. Our results use a 2048^2 texture and 4 elements per texel, which requires 720M to store the Gaussians and an additional 400M for the acceleration hierarchy. If using 1 element per texel (which may well be sufficient for most applications), the costs would be 180M and 100M respectively. The normal map itself would take 32M (in floating point precision).

7 Conclusion and future work

We presented an algorithm for efficient glint rendering on highly complex specular surfaces. We represent a surface as a *position-normal distribution*. We approximate this 4D distribution as a mixture of Gaussian *elements*. Combined with a Gaussian query for a given surface footprint and half-vector, this formulation admits an efficient closed-form solution, which can be accelerated using a 4D bounding box hierarchy. Our method is fast enough to be treated as a standard BRDF in a typical Monte Carlo path-tracer, along with the benefits of supporting multiple importance sampling, environment maps, and area lights (including ones with HDR emission textures). We believe our method will enable high-quality specular microstructure rendering to be integrated in practical systems.

As future work, we would like to consider surfaces whose normals are very rugged, or which are not even differentiable in the traditional sense, but Gaussian mixture approximations to their behavior should still exist. Adaptively distributing elements based on curvature may further reduce the number of elements needed. We are also interested in extending the framework to handle multi-resolution representations and refractive interfaces.

Acknowledgements

This work was supported in part by NSF grant 1451828, an Autodesk BUILD Grant, Intel, and the UC San Diego Center for Visual Computing.

References

DONG, Z., WALTER, B., MARSCHNER, S., AND GREENBERG, D. P. 2015. Predicting appearance from measured microgeometry of metal surfaces. *ACM Trans. Graph. 35*, 1.

DUNBAR, D., AND HUMPHREYS, G. 2006. A spatial data structure for fast poisson-disk sample generation. *ACM Trans. Graph. 25*, 3.

DUPUY, J., HEITZ, E., IEHL, J.-C., POULIN, P., NEYRET, F., AND OSTROMOUKHOV, V. 2013. Linear Efficient Antialiased Displacement and Reflectance Mapping. *ACM Trans. Graph. 32*, 6.

HAN, C., SUN, B., RAMAMOORTHI, R., AND GRINSPUN, E. 2007. Frequency domain normal map filtering. *ACM Trans. Graph. 26*, 3.

HEITZ, E. 2014. Understanding the masking-shadowing function in microfacet-based BRDFs. *Journal of Computer Graphics Techniques (JCGT) 3*, 2, 48–107.

IGEHY, H. 1999. Tracing ray differentials. SIGGRAPH 1999.

JAKOB, W., REGG, C., AND JAROSZ, W. 2011. Progressive expectation–maximization for hierarchical volumetric photon mapping. *Computer Graphics Forum (Proceedings of EGSR 2011) 30*, 4.

JAKOB, W., HAŠAN, M., YAN, L.-Q., LAWRENCE, J., RAMAMOORTHI, R., AND MARSCHNER, S. 2014. Discrete stochastic microfacet models. *ACM Trans. Graph. 33*, 4.

JAKOB, W., 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.

OLANO, M., AND BAKER, D. 2010. Lean mapping. ACM, I3D 2010, 181–188.

TOKSVIG, M. 2005. Mipmapping normal maps. *Journal of Graphics Tools 10*, 3, 65–71.

TSAI, Y.-T., AND SHIH, Z.-C. 2006. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Trans. Graph. 25*, 3.

VEACH, E. 1997. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University.

VERBEEK, J., NUNNINK, J., AND VLASSIS, N. 2006. Accelerated EM-based clustering of large data sets. *Data Mining and Knowledge Discovery 13*, 3, 291–307.

XU, K., SUN, W.-L., DONG, Z., ZHAO, D.-Y., WU, R.-D., AND HU, S.-M. 2013. Anisotropic spherical gaussians. *ACM Trans. Graph. 32*, 6.

YAN, L.-Q., HAŠAN, M., JAKOB, W., LAWRENCE, J., MARSCHNER, S., AND RAMAMOORTHI, R. 2014. Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Trans. Graph. 33*, 4.

ZIRR, T., AND KAPLANYAN, A. S. 2016. Real-time rendering of procedural multiscale materials. ACM, I3D 2016, 139–148.

Appendix

To compute Equation 11, we need to find the integral I:

$$I = \int_{\mathbb{R}^2} G_p(\mathbf{u}) G_i(\mathbf{u}, \mathbf{s}) \, \mathrm{d}\mathbf{u}.$$

The following three steps are required: 1) Turn the 4D Gaussian $G_i(\mathbf{u}, \mathbf{s})$ into a 2D Gaussian by fixing $\mathbf{s}$. 2) Analytically compute the product of the resulting 2D Gaussians. 3) Analytically integrate the final 2D Gaussian. Below we describe these three steps in more detail.

2D slice of a 4D Gaussian. Without loss of generality, we define this problem as rewriting $g = G(\mathbf{x}; c, \mathbf{0}, \mathbf{\Sigma})$ into $g = G(\mathbf{u}; c', \mathbf{u}_0, \Sigma')$, where $\mathbf{x} = (\mathbf{u}, \mathbf{s})$ is a 4D column vector, and $\mathbf{s}$ is a 2D column vector that is fixed. Note that, here we explicitly write a multivariate Gaussian $G(\mathbf{x}; c, \mu, \Sigma)$ with its scaling coefficient c, center μ and covariance matrix Σ. (Note: in practice it is usually better to store inverse covariance matrices.)

Since $\mathbf{\Sigma}$ is symmetric, $\mathbf{\Sigma}^{-1}$ is also symmetric. We first represent $\mathbf{\Sigma}^{-1}$ using 2×2 blocks

$$\mathbf{\Sigma}^{-1} = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}, \tag{15}$$

Then the 4D Gaussian can be written as

$$g = c \cdot \exp\left(-\frac{1}{2} \left(\mathbf{u}^T A \mathbf{u} + 2\mathbf{s}^T B^T \mathbf{u} + \mathbf{s}^T C \mathbf{s} \right) \right). \tag{16}$$

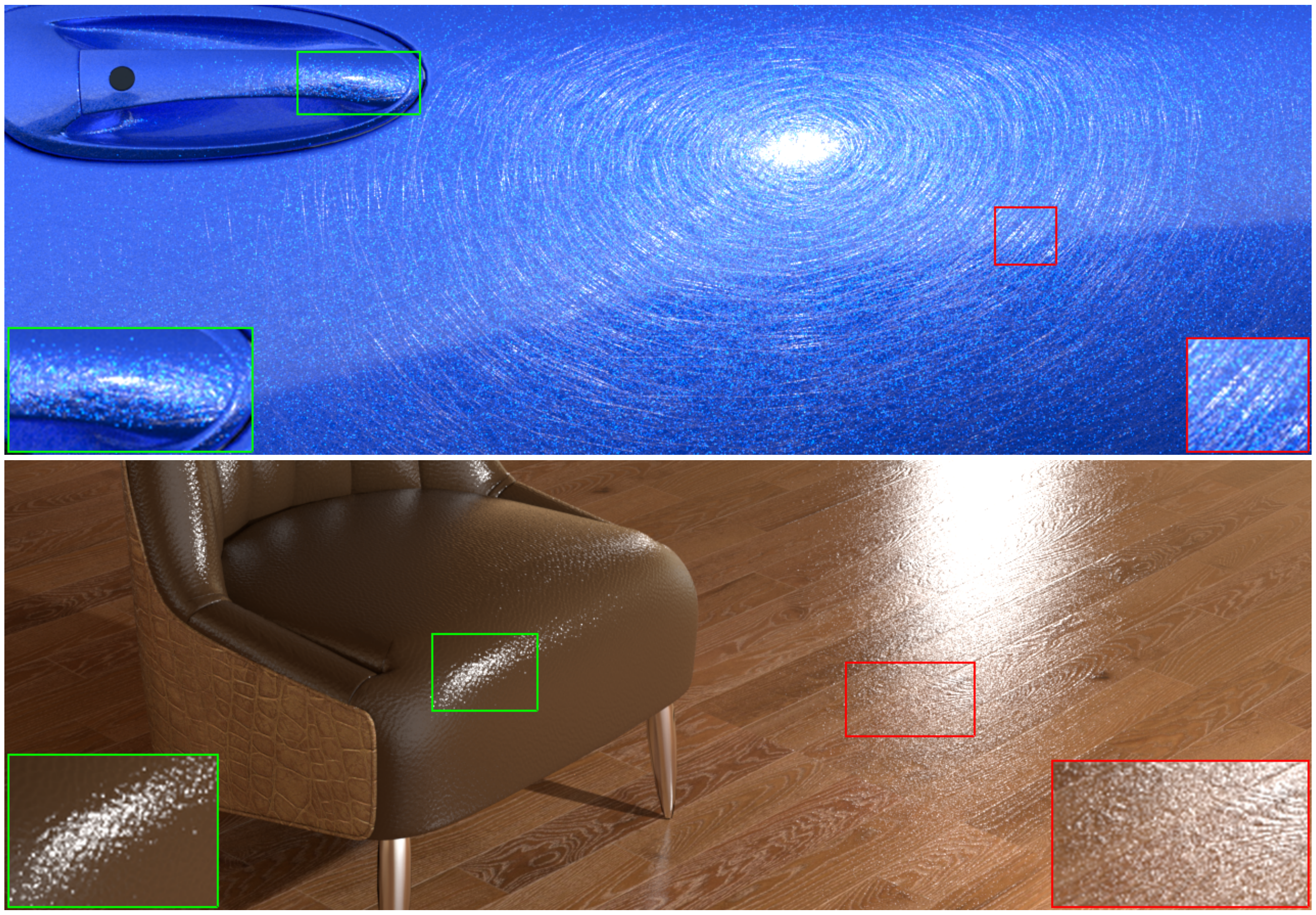


Figure 8: *Final image renderings using our method. Insets are zooms of image itself (not re-rendered at high resolution).* **Top:** *car paint (scratched coating with embedded metallic flakes). Both effects use our method; the scratches use curved elements and the metallic paint uses flat elements.* **Bottom:** *leather sofa on a wooden floor; both materials use a combination of a macro-level standard normal map, and a micro-level map handled using our technique.*

Our goal is to write g as a 2D Gaussian in $\mathbf{u}$:

$$g = c' \cdot \exp\left(-\frac{1}{2}(\mathbf{u}-\mathbf{u}_0)^T \Sigma'^{-1}(\mathbf{u}-\mathbf{u}_0)\right). \tag{17}$$

Expanding Equation 17 and comparing terms inside the exponential with Equation 16, we immediately have

$$\Sigma'^{-1} = A,\ \mathbf{u}_0 = -A^{-1}B\mathbf{s},\ c' = c\cdot\exp\left(-\frac{1}{2}(\mathbf{s}^T C\mathbf{s} - \mathbf{u}_0^T A\mathbf{u}_0)\right). \tag{18}$$

Product of two multivariate Gaussians. Given two multivariate Gaussians $G(\mathbf{x}; c_1, \mu_1, \Sigma_1)$ and $G(\mathbf{x}; c_2, \mu_2, \Sigma_2)$, their product is another multivariate Gaussian $G(\mathbf{x}; c, \mu, \Sigma)$, where

$$\Sigma^{-1} = \Sigma_1^{-1} + \Sigma_2^{-1}, \qquad \mu = \Sigma(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2). \tag{19}$$

To find the scaling coefficient c, we just evaluate the original product at the new mean:

$$c = G(\mu; c_1, \mu_1, \Sigma_1) \cdot G(\mu; c_2, \mu_2, \Sigma_2). \tag{20}$$

Integral of a multivariate Gaussian. Integrating a multivariate Gaussian over $\mathbb{R}^n$ results in

$$\int_{\mathbb{R}^n} G(\mathbf{x}; c, \mu, \Sigma)\,\mathrm{d}\mathbf{x} = c \cdot (2\pi)^{n/2}|\Sigma|^{1/2}. \tag{21}$$