DOI: 10.1111/cgf.13547
Pacific Graphics 2018
H. Fu, A. Ghosh, and J. Kopf
(Guest Editors)

Volume 37 (2018), Number 7

Fast Global Illumination with Discrete Stochastic Microfacets Using a Filterable Model

Beibei Wang[1,2], Lu Wang[3], Nicolas Holzschuch[4]

[1]School of Computer Science and Engineering, Nanjing University of Science and Technology
[2]Key Lab of Intelligent Perception and Systems for High-Dimensional Information of Ministry of Education
[3]School of Software, Shandong University
[4]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK

(a) *Our model (with filtering)* Total: 6.09 h, Cost: +1.5%

(b) *Our model (no filter),* Total: 6.35 h, Cost: +6%

(c) *Original [JHY*14],* Total: 8.96 h, Cost: +49%

(d) *Without Glints,* Total: 6.0 h

Figure 1: *Our algorithm, compared to the original Discrete Stochastic Microfacets model [JHY*14] (c). Converting the 4D search to a product of 2D searches (b) produces almost identical results. This is the basis for our filterable model (a), which allows fast global illumination with negligible cost.*

Abstract

Many real-life materials have a sparkling appearance, whether by design or by nature. Examples include metallic paints, sparkling varnish but also snow. These sparkles correspond to small, isolated, shiny particles reflecting light in a specific direction, on the surface or embedded inside the material. The particles responsible for these sparkles are usually small and discontinuous. These characteristics make it difficult to integrate them efficiently in a standard rendering pipeline, especially for indirect illumination. Existing approaches use a 4-dimensional hierarchy, searching for light-reflecting particles simultaneously in space and direction. The approach is accurate, but still expensive. In this paper, we show that this 4-dimensional search can be approximated using separate 2-dimensional steps. This approximation allows fast integration of glint contributions for large footprints, reducing the extra cost associated with glints be an order of magnitude.

1. Introduction

Many materials have a sparkling or glittering appearance. Examples include some car paints, glittery lipsticks but also natural materials such as snow or sand. These sparkles are an essential part of the material appearance. Simulating them is essential for photo-realistic rendering, but also difficult. These sparkles are caused by small patches, with random distribution. These patches are not necessarily part of a continuous surface. Each individual patch provides a bright reflection for a small set of directions, and is not visible otherwise.

As the patches are very small, sampling them accurately is time consuming, and they risk causing noise or aliasing by being under-sampled. We need the glints to have a random distribution, but we also require temporal coherence in animations.

Jakob et al. [JHY*14] introduced a discrete stochastic microfacet model for glints. They represent glints as small specular surfaces, as in the microfacet model [CT82] [WMLT07], but with a discrete

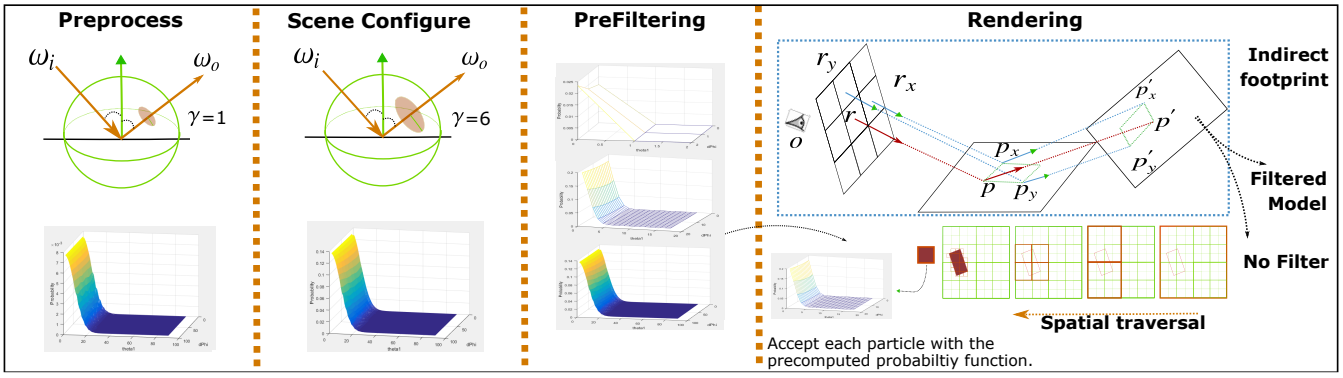


Figure 2: *Our algorithm: In a **preprocessing** step, we compute and store the probability for a particle to be reflected around the direction ω_o with a tolerance of 1 degree. This probability is indexed by the incoming and outgoing directions ω_i, ω_o. For each scene, given the scene-specific search range γ, we accumulate table values in the search range. We then prefilter the new table using Gaussian blurring to build a hierarchical representation. During **rendering** (right), we trace the path footprint for each surface interaction, and use it to compute the contribution. Depending on the footprint size, we either compute individual glint contributions or the average contribution from all glints.*

distribution instead of a continuous surface. These specular patches are organized in a hierarchy. They extend the microfacet model from a point-direction model to an area-solid angle model and a 4-dimensional search in that hierarchy, to find the number of particles that are located in the query domain, both in space and angle. These particles are used to compute the distribution function and the reflectance.

Their algorithm is purely procedural, without the need to store or compute any textures. It is also physically-based, and temporally coherent. Due to the 4-dimension traversal, computing glints is more expensive than computing standard reflectance model. The extra cost is acceptable for direct illumination, but becomes prohibitive for global illumination.

Our contributions in this paper are two fold: first, we show that the 4-dimension hierarchy traversal can be approximated with two separate 2-dimension traversals, one in the spatial domain and one in the angular domain. This decoupling allows several optimizations: we locate the spatial position directly, eliminating the need for a top-down traversal and we precompute the angular probability of glints depending on direction. Second, we use this simplified model for automatic filtering in global illumination: we compute the footprint of secondary rays and integrate the glints contribution directly. The global model results in very fast global illumination, 30 times faster than the original algorithm on complex scenes.

We review previous work on rendering glints and sparkles in the next section. We then present the original discrete stochastic microfacet model in Section 3. We describe our own algorithm in Section 4. In Section 5, we compare our method with previous works and reference solutions. We conclude in Section 6.

2. Previous Work

2.1. Glint Rendering

Jakob et al. [JHY*14] proposed a stochastic model based on microfacet theory to simulate glittery surfaces without any textures. Atanasov et al. [AK16] improve this work for faster computations and heavy-tailed microfacet distributions.

Glints and scratches can also be expressed using normal maps. Yan et al. [YHJ*14] compute accurate BRDF values for a normal map over arbitrary regions of the surface by using a hierarchical search to locate normals that are close to the half vector. Because it makes minimal assumptions about a pixel's NDF, this approach can also be used to compute glints. Yan et al. [YHMR16] improve this algorithm by representing a surface as a position normal distribution and approximating this 4D distribution as a mixture of Gaussian elements, along with overall performance improvements. Jakob et al. [JHY*14] does not require additional storage, but it limited to point-shaped glints. Yan et al. [YHJ*14, YHMR16] produce arbitrary shaped surface details, including scratches and glints, at the expense of extra storage.

Car paints often use flakes for appearance. Gunther et al. [GCG*05] use procedural normal maps to represent glitter. Rump et al. [RMS*08] use measured textures for photo-realistic rendering.

Our algorithm extends over Jakob et al. [JHY*14]. We begin by a short description of this work in Section 3. Our goal is to produce glittery effects quickly, with a lesser focus on physical accuracy.

2.2. Real-time Glint Rendering

Shopf [Sho12] used a 3D jittered grid of sparkle shapes to render sparkles in the snow surface in real-time. Bowles and Wang [BW15, WB16] introduced a procedural sparkle approach for snow; this technique has been used in production. Zirr et al. [ZK16] derived a stochastic biscale microfacet model to fit for real-time application. All these algorithms are faster than ours, but are not physically based.

2.3. Anti-aliasing

Aliasing is a fundamental problem of any rendering algorithm. By nature, sparkles are extremely susceptible to aliasing, including temporal aliasing. The problem is similar to point based rendering: Rusinkiewicz et al. [RL00] draw elliptical splats rotated and foreshortened depending on per-node normals, which reduces thickening and noise around silhouette edges. Zwicker et al. [ZPvBG01, ZPvBG02] used elliptical Gaussian kernels for high quality splatting to avoid aliasing artifacts.

Belcour et al [BYRN17] introduced a global anti-aliasing approach, propagating pixel footprints along each path, combined with prefiltering.

3. Background: Discrete Stochastic Microfacet Model

3.1. Description

Jakob et al. [JHY*14] introduced the discrete stochastic microfacet model for glittery surface. Their algorithm extends the microfacet BRDF model to take into account a finite extent in space and angle, instead of diracs:

$$\hat{f}_r(A, \omega_i, \Omega_o) := \frac{1}{a(A)\sigma(\Omega_o)} \int_A \int_{\Omega_o} f_r(\boldsymbol{x}, \omega_i, \omega_o)\, \mathrm{d}\omega_o\, \mathrm{d}\boldsymbol{x}. \quad (1)$$

Where A is a finite area around point $\boldsymbol{x}$, Ω_o a finite solid angle around outgoing direction ω_o. f_r is the usual microfacet BRDF [CT82] [WMLT07]:

$$f_r(\boldsymbol{x}, \omega_i, \omega_o) = \frac{F(\omega_i \cdot \omega_h) D(\boldsymbol{x}, \omega_h) G(\omega_i, \omega_o, \omega_h)}{4(\omega_i \cdot \boldsymbol{n}_x)(\omega_0 \cdot \boldsymbol{n}_x)}. \quad (2)$$

Where F represents the Fresnel reflection coefficient, D is the microfacet normal distribution, and G is the shadowing and masking term.

If we assume the surface to be made of discrete small mirrors or particles instead of continuous microfacets, we get the new flakes BRDF model:

$$\hat{f}_r(A, \omega_i, \Omega_o) = \frac{(\omega_i \cdot \omega_h) F(\omega_i \cdot \omega_h) \widehat{D}(A, \Omega_h) G(\omega_i, \omega_o, \omega_h)}{a(A)\sigma(\Omega_o)(\omega_i \cdot \boldsymbol{n}_x)(\omega_0 \cdot \boldsymbol{n}_x))}, \quad (3)$$

where $\widehat{D}$ is a sum over the finite set of particles:

$$\widehat{D}(A, \Omega_h) := \frac{1}{N} \sum_{k=1}^{N} \mathbf{1}_{\Omega_h}(\omega_h^k) \mathbf{1}_A(\boldsymbol{x}^k). \quad (4)$$

For each particle k, the element in the sum is 1, if its position $\boldsymbol{x}^k$ and half-vector ω_h^k fall within the search domain in space and angle $A \times \Omega_h$. $\widehat{D}$ is the number of particles that fall within this search domain. N is the particle count.

A usually corresponds to a pixel footprint, and Ω_h to a cone of directions around which particles can reflect light. The cone half-angle γ is a parameter of the model. In practical applications, it ranges from 1° to 6°.

3.2. Use in Path Tracing

To use the model in path-tracing, for each bounce, we integrate over outgoing directions:

$$L_o(A, \omega_i, \omega_o) = \int_\Omega L_i(\omega_i, \omega_o') \hat{f}_r(A, \omega_i, \omega_o')\, \mathrm{d}\omega_o' \quad (5)$$

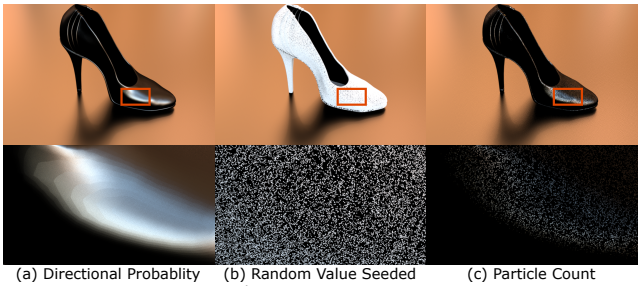


Figure 3: *We combine the Directional Probability Function (a) with random values computed using TEA (b) to get the particle count (c).*

To compute this integral in practice, the flakes BRDF model $\hat{f}_r$ is sampled to obtain the next outgoing direction ω_o'. The spread of outgoing directions depends on material roughness: for rough materials, the discrete stochastic model has a large variance. A large number of samples are then required for convergence, even only considering direct illumination.

4. Filterable Discrete Stochastic Microfacet Models

4.1. Our Separable Model

The main element in the Discrete Stochastic Microfacet Models is the 4-dimensional multiscale microfacet normal distribution $\hat{D}$ (see Equation 4). It corresponds to the count of particles located in the 4-dimensional finite domain $A \times \Omega_h$. To compute this number of particles, the original algorithm uses a 4-dimensional hierarchy traversal. Our first idea is to replace the particle count with a particle *probability function*, which can be carried out of the integral, allowing the decoupling between space and angle.

Under the assumption of separability, $\hat{D}$ can be considered as a joint distribution of two independent variables of $\boldsymbol{x}$ and (ω_i, ω_o). We introduce the *directional probability function* (DPF), $P(\omega_i, \omega_o, \gamma)$, expressing the probability that a particle exists that reflects light incoming from direction ω_i into a cone centered around direction ω_o with half-angle γ.

We rewrite Equation 4 using this directional probability function:

$$\hat{D}(A, \omega_i, \omega_o) = \left(\frac{1}{N} \sum_{k=1}^{N} \mathbf{1}_A(\boldsymbol{x}^k)\right) P(\omega_i, \omega_o, \gamma) \quad (6)$$

The directional probability function $P(\omega_i, \omega_o, \gamma)$ is expressed as

$$P(\omega_i, \omega_o, \gamma) \approx \frac{1}{N} \sum_{k=1}^{N} \mathbf{1}_{\Omega_h}(\omega_h^k) \quad (7)$$

The directional probability function P represents a *continuous* approximation of the flakes reflection. To reproduce the flakes *discrete* behaviour, we threshold it against a random variable of position (See Figure 3), $\lambda(\boldsymbol{x})$:

$$\hat{D}(A, \omega_i, \omega_o) = \frac{1}{N} \sum_{k=1}^{N} \mathbf{1}_A(\boldsymbol{x}^k) H\left(\lambda(\boldsymbol{x}) - P(\omega_i, \omega_o, \gamma)\right) \quad (8)$$

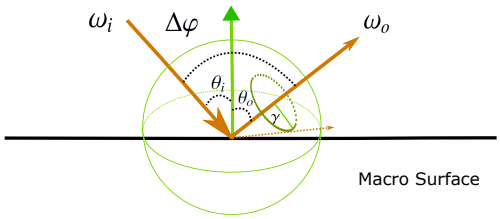


Figure 4: *We parameterize the directional probability function with $\theta_i, \theta_o, \Delta\varphi$ instead of $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}_o$. θ_i and θ_o denote the elevation angle for the incoming and outgoing direction, and $\Delta\varphi$ is the absolute difference of the azimuthal angle.*

$$H(u) = \begin{cases} 1, & \text{if } u > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

For this random variable of position, we use the Tiny Encryption Algorithm (TEA) [WN95], as in [JHY*14]. It provides a uniformly distributed random value; we seed it with a variable based on position, using the node ID.

4.2. Precomputing the Directional Probability Function

Our Directional Probability Function P (Equation 7) does not depend on the spatial position of the queries, only on the incoming and outgoing directions and the searching angle. We reparameterize it as a function of four parameters: $P(\theta_i, \theta_o, \Delta\varphi, \gamma)$: θ_i and θ_o are the elevation angles for incoming and outgoing directions, $\Delta\varphi = \phi_i - \phi_o$ is the difference in azimuthal angle (see Figure 4).

First, we compute and store P for $\gamma = 1^o$, by sampling regularly in each dimension, computing and storing the value for P. This first step is very fast, less than 1 s.

The search angle γ is a scene parameter. At runtime, we compute P for the given γ by accumulating samples from the 1^o table for each $\boldsymbol{\omega}_o$.

This precomputation step only depends on material properties; it takes 1 to 2 seconds, and requires 5.7 MB of storage (see Table 2).

4.3. Prefiltering the Directional Probability Function

We then build a hierarchical representation of our Directional Probability Function, using Gaussian blur. Each level is generated by blurring the finest level. In our implementation, we generate 9 hierarchical levels, for 1^o, 2^o, 5^o, 10^o, 20^o, 30^o, 45^o, 60^o and 90^o.

During rendering, we select the appropriate hierarchical level depending on material roughness and incoming light's frequency.

4.4. Two Step Hierarchy Traversal

Replacing the particle count with a particle probability function in Equation 6 allows us to replace the 4-dimensional hierarchy traversal with two separate hierarchy traversals: one for space and one for angle. Our goal is to find the number of particles inside the footprint for a given pixel.

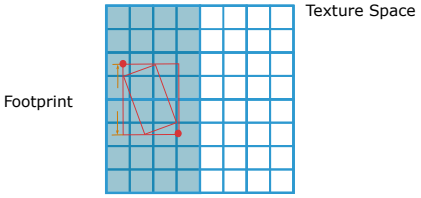


Figure 5: *We use the axis-aligned bounding box (in texture space) of the pixel footprint to find the starting node in the spatial hierarchy.*

Algorithm 1 Spatial Traversal

```
function D̂_s(A, ω_i, ω_o)
    query ← A
    queue ← node(N_start)
    count ← 0
    p ← P(ω_i, ω_o)
    while queue ≠ ∅ do
        node ← queue.pop()
        if node ∩ query == ∅ or |node| = 0 then pass
        else if node ⊆ query then n_k ← |node|
            for i < n_k do ψ ← λ(k)
                if ψ > p then count ← count + 1
                end if
            end for
        else if error criterion satisfied then
            overlap ← (node ∩ query).vol()/node.vol()
            n_k ← |node|
            for i < n_k do ψ ← λ(k)
                if ψ > p then count ← count + overlap
                end if
            end for
        else
            for c in node.split() do queue.push(c)
            end for
        end if
    end while
    return count
end function
```

- The traversal in angle corresponds to a look-up in our precomputed tables: we first select the appropriate hierarchical level, depending on material roughness and incoming light's frequency. We then extract the precomputed value for P at this hierarchical level.
- For the traversal in space, we conduct the traversal in texture space, as in [JHY*14] and assume flakes are distributed uniformly in this space. Having separated the spatial and angular traversals allows us to speed up the traversal (see Figure 5): we begin with the axis-aligned bounding box of the pixel footprint in texture space, and use it to find the starting point of the traversal. For each node encountered in the traversal, we compute its intersection with the pixel footprint, and use it to estimate the number of flakes generated. For each of these flakes we generate a random value, seeded with node ID. If this value is larger

than the probability evaluated during the search in the directional domain, then the particle will be reflected (see Algorithm 1).

The area corresponding to a pixel footprint is much smaller than the texture space of the object. Jumping straight to the hierarchical level corresponding to its bounding box greatly speeds up the spatial hierarchy traversal.

4.5. Filterable Glint Computation

The model described in the previous sections is an approximation of the stochastic microfacet model. By itself, it provides a significant reduction in computation time. But its strongest advantage is that it allows *filterable* glint computation: to compute the *average* contribution from glints in a certain area A, we only have to multiply the Directional Probability Function P by the surface area:

$$\hat{D}(A, \omega_i, \omega_o) = a(A) \times P(\omega_i, \omega_o, \gamma), \tag{10}$$

where $a(A)$ is the area of A. The estimated corresponding glint count is $N \times \hat{D}$.

This allows us to include glints in global illumination simulation efficiently:

- For each path, compute the path footprint at each light bounce (see Section 4.6).
- Compute the average glint contribution at this footprint using Equation 10.
- If the glint count is larger than a given threshold, use the average contribution.
- Otherwise, use the separable model described in Section 4.4.

The idea behind the algorithm is that, if a footprint covers a large surface area, individual glints are not noticeable, and the average contribution is what matters. In our experiments, 16 is a good threshold to switch from the accurate model to the filtered model.

In practice, this makes us switch to the filtered model when the glint material is far from the camera, or after several bounces in global illumination.

4.6. Approximate Footprint Computation

For each pixel, for the first intersection with geometry, we use ray differentials, as described by Igehy [Ige99].

For further interactions along the path, we keep tracing the main ray (the one centered at $\boldsymbol{p}$) as well as the offset rays, with origin $\boldsymbol{p}_x$ (resp. $\boldsymbol{p}_y$) and direction $\boldsymbol{r}_x$ (resp. $\boldsymbol{r}_y$). For specular bounces, the outgoing direction of the offset ray $\boldsymbol{r}'_x$ is the specular bounce of the incoming direction $\boldsymbol{r}_x$. For glossy reflections, we first compute the specular bounce, the extend the direction using a glossy cone angle, depending on the BSDF (see Figure 6). We used the glossy cone angle approximation defined by Günther et al. [GRG14]. For the Phong BSDF model, the angle for extension β at each bounce is approximated as:

$$\beta \approx 2\sqrt{\frac{2}{g+2}},$$

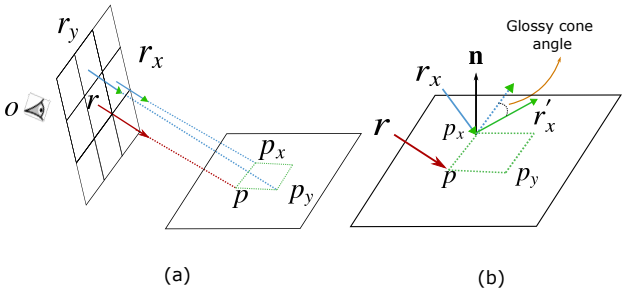


Figure 6: *We compute the UV partial derivatives for the first intersection (p) by computing p_x and p_y and solving an equation with $\frac{\partial p}{\partial u}$ and $\frac{\partial p}{\partial v}$. For the following glossy bounces, we use an approximated way to estimate the outgoing direction r'_x of the offset rays. r'_x is approximated by assuming a specular surface firstly to get a reflected outgoing direction (dashed blue line in (b)) and then increasing a glossy cone angle for the apex angle of this direction with respective to the surface normal ($\boldsymbol{n}$) and finally get the reflected offset ray (green line).*

where g is the phong exponent. For the Cook-Torrance model with the Beckmann normal distribution function, we use:

$$\beta \approx 2\alpha$$

where α is the roughness of the material.

We then convert the spatial footprint into screen space derivatives.

5. Results and Discussion

We have implemented our algorithm inside the Mitsuba Renderer [Jak10]. We compared our algorithm against [JHY*14] which we consider as the reference for quality validation. We use path tracing as our global illumination algorithm.

All timings in this section are measured on a 2.20GHz Intel i7 (40 cores) with 32 GB of main memory. Unless otherwise stated, we use the Beckmann microfacet model as the underlying smooth distribution D, which has a single roughness parameter with lower values corresponding to smoother surfaces.

Figures 1, 8, 9 and 10 show our algorithm (using filtering) with global illumination; Figure 7 shows direct illumination only.

5.1. Qualitative Validation

We first compare our method with the reference solution, Stochastic Microfacet Distribution, by Jakob et al. [JHY*14]. Figure 7 focuses on direct illumination. Qualitatively, our approach produces results that are visually identical to the reference. In terms of computation time, we have reduced the extra cost introduced by glints by a factor of 5.

In Figure 8, we do the same comparison, but with indirect illumination and reflection on a glossy floor. Visually, the results are,

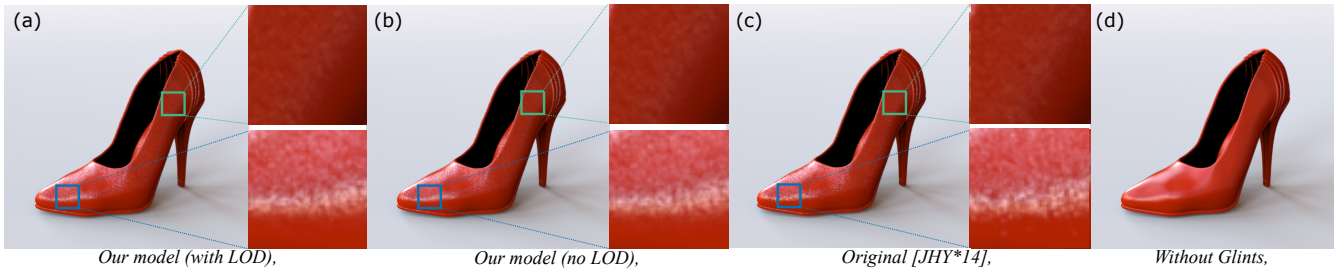


Figure 7: *Comparison with reference [JHY*14] on the Shoe Scene with direct illumination only.*

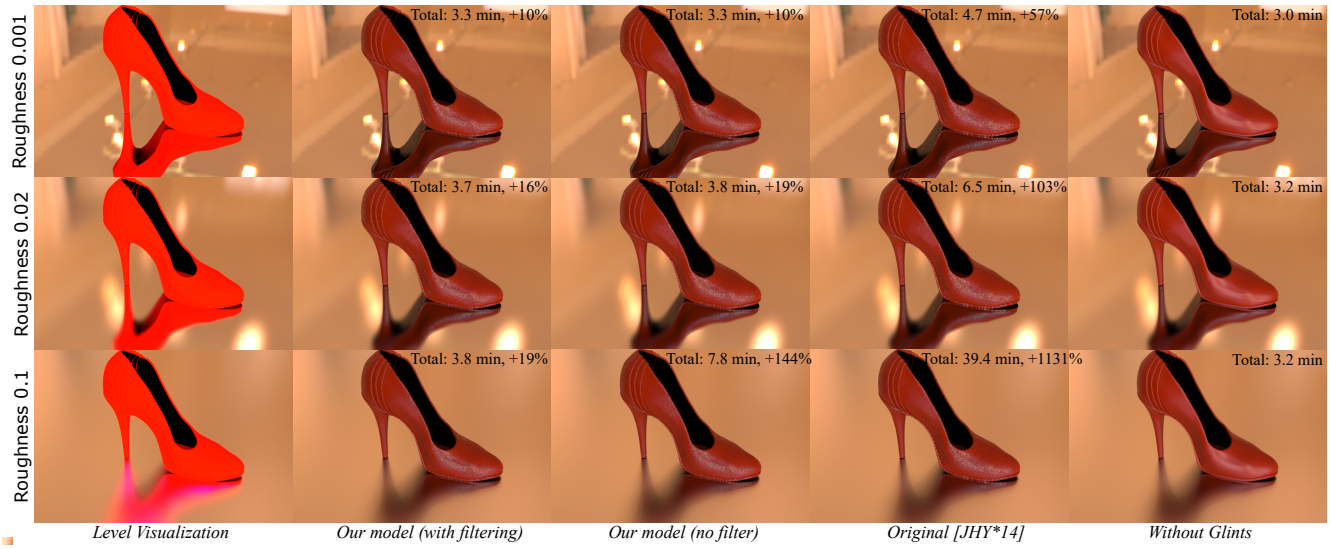


Figure 8: *Comparison with reference [JHY*14] with Shoes on a glossy floor Scene. The first column displays whether we use the accurate model (red) or the filtered version (blue). For the first two rows, the floor is highly specular, so we use the finest version everywhere; the computation cost is the same with and without filtering. For the third row, because of the rough floor, we switch to the filtered version more often, resulting in a significant cost reduction.*

again, almost identical to the reference. The computational advantage depends on the floor roughness: for a specular or almost specular floor (top two rows), we do not use the filtered version of the algorithm, because the glint count at each interaction remains below the threshold. Computation time is almost identical for the filtered and unfiltered version, while still 5 times faster than the reference.

For a glossy floor (roughness $\alpha = 0.1$, bottom row), the computation time increases significantly because of the larger footprint caused by the glossy reflection. Our method without filtering is about 8 times faster than the reference, but still quite expensive compared to the scene without glints. The filtered version is triggered for distant reflections, and results in a significant speedup.

One key advantage of our filtered algorithm is its *predictability*: the extra cost caused by the introduction of glints remains almost constant, between 1.3 and 1.6 mn, independent of the rest of the scene. For comparison, increasing glossiness causes the extra computation time for the original algorithm to raise from 1.7 mn to 36.2 mn.

The same behaviour is apparent in Figures 9 and 10: Our model without filtering provides a significant speedup (8 times faster for the Balls Scene, 6 times faster for the Car Scene). With filtering, the speedup is much larger (more than 25 times faster). With filtering activated, adding glints to the scene amounts to a small extra computational cost, from 15 % to 20 %.

Figure 1 corresponds to a complex scene, illuminated by an environment map and several point lights. Glints are evaluated during the path tracing step at each bounce. Our separable model, without filtering, obtains a 8× speedup for the glint evaluation part. Using filtering further increases the acceleration: the extra cost associated with glints is 33 times smaller than with the original method, to the point where it is almost negligible compared with the total computation time (1.5 % of total time).

Our algorithm is good at preserving temporal coherence; please see the companion video.

5.2. Performance and Timings

Table 1 displays the settings for the materials and timings for all our test scenes. For all test scenes, we report the computation time

Name	Depth	Res.	α	γ	N	#Sample	LOD	Glint Eval. Time (min)			Total Time (min)		
							(Deg.)	[JHY*14]	Ours	Speedup	Without	[JHY*14]	Ours
Shoe	2	768 x 512	0.1	6	10^6	2048	2	1.0	0.2	5.0x	1.7	2.7	1.9
Balls	10	768 x 431	0.1	6	10^6	2048	2	17.8	0.6	29.6x	4.6	22.4	5.2
Car	5	768 x 512	0.1	6	3×10^7	2048	2	20.6	0.8	25.8x	3.8	24.4	4.6
Kitchen	∞	1920 x 1080	0.1	6	10^6	16384	mixed	177.6	5.4	32.9x	360.0	537.6	365.4

Table 1: *Parameters and costs for the scenes used in this paper. Depth is the maximum tracing depth in path tracing. Res. is the resolution of the rendered image. α is the roughness of the materials. γ is the search angle for the directional domain. N is the total particle count. #Sample is the count of samples used during rendering. LOD represents the blurring angle.*

Our model (with filtering) Total: 5.2 min, Cost: +13%

Our model (no filter) Total: 6.9 min, Cost: +50%

*Original [JHY*14],* Total: 22.4 min, Cost: +387%

Without Glints, Total: 4.6 min

Figure 9: *Comparison with reference [JHY*14] on the Balls Scene with global illumination.*

α	Sample	Mem. (MB)	Time (Sec.)	LOD (Deg.)
0.8	$90 \times 90 \times 180$	5.7	1.5	30
0.5	$90 \times 90 \times 180$	5.7	1	10
0.1	$90 \times 90 \times 180$	5.7	0.5	2
0.05	$90 \times 90 \times 180$	5.7	0.5	1

Table 2: *Parameters and costs for precomputed table in the directional domain. We sample $\theta_i, \theta_o, \Delta\varphi$ per degree.*

without glints, and the computation time for the reference method by Jakob et al. [JHY*14] and our method with filtering. To compare methods, we report both the total computation time, and the extra time associated with glint computation.

The overall speedup depends on the complexity of the computations, and the impact of filtering. For direct illumination only, or specular reflections, filtering is not used, and the separable glint model provides a 5 times speedup. For more complex scenes, with indirect illumination and glossy reflections, the speedup is more important, about 30 times faster.

The precomputed directional probability function is extremely fast, and introduces a modest extra memory cost. Table 2 shows computation times and memory costs as a function of material roughness.

5.3. Parameter Analysis

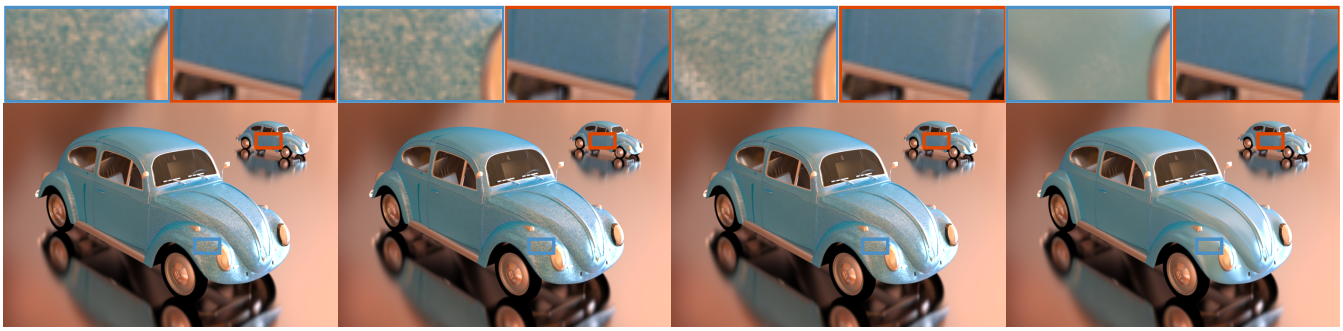
Figures 11 and 12 show the impact of glint parameters on our algorithm:

- Figure 11 displays computation time as a function of the search angle γ for the Car Scene (Figure 10). Computation time for our algorithm is a constant, independent from this parameter. Computation time for [JHY*14], on the other hand, increases with γ , as they are required to search for particles in a larger domain. As a consequence, the larger the search angle, the more important the speedup for our method.
- Figure 12 displays computation time as a function of the particle count N for the Car Scene (Figure 10). Increasing the number of particles increases the computation time for both [JHY*14] and our method, but the slope for our method is much smaller, resulting in a smaller increase in computation time.

Figures 13 and 14 show the impact of global illumination parameters on our algorithm:

- Figure 13 displays the impact of varying material roughness of the floor material in the Shoe Scene (see results in Figure 8). Increasing roughness increases computation time for all methods, due to the larger footprint for integration. Using our model with filtering keeps this increase manageable, thanks to our fast approximation for large footprints.
- Figure 14 displays computation time as a function of path tracing depth for the Balls Scene (Figure 9). A low value (2) corresponds to only direct illumination. In this setting, the cost of the reference algorithm [JHY*14] is not too high, though substantially more expensive than our method. As the depth increases, we compute more light bounces, increasing the computation time. The slope is much larger with the reference method than with our method using filtering. The computation time levels after 4 bounces, probably due to the scene configuration.

An important parameter for our method is the threshold for



Our model (with filtering) Total: 4.6 min, Cost: +21%

Our model (no filter) Total: 7.4 min, Cost: +95%

*Original [JHY*14],* Total: 24.4 min, Cost: +542%

Without Glints, Total: 3.8 min

Figure 10: *Comparison with reference [JHY*14] on the Car Scene with global illumination.*

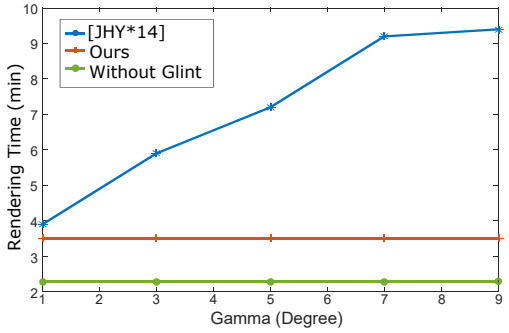


Figure 11: *Computation time for reference [JHY*14], our method and rendering without glints, as a function of the searching angle γ, for the Car Scene, particle count 3×10^7 and roughness $\alpha = 0.1$ (direct illumination only).*

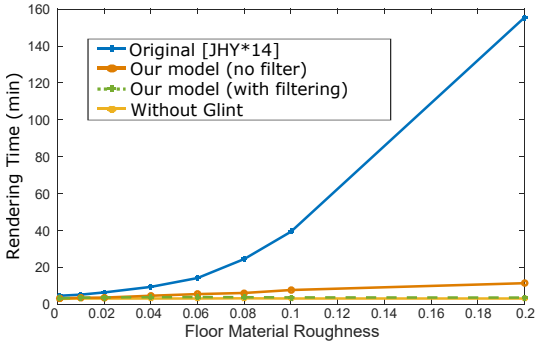


Figure 13: *Comparison between our algorithm (with and without filtering), reference [JHY*14] and without glint with varying roughness for Shoe Scene (global illumination).*

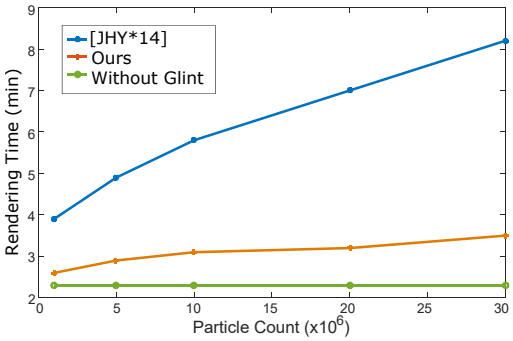


Figure 12: *Computation time for reference [JHY*14], our method and rendering without glints, as a function of the number of particles N, for the Car Scene with $\gamma = 6$ and roughness $\alpha = 0.1$ (direct illumination only).*

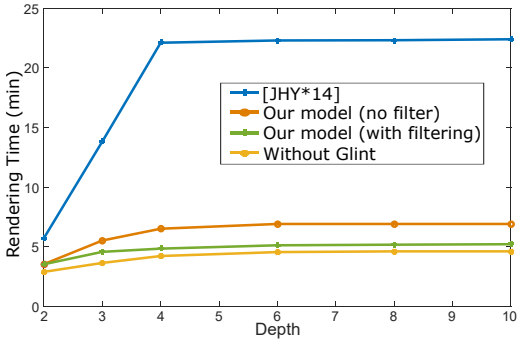


Figure 14: *Comparison between our algorithm (with and without filtering), reference [JHY*14] and computation without glint with varying path tracing depth for Balls Scene (global illumination).*

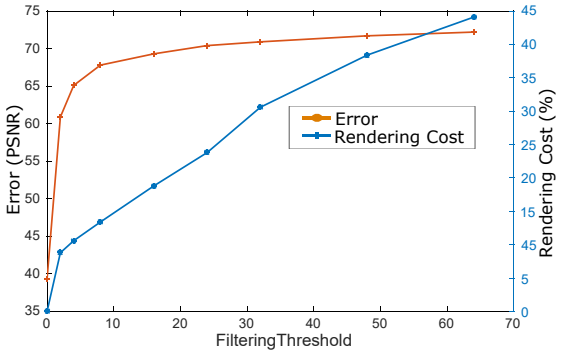
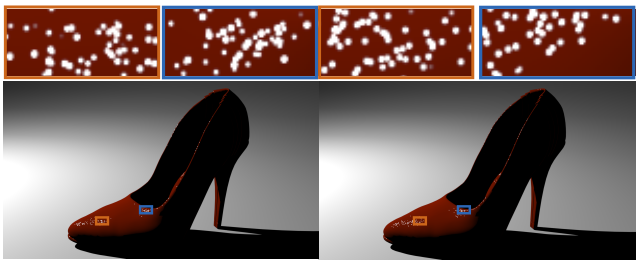


Figure 15: *Error (PSNR, larger value means smaller error) and rendering cost (as a percentage) as a function of filtering threshold for the Shoe Scene (global illumination).*



Our model, Cost: +9%

*Original [JHY*14],* Cost: +64%

Figure 16: *Comparison between our algorithm and [JHY*14] for scenes with sparse glints.*

switching from individual glint computation to filtered glint representation. We express it as a maximum number of glints in the footprint. Figure 15 displays the algorithm behaviour as we change this parameter, using the Shoe Scene (Figure 8), with a roughness of $\alpha = 0.1$. We compute both the error, evaluated by comparing our algorithm with filtering and without filtering (left scale), and the rendering cost, measured as the extra computation time compared to the same scene without glints (right scale). Increasing the filter threshold decreases the error, at the expense of rendering time. 16 glints as a threshold for switching to the filtered version appears to be a reasonable compromise.

5.4. Discussion and limitation

Our algorithm relies on several approximations and assumes a large number of flakes. The approximations fail for a smaller number of flakes, as in Figure 16. Our method is still much faster, but the glints do not have the accurate distribution provided by the reference method. Our method cannot simulate flakes that have a spatio-angular correlated distribution.

Also, glitter has a distinctive angular behaviour. If individual glints are visible, as the half vector moves through the hemisphere, the pixel value flickers brighter and darker. Our method does not keep this behaviour, as the DPF is a monotonically increasing function when moving towards to the normal direction. The thresholding function cannot provide this brighter and darker appearance. However, the effect is only visible with point lights, and becomes invisible with environment lighting or indirect lighting.

For applications where this flake behaviour is essential, we suggest keeping the original method [JHY*14] for close-range, directly visible glints, and switch to our method for indirect illumination and elements further away from the camera.

6. Conclusion

We have presented a fast model for computing an approximated stochastic microfacet model for glints and flakes in materials. We replace the 4-dimensional search for glints with two 2-dimensional searches, for a significant speedup.

This separation between space and angle in the search allows for further improvements: hierarchical representation for the directional table, jumping straight to the hierarchical level for the spatial search. We also prefilter the directional table, depending on the material roughness for faster convergence in low-frequency environments. Finally, we use this simplified model for automatic filtering in global illumination: we compute the footprint of secondary rays and integrate the glints contribution directly.

Our algorithm is especially interesting for indirect lighting and global illumination, where it is more than 30 times faster than the reference, without any visible quality degradation. Our algorithm is highly *consistent*: rendering time does not depend on material parameters, making it easy to predict the total rendering time for a given scene. This is an important feature for practical use.

In future work, we want to extend the filtered model to scratches and other local surface details. We also plan to port the model for interactive rendering on the GPU.

Acknowledgements. We thank the reviewer for the valuable comments. We thank the following blendswap.com artists for providing the scenes used in this paper: Angelo Borraccino (Kitchen) and Neil Taylor (Car). This work has been partially supported by the National Key R&D Program of China under grant No. 2017YFB0203000, the National natural science foundation of China under grant No. 61802187, 61472224, 61472225, U1713208, 61472187, the 973 Program under grant No. 2014CB349303, Program for Changjiang Scholars, the Natural Science Foundation of Jiangsu under grant No. BK20170857, the fundamental research funds for the central universities No. 30918011320, Open Project Program of the State Key Lab of CAD&CG Zhejiang University under grant No. A1804 and ANR project ANR-15-CE38-0005 "Materials".

References

[AK16] Atanasov A., Koylazov V.: A practical stochastic algorithm for rendering mirror-like flakes. In *ACM SIGGRAPH 2016 Talks* (2016), SIGGRAPH '16, pp. 67:1–67:2. 2

[BW15] Bowles H., Wang B.: Sparkly but not too sparkly! anti-aliasing a procedural sparkle effect. ACM SIGGRAPH 2015 Course, Aug. 2015. 2

[BYRN17] Belcour L., Yan L.-Q., Ramamoorthi R., Nowrouzezahrai D.: Antialiasing Complex Global Illumination Effects in Path-space. *ACM Transactions on Graphics 36*, 1 (2017). 3

[CT82] Cook R. L., Torrance K. E.: A reflectance model for computer graphics. *ACM Trans. Graph. 1*, 1 (Jan. 1982), 7–24. 1, 3

[GCG*05] Günther J., Chen T., Goesele M., Wald I., peter Seidel H.: Efficient acquisition and realistic rendering of car paint. In *Vision, Modeling, and Visualization 2005* (2005), pp. 487–494. 2

[GRG14] Günther T., Rohmer K., Grosch T.: Particle-based simulation of material aging. In *GPU Pro 5: Advanced Rendering Techniques*, Engel W., (Ed.). A K Peters / CRC Press, 2014, pp. 35–53. 5

[Ige99] Igehy H.: Tracing ray differentials. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (1999), SIGGRAPH '99, pp. 179–186. 5

[Jak10] Jakob W.: Mitsuba renderer. http://www.mitsuba-renderer.org/, 2010. 5

[JHY*14] Jakob W., Hašan M., Yan L.-Q., Lawrence J., Ramamoorthi R., Marschner S.: Discrete stochastic microfacet models. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014) 33*, 4 (2014). 1, 2, 3, 4, 5, 6, 7, 8, 9

[RL00] Rusinkiewicz S., Levoy M.: QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), pp. 343–352. 3

[RMS*08] Rump M., Müller G., Sarlette R., Koch D., Klein R.: Photo-realistic rendering of metallic car paint from image-based measurements. *Computer Graphics Forum 27*, 2 (Apr. 2008), 527–536. 2

[Sho12] Shopf J.: Gettin' procedural. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/Shopf-Procedural.pdf, 2012. 2

[WB16] Wang B., Bowles H.: A robust and flexible real-time sparkle effect. In *EGSR 2016 E&I - Eurographics Symposium on Rendering - Experimental Ideas & Implementations* (Dublin, Ireland, 2016), The Eurographics Association, pp. 49–54. 2

[WMLT07] Walter B., Marschner S. R., Li H., Torrance K. E.: Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (2007), EGSR'07, pp. 195–206. 1, 3

[WN95] Wheeler D. J., Needham R. M.: Tea, a tiny encryption algorithm. In *Fast Software Encryption* (Berlin, Heidelberg, 1995), Preneel B., (Ed.), Springer Berlin Heidelberg, pp. 363–366. 4

[YHJ*14] Yan L.-Q., Hašan M., Jakob W., Lawrence J., Marschner S., Ramamoorthi R.: Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014) 33*, 4 (2014). 2

[YHMR16] Yan L.-Q., Hašan M., Marschner S., Ramamoorthi R.: Position-normal distributions for efficient rendering of specular microstructure. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2016) 35*, 4 (2016). 2

[ZK16] Zirr T., Kaplanyan A. S.: Real-time rendering of procedural multiscale materials. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2016), I3D '16, pp. 139–148. 2

[ZPvBG01] Zwicker M., Pfister H., van Baar J., Gross M. H.: EWA volume splatting. In *IEEE Visualization 2001 Proceedings* (2001), pp. 29–36. 3

[ZPvBG02] Zwicker M., Pfister H., van Baar J., Gross M.: Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 223 – 238. 3