Dynamic sparkle rendering: photorealistic real-time visualization of glittering surfaces in interactive scenarios

R. ĎURIKOVIČ, R. KICA AND A. MIHÁLIK

Abstract

The need for photorealistic rendering of glittery materials, motivated by applications in gaming, film, virtual reality, and particularly automotive visualization, requires real-time techniques to depict millions of minute flakes in paint coatings. Existing approaches often rely on a single grid for glint mapping, leading to repetitive patterns, aliasing, and noticeable popping during camera movement. Addressing these issues is essential for achieving convincing sparkle under varying lighting and viewpoints.

We propose a new real-time method that integrates three procedurally generated microflake grids, each with multiple resolution levels, to ensure smooth transitions between levels of detail and eliminate tiling artifacts. Adaptive scaling dynamically adjusts sparkle size and brightness based on camera distance, while per-flake light-color interactions enhance reflection realism. Our custom shader framework, paired with an artist-friendly material editor, enables real-time adjustments to flake density, size, falloff, and color.

Compared to single-grid or fixed-scale methods, our multi-grid approach delivers dense, non-repetitive sparkles with consistent quality across distances, albeit with higher GPU demands in complex scenes. The controllable and diverse output of our system also makes it well-suited for generating synthetic training data for AI models focused on material recognition, lighting estimation, or view-dependent appearance. Future work will explore deferred rendering, 3D noise textures, optimized shader branching, and temporal filtering to enhance performance and visual quality.

Mathematics Subject Classification 2000: 68U05, 78A05, 68P01
Keywords: glitter, perception, Unity 3D, High Definition Render Pipeline.

1. INTRODUCTION

Our research focuses on glittery materials, which feature tiny, variably oriented flakes in their surfaces. These flakes reflect light at specific angles, producing a shimmering effect as the viewing or lighting direction changes. Common examples include metallic car paints, glitter on textiles, and natural materials such as snow. Accurately representing and rendering these complex materials involve two main challenges: appropriate microstructure representation and efficient rendering. In this section, we examine existing approaches to microstructure modeling and rendering in both offline and real-time contexts, analyzing their respective strengths and limitations.

In interactive applications, Physically Based Rendering (PBR) is used to realistically depict how surfaces appear by accounting for phenomena such as light

10.2478/jamsi-2025-0009

scattering, refraction, diffusion, and reflection in both opaque and transparent materials. The surrounding environment greatly influences a surface's appearance, a concept known as Global Illumination (GI) [Greenberg et al. 1997], for which various approximation techniques are employed. Materials can be either explicitly defined—using parametric UV mapping and image textures [Ďurikovič 2002], [Heckbert 1989]—or [Ďurikovič et al. 2019], [Pietroni et al. 2010], meaning they are defined parametrically and their textures are generated at runtime. To capture surface detail [Ďurikovič and Mihálik 2013], large textures or tiled smaller ones are typically used. In practice, explicit maps suffice for most surfaces, offering adequate sampling within a certain viewing range. MIP-mapped textures [Williams 1983] are used for distant views, while close-ups may use detail maps. However, these approaches fall short when representing fine features like scratches or glitter, as they cannot fully convey the underlying microstructure.

A fundamental concept in photorealistic image synthesis is the Rendering Equation, which models the flow of light in a scene [Immel et al. 1986]. It describes how light interacts with surfaces by combining both emitted and reflected components to compute the radiance observed from a specific point:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o)\, L_i(\mathbf{x}, \omega_i)\, (\mathbf{n} \cdot \omega_i)\, d\omega_i, \tag{1}$$

where $L_o(\mathbf{x}, \omega_o)$ is the outgoing radiance at surface point $\mathbf{x}$ in direction ω_o, representing the light reaching the observer. $L_e(\mathbf{x}, \omega_o)$ is the emitted radiance from the surface, typically nonzero only for light sources. Ω denotes the hemisphere of all incoming directions ω_i above the surface at $\mathbf{x}$ and $f_r(\mathbf{x}, \omega_i, \omega_o)$ is the Bidirectional Reflectance Distribution Function (BRDF), describing how incoming light is scattered toward the viewer. $L_i(\mathbf{x}, \omega_i)$ is the incoming radiance from direction ω_i.

In computer graphics, to simulate light–surface interaction efficiently, various simplified models have been developed [Ward 1992]. These models generally fall into two categories: empirical and physically based. Empirical models, such as the Phong and the more widely adopted Blinn–Phong models, approximate lighting using heuristic observations [Phong 1975]. They rely on local illumination and compute light as a sum of ambient, diffuse, and specular components based on light and material properties. While computationally efficient, these models lack physical accuracy.

In contrast, PBR models [Ershov et al. 2004] have gained popularity due to their realism, consistency under varying lighting conditions, and compatibility with

modern graphics hardware. These models often incorporate variants of the BRDF or the more general Bidirectional Scattering Distribution Function (BSDF), frequently using microfacet theory to simulate the statistical behavior of surface microstructures.

Glittery surface details, such as sparkles caused by tiny flakes, are most noticeable under strong lighting but are challenging to render due to their small size and complexity. To achieve realistic real-time rendering, an efficient and compact representation of surface microstructure is essential—one that captures key optical characteristics while minimizing storage and computational overhead. This is particularly important in complex scenes containing multiple materials with varying surface properties. Microstructures can be modeled either explicitly, by directly representing individual features, or implicitly, through mathematical or statistical approximations that balance realism with performance [Ershov et al. 2004], [Wang et al. 2018].

The macroscopic appearance in PBR is modeled stochastically by defining the distribution of microscopic facet orientations using a Normal Distribution Function (NDF). When calculated using the viewing direction, the NDF expresses how much light is reflected towards the observer [Cook and Torrance 1982]. To accurately determine the amount of reflected light, it is essential to account for factors such as the distribution of microfacet orientations [Trowbridge and Reitz 1975], as well as masking and shadowing effects [Heitz 2014].

Light–surface interaction can be described at different spatial scales, each exhibiting distinct optical properties. For glittery materials, both microscopic and macroscopic representations are important. At the microscopic level, surfaces contain small, variably oriented flakes that reflect light in specific directions, creating glints. While accurate modeling would require explicit geometry, this is often too costly [Ďurikovič and Martens 2003]. Instead, macroscopic models such as the BRDF are used to approximate the overall optical response [Ďurikovič et al. 2003]. Various approaches exist to model surface appearance at each scale, depending on the desired balance between realism and performance. In [Zirr and Kaplanyan 2016], a biscale NDF is introduced to provide control over both the global surface appearance and the shape of individual microdetails, while enabling efficient procedural detail generation. This approach simulates a hierarchy of scales and accurately estimates pixel footprints at multiple levels of detail, achieving temporal stability and effective antialiasing. To simulate the sparkling effect in lithophanes, the

authors adopted the surface reconstruction and micro-hole modeling [Ďurikovič et al. 2021], enabling photorealistic visualization of etched surfaces with dynamic sparkle.

A fast and practical procedural sparkle effect for snow and other glittery surfaces can be achieved by intersecting a jittered 3D grid of sparkle seed points with the rendered surface. This technique generates sparkle glints procedurally. Since the sparkle effect contains high-frequency components, careful handling is required to ensure an anti-aliased and noise-free visual result [Wang and Bowles 2016].

In [Chermain et al. 2020], the authors propose a physically based BRDF for real-time rendering of glints, capable of reproducing the appearance of sparkling materials such as rocks, rough plastics, and glitter fabrics. Their BRDF employs normalized NDFs and converges to the standard microfacet model as the number of microfacets increases. The method procedurally generates NDFs using a dictionary of 1D marginal distributions: at each location, two distributions are randomly selected, multiplied to form an NDF, rotated to introduce variation, and scaled to control roughness. The dictionary is multiscale, independent of roughness, and memory-efficient.

The approach described in [Deliot and Belcour 2023] enables real-time rendering of glittery materials resulting from discrete surface flakes. It estimates the number of flakes reflecting light toward the camera within each texel covered by a given pixel footprint. The authors derive a counting method for arbitrary footprints that yields accurate statistical results. This method is combined with an anisotropic parameterization of texture space, which reduces the number of texels covered by a pixel footprint, improving both efficiency and visual fidelity. This method still has several limitations. It is not fully physically accurate and may violate the law of energy conservation. The counting process is defined in terms of the NDF, which does not account for lighting, thereby restricting the method to rendering glints only with point light sources. Additionally, it relies on UV coordinates, and discontinuities in the UV map can result in visual artifacts.

2. MODELING AND RENDERING GLITTERY MATERIALS

The shimmering effect, or glint, arises from the interaction of light with numerous tiny, variably oriented flakes embedded in a material's surface. Accurately simulating this phenomenon in real-time involves addressing two primary challenges: representing the microstructure and rendering it efficiently. Our approach addresses both by employing a 3D jittered grid of spheres to simulate sparkles dynamically on the surface.

Glints are generated using a uniform 3D grid populated with small spheres, which are then projected onto the material surface. The grid's resolution is dynamically adjusted based on the camera distance. To avoid visible repetition and undesirable patterns associated with high glint densities, we introduce multiple grids. The number of grids and the random offset of their centers can be configured by the user, enabling dense glint distributions without perceivable regularity—particularly effective on flat surfaces and smooth geometries like spheres. However, increasing grid complexity results in higher computational cost.

To enhance visual realism, each glint is scaled based on its distance from the camera and blended across adjacent levels of detail, allowing higher grid densities without artifacts. Shading is computed by evaluating light intensity at a surface point P as seen from the camera position C. The glint color is influenced by the light source, while global surface roughness and an anisotropic specular lighting model control the surface appearance and direction-dependent reflections.

Assuming a uniform grid of spheres and attributing glints to their intersections with the surface results in a regular (see Fig. 1), repetitive pattern that lacks the visual characteristics of natural sparkle effects. To achieve a more realistic and visually convincing result, the grid must be randomized, and its scale adapted dynamically—typically based on the camera distance or other view-dependent parameters.

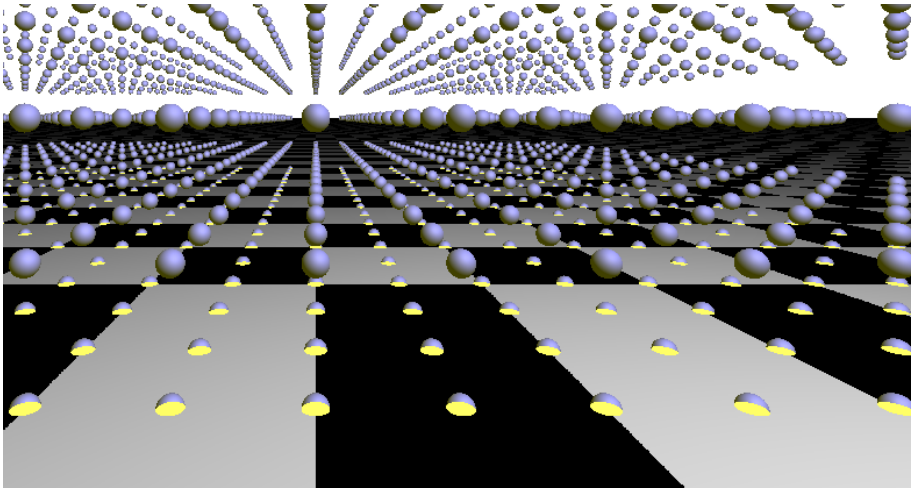

Fig. 1: A uniform grid of spheres, with sparkles represented by the intersection of each sphere with the surface.

The perceived size of sparkles and the effective density of the underlying grid are only appropriate within a limited range of camera distances. Outside this range, sparkles may appear excessively large or small, leading to noticeable aliasing artifacts. Although scaling sparkle size based on camera distance helps maintain a consistent screen-space appearance, this approach can cause sparkles in the background to exceed the bounds of their local grid cells, resulting in visual inconsistencies. Abrupt changes in grid scale can introduce discontinuities between adjacent grid levels, leading to visually distracting "popping" artifacts in the sparkle effect. To mitigate this issue, and drawing inspiration from the grid scale level computation described in [Wang and Bowles 2016], we combined three instances of grids, each supporting multiple scale levels. This approach ensures smoother transitions and more stable visual results across varying viewing distances.

To make the grid scale level of each instance $i \in \{-1,0,1\}$ adaptive based on the camera distance, the level parameter l is computed as follows:

$$l = \frac{1.2+5i}{10\zeta}, \tag{2}$$

where

$$\zeta = \frac{1.3^z - 2}{10},$$

$$z = \left\lfloor \frac{log_2(\frac{3}{10}\|P-C\|+3)}{0.37851162325} \right\rfloor,$$

C denotes the camera position and P is the surface point under consideration. The vector $P-C$ thus forms the view vector.

When rendering stochastic sparkles on distant surfaces, relying solely on the normalized view vector can introduce visual artifacts. As the view vector becomes nearly uniform across large surface areas far from the camera, it fails to provide sufficient variation for sparkle generation, resulting in a static, "frozen" pattern rather than a dynamic shimmer. To address this, additional sources of variation—such as screen-space coordinates or temporal components—are often introduced. However, a more coherent and deterministic approach is to remap the view vector itself. This remapping transforms the nearly constant view direction into a highly variable one, ensuring that each pixel receives a unique input. As a result, a visually convincing sparkle effect is maintained, regardless of the camera's distance.

According to the method described in [Wang and Bowles 2016], we define the

remapped view vector as:

$$v = \operatorname{sgn}(v)\left(|v| - \lfloor |v| \rfloor\right), \tag{3}$$

where

$$v = \zeta \frac{P - C}{\|P - C\|}.$$

This formulation ensures variation across pixels even when the original view vector is nearly constant.

To position particles in 3D space while maintaining randomness and incorporating view-dependent variation, we scale the world coordinates and add a remapped view vector. This ensures that the sparkle effect responds to changes in the viewer's position:

$$p = p_s \times l \times P + p_v \frac{v + \vec{r}}{\|v + \vec{r}\|}, \tag{4}$$

where p_s is a user-defined parameter controlling the scale relative to the size of the 3D model and the desired sparkle density, and p_v determines the influence of the view-dependent direction on the sparkle effect. P denotes the world-space position of the surface point, v is the remapped view vector, $\vec{r}$ is a small random offset introduced to increase spatial variation, and l is the grid scale level that adapts based on the camera's distance to the surface (see equation (2)). The grid lookup position is determined by flooring the computed randomized position, that is, $\lfloor p \rfloor$.

To decide if the sparkle is present at the point on the surface we need to compute an offset:

$$\Delta p = p - \lfloor p \rfloor \tag{5}$$

To incorporate randomness into the grid, we jitter its center using Perlin noise. The randomized offset is computed as:

$$\Delta p = p - \lfloor p \rfloor + \frac{1}{2} \begin{pmatrix} p_n \times \eta + \frac{1}{2} - \lfloor p_n \times \eta + \frac{1}{2} \rfloor \\ p_n \times \eta + \frac{1}{2} - \lfloor p_n \times \eta + \frac{1}{2} \rfloor \\ p_n \times \eta + \frac{1}{2} - \lfloor p_n \times \eta + \frac{1}{2} \rfloor \end{pmatrix} - \frac{1}{4} \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}, \tag{6}$$

where η is a scalar value obtained from 3D Perlin noise, as described in [Perlin 1985], evaluated at the point:

$$\vec{a} = l \times p_d \frac{\lfloor p \rfloor}{l \times p_s} \vec{r}, \tag{7}$$

and p_d is a user-defined parameter controlling the noise density. Here, p_n adjusts the jitter amount, l is the grid scale level, p_s is the spatial scale of the grid, and $\vec{r}$ is the random vector.

Intuitively, this jittering process can be understood as introducing slight perturbations to the sparkle grid, preventing sparkles from aligning in a rigid, artificial pattern. Perlin noise provides smooth, spatially coherent randomness—analogous to the irregular distribution of glitter flakes in real materials—so each sparkle is shifted in a controlled manner that reduces repetition while maintaining visual coherence across the surface.

When a 3D grid of sparkle points is viewed from shallow angles, the points become compressed and aligned into thin lines in screen space. This compression causes a moiré pattern or a sparse, undersampled appearance, as the sparkle glints appear too far apart to form a continuous shimmer. To address this, anisotropy is introduced by stretching the specular highlight of each sparkle point. Rather than rendering as small circular highlights, the sparkle points produce elongated streaks that overlap and visually connect. This overlap fills the gaps between adjacent sparkle points, creating a smooth, high-frequency shimmer that remains visually consistent and free of aliasing artifacts, even at grazing angles. To achieve this effect, we transform the spheres into ellipsoids elongated along the surface tangent direction. The modified offset is then computed as follows:

$$\rho = \Delta p + \frac{(|(P-C)\cdot N| - 1)\,(\Delta p \cdot \vec{d})\vec{d}}{\vec{d}\cdot\vec{d}}, \tag{8}$$

where

$$\vec{d} = \frac{P-C}{\|P-C\|} - ((P-C)\cdot N)N,$$

and N is the surface normal at point P.

To compute the contribution to the intensity at an arbitrary surface point with coordinates P, generated by a sparkle viewed from the camera at position C, we evaluate:

$$\iota = \begin{cases} 20\dfrac{\tau-\delta}{\tau}, & \text{if } \tau > \delta \\ 0, & \text{otherwise} \end{cases}, \tag{9}$$

where

$$\delta = \rho \cdot \rho, \quad \tau = \frac{p_s^2}{75000}. \tag{10}$$

The resulting intensity ι is then multiplied by the sparkle color to produce the final shading contribution. This algorithm is performed for each instance $i \in \{-1,0,1\}$, and the resulting intensities are summed. Since the vector $\vec{r}$ appearing in equations (4) and (7) is a random vector, the whole algorithm can be executed multiple times with different random seeds, and the results can be summed.

To obtain a plausible sparkle color, we multiply the resulting color by the specular coefficient s, computed using the anisotropic reflection model from [Ward 1992]:

$$s = \begin{cases} 0, & \text{if } (N \cdot L) \leq 0 \ \wedge \ (N \cdot V) \leq 0, \\ \alpha \times \beta \times \gamma, & \text{otherwise}, \end{cases} \tag{11}$$

where

$$\alpha = \sqrt{\max\left(0, \frac{0.045\,(N \cdot L)}{N \cdot V}\right)},$$
$$\beta = \max\left(0, \frac{N \cdot L}{4\pi\alpha_x\alpha_y}\right),$$
$$\gamma = e^{-2\left(\frac{t_1^2+t_2^2}{1+(H\cdot N)}\right)}.$$

Here, L is the normalized vector from the surface point P to the point light source,

$$V = \frac{C-P}{\|C-P\|},$$

and the halfway vector H is

$$H = \frac{L+V}{\|L+V\|}.$$

The parameters α_x and α_y are user-defined roughness values in the tangent and bitangent directions, respectively, introducing anisotropy. The variables t_1 and t_2 are defined as

$$t_1 = \frac{H \cdot T}{\alpha_x}, \quad t_2 = \frac{H \cdot B}{\alpha_y},$$

where T is the surface tangent and B is the bitangent:

$$B = N \times T.$$

3. IMPLEMENTATION

In our implementation, we compute the sparkle intensity (Equation (9)) three times for each instance $i \in \{-1, 0, 1\}$ using a different seed for the random vector $\vec{r}$ for each visible surface point of the 3D object. The resulting values are summed and then multiplied by the specular coefficient (Equation (11)). This product is further multiplied by a white RGB color to obtain the final sparkle color. The parameters used in the computation are listed in Table I.

Table I: Parameters used in the sparkle rendering implementation.

Parameter	Value
p_s	75
p_v	8
p_n	100
p_d	7.5
α_x	0.5
α_y	0.5

Our implementation is based on HLSL (High-Level Shading Language) within the High Definition Render Pipeline (HDRP) of Unity version 2022.1. HLSL is a programming language designed for writing shaders, enabling precise control over the GPU's rendering pipeline and allowing us to implement custom lighting and sparkle effects at a low level. The sparkle effect is integrated into the forward rendering path, where parameters for each visible surface point are computed prior to the application of standard diffuse and specular lighting. The method allows user-defined parameters to be adjusted in real time, enabling direct observation of the visual impact on the sparkle appearance. Randomness is introduced using a 3D Perlin noise function [Perlin 1985], evaluated directly at runtime with the open-source `FastNoiseLite` library [Peck 2024]. This approach avoids additional memory usage associated with precomputed 3D textures, at the expense of a moderate performance cost for dense sparkle grids. The system was tested on a machine equipped with an Intel i7-9750H processor and an NVIDIA GTX 1660 Ti GPU, ensuring stable performance at interactive frame rates.

4. RESULTS

We evaluated our sparkle rendering method with a focus on visual quality and performance. All experiments were conducted at 4K resolution (3840 $\times$ 2160, 8,294,400 pixels per frame) to ensure the GPU remained the main performance

bottleneck. In addition to average frame rate, we measured frametime consistency to assess temporal stability. Tests were performed using HDRP's *High Fidelity* settings with ambient occlusion enabled.

Our evaluation comprised three scenarios. The first two used static camera views to compare five different configurations of objects and lights, enabling controlled comparisons of performance and appearance without motion artifacts. The third scenario featured an animated camera in a complex scene containing multiple objects that shared the same rendering method but had slightly varied parameters. This dynamic setup was designed to reveal potential flickering or instability during motion.

The goal of this evaluation is to benchmark the method from Section 2 against other state-of-the-art techniques, including Chermain et al. [Chermain et al. 2020], Deliot and Belcour [Deliot and Belcour 2023], Zirr and Kaplanyan [Zirr and Kaplanyan 2016], and Wang and Bowles [Wang and Bowles 2016], under identical conditions. Each method was assessed in terms of ease of use, implementation complexity, limitations, performance impact, parameter relevance, and final visual quality. For visual inspection, we noted whether sparkles maintained stability or exhibited flicker under motion, as well as the presence of artifacts such as aliasing or inconsistent highlight shapes.

We begin our analysis with the first test scenario, designed to provide a controlled baseline for comparing the rendering methods. In this setup, a single point light with a range of 200 meters is positioned directly above the tested object. As shown in Fig. 2, we evaluate four different objects sequentially: a plane, a cube, a sphere, and a more complex 3D model. This progression allows us to observe how each method handles varying geometric complexity under otherwise identical lighting conditions. Average frame times for all methods and objects are presented in Table II.

Table II: Average delta time (in milliseconds) for rendering different 3D objects using various methods.

Method	**Plane**	**Cube**	**Sphere**	**Complex Model**
Chermain [Chermain et al. 2020]	41.90	15.06	19.70	13.50
Deliot [Deliot and Belcour 2023]	49.95	12.40	13.65	12.14
Zirr [Zirr and Kaplanyan 2016]	39.87	12.03	12.09	11.70
Wang [Wang and Bowles 2016]	19.33	10.60	10.11	9.80
Our	25.66	10.97	11.09	10.50

Overall, Wang's method achieves the lowest frame times across all objects, while Chermain and Deliot tend to be more computationally expensive. The plane shows

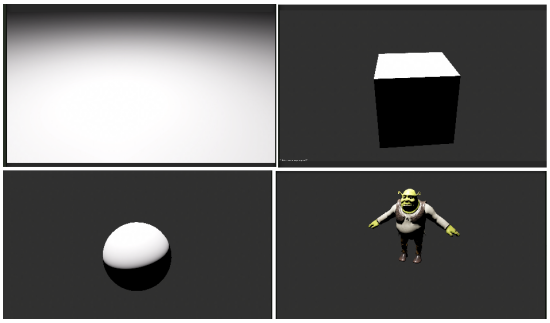

Fig. 2: Evaluation scenes for real-time rendering. The figure displays rendered scenes using a standard shader without sparkle effects, featuring a plane, cube, sphere, and intricate 3D model, utilized for evaluation purposes.

the worst performance because it exhibits the highest number of visible sparkles.

In contrast to the first scenario, the second scenario increases lighting complexity by using three point lights with the same $200m$ range, positioned side-by-side and facing away from the camera. The corresponding frame times for each method and object are reported in Table III.

Table III: Average frame time (ms) for rendering various 3D objects under three point lights using the tested methods.

Method	**Plane**	**Cube**	**Sphere**	**Complex Model**
Chermain [Chermain et al. 2020]	107.66	23.93	15.09	15.03
Deliot [Deliot and Belcour 2023]	106.61	14.87	13.51	12.22
Zirr [Zirr and Kaplanyan 2016]	84.74	15.13	13.19	13.76
Wang [Wang and Bowles 2016]	21.84	10.71	10.17	9.83
Our	44.07	12.06	11.33	10.95

The third testing scenario evaluates performance in a complex animated scene with numerous objects and dynamic lighting. All lights are point lights calculated in real time, with no precomputed lighting. The scene contains a total of 135 lights, although Unity culls those far from the camera during rendering. Additionally, a weak directional light simulates moonlight but does not contribute to the glint effect. At any given moment, approximately 35 objects exhibit glitter, and lights do not cast shadows. The animation averages around 450 draw calls, peaking at 683 draw calls (about 204 900 triangles and 153 600 vertices) at the 4-second mark. Performance results for all tested methods are presented in Table IV.

Because the 3D Perlin noise for the sparkle effect was generated directly at runtime, we also evaluated its performance impact in this demanding scene. The

Table IV: Performance in the animated complex scene. Frame times are in milliseconds.

Method	Average	Max	Min
Chermain [Chermain et al. 2020]	47.2	65.8	28.6
Deliot [Deliot and Belcour 2023]	48.1	66.2	32.5
Zirr [Zirr and Kaplanyan 2016]	42.0	62.8	26.7
Wang [Wang and Bowles 2016]	18.5	23.7	13.5
Our method with Perlin noise	44.8	65.8	32.4
Our method without Perlin noise	25.3	32.4	19.5

average frame time increased by approximately 43.5% compared to the baseline without procedural noise, demonstrating that real-time noise generation imposes a substantial computational cost, particularly under high geometric complexity and numerous active lights.

Following the quantitative results from the three test scenarios, we now provide a qualitative comparison of the evaluated methods, focusing on their visual characteristics, implementation constraints, and performance implications.

Chermain's method produces highly customizable and visually striking glints (Fig. 3), with the possibility of stretching them for anisotropic effects. However, performance costs grow proportionally with both the number of active light sources and the screen space covered by glints. While the glint size cannot be directly adjusted in the current implementation, this feature could be integrated. No major artifacts were observed during testing.

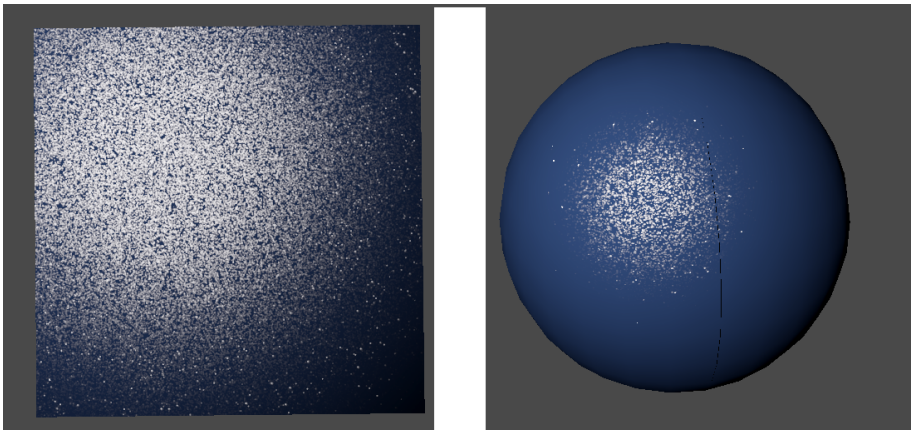

Fig. 3: Visualization of Chermain method.

Deliot's technique, illustrated in Fig. 4, is restricted to point lights and does not allow glint size adjustment. It has a relatively high performance cost but is more

consistent because it fits the pixel footprint using ratios, rotations, and scaling, which avoids the need for multiple evaluations. While visually the most appealing among the tested methods, its lack of variety in glint shapes and relatively high performance cost limit its flexibility.

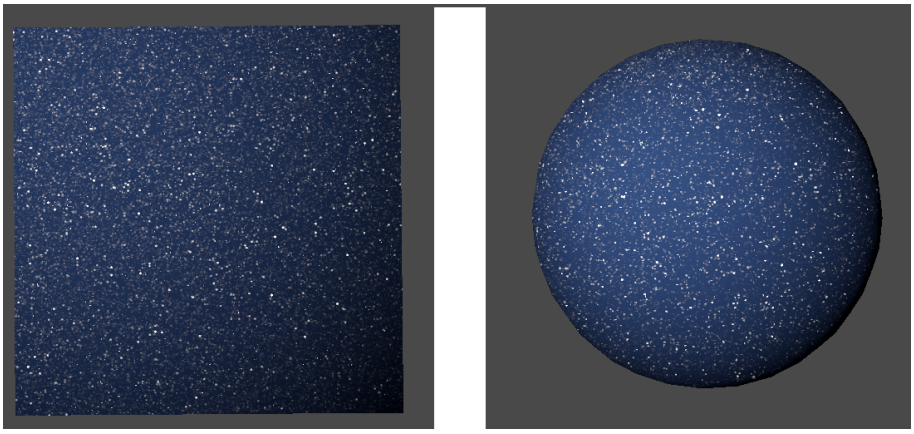

Fig. 4: Visualization of Deliot method.

The method by Zirr et al. (Fig. 5) offers only six parameters and does not support increasing glint size. Its implementation is relatively complex and computationally demanding. Achieving dense glints without artifacts is challenging, with common issues including grid-like patterns and stretched glints. While visually acceptable from a distance, these artifacts become apparent at close range.

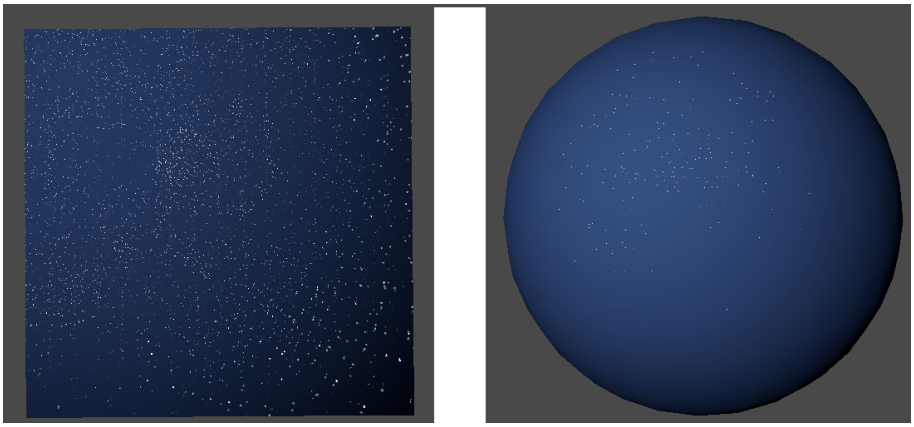

Fig. 5: Visualization of Zirr method.

The method by Wang et al. is computationally lightweight and simple to implement. It generates glints by placing a uniform grid of spheres intersecting the surface (Fig. 6). This approach produces sparse, small glints; increasing density introduces noticeable repeating patterns, limiting both maximum coverage and realism.

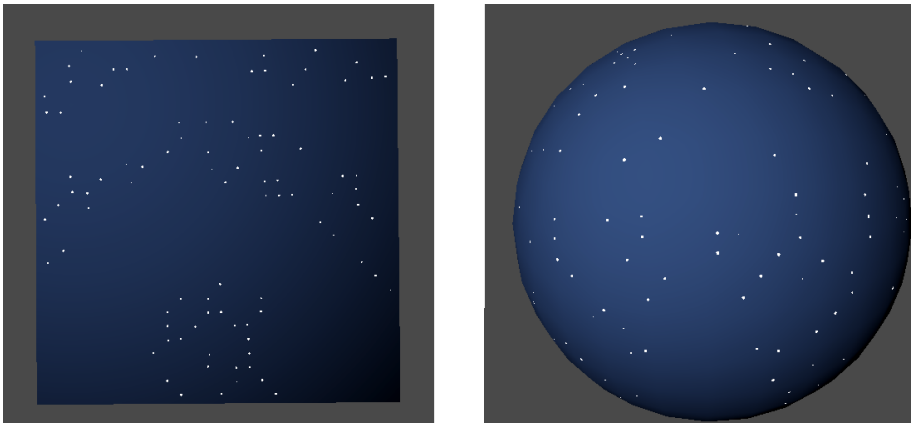

Fig. 6: Visualization of Wang method.

Our implementation builds on Wang's framework but introduces additional parameters to break up patterns and achieve denser glints (Fig. 7). This enables higher-quality results, particularly at medium-to-long viewing distances. However, the method requires careful parameter tuning to avoid flickering when glints become very small, and performance overhead is higher than Wang's due to the additional computations.

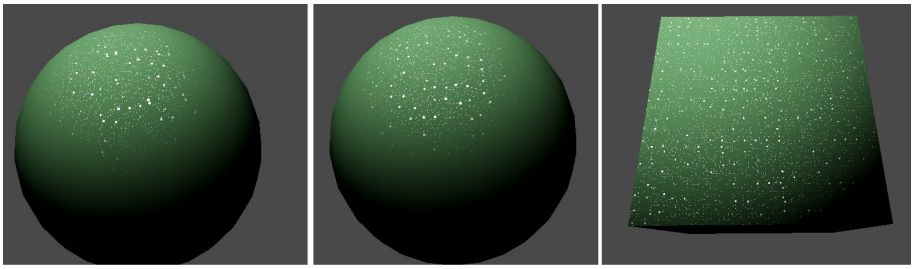

Fig. 7: Visualization of our method.

In summary, Chermain's approach [Chermain et al. 2020] offers strong visual customization but scales poorly with scene complexity. Deliot's method is visually

consistent and aesthetically pleasing but restricted in scope and computationally costly. Wang's method excels in speed and simplicity but sacrifices density and realism. Our method occupies a middle ground—more flexible and visually rich than Wang's, but with greater performance demands and tuning requirements.

5. CONCLUSION AND FUTURE WORK

Our approach sidesteps the classic UV-mapping pitfalls by introducing a multi-grid blending system. Instead of relying on surface coordinates, we overlay three randomized microflake grids—each with its own scale levels—to guarantee uniform glint size and density even on poorly mapped geometry. This innovation virtually eliminates the uneven sparkle and density "hot spots" that plague single-grid, UV-dependent methods.

In practice, our solution remains efficient and works seamlessly with all major light types. At extreme close-ups, very fine glints can still flicker—a known trade-off when pushing screen-space details to their limits. By contrast, the Deliot and Belcour technique delivers impressively stable visuals under changing view angles but is constrained to point lights and demands significantly more computation.

A practical approach would be to use the Deliot method for nearby objects and our implementation for distant geometry or non-point light scenarios, provided that tiny glints are removed and noise is precomputed. This hybrid solution would require a seamless transition between methods to avoid visual artifacts. Our comparative evaluation considered ease of use, performance, visual fidelity, implementation complexity, and limitations, showing that with tuned parameters, our method can achieve good visual quality.

Future work will focus on merging our method with that of Deliot et al. and addressing current deficiencies. Planned improvements include replacing runtime noise generation with a 3D texture, optimizing calculations to reduce GPU branching overhead, and mitigating flickering at small glint sizes. Real game environments often involve directional and spot lights, shadow casting, and complex post-processing pipelines. These factors may introduce additional challenges for sparkle stability and performance, particularly under high geometric complexity and dynamic lighting. Extending our method to fully support non-point light sources and validating scalability in production-scale game scenes will be an important step toward broader applicability. Additionally, exploring a deferred rendering pipeline could improve performance in scenes with many lights.

ACKNOWLEDGMENTS

The presented work was performed under a programme of the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project "InnovAIte Slovakia, Illuminating Pathways for AI-Driven Breakthroughs", contract No. 09I02-03-V01-00029.

REFERENCES

CHERMAIN, X., SAUVAGE, B., DISCHLER, J.-M., AND DACHSBACHER, C. 2020. Procedural physically based brdf for real-time rendering of glints. In *Computer Graphics Forum*. Vol. 39. Wiley Online Library, 243–253.

COOK, R. L. AND TORRANCE, K. E. 1982. A reflectance model for computer graphics. *ACM Trans. Graph. 1,* 1 (Jan.), 7–24.

DELIOT, T. AND BELCOUR, L. 2023. Real-Time Rendering of Glinty Appearances using Distributed Binomial Laws on Anisotropic Grids. *Computer Graphics Forum 42,* 8.

ERSHOV, S., ĎURIKOVIČ, R., KOLCHIN, K., AND MYSZKOWSKI, K. 2004. Reverse engineering approach to appearance-based design of metallic and pearlescent paints. *The Visual Computer 20,* 8, 586–600.

GREENBERG, D. P., TORRANCE, K. E., SHIRLEY, P., ARVO, J., LAFORTUNE, E., FERWERDA, J. A., WALTER, B., TRUMBORE, B., PATTANAIK, S., AND FOO, S.-C. 1997. A framework for realistic image synthesis. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., USA, 477–494.

HECKBERT, P. S. 1989. Fundamentals of texture mapping and image warping. Tech. Rep. UCB/CSD-89-516, EECS Department, University of California, Berkeley. Jun.

HEITZ, E. 2014. Understanding the masking-shadowing function in microfacet-based brdfs. *Journal of Computer Graphics Techniques 3,* 2, 32–91.

IMMEL, D. S., COHEN, M. F., AND GREENBERG, D. P. 1986. A radiosity method for non-diffuse environments. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. Association for Computing Machinery, New York, NY, USA, 133–142.

PECK, J. 2024. Fast noise lite. `https://github.com/Auburn/FastNoiseLite`. GitHub repository.

PERLIN, K. 1985. An image synthesizer. In *Computer Graphics (SIGGRAPH '85 Proceedings)*. Vol. 19. 287–296.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM 18,* 6 (June), 311–317.

PIETRONI, N., CIGNONI, P., OTADUY, M. A., AND SCOPIGNO, R. 2010. Solid-texture synthesis: A survey. *IEEE Computer Graphics and Applications 30*, 74–89.

TROWBRIDGE, T. S. AND REITZ, K. P. 1975. Average irregularity representation of a rough surface for ray reflection. *JOSA 65,* 5, 531–536.

ĎURIKOVIČ, R., KIMURA, R., AND KOLCHIN, K. 2003. Real-time visualization of japanese artcraft. In *Proceedings Computer Graphics International 2003*. 184–189.

ĎURIKOVIČ, R. AND MARTENS, W. L. 2003. Simulation of sparkling and depth effect in paints. In *Proceedings of the 19th Spring Conference on Computer Graphics*. SCCG '03. Association for Computing Machinery, New York, NY, USA, 193–198.

WANG, B. AND BOWLES, H. 2016. A robust and flexible real-time sparkle effect. In *EGSR 2016 E&I-Eurographics Symposium on Rendering-Experimental Ideas & Implementations*. The Eurographics Association, 49–54.

WANG, B., WANG, L., AND HOLZSCHUCH, N. 2018. Fast global illumination with discrete stochastic microfacets using a filterable model. *Computer Graphics Forum 37,* 7, 55–64.

WARD, G. J. 1992. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. 265–272.

WILLIAMS, L. 1983. Pyramidal parametrics. *Computer Graphics (SIGGRAPH '83 Proceedings) 17,* 3, 1–11.

ZIRR, T. AND KAPLANYAN, A. S. 2016. Real-time rendering of procedural multiscale materials. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 139–148.

ĎURIKOVIČ, R. 2002. Explicit method of sparkling effect simulation. *Journal of Three Dimensional Images 16,* 4, 96–100.

ĎURIKOVIČ, R. ET AL. 2021. Modelling and production design of lithophanes showing the sparkling effect. In *IEEE Conference Proceedings*. IEEE.

ĎURIKOVIČ, R., KUNOVSKÁ, L., AND MIHÁLIK, A. 2019. Sparkling effect in virtual reality device. In *Mathematical Insights into Advanced Computer Graphics Techniques*, Y. Dobashi, S. Kaji, and K. Iwasaki, Eds. Mathematics for Industry, vol. 32. Springer, Singapore, 51–58.

ĎURIKOVIČ, R. AND MIHÁLIK, A. 2013. Metallic paint appearance measurement and rendering. *Journal of Applied Mathematics, Statistics and Informatics 9,* 2, 25–39.

Roman Ďurikovič
Faculty of Mathematics, Physics and Informatics,
Comenius University,
842 48 Bratislava, Slovak Republic
http://www.fmph.uniba.sk
Email: roman.durikovic@fmph.uniba.sk

Róbert Kica
Faculty of Mathematics, Physics and Informatics,
Comenius University,
842 48 Bratislava, Slovak Republic
http://www.fmph.uniba.sk
Email: kica4@uniba.sk

Andrej Mihálik
Faculty of Mathematics, Physics and Informatics,
Comenius University,
842 48 Bratislava, Slovak Republic
http://www.fmph.uniba.sk
Email: andrej.mihalik@fmph.uniba.sk