Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2016)
E. Eisemann and E. Fiume (Editors)

A Robust and Flexible Real-Time Sparkle Effect

Beibei Wang[*] How Bowles[+]

[*]INRIA; Université Grenoble-Alpes, LJK; CNRS, LJK [+]Studio Gobo

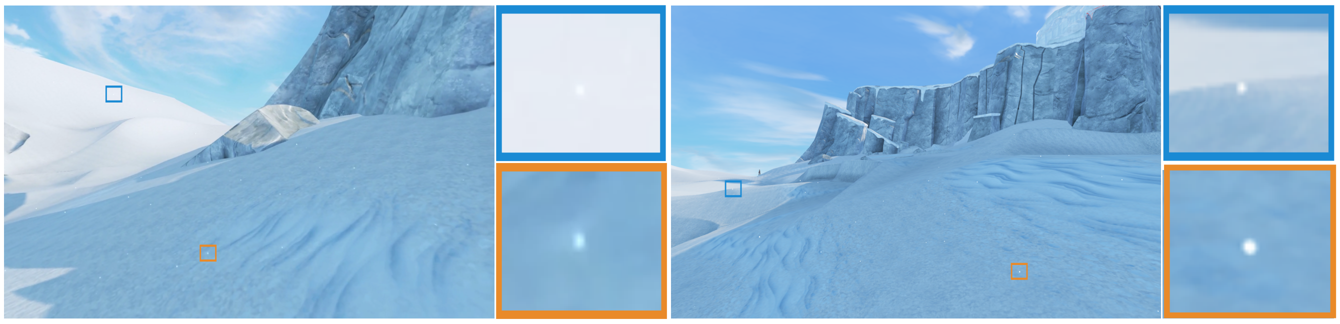

Figure 1: *Two scenes with sparkles.*

Abstract

We present a fast and practical procedural sparkle effect for snow and other sparkly surfaces which we integrated into a recent video game. Following from previous work, we generate the sparkle glints by intersecting a jittered 3D grid of sparkle seed points with the rendered surface. By their very nature, the sparkle effect consists of high frequencies which must be dealt with carefully to ensure an anti-aliased and noise free result. We identify a number of sources of aliasing and provide effective techniques to construct a signal that has an appropriate frequency content ready for sampling at pixels at both foreground and background ranges of the scene. This enables artists to push down the sparkle size to the order of 1 pixel and achieve a solid result free from noisy flickering or other aliasing problems, with only a few intuitive tweakable inputs to manage.

1. Introduction

Sparkle is an important visual effect that can add visual interest to snow and other surfaces, but is difficult to simulate robustly due to its high frequency nature. In this work our aim is not to simulate a realistic snow sparkle effect, but to make an interesting and dynamic effect suitable for games and other high performance applications. The challenge of this problem is to produce a sharp glints of light less than a few pixels in size, which appear randomly distributed and respond in a believable way when the viewer moves, and are free from aliasing artifacts.

This effect is difficult to achieve with normal maps, so we resort to procedural solutions. Shopf [Sho12] introduced a method which embeds sparkles into 3D space on a grid, which is then intersected with sparkly surfaces. The grid is deformed by a noise function to break up patterns and the view vector is also taken into account to get a dynamic sparkly effect when the view changes. This method is simple but has several issues, such as visibly distorted sparkle shapes, limited depth range and insensitivity to the pixel sampling rate leading to aliasing.

We provide a number of practical improvements to address these issues. We swap the sparkle shape for one that is more regular to address aliasing. We introduce a number of improvements to expand the distance range including a depth-dependent sparkle scale and an adaptive yet stationary grid, and fixes to make the view dependent work regardless of depth. We also change the way noise is applied to prevent strong distortion of the sparkle shape that is prone to aliasing. Finally, to handle anisotropy at shallow view angles we adjust the shading signal analytically without resorting to super sampling with just a few additional instructions.

To summarize, the main contributions of this paper include:

- a new sparkle shape that is simpler, cleaner and more robust,
- an adaptive sparkle density according to the depth in order to expand distance range,

DOI: 10.2312/sre.20161209

- extensions to ensure the effect sparkles in the distance where the view vector is long,
- noise distortion on grid centers rather than the each grid lookup to produce a clean and anti-aliased sparkle,
- an anisotropic sparkle shape to eliminate aliasing at shallow viewing angles.

Our approach can be easily integrated into the lighting pass of existing renderers. The details of the integration can depend on art direction, for example we opted to not take shadows into account. The result is a compelling and flexible effect which is performant and suitable for use in production.

2. Previous Work

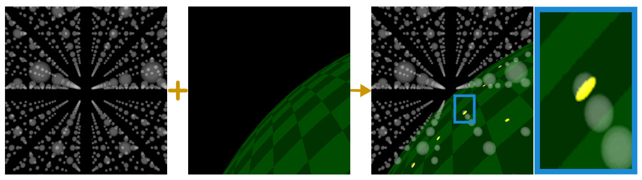

Figure 2: *Sparkle algorithm proposed by Shopf [2012]. Noise is disabled here for clarity.*

Shopf [Sho12] proposed a procedural approach by intersecting the shaded surface with a 3D jittered grid of sparkle shapes (Figure 2), which works well for simple scenes/viewpoints. This is an elegant solution but suffers from a severely limited depth range, and is prone to strong aliasing due to the following factors. The sparkle shape is sharp and pointy and does not scale down gracefully (Figure 3a). The way noise was applied to jitter the grid was also prone to aliasing and distorted sparkle shapes. Additionally no accommodation was made for anisotropic sampling rates, leading to aliasing at shallow view angles (see blue part in Figure 7). In this paper, we address each of these issues and produce a stable and coherent solution.

Oatet al. [OTI03] introduced a technique to simulate car paint by using multiple layers, one of which is for the metallic flakes.

Edwards [Edw12] described a sparkle effect implemented for the video game *Journey* achieved using a noisy normal map with its mip textures artificially scaled up to boost low frequencies and help extend the range of the sparkle. The range is still limited however by the normal map resolution, and specific code is required to produce the custom filtered mip chain. Our approach does not require supporting systems and works over all ranges of depth empowering the artist to decide where the effect should be applied. Having sparkles in the far distance had a miniaturizing effect which matched the art style and theme of our game.

Yan et al. [YHJ*14] introduced an analytic sparkle renderer to compute accurate and high quality sparkle effects for offline rendering. [JHY*14] proposed a stochastic model based on microfacet theory to simulate glittery surfaces.Both of these methods are not suitable for real-time rendering. Zirr et al. [ZK16] derived a stochastic biscale microfacet model to fit for real-time application, however, it is still slower than our method.

3. Sparkle Effect

The following sections describe the improvements we made to the approach from Shopf [Sho12].

3.1. Sparkle Shape

The sparkle shape used by Shopf is shown in Figure 3a and is produced by Code 1, where *shadingPoint* is the world space position of the surface point and k is a user input to control the size of the sparkles.

```
float3 p = fract(shadingPoint);
p *= 1.0 - p;
sparkle = saturate(1.0 - k * (p.x + p.y + p.z));
```

Code 1: *Sparkle function from Shopf.*

The sparkle shape is sharp and makes aliasing difficult to control, so we propose to use an inverted squared distance kernel shown in Figure 3b computed by lines 42 to 44 of Code 2. The comparison of these two shapes are shown in Figure 3. Our sparkle shape is much more regular and scales gracefully.

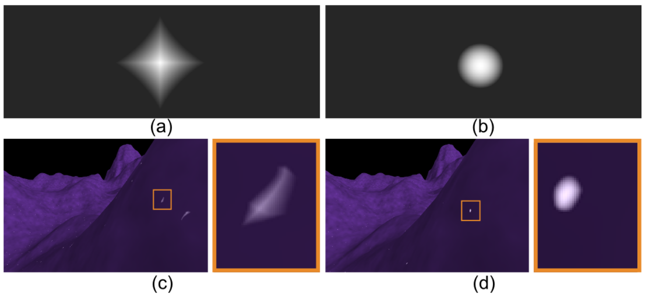

Figure 3: *Comparison of the star shape from Shopf (left) and our round kernel (right).*

3.2. Distance Range

In the approach from Shopf, the sparkle size and grid density are only appropriate within a narrow range of distances from the viewer. Sparkles outside this range are either too large (cf. orange part in Figure 7) or too small and prone to aliasing (cf. green part in Figure 7). Scaling the sparkle size by distance produces a constant sparkle size in screen space, however sparkles quickly outgrow the grid in the background (Figure 7b). Instead, we specify the sparkle size as a proportion of the grid size, and then scale the grid by the distance to ensure the grid is appropriate at all depths. The grid scale is stepped so that the grid is stationary within each step. We found the logarithmic distribution used in lines 10 to 15 of Code 2 produced a fairly even distribution of grid levels for typical views (Figure 4). The downside of using a step function is the discontinuity between grid scales which can make sparkles pop. One way to fix this would be to cross-fade sparkle grids, but this would require multiple sparkle computations. Another option would be to use a noise function to break up the boundaries between levels. We found any issues arising from this discontinuity were masked by the sparkling effect and we left the simple implementation.

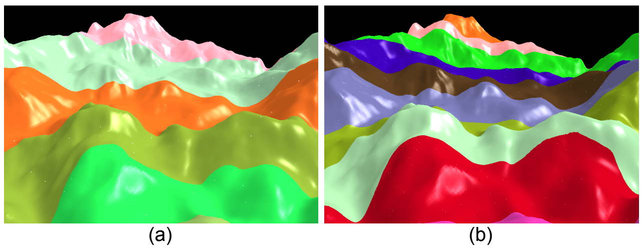

Figure 4: *Grid level visualization for (a) step linear distribution, (b) step logarithmic distribution*

3.3. View Vector Remapping

To achieve a sparkling effect when the view changes, the view vector is normalized and is used to offset the lookup into the sparkle grid. This works well for a range of depths, but breaks down in the distance where the view vector is long, because the normalized view vector does not change much when the view point moves (Figure 5a). Sparkles will appear static and stuck to the surface, breaking the illusion of the sparkle effect. To solve this issue, we apply domain repetition to the view vector so that moderate changes in the view vector are not lost after normalization (Figure 5b). See Code 2, line 20 for the implementation. This again introduces discontinuities, but again we found these were masked by the sparkling effect and were not detrimental to the render.

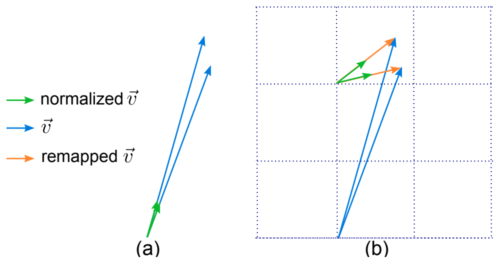

Figure 5: *View vector remapping. (a) View vectors (blue) very similar when normalized (green). (b) Modulus (orange) before normalization ensures variation in result (green).*

3.4. Noise Distortion

To hide the regular nature of the grid and avoid noticeable patterns, Shopf proposed to jitter the lookup into the grid by a noise function. This has the side effect of producing strong distortions of the sparkle shape (Figure 3c) which makes aliasing difficult to control. Rather than jittering the lookup position which affects the sparkle shape, we propose to jitter each sparkle grid *center position*. For each shading point we compute the nearby sparkle grid point and jitter its position, before evaluating the sparkle function as before (Code 2, lines 31 to 33). In this way we always obtain a clean sparkle at the jittered position (Figure 3d).

3.5. Anisotropic Sparkle Shape

At shallow view angles, the sparkle grid is undersampled in one direction, producing objectionable flickering as sparkles fall between pixels. We address this with an approach inspired by Elliptical Weighted Average texture filtering [Hec89]. We elongate the sparkle kernel from a sphere to an ellipse, where the major axis is a tangent to the surface that points in the undersampled direction (Figure 6). The grid lookup is then shifted towards the sparkle center along this axis based on the cosine of the viewing angle (Code 2, line 39). The resulting code collapses elegantly down to a few cheap instructions and eliminates aliasing at shallow view angles (Figure 9).

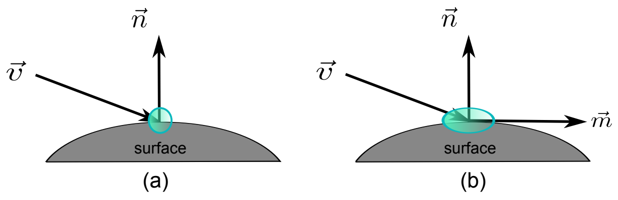

Figure 6: *Normal sparkle shapes (a) vs. our anisotropic shapes (b).*

4. Implementation

Code 2 provides sample shader code for computing sparkle intensity which can then be multiplied by lighting or other factors as required for the material look. The input variables of our technique are sparkle size and sparkle density which are intuitive and easy to tweak. The shader uses a 3D noise texture, in our results we used the 128^3 noise texture provided with RenderMonkey [ATI]. After experimentation we found that stepping the length of the view vector (Code 2, line 19) helped stabilize the sparkles under fast camera motion.

5. Results and Discussion

We implemented our effect into a AAA video game title, from which the snow scene is a captured (Figure 1, 11). We also implemented the effect in RenderMonkey, which we use to compare against the approach from Shopf.

Comparison to Previous Work. In Figure 7, we compare our approach to Shopf. Our approach is effective over a large range of depths, while Shopf only supports a narrow depth range. Sparkles outside this range tend to appear large and distorted or very small and prone to aliasing. Our approach does not suffer from such aliasing issues under camera motion, refer to the video material for a comparison.

In Figure 8, we move the view slightly and show the result with and without view vector remapping. Without the remapping, sparkles in the background do not change under camera motion. This is further demonstrated in the accompanying video material.

Figure 9 demonstrates how the anisotropic sparkle shape adapts to shallow view angles and prevents aliasing.

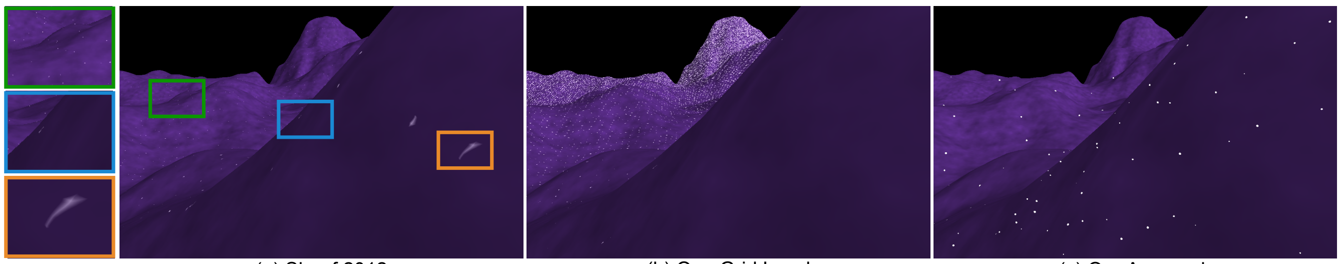

Figure 7: *Comparison to Shopf [2012]. Our approach can generate consistent sparkles over entire depth range if desired.*

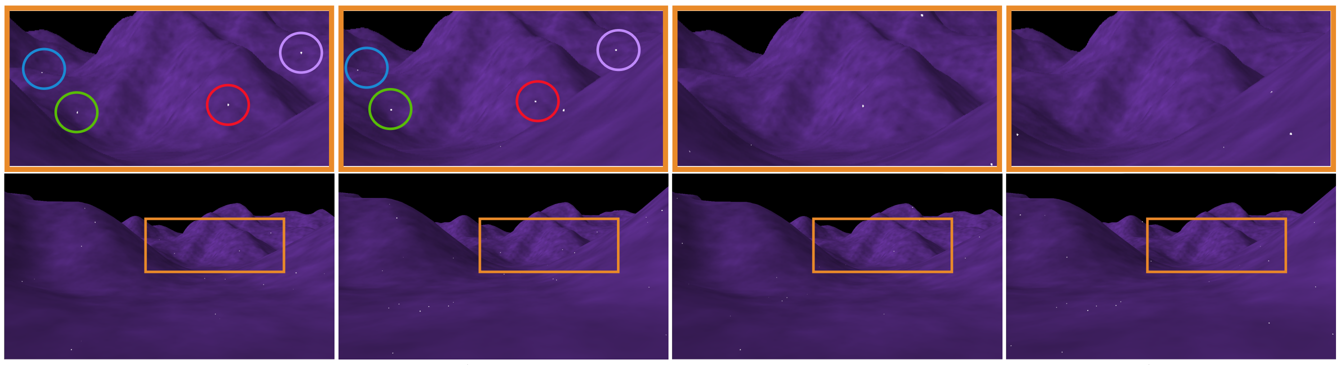
(a) Frame #n, without remapping
(b) Frame #$n + x$, without remapping
(c) Frame #n, with remapping
(d) Frame #$n + x$, with remapping

Figure 8: *Comparison of results without/with view vector remapping. The circles with the same color represent the same sparkle.*

Figure 10 demonstrates how jittering only the sparkle centers avoids distorted sparkle shapes which are unattractive and prone to aliasing.

Our sparkle size is independent of the distance to the surface, which avoids sparkles which are too small for the pixel resolution and exhibit aliasing as a result. Figure 7 (green) compares sparkles in the background, and the accompanying video material compares the approaches under motion.

In Figure 12, the sparkle effect is used on shiny surfaces. The zoomed views show the size of the sparkle can be in the order of 1 pixel.

Performance and Timings. Our approach shipped on a wide range of platforms including the last genereration consoles *Xbox 360* and *Playstation 3*. The latter proved to be the most challenging performance-wise. We measured an additional cost of 2.4ms for a full screen view of a sparkle material at 1280×720 with sparkles enabled. On modern hardware the overhead is minimal - on a GeForce GTX 980 card we could perform the computation 6400 times per pixel at 1920×1080 resolution in 30ms, averaging 0.00469ms per computation.

Limitations. As discussed, we are stepping the sparkle grid scale, and performing domain repetition on the view vector, both of which create discontinuities. We found however that these were masked by the sparkling effect and not objectionable in practice.

6. Conclusion

We present a robust procedural sparkle effect suitable for snow or other sparkly surfaces. Compared to previous methods our approach is effective over the entire depth range, and our artists could push the sparkle size down to 1-2 pixels without flickering or other aliasing artifacts. Our method is purely shader-based and can be easily integrated into the lighting pass of existing renderers. We found this effect to be very suitable for production due to its flexibility and strong performance even on last generation hardware.

7. Acknowledgments

We thank shadertoy community [bea] and Rendermonkey for knowledge and inspiration, as well as the amazing team that worked on the game that we cannot name. We thank Tom Williams, Jim Callin and Kenny Mitchell for the conversations and support.

References

[ATI] ATI/3DLABS: Rendermonkey. http://developer.amd.com/tools-and-sdks/archive/legacy-cpu-gpu-tools/rendermonkey-toolsuite/.

[bea] BEAUTYPI: Shadertoy. http://www.shadertoy.com/.

[Edw12] EDWARDS J.: Dynamic sand simulation and rendering in journey. ACM SIGGRAPH 2012 Course, 2012.

[Hec89] HECKBERT P. S.: *Fundamentals of Texture Mapping and Image Warping*. Tech. rep., 1989.

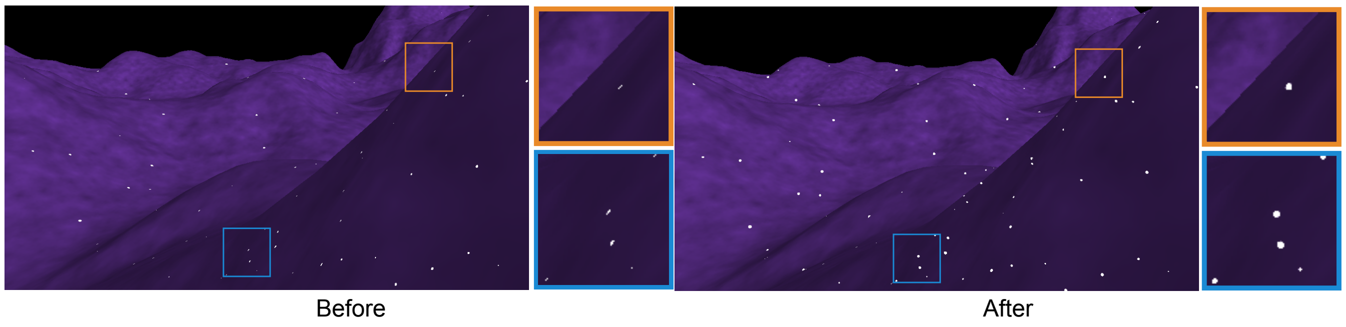

Figure 9: *Our anisotropic sparkle shape (bottom) eliminates aliasing from sharp slithers at grazing view angles.*

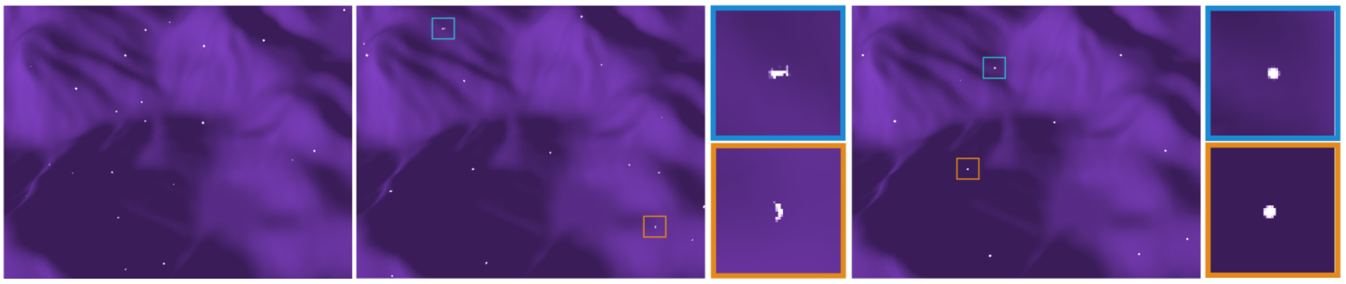

Figure 10: *Noise comparisons. Left: No noise, regular nature of grid visible. Middle: Noise applied to grid lookups, distorting sparkle shape. Right: We propose to apply noise to sparkle center only to obtain clean and alias-free results.*

[JHY*14] JAKOB W., HAŠAN M., YAN L.-Q., LAWRENCE J., RAMAMOORTHI R., MARSCHNER S.: Discrete stochastic microfacet models. *ACM Trans. Graph. 33*, 4 (July 2014), 115:1–115:10.

[OTI03] OAT C., TATARCHUK N., ISIDORO J.: Layered car paint shader. http://www.chrisoat.com/papers/Oat-Tatarchuk-Isidoro-Layered _Car _Paint _Shader _Print.pdf, 2003.

[Sho12] SHOPF J.: Gettin' procedural. AMD Tech. Article, 2012.

[YHJ*14] YAN L.-Q., HAÅAN M., JAKOB W., LAWRENCE J., MARSCHNER S., RAMAMOORTHI R.: Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Trans. Graph. 33*, 4 (2014).

[ZK16] ZIRR T., KAPLANYAN A. S.: Real-time rendering of procedural multiscale materials. In *Proceedings of I3D 2016* (2016), I3D '16, pp. 139–148.

Appendix A: Sample Shader Code

```
// Inputs
float sparkle_size, sparkle_dens;
vec3 world_pos;
vec3 view;
float view_dep;
float noise_amt, noise_dens;
sampler3D noise_tex;

// Compute grid level (scale)
float z = length(view);
float e = floor(log2(0.3*z+3.0)/0.3785116);
float level_z = 0.1*pow(1.3,e)-0.2;
float level = 0.12 / level_z;
sparkle_dens *= level;
noise_dens *= level;

// Remapping the view vector
vec3 v = view / z;
vec3 view_new = v*level_z;
view_new = sign(view_new)*fract(abs(view_new));

// Grid lookup position
vec3 pos = sparkle_dens*world_pos
         + view_dep*normalize(view_new);

// Generate the grid
vec3 g_index = floor(pos);
vec3 P_c = g_index/sparkle_dens;

// Compute offset to jittered grid center
vec3 noise = noise_amt*tex3D(noise_tex,noise_dens*P_c);
vec3 offset = vec3(0.75,0.75,0.75);
vec3 P_x = pos-g_index+0.5*fract(noise+0.5)-offset;

// Anisotropy
float dotvn = dot(v,n);
vec3 ma = v - dotvn*n; // Ellipse major axis
vec3 P_x_proj = dot(P_x,ma)*ma;
P_x += (abs(dotvn)-1.0)*P_x_proj/dot(ma,ma);

// Compute sparkle kernel
float dist2 = dot(P_x, P_x);
float thresh = 1.0 - sparkle_size;
```

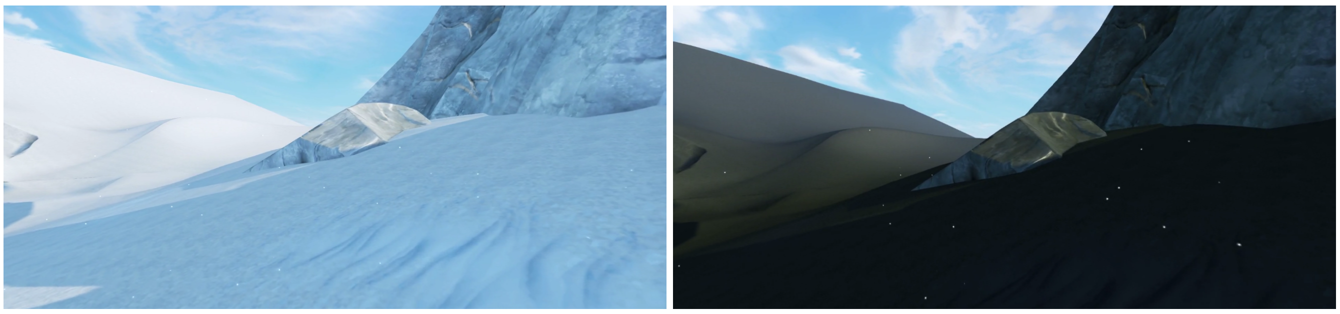

Figure 11: *Sparkle Effect in snow scene from our game. The right image has diffuse knocked out to better show the effect.*

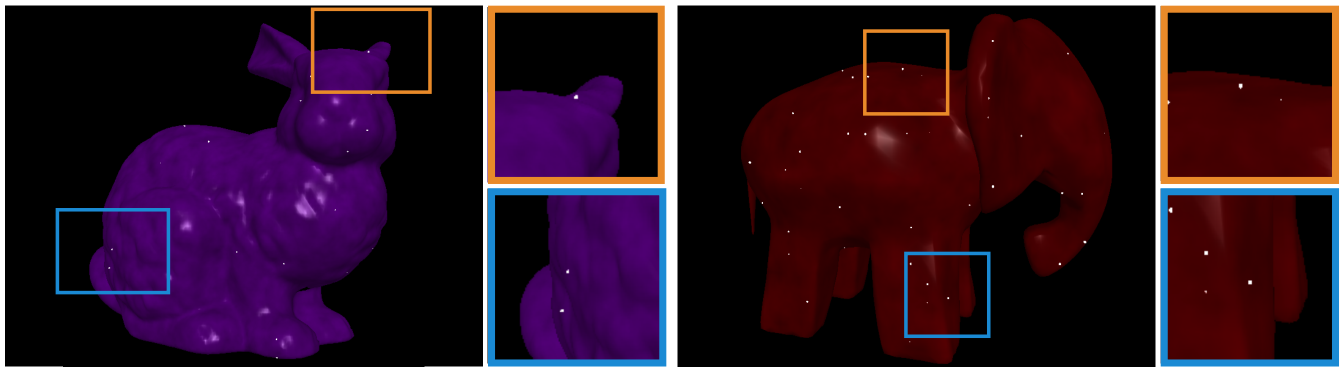

Figure 12: *Sparkle effect on shiny animal surfaces.*

```
return  dist2>thresh  ?  0.0  :  1.0-dist2/thresh;
```

Code 2: *Sparkle effect fragment code.*