Exaggerating Highlights of Small Scratches

April 23, 2024

Tom Sarry
tom.sarry@umontreal.ca
DIRO, Université de Montréal

Abstract

Adding surface imperfections in a scene can help increase its realism. Scratches, for instance, are generally extremely thin and can be modeled as a perturbation of the normal along an otherwise smooth surface. Such imperfections are of interest because of the anisotropic highlights they exhibit. Rendering detailed scratches through a high-resolution texture map, for example, can have substantial memory requirements while failing to ensure that those tiny details will be visible on the final image. Below are models that allow users to provide very detailed scratch textures, no matter how thin, and exaggerate the highlights of these imperfections.

1 Introduction

A common issue in graphic rendering is that images created can feel *too perfect*. One of the reasons is that we are used to seeing many imperfections on our day-to-day objects like weathering, scratches, or dust, to only name a few. It is therefore interesting to study how to create more realistic images by adding worn-down effects on materials.

In this report, we will focus on a limited number of thin scratches and will model them as a perturbation of the surface normal, which we will store in a normal map. Although better for time complexity, the use of a normal map can lead to an explosion of memory requirements when representing details with a high-resolution texture, with no guarantee that any details will be visible on the final rendered image. Indeed, very thin scratches will cover only a small number of texels, and consequently, rays are not likely to hit them when rendering (see Figure 1).

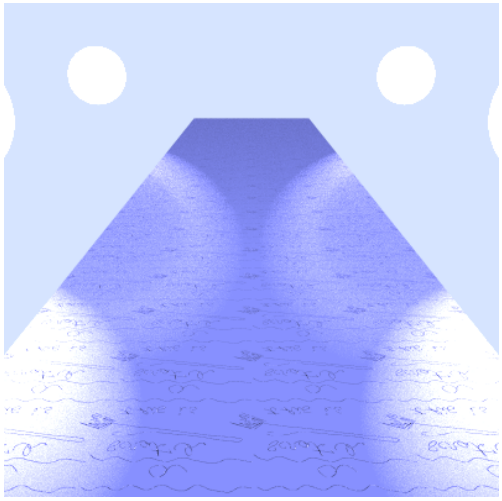


Figure 1: Our perspective scene, with thin scratches on a specular surface and multiple light sources. Notice how scratches become increasingly harder to distinguish the further we look.

We are therefore interested in a model that would allow for arbitrarily thin scratches to be created through normal maps, while also being able to show the anisotropic highlights that such imperfections exhibit. We present in this report multiple models that try to fill these requirements.

The following report will be organized as follows: firstly, we will discuss some of the related work in scratch and glint rendering. We will then present our model, along with its performance and limitations in Section 3, Section 4, and Section 5, respectively. Finally, a conclusion and discussion about the technique will be offered in Section 6.

2 Related Work

Multiple ways can be used to store information about scratches. Historically, the first option was to explicitly store them as a texture for instance [1]–[5]. This method generally allows for only a small computation overhead and allows for filtering [6], [7], while also giving a lot of freedom for artists to fine-tune the texture's details.

Perhaps the first major contribution to the space was a rule-based method to generate surface imperfections (splotches and scratches) using fractal noise [1]. This was also complimented with a Natural Language interface, giving the ability to easily produce objects with imperfections, without any artistic skill required. Instead of using a noise function, a model where a random distribution of scratches was precomputed on an image texture was studied [2], which was used to determine possible highlights on a transparent surface. Additionally, since light interactions on a scratch can be extremely varied, the idea of using image textures to store scratches was extended using *scratch maps* [3], to store scratch information. The texture was then used to know when to apply a *regular* BRDF or the *scratch* BRDF, measured on real scratched objects by the authors.

However, as textures increase in detail, their memory requirements can grow extremely fast. Another option is to only store parametric information about the scratches, like control points for instance [8]–[11]. In addition, cross sections of scratches are generally used, to accurately compute light scattering without the need to store complex micro-geometry. Although more computationally expensive, parametric scratch models have also been derived for real-time rendering [9].

Following the idea of approximating the micro-geometry of scratches, multiple models have been built on microfacets' theory and use a BRDF determined by a Normal Distribution Function [12] to encompass the complex light interaction over the complex geometry [13], [4]. While these models offer realistic results, most rely on a highly repetitive scratch structure, and can therefore be limited in the variety of imperfections displayed. However, this is not the case for all of them, as some use high-resolution normal maps like us, but with Gaussian-based pixel footprint kernel [4].

Although more sparse, hybrid models in between manual texture synthesis and physical simulation have also been researched in the past [14]. However, with the surge of high-quality textures online and interactive texture creation in 3D modeling software (*Blender* for instance), the interest in frameworks to generate textures has mostly diminished.

Rendering glints of scratched materials has been studied extensively, but is still an open problem. Because of their microscopic geometry, scratches can exhibit iridescent highlights that can be rendered using wave optical models [11], [15], even in real-time [16]. Other complex effects, including the Fresnel factor and the inter-reflection inside scratches, have also been studied [13].

In this report, we present multiple exaggeration techniques that can be used with high-resolution normal maps, similar to the work in [4], but while investigating other kernel options and integrating importance sampling

concerning distance and orientation of the normals.

3 Work

Firstly, let us define the terms and abbreviations used below.

- **Pixel Footprint**: Area covered by the projection of a pixel onto a surface.
- **Fixed-Size Kernel**: Filter kernel that will always have the same size and shape during a render.
- **Size-Varying Kernel**: Filter kernel that may have a different shape and size depending on the pixel footprint, taking into account the orientation and distance of the material from the viewpoint.

3.1 Fixed-Size Square Kernel

3.1.1 Random Selection of Normal in Neighborhood

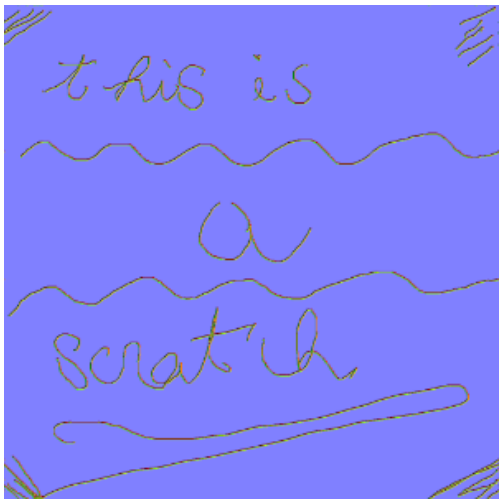


Figure 2: 1024^2 normal map used in our scene, scaled down 4 times and repeating on the material, effectively creating a 4K texture.

As our scratch information is stored in a texture (normal map, see Figure 2), it is easy to explore the neighborhood of a texel to gather additional information (*ie.* if there is a scratch close to it). We describe in this section a model to exaggerate details using a kernel of fixed size.

During ray tracing, if a ray does not hit an *interesting* texel (one that belongs to a scratch), we will look in the neighborhood for all interesting texels and choose one of them instead. The neighborhood in this case is a square of n texels around the original texel.

Figure 4 shows render results for multiple kernel sizes. Notice how details get increasingly visible, as scratches become larger when we increase the kernel size.

3.1.2 Random Selection of Closest Normal in Neighborhood

One of the issues with the previous model is the fact that the normal in the neighborhood will be chosen at random. This can lead to non-coherent highlights where we are using information that is too far away.

Instead, we would like to only consider scratch normals with the minimal distance[1] away from the original texel, and choose amongst them.

Figure 5 shows render results similar to the previous model but with this added constraint of choosing the closest scratch normal. As results only differ slightly, we offer in Figure 6 a pixel-to-pixel comparison with the previous model. Notice how the variations between the models are located on the corners of the repeating texture boundary, where the density of scratches is more important.

These models are very simple, albeit at the cost of unrealistic exaggeration of close scratches (that are extremely thin and become quite large for kernel sizes bigger than 3x3).

Indeed, the fact that the kernel size is deterministic is a problem since we will over-exaggerate details up close (by gathering information outside the pixel) and under-exaggerate

[1] Using Manhattan distance for simplicity.

them when they are far away (by not obtaining all the information contained in a pixel).

3.2 Pixel Footprint Kernel

To analyze a more representative neighborhood of texels, we chose to implement a size-varying kernel, that approximates the area of the projected pixel onto the surface.

Before rendering, we send *bounding rays* for the pixel: two rays from opposite pixel corners (in this case, the top left and bottom right ones). Then if these rays hit the same surface (without allowing for scattering, for simplicity), we can retrieve the u,v boundary coordinates of our estimated pixel footprint. Finally, we push down this information and use it to create a square neighborhood when intersecting the surface while rendering.

3.2.1 Random Selection of Scratch Normal

As a first simple model, we chose to randomly choose a perturbed normal (one that belongs to a scratch) present in the texel's neighborhood. Since we are always selecting a scratch normal, the exaggerations will be quite important, as any projected pixel that contains a scratch will have a perturbed normal. This effect is especially visible at grazing angles (as seen in Figure 3), where the pixel covers many texels and is very likely to contain at least one scratch.

However, note the difference in over-exaggeration with the previous models. Here, highlights (especially the close ones) are of higher intensity but do not make the scratch larger. This is explained by the fact that we are mostly looking at the pixel footprint, hence not leaking scratch information to the adjacent pixels.

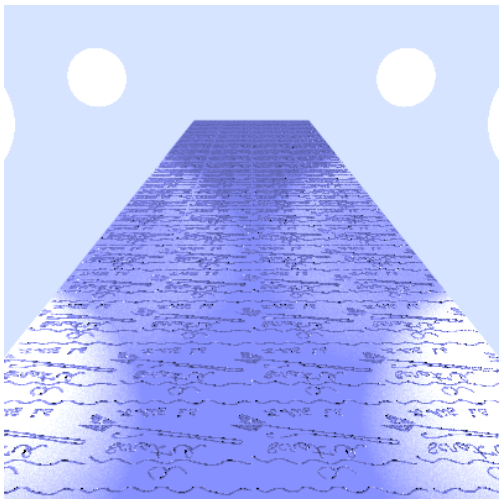


Figure 3: Rendered image made with a size-varying kernel without computing weights and always picking a scratch normal if one is present in the neighborhood.

Similarly to the improvements made to our fixed-size kernel, we would like a way to increase the importance of normals based on their distance and orientation to the original texel. In the next section, we offer such a model, that computes weights for each normal, giving us a probability of choosing it.

3.2.2 Random Selection Based on Normal Weight

During ray tracing and upon landing on our scratched material, instead of randomly picking a scratch normal in the pixel footprint, we will probabilistically choose one with the following method:

1. Collect all normals, including non-scratch ones, in the kernel for that pixel
2. Compute weights for each normal (arbitrarily defined in Eq. 1), depending on the distance from the original texel and the normal's orientation[2]
3. Select a new normal based on the probabilities defined by this discrete probability distribution

[2] As an optimization, all outward normals (not a scratch) are treated as one normal with a higher probability.

$$w = \frac{1}{d^2} + \frac{1}{(n \cdot o)^2} - (1 - \varepsilon) \qquad (1)$$

Where w is the weight of normal n, d the Manhattan distance from the original texel, o the outward normal, and $\varepsilon \geq 0$ is a user parameter to adjust the exaggeration of scratches, by modifying the importance of the orientation.

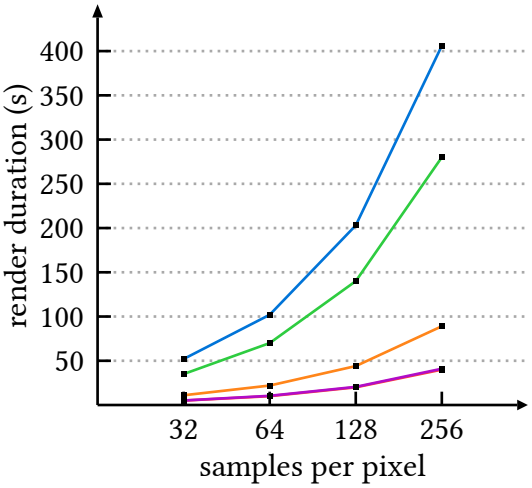
In Figure 7, we show render results for multiple values of ε. Notice the increase of details in the distance when decreasing ε, as well as the much smoother results at shorter distances compared to the previous models. We also provide in Figure 8 render results on a more specular surface with lights at different locations, to appreciate the change of highlights depending on the light direction.

Finally, the above model was improved regarding running time complexity by caching all normals and their weights for each pixel. This allowed the reuse of information between each supersampling pass, saving an important amount of time, as discussed in the following section.

4 Performance

In this section, we offer a performance analysis of both time and memory usage of the models described above[3].

Regarding time complexity (see Graph 1), our most complex model that computes a weight for every normal in the neighborhood of a texel based on the pixel footprint takes approximately 10 times the duration of a regular render. This important computation time can be reduced to 7 times if normals and their weights are cached for each pixel. However, still taking our size-varying kernel but without computing weights and dismissing any normal that does not belong to a scratch brings down the slowdown factor to only 2. Finally, having a fixed size kernel, in this case, a 3x3 square around the texel, only incurs a slowdown of around 2% of a regular render.



Graph 1: Render time of a 400x400 picture based on supersampling number for no exaggeration (red), 3x3 kernel (purple), size-varying kernel without computing weights (orange), size-varying kernel (blue), and size-varying kernel with caching (green).

Furthermore, as shown on Table 1, the memory requirements of the model are negligible (3% at most) compared to that of a regular ray tracer.

Model	**Max physical RAM (MB)**
Ground Truth	858
Square (3x3)	858
Varying (no weights)	859
Varying	884
Varying + caching	884

Table 1: Maximum physical RAM used for each rendering model. Note that the number of samples per pixel had no consequence on these values for the process.

[3]All renders were done using a CPU ray tracer on a 11th Gen Intel i7-1165G7 (8) @ 4.700GHz.

It is interesting to note that with a simple caching of normals and their weights, we can save up to 40% computing time with no real memory cost (since all similar normals will be stored as one, with the sum of their weights). Secondly, a simpler model that does not compute weights and ignores normals that do not belong to scratches performed much better than our other models, in both speed (only $2x$ slowdown) and peak RAM (negligible increase).

Finally, it is important to remind ourselves that these figures are only relevant for our scene and scratch configuration. Indeed, if our normal map with scratches had a lot more perturbations, the cost of caching would be much more important for instance.

5 Limitations & Future Work

One of the interesting additions this model could benefit from would be a more exact kernel size. Instead of taking a square with only the *uv*-coordinates of the top-left and bottom-right corners of the pixel, the four corners could be considered to create any kind of quadrilateral (which is the case normally, especially at grazing angles).

Secondly, important optimization work would have to be performed to make this model suitable for real use. Indeed, a slowdown of $\sim 10x$ ($\sim 7x$ after caching normals and weights for each pixel) was observed in our scene while exaggerating with the size-varying kernel. One idea would be to precompute and store in a hierarchical structure (quadtree or kdtree) an approximation of the scratches' location. Then, instead of collecting and weighting each normal in the kernel, an importance sampling could be performed instead, by selecting multiple normals from nearby scratches (efficiently retrieved with our hierarchical structure).

Finally, one of the simplifications done when computing the pixel footprint was to disallow any ray scattering. As a consequence, a scratched surface visible through a perfect mirror would not benefit from the exaggerations presented in this paper for instance. It would therefore be interesting to allow for a limited number of ray bounces while computing the bounding rays for each pixel. However, note that this change would largely complexify the model, as multiple kernels would have to be stored for each pixel, as a bounding ray could hit multiple surfaces with scratches.

6 Conclusion

In this paper, we presented a model to exaggerate tiny details stored in textures, such as scratches stored in normal maps in our example. Our technique relies on a neighborhood search around the texel hit while ray tracing and tries to select a nearby scratch normal if possible, considering the distance to the original texel and its orientation. Firstly, we considered a square kernel of fixed size. Although simpler to implement and faster, it failed to exaggerate details at grazing angles, where a pixel covers many texels. To solve this issue, we created a kernel of varying size depending on the projected pixel's area on the surface. With an important cost in performance, this model was able to exaggerate details even at grazing angles.

Bibliography

[1] W. Becket and N. I. Badler, "Imperfection for realistic image synthesis," *The Journal of Visualization and Computer Animation*, vol. 1, no. 1, pp. 26–32, 1990, doi: https://doi.org/10.1002/vis.4340010108.

[2] J. W. Buchanan and P. Lalonde, "An observational model for illuminating isolated scratches," 1999. [Online].

Available: https://api.semanticscholar.org/CorpusID:17068528

[3] S. Mérillou, J.-M. Dischler, and D. Ghazanfarpour, "Surface scratches: measuring, modeling and rendering," *The Visual Computer*, vol. 17, pp. 30–45, 2001.

[4] L.-Q. Yan, M. Hašan, W. Jakob, J. Lawrence, S. Marschner, and R. Ramamoorthi, "Rendering glints on high-resolution normal-mapped specular surfaces," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–9, 2014.

[5] W. Jakob, M. Hašan, L.-Q. Yan, J. Lawrence, R. Ramamoorthi, and S. Marschner, "Discrete stochastic microfacet models," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–10, 2014.

[6] M. Olano and D. Baker, "Lean mapping," in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2010, pp. 181–188.

[7] J. Dupuy, E. Heitz, J.-C. Iehl, P. Poulin, F. Neyret, and V. Ostromoukhov, "Linear efficient antialiased displacement and reflectance mapping," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, pp. 1–11, 2013.

[8] C. Bosch, X. Pueyo, S. Mérillou, and D. Ghazanfarpour, "A Physically-Based Model for Rendering Realistic Scratches," *Computer Graphics Forum*, vol. 23, pp. 361–370, 2004, doi: 10.1111/j.1467-8659.2004.00767.x.

[9] C. Bosch and G. Patow, "Real time scratches and grooves," in *XVII Congreso Espanol de Informática Gráfica (CEIG'07)*, 2007.

[10] C. Bosch, X. Pueyo, S. Mérillou, and D. Ghazanfarpour, "A Resolution Independent Approach for the Accurate Rendering of Grooved Surfaces," *Computer Graphics Forum*, vol. 27, no. 7, pp. 1937–1944, 2008, doi: https://doi.org/10.1111/j.1467-8659.2008.01342.x.

[11] S. Werner, Z. Velinov, W. Jakob, and M. B. Hullin, "Scratch iridescence: Wave-optical rendering of diffractive surface structure," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pp. 1–14, 2017.

[12] K. E. Torrance and E. M. Sparrow, "Theory for off-specular reflection from roughened surfaces," *Josa*, vol. 57, no. 9, pp. 1105–1114, 1967.

[13] B. Raymond, G. Guennebaud, and P. Barla, "Multi-scale rendering of scratched materials using a structured SV-BRDF model," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–11, 2016.

[14] T.-T. Wong, W.-Y. Ng, and P.-A. Heng, "A Geometry Dependent Texture Generation Framework for Simulating Surface Imperfections," 1997, pp. 139–150. doi: 10.1007/978-3-7091-6858-5_13.

[15] L.-Q. Yan, M. Hašan, B. Walter, S. Marschner, and R. Ramamoorthi, "Rendering specular microgeometry with wave optics," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–10, 2018.

[16] Z. Velinov, S. Werner, and M. B. Hullin, "Real-time rendering of wave-optical effects on scratched surfaces," in *Computer Graphics Forum*, 2018, pp. 123–134.

7 Appendix

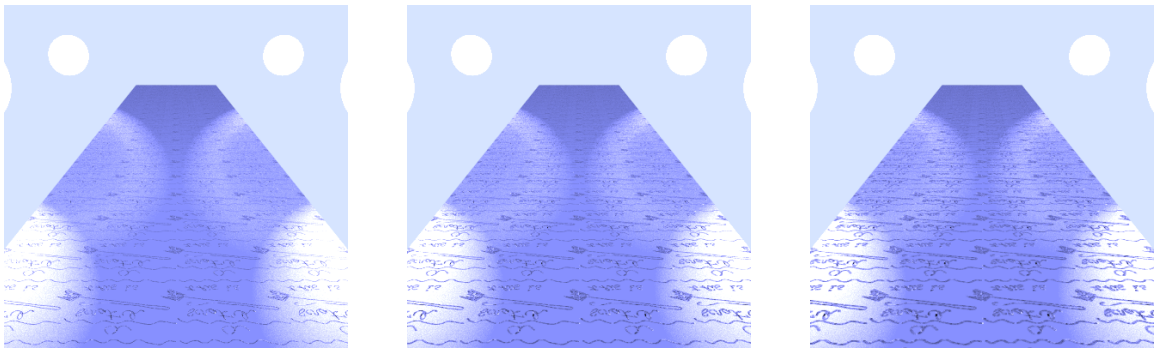


Figure 4: Comparison of renders made with a fixed-size square kernel. **Top**: 3x3. **Middle**: 5x5. **Right**: 7x7. Notice the over-exaggeration of close scratches as the kernel size increases.

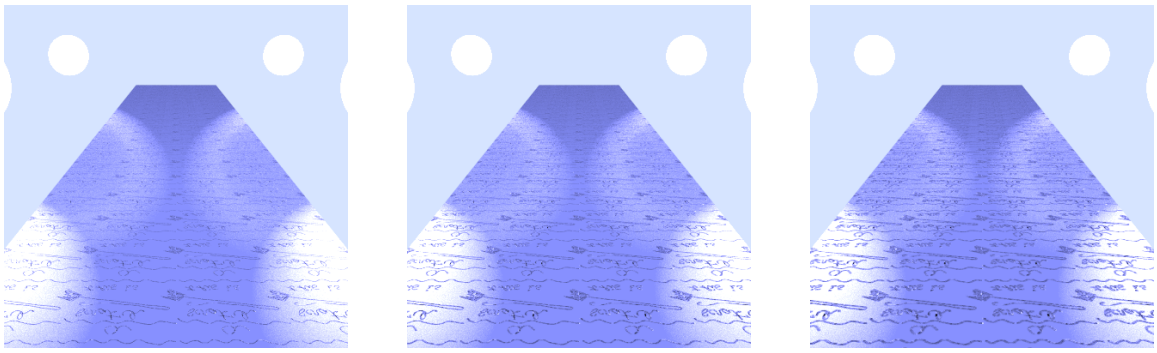


Figure 5: Comparison of renders made with a fixed-size square kernel using the closest scratch normal. **Top**: 3x3. **Middle**: 5x5. **Right**: 7x7.

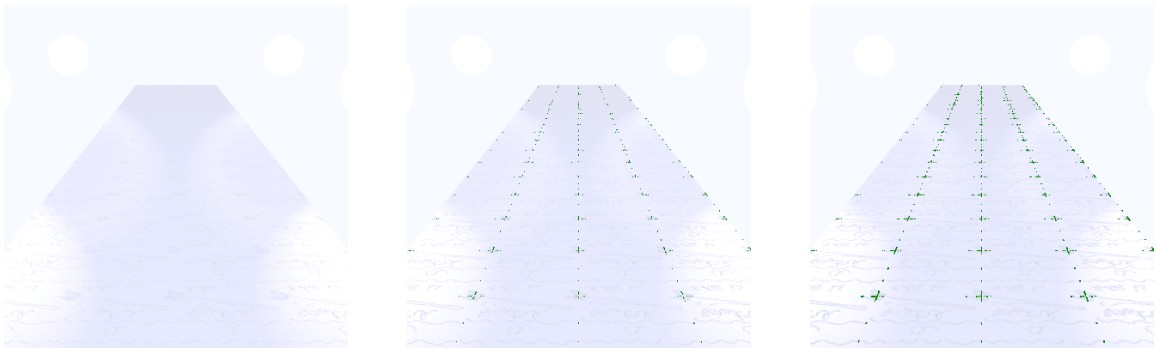


Figure 6: Pixel-to-pixel comparison (green pixels are different) of Figure 4 and Figure 5. Picking the closest normal produces slightly different highlights when using a larger kernel, as expected.

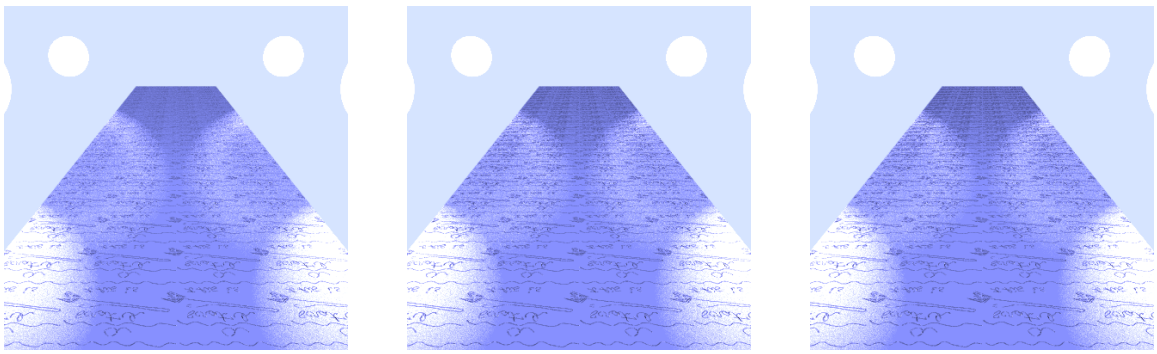


Figure 7: Comparison of renders made with the size-varying kernel and weight computations. **Top**: $\varepsilon = 0.1$. **Middle**: $\varepsilon = 0.01$. **Right**: $\varepsilon = 0.001$.

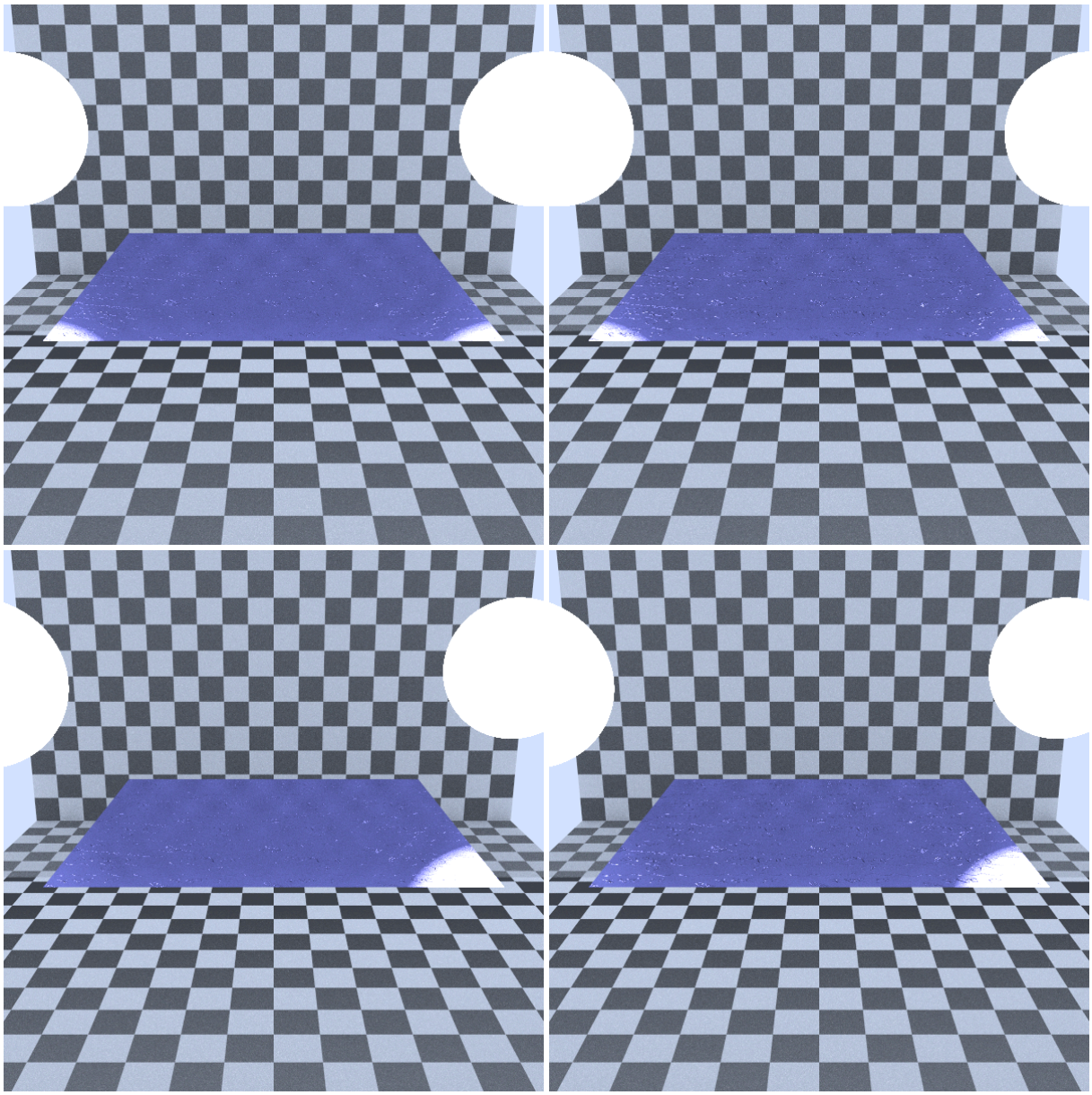


Figure 8: Comparison of renders ($\varepsilon = 1$) with light sources at different locations on a 90% specular surface between a regular ray tracer (left) and our method (right). Notice how our method follows the specularity highlights for a regular ray tracer but with higher intensity.