Real-Time Rendering of Glossy Reflections Using Ray Tracing and Two-Level Radiance Caching

Kenta Eto
Advanced Micro Devices, Inc.
Tokyo, Japan
Kenta.Eto@amd.com

Sylvain Meunier
Advanced Micro Devices, Inc.
Paris, France
Sylvain.Meunier@amd.com

Takahiro Harada
Advanced Micro Devices, Inc.
Santa Clara, United States
Takahiro.Harada@amd.com

Guillaume Boissé
Advanced Micro Devices, Inc.
Paris, France
Guillaume.Boisse@amd.com

Figure 1: Diffuse and specular global illumination estimated in 5.83 ms at 1080p on an AMD Radeon™ RX 7900 XTX GPU.

ABSTRACT

Estimation of glossy reflections remains a challenging topic for real-time renderers. Ray tracing is a robust solution for evaluating the specular lobe of a given BRDF; however, it is computationally expensive and introduces noise that requires filtering. Other solutions, such as light probe systems, offer to approximate the signal with little to no noise and better performance but tend to introduce additional bias in the form of overly blurred visuals. This paper introduces a novel approach to rendering reflections in real time that combines the radiance probes of an existing diffuse global illumination framework with denoised ray-traced reflections calculated at a low sampling rate. We will show how combining these two sources allows producing an efficient and high-quality estimation of glossy reflections that is suitable for real-time applications such as games.

CCS CONCEPTS

• **Computing methodologies** → **Ray tracing**.

KEYWORDS

rendering, real-time, ray tracing

ACM Reference Format:
Kenta Eto, Sylvain Meunier, Takahiro Harada, and Guillaume Boissé. 2023. Real-Time Rendering of Glossy Reflections Using Ray Tracing and Two-Level Radiance Caching. In *SIGGRAPH Asia 2023 Technical Communications (SA Technical Communications '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3610543.3626167

1 INTRODUCTION

Radiance and irradiance probes are widely adopted techniques that approximate diffuse and specular indirect lighting in real-time applications and engines. Such techniques propose to pre-calculate the lighting in the scene at specific locations of space and store the result in structures generally referred to as light probes [Majercik et al. 2019]. One issue that arises with such systems is that the illumination is estimated at the probe rather than the pixel location. This leads to artifacts such as leaking and over-occlusion. Such problems can be alleviated to some extent through a better probe placement, either manual [Hooker 2016] or automated [Wang et al. 2019].

Alternative placement strategies, such as screen-space radiance caching, fix most leaking and over-occlusion issues by placing the probes directly onto screen pixels [Wright 2021]. Such probes can then be used to evaluate indirect diffuse and indirect specular illumination. However, due to the low-resolution nature of their representation, these are unsuitable for evaluating reflections on smooth surfaces. Indeed, the specular lobe can be significantly smaller than the angular resolution of the probe, resulting in overly blurred visuals.

On the other hand, ray tracing offers a robust way of estimating glossy reflections. Indeed, Monte Carlo integration allows us to approximate the lighting integral better as the sample count increases. Unfortunately, this comes at a significant performance cost, forcing real-time renderers to typically use less than one sample per pixel to meet the performance target. The noisy signal needs to be upscaled and denoised before being presented to the viewer [Stachowiak 2018]. Furthermore, the rougher the surface, the more divergent the rays, resulting in poorer overall performance.

This paper proposes to combine the probe system from an existing real-time diffuse global illumination framework [Boissé et al. 2023] with reduced rate ray tracing and a dedicated denoiser for estimating glossy reflections of a scene. We will demonstrate how this approach enables evaluating the specular lobe of a BRDF in an

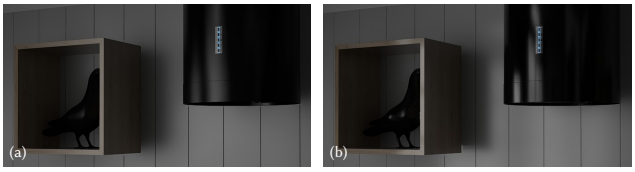

Figure 2: Blurred rendering results when using screen probes for low-roughness surfaces. (a) shows the result when using screen probes. (b) shows the result when using ray tracing.

efficient way that is suitable for use in real-time applications such as games.

2 OUR METHOD

As mentioned in Section 1, one of the methods used in our approach for rendering glossy reflections builds upon an existing probe system initially meant to estimate diffuse global illumination [Boissé et al. 2023]. As [Wright 2021], the system spawns the probes directly on the screen onto the visible pixels. We, therefore, refer to these as *screen probes*. In our implementation, each screen probe encodes the incoming radiance across the oriented hemisphere into an 8x8 storage. We leverage hardware-accelerated ray tracing to estimate the content of each of the probes and rely on a second technique for storing the outgoing radiance at hit points, which makes use of spatial hashing to maintain the caching data structure [Boissé et al. 2023]. We, therefore, refer to this data structure as *hash grid cache*, and its individual elements as *hash cells*.

In this section, we show how we can leverage both caching techniques to accelerate the rendering of our glossy reflections and our denoiser to reduce noises in estimated glossy reflections.

2.1 Glossy reflection rendering

Given the incoming radiance $L_i(\omega_i)$, we temporally accumulate the incoming radiance in each frame. $L_i^{acc}(\omega_o) = \frac{1}{n}\sum_{t=0}^{n} L_i(\omega_{i,t})$. Here, $L_i(\omega_{i,t})$ is an evaluated incoming radiance in the tth frame. We denoise the accumulated radiance $L_i^{acc}(\omega_o)$ and evaluate the lighting at the shading point by using a split-sum approximation as done in [Karis 2013]. $L_{spec}(\omega_o) = \text{BRDF LUT}(\omega_o) \times L_i^d(\omega_o)$. Here, $L_i^d(\omega_o)$ is a denoised result of temporally accumulated radiance, and BRDF LUT(ω_o) is a lookup table that stores the directional albedo of the GGX BRDF. Using split-sum approximation, we can decouple the lighting from the material evaluation, and it allows us to simplify our algorithms for evaluating incoming radiance and denoising. We explain the evaluation of incoming radiance and our denoising algorithm in Sections 2.2 and 2.4, respectively.

2.2 Radiance evaluation

To evaluate the incoming radiance, we generate a random direction ω_i by importance sampling the GGX lobe of the BRDF at the shading point using [Heitz 2018]. We then evaluate the incoming radiance using screen probes or ray tracing. To clarify, we call radiance evaluation using screen probes by specular interpolation and we call radiance evaluation using ray tracing by ray-traced reflection.

Direction sampling can introduce significant variance when the roughness at the shading point is very high. To remove this variance, we fall back to the diffuse rendering when the roughness at the shading point is higher than the threshold $r_{diffuse}$.

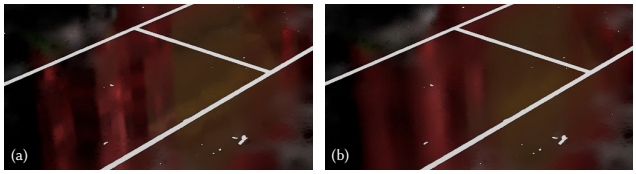

Figure 3: Comparison of jittering the hit point when accessing the hash grid cache. (a) shows the result of no jittering. There are some grid-like artifacts. (b) shows the result of jittering. Jittering can reduce grid-like artifacts.

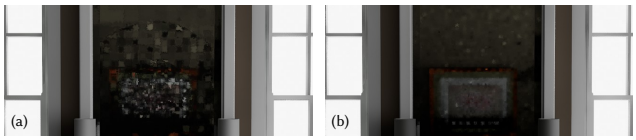

Figure 4: Comparison of placement strategy of hash grid cells. (a) shows the rendering without considering the BRDF at the shading point. The failure to fetch radiance from the hash grid cell creates grid-like black artifacts. (b) shows the rendering considering the BRDF at the shading point. Considering the BRDF can reduce black artifacts.

2.2.1 Specular interpolation. Given the direction sample ω_i, we fetch the radiance value from the corresponding texel in the 8x8 storage of screen probes. Since the resolution of the screen probes is limited, we cannot use them for rendering reflections with a low roughness value. Indeed, this can create blurred rendering results (Figure 2). To avoid this, we use screen probes for the surfaces with roughness higher than the specified threshold r_{rt}. When the roughness is below r_{rt}, we fall back to ray-traced reflection.

2.2.2 Ray-traced reflection. Given the direction sample ω_i, we ray-trace along ω_i and get a hit point. To achieve higher rendering performance, we do ray-traced reflection in half resolution, and the rendering will be 2x-upscaled later. For the details of upscaling, see Section 2.4. When the hit point is visible in the previous frame, we use the pixel color of the previous frame as an incoming radiance at the shading point. The pixel color from the previous frame can give us a higher-quality result than the hash grid cache since the resolution of the hash grid cache is limited. Also, this can simulate multi-bounce lighting. When the previous frame is unavailable, we fetch the radiance from the area light or hash grid cache at the hit point. We jitter the hit point when fetching the radiance from the hash grid cache to reduce grid-like artifacts. See Figure 3 for the comparison. When the hit point is in the sky, we fetch the incoming radiance from the environment map.

2.3 Hash grid cell placement

The placement of hash grid cells is important for rendering glossy reflections correctly. In ray-traced reflection, if there is no hash grid cell at the hit point, we fail to fetch the incoming radiance, and the rendering will be black (Figure 4a).

To avoid this, we place hash grid cells according to the BRDF at the shading point visible from a camera (Figure 4b). We first stochastically choose the diffuse or specular BRDF by the weight of each BRDF. If the diffuse BRDF is selected, we do ray guiding the outgoing direction by screen probes [Boissé et al. 2023]. If specular

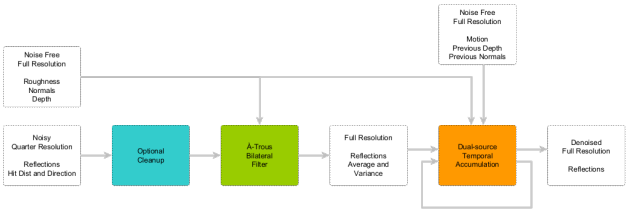

Figure 5: White boxes describe inputs and outputs. The À-Trous bilateral filter is computed over several frames using temporal accumulation.

BRDF is selected, we importance-sample the outgoing direction from GGX BRDF [Heitz 2018].

After sampling the outgoing directions, we ray-trace along the outgoing direction and allocate a hash grid cell at the hit point. We do this process every frame to handle dynamic lighting.

2.4 Denoising and upscaling

Our denoising pipeline can be divided into three components: an À-Trous bilateral filter [Dammertz et al. 2010], a dual-source temporal accumulation step [Stachowiak 2018] and a cleanup pass, as shown in Figure 5.

2.4.1 À-Trous bilateral filter. We propose to denoise and upscale reflections using bilateral filtering. Our bilateral filter is based on a variation of the À-Trous transform using an edge-stopping weight function w_t tailored for reflections. In previous works [Dammertz et al. 2010; Schied et al. 2017], the À-Trous transform is a discrete transform operating on pixel grids, and upscaling is not considered. We modify the À-Trous transform to use input samples from randomized and/or jittered positions, which helps with upscaling where jittering is commonly used for filtering subpixel details. Let us assume R_t and w_t are continuous functions, $t \in [0, k-1]$ being the iteration index, we define the À-Trous transform as:

$$R_{t+1}(x,y) = \frac{\int_{u,v} w_t(x,y,u,v) R_t(x+u, y+v) du dv}{\int_{u,v} w_t(x,y,u,v) du dv}$$

R_0 represents our initial noisy reflections rendered at a low sampling rate. After k iterations, the continuous function R_k is sampled at upscaled pixel centers so that we obtain a denoised reconstruction of reflections at a high resolution.

In previous works, weight functions rely on a discrete spline kernel h_t with holes. We propose to replace h_0 with a truncated Gaussian g_0 and h_1 to h_k by a sum of truncated Gaussians g_1 to g_k as illustrated in Figure 6. w_t is tailored to preserve reflection details as shown in Figure 7. As most reflection details come from the variation of roughness and normals across the surface, we define w_t as the product of the reflection sampling PDF p, a scaled and saturated distance d parameter between the tangent plane at the filtered and the sample position and our hole function g_t. Scale g_0 is an important parameter that controls performance and quality. In practice, we use a 5x5 pixel footprint at an upscaled resolution. The second trade-off between performance and quality is the number of filtering iterations k. We found $k = 4$ to work well on all our test scenes. See Figure 8 for the comparison. Finally, we observed

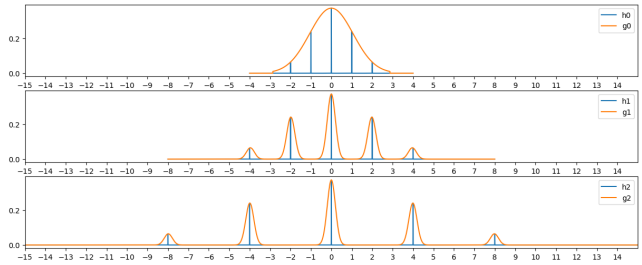

Figure 6: The kernels proposed in this paper. We replace the common à-trous spline kernels (blue) with sums of truncated Gaussians (orange). Gaussians are chosen to match the à-trous transform kernel. Gaussian scales can be increased for first frames so no upscaled pixel remains black.

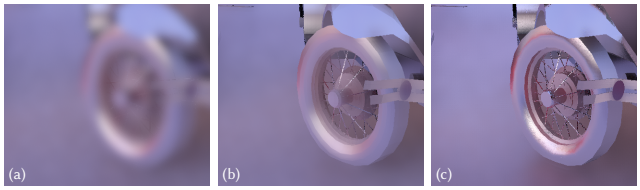

Figure 7: Comparison of the choice of w_t. (a) $w_t = g_t$. Everything is blurred in the range. (b) $w_t = g_t \cdot d$. Object edges are preserved. (c) $w_t = g_t \cdot d \cdot p$. Object edges and roughness are preserved.

Figure 8: (a) uses 2 iterations, shiniest surfaces aren't stable (ground and heater body). (b) uses 3 iterations, shiniest surfaces are stable for others not (heater base). (c) uses 4 iterations, most surfaces are stable including the matte wall.

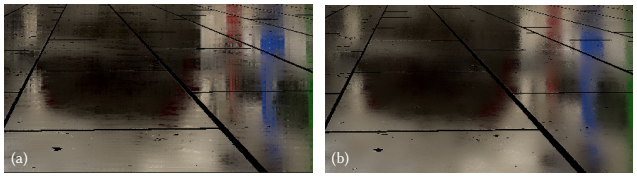

Figure 9: When roughness details are smaller than the À-Trous hole size, grid patterns can appear (a). Using jittering and temporal accumulation for these areas remove this issue, trading artifacts for additional blurriness (b). When roughness is varying slowly, jittering isn't used and no additional blur affects the roughness appearance.

that using the upscaled resolution during the last iteration and a lower resolution for the previous ones presented little quality loss and better performance. A choice of w_t can create grid patterns, as shown in Figure 9. These patterns are visible when the details created by the roughness and normal maps are roughly the hole size. We mitigate this issue by jittering sample positions proportional to

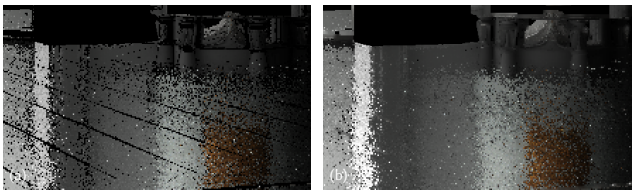

Figure 10: (a) are input noisy reflections, black reflections are missing or just dark reflections. (b) is after cleanup, missing and darker reflections are replaced by filtering valid surrounding value (noise is lower at the heater base, some dark details are moved).

(a) Breakfast Room **(b)** Kitchen **(c)** Salle de bain **(d)** Sponza

Figure 11: Scenes used for creating performance results

this size. In practice, we enable this during the last iteration and limit its use to surfaces with highly varying roughness.

2.4.2 Dual-source temporal accumulation. The À-Trous bilateral filter is computed over several frames using temporal accumulation [Yang et al. 2020]. In practice, we upscale reflections with a constant scale factor of 2, and samples are placed following a Z-curve cycle between upscaled pixel centers. This method covers all upscaled pixels with the same number of samples over time. Reflections do not have proper motion vectors available for tracking them, so we rely on a dual-source reprojection method [Stachowiak 2018]. This method considers two reprojection points, one on the surfaces and the other at the hit positions, and outputs a color by blending both reprojected colors using a distance based on the brightness difference between the current color and the reprojected ones. It also mitigates ghosting by clamping reprojected colors according to the current color neighborhood. The reflection average and variance are provided by the À-Trous bilateral filter.

2.4.3 Cleanup pass. Fireflies and missing reflections are common, so we mitigate noise by replacing them before denoising and upscaling reflections. Cleanup is important for using the dual-source temporal reprojection, which needs a good enough estimation of the reflection neighborhood, or clamping reprojected colors will introduce noise and/or aliasing as shown in Figure 10. Missing reflections are marked using input flags we compute during reflection estimation described in Section 2.2. Inspired by [Mara et al. 2017], fireflies are marked by counting darker and brighter samples in a neighborhood and testing if there is enough brighter neighbors. Then marked samples are replaced by filtering non-marked neighbors using a 3x3 bilateral filter. If all neighbors are marked, we just use a pass-through. This is important for handling near constant areas where the counting method does not work and there are no fireflies.

3 RESULT

Here we show the performance of our method measured at 1080p in AMD Radeon™ RX 7900 XTX GPU. We used $r_{rt} = 0.2, r_{diffuse} =$ 0.6 for the roughness threshold. Figure 11 shows scenes used for measuring the performance. Table 1 shows the performance result.

Table 1: Performance results (in ms) at 1080p.

	Fig 11a	Fig 11b	Fig 11c	Fig 11d
Diffuse/Specular interpolation	0.31	0.31	0.37	0.33
Ray-traced reflection	0.28	0.26	0.18	0.18
Denoising&Upscaling	0.60	0.85	0.96	0.46
GI-1.0 [Boissé et al. 2023]	3.09	3.66	3.10	3.14
Total	4.28	5.08	4.61	4.11

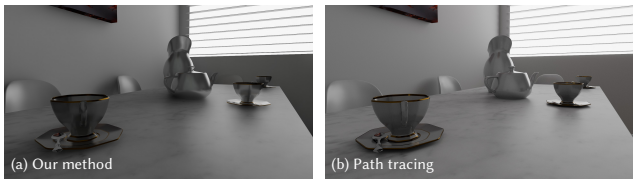

Figure 12: Comparison of our method with path tracing (3-bounce).

As shown in Figure 1, our method can render glossy reflections for surfaces with varying roughness. Table 1 shows our method works around 5 ms in total, and the overhead of the reflection is around 1 ms. This is enough for real-time rendering purposes.

Figure 12 compares our method with path tracing. Our method cannot handle multi-bounce lighting precisely since the hash grid cache only contains the direct lighting at the hit point. Due to this, our method gives a darker result than that of path tracing. In future work, we want to handle multi-bounce lighting more accurately.

ACKNOWLEDGMENTS

Bedroom, Fireplace Room, Breakfast Room, Kitchen, Salle de bain, and Sponza were created by Evermotion, Wig42, Nacimus Ait Cherif, and Frank Meinl, Crytek, respectively. We would like to thank the members of Advanced Rendering Reseach team for their support and discussions.

REFERENCES

Guillaume Boissé, Sylvain Meunier, Heloise de Dinechin, Pieterjan Bartels, Alexander Veselov, Kenta Eto, and Takahiro Harada. 2023. GI-1.0: A Fast Scalable Two-Level Radiance Caching Scheme for Real-Time Global Illumination.

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik P. A. Lensch. 2010. Edge-Avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering. In *HPG '10*. 67–75.

Eric Heitz. 2018. Sampling the GGX Distribution of Visible Normals. *Journal of Computer Graphics Techniques (JCGT)* 7, 4 (30 November 2018), 1–13.

JT Hooker. 2016. Volumetric Global Illumination At Treyarch.

Brian Karis. 2013. Real Shading in Unreal Engine 4.

Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. 2019. Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields. *Journal of Computer Graphics Techniques (JCGT)* 8, 2 (5 June 2019), 1–30.

Michael Mara, Morgan McGuire, Benedikt Bitterli, and Wojciech Jarosz. 2017. An Efficient Denoising Algorithm for Global Illumination. In *HPG '17*. Article 3, 7 pages.

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. In *HPG '17*. Article 2, 12 pages.

Tomasz Stachowiak. 2018. Stochastic All The Things: Raytracing in Hybrid Real-Time Rendering.

Yue Wang, Soufiane Khiat, Paul G. Kry, and Derek Nowrouzezahrai. 2019. Fast Non-Uniform Radiance Probe Placement and Tracing. In *I3D '19*. Article 5, 9 pages.

Daniel Wright. 2021. Radiance Caching for Real-Time Global Illumination.

Lei Yang, Shiqiu Liu, and Marco Salvi. 2020. A Survey of Temporal Antialiasing Techniques. *Computer Graphics Forum* 39, 2 (2020), 607–621.