EG UK Computer Graphics & Visual Computing (2021)
K. Xu and M. Turner (Editors)

Improving Ray Tracing Performance with Variable Rate Shading

A. Dahlin[†1] and V. Sundstedt[1]

[1]Blekinge Institute of Technology, Sweden

Abstract

Hardware-accelerated ray tracing has enabled ray traced reflections for real-time applications such as games. However, the number of traced rays during each frame must be kept low to achieve expected frame rates. Therefore, techniques such as rendering the reflections at quarter resolution are used to limit the number of rays. The recent hardware features inline ray tracing, and variable rate shading (VRS) could be combined to limit the number of rays even further. This research aims to use hardware VRS to limit the number of rays while maintaining the visual quality in the final rendered image. An experiment with performance tests is performed on a rendering pipeline using different techniques to generate rays. The techniques use inline ray tracing combined with VRS and ray generation shaders. These are compared and evaluated using performance tests and the image evaluator ꟻLIP. The results show that limiting the number of rays with hardware VRS leads to improved performance while the difference in visual quality remains comparable.

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing;***

1. Introduction

Real-time ray tracing has been a long-term goal within computer graphics [Shi05, KKW*13, KVBB*19], and hardware-accelerated ray tracing is one step closer towards this goal. Current games use hardware-accelerated ray tracing for specific effects, for example reflections [DS19]. The result is more realistic images than current techniques [Sta18, Gro20], such as *screen-space reflections* (SSR) that fails when information is not available within screen-space. However, hardware-accelerated ray tracing is still expensive in terms of time and requires low ray counts on current consumer hardware. This requires aggressive denoising algorithms [Sta18, DS19] since low ray counts result in a noisy image, but the frame rates are still not up to the standard that many users expect.

This paper looks at the performance characteristics of using *inline ray tracing* combined with VRS to limit the number of rays when rendering reflections. Two methods of determining shading rates are tested. One using the average surface roughness and the other on the average number of grazing angles within a screen area. The performance and visual quality are compared against the existing solution that trace rays at quarter resolution [Sta18], using ray generation shaders. Here visual quality is a measurement of perceivable noise in the final images and the reflections. The performance and visual quality of the techniques are evaluated using performance tests and ꟻLIP [ANAM*20]. ꟻLIP is a tool that calculates the perceived difference between two images based on human perception and viewing conditions [ANAM*20]. This paper is limited to single bounce ray traced reflections in static scenes.

2. Background

Hardware-accelerated ray tracing is available through the DirectX Ray Tracing (DXR) API [HAM19, Mic21]. The first version of DXR uses ray generation shaders to trace rays through an acceleration structure. Closest-hit and miss shaders are used to gather the results from the ray cast [HAM19, Mic21]. Inline ray tracing enables ray tracing in all shader types, which gives more control to the developer [Mic21].

VRS is a recent technique that is available in DirectX 12 [Mic19a, Mic19b, AMD20]. VRS controls the pixel shader invocations, and the result from one invocation can be stored in multiple texels. Fewer innvocations saves time when not all parts of the image need to be shaded with the same amount of detail. D3D12 divide VRS into tier 1 and tier 2, depending on hardware-support [Mic19a, Mic19b]. Tier 1 enables control of the shading rate for all fragments at once [Mic19b]. Tier 2 has the same capabilities, but the shading rate can also be controlled with an image and per primitive.

3. Related Work

There are various methods to minimize the time spent on shading. One solution is to trace rays at quarter resolution [Sta18, HAM19].

† alexander.dahlins@gmail.com

DOI: 10.2312/cgvc.20211319

The quarter resolution image gets reconstructed to a full resolution image in a subsequent spatial filter using neighboring pixels' hits, resulting in more samples [Sta18, HAM19]. Backprojection is also used to increase the number of samples. The last noise is reduced with a final blur based on the samples' variance.

Variable Rate Tracing (VRT) divides the screen into tiles, where each tile gets a certain number of rays in each frame. The number of rays are allocated so that grazing angles and specific surfaces, such as water, receive a higher ray count [DS19].

Software-based VRS techniques enable VRS on hardware that does not support it natively, which also enables smaller tile sizes [Dro20]. The shading rate is based on the minimum RGB value and luma. Hardware VRS has also been used successfully. One solution is to determine the shading rate with an edge detection filter based on the image's luminance. Both techniques resulted in performance improvements around 15% [Dro20, VR21].

4. Method

The techniques in this paper are implemented in a custom framework using the D3D12 API. Similar to previous work, a deferred renderer is used, removing the need to trace camera rays [Sta18]. The deferred renderer stores albedo color, world normals, material information, geometric world normals, and velocity. The images from the techniques are then compared to a version rendered in full resolution.

4.1. Variable Rate Shading

The shading rate is based on an image, requiring VRS tier 2. Each texel in the image corresponds to a tile, where each tile corresponds to an area of 16x16 pixels when rendering. The VRS image gets generated after the geometry pass, using a compute shader. Each thread calculates an average heuristic value of the tile, which determines the shading rate. The heuristic is either the average roughness or the average number of grazing angles in the tile. Table 1 shows the shading rates with corresponding thresholds. High roughness and geometric normal vectors parallel to the view direction use a lower shading rate, in this case, 2x2. Low roughness and vectors close to perpendicular to the surface use the shading rate 1x1.

Roughness	Avg. Dot Product	Shading Rate
≥ 0.75	≤ 1.0	2x2
≥ 0.25	≤ 0.75	2x1
≥ 0.0	≤ 0.5	1x1

Table 1: *Thresholds of the average roughness and dot product (Between the surface normal and the camera view direction) that correspond to a certain shading rate.*

4.1.1. Ray Traced Reflections

The reflection rays get traced at quarter resolution, and each output pixel corresponds to a 2x2 quad of the full resolution. Each frame one pixel per quad is used to trace a ray. The pixel changes between frames and is selected using blue noise [Uli88, GF16]. The scene's depth, rendered in previous passes, is used to calculate the world-space position, used as the ray origin. A small offset in the surface normal direction is applied to avoid self-intersections. Importance sampling, based on the normal distribution function, similar to previous work [Kar13], is used to calculate the ray direction. The importance sampling function uses a Halton-sequence with a Cranley-Patterson rotation to get a unique low-discrepancy sequence for each pixel [CP76, KK02]. When tracing the ray, all primitives are included, and back-face culling is turned on. From each ray, the hit color, the direction, and the inverse *probability distribution function* (PDF) are stored. The hit color comes from either the closest-hit or miss shader and corresponds to the direct lighting contribution or the skybox. The closest-hit shader calculates texture coordinates, position, normal, and tangent vectors by interpolating between the vertices of the triangle that got hit. The shader also gathers material information. Since the closest-hit shader does not have access to partial derivatives as the pixel shader does, a fixed *mip-level* has to be used. Mip-level two was chosen since it seemed to improve the quality of the reflections over mip-level zero. The miss shader samples the skybox texture with the ray direction. The inline ray tracing implementation uses a pixel shader required to utilize VRS. The vertex shader generates a full-screen triangle to generate invocations for the whole screen. The pixel shader is very similar to the ray generation shader. However, ray queries are used to trace the rays. The result from the trace causes a branch that either executes code for a hit or a miss. If the trace results in a hit, a very similar code block to the closest-hit shader is executed. Otherwise, the shader samples the skybox texture with the ray direction. The results are in the end written to render targets.

4.1.2. Reconstruction

The reconstruction pass scales the image up to full-resolution. All techniques make use of the same reconstruction filter. The filter is similar to previous work by Stachowiak [Sta18], where each pixel gets classified into four classes using blue noise so that each full resolution pixel has a unique set of ray samples. This results in an array of offsets that are used to sample neighboring pixels. For each sample, the *bidirectional distribution function* (BRDF) response for a full resolution pixel gets calculated. The local BRDF gets multiplied by the original inverse PDF to weigh the color from each hit point. The weighted color for each sample is summed and normalized with the total weight of all samples. This method was presented by Stachowiak and Uludag [SU15]. *Backprojection*, using the G-Buffers velocity-buffer, is used to find the previous frame's color in a following temporal step. The current and previous samples are accumulated using an exponential moving average and stored in a history buffer. Each sample in the history buffer gets validated by backprojecting the geometric world-space normal and compare with the current one to ensure consistency over frames. Ghosting is prevented by using neighbourhood clipping similar to previous work [Kar14, Ped16, Sal16], which uses the estimated variance of the samples. The number of history samples to keep in the history buffer depends on the roughness of the surface. Rougher surfaces allow a longer history length. The current history length is stored in the history buffer, and if the history sample is rejected, the length is set to one. The final pass for the reflections is a bilateral Gaussian blur with a variable kernel size. The kernel size is based

on the pixel's variance, calculated during the spatial reconstruction. Pixels with higher variance use a wider kernel.

4.2. Experiment Setup and Execution

The system that was used to perform the performance tests and rendering consists of an i7 9700k CPU from Intel clocked at 3.6 GHz at base speed. However, it can boost up to 4.9GHz. The graphics card is an Nvidia RTX 3090 Founders Edition. The OS used during the performance tests was Windows 10 Pro (20H2) using the newest build available, 19042.928, with the latest Nvidia GPU drivers that were available at the time, 466.27. The resolution of the rendered output was 1920*x*1080, the full resolution of the application. The performance tests were performed by having a camera fly through the scene and collect 5000 samples of the frame time. The camera is moving with a fixed time-step of 16.66 milliseconds and is independent from the rendering time, resulting in similar frames during each test. ꟻLIP uses the resolution of the display and the distance from the display to the viewer as input to calculate the perceived error correctly [ANAM*20]. The width of the display in pixels was 2560 and 0.5 meters. The distance to the display was set to 0.62 meters. All the used images were captured using Nvidia Nsight in a separate run of the application so that Nsight did not impact the results of the performance tests. The images were exported from Nsight in a low-dynamic range format.

5. Results and Analysis

The results are presented in Table 2, showing the GPU frame time from each test and the errors that ꟻLIP outputs, shown in Table 3. The results also show some examples of the final reflections and rendered images. These images are shown with an error map that is the output from ꟻLIP. The timings, shown in Table 2, reveal a significant difference between the reference, tracing one ray per full resolution pixel, and the techniques that trace one ray per quarter resolution pixel. However, the increase is dependent on the scene since the Sun Temple scene does not show as dramatic improvements. When comparing the reference technique to the others, one can see that the reference technique has close to double the average frame time in the ray generation stage. Introducing VRS only improves the timings. The VRS image generation step does not take up a significant portion of the frame time in either of the two techniques. Table 2 shows that this stage takes up zero frame time for all techniques without VRS since they do not have a VRS image to generate. The stages that utilize VRS are *inline VRS rough* and *inline VRS graze*. Overall Table 2 shows that using VRS results in increased performance in all the scenes. The *inline VRS rough* technique performs roughly 30.3% better compared to the *quarter resolution* technique, and the *inline VRS graze* performs roughly 5.5% better in the Sponza scene. In the Sun Temple, *inline VRS rough* performs roughly 9.8% better compared to the *quarter resolution* technique, and the *inline VRS graze* performs roughly 17.1% better. The final scene, the Bistro scene, has a roughly 23.1% and 22.2% improvement respectively for *inline VRS rough* and *inline VRS graze*.

Figure 1 show that the quarter resolution techniques have many differences highlighted when compared to the full resolution reference. Limiting the number of rays further with VRS increases the highlights in the ꟻLIP error maps. It is also shown that the techniques using VRS have the highest error in the final render. Therefore, when combining inline ray tracing with VRS, there is a slight cost in visual quality. This is supported by the values in Table 3, which show the mean values for the ꟻLIP error maps. These values are proportional to the error perceived by human beings. The perceived error between the techniques and the reference image is also more noticeable on more reflective surfaces, such as the metallic ribbons on the colored curtains in the Sponza scene, which can be seen in Figure 1. However, even though the error maps show significant differences, the final render does not show this. Table 3 shows that *Inline VRS Rough* has the highest mean error in the Sponza scene. In the Sun Temple scene, *Inline VRS Graze* has the highest error. The error for the Sun Temple scene is lower than for the Sponza scene, which is interesting since the Sun Temple has more reflective surfaces. The Lumberyard Bistro follows the same results as the other scenes, but it has the highest mean error of all the scenes.

	Sponza	**Sun Temple**	**Bistro**
Technique	Avg (ms)	Avg (ms)	Avg (ms)
VRS Image Generation			
Reference	0.000	0.000	0.000
Quarter Resolution	0.000	0.000	0.000
Inline VRS Rough	0.024	0.024	0.024
Inline VRS Graze	0.022	0.017	0.017
Dispatch Rays			
Reference	0.797	1.535	2.995
Quarter Resolution	0.279	0.508	0.885
Inline VRS Rough	0.118	0.346	0.517
Inline VRS Graze	0.169	0.271	0.536
Ray Tracing Total			
Reference	**1.363**	**2.107**	**3.619**
Quarter Resolution	**0.971**	**1.080**	**1.481**
Inline VRS Rough	**0.677**	**0.974**	**1.139**
Inline VRS Graze	**0.918**	**0.895**	**1.152**

Table 2: *Timings from the Crytek Sponza, Sun Temple, and Lumberyard Bistro scenes.*

	Sponza	**Sun Temple**	**Bistro**
Technique	Mean	Mean	Mean
Final Reflections			
Inline (VRS Rough)	0.072	0.043	0.086
Inline (VRS Graze)	0.071	0.046	0.086
Ray Gen (Quarter Res)	0.063	0.040	0.078
Final Render			
Inline (VRS Rough)	0.006	0.000	0.000
Inline (VRS Graze)	0.006	0.000	0.000
Ray Gen (Quarter Res)	0.005	0.000	0.000

Table 3: *The error in numbers for the error images and the corresponding technique. This table shows the error for the Sponza, Sun Temple, and Lumberyard Bistro scenes.*

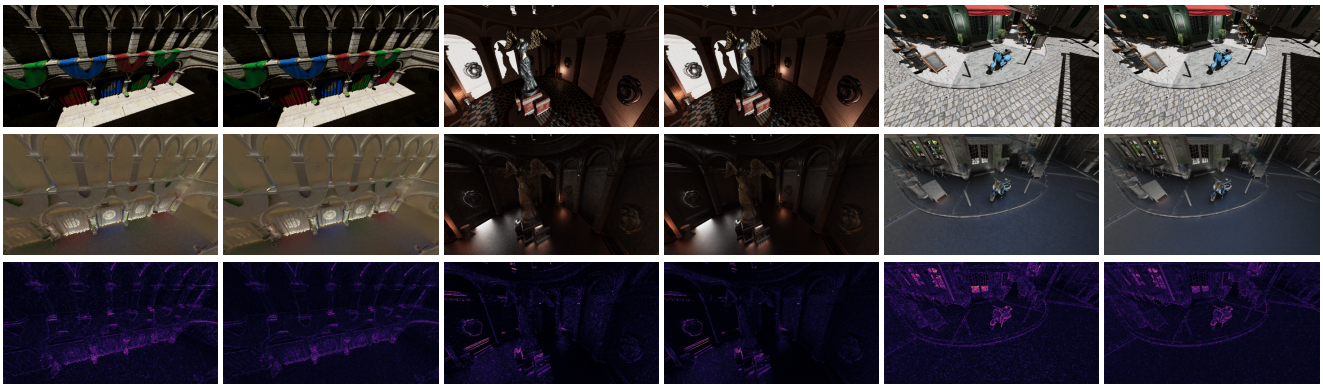

Figure 1: *Visual results from the Crytek Sponza scene (Two Left Columns), Epic Games Sun Temple scene (Two Middle Columns), and the Amazon Lumberyard Bistro scene (Two Right Columns). Rendered with inline ray tracing at quarter resolution with VRS enabled (First Column) and ray generation shaders at quarter resolution (Second Column). Grazing angles were used to determine the shading rate in this example. The middle row of images show the reflections whilst the bottom row show the FLIP error map with the reference images. Black means no perceived error, and yellow means high error [ANAM*20]. Few highlighted areas mean better visual quality.*

6. Discussion

In Section 5 results from the performance tests were presented, which showed that combining inline ray tracing with VRS results in a lower frame time. The explanation for this is that the number of rays that are traced has been decreased, reducing the number of calculations performed. However, the best method to generate the VRS image seems to heavily depend on the scene since the results vary between the test scenes. In the Sponza scene, using the roughness to determine the shading rate gives better frame time, while in the other scenes, the results are close. Nonetheless, using image-based VRS is a competitive method compared to rendering at a quarter resolution only. This means that if the noise could be reduced even further without decreasing the performance, this could be a method for reducing the number of rays that are traced in a real-time ray traced application.

Inline ray tracing combined with VRS gives a performance increase from 5.5% to 30.3%. One could compare this against the related work, where the performance increase was up to around 15% [Dro20, VR21]. This is in the range of the results found in this paper. The performance might be improved further if more threads on the GPU took the same code path. As the ray directions are seemingly random, threads often has to take different code paths across lanes, or access different memory locations, which can lead to degraded performance. One ray could hit a primitive while the neighboring threads are calculating rays that miss. In the current implementation, dynamic branching occurs with many texture fetches for the different materials, which could degrade the performance. This could be improved by performing a ray binning step before the rays are traced, which has been tried before [DS19]. However, VRS was not free in terms of visual quality since it did have an impact. This means that it may need more work to reduce the noise in the reflection image before it is used in a production environment. However, it shows some promising results.

7. Conclusions and Future Work

This paper concludes that VRS does improve the performance of a real-time ray traced application compared to the quarter resolution technique. The performance of inline ray tracing combined with VRS showed an improvement of at least 5%. However, there is a cost in visual quality due to the lowered number of rays. Therefore, more work is needed before the technique is used in production. Whether roughness or grazing angles should determine the shading rate in the VRS image depends on the scene. When using roughness, it results in improved performance for scenes containing many surfaces with high roughness values. In the current implementation, using grazing angles result in blocky or blurry results on mirror-like surfaces. When viewing a mirror so that the mirror's normal is parallel with the view vector, the shading rate is lowered, resulting in artifacts. Therefore, a combination of roughness and grazing angles might be a better technique for determining the shading rate. When lowering the number of rays cast in the scene, the visual quality should remain comparable with existing techniques, with this paper comparing against quarter resolution rendering. Future work should generate rays in a separate step and bin these so that rays with similar directions and world positions are grouped. The similarity of the rays after the ray binning step could then be used to generate a VRS image. Similar rays would share the same hit result instead of shooting one ray for each. The ray that is used within that group could be varied between frames. Software VRS methods could also be explored, and it would be interesting to check the performance characteristics of the ray generation shaders combined with software VRS in the future since these could be more optimized for these types of workloads.

References

[AMD20] AMD: AMD RDNA™ 2 - DirectX® 12 Ultimate: Variable Rate Shading, Nov. 2020. [Online]. Accessed: 2021-01-26. Available: https://www.youtube.com/watch?v=LqpqYx3AiAQ. 1

[ANAM*20] ANDERSSON P., NILSSON J., AKENINE-MÖLLER T., OSKARSSON M., AASTRÖM K., FAIRCHILD M. D.: ꟻLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 3*, 2 (2020), 15:1–15:23. 1, 3, 4

[CP76] CRANLEY R., PATTERSON T. N. L.: Randomization of Number Theoretic Methods for Multiple Integration. *SIAM Journal on Numerical Analysis 13*, 6 (1976), 904–914. Publisher: Society for Industrial and Applied Mathematics. 2

[Dro20] DROBOT M.: Software-based Variable Rate Shading in Call of Duty: Modern Warfare, 2020. Advances in Real-Time Rendering in Games: Part I, ACM SIGGRAPH 2020 Courses. 2, 4

[DS19] DELIGIANNIS J., SCHMID J.: "It Just Works": Ray-Traced Reflections in 'Battlefield V', Mar. 2019. Game Developers Conferance 2019. 1, 2, 4

[GF16] GEORGIEV I., FAJARDO M.: Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks* (Anaheim California, July 2016), ACM, pp. 1–1. 2

[Gro20] GROHLVANA: PlayStation 5 Ray Tracing First Look: Gran Turismo 7, Ratchet & Clank, Pragmata + More!, June 2020. [Online]. Accessed: 2021-01-26. Available: https://www.youtube.com/watch?v=Azl772uylh4. 1

[HAM19] HAINES E., AKENINE-MÖLLER T. (Eds.): *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*. Apress, Berkeley, CA, 2019. 1, 2

[Kar13] KARIS B.: Real Shading in Unreal Engine 4, 2013. Physically Based Shading in Theory and Practice, ACM SIGGRAPH 2013 Courses, 2013. 2

[Kar14] KARIS B.: High Quality Temporal Supersampling, 2014. Advances in Real-Time Rendering in Games: Part I, ACM SIGGRAPH 2014 Courses. 2

[KK02] KOLLIG T., KELLER A.: Efficient Multidimensional Sampling. *Computer Graphics Forum 21*, 3 (2002), 557–563. 2

[KKW*13] KELLER A., KARRAS T., WALD I., AILA T., LAINE S., BIKKER J., GRIBBLE C., LEE W.-J., MCCOMBE J.: Ray tracing is the future and ever will be... In *ACM SIGGRAPH 2013 Courses* (New York, NY, USA, July 2013), SIGGRAPH '13, Association for Computing Machinery, pp. 1–7. 1

[KVBB*19] KELLER A., VIITANEN T., BARRÉ-BRISEBOIS C., SCHIED C., MCGUIRE M.: Are we done with ray tracing? In *ACM SIGGRAPH 2019 Courses* (New York, NY, USA, July 2019), SIGGRAPH '19, Association for Computing Machinery, pp. 1–381. 1

[Mic19a] MICROSOFT: Boost rendering performance with Variable Rate Shading | Game Developers Conference 2019, May 2019. [Online]. Accessed: 2021-01-20. Available: https://www.youtube.com/watch?v=f-SklVb2MDI. 1

[Mic19b] MICROSOFT: Variable-rate shading (VRS) - Win32 apps, 2019. [Online]. Accessed: 2021-04-14. Available: https://docs.microsoft.com/en-us/windows/win32/direct3d12/vrs. 1

[Mic21] MICROSOFT: DirectX Raytracing (DXR) Functional Spec, Mar. 2021. [Online]. Accessed: 2021-01-26. Available: https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html. 1

[Ped16] PEDERSEN L. J. F.: Temporal Reprojection Anti-Aliasing in INSIDE, 2016. Game Developers Conference, 2016. 2

[Sal16] SALVI M.: An excursion in temporal supersampling, Aug. 2016. Game Developers Conference, 2016. 2

[Shi05] SHIRLEY P.: Ray tracing. In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, July 2005), SIGGRAPH '05, Association for Computing Machinery, pp. 2–es. 1

[Sta18] STACHOWIAK T.: Stochastic all the things: Raytracing in Hybrid Real-Time Rendering, July 2018. Digital Dragons Presentation, 2018. 1, 2

[SU15] STACHOWIAK T., ULUDAG Y.: Stochastic Screen-Space Reflections - Frostbite, Aug. 2015. Advances in Real-Time Rendering in Games, ACM SIGGRAPH 2015 Courses. 2

[Uli88] ULICHNEY R.: Dithering with blue noise. *Proceedings of the IEEE 76*, 1 (Jan. 1988), 56–79. 2

[VR21] VAN RHYN J.: Moving Gears to Tier 2 Variable Rate Shading, Jan. 2021. [Online]. Accessed: 2021-04-27. Available: https://devblogs.microsoft.com/directx/gears-vrs-tier2. 2, 4