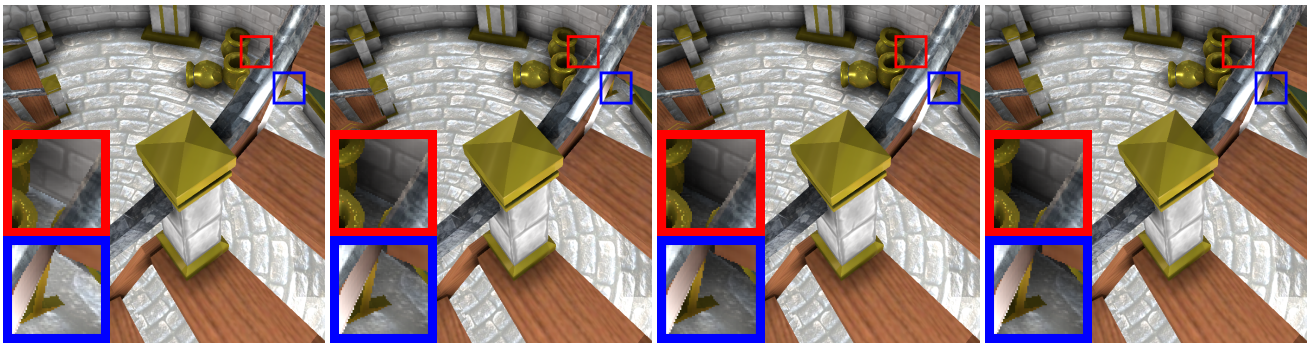
Eurographics Symposium on Rendering (2024)
E. Garces and E. Haines (Editors)

Ray Traced Stochastic Depth Map for Ambient Occlusion

Felix Brüll  René Kern Thorsten Grosch

TU Clausthal, Germany



(a) *Volumetric AO (0.28ms) [LS10]* **(b)** + *Stochastic Depth (0.68ms) [VSE21]* **(c)** + *Ours (0.54ms)* **(d)** *Reference (1.20ms) [BKG22]*

Figure 1: *Ambient occlusion with many occluders in a small tower scene with a spiral staircase (77K triangles). Our method is able to represent ambient occlusion faithfully, even with multiple overlapping occluders. The performance improvements of our method become more significant for larger scenes.*

Abstract

Screen-space ambient occlusion is a popular technique for approximating global illumination in real-time rendering. However, it suffers from artifacts due to the lack of information from the depth buffer. A stochastic depth map [VSE21] can be used to retrieve most of the missing information, but it is not suitable for real-time rendering in large scenes. In this paper, we propose a new stochastic depth map acquisition method powered by hardware ray tracing, which shows better performance characteristics than the previous method. We present further improvements that increase the quality and performance of the stochastic depth map generation. Furthermore, the results are almost indistinguishable from a ground truth solution with all depth samples.

CCS Concepts

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

1. Introduction

Ambient Occlusion (AO) approximates global illumination by darkening corners and crevices of objects. For real-time applications, it is generally computed in screen-space, using the depth buffer and the normal buffer. Unfortunately, the fidelity of AO is compromised when relying solely on a single depth buffer, which leads to perceptible artifacts such as incomplete shadowing and halos around objects as shown in Fig. 1a. Additional depth values can be retrieved through depth peeling [Eve01; MMNL16], *k*-buffers [Sal13], ray tracing [BKG22], or **Stochastic Depth maps (SD-maps)** [VSE21].

Ray tracing can also be used to compute ground-truth AO directly, but it requires a temporal denoiser to be usable in real-time applications and thus tends to suffers from temporal disocclusion artifacts [Mic19; Zhd21]. Another branch of research focuses on synthesizing AO from geometry buffers via neural networks, though these approaches cannot be executed in real-time yet [NAM*17; ZXL*20; WZZ*23].

In this work we present an alternative approach to acquire a stochastic depth map. In particular, we contribute:

- Sec. 3 A novel SD-map acquisition via hardware ray tracing.
- Sec. 3.1 An improved collection algorithm for better quality.
- Sec. 3.2 Ray interval optimizations for performance & quality.
- Sec. 3.3 A low resolution SD-map for better performance.
- Sec. 3.4 A radius cutoff to reduce the number of required rays.
- Sec. 3.5 A lazy guard band to mitigate screen-space artifacts.

DOI: 10.2312/sr.20241160

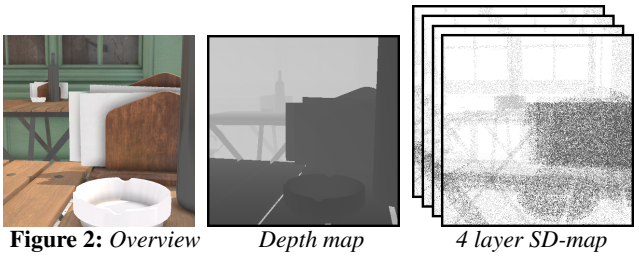


Figure 2: *Overview* *Depth map* *4 layer SD-map*

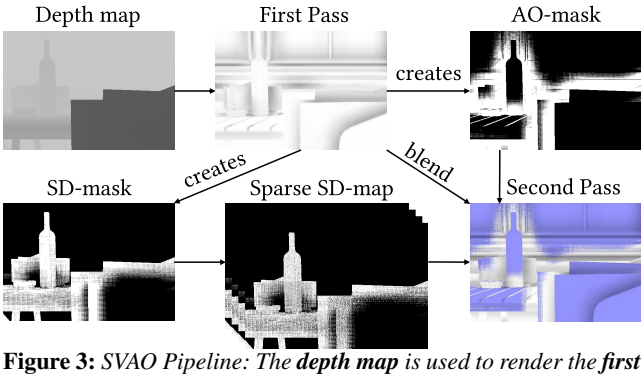


Figure 3: *SVAO Pipeline: The **depth map** is used to render the **first AO pass**, which also generates two masks: The **AO-mask** marks partially occluded pixels with potential artifacts. The **SD-mask** pinpoints occluder pixels that require additional depth values. The **sparse SD-map** is constructed for the pixels flagged by the SD-mask. The **second AO pass** utilizes the SD-map to refine the AO values (depicted in white). Values in purple remain unaltered from the initial pass.*

2. Previous Work

A first algorithm to compute AO for static scenes was proposed by Zhukov et al. [ZIK98]. Attempts for dynamic scenes were made [Bun05; KL05], and in the end, screen-space AO became the state of the art for real-time AO [Mit07; SA07; BSD08; TP16; LS10; JWPJ16; HSEE15].

SD-maps are multi-layered depth maps, where each layer contains one random depth value for each pixel (see Fig. 2).

In the original paper, a full SD-map is created for each frame, which can be unsuitable for real-time applications with bigger scenes. However, not all depth values are required for AO and a sparse SD-map can be sufficient.

A pipeline for the creation of a sparse SD-map was proposed in **Stenciled Volumetric AO (SVAO)** [BKG22]. First, an initial AO pass is executed with the regular depth buffer. The first pass also marks areas, where the SD-map is required (SD-mask). Then, the SD-map is created for the marked areas. Finally, the AO pass is executed again, with additional information from the SD-map. This procedure is visualized in Fig. 3.

The SVAO paper also proposes an alternative to the SD-map, which is based on ray tracing. Here, the second AO pass traces the missing depth values directly and the SD-mask is not required.

Listing 1: *Ray-SD map any-hit shader.*

```
1  void anyHit(inout RayPayload d, attribs) {
2   float rng = hash(attribs.barycentrics);//[0,1)
3   uint slot = d.count;
4   d.count += 1;
5   if (d.count > N)
6      slot = uint(rng * d.count);
7
8   if (slot < N)
9      if (RayTCurrent() < d.depths[slot])//z-cull
10        if (alphaTest(attribs))
11           d.depths[slot] = RayTCurrent();
12
13  if (d.count < MAX_COUNT)
14     IgnoreHit(); // continue traversal
15 }
```

3. Ray Traced Stochastic Depth Map

Previous work has shown significant performance improvements for the SD-map when it is only created for a subset of pixels instead of the whole screen [BKG22]. However, the traditional rasterizer based SD-map needs to process all vertices in the view frustum nevertheless. By utilizing hardware ray tracing we cull geometry more efficiently, which results in a performance improvement for sparse SD-maps in larger scenes.

For simplicity, we will omit some implementation details, but the full source code can be found on Github: https://github.com/TU-Clausthal-Rendering/Ray-Traced-Stochastic-Depth-Map.

3.1. Stochastic Collection

The original SD-map [VSE21] implementation uses the coverage mask from multisampling to discard and replace samples in a stochastic manner. This approach is very fast, but it has some inherent quality issues. For instance, there is a potential scenario where the shader generates the same coverage mask for all depth samples, in which case only the last depth sample will be stored.

For ray tracing we can solve this problem with reservoir sampling [Vit85]. For every pixel, we trace a single view ray and handle all depth values within the any-hit shader. Within the ray payload, we maintain N depth values and a counter, which are required for the reservoir sampling as shown in Listing 1. Reservoir sampling will first fill the buffer with N distinct depth samples. Subsequently, each additional sample replaces a payload sample with a probability of $N/count$. Additionally, we introduce a MAX_COUNT variable to mitigate potential performance degradation when the ray traverses too much geometry. In our examples we use $N = 4$ and MAX_COUNT$= 8$. We also use z-culling when overwriting existing depth samples, i.e. we prioritize smaller depth values. As a last step we also need to perform alpha testing for certain geometry.

The 2D hash function for reservoir sampling was taken from Wyman et al. [WM19]:

$$frac(1.0e4 * sin(17.0 * u + 0.1 * v) * (0.1 + abs(sin(13.0 * v + u))))$$

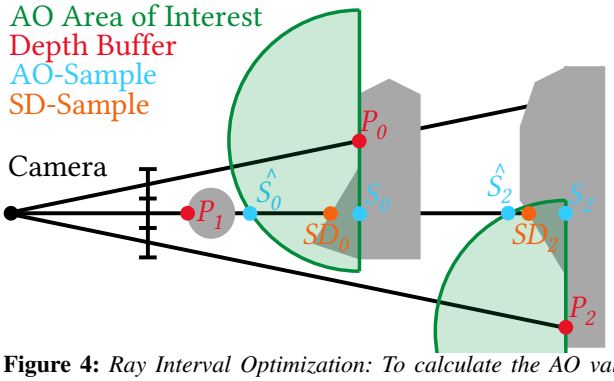


Figure 4: *Ray Interval Optimization: To calculate the AO value at P_0, our algorithm samples up to 8 different positions in a disc around P_0. The green hemisphere represents the area of interest for our AO calculation. Here, the first sample position S_0 reprojects to P_1 from the depth buffer, which is in front of our area of interest. All relevant depth values for S_0 are between $\hat{S}_0$ and S_0. In this case we require the depth value SD_0. The same scenario applies to P_2 and S_2. Thus, we need to trace a ray between $\hat{S}_0$ and S_2 to obtain the relevant depth values (SD_0 and SD_2) for our SD-map.*

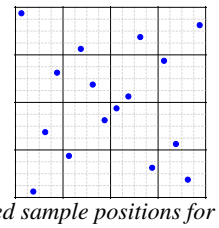


Figure 5: *Jittered sample positions for a 4x4 pixel grid.*

3.2. Ray Interval Optimization

Another advantage of the ray tracing pipeline is the ability to cull geometry with ray intervals on a per-pixel basis. This is especially useful for our AO computations, since we are only interested in depth values within a limited range. The concept is illustrated and explained in Fig. 4.

To realize this optimization, we keep track of the *rayMin* and *rayMax* values with two additional textures. Initially, the *rayMin* and *rayMax* textures are filled with FLOAT_MAX and 0 respectively. In the first AO pass, we update the *rayMin* and *rayMax* values via *atomicMin* and *atomicMax* instructions when required. Atomic operations are necessary because two different pixels can update the same ray values as shown in Fig. 4. In the second AO pass, we then use the *rayMin* and *rayMax* values as our *TMin* and *TMax* values for the ray.

Note, that atomic operations are generally not supported for floating-point values on GPUs. We circumvent this issue by reinterpreting the floating-point values as integer values. This is possible because the order of positive floating-point values is the same as the order of their bitwise integer representation, which is required for the min and max operations.

3.3. Low Resolution Stochastic Depth Map

The SD-map can be created at a lower resolution than the depth buffer, which will result in a significant performance improvement. However, operating at a lower resolution will increase aliasing near edges. Fortunately, this is limited to the small subset of samples that are refined during the second AO pass. To mitigate aliasing artifacts, we leverage the sample positions depicted in Fig. 5 for each 4x4 pixel grid, as opposed to sampling the pixel centers. The sample positions were generated from a 2D Sobol sequence [Sob67], which has the advantageous property that the associated 16x16 grid aligns with latin hypercube sampling. It is crucial to note that the use of per-pixel jitter is exclusive to the ray tracing pipeline and is not compatible with the rasterizer.

3.4. Radius Cutoff

The predominant artifacts in AO, halos around objects, are particularly noticeable for objects in proximity to the camera. The extent of a potential screen-space halo is proportional to the distance between the camera and the object of interest. To enhance the efficiency of creating our SD-map without compromising the AO quality, we limit the SD-map to pixels that are more likely to generate larger halos.

To seamlessly blend the SD-enhanced AO with the non-enhanced AO, we opt for a per-sample decision rather than a per-pixel decision: Each pixel assesses, for each of its 8 AO samples, whether the screen-space distance in pixels is below a threshold t. For example, consider pixel P_0 in Fig. 4, which calculates the screen-space distance between P_0 and sample S_0 to determine whether a query for sample S_0 should be included in the SD-map (In this example, the screen space distance between P_0 and S_0 is one pixel). If the screen-space distance is less than t, the respective sample is only evaluated in the first AO pass, and excluded from the SD-map. We use $t = 6$ pixels in our examples, as this did not produce visible halos on our 100 DPI monitor.

3.5. Stochastic Depth Guard Band

Screen-space AO frequently encounters flickering artifacts near the screen borders if the required depth values lie outside the screen. To address this problem, it is common practice to employ a guard band that enlarges the screen by a few pixels. By default, our approach incorporates a 64-pixel guard band, which adequately resolves the issue in most scenarios. Nevertheless, this guard band size may prove inadequate for objects close to the screen border that demand a larger screen-space radius than 64 pixels. This situation arises occasionally in our method because we utilize a world-space radius for AO calculations, leading to larger screen-space radii for objects in proximity.

An improvised approach involves capping the screen space radius at 64 pixels by reducing the world-space radius whenever it surpasses this threshold. This adjustment results in the AO shadows becoming smaller as the camera approaches an object. Thus, we do not recommend it.

Our empirical analysis revealed that constraining the AO radius to 512 screen space pixels does not lead to noticeable shrinkage of

the shadows. As a result, setting the AO radius limit to 512 pixels and using a 512-pixel guard band effectively eliminates artifacts near the screen edges. However, this approach considerably decelerates the generation of the primary depth buffer.

Therefore, we suggest a different approach: We propose expanding the SD-map by 512 pixels in each direction. With this approach, ray tracing will be used to obtain depth values from beyond the screen borders only if necessary. It is important to note that, in the case of the 1/4 resolution SD-map, this expansion requires just 128 pixels in each direction.

4. Results

In this section, we present the results of our **ray traced SD-map (Ray-SD)**. We will compare our results with the **raster-based SD-map (Raster-SVAO)** and against the **ray traced SVAO (Ray-SVAO)** from the SVAO paper [BKG22]. Note that the Ray-SVAO traces up to 8 rays per pixel and produces accurate AO results that serve as the ground truth reference for our method. All algorithms are implemented in the FALCOR framework [KCK*22] and executed on an NVIDIA RTX 2080 Ti. We recorded all images at a resolution of 1920x1080 with a default 64 pixel guard band to mitigate AO artifacts at the screen borders (2048x1208).

For our performance analysis, we assume that ray tracing is already integrated at some point in the pipeline, such as for shadows or reflections. Consequently, we presume that the ray tracing acceleration structure is readily available and pre-built. This assumption includes the necessity for potential per-frame updates that involve rebuilding the acceleration structure. We focus on the additional overhead introduced by our AO method, without accounting for the initial setup of the ray tracing infrastructure.

For better contrast we doubled the AO exponent for all images in the results section. The video in the supplementary material use the original exponent, since artifacts are more noticeable in motion. Additionally, the video use Falcor's built-in temporal anti-aliasing.

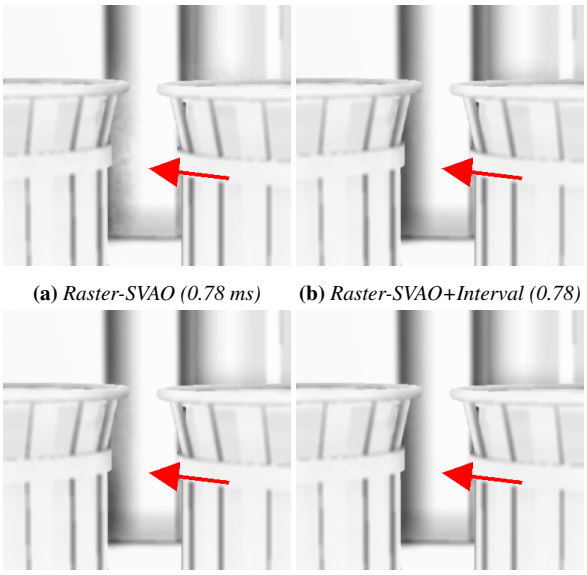
The rendering times include the full SVAO pipeline (two AO passes and SD-map creation). These times do not include the GBuffer or the final bilateral filter which blurs the AO (~0.2ms).

We will start with a quality and performance analysis of our proposed optimizations from Sec. 3.1 - Sec. 3.5 and conclude with an overall comparison of the following scenes:

- **Bistro** [Lum17] (2.8M triangles): A medium-sized scene with a lot of small detailed geometry.
- **Emerald Square** [NB17] (10M triangles): A large scene with a lot of alpha tested foliage, which is notoriously expensive to render with the previous SVAO implementation.
- **Sun Temple** [Gam17] (600K triangles): A smaller scene with a lot of large occluders.

4.1. Stochastic Collection and Ray Interval Optimization

Fig. 6 shows a comparison of the coverage mask based collection from the rasterizer (Fig. 6a) and our reservoir sampling based collection in the ray tracer (Fig. 6c). It is already apparent that the coverage mask collection misses more depth values than our collection



(a) *Raster-SVAO (0.78 ms)* **(b)** *Raster-SVAO+Interval (0.78)*

(c) *Ray-SD (0.35 ms)* **(d)** *Ray-SD+Interval (0.21 ms)*

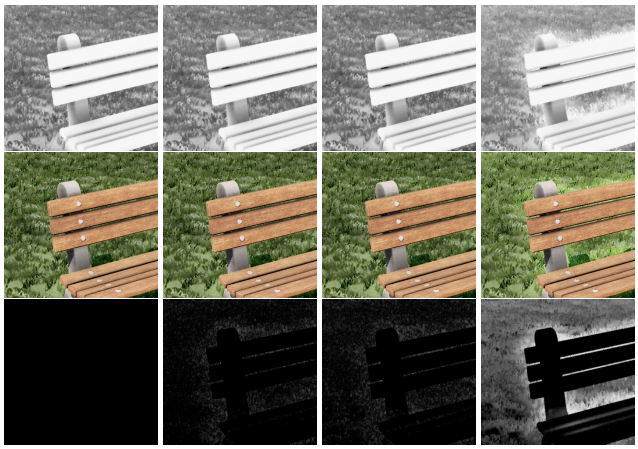
Figure 6: *AO with trashcans in the Emerald Square*

Table 1: *Performance breakdown of our 1/4 resolution SD-map for the Emerald Square at 0 and 11 seconds (Fig. 15). The timings in milliseconds were recorded without and with our interval optimization (**+Int**).*

	Emerald 0s		**Emerald 11s**	
		+Int		**+Int**
Clear SD-Mask/MinMax	0	0.01	0	0.02
AO1: read depth & calc	0.23		0.25	
AO1: SD-Mask/MinMax	0.01	0.03	0.01	0.06
1/4 SD-map (ray traced)	0.66	0.25	0.62	0.3
AO2: read depth & calc	0.09		0.13	
AO2: read SD-map	0.03		0.05	
Total:	1.02	0.64	1.06	0.81

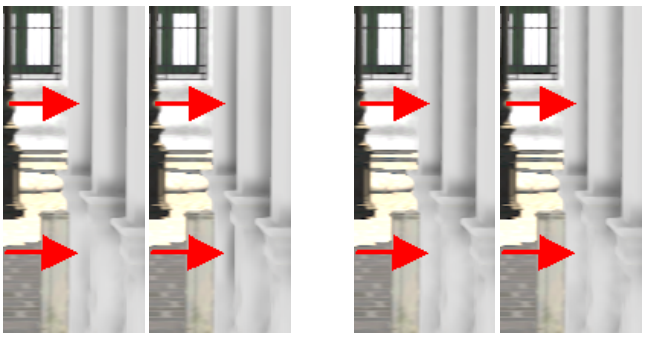
algorithm. Introducing our ray interval optimization in Fig. 6b and Fig. 6d improves the quality of both maps, but only the ray tracer gains a performance improvement. The quality improvements stem from the fact, that the back side from the trashcans will be excluded from the sample collection. Thus, more relevant samples from the background wall will be collected, a similar scenario as in Fig. 1. The performance improvement is due to the fact that the ray tracer can terminate early when all intersections in the given interval were handled. The rasterizer always needs to process all fragments in the pixel shader, this includes the trashcan geometry and hidden geometry behind the wall.

A thorough performance analysis of our interval optimization is shown in Tab. 1. The table reveals that our interval optimizations incur some initial penalties for clearing and writing to the *rayMin* and *rayMax* textures. Nonetheless, the performance improvements in the SD-map pass justify the additional cost.



(a) *Full res.*	**(b)** 1/2 *res.*	**(c)** 1/4 *res.*	**(d)** *no SD-map*
1.06 ms	*0.71 ms*	*0.61 ms*	*0.24 ms*

Figure 7: *Different resolutions of our SD-map. Top: AO, Middle: colorized, Bottom: AO difference to full res.*



(a) *No Jitter* **(b)** *Jitter*

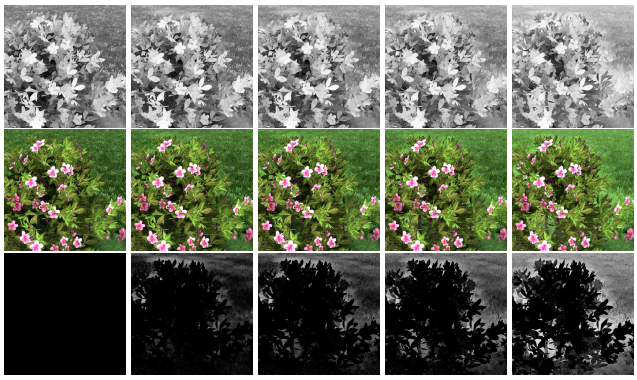
Figure 8: *Two consecutive frames of a camera path through a tunnel near the Bistro. The 1/4 resolution SD-map suffers from flickering without per-pixel jitter (left). Using the jittered sample positions (right) mitigates the artifacts. Refer to the supplemental material for the video.*

4.2. Low Resolution Stochastic Depth Map

Fig. 7 shows a comparison of our SD-map in full resolution (2048x1208), half resolution (1024x604) and quarter resolution (512x302). Based on the provided difference images, we can see that there are some minor differences in the lower resolution maps, but they are not noticeable in the final AO image.

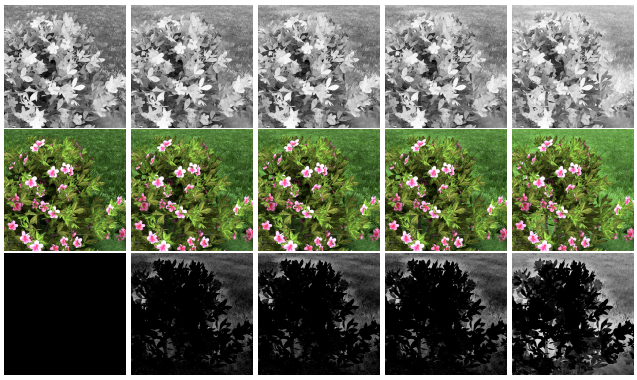
Without our per-pixel jitter from Fig. 5, the quarter resolution SD-map suffers from flickering artifacts near edges as indicated by the red arrows in Fig. 8a. Fortunately, our jittered sample positions mostly mitigate the artifacts as shown in Fig. 8b.

We also tried resolutions below 1/4, but we advise against their usage for two key reasons. First, the performance gains are negligible, as the cost of writing to the ray interval textures increases in tandem with the decrease in ray tracing time. This is because more atomic operations overlap at lower resolutions. Secondly, aliasing artifacts become more problematic.



(a) *Reference*	**(b)** $N = 8$	**(c)** $N = 4$	**(d)** $N = 2$	**(e)** *VAO*
4.71 ms	*0.77 ms*	*0.71 ms*	*0.69 ms*	*0.22 ms*
	MSE 0.0019	*MSE 0.0048*	*MSE 0.011*	*MSE 0.04*

Figure 9: *Different values of N with MAX_COUNT= 8. Top: AO, Middle: colorized, Bottom: AO difference to reference.*



(a) *Reference*	**(b)** $M = 16$	**(c)** $M = 8$	**(d)** $M = 4$	**(e)** *VAO*
4.71 ms	*0.77 ms*	*0.71 ms*	*0.66 ms*	*0.22 ms*
	MSE 0.0031	*MSE 0.0048*	*MSE 0.0089*	*MSE 0.04*

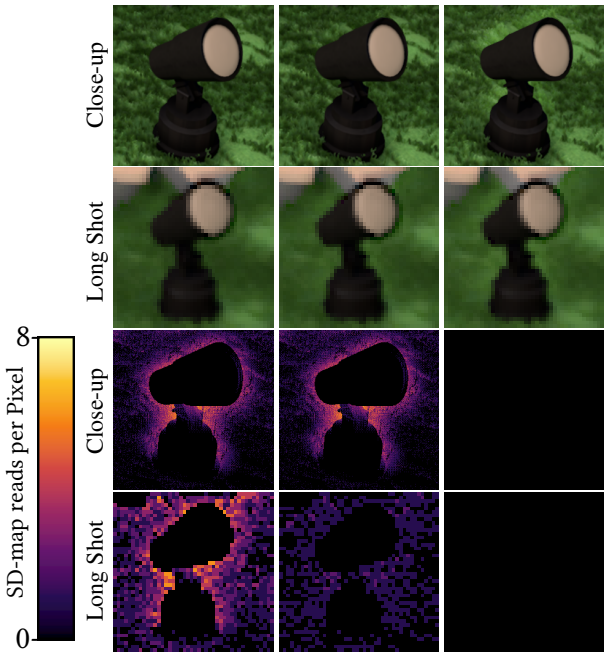
Figure 10: *Different values of M=MAX_COUNT with $N = 4$. Top: AO, Middle: colorized, Bottom: AO difference to reference.*

Fig. 9 and Fig. 10 show our 1/4 resolution SD-map with different values for N and MAX_COUNT. Overall, we found that $N = 4$ and MAX_COUNT= 8 provide a good balance between quality and performance. The figures show an example with a depth complexity that occasionally exceeds MAX_COUNT= 16. Even in this difficult scenario, the quality of the AO is acceptable but slightly brighter overall.

4.3. Radius Cutoff

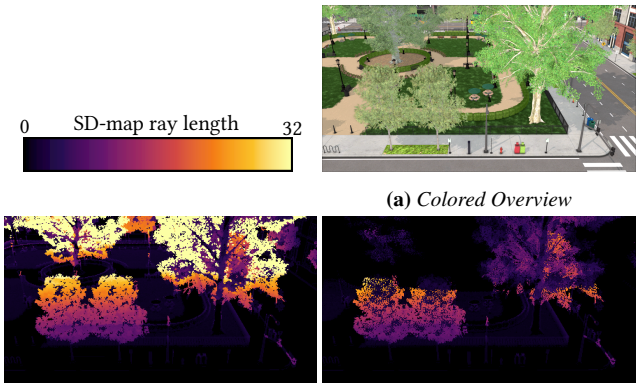
An example of our radius cutoff is shown in Fig. 11b. For the close-up view, the same amount of rays are required with and without the cutoff to prevent halos. However, for the long shot, the cutoff is able to reduce the number of required SD-map fetches without introducing major artifacts.

An overview of the full long shot is shown in Fig. 12. The cutoff is able to reduce the total number of rays, and it also shortens the ray interval lengths.



	(a) *1/4 SD-map*	**(b)** + *cutoff*	**(c)** *no SD-map*
Close-up:	*0.82 ms*	*0.81 ms*	*0.24 ms*
Long shot:	*1.03 ms*	*0.69 ms*	*0.24 ms*

Figure 11: *Close-up view and long shot of a spotlight. The 1/4 res. SD-map (a) is able to prevent halos. The radius cutoff (b) enhances performance without introducing noticeable halos. The variant without the SD-map (c) exhibits significant halo artifacts in the close-up view. The heatmap shows the number of samples per pixel, which are refined in the second AO pass (and will read data from the SD-map).*



(a) *Colored Overview*

(b) *1/4 res. SD-map (1.03 ms)* **(c)** + *cutoff (0.69 ms)*

Figure 12: *The full long shot of Fig. 11 is shown in (a). The ray interval lengths for the SD-map are visualized in (b) and (c).*

Overall, the cutoff is able to increase the performance without introducing noticeable artifacts. The performance increase depends on the camera position. In our example, the close-up view has almost identical performance with and without the cutoff. However, in the long shot the cutoff saves around 30% of the rendering time. This is a favorable behavior, since ray tracing times usually increase when a larger portion of the scene is in the view frustum, but our cutoff will reduce the number of rays in such a scenario.

4.4. Stochastic Depth Guard Band

Fig. 13 shows an example where our guard band expansion is required. The SD-mask in Fig. 13c shows the usual SD-mask in the center and the new additional samples on the border. The SD guard band removes all flickering artifacts near the screen borders.

Fig. 14 shows the performance breakdown of our SD guard band along the camera path in the Bistro. A naive 512 pixel default guard band significantly increases the rendering time of the depth buffer, which increases the overall rendering time noticeably. Our SD guard band on the 1/4 resolution SD-map only causes a slight increase in the overall rendering time (depth buffer + AO rendering). The SD guard band only increases the rendering time of the AO algorithm, but it has no impact on the rendering time of the depth buffer. Our remaining test scenes showed a similar behavior.

It should be mentioned that the SD guard band is only required when the renderer uses an unusually large AO radius. This is only the case in our Tower scene. In all other scenes, the default 64 pixel guard band is generally sufficient.

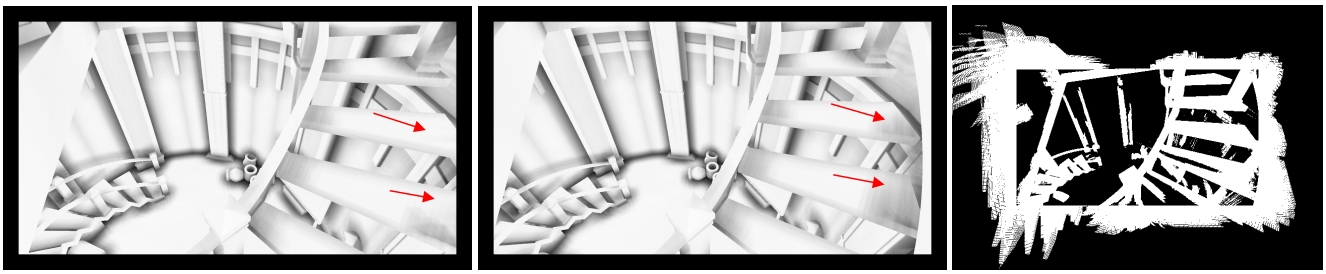
4.5. Overall Performance

For the overall comparison we recorded the rendering times along the default camera path of the Bistro, Emerald Square and Sun Temple. We used the radius cutoff for all techniques and also added the interval optimization for the rasterized SD-map to improve the quality. The results are shown in Fig. 15 and the full videos are available on our Github project page.

In general, we can see that our 1/4 resolution Ray-SD is able to outperform all other techniques. Furthermore, the previous Ray-SVAO displays the most variance in its rendering times, since it traces between 0 and 8 rays per pixel. Our Ray-SD on the other hand will trace at most one ray per pixel, or 1 ray per 16 pixels for the 1/4 resolution SD-map. We can also see that the ray based SD-map tends to outperform the raster based SD-map for the larger scenes (Bistro and Emerald Square), but the raster based SD-map gets faster for the smaller scenes (Sun Temple). This can be explained by the fact, that the time complexity for tracing a ray is logarithmic in the number of primitives, while the rasterizer has a linear time complexity.

In the following we will discuss the artifacts that can be seen on the camera paths for our 1/4 resolution Ray-SD and explain the performance drops.

Bistro: There are some flickering artifacts that can be seen on the bakery near the 8s mark. This is not caused by our low-resolution SD-map per se, since the same artifacts can be seen in the Ray-SVAO path. The artifacts are caused by the thin and detailed geometry of the bakery, which is hard to handle for AO techniques with a small number of samples. Other than this, our Ray-SD shows significant improvements over the single depth buffer AO. All techniques have a performance drop at the 84s mark, which is due to the high number of alpha tested foliage in the shot.



(a) *Camera 1* **(b)** *Camera 2* **(c)** *SD-mask with SD guard band*

Figure 13: *AO in the Tower with a 64 pixel guard band and without our SD guard band expansion. The red arrows indicate the areas where AO is flickering during slight camera motion from (a) to (b). The SD-mask shows the areas where our extended SD guard band would request additional depth values in the outer border.*

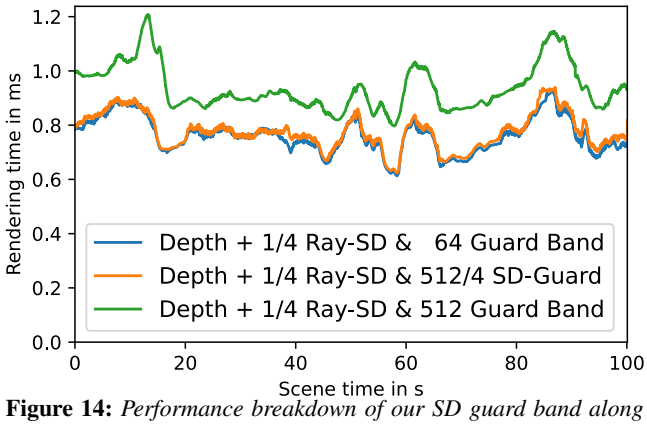


Figure 14: *Performance breakdown of our SD guard band along the camera path in the Bistro. All times include the rendering of the primary depth buffer plus the time of our AO calculation. The first variant only uses the 64 pixel default guard band and will suffer from artifacts near the border. The second variant uses our SD guard band expansion and is able to completely remove those artifacts. The third variant uses a 512 pixel default guard band instead.*

Emerald Square: There are no noticeable artifacts in the colored output. However, when inspecting the grayscale AO one can see some very small halo-like artifacts on the foliage, but they are hardly noticeable. The most notable performance drop at 11s is caused by the large number of alpha tested foliage in the shot. Fig. 16 illustrates the samples that are stored in the SD-map for the 11s mark.

The other performance drop at 41s mostly affects the rasterizer: For the ray based methods, the trees in the background are culled by the ray interval optimization, but the rasterizer processes those fragments nevertheless.

Sun Temple: Some minor flickering artifacts can be seen at a pillar near the 37s mark. This time, the artifacts are caused by the lower resolution SD-map, but they are hard to notice.

Overall we could not find striking differences between our Ray-SD and the (reference) Ray-SVAO, when comparing them side by side. However, some pixel differences are indeed visible when flipping between the Ray-SVAO and Ray-SD images, but they are generally very minor. One area where differences are more noticeable is on dense foliage. In these areas the Ray-SD results are generally marginally brighter, due to some missing depth values. Nevertheless, the 1/4 resolution Ray-SD is still able to prevent perceivable halos in these areas.

5. Conclusion

We presented a novel algorithm for the creation of a stochastic depth map via hardware ray tracing. We introduced a better stochastic collection algorithm and a ray interval optimization to improve the quality of the resulting depth samples. Furthermore, we showed that a low resolution SD-map can be used to improve the performance without major artifacts. We presented a radius cutoff to reduce the number of required rays for the SD-map which can improve the performance without a noticeable loss in quality. Finally, we introduced a method that handles out-of-screen depth values more efficiently than a conventional guard band. This is achieved by augmenting our 1/4 resolution SD-map with a guard band that is evaluated only on demand.

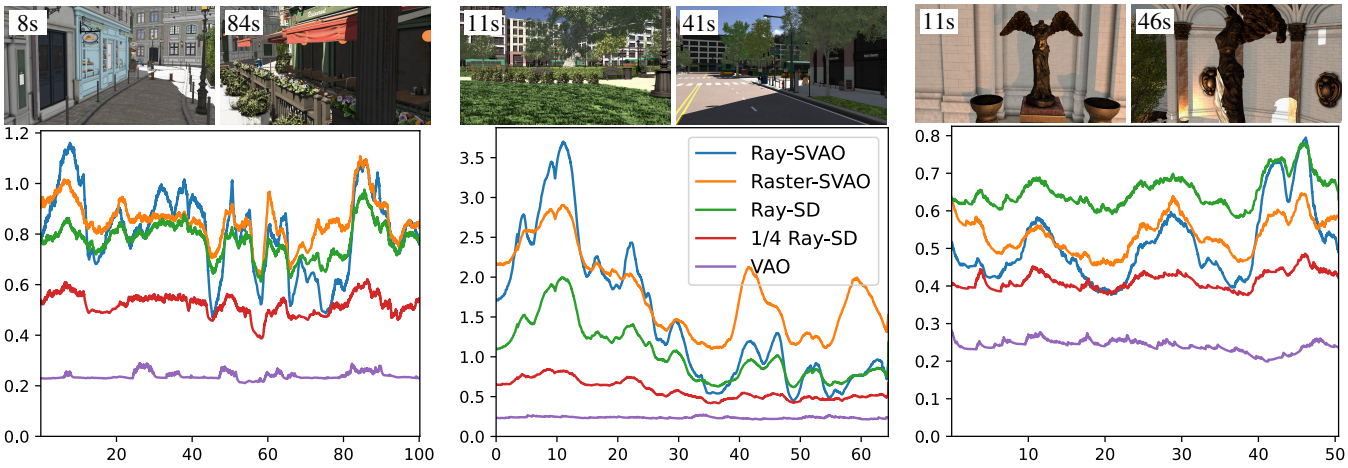
Our algorithm is able to outperform the previous methods for large scenes and appears to remain relatively stable in terms of performance. We believe that our algorithm can already be used in production to improve the quality of AO in real-time applications.

Most of our optimizations can also be applied to the raster based SD-map, which mainly improves its quality. The lower resolution versions of the SD-map are also compatible with the rasterizer, but the per-pixel jitter is not, which results in stronger aliasing. Furthermore, the rasterizer does not always benefit as much from the lower resolution as the ray tracer, since the rasterizer still needs to process all vertices in the vertex shader.

6. Future Work

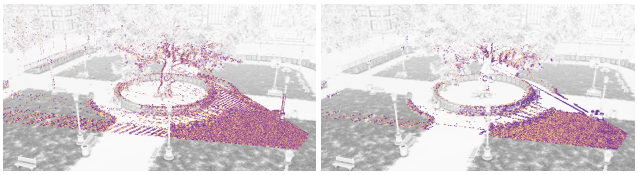
Some aliasing issues are caused by the 4x4 rotated sampling pattern of the underlying VAO algorithm [BKG22]. This can be very noticeable in some camera motion paths, but it is a very common artifact that is also present in the Unreal Engine and other screen space AO techniques.

The remaining aliasing issues of the low resolution SD-map could be addressed with an adaptive resolution.



(a) *Bistro* **(b)** *Emerald Square* **(c)** *Sun Temple*

Figure 15: *Rendering times of our Ray-SD with full and 1/4 resolution SD-map vs the previous SVAO techniques. The graphs plot the rendering time in ms on the y-axis and the scene time of the camera path on the x-axis. The images on top were recorded with our 1/4 Ray-SD and are without noticeable artifacts.*



(a) *Coverage Mask [VSE21] + SD-mask [BKG22]* **(b)** *(Our) Reservoir Sampling + (Our) Ray Interval Optimization*

Figure 16: *Overview of samples in the SD-map for the 11s camera path in Emerald Square. Our improved SD-map stores fewer, but more relevant, samples. The samples are colored based on their slot in the SD-map. The self-occlusion of the grass accounts for a significant portion of the samples.*

The radius cutoff could also be extended to depend on other factors such as the diffuse color of the object, since darker objects are less likely to produce visible halos. The overall type of object might also be a factor, since halos are often more difficult to notice on dense foliage.

The performance of our SD guard band could be further improved by choosing an even lower resolution for the area outside the screen.

AO shadows can still disappear if the camera's view direction is parallel to the occluder (see [BKG22] Fig. 14). This issue is challenging to address because it requires a ray to track whether it is inside an object, which only works if the scene is completely watertight. Furthermore, this limitation would prevent us from using our interval optimization.

We also tested our algorithm with Horizon Based Ambient Occlusion (HBAO) [BSD08] and found it to be compatible with our Ray-SD map (see Appendix). However, a specialized implementation for HBAO and other AO algorithms would be of interest.

Supplementary Material

The source code, an executable demo and additional videos can be found on our Github project page: https://github.com/TU-Clausthal-Rendering/Ray-Traced-Stochastic-Depth-Map.

Acknowledgements

This work was partially supported by the German Research Foundation (DFG) grant GR 3833/4-1, Project Nr. 524961573.

References

[BJ13] BAVOIL, LOUIS and JANSEN, JON. "Particle Shadows & Cache-Efficient Post-Processing". *Proceedings of the Game Developers Conference (GDC)*. Session on Advanced Visual Effects with DirectX 11. NVIDIA. 2013. URL: https://developer.nvidia.com/gdc-2013 10.

[BKG22] BRÜLL, FELIX, KERN, RENÉ, and GROSCH, THORSTEN. "Stenciled Volumetric Ambient Occlusion". *Eurographics Symposium on Rendering*. Ed. by GHOSH, ABHIJEET and WEI, LI-YI. The Eurographics Association, 2022. DOI: 10.2312/sr.20221153 1, 2, 4, 7, 8.

[BSD08] BAVOIL, LOUIS, SAINZ, MIGUEL, and DIMITROV, ROUSLAN. "Image-space horizon-based ambient occlusion". *ShaderX7 Advanced Rendering Techniques*. July 2008. DOI: 10.1145/1401032.1401061 2, 8.

[Bun05] BUNNELL, MICHAEL. "Dynamic Ambient Occlusion and Indirect Lighting". *GPU Gems 2*. Ed. by PHARR, MATT. Addison-Wesley, 2005, 223–233. URL: https://developer.nvidia.com/gpugems/gpugems2/part-ii-shading-lighting-and-shadows/chapter-14-dynamic-ambient-occlusion-and 2.

[Eve01] EVERITT, CASS W. "Interactive Order-Independent Transparency". 2001. URL: https://api.semanticscholar.org/CorpusID:5813703 1.

[Gam17] GAMES, EPIC. *Unreal Engine Sun Temple, Open Research Content Archive (ORCA)*. Oct. 2017. URL: https://developer.nvidia.com/ue4-sun-temple 4.

[HSEE15] HENDRICKX, QUINTJIN, SCANDOLO, LEONARDO, EISEMANN, MARTIN, and EISEMANN, ELMAR. "Adaptively Layered Statistical Volumetric Obscurance". HPG '15. Los Angeles, California: Association for Computing Machinery, 2015, 77–84. DOI: 10.1145/2790060.2790070 2.

[JWPJ16] JIMENEZ, JORGE, WU, XIAN-CHUN, PESCE, ANGELO, and JARABO, ADRIAN. "Practical Real-Time Strategies for Accurate Indirect Occlusion". *SIGGRAPH 2016 Courses: Physically Based Shading in Theory and Practice*. 2016. URL: https://www.activision.com/cdn/research/Practical_Real_Time_Strategies_for_Accurate_Indirect_Occlusion_NEW%20VERSION_COLOR.pdf 2.

[KCK*22] KALLWEIT, SIMON, CLARBERG, PETRIK, KOLB, CRAIG, et al. *The Falcor Rendering Framework*. https://github.com/NVIDIAGameWorks/Falcor. Aug. 2022. URL: https://github.com/NVIDIAGameWorks/Falcor 4.

[KL05] KONTKANEN, JANNE and LAINE, SAMULI. "Ambient Occlusion Fields". I3D '05. Washington, District of Columbia: Association for Computing Machinery, 2005, 41–48. DOI: 10.1145/1053427.1053434 2.

[LS10] LOOS, BRADFORD JAMES and SLOAN, PETER-PIKE. "Volumetric Obscurance". I3D '10. Washington, D.C.: Association for Computing Machinery, 2010, 151–156. DOI: 10.1145/1730804.1730829 1, 2, 10.

[Lum17] LUMBERYARD, AMAZON. *Amazon Lumberyard Bistro, Open Research Content Archive (ORCA)*. July 2017. URL: http://developer.nvidia.com/orca/amazon-lumberyard-bistro 4.

[Mic19] MICROSOFT. *Raytracing Real-Time Denoised Ambient Occlusion*. https://github.com/microsoft/DirectX-Graphics-Samples/tree/master/Samples/Desktop/D3D12Raytracing/src/D3D12RaytracingRealTimeDenoisedAmbientOcclusion. 2019 1.

[Mit07] MITTRING, MARTIN. "Finding next gen: CryEngine 2." *SIGGRAPH Courses*. Ed. by MCMAINS, SARA and SLOAN, PETER-PIKE. ACM, 2007, 97–121. URL: http://dblp.uni-trier.de/db/conf/siggraph/siggraph2007courses.html#Mittring07 2.

[MMNL16] MARA, MICHAEL, MCGUIRE, MORGAN, NOWROUZEZAHRAI, DEREK, and LUEBKE, DAVID. "Deep G-Buffers for Stable Global Illumination Approximation". HPG '16. June 2016, 11. URL: https://casual-effects.com/research/Mara2016DeepGBuffer/index.html 1.

[NAM*17] NALBACH, OLIVER, ARABADZHIYSKA, ELENA, MEHTA, DUSHYANT, et al. "Deep Shading: Convolutional Neural Networks for Screen Space Shading". *Computer Graphics Forum* (2017). ISSN: 1467-8659. DOI: 10.1111/cgf.13225 1.

[NB17] NICHOLAS HULL, KATE ANDERSON and BENTY, NIR. *NVIDIA Emerald Square, Open Research Content Archive (ORCA)*. July 2017. URL: http://developer.nvidia.com/orca/nvidia-emerald-square 4.

[SA07] SHANMUGAM, PERUMAAL and ARIKAN, OKAN. "Hardware Accelerated Ambient Occlusion Techniques on GPUs". I3D '07. Seattle, Washington: Association for Computing Machinery, 2007, 73–80. DOI: 10.1145/1230100.1230113 2.

[Sal13] SALVI, MARCO. "Pixel synchronization: solving old graphics problems with new data structures". *Advances in Real-time Rendering* (2013). URL: https://advances.realtimerendering.com/s2013/2013-07-23-SIGGRAPH-PixelSync.pdf 1.

[Sob67] SOBOL', I.M. "On the distribution of points in a cube and the approximate evaluation of integrals". *USSR Computational Mathematics and Mathematical Physics* 7.4 (1967), 86–112. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(67)90144-9 3.

[TP16] TATARINOV, ANDREI and PANTELEEV, ALEXEY. "Advanced Ambient Occlusion Methods for Modern Games". *Game Developer Conference*. https://developer.download.nvidia.com/gameworks/events/GDC2016/atatarinov_alpanteleev_advanced_ao.pdf Accessed: 2024-04-03. 2016 2, 10.

[Vit85] VITTER, JEFFREY S. "Random Sampling with a Reservoir". *ACM Trans. Math. Softw.* 11.1 (Mar. 1985), 37–57. DOI: 10.1145/3147.3165 2.

[VSE21] VERMEER, JOP, SCANDOLO, LEONARDO, and EISEMANN, ELMAR. "Stochastic-Depth Ambient Occlusion". I3D '21 4.1 (Apr. 2021). DOI: 10.1145/3451268 1, 2, 8.

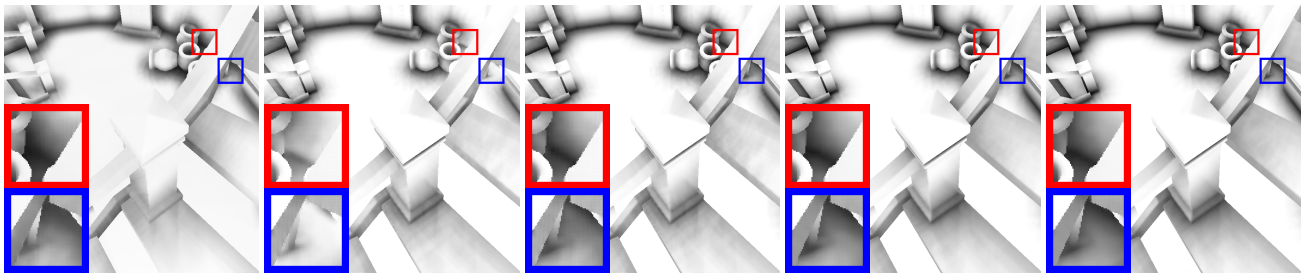
[WM19] WYMAN, CHRIS and MCGUIRE, MORGAN. "Improved Alpha Testing Using Hashed Sampling". *IEEE Transactions on Visualization and Computer Graphics* 25.2 (2019), 1309–1320. DOI: 10.1109/TVCG.2017.2739149 2.

[WZZ*23] WANG, JIAYI, ZHOU, FAN, ZHOU, XIANG, et al. "AO-Net: Efficient Neural Network for Ambient Occlusion". *Graphics Interface 2023*. 2023. URL: https://openreview.net/forum?id=b-3cQ_h4kqz 1.

[Zhd21] ZHDAN, DMITRY. "ReBLUR: A Hierarchical Recurrent Denoiser". *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*. Ed. by MARRS, ADAM, SHIRLEY, PETER, and WALD, INGO. Berkeley, CA: Apress, 2021, 823–844. DOI: 10.1007/978-1-4842-7185-8_49 1.

[ZIK98] ZHUKOV, SERGEY, IONES, ANDREI, and KRONIN, GRIGORIJ. "An Ambient Light Illumination Model." *Rendering Techniques*. Ed. by DRETTAKIS, GEORGE and MAX, NELSON L. Eurographics. Springer, 1998, 45–56. URL: http://dblp.uni-trier.de/db/conf/rt/rt1998.html#ZhukovIK98 2.

[ZXL*20] ZHANG, DONGJIU, XIAN, CHUHUA, LUO, GUOLIANG, et al. "DeepAO: Efficient Screen Space Ambient Occlusion Generation via Deep Network". *IEEE Access* 8 (2020), 64434–64441. DOI: 10.1109/ACCESS.2020.2984771 1.



(a) *VAO 8SPP + 1/4 Ray-SD (0.54ms)* **(b)** *HBAO 8SPP (0.28ms)* **(c)** *HBAO 8SPP + 1/4 Ray-SD (0.63ms)* **(d)** *HBAO 16SPP + 1/4 Ray-SD (0.98ms)* **(e)** *Reference 16SPP (1.74ms)*

Figure 17: *AO comparison with VAO, HBAO and our Ray-SD map. The first three images use 8 AO samples per pixel, while the last two use 16 AO samples per pixel. The reference traces all missing depth values (Similar to Ray-SVAO). Notice the increased banding artifacts on the staircase in (b) and (c). Using 16 SPP reduces the banding artifacts, but also increases the rendering time significantly.*

7. Appendix

7.1. Other AO Kernels

We implemented the SD-map using VAO [LS10] as the AO kernel. However, other AO kernels can be utilized with minor adjustments.

As a proof of concept, we replaced the VAO kernel with the HBAO+ kernel [TP16] and adjusted the sample distribution accordingly. For reference, a single AO sample a_i in HBAO+ is calculated as follows:

$$a_i = \max\left(0,\ N \cdot \frac{V(S_i)}{|V(S_i)|} - 0.1\right) \max\left(0,\ 1 - \frac{|V(S_i)|^2}{r^2}\right) \qquad (1)$$

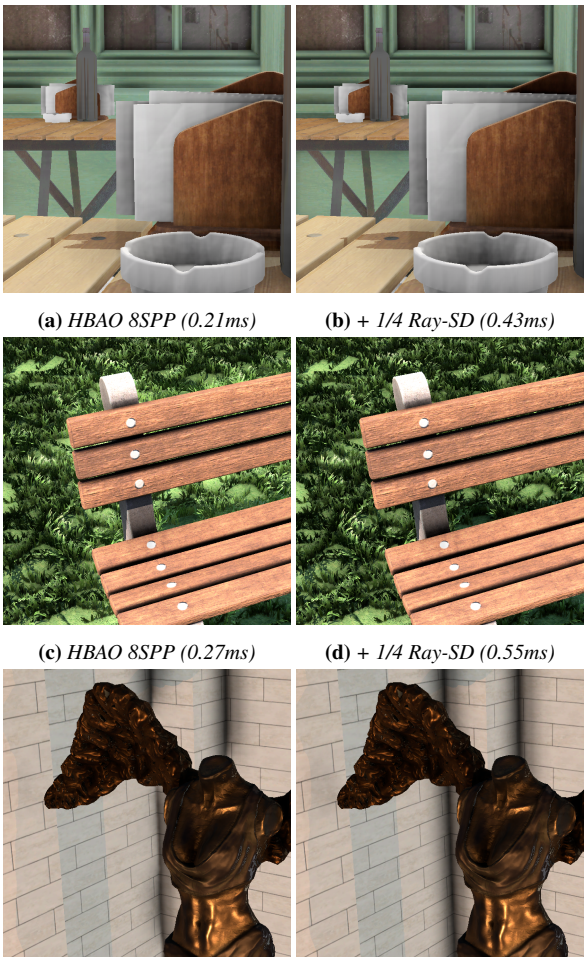
where N is the normal vector at the shading point, V is the vector from the shading point to the sample point S_i, and r is the radius of the sampling hemisphere. Previously, we used $r = 0.2$ for VAO, but we set $r = 0.3$ for HBAO+ to achieve a similar appearance.

First, we examined the HBAO+ kernel with 8 **samples per pixel (SPP)** and compared it to our implementation with VAO. Switching the AO kernel to HBAO+ did not change the runtime behavior significantly. Overall, the rendering times were not significantly slower than before, being up to 20% slower depending on the scene. Results with 8 SPP are shown in Fig. 17c and Fig. 18.

We noticed increased banding artifacts with the 8 SPP version in the Tower scene, so we tested a 16 SPP version as well (see Fig. 17d). This version does not exhibit significant banding issues, but it took more than 50% longer to render. Surprisingly, the time to ray trace the SD-map was almost the same in both versions.

We concluded that the increased rendering time is due to an increased number of texture fetches (and cache misses), a well-known problem for unoptimized HBAO implementations. This problem can be mitigated by utilizing interleaved rendering [BJ13], which we leave for future work.

Generally, all AO kernels that work with a single depth buffer can be used with our Ray-SD map. For the ray interval optimization, the AO kernel must be limited to a specified area. It appears that the 1/4 resolution Ray-SD map can keep the overall ray tracing times minimal due to its low resolution, even if the AO kernel itself requests slightly more SD samples, which was the case for the HBAO+ kernel.



(a) *HBAO 8SPP (0.21ms)* **(b)** *+ 1/4 Ray-SD (0.43ms)*

(c) *HBAO 8SPP (0.27ms)* **(d)** *+ 1/4 Ray-SD (0.55ms)*

(e) *HBAO 8SPP (0.25ms)* **(f)** *+ 1/4 Ray-SD (0.46ms)*

Figure 18: *More examples of HBAO with and without our Ray-SD map (left vs right). From top to bottom: Bistro, Emerald Square and Sun Temple.*