Stochastic-Depth Ambient Occlusion

JOP VERMEER, Delft University of Technology, the Netherlands
LEONARDO SCANDOLO, Delft University of Technology, the Netherlands
ELMAR EISEMANN, Delft University of Technology, the Netherlands

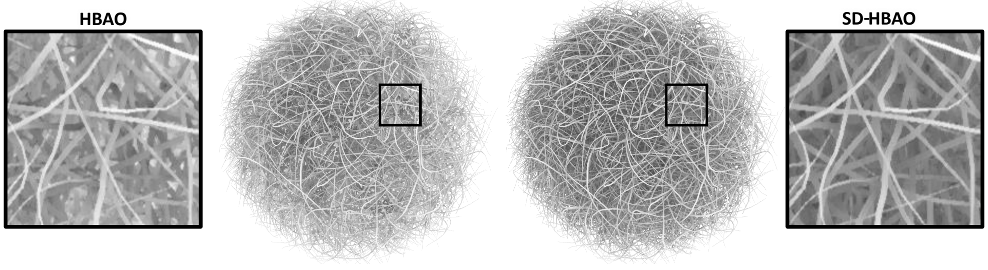

Fig. 1. *Left:* Horizon-based ambient occlusion (HBAO) uses only a depth map and underestimates occlusion due to hidden geometry. *Right:* Our stochastic-depth HBAO captures occluded geometry stochastically (2ms in full HD).

Ambient occlusion (AO) is a popular rendering technique that enhances depth perception and realism by darkening locations that are less exposed to ambient light (e.g., corners and creases). In real-time applications, screen-space variants, relying on the depth buffer, are used due to their high performance and good visual quality. However, these only take visible surfaces into account, resulting in inconsistencies, especially during motion. *Stochastic-Depth Ambient Occlusion* is a novel AO algorithm that accounts for occluded geometry by relying on a *stochastic depth map*, capturing multiple scene layers per pixel at random. Hereby, we efficiently gather missing information in order to improve upon the accuracy and spatial stability of conventional screen-space approximations, while maintaining real-time performance. Our approach integrates well into existing rendering pipelines and improves the robustness of many different AO techniques, including multi-view solutions.

CCS Concepts: • **Computing methodologies → Image-based rendering**; *Rasterization.*

Additional Key Words and Phrases: Ambient occlusion, stochastic transparency, screen-space techniques

ACM Reference Format:
Jop Vermeer, Leonardo Scandolo, and Elmar Eisemann. 2021. Stochastic-Depth Ambient Occlusion. *Proc. ACM Comput. Graph. Interact. Tech.* 4, 1 (May 2021), 15 pages. https://doi.org/10.1145/3451268

1 INTRODUCTION

Realistic global illumination is a hard problem in the field of computer graphics. Real-time applications, with a budget of milliseconds per frame, have to rely on approximations to represent indirect lighting. A popular technique to approximate these phenomena is *ambient occlusion* (AO). It can

Authors' addresses: Jop Vermeer, Delft University of Technology, Delft, the Netherlands, j.vermeer@bulletwhale.games; Leonardo Scandolo, Delft University of Technology, Delft, the Netherlands, l.scandolo@tudelft.nl; Elmar Eisemann, Delft University of Technology, Delft, the Netherlands, e.eisemann@tudelft.nl.

help a viewer better understand the scene's geometry because it captures geometric proximity by modulating the amount of ambient light reaching a point by its accessibility [Zhukov and Iones 1998], i.e., the portion of directions from which light can reach the point. Although not physically accurate, it is comparatively fast to compute and yields convincing results in practice, leading to a widespread use [Landis 2002].

Most real-time implementations rely on *screen-space* AO [McGuire et al. 2011], using the depth buffer as only input. This reduces the complexity of the problem and leads to high performance on GPUs. Nevertheless, the loss of information when working only with a 2D projection of the geometry results in artifacts. A depth map captures only the first visible surfaces in a scene and cannot account for occluded objects. For AO computations, it means that a point's accessibility is often underestimated because hidden geometry is ignored. This results in AO *pop-in*; as the camera moves, occluded objects come into view, inducing sudden changes.

In this work, we propose a novel screen-space AO algorithm that can account for occluded objects in order to avoid pop-in artifacts via the use of a *stochastic depth map*. Inspired by the stochastic transparency algorithm [Enderton et al. 2010], we efficiently create a depth map, where each pixel contains (potentially hidden) surfaces from random depth layers. We compute AO values using modified versions of well-known AO algorithms, taking this modified input into account.

Furthermore, our method can be extended to multiple views to avoid artifacts by polygons perpendicular to the camera view, which are not represented in the view's depth map. While conventional multiple viewpoint AO methods struggle with camera placement, since it is hard to ensure that additional cameras are not occluded by scene geometry, our stochastic depth map effectively *sees through* geometry, simplifying this placement.

2 RELATED WORK

The idea of modulating light based on local geometry was introduced by Zhukov et al. [Zhukov and Iones 1998], and Landis et al. [Landis 2002] in the context of ray tracing. Pharr and Green [Pharr and Green 2004] propose to use precomputation in static scenes for use in real-time rendering. Sloan et al. [Sloan et al. 2002] and Ren et al. [Ren et al. 2006] used spherical proxies to simplify scene geometry and store accessibility as spherical harmonics.

Bunnell [Bunnell 2005] presented a real-time method that generates discs at vertex locations, which are used to approximate occlusion. Shanmugam et al. [Shanmugam and Arikan 2007] proposed to splat proxy spheres to a screen buffer which are then sampled to compute AO. Mittring [Mittring 2007] relied on a sample-based screen-space method using only the depth buffer. McGuire et al. [McGuire et al. 2012, 2011] improved this solution via artist-adjustable parameters, low-bitrate values, variable sampling rates, and multiple LODs. Hoang et al. [Hoang and Low 2012] presented a multi-resolution sampling approach with temporal filtering. In *horizon-based ambient occlusion* (HBAO) [Bavoil et al. 2008] the depth buffer is sampled along a few directions to find the horizon angle under which all directions are assumed occluded. This method was later extended [Jiménez et al. 2016; Tatarinov and Panteleev 2016] to improve the approximation in the presence of thin occluders. Other works [Hendrickx et al. 2015; Loos and Sloan 2010] proposed a statistical volumetric approximation to compute occlusion, which avoids the bilateral filtering step commonly employed in sampling-based methods.

To capture invisible geometry, the scene can be voxelized on-the-fly [Reinbothe et al. 2009], using a single-pass voxelization algorithm [Eisemann and Décoret 2006], and ray-traced AO values based on the voxel grid. Nevertheless, the method remains costly compared to screen-space alternatives. For real-time applications, the voxel grid needs to be of low resolution, which results in a loss of high-frequency details and severe shortcomings in the presence of thin occluders. Kroes et al. [Kroes et al. 2015] proposed a simplified method for voxel-based ambient occlusion for volume rendering.

Ambient occlusion fields [Kontkanen and Laine 2005] are volumes surrounding each object that directly store the occlusion caused by this object. *Ambient occlusion volumes* [Laine and Karras 2010; McGuire 2010] extended this approach for use with dynamic geometry, by creating per-triangle volumes similar to *shadow-volume* techniques [Crow 1977]. While these techniques can provide higher quality ambient occlusion than screen-space algorithms, they are less efficient due to the overdraw created by the large amount of rendered volumes.

Multi-view ambient occlusion [Vardis et al. 2013] enhances traditional screen-space algorithms using information from multiple viewpoints, either from other available depth buffers such as shadow maps, or dedicated views. In complex scenes with many occluders, secondary buffers may not capture enough information for an accurate AO estimation, therefore the effectiveness of this approach is scene dependent, but its performance is suitable for real-time applications.

Another approach is to extend screen-space ambient occlusion using transparency techniques. Some techniques compute and store all depth layers from the main camera viewpoint in per-pixel lists. This can be accomplished via depth peeling [Bavoil and Sainz 2009; Ritschel et al. 2009] or A-buffer [Carpenter 1984; Yang et al. 2010] construction [Bauer et al. 2013], where all layers are recorded and the one providing maximum occlusion is used. Although these techniques achieve good results, their efficiency is limited by the amount of recorded depth layers, making them difficult to apply for real-time applications, especially in complex scenes. To overcome this limitation, Mara et al. [Mara et al. 2016] and the HBAO+ algorithm [Tatarinov and Panteleev 2016] restrict the computation to only two depth layers. The disadvantage of using a fixed amount of depth layers is that it does not work well for every scene, e.g., scenes with many obstructing objects could still show view-dependent artifacts. A slightly different approach is to create deep buffers [Nalbach et al. 2014], where only part of the scene is captured using proxies known as *surfels*. These work well in practice, but the splatting procedure can be costly, especially if done only for AO.

Real-time ray tracing is also an alternative for obtaining access to the entire scene geometry when computing AO [Gautron 2020]. Nevertheless, this requires the construction of acceleration structures for all dynamic parts of the scene at each frame. This severely limits its applicability for real-time applications, especially in the presence of shader-based animation techniques (e.g. skeleton rigging), since they do not map well to ray-traced solutions, but are handled seamlessly by image-based solutions.

3 STOCHASTIC-DEPTH AMBIENT OCCLUSION

In this section, we will present our AO solution. We start with the characteristics and creation procedure of our stochastic depth map (Sec. 3.1), which serves as input to the AO computation. Then, we explain how to use it in a screen-space algorithm by extending popular single-layer ambient occlusion techniques (Sec. 3.2) and introduce a simple heuristic to decrease computational costs (Sec. 3.3). Finally, we extend our method using a second viewpoint to avoid artifacts arising from view-perpendicular polygons (Sec. 3.4).

3.1 Stochastic depth map

The stochastic depth map is a multisample texture, containing one or more depth values per pixel, corresponding to random scene surfaces mapping to the pixel. In previous work [Bauer et al. 2013; Bavoil and Sainz 2009; Liu et al. 2013], only the first n visible layers have been used for this purpose, through depth-peeling or the usage of A-buffers. These methods either require multiple render passes, or have large and unbounded memory requirements to achieve this goal. In contrast, we revisit ideas from stochastic transparency, commonly used for *order-independent*

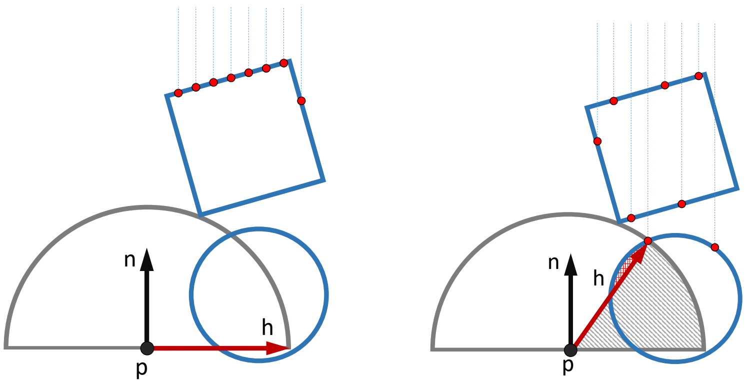

Fig. 2. Standard HBAO (left) uses samples from the first layer, which misses occlusions, while stochastic-depth samples (right) capture them much better.

transparency (OIT) [Enderton et al. 2010]. Instead of an ordered list of the first visible layers, a stochastic depth-map pixel contains a random subset of the scene layers. To efficiently capture these layers, we use a multisample texture, where all sample locations are set to the pixel center. Hereby, we can store all samples in a single render pass.

The creation of this map is similar to rendering a regular depth map, and is thus simple and fast. The only difference lies in the fragment shader, where we randomly discard fragments for each sample based on a global transparency value α. The fastest and easiest solution to generate stochastic samples is to use a hashing function in the fragment shader, using the pixel location, sample index, and sample depth, to generate a uniformly distributed random number between 0 and 1. The value is compared to α in order to decide on storing or discarding it. In order to avoid over or under representation of certain layers, a more sophisticated method can be used. In practice, we use a stratified sampling approach similar to that of stochastic transparency [Enderton et al. 2010], where we keep S per-pixel samples (typically S is between 1 and 16). An incoming fragment f_i will be stored in R_i of these pixel samples, where we compute R_i as:

$$R_i = \lfloor \alpha S + \xi \rfloor \tag{1}$$

Here, ξ is a uniformly distributed random number between 0 and 1. This formula implies that the fragment is expected to be stored in αS samples. To determine in which R_i samples we store f_i, we precompute all possible subsets of R_i samples out of the total of S samples in a lookup table. The table contains S-bit numbers with R_i active bits. At run time, we retrieve one such number for each fragment-pixel combination and if its jth bit is set, we store fragment f_i in sample j and discard f_i otherwise. Algorithm 1 shows pseudocode of the stochastic-depth fragment shader, which uses stratified sampling. We found that global α values between 0.2 and 0.5 work well in most situations and use 0.2 for all tests in Sec. 4.

Alg. 1 runs once per fragment, deciding which set of samples will discard the depth value by way of setting a built-in sample mask (`gl_SampleMask` in OpenGL). Since we are creating a depth map, we can take advantage of hardware-supported depth testing to reduce overdraw and increase efficiency. Therefore, for samples that do not discard a fragment, the corresponding depth value is still subject to the regular depth test. While this biases the samples towards the closest layers, these are often the most relevant for AO values. Finally, since we capture positions, we linearize the depth value before storage.

Algorithm 1 Stratified stochastic depth

```
1:  $S \leftarrow$ number of samples
2:  $i \leftarrow$ current fragment
3:  $z_i \leftarrow$ depth of fragment $i$
4:  $\xi \leftarrow$ random number between 0 and 1, using $i$ and $z_i$ as seed
5:  $R_i \leftarrow \lfloor \alpha S + \xi \rfloor$
6:  if $R_i == S$ then                          ▷ Trivial case, all samples stored
7:      coverage mask ← 0xffff
8:  else if $R_i == 0$ then                     ▷ Trivial case, no samples stored
9:      discard fragment
10: else                                        ▷ Store a random precomputed subset of size $R_i$
11:     coverage mask ← random subset mask with $R_i$ set bits
```

One caveat of stochastically discarding fragments when generating the stochastic depth map is that early-z fragment culling cannot be performed by the GPU, resulting in decreased performance. Regular depth map creation times are 20% to 50% faster in our tests than 1-sample stochastic depth map creation times (see Table 2). Furthermore, in most real-time applications, CPU culling is performed in order to avoid issuing render calls for parts of the scene which are not visible from the current viewpoint. Using a stochastic depth map would require modifications to such culling schemes to avoid culling parts of the scene within the camera frustum that would be occluded by the first visible layers. When considering culling strategies in combination with our method, distance criteria can be considered, as nearby AO is of higher relevance than far away. Nevertheless, adapting culling strategies generally remains future work.

3.2 Ambient occlusion computation

For a point p with normal n_p, AO is typically computed as:

$$AO = 1 - \frac{1}{\pi}\int_{\Omega}(n_p \cdot \omega)V(p,\omega)\, d\omega \tag{2}$$

Here, Ω is the normal-facing hemisphere at point p and $V(p,\omega)$ is the visibility function, returning values between 0 and 1, representing the occlusion of p in direction ω. Once the stochastic depth map is computed, we can integrate it in different screen-space AO techniques and implemented solutions for SSAO [McGuire et al. 2011], HBAO [Bavoil et al. 2008], and HBAO+ [Nvidia 2013].

Stochastic-depth SSAO. SSAO is a simple and effective approximation. With respect to Eq. 2, the hemisphere is translated to a screen-space area (a circle or a box) around the screen-space position of point p. Instead of sampling individual rays, locations are chosen from this region. The depth sample at this location results in a contribution that is one if the sample is within the hemisphere around p and at a distance closer than a given threshold, otherwise, it contributes with zero. Typically, a smooth distance-based falloff function is used in practice and the weight of the sample is modulated by the cosine of the angle of its position with respect to the normal.

To integrate our stochastic depth map, we sample all its depth layers when a screen-space location around p is selected. Small optimizations can be done in the world-space translation due to the samples sharing the same screen-space location.

Stochastic-depth HBAO(+). We also extend *Horizon-Based Ambient Occlusion* (HBAO) [Bavoil et al. 2008], and HBAO+[Nvidia 2013], two well-known screen space AO techniques. Conceptually, for a point p with normal n_p, HBAO computes the maximum horizon angle h_θ for all directions θ

perpendicular to n_p. Thus, one can walk along the direction θ and test the stored depth samples and assume that all rays passing underneath the encountered sample are blocked:

$$AO = 1 - \frac{1}{2\pi} \int_0^{2\pi} \int_{h_\theta}^{\pi} \cos(h)\, dh\, d\theta \tag{3}$$

In practice, the outer integral of Eq. 3 is computed via a Monte-Carlo approximation, where we only compute the result for a small set of N_d directions. Along each direction we take N_s samples from the depth map and keep track of the highest horizon-angle found. This horizon-angle is then used to compute the inner integral, which analytically has the form $1 - \sin(h_\theta)$. Since AO is a local effect with a user-defined radius r, depth samples beyond that radius are discarded. Finally, a distance-based attenuation function W can be used to decrease the impact of far-away occluders.

For our extension, at each step, we examine all samples, and only keep the one within the user-set radius that would result in the maximum occlusion, taking the angle and attenuation function W into account. Algorithm 2 shows the pseudocode and Fig. 2 shows a graphic representation of the improvement achieved by using stochastic depth samples over standard HBAO.

HBAO+ is another variant of HBAO but uses a simpler approximation to compute the occlusion value. It also works by exploring perpendicular directions to a point, but it does not attempt to locate a horizon angle. Instead, once a sample s is obtained, $s - p$ is projected to the normal direction and modulated via the standard distance weight function. This value is averaged over all samples within the given radius, in order to avoid overestimating the AO contribution of thin occluders. For our extension, we simply include all stochastic samples in the HBAO+ calculation.

3.3 Sampling efficiency

Retrieving all depth samples from the stochastic depth buffer when computing HBAO and HBAO+ can be costly. In simple geometric configurations (e.g. a flat surface), the depth buffer already contains all the necessary information. Therefore, when sampling a screen-space location, we first query the depth buffer and decide whether the stochastic depth buffer needs to be accessed. Since all samples in both depth buffers share the same screen-space coordinates, the regular depth map sample contains the lowest depth and would result in the largest angle to the tangent vector. Although the falloff function could result in a slightly higher AO value for larger depths for some geometric configurations, we found no visible difference in our results, and thus opt to forego querying the stochastic samples when the regular depth map sample is within the AO radius in order to improve runtime performance. Similarly, when computing SSAO and have chosen a sampling location, we do not access the samples in the stochastic depth map if the regular depth map sample already represents an occluder.

3.4 Multiple viewpoints

A common problem of screen-space methods are flat surfaces almost aligned with the camera view direction, as the projection of these triangles does not result in any (or very few) fragments, meaning that they are ignored when using screen-space computations. For AO methods, missing occlusion (as seen in Fig. 3) can suddenly appear when the view direction changes.

To avoid this artifact, we propose to use a secondary stochastic depth map at an angle from the main viewpoint, hence, better capturing aligned surfaces. In previous work [Ritschel et al. 2009; Vardis et al. 2013], multiple-viewpoint AO suffered when geometry remained occluded. Yet, it is challenging to place the secondary camera such as to avoid occlusions, which is necessary to have a contribution towards the AO computation. In contrast, using a stochastic buffer for the secondary view prevents occlusion problems.

Algorithm 2 Stochastic depth HBAO

```
 1: $p \leftarrow$ view space fragment position
 2: $n_p \leftarrow$ view space fragment normal
 3: $r' \leftarrow$ projected radius $r$ onto image plane
 4: Determine stepsize as $r'/(N_s + 1)$
 5: Determine directions with random offset
 6: $AO \leftarrow 0$
 7: for direction $d_i$ where $i = 0$ to $N_d$ do
 8:     $t_p \leftarrow$ tangent vector in direction $d_i$
 9:     $t_\theta \leftarrow$ angle of $t_p$ with $XY$-plane
10:     $h_\theta \leftarrow t_\theta + bias$                         ▷ Current horizon angle
11:     for step $s_j$ where $j = 0$ to $N_s$ do
12:         Sample $s \leftarrow$ view-space pos. at step $j$ (reg. depth buffer)
13:         $v \leftarrow s - p$
14:         $\phi \leftarrow$ angle of $v$ with $XY$-plane
15:         if $\phi < h_\theta$ then
16:             continue
17:         if $||v|| < r$ then
18:             $AO \leftarrow AO + W(||v||)(sin(\phi) - sin(h_\theta))$
19:             $h_\theta \leftarrow \phi$
20:         else
21:             $AO_{max} \leftarrow 0$
22:             $h_{max} \leftarrow 0$
23:             $sds[] \leftarrow$ stochastic depth values at current position
24:             for each depth value $k$ in $sds$ do
25:                 $s_k \leftarrow$ view space position using depth $k$
26:                 $v_k \leftarrow s_k - p$
27:                 $\phi_k \leftarrow$ angle of $v_k$ $XY$-plane
28:                 if $\phi_k > h_\theta$ and $||v_k|| < r$ then
29:                     $AO_k \leftarrow W(||v_k||)(sin(\phi_k) - sin(h_\theta))$
30:                     if $AO_k > AO_{max}$ then
31:                         $AO_{max} \leftarrow AO_k$
32:                         $h_{max} \leftarrow \phi_k$
33:             if $AO_{max} > 0$ then
34:                 $AO \leftarrow AO + AO_{max}$
35:                 $h_\theta \leftarrow h_{max}$
36: $AO \leftarrow 1 - \frac{AO}{N_d}$
37: return $AO$
```

In our implementation, we place the secondary view at a set distance to the side of the original camera, and angled such that the original camera frustum is fully covered. We extended the stochastic-depth HBAO+ algorithm to account for the extra viewpoint information (as shown in Algorithm 3), since it only requires a position p and a normal n, and unlike HBAO it does not require us to compute a tangent vector.

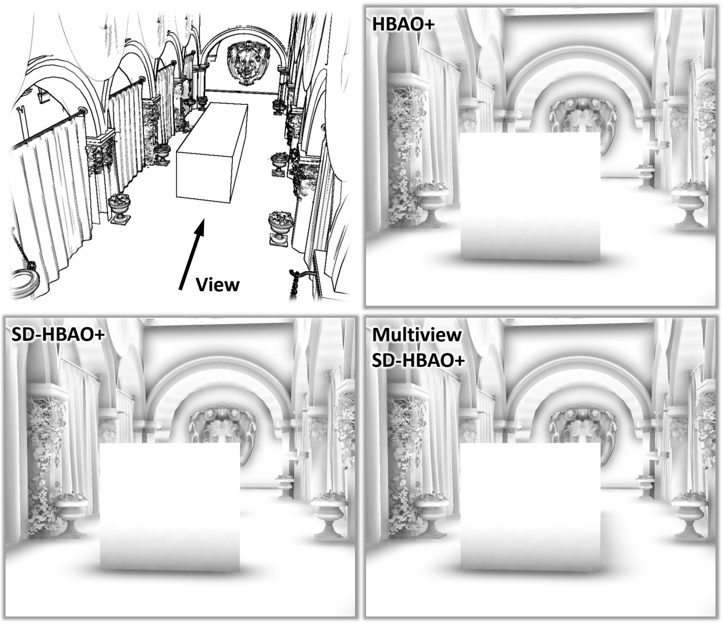

Fig. 3. With single-view screen space ambient occlusion techniques, the sides of the rectangle are under a grazing angle with the main camera, resulting in missing ambient occlusion. Using multi-view stochastic-depth ambient occlusion, we can correctly capture this missing occlusion.

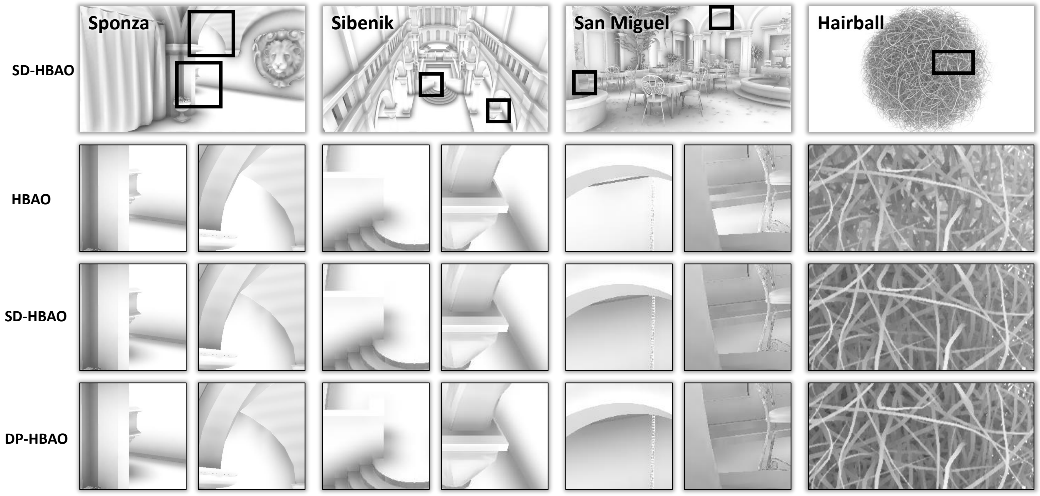

Fig. 4. Comparison of horizon-based ambient occlusion using a single depth layer, our stochastic depth map with 4 samples per pixel, and the full depth-peeled set of layers for our test scenes.

4 RESULTS

All results presented in this section were obtained at full HD (1920x1080) resolution on an Intel i7-5820K CPU with a Nvidia Titan V GPU running Windows 10. Our solution consists of two

Algorithm 3 Multi-view stochastic depth HBAO+

```
1: $p \leftarrow$ view space position fragment position
2: $n_p \leftarrow$ view space position fragment normal
3: $AO \leftarrow 0$
4: for each camera $l$ do
5:     $AO \leftarrow max(AO, ComputeAO(p, n_p, l))$
6: $AO \leftarrow 1 - \frac{AO}{N_d \cdot N_s}$
7: return $AO$

8: function ComputeAO($p, n_p, l$)
9:     $p_l \leftarrow p$ transformed to the view space of camera $l$
10:    $p_{screen} \leftarrow p_l$ projected to the screen space of camera $l$
11:    Normal $n_l \leftarrow n_p$ transformed to the view space of camera $l$
12:    Determine step size as $r'/(N_s + 1)$
13:    Determine directions with random offset
14:    $AO \leftarrow 0$
15:    for direction $d_i$ where $i = 0$ to $N_d$ do
16:        Randomly offset first step
17:        for step $s_j$ where $j = 0$ to $N_s$ do
18:            $AO_{max} \leftarrow 0$
19:            sds[] $\leftarrow$ stoch. depths (texture $l$, step $j$, dir. $d_i$)
20:            for each depth value $k$ in sds do
21:                $s_k \leftarrow$ view-space position using depth $k$
22:                $v_k \leftarrow s_k - p_l$
23:                $AO_k \leftarrow W(||v_k||)max(\frac{v_k \cdot n_l}{||v_k||} - bias, 0)$
24:                $AO_{max} \leftarrow max(AO_{max}, AO_k)$
25:            $AO \leftarrow AO + AO_{max}$
26:    return $AO$
```

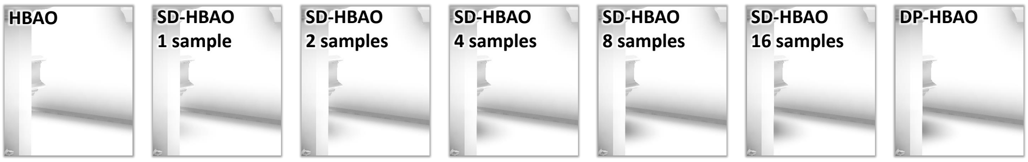

Fig. 5. Comparison of results with different stochastic sample counts for a closeup of the Sponza scene.

separate render passes: the render pass of the stochastic depth map and a screen-space AO compute pass. A conventional depth map is used by our AO compute pass, which most common pipelines offer as a result of a depth pre-pass or g-buffer generation, which is why we do not include its creation time in the results. Furthermore, a (separable) bilateral filter step is applied, which has a small fixed runtime cost (0.23ms), and is not included in the timings to better compare the cost of the core AO steps for single-layer and stochastic versions of the algorithms. The timings of the creation of the stochastic depth map is scene dependent, as the stochastic treatment of fragments can create overdraw, depending on the scene complexity. Conversely, the AO computation step is

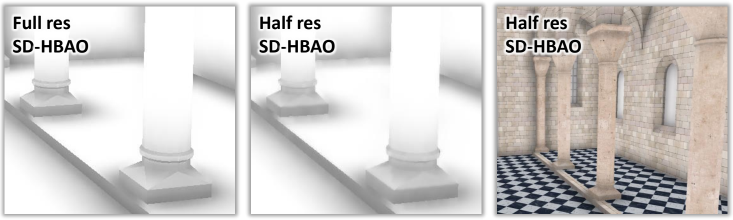

Fig. 6. Close-up comparison of our stochastic counterpart at full resolution and half resolution in the Sibenik scene. The low frequency nature of AO leads to good results even at lower resolutions.

	SSAO	HBAO	HBAO+
Regular	0.926	0.734	0.694
1 stochastic sample	1.192	1.124	1.081
2 stochastic samples	1.460	1.302	1.372
4 stochastic samples	2.161	1.762	1.978
8 stochastic samples	3.557	2.528	3.212
16 stochastic samples	5.967	4.823	5.527

Table 1. Timings (in ms) of stochastic-depth ambient occlusion (AO computation only) with the different screen-space ambient occlusion algorithms in the Sponza scene.

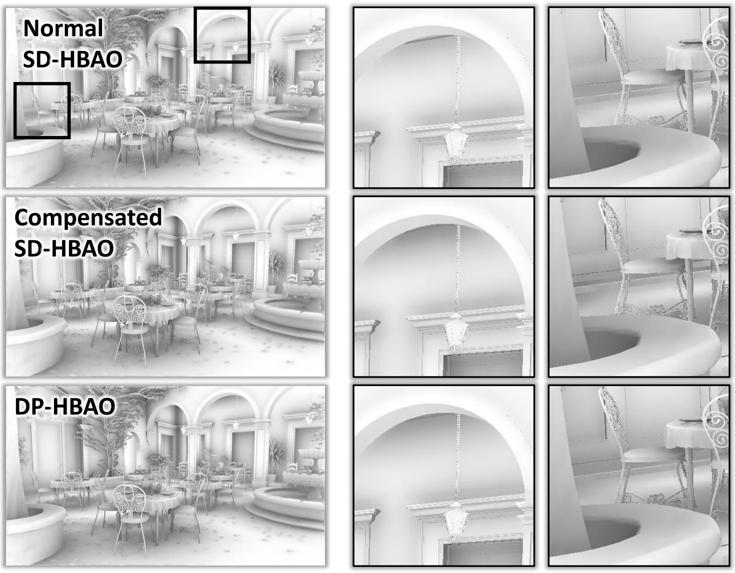

Fig. 7. Compensating for missing samples by increasing (in this example doubling) the contribution of stochastic samples can reintroduce missing occlusion. In this case, 2 stochastic samples per-pixel were used.

largely scene-agnostic, although some optimizations, as discussed in Sec. 3.3, may alter run-time performance depending on the map's complexity.

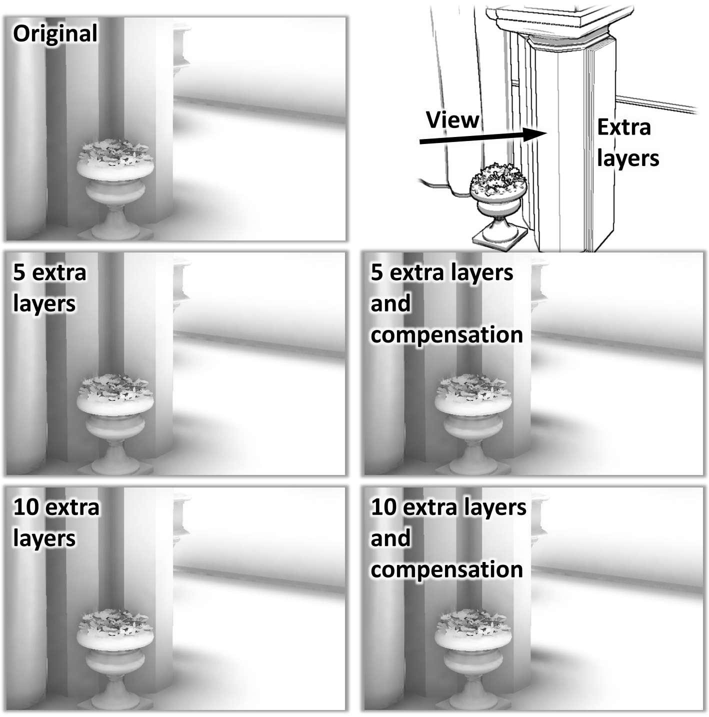
Original
View
Extra layers
5 extra layers
5 extra layers and compensation
10 extra layers
10 extra layers and compensation

Fig. 8. Adding extra layers to the pillar in the Sponza scene reduces the number of stochastic depth samples from background layers, which results in decreased AO values. Adding a compensation multiplier for AO from stochastic layers can restore some of the missing occlusion.

		Sponza	Sibenik	San Miguel	Hairball
Regular HBAO+	Full resolution	1.042	0.848	0.608	0.252
	Half resolution	0.226	0.179	0.195	0.077
1-sample SD-HBAO+	Full resolution	1.259 (0.178)	1.344 (0.096)	2.285 (0.845)	1.718 (1.213)
	Half resolution	0.382 (0.071)	0.333 (0.040)	1.127 (0.809)	0.774 (0.621)
2-sample SD-HBAO+	Full resolution	1.577 (0.205)	1.506 (0.111)	2.508 (0.862)	1.900 (1.221)
	Half resolution	0.436 (0.076)	0.412 (0.043)	1.165 (0.807)	0.841 (0.642)
4-sample SD-HBAO+	Full resolution	2.313 (0.334)	2.131 (0.157)	3.243 (0.936)	2.494 (1.540)
	Half resolution	0.622 (0.114)	0.579 (0.055)	1.312 (0.816)	1.074 (0.794)
8-sample SD-HBAO+	Full resolution	3.837 (0.625)	3.667 (0.257)	4.619 (1.107)	3.739 (2.120)
	Half resolution	1.038 (0.175)	0.979 (0.081)	1.697 (0.849)	1.452 (0.989)
16-sample SD-HBAO+	Full resolution	7.005 (1.467)	6.727 (0.660)	7.323 (1.753)	5.993 (3.117)
	Half resolution	1.922 (0.390)	1.805 (0.176)	2.473 (0.972)	2.427 (1.622)

Table 2. Timings in ms of stochastic-depth ambient occlusion with a different amount of stochastic depth samples on both full and half resolution (1920x1080 and 960x540 respectively). Numbers in parenthesis indicate the time for the creation of the stochastic depth map only, which are included in the total time.

Table 1 shows timings for different stochastic AO methods presented in Sec. 3.2. HBAO and HBAO+ variants were evaluated with 8 directions per pixel and 4 samples per direction, and SSAO variants were evaluated with 32 samples per pixel to obtain similar performance. The stochastic depth map is likewise created using stratified sampling and a common α of 0.2 for all layers. For

	Single Camera Full resolution	Multi-view Full resolution	Multi-view Half resolution	Multi-view Half resolution only sec. view
1-sample SD-HBAO+	1.344 (0.096)	2.852 (0.211)	0.615 (0.084)	2.157 (0.142)
2-sample SD-HBAO+	1.506 (0.111)	2.885 (0.226)	0.698 (0.086)	2.462 (0.156)
4-sample SD-HBAO+	2.131 (0.157)	3.836 (0.272)	0.873 (0.101)	3.116 (0.201)
8-sample SD-HBAO+	3.667 (0.257)	5.419 (0.376)	1.223 (0.125)	4.523 (0.304)

Table 3. Timings in ms of the HBAO+ variant of stochastic-depth ambient occlusion in the Sibenik scene at FullHD resolution with different multi-view and resolution configurations. Numbers in parenthesis indicate the time for the creation of the stochastic depth map only, which are included in the total time.

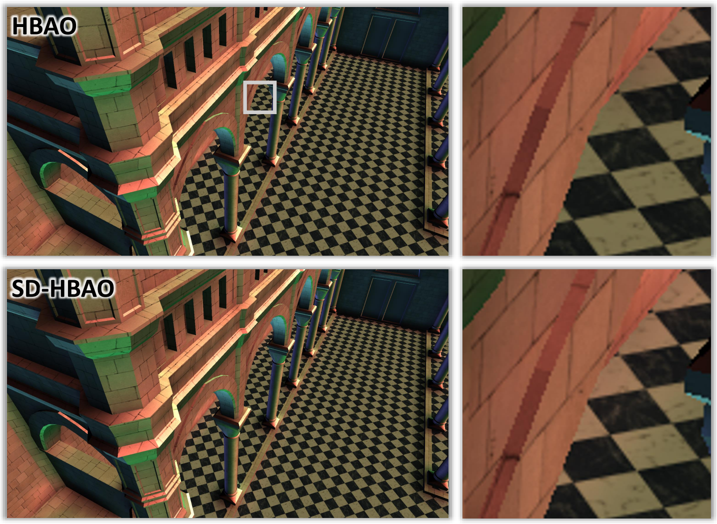

Fig. 9. Bent normals and cones computation can use stochastic samples similarly to AO computation. In this image, bent normals along the wall edge remain consistent when using stochastic depth samples, producing a red tint.

the rest of this section, we focus on timings of our stochastic-depth HBAO+ (SD-HBAO+) solution, since it usually falls between the other two variants for our configuration (Table 1). In most figures, we show HBAO results, as it produces higher AO values, which makes differences between methods and configurations easier to spot, and we refer the reader to our supplementary video for a side-by-side example of the different methods. Unless explicitly stated, the mentioned parameters were maintained for all tests in this section.

Fig. 4 shows a comparison of our stochastic-depth HBAO (SD-HBAO) using 4 samples against a standard HBAO implementation, and an HBAO implementation that precomputes all scene layers through depth peeling (DP-HBAO). This last version has access to all scene geometry and showcases a best-result scenario for screen-space AO methods.

Sample count. Table 2 shows timings of standard HBAO+ and our method in several test scenes [McGuire 2017] at full-HD and half full-HD resolution. The screen-space AO computation is largely scene independent, and scales sub-linearly with the processed samples. In contrast, the creation time of the stochastic depth map varies with scene complexity. It is higher for San Miguel and Hairball, but moderate for Sponza and Sibenik, which are more representative of typical real-time complexity. Using up to 4 stochastic samples remains an attractive alternative to traditional AO algorithms, as the cost remains between 1ms and 4ms. Higher sample counts become less practical for high-performance applications. Fig. 5 shows a comparison of the results using different stochastic sample counts for the Sponza scene. In practical applications, lower level-of-detail geometry and appropriate visibility culling can improve the creation time of the stochastic depth map, especially for objects far away from the camera. For time-critical applications, our solution can be computed at lower resolution (AO is typically smooth). Fig. 6 shows an example of SD-HBAO rendered at half resolution in the Sibenik scene. Table 1 shows that, as expected, halving the resolution results in roughly 4x increase in speed across all configurations.

Multi-view ambient occlusion. Table 3 shows the timings for our multi-view solution in the Sponza scene. The extra time is due to the additional render pass for the secondary depth map, as well as the evaluation of these extra samples during the AO computation. Render time can be reduced again by using a half-resolution buffer (Table 3).

Missing occlusion compensation. The results highlighted in Fig. 4 and Fig. 5 show that our stochastic-depth AO variants can capture the missing occlusion in traditional screen-space AO algorithms. Nevertheless, compared to the reference via depth peeling, it necessarily underestimates the AO value, since the stochastic samples may miss occluders partially. We can reintroduce missing occlusions in an ad-hoc manner by artificially increasing the occlusion contribution of the stochastic samples, which will be distributed during the bilateral filter pass. We empirically found that doubling the AO contribution of stochastic samples can restore occlusion values that better match full depth-peeling-based AO at lower sample counts. Fig. 7 shows an example of the results obtained for the San Miguel scene with two stochastic samples per pixel. Only Fig. 7 and Fig. 8 contain such compensation.

Bent normals and cones. When using image-based lighting (IBL), or other forms of environment lighting, it is common to involve the fragment's normal. Bent normals [Landis 2002] and cones[Klehm et al. 2012] use an approximation of the average locally unoccluded directions instead of the fragment normal, which can be efficiently computed along with AO. Including stochastic samples in the bent normals and cones computations is straightforward. Our tests reveal only a slight cost increase (<0.2ms for 4 stochastic samples). As for AO, our stochastic samples for bent-normal and cone computation provides increased robustness, especially, under motion, when geometry becomes occluded or unoccluded. Yet, the effects tend to be more subtle. Fig. 9 shows an example of the difference when using stochastic samples to compute bent normals.

Limitations. When many layers exist between an occluder and the camera, AO values can be underestimated since fewer samples will be captured in the stochastic depth map. Fig. 8 shows an example of this problem, where 5 and 10 extra geometry layers are added to a pillar between the camera and a background occluder. Using a compensation factor can restore part of the missing occlusion, but may cause an overestimation in some areas. A possible solution is to lower the global transparency value α, while increasing the sample count. With a fixed sample count, this usually results in higher noise levels, which can be alleviated via an increased bilateral blur, or, as possible future work, temporal filtering.

Sampling-based screen-space AO techniques typically result in noisy AO values, which are smoothed using a bilateral filter. Our solution works in the same way, but the stochastic nature of samples can increase the variance of AO values for regions receiving AO only from stochastic samples, especially when using a single stochastic sample per pixel. This can result in noticeable low-frequency noise under motion (see supplementary video). This artifact can be alleviated by increasing the spatial bilateral filter radius, hereby increasing the amount of AO samples per pixel, or by performing temporal filtering. Future work could focus on identifying such regions and adaptively increasing the sample counts.

5 CONCLUSION

Our novel real-time solution using stochastic depth maps computes ambient occlusion in screen space. It avoids the problem of hidden geometry in a simple yet robust manner and extends to multiple views to eliminate artifacts that arise with view-aligned surfaces. Furthermore, bent-normal and cone computations also profit from our method, leading to more realistic shading.

6 ACKNOWLEDGMENTS

We would like to thank the reviewer who pointed out an inefficient step in our stochastic depth map creation algorithm. The test scenes were obtained from Morgan McGuire's Computer Graphics archive This work is partly supported by VIDI NextView, funded by NWO Vernieuwingsimpuls.

REFERENCES

Fabian Bauer, Martin Knuth, Arjan Kuijper, and Jan Bender. 2013. Screen-space ambient occlusion using a-buffer techniques. In *2013 International Conference on Computer-Aided Design and Computer Graphics*. IEEE, 140–147.

Louis Bavoil and Miguel Sainz. 2009. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH 2009: Talks*.

Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. 2008. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*.

Michael Bunnell. 2005. Dynamic ambient occlusion and indirect lighting. *Gpu gems* 2, 2 (2005), 223–233.

Loren Carpenter. 1984. The A-buffer, an antialiased hidden surface method. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. 103–108.

Franklin C. Crow. 1977. Shadow Algorithms for Computer Graphics. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques* (San Jose, California) *(SIGGRAPH '77)*. Association for Computing Machinery, New York, NY, USA, 242–248. https://doi.org/10.1145/563858.563901

Elmar Eisemann and Xavier Décoret. 2006. Fast Scene Voxelization and Applications. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (Redwood City, California) *(I3D '06)*. Association for Computing Machinery, New York, NY, USA, 71–78. https://doi.org/10.1145/1111411.1111424

Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. 2010. Stochastic transparency. *IEEE transactions on visualization and computer graphics* 17, 8 (2010), 1036–1047.

Pascal Gautron. 2020. Real-Time Ray-Traced Ambient Occlusion of Complex Scenes using Spatial Hashing. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks*. 1–2.

Quintjin Hendrickx, Leonardo Scandolo, Martin Eisemann, and Elmar Eisemann. 2015. Adaptively layered statistical volumetric obscurance. In *Proceedings of the 7th Conference on High-Performance Graphics*. 77–84.

Thai-Duong Hoang and Kok-Lim Low. 2012. Efficient screen-space approach to high-quality multiscale ambient occlusion. *The Visual Computer* 28, 3 (2012), 289–304.

Jorge Jiménez, Xianchun Wu, Angelo Pesce, and Adrian Jarabo. 2016. Practical real-time strategies for accurate indirect occlusion. *SIGGRAPH 2016 Courses: Physically Based Shading in Theory and Practice* (2016).

Oliver Klehm, Tobias Ritschel, Elmar Eisemann, and Hans-Peter Seidel. 2012. Screen-space bent cones: A practical approach. *GPU Pro* 3 (2012), 191–207.

Janne Kontkanen and Samuli Laine. 2005. Ambient occlusion fields. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*. 41–48.

Thomas Kroes, Dirk Schut, and Elmar Eisemann. 2015. Smooth probabilistic ambient occlusion for volume rendering. *GPU Pro 6: Advanced Rendering Techniques* (2015), 475.

Samuli Laine and Tero Karras. 2010. Two Methods for Fast Ray-Cast Ambient Occlusion. In *Proceedings of the 21st Eurographics Conference on Rendering* (Saarbrücken, Germany) *(EGSR'10)*. Eurographics Association, Goslar, DEU, 1325–1333. https://doi.org/10.1111/j.1467-8659.2010.01728.x

Hayden Landis. 2002. Production-ready global illumination. *Siggraph course notes* 16, 2002 (2002), 11.

Fang Liu, Yunpeng Song, Xuehui Liu, and Xianchao Xu. 2013. Multi-layer screen-space ambient occlusion using hybrid sampling. In *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. 71–76.

Bradford James Loos and Peter-Pike Sloan. 2010. Volumetric obscurance. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. 151–156.

M. Mara, M. McGuire, D. Nowrouzezahrai, and D. Luebke. 2016. Deep G-Buffers for Stable Global Illumination Approximation. In *Proceedings of High Performance Graphics* (Dublin, Ireland) *(HPG '16)*. Eurographics Association, Goslar, DEU, 87–98.

Morgan McGuire. 2010. Ambient occlusion volumes. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*.

Morgan McGuire. 2017. Computer Graphics Archive. https://casual-effects.com/data `https://casual-effects.com/data`.

Morgan McGuire, Michael Mara, and David Luebke. 2012. Scalable ambient obscurance. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*. Eurographics Association, 97–103.

Morgan McGuire, Brian Osman, Michael Bukowski, and Padraic Hennessy. 2011. The alchemy screen-space ambient obscurance algorithm. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. 25–32.

Martin Mittring. 2007. Finding next Gen: CryEngine 2. In *ACM SIGGRAPH 2007 Courses* (San Diego, California) *(SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 97–121. https://doi.org/10.1145/1281500.1281671

Oliver Nalbach, Tobias Ritschel, and Hans-Peter Seidel. 2014. Deep screen space. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 79–86.

Nvidia. 2013. ShadowWorks. https://developer.nvidia.com/shadowworks Accessed: 2020-12-22.

Matt Pharr and Simon Green. 2004. „Ambient Occlusion", GPU Gems.

Christoph Reinbothe, Tamy Boubekeur, and Marc Alexa. 2009. Hybrid Ambient Occlusion.. In *Eurographics (Areas Papers)*. 51–57.

Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. 2006. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *ACM SIGGRAPH 2006 Papers*. 977–986.

Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. 2009. Approximating Dynamic Global Illumination in Image Space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (Boston, Massachusetts) *(I3D '09)*. Association for Computing Machinery, New York, NY, USA, 75–82. https://doi.org/10.1145/1507149.1507161

Perumaal Shanmugam and Okan Arikan. 2007. Hardware Accelerated Ambient Occlusion Techniques on GPUs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (Seattle, Washington) *(I3D '07)*. Association for Computing Machinery, New York, NY, USA, 73–80. https://doi.org/10.1145/1230100.1230113

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 527–536.

Andrei Tatarinov and Alexey Panteleev. 2016. Advanced Ambient Occlusion Methods for Modern Games. Game Developers Conference. http://developer.download.nvidia.com/gameworks/events/GDC2016/atatarinov_alpanteleev_advanced_ao.pdf Accessed: 2020-07-15.

Kostas Vardis, Georgios Papaioannou, and Athanasios Gaitatzes. 2013. Multi-View Ambient Occlusion with Importance Sampling. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Orlando, Florida) *(I3D '13)*. Association for Computing Machinery, New York, NY, USA, 111–118. https://doi.org/10.1145/2448196.2448214

Jason C. Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. 2010. Real-Time Concurrent Linked List Construction on the GPU. *Computer Graphics Forum* 29, 4 (2010), 1297–1304. https://doi.org/10.1111/j.1467-8659.2010.01725.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2010.01725.x

S. Zhukov and G. Iones, A.and Kronin. 1998. An ambient light illumination model. In *Rendering Techniques '98*. Springer Vienna, Vienna, 45–55.