Pyramid HBAO — a Scalable Horizon-based Ambient Occlusion Method

Andrew Astapov[1], Vladimir Frolov[1,2] and Vladimir Galaktionov[2]

[1]Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia
[2]Keldysh Institute of Applied Mathematics, Miusskaya sq., 4, Moscow, 125047, Russia

Abstract

Screen-space Ambient Occlusion (SSAO) methods have become an integral part of the process of calculating global illumination effects in real-time applications. The use of ambient occlusion improves the perception of the geometry of the scene, and also makes a significant contribution to the realism of the rendered image. However, the problems of accuracy and efficiency of algorithms of calculating ambient occlusion remain relevant. Most of the existing methods have similar algorithmic complexity, what makes their use in real-time applications very limited. The performance issues of methods working in the screen space are particularly acute in the current growing spreadness of 4K (3840 x 2160 pixels) resolution of the rendered image. In this paper we provide our own algorithm Pyramid HBAO, which enhances the classic HBAO method by changing its calculation complexity for high resolution.

Keywords

Ambient Occlusion, realtime rendering, screen space, shading

1. Introduction

Ambient Occlusion (AO)) is a concept used in lighting theory that represents the level of reduction of indirect light at a point in space, depending on percentage of incoming indirect light rays blocked by the geometry surrounding the point.

The use of AO creates visual effects such as shading the global illumination of corners, wrinkles, small raised irregularities and cracks. This improves the perception of the scene geometry and makes the image more realistic (see Fig. 1).

The original problem of Ambient Occlusion calculation is a computationally complex process based on ray tracing [2]. In real-time applications, this problem is usually solved by a precalculation. Although there are some devices with ray tracing hardware acceleration support (such as the RTX series of Nvidia video cards, Playstation 5 and Xbox Series consoles), their capabilities remain quite limited, and the available RT cores are often used for more specific tasks.

To approximate Ambient Occlusion in real time, in 2007 Mittring (a leading graphics programmer at Crytek) introduced the **Screen Space Ambient Occlusion (SSAO)**[3] method, which calculates the approximation of ambient occlusion function using the information available

GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27–30, 2021, Nizhny Novgorod, Russia

andrey.astapov@graphics.cs.msu.ru (A. Astapov); vfrolov@graphics.cs.msu.ru (V. Frolov); vlgal@gin.keldysh.ru (V. Galaktionov)

0000-0002-6291-3150 (A. Astapov); 0000-0001-8829-9884 (V. Frolov); 0000-0001-6460-7539 (V. Galaktionov)

CEUR Workshop Proceedings (CEUR-WS.org)

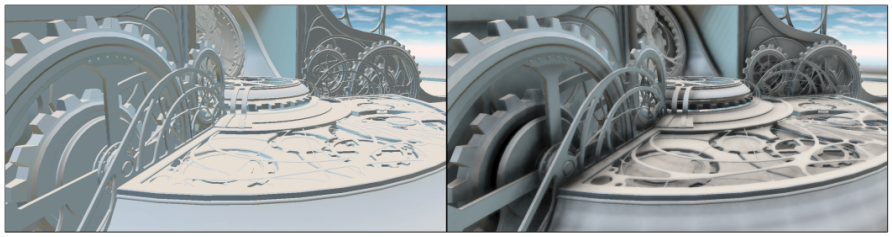


Figure 1: Left: simple environment lightning. Right: environment lightning with usage of ambient occlusion [1].

in the depth buffer. Despite the fact that this approach produces artifacts, because it does not have access to complete information about the geometry of the scene, it has become almost the standard for calculating AO in real-time applications. Another advantage of this algorithm is its ease of use. Since the SSAO-like algorithm only needs a depth buffer and a normal map, it can be built into the graphics pipeline as a post-processing stage, and can be used even in isolation from the graphics engine (a graphics engine is software which main task is to visualize three-dimensional computer graphics).

Although many improvements [4, 5, 6] have been introduced since the introduction of SSAO in 2007, most of them retain high algorithmic complexity. Such methods require a large number of samples from the depth texture to achieve visually acceptable results, which makes usage of even such approximation limited in real-time applications. Therefore, AO is often calculated on textures of 2-4 times lower resolution than the resolution of the final image, which creates additional artifacts and violates the integrity of the result along sharp high-frequency boundaries.

2. Related Work

2.1. Ray tracing and occlusion maps

The concept of **Ambient Occlusion** was first introduced in [2, 7] as part of a lighting model that complements the ambient component of the Phong shading model[8].

Common statement of Ambient Occlusion calculation task is following:

$$A = 1 - \frac{1}{2\pi} \int_{\Omega} V(\vec{\omega}) W(\vec{\omega}) d\omega, \text{ where} \tag{1}$$

- Ω — a hemisphere centered at a point O in 3d space, oriented along the normal at this point (see Fig. 2a).
- V — visibility function that returns 1 if the ray traced from the point O in the direction $\vec{\omega}$ intersects with the surrounding objects, 0 otherwise.
- W — an attenuation function that decreases with increasing distance to the intersection point of the ray traced from the point O in the direction $\vec{\omega}$ with the surrounding geometry.
- R — the maximum distance at which objects are be considered by the algorithm as occluders.

The proposed method[2] of baking occlusion maps for scene objects allows to simulate the effects of global lighting in real time, which increases the realism of the final image. The method was also useful in other areas of the media industry [9], because it could be used without expensive calculations of radiosity and offers a visually acceptable result.

However, this approach allows to correctly process only static scenes in real time. For dynamic objects such a precalculation cannot be performed.

2.2. Ambient Occlusion calculation in screen space

To avoid expensive calculations of the visibility function V in the three-dimensional space of objects, the **Screen Space Ambient Occlusion (SSAO)** method was developed. It was presented by Mittring [3] as part of the innovations of the graphics component of the Cryengine 2. The main idea of the method is to consider as occluders only the geometry visible on the screen. To do this, for each pixel P of the final occlusion map at the corresponding point in the 3D space O(P) in a spherical neighborhood (of radius R) K points in the 3D object space are selected. The level of the final occlusion at the point O(P) is defined as the percentage of the selected points in the neighborhood located below the surface formed by the values of the depth buffer (see Fig. 2b). In this form, the method was widely used, despite the fact that it often created artifacts in the form of dark halos around the silhouettes of objects, or white highlights on the borders of polygons.

Another significant drawback of the method presented by Mittring was selfoccludance (see Fig. 3a). For example, even a completely flat surface without other objects nearby is considered half-occluded, because half of the points selected in the spherical neighborhood of the point O(P) are under the surface formed by the depth buffer values.

This problem was solved in the SSAO+ [4] method presented a year later by using a different strategy for selecting points in the neighborhood of O(P). Instead of a spherical neighborhood of O(P), a hemisphere oriented along the normal at O(P) was used as the area for generating points (see Fig. 2c). This allowed to partially get rid of the highlights on the borders of flat surfaces and reduced the number of necessary samples from the depth texture, but dark halos around the objects still remained.

Another important innovation of the SSAO+ method was the use of filtering the resulting occlusion texture, which became an integral part of the algorithms presented in subsequent works. To suppress the noise obtained during the generation of the occlusion map, the authors used a bilateral filter that varies the weight of the Gaussian filter depending on the proximity of the depth and normal values of neighboring pixels.

However, the random nature of the selection of points in the neighborhood of O(P) led to a large number of misses in the texture cache of GPU. Therefore, the authors decided to generate a copy of the depth buffer, but with a resolution 2-4 times smaller than the original one. Such approach increased the efficiency of the shader but reduced the level of detail of the resulting occlusion map.

2.2.1. Horizon Based Ambient Occlusion

Although methods such as SSAO+ allowed for a more realistic final image, they had very little to do with estimation of the occlusion equation (1). Therefore, the **Horizon Based Ambient Occlusion (HBAO)** [5] method, which had a larger theoretical basis, soon became one of the most popular solutions for calculating ambient occlusion in screen space.

Let's write the occlusion equation 1 in terms of the polar coordinates θ (azimuth angle) and α (elevation angle):

$$A = 1 - \frac{1}{2\pi} \int_{\theta=-\pi}^{\pi} \int_{\alpha=0}^{\pi} V(\vec{\omega}) W(\vec{\omega}) cos(\alpha) d\alpha d\theta, \text{ where} \tag{2}$$

$$\vec{\omega}(\alpha, \theta) = (cos(\alpha) * cos(\theta), sin(\theta), sin(\alpha) * cos(\theta))$$

The main concept used in the HBAO algorithm is the horizon angle $h(\theta)$ - the maximum elevation angle at which in direction $\vec{\omega}(h(\theta), \theta)$ an occluder is found (see Fig. 2d). In this case, the equation 2 can be rewritten as:

$$A = 1 - \frac{1}{2\pi} \int_{\theta=-\pi}^{\pi} \int_{\alpha=0}^{h(\theta)} W(\vec{\omega}(\alpha, \theta)) cos(\alpha) d\alpha d\theta \tag{3}$$

Next, the authors simplify the equation (3) by substituting the attenuation function $W(\vec{\omega}(\alpha, \theta))$ by $W(\theta)$, which depends only on the azimuth angle θ:

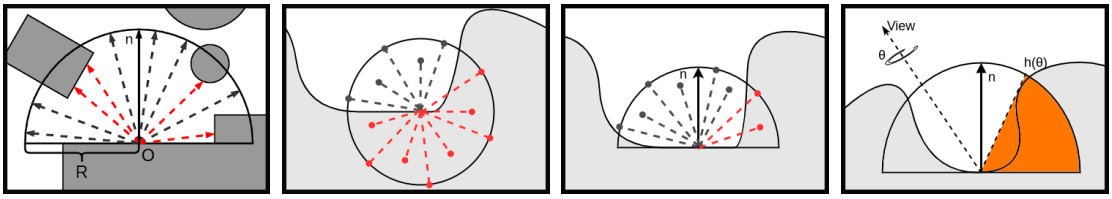
$$W(\theta) = max(0, 1 - r(\theta)/R), \text{ where} \tag{4}$$

- $r(\theta)$ — distance from point O(P) to the nearest occluder in the direction $\vec{\omega}(h(\theta), \theta)$.
- R — the maximum distance at which objects are be considered by the algorithm as occluders.

Then the equation (3) can be simplified:

$$A \approx 1 - \frac{1}{2\pi} \int_{\theta=-\pi}^{\pi} W(\theta) sin(h(\theta)) d\theta \tag{5}$$

The resulting expression (5) is calculated by the Monte Carlo method for a random selected azimuth angles θ. Thus, the occlusion function (1) is approximated by estimating the openness of the horizon around the point of interest O(P).



(a) Object space AO (b) Crytek SSAO (c) SSAO+ (d) HBAO

Figure 2: Strategies for calculating the occlusion equation.

2.2.2. Improvement ways

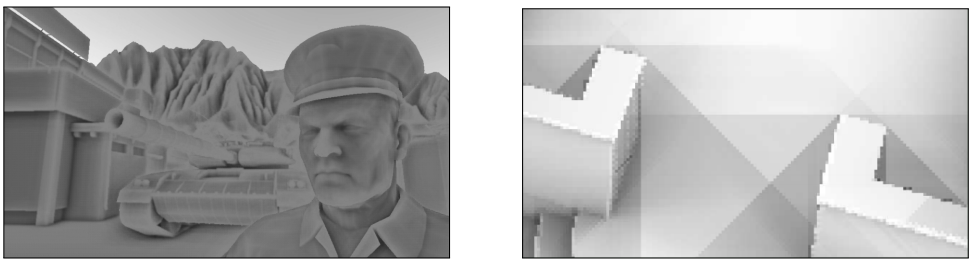
In subsequent works, various ways to improve certain aspects of the algorithm were proposed. For example, the Poisson Disk Ray-Marched Ambient Occlusion [10] method approximates the result obtained by ray tracing in the screen space, using a much smaller number of samples from the depth texture. However, due to ray marching basis of the algorithm its execution time did not allow it to be included in most of real-time applications.

In other works (such as [1]), the main goal was to reduce the number of artifacts that occur and the required number of samples from the depth texture per pixel, which was achieved by further simplifying the theoretical model. However, the algorithmic complexity of such algorithms remained the same.

One of the first attempts to change the algorithmic complexity was made in [11]. The proposed method improved the quality and efficiency of HBAO at large values of the radius R. This was achieved by building pyramids of normal texture and depth buffer, which were used to calculate occlusion maps of different sizes. At the next stage, the resulting occlusion texture was constructed by mixing the obtained occlusion maps of different resolutions. However, the algorithm had the additional memory overhead of storing pyramids of normal textures, a depth buffer, and occlusion maps. In addition, due to the large number of mixing steps and heavy bilateral texture filtering, the minimal execution time of the method did not allow it to be used in many real-time applications.

A bit later McGuire proposed method Scalable AO [6] which was an hierarchical enhancement of previous Alchemy AO [1] algorithm. In contrast to [11] this method requires generation of depth pyramid only. It calculates ambient occlusion in target resolution by aggregating information from different levels of depth buffer pyramid. Unlike [11] this method proved to be applicable in real-time applications. But like basic Alchemy AO [1] this algorithm has a simplified theoretical statement and looses occluders in many cases.

The Line-Sweep Ambient Obscurance [12] algorithm should be noted separately. This method has a lower algorithmic complexity due to the grouping samples from the depth texture by azimuth directions. This approach makes it possible to significantly reduce the required number of reading operations from the depth buffer. The drawback of this optimization is visible artifacts-bands that appear in case of small number of azimuth directions (see Fig. 3b).



(a) Crytek SSAO selfoccludance (b) LSAO linear banding artifacts

Figure 3: Artifacts of SSAO methods.

With the development of temporal methods, there appeared works that adapted various algorithms for the use of information from previous frames. For example, GTAO [13], based on

HBAO[5], achieves high efficiency through the use of temporal techniques. But this method can not be implemented in every graphics engine, because it requires the presence of calculated motion vectors for the projection of information from previous frames. In addition, like all temporal methods, this approach has low accuracy (or even generates artifacts) in the areas of dynamic objects. One of the most common artifacts of temporal algorithms is the effect of multiple "ghost" silhouettes of dynamic objects (ghosting).

2.2.3. Neural network methods

In the last few years, machine learning-based methods for approximating ambient occlusion have begun to appear. Just like other SSAO methods, neural network models take a depth buffer and a normal map as input and use them to generate an occlusion texture.

For example, Holden et al. [14] used a small perceptron consisting of several layers for ambient occlusion evaluation. Moreover, the size of such a network is so small that it can be used directly inside the fragment shader call. However, due to such a small number of trainable parameters, such network is not able to learn complex-structured features, and therefore is unable to generate a high-quality occlusion texture.

Therefore, later appeared the works [15, 16], which use a deeper U-Net-like architecture of the Encoder-Decoder type, which allows the network to extract more complex features by creating feature maps of different scales.

Such neural network algorithms achieve closer results to the original AO (calculated using ray tracing) than the classic SSAO methods, since they are trained to predict exactly the occlusion of the environment calculated in the space of objects in the scene. However, neural network methods have some common problems:

- Temporal stability of neural network methods.
- Portability of neural network methods to different scenes.
- Flexibility to adjust the algorithm result (via parameters such as the maximum occlusion radius R).

2.3. Calculation of Ambient Occlusion in object space in real time

Although the methods of calculating ambient occlusion in screen space are still the most practical way to calculate the occlusion map in real time, the result obtained with their help has many disadvantages.

An alternative approach to solving the problem of calculating ambient occlusion is Voxel-space ambient occlusion (VXAO) [17]. Instead of using the information available in screen space, this method builds a voxel representation of the scene geometry that covers a large area around the camera. Thus, objects that are out of the observer's field of view will contribute to the resulting value of the occlusion function (1).

For example, in Fig. 4b, you can see that the surface under the bottom of the tank is shaded. At the same time, in Figure 4a, this area remains light, because the depth buffer does not contain the necessary information, and therefore the SSAO method does not find the necessary occluder.

(a) SSAO (b) VXAO

Figure 4: Example results of a method that uses screen space representation the scene geometry, and a method that uses a voxel representation of the scene.

2.4. Conclusion

In total there are rather few SSAO methods that are temporally stable, easy to integrate in rendering pipeline and fast enough to be applicable for calculation high resolution AO map (or in case of high values of radius R) in real-time applications. In the next section we describe our algorithm Pyramid HBAO, which enhances the classic HBAO method by changing its calculation complexity for high resolution cases.

3. Proposed method

3.1. The performance problem of classic SSAO methods

Computational complexity of classic SSAO algorithms increases quadratically with the increase of the maximum radius R, at which distance the algorithm considers objects surrounding the point O(P) as occluders. This occurs due to the fact that in order to maintain stability of the result, such methods need to maintain the level of density of samples from the depth buffer in the screen space neighborhood of the pixel P set by the algorithm. Reducing this level leads to an increase of noise and the appearance of artifacts in resulting occlusion map.

For example, if we increase the radius R in HBAO by 2 times, we'll need to double the number of azimuth directions θ and also the number of samples from the depth buffer along each direction. Otherwise, the banding artifacts will appear.

A similar problem occurs when the resolution of the generated occlusion map is increased. Let R_{screen} be the radius of the screen space neighborhood of pixel P, obtained by projecting a three-dimensional spherical neighborhood of point O(P) of radius R onto the screen. With increasing the resolution of the output occlusion map by 2 times for each axis, the size of the radius R_{screen} (measured in pixels of the image) also doubles. And therefore, as in the case of increasing the radius of R, the number of necessary reads from the depth buffer grows quadratically.

In addition, as the R_{screen} radius increases, in case of classic SSAO algorithms (including HBAO) the number of misses in the texture cache also raises, which negatively affects performance.

For these reasons, AO is often calculated on textures of 2-4 times lower resolution than the resolution of the final rendered image.

3.2. Pyramid HBAO scheme

In order to solve the performance problem of SSAO methods in case of large values of R, we propose Pyramid HBAO method which enhances basic HBAO algorithm [5] by a two-stage process:

1. By given depth map generate a depth pyramid, where each subsequent level is a less detailed representation of the depth buffer than the previous one (like in [6]).
2. Approximate the occlusion function (1) at each pixel P, using information from different levels of the depth pyramid as screen space approximations of different scales of the scene geometry.

3.3. Depth pyramid generation

The resolution of each subsequent depth pyramid level is K times less (rounded up) on each axis of the resolution of the previous one. At the same time, the first level of the pyramid has a resolution K times less than the resolution of the original depth buffer.

Calculation of the first level of the depth pyramid consists of next steps for each pixel P:

1. The values of the depth buffer are read from the the KxK kernel with center of pixel P.
2. Each of the obtained depth values is translated to linear space:

$$z = \frac{z_n * z_f}{z_f - depth * (z_f - z_n)}, \text{ where} \tag{6}$$

   - $depth$ — value read from the depth buffer.
   - z — depth value in linear space.
   - z_n, z_f — near and far clipping planes respectively.

3. The mean value of the obtained linear depth values is written to the pyramid texture.

Each subsequent depth pyramid level is obtained from the previous one by averaging its values in a KxK pixel window. The resolution reduction scale K was chosen to be 3, because in this case, the resulting pyramid takes up less GPU memory, and its construction is faster than in the case of a resolution reduction scale of 2. At the same time, the difference in the quality of the resulting occlusion map is insignificant.

3.4. Approximation of occlusion equation

Proposed method Pyramid HBAO like basic algorithm Horizon Based Ambient Occlusion (HBAO) [5] consists of the same steps:

1. Select D azimuth directions θ.
2. Approximate the occlusion function (1) along each azimuth direction.

3. Average the received directional values.

Pyramid HBAO uses the same linear attenuation function:

$$W(r) = max(0, 1 - r/R), \tag{7}$$

where r is a distance from point O(P) to an occluder.

However, instead of finding the maximum elevation angle $h(\theta)$ the proposed method considers the occluder that gives the maximum weighted contribution as an approximation of the occlusion along the direction θ:

$$A(\theta) = max_{\alpha}(W(r(\alpha, \theta))sin(\alpha)), \text{ where} \tag{8}$$

- α — current elevation angle found at azimuth direction θ.
- $r(\alpha, \theta)$ — distance from point O(P) to the nearest occluder in the direction $\vec{\omega}(\alpha, \theta)$.

This strategy produces less artifacts in the resulting occlusion map.

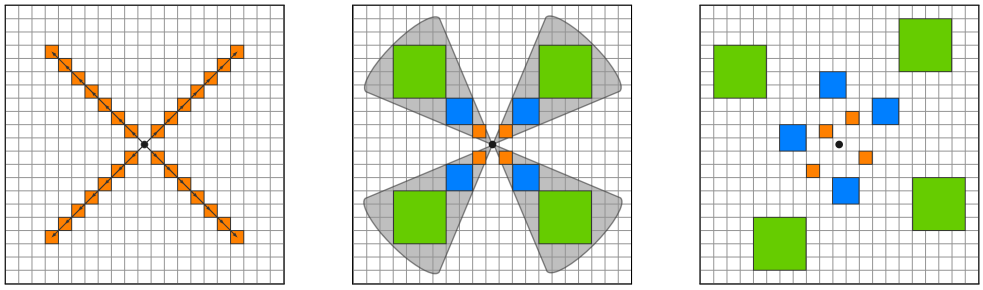
Let's assume that in the previous step, a depth pyramid was built with L levels. Then the search for occlusion values in each direction is performed as follows for each pixel P (see Fig. 5b):

1. Calculate three-dimensional space position O(P) by given value from depth buffer. Read the normal at this point from the normal map.
2. For each depth pyramid level (counting the depth buffer itself as the 0th level of the pyramid) do:
   a) Along each azimuth direction make $step\_count$ consecutive reads with some step size from current depth pyramid level. For each of the obtained depth values restore the position in the three-dimensional space and update maximum of the function (8).
   b) Multiply the value of the step size by the depth pyramid scale factor K. Move to the next depth pyramid level.
3. Average the received directional values.

Thus, near the point of interest P, the reading is made from the levels of the depth pyramid with high resolution, and at a higher distance from the point P — with low resolution. I.e., instead of tracing a ray along the azimuth direction θ (as is done in HBAO), the proposed method makes a similar tracing of the cone along the depth map.

Usage of depth pyramid like in Scalable AO [6] gives next advantages of the proposed method:

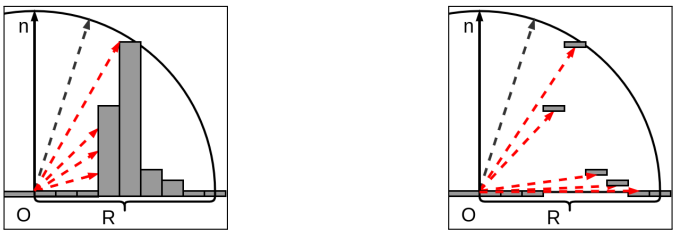
- Due to the exponential growth of the step size, it is possible to achieve larger sizes of the radius R without quadratic growth of reads from depth buffer.
- By switching to smaller textures, reading is produced from a small neighborhood of the point of interest P, from which fewer texture cache misses can be achieved.
- After finding the optimal number of azimuth directions $D_{optimal}$ that gives an acceptable quality, this number no longer needs to be increased if the required radius R increases, since such hierarchical approach maintains the same density of samples from the levels of the depth pyramid.



(a) HBAO (b) Pyramid HBAO (c) Scalable AO

Figure 5: Depth sampling strategies of different SSAO methods. Different pixel colors mean different depth pyramid levels: orange – 0 (or original depth buffer), blue – 1, green – 2.

The main difference in results produced by Scalable AO [6] and proposed method Pyramid HBAO comes from different strategies of approximation of scene geometry by screen space depth buffer (see Fig. 6). Scalable AO approach bases on Alchemy AO [1] which considers scene as a discontinuous number of points. So nearby points doesn't produce occluders that block rays in directions of faraway points. Thus Scalable AO has a tendency to under-occludance. In contrast Pyramid HBAO based on HBAO [5] assumes that scene geometry is a continuous height field produced by depth buffer. Such approach in many cases produces a closer approximation to real geometry of a scene. But it has the opposite problems of over-occludance and appearing dark halos around object silhouettes.



(a) Pyramid HBAO (b) Scalable AO

Figure 6: Different strategies of approximating scene geometry by screen space depth buffer.

3.5. Artifact suppression

To prevent the occurrence of regular patterns in the resulting AO texture, for each pixel P uniformly selected azimuth directions are rotated by a random angle generated based on a value from blue noise texture (with resolution 32x32 pixels).

However, the resulting occlusion map still showed stepwise patterns and dark halos around the objects. To suppress these artifacts, the uniform distribution of azimuth directions was changed to a random one. Also starting step size was multiplied by a random value selected for each azimuth direction.

As can be seen from Fig. 7b, 7e, such solution led to an increase of noise in the resulting AO

map. To suppress it, we used a separable Gaussian filter with the value $sigma$ = 1.5 and usage of 5 samples from the texture for each direction.

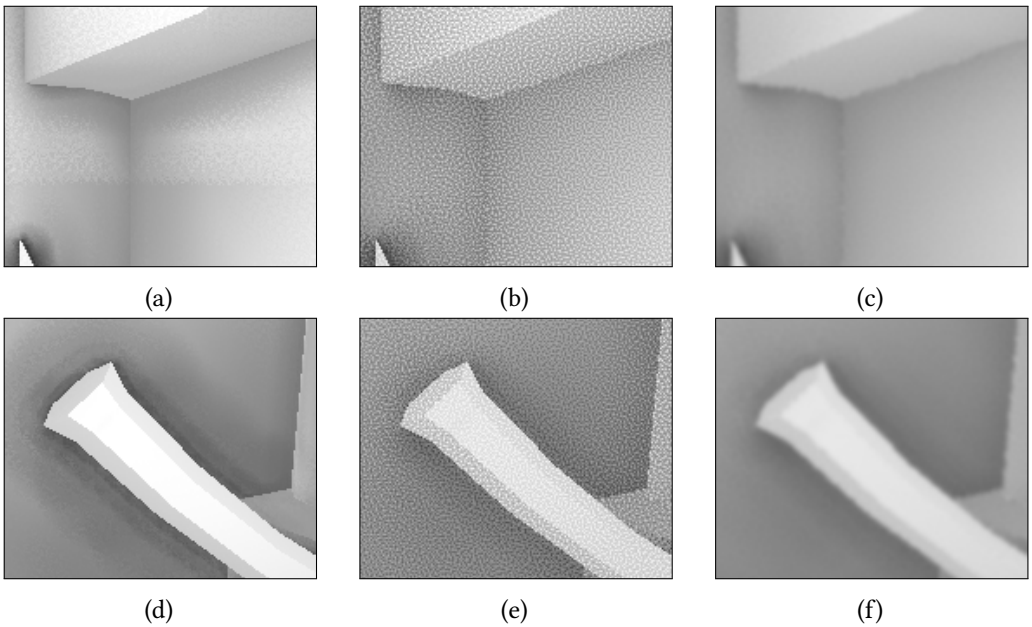


Figure 7: Example of suppression of regular stepwise patterns (7a, 7b, 7c) and dark halos around objects (7d, 7e, 7f).

4. Experiments

The experiments were conducted on Intel Core i7-9750H and NVIDIA GeForce RTX 2080 Mobile.

The resolution of the generated occlusion maps is — 1280x720. We used 2 scenes for testing: Viking Room (see Fig.9) and Crytek Sponza (see Fig.10).

Time measurements were conducted on a large number of frames (more than 1000 frames) for each set of parameters for each method. Then the obtained time values were averaged. Since execution time of screen-space methods does not depend on the camera angle and scene complexity, the same time is specified in the table (1a, 1b, 1c) for both scenes.

The reference result was obtained in Blender by ray tracing in three-dimensional object space. For the smoothness of the resulting AO map, 1024 rays per pixel were used.

We used two target metrics for quality measurement:

1. Mean absolute error (MAE):

$$MAE = \frac{1}{M * N} \sum_{i=0}^{M} \sum_{j=0}^{N} |x_{ij} - y_{ij}|, \text{ where} \tag{9}$$

- M, N — image resolution;

- x_{ij}, y_{ij} — pixel value at coord (i,j) of images X and Y respectively.

2. Structural similarity index (SSIM). Passes a sliding window of size KxK over two images, compares the windows and returns the average of the obtained values as a result. The result takes a value from -1 to 1. The difference between two windows x and y is calculated as follows:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \text{ where} \tag{10}$$

- μ_x, μ_y — mean for windows x and y respectively;
- σ_x^2, σ_y^2 — variance for windows x and y respectively;
- σ_{xy} — covariance of values from windows x and y;
- $c1 = (k_1 Q)^2$, $c2 = (k_2 Q)^2$ — constants defined by values:
  - Q — dynamic range of image values;
  - $k_1 = 0.001$, $k_2 = 0.03$ — constants.

For SSIM metric, the window size was selected to be 11x11 pixels.
Let's define the following notation:

- **D** — the number of azimuth directions;
- **S** — the number of depth samples per azimuth direction;
- **L** — the number of levels in depth buffer pyramid.

The results of measuring target metrics and execution time for different methods are presented in the tables (1a, 1b, 1c). In all tables, total execution time is given without taking into account execution time of the separable Gaussian filter, which takes 0.25 ms. To make time results comparable we used the same number of samples per each depth pyramid level in both Scalable AO and Pyramid HBAO methods.

As can be seen from the tables (1a, 1b, 1c) and plots (Fig. 8), the computational complexity of HBAO method increases quadratically with increasing radius of the considered screen space neighborhood of the pixel P. At the same time, the increase of execution time of Pyramid HBAO (and Scalable AO) is close to linear. This is achieved for two reasons:

- The selected number of azimuth directions D=8 and the number of samples from each level of the depth pyramid remain unchanged. When the radius is increased, only the number of pyramid levels changes.
- The field of view of the algorithm grows exponentially with an increasing of the number of pyramid levels. Therefore, with a multiple increase in the radius of R, the required number of samples from the pyramid along each direction increases linearly.

It is important to note that Scalable AO having the same complexity as Pyramid HBAO shows results which are less close to a reference AO map. This can be explained by different approaches of approximating scene geometry by screen space depth buffer (see Fig.6).

Table 1
Target metrics and execution time results

(a) HBAO

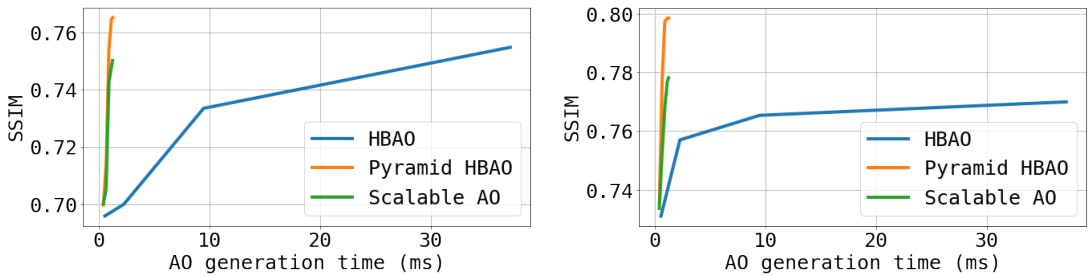
D	S	Viking Room MAE	Viking Room SSIM	Crytek Sponza MAE	Crytek Sponza SSIM	T(ms)
8	10	0.1208	0.7310	0.2398	0.6959	0.58
16	20	0.0940	0.7570	0.2186	0.7000	2.29
32	40	0.0816	0.7654	0.1938	0.7335	9.5
64	80	0.0800	0.7700	0.1737	0.7548	37.2

(b) Pyramid HBAO

L	Viking Room MAE	Viking Room SSIM	Crytek Sponza MAE	Crytek Sponza SSIM	T(ms)
1	0.1208	0.7357	0.2435	0.6997	0.44
2	0.0822	0.7776	0.2123	0.7140	0.69
3	0.0788	0.7975	0.1759	0.7537	0.93
4	0.0773	0.7986	0.1458	0.7646	1.16
5	0.0773	0.7986	0.1414	0.7652	1.28

(c) Scalable AO

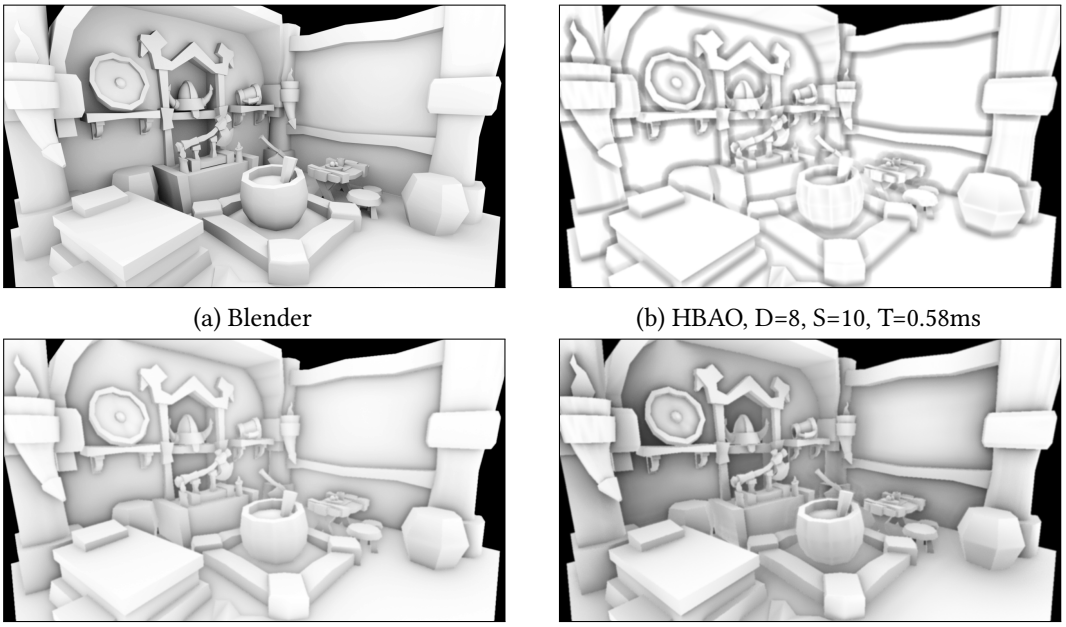
L	Viking Room MAE	Viking Room SSIM	Crytek Sponza MAE	Crytek Sponza SSIM	T(ms)
1	0.1215	0.7336	0.2445	0.7003	0.43
2	0.0920	0.7523	0.2223	0.7051	0.68
3	0.0897	0.7686	0.1899	0.7428	0.94
4	0.0874	0.7769	0.1688	0.7476	1.15
5	0.0876	0.7782	0.1660	0.7502	1.26



(a) Sponza SSIM

(b) Viking Room SSIM

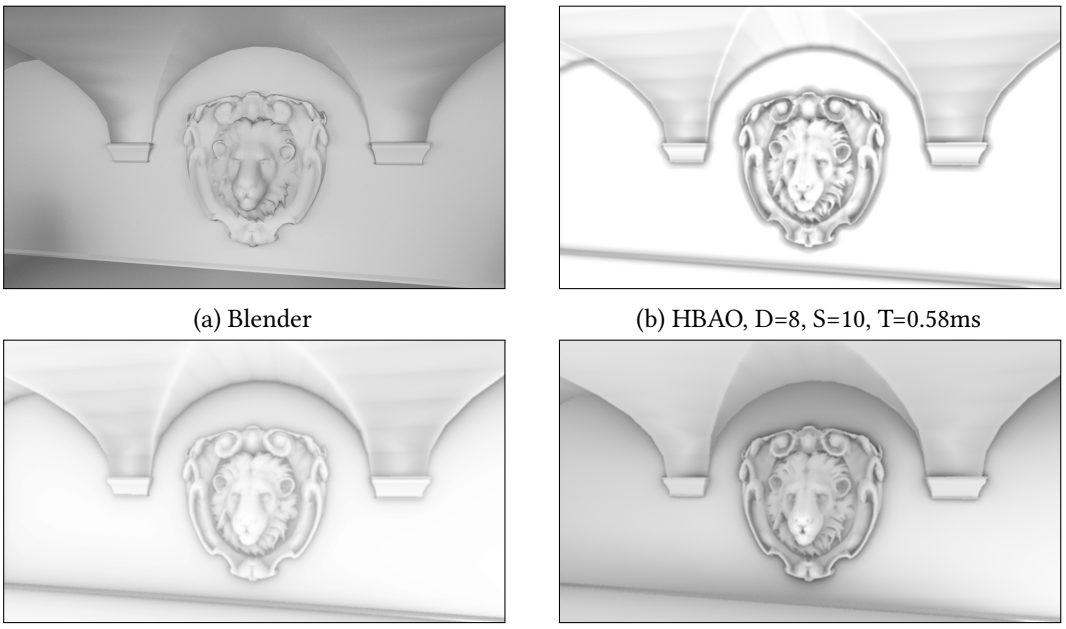
Figure 8: Dependency of SSIM metric on the AO map generation time.



(a) Blender (b) HBAO, D=8, S=10, T=0.58ms

(c) Scalable AO, L=5, T=1.26ms (d) Pyramid HBAO, L=5, T=1.28ms

Figure 9: Examples of generated AO maps on Viking Room scene.



(a) Blender (b) HBAO, D=8, S=10, T=0.58ms

(c) Scalable AO, L=5, T=1.26ms (d) Pyramid HBAO, L=5, T=1.28ms

Figure 10: Examples of generated AO maps on Crytek Sponza scene.

5. Conclusion

We have presented Pyramid HBAO algorithm, which enhances the classic HBAO method by changing its calculation complexity for high resolution and high values of radius R. As has been shown in experimental section Pyramid HBAO shows comparable to classic HBAO qualitative results and its execution time stays low enough to be used in real-time applications.

References

[1] M. McGuire, B. Osman, M. Bukowski, P. Hennessy, The alchemy screen-space ambient obscurance algorithm, in: Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics, 2011, pp. 25–32. doi:10.1145/2018323.2018327.

[2] S. Zhukov, A. Iones, G. Kronin, An ambient light illumination model, in: Eurographics Workshop on Rendering Techniques, Springer, 1998, pp. 45–55. doi:10.1007/978-3-7091-6453-2_5.

[3] M. Mittring, Finding next gen: Cryengine 2, in: ACM SIGGRAPH 2007 courses, 2007, pp. 97–121. doi:10.1145/1281500.1281671.

[4] D. Filion, R. McNaughton, Effects & techniques, in: ACM SIGGRAPH 2008 Games, 2008, pp. 133–164. doi:10.1145/1404435.1404441.

[5] L. Bavoil, M. Sainz, R. Dimitrov, Image-space horizon-based ambient occlusion, in: ACM SIGGRAPH 2008 talks, 2008, pp. 1–1. doi:10.1145/1401032.1401061.

[6] M. McGuire, M. Mara, D. P. Luebke, Scalable ambient obscurance., in: High Performance Graphics, Citeseer, 2012, pp. 97–103. doi:10.2312/EGGH/HPG12/097-103.

[7] A. Iones, A. Krupkin, M. Sbert, S. Zhukov, Fast, realistic lighting for video games, IEEE computer graphics and applications 23 (2003) 54–64. doi:10.1109/MCG.2003.1198263.

[8] B. T. Phong, Illumination for computer generated pictures, Communications of the ACM 18 (1975) 311–317. doi:10.1145/360825.360839.

[9] H. Landis, Production-ready global illumination, in: Siggraph 2002, volume 5, 2002, pp. 93–95.

[10] G. Sourimant, P. Gautron, J.-E. Marvie, Poisson disk ray-marched ambient occlusion, in: Symposium on Interactive 3D Graphics and Games, 2011, pp. 210–210. doi:10.1145/1944745.1944790.

[11] T.-D. Hoang, K.-L. Low, Efficient screen-space approach to high-quality multiscale ambient occlusion, The Visual Computer 28 (2012) 289–304. doi:10.1007/s00371-011-0639-y.

[12] V. Timonen, Line-sweep ambient obscurance, in: Computer graphics forum, volume 32, Wiley Online Library, 2013, pp. 97–105. doi:10.1111/cgf.12155.

[13] J. Jiménez, X. Wu, A. Pesce, A. Jarabo, Practical real-time strategies for accurate indirect occlusion, SIGGRAPH 2016 Courses: Physically Based Shading in Theory and Practice (2016).

[14] D. Holden, J. Saito, T. Komura, Neural network ambient occlusion, in: SIGGRAPH ASIA 2016 Technical Briefs, 2016, pp. 1–4. doi:10.1145/3005358.3005387.

[15] O. Nalbach, E. Arabadzhiyska, D. Mehta, H.-P. Seidel, T. Ritschel, Deep shading: convolu-

tional neural networks for screen space shading, in: Computer graphics forum, volume 36, Wiley Online Library, 2017, pp. 65–78. doi:10.1111/cgf.13225.

[16] D. Zhang, C. Xian, G. Luo, Y. Xiong, C. Han, Deepao: Efficient screen space ambient occlusion generation via deep network, IEEE Access 8 (2020) 64434–64441. doi:10.1109/ACCESS.2020.2984771.

[17] R. Penmatsa, G. Nichols, C. Wyman, Voxel-space ambient occlusion, in: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, 2010, pp. 1–1. doi:10.1145/1730804.1730989.