Screen-Space Far-Field Ambient Obscurance

Ville Timonen*
Turku Centre for Computer Science
Åbo Akademi University

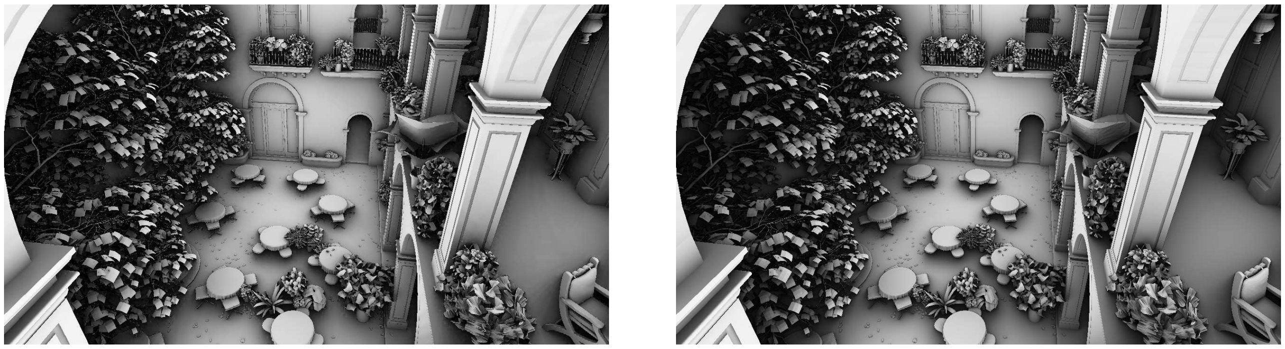


Figure 1: *Left: screen-space far-field ($\geq$ 15 px) occlusion component solved by our method in 4.6 ms on a 1280(+256)$\times$720(+144) depth buffer. Right: ray traced screen-space reference result.*

Abstract

Ambient obscurance (AO) is an effective approximation of global illumination, and its screen-space (SSAO) versions that operate on depth buffers only are widely used in real-time applications. We present an SSAO method that allows the obscurance effect to be determined from the entire depth buffer for each pixel. Our contribution is two-fold: Firstly, we build an obscurance estimator that accurately converges to ray traced reference results on the same screen-space geometry. Secondly, we generate an intermediate representation of the depth field which, when sampled, gives local peaks of the geometry from the point of view of the receiver. Only a small number of such samples are required to capture AO effects without undersampling artefacts that plague previous methods. Our method is unaffected by the radius of the AO effect or by the complexity of the falloff function and produces results within a few percent of a ray traced screen-space reference at constant real-time frame rates.

1 Introduction

Ambient occlusion approximates global illumination under the assumption that the scene is uniformly lit by blocking part of the incident light due to surrounding occluders. Ambient *obscurance* extends ambient occlusion by introducing a falloff term which attenuates the occlusion effect as a function of occluder distance. The appropriate choice for a falloff term depends on various factors and therefore an ambient obscurance algorithm should be able to support any such distance dependent falloff function.

Screen-space ambient occlusion and obscurance (SSAO) methods have recently become very popular in real-time applications because they only require the depth buffer as input and are therefore easily applied as a post-process or plugged into a deferred renderer, work on fully dynamic scenes, and are insensitive to scene complexity. The falloff term is defined in eye-space distances, which means that the obscurance radius in screen-space depends on the camera's distance to the geometry and may get arbitrarily large. If this was not the case, objects would change appearance as they get closer to the camera. Indeed, an AO effect that conveys the proper global illumination feel often extends a significant radius in screen-space. The best any SSAO method can do given the information available in the depth buffer is represented by a ray tracer: From each receiver pixel rays are traced over the hemisphere to their nearest intersection with the depth field, and then falloff and cosine weighted.

A majority of real-time SSAO methods rely on taking point samples of the depth buffer in the immediate pixel neighborhood of the receiver. This approach is efficient for gathering ambient occlusion from a small neighborhood around the receiver, making screen-space near-field occlusion largely a solved problem. However, as the sampled environment grows in radius, geometry will be missed and noise is produced. Keeping up with the increased radius requires quadratic work and has not been found feasible, even after accepting a blurrable amount of noise. It is possible, however, to use mipmapped depth data such that lower resolution levels are used when sampling farther from the receiver. This approach is used by most state-of-the-art methods and does not miss geometry, but simply averaging the depth field corrupts occluders as seen from the receiver and causes erroneous results.

In this paper our primary contribution is an intermediate representation of the depth field that can be sampled at various distances from the receiver to get virtual scene points that reconstruct features important for AO. The intermediate representation is generated in a pre-pass which scans through the depth field, runs in time that is linear in the depth field size, and generally takes a small fraction of the total time. Our secondary contribution is an obscurance estimator that is fast to evaluate and converges accurately to the AO integral [Zhukov et al. 1998]. When evaluated with scene samples from our intermediate representation, the estimator reaches results within a few percent of the ray traced screen-space reference at real-time frame rates.

Our algorithm executes in three passes: First the depth field is preprocessed by scanning along multiple azimuthal directions, next the output is traversed orthogonally to the scanning directions to pre-integrate dominant occluders, and finally obscurance is evaluated

*e-mail: vtimonen@abo.fi

per-pixel from the reconstructed occluders. The key features of our SSAO solution are:

- Constant time, unbounded radius (the effect may span the entire screen)
- Does not miss important occluders (no noise or need to filter)
- Supports arbitrary falloff functions (no render time evaluation)

2 Previous work

In this section we cover only previous work most relevant for our method; for a recent review of ambient occlusion and SSAO methods, consult [Ritschel et al. 2012].

The main branch of present SSAO methods follows from the works of [Mittring 2007] and [Shanmugam and Arikan 2007] where point samples around the receiver are taken to approximate the visibility of the hemisphere. In order to avoid overocclusion when samples farther from the receiver are evaluated, it is important to know whether there is intersecting geometry closer to the receiver which would render the sampled point invisible. To this end, it is possible to connect the samples along one azimuthal direction to get one horizon value instead, as done in [Bavoil et al. 2008]. AO is calculated based on the global horizon angle and rays below the horizon are assumed to be occluded. However in ambient *obscurance*, when a non-constant falloff term is used, occluders' distances below the horizon affect the amount of occlusion and need to be known. Our method tracks the horizon incrementally as geometry is traversed outwards from the receiver, and occlusion coming from geometry visible to the receiver below the global horizon is properly weighted by distance.

Global horizons for a height field are calculated efficiently in [Timonen and Westerholm 2010] for direct lighting of a height field, however for ambient obscurance the same single-horizon problem applies: No information is kept of the geometry below the global horizon, and weighting the occlusion properly according to a falloff function is not possible. Errors can get arbitrarily large because it is not known how far the geometry below the global horizon is from the receiver. [Timonen 2013] fits this method to SSAO by finding the largest falloff attenuated occluder for each direction instead of the global horizon. While this is much more useful for SSAO, geometry above the largest occluder is ignored and the distance to the geometry below the largest occluder is still unknown. While this is suitable for approximate SSAO, results do not converge to a ray traced reference or scale to very high quality like those of our method. Also, both [Timonen and Westerholm 2010] and [Timonen 2013] sample the height field along straight lines whereas we account for the visibility of the geometry over the full azimuth. Considering geometry only along a set of straight lines significantly accentuates banding (cf. Figure 10).

Lower resolution (mipmapped) depth buffers can be used for sampling farther from the receiver as done by [Bavoil and Sainz 2009] [Hoang and Low 2012] [McGuire et al. 2012]. An artefact-free sampling of this multi-resolution representation is not a trivial task, as shown in [Snyder and Nowrouzezahrai 2008]. Regardless of the used low-pass filter, reducing the depth field over an area into a single-valued texel does not capture the view-dependency when the depth field is viewed from an arbitrary receiver. While we also use a resolution hierarchy, we capture information of the enclosed geometry such that it retains the approximated local peaks as viewed from any receiver. Furthermore, levels in our hierarchy are well-aligned (do not overlap or have gaps), making artefact-free sampling straight-forward.

Methods that sample an area around the receiver that is fixed in screen-space may produce fast results [Loos and Sloan 2010], but these methods neither scale to far-field AO nor respect a falloff term. It is also possible to use a forward rendering approach to AO whereby scene geometry prior to rendering is expanded and occlusion is spread onto the area of influence. This approach is pursued in [McGuire 2010], but the method does not scale to far-field effects because of increased overocclusion and high fillrate requirements.

A near-field screen-space search can be coupled with a far-field *world-space* method. A voxelization of the scene is ray traced in [Reinbothe et al. 2009], and [Shanmugam and Arikan 2007] use spherical proxies to approximate scene polygons. World-space methods have significantly different characteristics to screen-space methods: While they have the possibility to include geometry not visible in the depth buffer, they are forced to evaluate the visibility of many geometric primitives per pixel. This is costly and prone to produce overocclusion. Results and performance depend on the scene geometry whereas pure screen-space methods are insensitive to scene complexity.

Finally, purely ray traced AO methods such as [Laine and Karras 2010] produce results similar in quality to ours but do not suffer from the limitations of screen-space information. These methods are, however, at least an order of magnitude slower.

3 Algorithm overview

Our algorithm takes as input the depth and normal buffers, the projection matrix, and a pointer to the falloff function. The depth and normal buffers can change freely between frames, and the depth buffer may include optional guard bands. Geometry within the guard bands are considered as occluders, but obscurance values are not calculated for pixels in the guard band. The output of our algorithm is a floating point map of the ambient light.

We evaluate the 2D ambient obscurance integral in K azimuthal slices. In Section 4 we describe our obscurance estimator that is evaluated for each screen pixel. It takes scene points along each azimuthal direction as input. In order to generate these points, our method first scans the depth buffer in parallel lines in K azimuthal directions and writes out an intermediate representation at regularly spaced intervals along the lines as described in detail in Section 6. This is illustrated for one azimuthal direction to the left in Figure 2. This intermediate data is then optionally (when the depth field can be assumed continuous) traversed perpendicular to the azimuthal scan direction and turned into a prefix sum, as shown to the center in Figure 2. The purpose of the prefix sum is to allow averaging of the intermediate data over each of the K azimuthal sectors which effectively avoids azimuthal undersampling and reduces banding. This phase is described in Section 7. Finally, the prefix sum is sampled per pixel, shown to the right in Figure 2, to construct points (as input to our obscurance estimator) that track local peaks of the depth field. As a reference method we use a mipmapped depth buffer, described in Section 5, which is used in present state-of-the-art. Results are presented in Section 8.

The remaining perceptually dominant artefact in our method is banding. We propose three mutually complementary strategies to reduce banding:

1. Averaging scene points over sectors in continuous depth fields (Section 7)
2. Sparse evaluation of sectors which trades banding for blur and reduces render times (Section 10)
3. Jittering sampling directions per-pixel which trades banding for noise (Section 11.1)

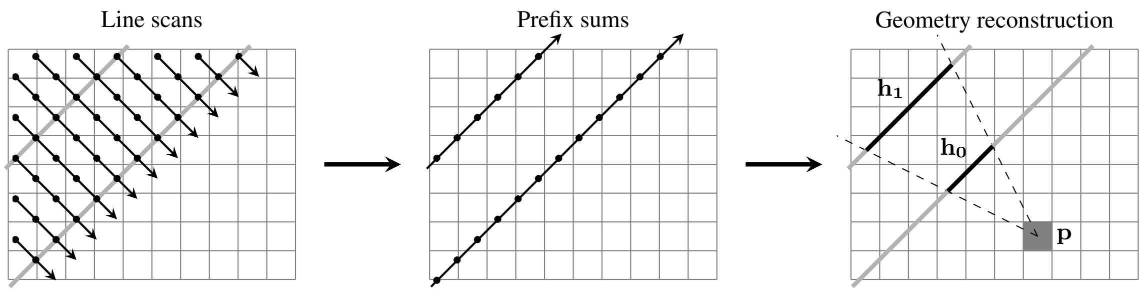


Figure 2: *The three main stages of our method for one azimuthal direction. The first stage scans the depth buffer in parallel lines (arrows to the left) and outputs intermediate geometry at regularly spaced intervals (gray lines). The second stage traverses the output (arrows in the middle) across multiple scan lines and generates prefix sums. The third stage samples the prefix sums per pixel (one pixel highlighted in gray to the right) to construct scene points* $\mathbf{h_i}$ *used as input by our obscurance estimator.*

Our method is most useful for finding far-field occluders and can be coupled with a lighter weight near-field search as discussed in Section 9. The usual limitations of depth buffer geometry apply and are discussed with regards to our method in Section 11.

4 Obscurance estimator

We build our obscurance estimator such that it converges to the original definition of ambient obscurance [Zhukov et al. 1998]:

$$AO(\mathbf{p},\vec{n}) = \frac{1}{\pi}\int_{\Omega} \rho(d(\mathbf{p},\vec{\omega}))\max(0,\vec{n}\cdot\vec{\omega})d\vec{\omega} \qquad (1)$$

where $d(\mathbf{p},\vec{\omega})$ is the distance of the nearest occluder from $\mathbf{p}$ in direction $\vec{\omega}$, $\rho \rightarrow [0,1]$ is the falloff term as a function of distance, and Ω denotes the unit sphere in $\mathbb{R}^3$. The falloff function should be smooth and it typically applies that $\rho(0) = 1$ and $\rho(\infty) = 0$, but the exact type and rate of decay is defined by the application. We support any such falloff function and its complexity only affects pre-calculation, not runtime performance.

For geometrically correct SSAO the surrounding geometry for a receiver has to be traversed in azimuthal directions starting from near the receiver and progressing outwards, an approach first taken by [Bavoil et al. 2008]. This way the nearest occluder along a direction from the receiver is guaranteed to be found and contribution from invisible geometry (behind a nearer occluder) can be correctly ignored. At each receiver we consider the vector from the receiver to the camera to represent zenith, and the sphere around the receiver is split into K azimuthal sectors. Therefore the 2D integral of Eqn. 1 is decomposed into K 1D integrals. Each of the 1D integrals is evaluated on a plane that includes the zenith vector and the vector pointing towards the azimuthal angle $k2\pi/K$:

$$AO(\mathbf{p},\vec{n}) \approx$$
$$\frac{1}{\pi}\sum_{k=0}^{K-1}\left(w_k\int_0^{\pi}\rho(d(\mathbf{p},\vec{\theta}_k))\max(0,\vec{n}_k\cdot\vec{\theta}_k)sin\theta d\theta\right) \qquad (2)$$

where $\vec{\theta}_k$ is a unit vector in the k:th azimuthal plane towards the horizon angle θ, and $\vec{n}_k$ is the projection of $\vec{n}$ onto this plane. Geometry will be sampled from the depth buffer along equal size azimuthal sectors in screen-space which map to relative sizes w_k as measured around the zenith for each $\mathbf{p}$. The term $sin\theta$ accounts for the width of the sector as a function of the horizon angle and goes to zero near the zenith.

The integral in Eqn. 2 is evaluated piecewise from depth field points in each sector. The points should be culled to form a series of increasing slopes as measured from the receiver $\mathbf{p}$, i.e. the points should be visible to $\mathbf{p}$. In practice, when the depth field is sampled progressively farther from the receiver, the largest horizon angle from the receiver to the sampled scene point is tracked and new points are known to be visible when their horizon angle exceed the previous maximum horizon angle. As we will only evaluate obscurance from a set of points along a sector that are apart from each other, we do not know the distance of the geometry between the points. For conservative obscurance, we assume that each sampled scene point represents a "slab" of geometry that extends outwards from the camera along the negative zenith direction, as illustrated in Figure 3. The slabs extend the thickness of the depth field; to infinity if the depth field is assumed continuous. While it would

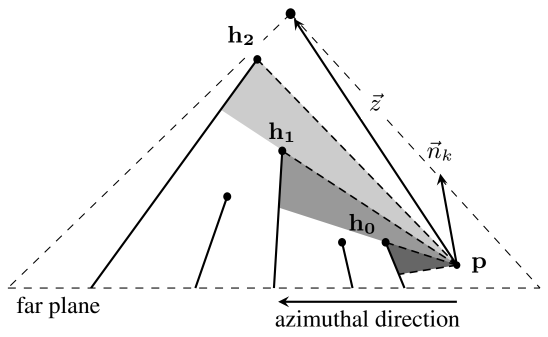


Figure 3: *A sequence of* $I_k = 3$ *partially visible slabs along sector* k. *Points* $\mathbf{h_i}$ *are the top points of the visible slabs. The strength of the falloff term is noted by shading.*

require a slab for every visible point in the depth field along the azimuthal direction to represent the integral in Eqn. 2 *exactly*, we note that only a few (less than 10) points are required to give values not more than 1% off from the ray traced values if the points are chosen carefully, which we will show in Section 8.

The piecewise evaluation of Eqn. 2 in I_k visible slabs now becomes:

$$\int_0^{\pi}\rho(d(\mathbf{p},\vec{\theta}_k))\max(0,\vec{n}_k\cdot\vec{\theta}_k)sin\theta d\theta \approx$$
$$\sum_{i=0}^{I_k-1}||\vec{n}_k||\left(L(a_i,b_i,c_i,d)-L(a_{i-1},b_i,c_i,d)\right), \qquad (3)$$
$$a_i = \angle(\mathbf{h_i}-\mathbf{p},\vec{z}),$$
$$b_i = ((\mathbf{p}_y-\mathbf{c}_y)(\mathbf{h_i}_x-\mathbf{c}_x)-(\mathbf{p}_x-\mathbf{c}_x)(\mathbf{h_i}_y-\mathbf{c}_y))/||\mathbf{h_i}-\mathbf{c}||,$$
$$c_i = \angle(\mathbf{h_i}-\mathbf{c},\vec{z}), d = \angle(\vec{n}_k,\vec{z})$$

where L is a 4D pre-calculated table. The four arguments of L, in order, are

1. The angle a_i of the vector from the receiver to the sampled scene point

2. The closest distance b_i from the receiver to the line formed by the slab, $\mathbf{h_i} + t(\mathbf{h_i} - \mathbf{c})$

3. The angle c_i of the slab

4. The angle d of the projected normal

where all angles are with respect to the zenith vector $\vec{z} = \mathbf{c} - \mathbf{p}$. L is constructed in an offline pre-pass by generating the corresponding slabs and evaluating the falloff weighted integral numerically by ray casting. We have implemented L as a 3D texture where c_i and d share an axis. The sharing is implemented by first splitting the axis into segments, one for each discretized value of d, and then placing consecutive values of c_i consecutively within each segment. Therefore a_i, b_i, and c_i can be linearly interpolated and d is chosen as the nearest discretized value. Although L is of high dimensionality, the involved functions are very smooth and therefore low resolutions are sufficient which helps to keep the texture size moderate (within a few MB).

5 Reference method: mipmap

Our obscurance estimator needs as input the scene points along azimuthal directions in the depth buffer for each receiver. In the simplest case these points can be direct depth buffer samples deprojected into eye-space. While this is the most widely taken approach in current SSAO methods, it does not scale well to far-field: Dense sampling translates into high render times and sparse sampling causes undersampling artefacts because important geometry might be missed.

The approach taken by previous state-of-the-art SSAO methods is to generate a depth pyramid (mipmap) that has a series of lower resolution levels of the depth buffer. Regardless of the filter used to generate the lower resolution levels from the base level, this approach does not retain the view-dependent information of the depth field necessary for accurate AO. We tried several filters including the ones covered in [McGuire et al. 2012] and considered averaging to produce the best results as it does not introduce sudden changes to obscurance like, for example, max-mipmaps do. However, when the depth field cannot be assumed continuous, only points in the original depth buffer can be used. In this case we found max-mipmaps to perform best and we use them in Section 11. Until Section 11 we assume a continuous depth field and compare our geometry representation against averaged mipmaps.

In the mipmap method, from each receiver point, we start traversing the surrounding depth buffer along each of the K azimuthal directions by first sampling the base level texture at a distance of one texel. After each sample, the step size is multiplied by a constant and the sampling distance is accumulated by the step size. This yields an exponentially sparser sampling. We found the constant of 1.5 to produce the best performance-quality tradeoff when used with $K = 16$ and these parameters are used in Section 8.

We use trilinear filtering available in hardware, and choose the mipmap level in such a way that the sample's coverage of the depth buffer matches the sector's width at any given sampling distance. In order to avoid sudden changes in obscurance when the last sampling position goes outside the depth buffer, an extra sample is always taken at the very edge of the depth buffer.

6 Intermediate geometry along scanlines

In this section we describe how our method scans the depth buffer in order to create an intermediate representation of the depth field. We scan the depth buffer in a dense set of parallel lines for each K azimuthal direction and incrementally track the depth field profile along those lines. The parallel lines are spaced one texel width apart along the depth buffer and the lines are traversed one texel width step at a time. In a threaded implementation one thread processes one line. A scan along one azimuthal direction in a depth buffer is shown to the left in Figure 4.

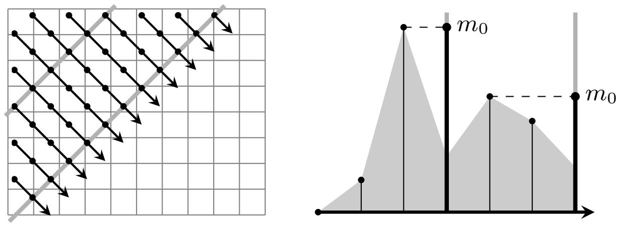


Figure 4: *The depth buffer (grid shown in the background) is scanned in one of the K azimuthal directions in parallel lines (arrows). The maximum height m_0 of each line every $B_0 = 3$ steps (thick gray lines) is written to a buffer holding the intermediate data. Progression along one line is shown to the right.*

We base our method on the intuition that points important for AO are local peaks in the depth field. To track these local peaks, we find the highest (nearest to the camera) depth field points along the lines. In practice, we step through each line incrementally and at each point we sample the depth buffer and deproject the scene point into eye-space. The maximum height value is then remembered along the line until B_0 steps have been taken. After B_0 steps the maximum height is written into an intermediate geometry buffer and reset. This is illustrated to the right in Figure 4. The process is repeated until the end of the depth buffer.

However, which local peak has the highest contribution to AO is dependent on the angle at which the receiver views the peak. The maximum height value is guaranteed to represent the highest horizon value for a receiver that is at the same height, i.e. when the peak is viewed directly horizontally. However, receivers (points along the line) may reside at various heights and therefore view the depth field peaks from different angles. Instead of storing the max height value as viewed directly horizontally, we store 2 max height values: one as viewed horizontally (m_o) and one as viewed at a downwards angle (m_1). We call the angles along which the max height values are viewed *receiver angles*. This is illustrated to the left in Figure 5.

During the evaluation of the obscurance estimator the intermediate geometry buffer is read and a virtual point is reconstructed at the intersection of the receiver angles as shown to the right in Figure 5. Intuitively this virtual point is a view-dependent (for a likely receiver) approximation of the highest peak within the interval of B_0 steps along the scanning line. After culling the invisible points per receiver, these points can be directly used as h_i by the obscurance estimator in Eqn. 3.

We have chosen to use slopes 0 (horizontal) and -1 (45 degrees downward) for the receiver angles. We found that the choice for the receiver angles does not make a large difference to results, however it is important that there are two different angles such that the reconstructed virtual point will have both a depth and a distance value. Algorithm 1 lists the pseudocode for scanning one line in the depth buffer.

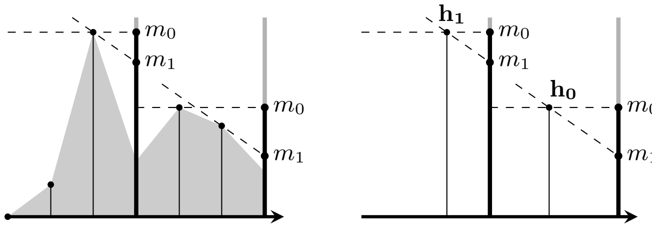


Figure 5: *The maximum heights (denoted by m_0 and m_1) as viewed along 2 receiver angles are written to the intermediate buffer every $B_0 = 3$ steps. The virtual points ($\mathbf{h_i}$) as geometry used by the obscurance estimator are reconstructed at the intersection of the corresponding receiver angles positioned at m_0 and m_1, as shown to the right.*

Algorithm 1 ScanLine(float2 pos, float2 dir, int steps, int lineNo)

Pos *is the coordinate of the first step in the depth buffer and* dir *is a vector for one step along the scanline.*

```
1   float m_0 = −∞
2   float m_1 = −∞
3
4   while (steps−−)
5   {
6       float3 p = deProj(sampleDepth(pos), pos)
7       // p is projected onto k:th azimuthal plane
8       float2 p_k = (p.xy · dir, p.z)
9
10      m_0 = max(m_0, p_k.y)
11      // s = slope of the downwards receiver angle (−1)
12      m_1 = max(m_1, p_k.y + s·p_k.x)
13
14      if (steps modulo B_0 == 0)
15      {
16          // iBuf = the intermediate buffer (output of this stage)
17          iBuf[lineNo][steps/B_0] = (m_0, m_1)
18          m_0 = −∞
19          m_1 = −∞
20      }
21
22      pos += dir
23  }
```

Algorithm 2 lists the pseudocode for evaluating obscurance at one screen pixel along one azimuthal direction. Since SSAO is separable in azimuthal directions, Algorithms 1 and 2 can be calculated sequentially for each K, in which case our method requires $O(W_0 \cdot H_0/B_0)$ space for the intermediate geometry buffer, where W_0 and H_0 are the depth buffer dimensions including guard bands. For a typical case of $W_0 = 1280+256$, $H_0 = 720+144$, $B_0 = 10$ this is roughly 1 MB. If the application is not memory constrained it is faster to evaluate Algorithm 1 for all K simultaneously as to maximize the number of concurrent threads and therefore improve utilization of a GPU. For $K = 16$ the respective memory requirement becomes 16.2 MB.

The accuracy of our intermediate geometry becomes progressively better compared to mipmaps when the interval size increases. Due to the falloff function occluders far from the receiver get less weight and also map to smaller swaths of the horizontal angle than nearby occluders. Therefore it is sensible to construct occluders progressively more sparsely when farther from the receiver. Similarly

Algorithm 2 EvalObscurance(float2 pixelPos, float2 dir)

```
int lineNo = find the line with direction dir closest to pixelPos          1
int iVal = find the nearest interval in lineNo that is at least B_0        2
    steps from pixelPos
                                                                          3
float3 p = deProj(sampleDepth(pixelPos), pixelPos)                        4
// zScale scales z onto the slanted azimuthal plane                       5
float zScale = $\sqrt{1 + (p.x/p.z \cdot dir.y - p.y/p.z \cdot dir.x)^2}$  6
float2 p_k = (p.xy · dir, p.z·zScale)                                     7
                                                                          8
float (AO, maxAngle) =                                                    9
    EvalNearField(pixelPos, dir, distance to iVal)
                                                                          10
while (iVal ≥ 0) {                                                        11
    float (m_0,m_1) = iBuf[lineNo][iVal]                                  12
    // s = slope of the downwards receiver angle                          13
    float2 h = ((m_0 − m_1)/s, m_0·zScale)                                14
    float angle = $\angle((h - p_k), -\vec{p_k})$ // c is at the origin    15
    if (angle > maxAngle)                                                 16
    {                                                                     17
        // Obs(a_i, a_i-1, h_i, p) evaluates i:th segment from Eqn. 3     18
        AO += Obs(angle, maxAngle, h, p_k)                                19
        maxAngle = angle                                                  20
    }                                                                     21
    iVal−−                                                                22
}                                                                         23
                                                                          24
return AO                                                                 25
```

to building multiple resolutions of the depth field in the form of mipmaps, our intermediate geometry can be made into a 1D pyramid. We form levels of the intermediate geometry such that their intervals increase exponentially from the base level's, B_0. Therefore the interval of level n is $B_n = B_0 \cdot 2^n$. The levels can be efficiently generated by taking the max m_0 and m_1 from the two corresponding lower level intervals. Generating the exponential hierarchy roughly doubles the required space but reduces the per-pixel time complexity from $O(n)$ to $O(log(n))$ where n is the pixel distance from the receiver to the edge of the depth buffer.

7 Averaging sectors

In Section 6 we described how to construct virtual points for the obscurance estimator defined in Section 4 from geometry along a single line in the depth buffer. We propose this approach when it is not possible to average or interpolate depth field values, which is the case in Section 11 where the depth field is assumed to represent a volume of a finite thickness. In this section we assume that the depth field is continuous and averaging is thereby allowed.

Ideally the virtual points should represent the entire sector instead of the thin texel wide line along the center of the azimuthal sector. The sector's width increases linearly in the distance from the receiver as demonstrated in Figure 6. In Figure 6 lines contributing to the obscurance at $\mathbf{p}$ are shown as arrows. The horizontal intervals are equal to the gray lines perpendicular to the scanning direction previously shown to the left in Figure 4. In order to construct the averaged point for the highlighted middlemost interval shown in Figure 6, we simply average m_0 and m_1 over the parallel lines $l_A...l_B$ fitting into the sector at that specific distance: $(m_0^a, m_1^a) = 1/(l_B - l_A + 1) \cdot \Sigma_{l_i=l_A}^{l_B} \text{iBuf}[l_i][iVal]$. When the virtual point is constructed (line 14 in Algorithm 2) m_0^a and m_1^a are used instead of m_0 and m_1. In order to calculate the average in

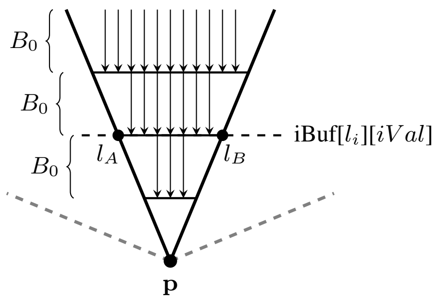


Figure 6: *The downward arrows denote scan lines along one azimuthal scanning direction. Lines indexed $l_i \in [l_A, l_B]$ at the highlighted interval (constant index $iVal$) fit into the sector from receiver $\mathbf{p}$ and their m_0 and m_1 should be averaged.*

constant time, we turn the buffer iBuf into a per-interval prefix sum iBuf^P such that iBuf$^P[l_i][iVal] = \Sigma_{l=0}^{l_i}$iBuf$[l][iVal]$. From the prefix sum the average over any line range $l_0...l_1$ for interval $iVal$ can then be efficiently calculated as (iBuf$^P[l_1][iVal]$ − iBuf$^P[l_0 - 1][iVal])/(l_1 - l_0 + 1)$.

We therefore introduce another stage between Algorithm 1 and 2 which traverses the intermediate geometry buffer iBuf *perpendicularly* to the scan direction in Algorithm 1 and accumulates m_0 and m_1 values over lines. This stage produces the prefix summed version iBuf^P which can, in fact, be built in-place over the original iBuf.

Finally, when evaluating obscurance, instead of averaging the intervals across the entire sector width, the obscurance can be evaluated in multiple segments to increase azimuthal resolution. While doing so does not produce results quite as accurate as if the number of sectors K is increased by a corresponding factor, evaluating a sector in multiple segments is computationally lighter than increasing the number of sectors and has the same effect on reducing banding. More importantly, evaluation in multiple segments does not require extra azimuthal scans over the depth buffer. We have chosen to split each sector in half and evaluate obscurance in 2 segments per sector. From now on, we denote this by adding a multiplier to K, e.g. $K = 8 \times 2$ for eight azimuthal directions and two segments.

8 Results

We ran our algorithm on AMD Radeon HD 7970 (OpenCL) and NVIDIA GeForce GTX 580 (CUDA). Sources are available under the BSD license online at `http://wili.cc/research/ffao/`. The mipmap method using our obscurance estimator and Horizon-Based Ambient Occlusion (HBAO) [Bavoil et al. 2008] are implemented as OpenGL fragment shaders. Performance and quality comparison between other recent SSAO methods can be found in [Vardis et al. 2013] and [McGuire 2010].

Our algorithm calculates the far-field SSAO in 3 kernels:

1. The *Scan* kernel scans through the depth buffer in K azimuthal scanning directions in parallel lines and finds local peaks of the depth field at regularly spaced intervals.
2. The *Prefix sum* kernel reads through the values over multiple lines in a direction perpendicular to the scan direction and generates prefix sums.
3. The *Obscurance* kernel reconstructs virtual points from the prefix sums that are averaged over the azimuthal sector width at each screen pixel. The final obscurance value per pixel is calculated by evaluating the obscurance estimator with the virtual points.

We have chosen two scenes as our main test material. The first scene is an architectural scene with simple planar geometry (Figure 7, top), and the second scene shows complex geometry and foliage (Figure 7, bottom). The scenes are rendered using exponentially decreasing falloff functions whereby in the first scene the falloff function decays slower than in the second scene. All renderings use a 10% guard band (extending 10% of the visible framebuffer width or height at each side) which is denoted by postfixing it to the resolution in parentheses.

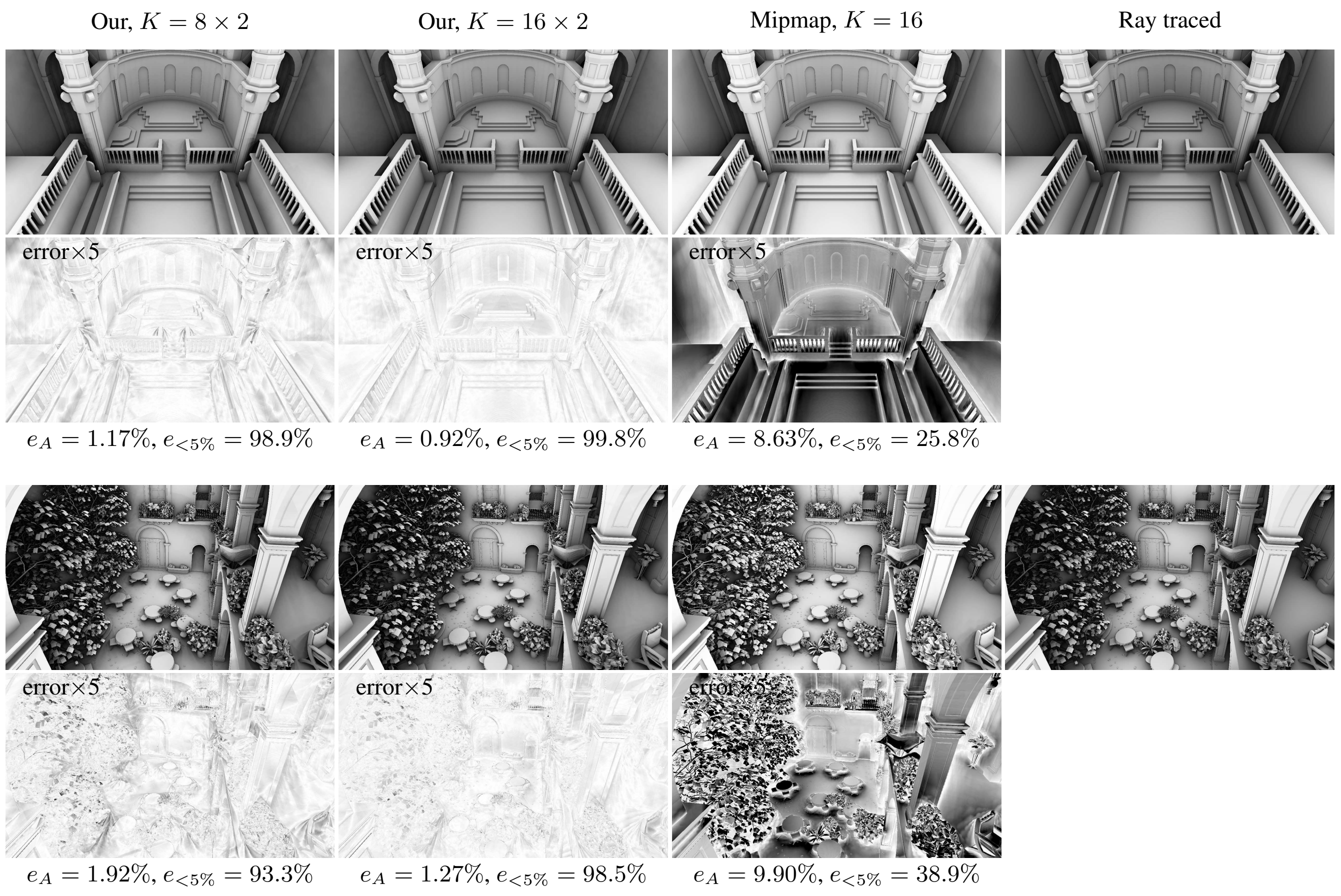
For our method we use $K = 8 \times 2$ and $K = 16 \times 2$ and for the mipmap method we use $K = 16$. As reference we use ray tracing on the same geometry (a single-layer depth buffer with a 10% guard band). In the ray traced result rays with cosine-weighted directions are cast around the hemisphere for each receiver pixel, and stepped through in small steps until geometry is being intersected. The intersection distance is then weighted by the falloff function and accumulated to the result. This can arguably be considered the best result any method can do with the available screen-space data.

In addition to difference images, we measure the error using two metrics: e_A measures the average per-pixel variation from the ray traced values and $e_{<5\%}$ measures the number of pixels within 5% of the ray traced values. A high value in $e_{<5\%}$ denotes that only few pixels behave abnormally, which also implies temporal stability since the reference values do not wave or flicker. In Figure 7 we have rendered the two scenes using our method and the mipmap method and compared the results against the ray tracing.

Recall that our obscurance estimator conservatively assumes a scene point to represent a slab of geometry which extends along the negative zenith. However, actual depth field geometry between slabs can be closer to the receiver. The error coming from the overestimated distance is relative to the density of the slabs, which we in this section keep constant at roughly 8 slabs per azimuthal direction. The average error introduced by this in the first scene is $e_A \approx 0.6\%$ and in the second scene $e_A \approx 0.8\%$, which sets a lower bound for the error as K is increased. Obscurance as estimated from the geometry produced by our method is very accurate even when a low number of sectors is being used, mainly because the evaluated virtual scene points are tailored to capture the features of the scene geometry that specifically contribute to AO. The mipmap method is inadequate in capturing the "profile" of the geometry within the sample's radius and produces erroneous results. Furthermore, the mipmap method converges to the ray traced values very slowly: It takes over 300 ms to achieve the same level of error as in our method at $K = 8 \times 2$ and several seconds to match $K = 16 \times 2$.

While the quantitative error in our method is small, there can still be banding that is perceptually prominent. The level of banding depends on the geometric content: Bands are cast by sharp tall edges and are visible on planar surfaces. Averaging the virtual points over the widths of each sector reduces banding, especially from occluders far from the receiver where the banding is almost completely removed. Banding is discussed in more detail in Section 10. Temporal coherence can be a major concern in SSAO methods that exhibit undersampling, whereas the dense azimuthal scans employed by our method do not skip geometry. Overall we observe that the results of our method look temporally stable (under motion) which is to be expected given the small variation with respect to the stable ray traced values.

In comparison, Figure 8 shows results as rendered by HBAO using the same falloff function. HBAO does not assume a continuous



Our, $K = 8 \times 2$
Our, $K = 16 \times 2$
Mipmap, $K = 16$
Ray traced
error×5
error×5
error×5
$e_A = 1.17\%$, $e_{<5\%} = 98.9\%$
$e_A = 0.92\%$, $e_{<5\%} = 99.8\%$
$e_A = 8.63\%$, $e_{<5\%} = 25.8\%$
error×5
error×5
error×5
$e_A = 1.92\%$, $e_{<5\%} = 93.3\%$
$e_A = 1.27\%$, $e_{<5\%} = 98.5\%$
$e_A = 9.90\%$, $e_{<5\%} = 38.9\%$

Figure 7: *Two scenes rendered by our method and the mipmap method and their respective error images (white = 0%, black $\geq$ 20%, brighter is better), average error (e_A, lower is better) and the number of pixels within 5% ($e_{<5\%}$, higher is better) of the ray traced reference.*

depht field. Instead, it assumes that geometry between two consecutive visible points along an azimuthal direction is at the same distance from the receiver as the higher of the two visible points. As HBAO's obscurance estimator is not built to converge to Eqn. 1 results look dissimilar to the ray traced reference. HBAO requires very many samples per pixel to cover far-field effects accurately which shows up as impractically high render times.

Let $W_0 \times H_0$ be the resolution of the depth buffer with guard bands, and $W \times H$ without. Then the time complexity of our method for kernel 1 is $O(K \cdot W_0 \cdot H_0)$, for kernel 2 $O(K \cdot W_0 \cdot H_0/B_0)$, and for kernel 3 $O(K \cdot W \cdot H \cdot log(W_0 + H_0))$. We consider the scaling favorable, as the per-pixel cost increases only logarithmically in the resolution while the full depth field is still considered for each pixel.

Our method is insensitive to the geometric content of the depth buffer and the obscurance radius has no effect on the render times; obscurance is gathered from the entire guard banded depth buffer for every pixel. Table 1 lists the total execution time of the second scene in Figure 7 for our method and for the mipmap method using two different GPUs and two common screen resolutions. All timings only include far-field AO, which starts at approximately $1.5B_0$ pixels from the receiver. The same far-field boundary is used for both methods.

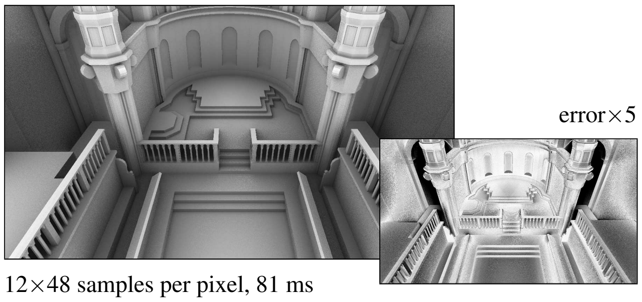
Table 2 shows how the execution time is split between the three stages of our method. In the Obscurance kernel, we measure the average number of constructed virtual points to be 7.8 per sector per pixel. The mipmap method takes an average of 8.5 samples per sector per pixel.

Table 1: *Total render times of the far-field AO component.*

Method	Radeon 7970	GTX 580
$1280(+256) \times 720(+144)$, $B_0 = 10$:		
Our, $K = 8 \times 2$	7.26 ms	12.0 ms
Our, $K = 16 \times 2$	13.3 ms	23.6 ms
Mipmap, $K = 16$	19.2 ms	17.7 ms
$1920(+384) \times 1080(+216)$, $B_0 = 10$:		
Our, $K = 8 \times 2$	16.7 ms	29.4 ms
Our, $K = 16 \times 2$	31.6 ms	58.1 ms
Mipmap, $K = 16$	31.5 ms	37.9 ms

Table 2: *Render time breakdown of our method per kernel.*

Phase	Radeon 7970	GTX 580
$1280(+256) \times 720(+144)$, $K = 8 \times 2$, $B_0 = 10$:		
Scan	0.537 ms	0.489 ms
Prefix sum	0.945 ms	0.617 ms
Obscurance	5.77 ms	10.9 ms



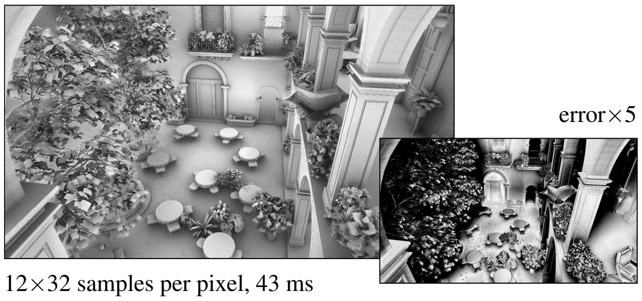


Figure 8: *Scenes from Figure 7 as rendered by HBAO on a GeForce GTX 580 at 1280(+256)×720(+144). Obscurance is calculated in $K = 12$ azimuthal directions that are randomly rotated per pixel.*

9 Integration with near-field

From a time complexity point of view our method is effective in treating near-field obscurance as well, however our method's benefits become significant only when the geometry enclosed by the interval B_i covers a large distance. In order to bring the nearest interval in our method closer to the receiver, B_0 has to be reduced, which increases the execution time and the memory footprint. We suggest that our method be combined with a lightweight near-field search that gathers obscurance from an area around the receiver that is at least a couple of pixels in radius. Where exactly the boundary between the near-field and our method should be depends on the characteristics of the near-field search: Our method should generally take over at a distance where the near-field method is no longer faster. The interval of the base level, B_0, determines the nearest distance at which our method can take over. Halving B_0 causes one extra interval per sector to be evaluated (roughly a constant increase in execution time), and reduces the area of influence of the near-field search to quarter. Table 3 lists our method's execution times for different values of B_0.

Table 3: *Far-field render times of our method using different values for the base level interval B_0.*

Base level	Radeon 7970	GTX 580
$1280(+256) \times 720(+144)$, $K = 8 \times 2$:		
$B_0 = 20$	6.06 ms	9.77 ms
$B_0 = 10$	7.26 ms	12.0 ms
$B_0 = 5$	8.89 ms	14.7 ms

When integrating a near-field method with our method, the near-field method should be executed first and provide the maximum horizon angles from the near-field range along the K sectors for each pixel as shown at line 9 in Algorithm 2. After this, our method continues accumulating the obscurance from the horizon angle upwards until the edge of the depth field. We expect $B_0 \in [5, 10]$ to be a suitable choice for a typical state-of-the-art near-field search. Figure 9 shows the contribution of the far-field and the near-field obscurance components on a 720p depth buffer using $B_0 = 10$.

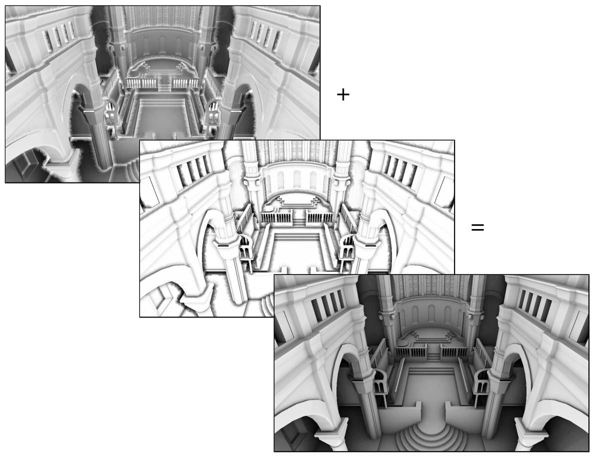


Figure 9: *The far-field component ($\geq$ 15 px) as produced by our method and the near-field component ($<$ 15 px) together form the final ambient obscurance result (bottom).*

10 Banding

While the error in our method is small, as established in Section 8, there still might be some visible banding even though we average occluders across the width of a sector. In order to gain intuition on why banding happens, consider the case where an occluder—say, a wall—enters an otherwise flat sector in a linear motion. If the sector was split into infinitely many subsectors, obscurance would increase linearly as the wall occupied a larger swath of the sector. In our method, the entering wall increases the average *height* of an interval linearly, which might not map to linear change in obscurance. This is especially evident for very tall occluders which cause the obscurance value to increase faster than linearly when only a small portion of the occluder is occupying the sector. So, while the obscurance values at band boundaries in our method do not jump abruptly, they don't follow the physically correct curve either. In Figure 10 we show a split screen of a scene rendered using $K = 8 \times 2$ averaged sectors as described in Section 7 and then using $K = 16$ straight sampling lines that go through the center line of each sector. Averaging eliminates far-field banding almost completely, which is often the hardest to rid, but near-field banding still persists.

One solution to the banding problem is to increase the number of scanning directions K which shows up as a roughly linear increase in execution times of the Scan and Prefix sum stages. Instead of evaluating all K sectors for every pixel, it is possible to evaluate the sectors sparsely. We use the Separable Approximation of Ambient Occlusion (SAAO) approach from [Huang et al. 2011], and evaluate $K = 18 \times 2$ sectors in groups of 3×3 pixels. Obscurance is therefore evaluated in an interleaved pattern such that only $K = 2 \times 2$ sectors are evaluated per pixel and the results are gathered using an edge-aware 3×3 box filter as a post-process. *Any* 3×3 pixel neighborhood includes all $K = 18 \times 2$ sectors and therefore no noise is produced to the image. SAAO produces errors primarily at edges and depth discontinuities in the depth buffer. While the far-field AO component can also change by unbounded amounts

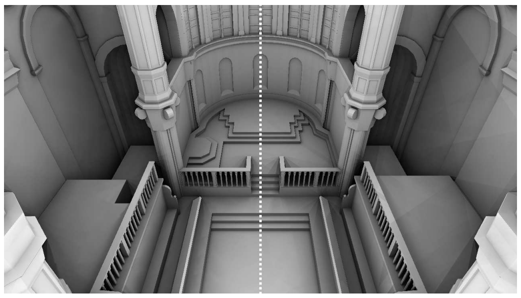


Figure 10: *Left: our method using $K = 8 \times 2$ averaged sectors. Right: our method using $K = 16$ straight sampling lines.*

between adjacent pixels in the screen, the error is mainly in the near-field.

We have incorporated SAAO in our method by combining a 3×3 separated far-field AO with a full near-field AO. The primary artefacts are small integration errors (noise) at the boundary of the far-field and near-field AO components because they are of different sparsity. The error can be hidden to a large extent by a selective blur that uses a small intensity threshold and does not currupt the image. In Figure 11 a result without blurring is shown to the right, which shows minor noise, and the image to the left includes a 5×5 bilateral box blur. Figure 1 is also rendered using the method shown to the left.

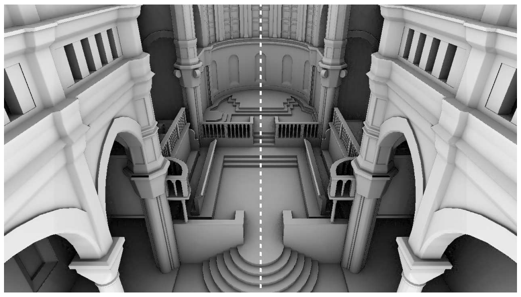


Figure 11: *Our method using $K = 18 \times 2$ with SAAO. Left: selectively blurred result. Right: result without blurring.*

We implement SAAO by introducing two new kernels:

Box average kernel is an edge-aware filter which averages depth and the normal vectors from a 3×3 pixel neighborhood of each framebuffer pixel. If the dot product of the normal of the source pixel and the candidate pixel is higher than 0.5 and the relative difference of pixel depths is below 3%, the pixel is accepted into the average. The accepted pixels are the pixels from which the far-field obscurance is gathered in the Gather kernel, and therefore a bit mask representing the accepted pixels is carried to the Gather kernel. The averaged depth and normal vectors are used in the Obscurance kernel instead of the original pixel's values.

Gather kernel combines the per-pixel near-field obscurance with an average of the far-field obscurance from pixels that were marked as accepted in Box average kernel. Results are optionally blurred using a bilateral box filter of size 5×5 with a threshold of 7% (pixels within a maximum of 7% color difference are included into the average).

The SAAO enabled render times are shown in Table 4. The execu-

Phase	Radeon 7970	GTX 580
$K = 18 \times 2$ (3×3 separation), $B_0 = 10$, $1280(+256) \times 720(+144)$:		
Scan	1.07 ms	1.08 ms
Prefix sum	1.09 ms	1.04 ms
Box average	0.265 ms	0.400 ms
Obscurance, sep.	1.60 ms	3.00 ms
Gather (with blur)	0.255 (0.591) ms	0.290 (0.652) ms
total (with blur)	4.28 (4.62) ms	5.81 (6.17) ms
$1920(+384) \times 1080(+216)$:		
Scan	2.32 ms	3.08 ms
Prefix sum	2.03 ms	2.08 ms
Box average	0.566 ms	0.894 ms
Obscurance, sep.	3.82 ms	7.25 ms
Gather (with blur)	0.557 (1.31) ms	0.651 (1.46) ms
total (with blur)	9.29 (10.0) ms	14.0 (14.8) ms

Table 4: *Render time breakdown of our method per kernel with SAAO enabled.*

tion time of the Obscurance kernel decreases linearly in the number of evaluated sectors. In fact, the Scan and Prefix sum kernels now take more time than the Obscurance kernel in some cases, and speeding up these two kernels would be the logical next step in improving the execution times and is briefly discussed in Section 13. Overall, SAAO is an efficient way to trade banding for noise or blur while also improving render times significantly.

11 Limitations of depth buffer geometry

Scene geometry in a depth buffer is incomplete in two ways: (i) geometry outside the view frustum is unknown and (ii) geometry below the first depth layer is unknown. The first limitation is usually addressed by introducing a guard band around the depth buffer. In this paper we have used a guard band of 10% (extending the depth buffer by 10% of its width or height in each direction). As the screen-space radius of the obscurance effect may become arbitrarily large when scene geometry is close to the camera, it cannot be entirely contained within a guard band in any SSAO method. However, the slower the decay of the falloff function the larger the guard band generally needs to be and thus becomes important to our method. Fortunately the Z pre-pass is usually quick and lower resolution rasterization can be used in the guard bands to further minimize its cost. As long as the Z pre-pass does not have a high cost, we recommend even larger than 10% guard bands when calculating far-field AO effects in screen-space. It is also possible to construct a simplified *world-space* representation of occluders around the camera that are outside the depth buffer and accumulate SSAO with occlusion from them using a global AO method, however this approach is outside the scope of this paper

Most SSAO methods use only a single depth layer and make a generic assumption about the geometry below the nearest depth layer. Such an approach can never produce correct results in all scenes and the artefacts vary. Until this section we have assumed the depth field to be continuous, i.e. an infinitely thick volume. This has the benefits, for example, that depth field points can be averaged and interpolated, and objects appearing behind nearer geometry within the view frustum will not cause abrupt changes to obscurance. The downside is that obscurance is often overestimated, and to a large degree if there are thin objects, such as chains hanging in the air, near the camera. Because we in this paper advocate a high quality and physically correct SSAO, we find more promise in approaches that attempt to fill the missing scene geometry with real information of the scene instead of fitting a scene-dependent assumption. In previous work such information has been introduced in the form of multiple depth layers [Bavoil and Sainz 2009] and multiple views [Vardis et al. 2013]. Extending our method into that direction is left as future work.

Instead, we briefly demonstrate our method under the assumption that the depth field has a fixed finite thickness, an approach taken by many prior works such as [Loos and Sloan 2010] [McGuire et al. 2011] [McGuire et al. 2012]. While this will not work for arbitrary views or scenes that have varied objects, it produces plausible results when the depth field thickness is carefully selected and matches that of the viewed objects. Fixing the thickness requires a small change in the obscurance estimator in Eqn. 3: a_{i-1} is replaced with $max(a_{i-1}, \angle(\mathbf{h_i} + t(\mathbf{h_i} - \mathbf{c})/||\mathbf{h_i} - \mathbf{c}|| - \mathbf{p}, \vec{z}))$ where t is the thickness of the depth field.

When the depth field thickness is finite the depth field becomes discontinuous. Therefore it is not allowed to generate new points through interpolation or averaging. This limitation does not much impact direct depth buffer samples which can be snapped to texel centers as done in [Bavoil et al. 2008]. In our method this means that averaging across the sector's width as described in Section 7 cannot be used, however we still retain the advantage over direct samples that our method tracks the local peaks along each sampling line. The mipmap method, however, is most impacted: Not only is interpolation spatially and across mip levels forbidden, but lower resolution level textures have to reuse values found from the base level and no averaging is possible. Out of various filters we found max-mipmaps to produce best results for mipmapping.

In Figure 12 we show the Stanford Dragon as rendered by our obscurance estimator with a fixed depth field thickness using our intermediate geometry samples, direct depth buffer samples, and max-mipmaps. All methods evaluate roughly 8 far-field samples per azimuthal direction. SAAO is not used. Direct depth field samples and our method produce banding especially since sector averaging cannot be used for mitigation. Therefore our method does not use the Prefix sum stage and reconstructs scene points as per Algorithm 1 and 2 directly. Our method produces smooth obscurance *along* each azimuthal direction whereas direct depth buffer samples produce artefacts depending on whether the samples hit or miss local peaks in the depth field. Due to the missed geometry, direct sampling also produces systematic underocclusion. Max-mipmaps do not miss geometry but systematically overestimate it by always picking the largest occluder within the sample's radius. Also, as linear interpolation cannot be used the results are blocky.

11.1 Jittering

It is possible to jitter the sampling directions per-pixel to trade banding for noise. In our method this can be achieved by adding or substracting an offset value from the sampled line indices l_A and l_B shown in Figure 6. The offset is randomly selected per pixel, and sampling along every direction is offset by the same amount as not to cause bias. The offset is scaled according to the distance from the receiver to the interval. Here, when averaging is not allowed and only a single line is sampled along each azimuthal direction, banding becomes especially severe if jittering is not used. Overall we find that jittering the sampling direction within the sector boundaries efficiently eliminates banding in return for some noise. Our method is the cache-friendliest of the three methods with respect to jittering because neighboring pixels access the same intervals which are laid out in memory consecutively and accesses are likely to hit the same cache lines. Mipmapping exhibits better cache locality than the sparser direct samples and its render times are not impacted significantly by jittering.

12 Conclusion

We have presented a method to solve ambient obscurance in screen-space from occluders that are beyond the immediate neighborhood of the receiving pixel. We do this by first scanning the depth buffer in a number of azimuthal directions while tracking local height maxima and writing them into an intermediate geometry buffer. After creating prefix sums of the intermediate geometry buffer, it can be sampled per-pixel to obtain approximated local peaks in the environment as seen from the receiver point, at various distances. These reconstructed scene points are then evaluated using an obscurance estimator to approximate the AO integral over the receiver's hemisphere. The obscurance effect in our method is only limited by the falloff term, and our method can incorporate any such term without its evaluation affecting render times. Overall our method is able to produce very high quality AO effects that are close to a ray traced screen-space reference.

The intended use for our algorithm is to couple it with a lightweight near-field search to build a robust SSAO solution that accurately integrates ambient obscurance from the entire guard banded depth buffer.

13 Future work

Currently we scan the depth buffer densely, which is justifiable since the Obscurance stage takes most of the execution time and is not impacted by scanning density. However, a dense scan becomes costly when SAAO is used as a significant amount of the total time is spent in the Scan and Prefix sum stages that scale linearly in the number of scanned lines. It is possible to leave out some of the parallel lines during azimuthal scans without much impact on the calculated obscurance because the lines are always approximately facing the receiver and therefore have a limited contribution to AO. Ideally, processing every n:th line reduces the execution time of the Scan and Prefix sum stages by the factor $1/n$.

Overall there are four main strategies to reduce the render time of our method:

- Sparse scans as described above
- Separated obscurance evaluation sparser than 3×3 (which is used in Section 10), such as 5×5
- Reducing K and increasing the number of segments in which each sector is evaluated (e.g. $K = 8 \times 4$)
- Constructing fewer points (larger intervals) per azimuthal direction per pixel.

We are also investigating the possibility of extending our method to handle multiple depth layers [Bavoil and Sainz 2009] or multiple views [Vardis et al. 2013] which—when coupled with sufficiently large guard bands—would alleviate the screen-space problem of missing scene geometry. This could allow our method to produce results comparable to global ray tracing.

References

BAVOIL, L., AND SAINZ, M. 2009. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH '09 Talks*, ACM.

BAVOIL, L., SAINZ, M., AND DIMITROV, R. 2008. Image-space horizon-based ambient occlusion. In *SIGGRAPH '08 Talks*.

HOANG, T.-D., AND LOW, K.-L. 2012. Efficient screen-space approach to high-quality multiscale ambient occlusion. *The Visual Computer 28*, 3, 289–304.

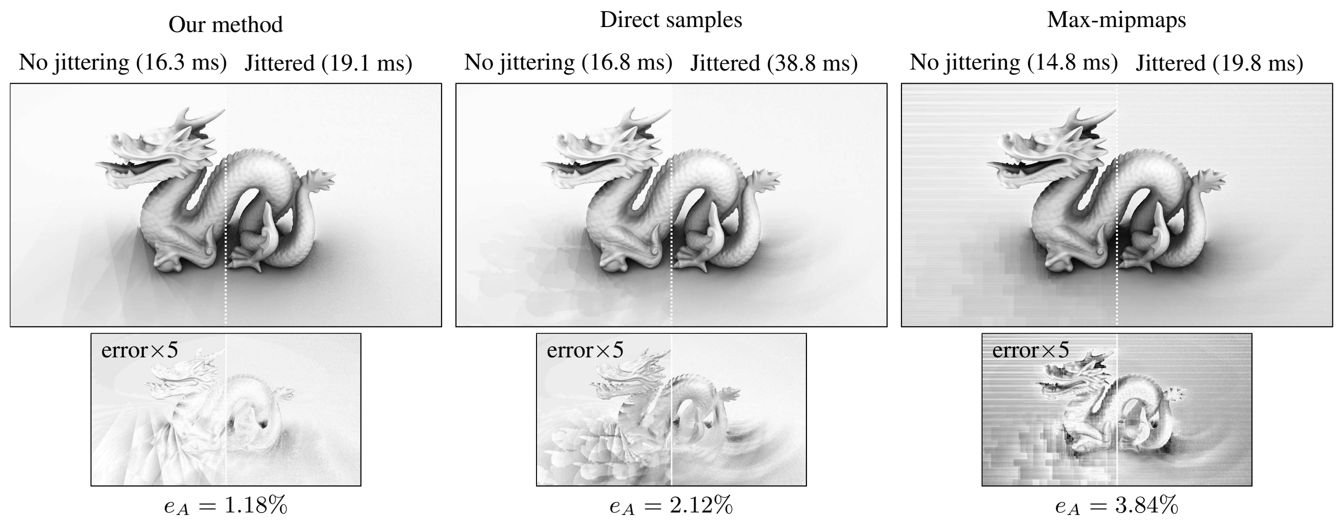


Figure 12: *The Stanford Dragon rendered in $K = 16$ azimuthal directions with a hand-picked thickness t using our intermediate geometry (left), direct depth buffer samples (middle), and max-mipmaps (right). The right side of each image uses azimuthal directions that are randomly jittered per-pixel. The resolution is 1280(+256)×720(+144) and the render times are reported for the far-field ($B_0 = 10$) AO component on a GeForce GTX 580.*

HUANG, J., BOUBEKEUR, T., RITSCHEL, T., HOLLÄNDER, M., AND EISEMANN, E. 2011. Separable approximation of ambient occlusion. In *Eurographics 2011 - Short papers*.

LAINE, S., AND KARRAS, T. 2010. Two methods for fast ray-cast ambient occlusion. *CGF: Proceedings of EGSR 2010 29*, 4.

LOOS, B. J., AND SLOAN, P.-P. 2010. Volumetric obscurance. In *Proceedings of I3D 2010*, ACM.

MCGUIRE, M., OSMAN, B., BUKOWSKI, M., AND HENNESSY, P. 2011. The alchemy screen-space ambient obscurance algorithm. In *Proc. HPG*, ACM, HPG '11, 25–32.

MCGUIRE, M., MARA, M., AND LUEBKE, D. 2012. Scalable ambient obscurance. In *High-Performance Graphics 2012*.

MCGUIRE, M. 2010. Ambient occlusion volumes. In *Proceedings of High Performance Graphics 2010*.

MITTRING, M. 2007. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, ACM, 97–121.

REINBOTHE, C., BOUBEKEUR, T., AND ALEXA, M. 2009. Hybrid ambient occlusion. *EUROGRAPHICS 2009 Areas Papers*.

RITSCHEL, T., DACHSBACHER, C., GROSCH, T., AND KAUTZ, J. 2012. The state of the art in interactive global illumination. *Computer Graphics Forum 31* (Feb.).

SHANMUGAM, P., AND ARIKAN, O. 2007. Hardware accelerated ambient occlusion techniques on gpus. In *Proc. I3D '07*, ACM.

SNYDER, J., AND NOWROUZEZAHRAI, D. 2008. Fast soft self-shadowing on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering* (June).

TIMONEN, V., AND WESTERHOLM, J. 2010. Scalable Height Field Self-Shadowing. *Computer Graphics Forum (Proceedings of Eurographics 2010) 29*, 2 (May), 723–731.

TIMONEN, V. 2013. Line-Sweep Ambient Obscurance. *Computer Graphics Forum (Proceedings of EGSR 2013) 32*, 4.

VARDIS, K., PAPAIOANNOU, G., AND GAITATZES, A. 2013. Multi-view ambient occlusion with importance sampling. In *Proc. i3D*, I3D '13, 111–118.

ZHUKOV, S., INOES, A., AND KRONIN, G. 1998. An Ambient Light Illumination Model. In *Rendering Techniques '98*, Springer-Verlag Wien New York, G. Drettakis and N. Max, Eds., Eurographics, 45–56.