# Solid Angle based Ambient Obscurance in Image Space

Dario Scarpa[1] and Ugo Erra[2]

[1] SpinVector, Benevento, Italy
darioscarpa@gmail.com
[2] Università degli Studi della Basilicata, Potenza, Italy
Dipartimento di Matematica, Informatica ed Economia
ugo.erra@unibas.it

**Abstract.** We derive a new approximation of ambient obscurance to improve the quality of state-of-the-art techniques used in real-time rendering. We attempt to stay close to the original definition of ambient obscurance and, while building on the deferred rendering approach, bring into image-space information that is suitable for accurate estimations of visibility that take account of the position and orientation of near occluding geometry. The approach is based on the approximation of a covered solid angle, considering the area of surfaces, and hemisphere partitioning that gives directional information about coverage, both done in image space. The immediate advantage of our technique is that we avoid over-occlusion caused by multiple occluders covering each other but covering from the same direction. In some cases our implementation achieves lower performance with respect to some currently popular and widely adopted screen-space ambient obscurance approximations, but still obtains real-time frame rates on the current generation of hardware.

**Keywords:** Ambient Obscurance, Screen-space, Hemisphere Partitioning

## 1 Introduction

Shadowing of ambient light is called ambient occlusion. It has been shown in [8] that ambient occlusion offers a better perception of the 3D shape of displayed objects, and its effectiveness is evident in its popularity in videogame engines [11]. The mathematical definition of ambient occlusion is related to the concept of the solid angle. In fact, the occlusion $A_{\mathbf{p}}$ at a point $\mathbf{p}$ on a surface with normal $\mathbf{n}$ can be computed by integrating the visibility function over the hemisphere $\Omega$ with respect to the projected solid angle:

$$A_{\mathbf{p}} = \frac{1}{\pi} \int_{\Omega} V_{\mathbf{p},\omega}(\mathbf{n} \cdot \omega)\, d\omega \tag{1}$$

where $V_{\mathbf{p},\omega}$ is the visibility function at $\mathbf{p}$ along a direction $\omega$. A simple method to approximate this integral in practice, in off-line rendering, is based on ray-tracing. Rays are shot in a uniform pattern across the hemisphere over point

**p**, and an occlusion value can be calculated as the number of rays that hit the geometry divided by the total number of rays shot. Rays can be restricted to a certain length, avoiding distant geometry to be taken into account while calculating the occlusion value. This is fundamental in closed environments, which would otherwise result in total occlusion at every point and subsequently the complete removal of ambient light.

*Ambient Obscurance* (AO) is an extension of ambient occlusion defined in [20]. A falloff function that reduces the influence of occlusion with distance is introduced in the formula:

$$AO_{\mathbf{p}} = \frac{1}{\pi} \int_{\Omega} \rho(D_{\mathbf{p},\omega})(\mathbf{n} \cdot \omega)\, d\omega \tag{2}$$

Comparing this expression with the formula of ambient occlusion , in place of the binary visibility function $V_{\mathbf{p},\omega}$ we have the function $\rho(D_{\mathbf{p},\omega})$, where $D_{\mathbf{p},\omega}$ is the distance between $\mathbf{p}$ and the first intersection point along $\omega$. If there are no intersections along $\omega$, its value is $+\infty$. Here, $\rho$ is the decreasing falloff function, with $\rho(0) = 1$ and $\rho(x) = 0$ if $x > r$, where $r$ is the maximum distance at which any intersecting geometry is considered as producing occlusion.

In this paper, we derive a new approximation of ambient obscurance focused on improving the quality of state-of-the-art techniques used in real-time rendering. Our approach is called Solid-angle Screen Space Ambient Obscurance (SaSSAO). We attempt to remain close to the original definition of ambient obscurance and, while building on the deferred rendering approach, bring into image-space information that is suitable to an accurate estimate of visibility that takes account of the position and orientation of near occluding geometry. The main idea is to track the amount of occlusion from the set of directions. This approach handles the over-occlusion that results from the same directions where thin objects are stacked. To the best of our knowledge, the approximation of a covered solid angle (used in our occlusion estimate) considering the area of surfaces, and the hemisphere partitioning that gives directional information about coverage, both done in image space, are original contributions to the field.

To evaluate the quality of results, we compare screen-shots from our implementation with images rendered off-line in Blender, a popular open-source 3D graphics software that features a configurable ray-traced calculation of ambient occlusion. For the comparison, we use the Structural Similarity Index [19], a metric that attempts to measure similarity between images in a way that is consistent with human eye perception. In some cases our implementation achieves lower performances with respect to some currently popular and widely adopted screen-space ambient obscurance (SSAO) approximations, but still obtains real-time frame rates on the current hardware generation. Moreover, it offers many parameters that can be adjusted to trade quality for efficiency.

## 2 Related Works

Real-time global illumination is a "hot topic" in computer graphics research, and an impressive number of related works have been published. The 2012 survey [14]

provides a general overview of the field, and works about ambient occlusion offer further insights. Here, we restrict the scope to techniques most closely related to our own: ambient occlusion/obscurance approximations suitable to real-time rendering that operate in image space (also called screen space).

Even considering only this category of algorithms, a variety of approaches exist. Some sources attempt to correlate, compare and evaluate such techniques, and the interested reader may like to consult [1] and [6], two recent theses that both agree that the Alchemy algorithm is the current state of the art. In the following sections, we briefly cover the more influential works.

In his seminal work [3], Micheal Bunnell of NVIDIA corporation describes a technique that approximates polygon meshes as a set of surface elements (discs) that can emit, transmit, or reflect light and that can shadow each other. He defines an approximation of ambient occlusion on the basis of the calculated coverage between discs, and an approximation of indirect lighting by estimating the disc-to-disc radiance transfer. Major drawbacks of this algorithm are the dependence on scene complexity and the need to preprocess geometry, which must be well tessellated to give good results. This also implies that the technique is not suitable for deformable objects. Note that this is not an image-space technique, but a geometry approximation one. Nevertheless, we mention it because it has to some extent inspired subsequent works, including our own.

In [16] Shanmugam and Arikan approximate ambient occlusion through spherical proxies. Their interesting idea is to reconstruct approximately the surface represented by a pixel using a sphere in world-space that roughly projects to that pixel on the screen. By using deferred rendering, a ND -buffer storing normals and depths is created. From this information, each pixel can be mapped to a sample of some surface in world-space, which can be considered as an occluder to other pixels. The algorithm also uses a separate calculation (non-screen space) for low-frequency occlusion due to distant occluders, and then combines the results. Despite its original and interesting ideas, this technique is perhaps over-complicated and had little success and was subsequently surpassed in both quality and speed by simpler techniques.

In [11] the denomination "screen-space ambient occlusion" appears for the first time. In its CryEngine, Crytek implements this technique, which works by sampling the surroundings of a pixel and, on the basis of the z-buffer, performs depth comparisons. Sample positions are distributed in a sphere around the pixel, and some randomness is introduced by reflecting position vectors on a random plane passing through the sphere origin. The occlusion factor depends only on the depth difference between sampled points and the current point. This, combined with the simple distribution of samples (around a sphere and not a hemisphere) causes some over-darkening: even flat, non-occluded areas result in some samples considered as occluders.

Some improvements over the CryEngine approach are shown in [5]. Samples are offset in 3D space from the current point being computed, and then projected back to screen space to sample the depth of the sample location. Normals (this algorithm is also based on deferred rendering) are used to flip the vectors that fall

in the hemisphere below the current point, avoiding the self-occlusion exhibited by the Crytek algorithm. An occlusion function maps the relationship between the depth delta and the amount of occlusion. A number of details are taken care of (sample randomization, filtering, down-sampled calculation) to improve performance and to obtain a production-ready solution.

The method in [2] interprets the depth buffer as a height field and works by performing a type of ray marching in screen space. It considers the tallest occluder along each azimuthal direction to determine the visible horizon on the hemisphere around the current point. This assumes a continuous depth buffer, so the occlusion does not take into account the unoccluded portion of hemisphere in case of floating occluders. Subsequent refinements of the method have been published, but the method remains expensive in relation to the quality it produces.

The Alchemy SSAO algorithm, presented in [10], has been developed with the goal of artistic expressiveness rather than physically grounded realism. The strength of Alchemy is in the way it derives its estimator: the chosen falloff function cancels some expensive operations while staying meaningful. The obtained highly efficient estimator is then applied to points sampled in the hemisphere, as in some previous methods . Alchemy features a number of artist-tweakable parameters and generally gives good quality results with high performance. Some improvements and modifications of the algorithm are discussed in [9].

## 3 Hemisphere Partitioning

To derive an approximation close to ambient obscurance into image space, we adopt a scheme to discretize the sphere into solid angles as proposed in [7]. Let us consider a unit sphere centred on the origin of Euclidean space. The origin divides each of the axes into two halves: a positive and a negative semi-axis. Let us refer to the slice of sphere delimited by the three positive semi-axes as the positive octant of the sphere (see Figure 1).

Let us consider the $x + y + z = 1$ plane and the equilateral triangle that lies on it with vertices $\mathbf{v_0} = (1, 0, 0)$, $\mathbf{v_1} = (0, 1, 0)$, and $\mathbf{v_0} = (0, 0, 1)$.

We split each of the $\overline{\mathbf{v_0 v_1}}$, $\overline{\mathbf{v_0 v_2}}$, and $\overline{\mathbf{v_1 v_2}}$ edges into $n$ equal units. For edge $\overline{\mathbf{v_0 v_1}}$, we connect subdivisions between the other two edges with line segments parallel to $\overline{\mathbf{v_0 v_1}}$. Then, we repeat the same process for the remaining two edges, obtaining a tessellation of the original triangle into $n^2$ triangles. The process is illustrated in Figure 2.

If we project onto the surface of the sphere the vertices of this tessellation, by normalization we obtain a partition of the octant into spherical triangles. Each of these spherical triangles represents a solid angle $\omega$, associated with the direction $\Theta_\omega$ passing through the triangle centroid. Individual triangles are assigned unique identifiers, as shown in Figure 2.

Given an arbitrary direction $\vartheta$ starting from the sphere's centre, we can identify the associated triangle in constant time by using the procedure illustrated

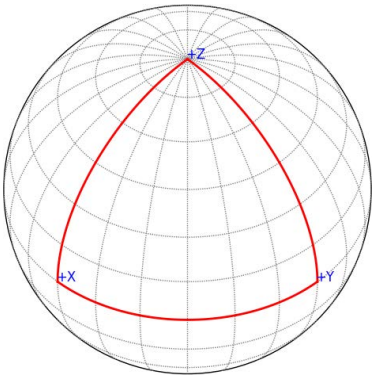


Fig. 1: The surface above the positive octant of a sphere.

in [7]. Then, from the unit vector along $\vartheta$, the procedure takes as input its intersection point $p_\vartheta$ with the $x + y + z = 1$. We report the procedure for simplicity below.

**Algorithm 1** getTriangle

**procedure** GETTRIANGLE($p_\vartheta$)
$x = np_\vartheta$; $z = np_\vartheta$
$x_i = \lfloor x \rfloor$; $z_i = \lfloor z \rfloor$
$\xi_\vartheta = z_i(2n - z_i) + 2x_i$
$diag = (x - x_i) + (z - z_i)$
**return** $\xi'_\vartheta = \xi_\vartheta + \lfloor diag \rfloor$

For the ambient obscurance calculation, we are interested only in directions associated with the hemisphere "surrounding" the normal of the point, so we repeat the process for four octants, building a pyramid. We have shown how for $n$ subdivisions we obtain $n^2$ triangles, so for four octants we obtain $4\,n^2$ total triangles. The triangle identifiers follow the pattern shown in Figure 2, but with an additional offset added, depending on the slice of hemisphere to which they are related: for example, the third slice will have triangle identifiers ranging between $2n$ and $3n - 1$.

The solid angle covering the full hemisphere is $2\pi$, so the solid angle associated with every "bucket" is

$$\omega = \frac{2\pi}{4n^2} \tag{3}$$

The algorithm to find the bucket associated with any direction vector is easily adapted: the signs of the vector coordinates indicate in which of the four octants of the hemisphere we have to look.

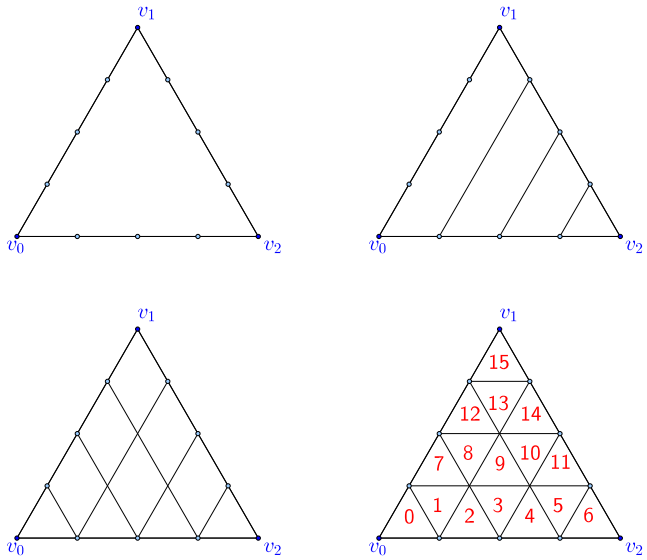


Fig. 2: Steps of triangle subdivision for $n = 4$. The last triangle has numeric identifiers for each individual triangles.

## 4 Solid-Angle-based Ambient Obscurance

On a higher level, our approach proceeds in three passes. In the first pass, the geometry shader computes an area value relative to each triangle processed, then forwards it to the fragment shader, which saves it in the G-buffer together with normals and depths. In the second pass, for each pixel, a fragment shader samples the G-buffer to calculate the ambient obscurance (AO). In the third pass, the AO-buffer is filtered to lower the noise caused by the sampling and used to modulate the ambient factor in the final compositing of the rendered image. In the following sections, we provide details of each pass.

### 4.1 Area calculation

Whereas the vertex shader operates on a per-vertex level, the geometry shader can access whole primitives (in our implementation, we use only triangles). So, for each triangle, the camera space position of its vertices is used as a basis to compute an area that will later be used in the AO calculation. The main idea is to approximate triangle meshes as a set of circumscribed circles or of inscribed circles that will be used as occluders for calculating the AO for a given point.

Given a triangle in the camera space position, we calculate the lengths of its sides, $a$, $b$, and $c$. Then, by using the Heron formula, in which $s$ is the semi-perimeter of the triangle, we can calculate the triangle area $A_t$:

$$A_t = \sqrt{s(s-a)(s-b)(s-c)} \tag{4}$$

We can calculate additional quantities related to the triangle, such as the area of the circumscribed circle $A_{cct}$ as

$$radius_{cct} = \frac{abc}{4A_t} \Rightarrow A_{cct} = \pi \cdot radius_{cct}^2 \tag{5}$$

or the area of the inscribed circle $A_{ict}$ as

$$radius_{ict} = \frac{2A_t}{(a+b+c)} \Rightarrow A_{ict} = \pi \cdot radius_{ict}^2 \tag{6}$$

Further area calculations are possible, and as we show in the following, these can be taken as a configurable parameter.

### 4.2 Ambient obscurance calculation

The second pass operates in image space, accessing the G-buffer created in the first pass. Specifically, we can retrieve some geometry-related information from the G-buffer according to a sampling pattern, then use it in our calculation of the ambient obscurance.

For a pixel $p$, at screen coordinates $x_p$ and $y_p$, from the G-buffer we have the following data:

- $d_p$: the depth of the pixel $p$, from the z-buffer, normalized to [0..1]
- $\mathbf{n}_p$: the normal of the geometry surface at $p$
- $\mathbf{c}_p$: the camera space position of $p$
- $a_p$: the area related to the triangle to which $p$ belongs, calculated as described in Section 4.1.

Selection of the screen-space positions to take samples for calculating the AO for point $p$ is important. Basically, two categories of approaches are possible, both involving a radius around $p$, named *samplingRadius*, often scaled by $d_p$, which limits the distance samples can be taken. The flat sampling locates points around $p$, in a circle of radius *samplingRadius*, considering the 2D screen-space coordinates $x_p$ and $y_p$. While the 3D sampling considers the hemisphere around $\mathbf{n}_p$ having radius *samplingRadius* and takes points in the screen-space area delimited by this hemisphere.

Perspective projection introduces same complications when returning to camera space. Pixels selected with both approaches may result in useless samples, related to points outside the area considered in the AO calculation. Randomization is a crucial aspect of every sampling technique adopted. If we adhere to a static, regular pattern, some banding artefacts will appear in the calculated AO.

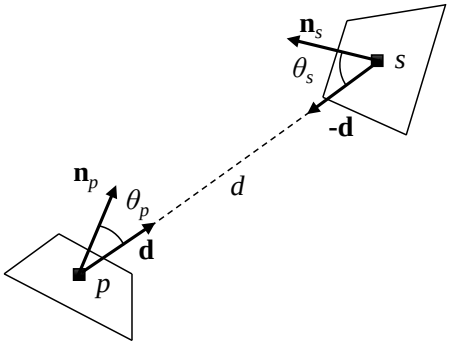


Fig. 3: The angles and vectors related to the solid-angle approximation between two pixels $p$ and $s$.

By applying some form of randomization, we avoid such artefacts, at the price of some high-frequency noise that can be handled with filtering, as will be shown in Section 4.3. A simple way to introduce randomization is to use some form of rotation dependent on a random value derived from the pixel coordinates. In flat sampling, a kernel of points randomly placed around the centre $p$ is rotated around $p$. While in 3D sampling, a kernel of vectors reaching random points in the hemisphere is rotated by using the normal of point $p$ as the rotation axis. In our experiment, we tested both approaches (including some variants) to evaluate the best option.

**Solid-angle estimator.** In [3], scene geometry is approximated with oriented discs considered as occluders to calculate per-vertex occlusion on the GPU. Our approach takes inspiration from this technique but brings it to the image-space domain.

Let us consider two pixels, $p$ and $s$, which belong to two different triangles. We want to estimate the solid angle at pixel $p$ that is covered by the surface to which the point $s$ belongs. Figure 3 visualizes the involved entities.

Let $\mathbf{d} = \mathbf{c}_s - \mathbf{c}_p$ be the normalized vector from the camera space position of $p$ to the camera space position of $s$, and let $d$ the distance between $\mathbf{c}_s$ and $\mathbf{c}_p$. We also define $\theta_p$ as the angle formed by $\mathbf{d}$ and $\mathbf{n}_p$ and $\theta_s$ as the angle formed by $-\mathbf{d}$ and $\mathbf{n}_s$. A possible solid-angle approximation is

$$s_p = \frac{a_s \max(0, \mathbf{d} \cdot \mathbf{n}_p)}{d^2} \tag{7}$$

The key idea is that $\max(0, \mathbf{d} \cdot \mathbf{n}_p)$ decreases the impact of occluders that block only incident light at shallow angles (which is radiometrically correct ). Conversely, multiplying by the area related to the occlusor surface modulates the contribution according to the dimensions of the surface.

If we want to also consider the orientation of the occlusor, we must introduce $\theta_s$ into the equation. Ideally, if we consider a disc of area $a_s$ and oriented according to $\mathbf{n}_s$, its projected solid angle would be

$$\omega_s = \frac{a_s \cos\theta_s}{d^2} \tag{8}$$

This is based on the differential area being related to the differential solid angle (as viewed from point $p$) by

$$d\omega = \frac{dA \cos\theta}{r^2} \tag{9}$$

where $\theta$ is the angle between the surface normal of $dA$ and the vector to $p$, and $r$ is the distance from $p$ to $dA$, as shown in Figure 4.

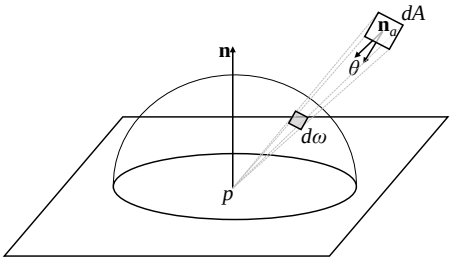


Fig. 4: Differential solid angle and differential area (image taken from [13]).

This equation can be understood intuitively as follows. If $dA$ is at distance 1 from $p$ and it is aligned so that it is exactly perpendicular to $d\omega$, then $d\omega = dA$ and $\cos\theta = 1$, and the equation holds. As $dA$ moves farther away from $p$, the $r^2$ term increases, and so dividing by it reduces $d\omega$ accordingly. Conversely, as $dA$ rotates so that it is not aligned with the direction of $d\omega$, the $\cos\theta$ term decreases, reducing $d\omega$ accordingly.

Unfortunately, applying this equation with screen-space estimators results in bad artefacts, because often the portion of an object visible to the camera is not the same one that is oriented towards the surface for which we are calculating the occlusion, as shown in Figure 5.

A possible approximate solution can be obtained by flipping the normal in such cases: after all, regarding solid-angle coverage, we are interested in whether or not there is some geometry occluding light, not whether or not it is oriented towards the occluded surface.

In terms of calculations, instead of holding the cosine value to 0, we can take its absolute value: so, if the $\theta_s$ angle falls in the [90...180] degrees range, the negative cosine value relative to $\mathbf{n}_s$ becomes a positive value for $-\mathbf{n}_s$.

$$\omega'_s = a_s \cdot \mathrm{abs}(\mathbf{d} \cdot \mathbf{n}_s) \tag{10}$$

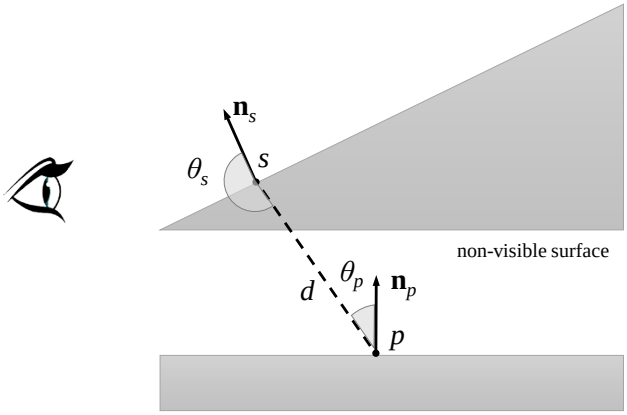


Fig. 5: No occlusion would be calculated if applying the basic solid-angle formula calculation.

Modifying our approximation according to this factor, we obtain:

$$s_{p,s} = \frac{a_s \cdot \text{abs}(\mathbf{d} \cdot \mathbf{n}_s) \cdot \max(0, \mathbf{d} \cdot \mathbf{n}_p)}{d^2} \tag{11}$$

Our approximation of ambient obscurance allows the solid-angle approximation formula to be changed to evaluate different approaches easily. As for instance, in [12] the authors derive an analytical expression for the solid angle subtended by a plane triangle at some arbitrary point in the space. This expression should be used to obtain ambient obscurance in our approach in place of 11. However, this expression is more expensive because its computation involves 32 multiplications, 20 additions, 3 square roots, and 1 ATAN2.

A falloff function relative to the occluding geometry distance allows the obscurance contribution to be smoothed with distance. We adopt the falloff function proposed by the Alchemy SSAO algorithm in [10].

**SaSSAO algorithm.** Let $triangleDivs$ be the triangle subdivision factor, which defines how the hemisphere is discretized. From this factor (as explained in Section 3), we derive $hemisphereBuckets$, which is $4 \cdot triangleDivs^2$ and the maximum value of the solid angle of each bucket as:

$$bucketSolidAngle = \frac{2\pi}{hemisphereBuckets} \tag{12}$$

Now, given the discretized hemisphere over the the current pixel $p$, a fragment shader computes the ambient obscurance estimating its visibility as we describe below (see Algorithm 2).

The depth $d_p$ is read and used to check whether the pixel $p$ is part of the geometry or the background. If it is part of the background, no processing takes place . Otherwise, $\mathbf{n}_p$ is retrieved and used, together with $d_p$, to calculate the screen-space position of the samples to be taken.

**Algorithm 2** SaSSAO.

```
procedure AmbientObscurance(p)
    if foreground(d_p) then
        for i ← 0, k − 1 do
            S_i ← sample_pixel(p)
            if foreground(d_{S_i}) then
                sampleDir ← c_{S_i} − c_p
                if length(sampleDir) ≤ maxDist then
                    t_id ← getTriangle(S_i)
                    b ← bucket(t_id)
                    if b ≤ bucketSolidAngle then
                        s_{p,s} ← getSolidAngle(p, S_i)
                        b ← min(bucketSolidAngle, b + s_{p,s})
        for i ← 1, hemisphereBuckets do
            em ← em + b_i
        AO = em/2π
```

Let $k$ be the number of samples to take, and $S_i, i \in [0...k-1]$, be the $i$-th sample pixel position in screen space, located depending on the adopted sampling pattern, as discussed in Section 4.2. Then, for each $S_i$, the depth $d_{S_i}$ is read. If the pixel is part of the background, the processing skips to the next sample. Otherwise, the $c_{S_i}$ is calculated in order to obtain the vector *sampleDir* that goes from $c_p$ to $c_{S_i}$ and its length *sampleDist*. Because of perspective projection, this length can be significantly greater than the screen-space distance between $p$ and $S_i$. If *sampleDist* is greater than the *maxDist* parameter, the sampled point is too distant to be taken into account for the AO calculation, and so processing skips to the next sample.

Otherwise, by the process described in Section 3 and adapted to work for the four octants of the hemisphere, the triangle id in which *sampleDir* falls is found. So, let $t_i$ be the triangle id found for $S_i$. We check whether the bucket $t_i$ is already fully covered (meaning that we know that the direction to which the sample belongs is already occluded). If it is, we skip to the next sample. Otherwise, we compute an estimate of the covered solid angle as described in the approximation formula 11, and we add it to the current coverage value for the bucket. The last check is to determine whether in the current bucket an over-occlusion from the same directions occurs. If this is the case, the value of the current bucket is set to the maximum value *bucketSolidAngle*.

After processing all the samples, we have an estimate of the visibility around the current processing point in the form of the coverage values for all the buckets of our discretization. We know that the solid angle for the full hemisphere is $2\pi$, so we sum the coverage values of all the buckets and divide this sum by $2\pi$ to obtain a global occlusion value. Of course, random sampling is no guarantee that samples will be obtained on every near-field occluder, but this is true for every SSAO technique. The immediate advantage of our technique is that we avoid

over-occlusion caused by multiple occluders covering each other but covering from the same direction.

### 4.3 AO filtering

Random sampling avoids banding issues, but introduces high-frequency noise. This could be removed with a basic Gaussian blur but the main problem with using this filter with AO is that it would also cause some shadow bleeding between surfaces at different depths or orientations. Because we have normals and depths at our disposal, so a more intelligent filtering can be done.

We decided to use a filtering function defined in [6], that is, a filter with bilateral weights based on normal and depth differences, and not Gaussian weights:

$$I^{\text{filtered}}(x) = \frac{\sum_{x_i \in \Omega} color(x_i) w(x, x_i)}{\sum_{x_i \in \Omega} w(x, x_i)} \tag{13}$$

where

$$w(x, x_i) = w_{normal}(x, x_i) w_{depth}(x, x_i) \tag{14}$$

$$w_{normal}(x, x_i) = \left( \frac{n_x \cdot n_{x_i} + 1}{2} \right)^{k_n} \tag{15}$$

$$w_{depth}(x, x_i) = \left( \frac{1}{1 + |d_x - d_{x_i}|} \right)^{k_d} \tag{16}$$

Here, $k_n$ and $k_d$ are two constants that can be tuned to alter the contribution of the normal/depth discriminators in the weight calculation.

## 5 Results

In this section, we present our methodology for evaluating the validity of our rendering technique. In terms of quality, we use Blender to off-line render ray-traced ambient occlusion as the reference image. The Blender renderings were conducted setting the number of samples to 64, a high number that gives excellent quality images. The resolution of all the images is $1280 \times 720$ pixels. To define the similarity between two images, we adopted the Structural Similarity Index (SSIM) [19] as a metric to measure similarity between images in a way that is consistent with human eye perception. In terms of efficiency, we implemented the Alchemy algorithm to allow us to evaluate SaSSAO against an already established technique. We implemented Alchemy while performing the minimal changes needed to our already active pipeline, so that we could share many parameters between the two techniques and make meaningful comparisons. However, evaluating the validity and efficiency of the technique against other techniques is complicated. Many parameters are involved, and even if the

source code of some other technique is available, comparisons are not straightforward: different algorithms in many cases do not use the same parameters, and even minor adjustments may dramatically change the quality of results and performance. Moreover, there may be some scene dependency, causing one technique to perform better than others in only some scenes, and then the results are merely indicative.

### 5.1 Test results

Our testing system was equipped with an Intel Core i7-3820 CPU 3.60 GHz, 16 GB of RAM, and a GeForce GTX TITAN GPU. The model used in our experiments was Sponza by Crytek [4]. For SaSSAO, we used the area of circumscribed circle as area approximation, and hemisphere sampling. These parameters showed the best results during our tests. We also used an angle bias parameter . This parameter often appears in SSAO techniques and is used to limit the self-occlusion and the artefacts caused when the geometry is almost co-planar with the geometry of the current point. If the cosine of the angle between the current point normal and the direction to the occluder was less than the angle bias, the sample was ignored. Tables 1 and 2 list the test scenes and associated parameters used. For Alchemy, we used flat sampling because it clearly shows good results. Other parameters were adjusted manually in an attempt to improve results or to show some particular behaviour. However , because of the difficulty of performing a comparison, we attempted to obtain for each selected scene the best result by tuning the parameters manually. Figures from 6 to 11 show some results from the Atrium Sponza 3D model [4].

Overall results are encouraging, with our algorithm often producing results of comparable quality to Alchemy (sometimes even slightly superior). Alchemy performs fewer calculations for each sample, so with the same number of samples it is generally faster. However, the interesting aspect is that with fewer samples and more calculations, sometimes approximately the same quality was obtained at a similar frame rate, which could be useful in bandwidth-limited situations. FPS count in our approach is generally lower because of the many access into the triangle buckets associated with each pixel, and also to perform the comparison to check whether a region is fully covered by triangles . Nevertheless, we are confident that some optimizations can be implemented to increase performance.

Ultimately, it appears that properly adjusting the parameters is a large factor in the results obtained. Moreover, a higher-SSIM image may not always match a human observer's choice of the best result, so we are not completely confident that this index is a particularly effective metric for ambient obscurance comparisons. For example, sometimes, smoother images taken with more samples receive a lower SSIM.

## 6 Conclusions and Future Work

We have developed a new technique for ambient obscurance exploiting screen space. The approach common in the literature is based on deferred rendering

Table 1: Test scenes used with maximum distance, angle bias, and radius sampling chosen as shared parameters.

| Scenes | Max distance | Angle bias | Radius |
|---|---|---|---|
| Lion head close up | 0.5 | 0.3 | 0.4 |
| Lion head close up | 1.0 | 0.3 | 1 |
| Lion head and drapes | 0.25 | 0.6 | 0.2 |
| Lion head and drapes | 1.5 | 0.4 | 1.5 |
| Atrium (from top) | 0.8 | 0.3 | 0.8 |
| Atrium | 0.8 | 0.3 | 0.5 |

Table 2: Test scenes used with values associated with the SaSSAO and Alchemy approaches. Note that we tuned these values independently to obtain the best results for each one in terms of Structural Similarity Index (SSIM) and FPS.

| | SaSSAO | | | | Alchemy | | |
|---|---|---|---|---|---|---|---|
| Scenes | SSIM | FPS | Samples | tDivs | SSIM | FPS | Samples |
| Lion head close up | 92.80% | 35.90 | 16 | 4 | 92.71% | 22.26 | 64 |
| Lion head close up | 91.87% | 22.86 | 32 | 4 | 92.95% | 22.96 | 64 |
| Lion head and drapes | 92.64% | 14.32 | 64 | 3 | 91.07% | 55.41 | 16 |
| Lion head and drapes | 90.46% | 11.53 | 64 | 4 | 84.81% | 83.19 | 16 |
| Atrium (from top) | 85.51% | 35.59 | 64 | 4 | 81.16% | 84.10 | 16 |
| Atrium | 87.77% | 13.28 | 64 | 4 | 86.91% | 84.52 | 16 |

and G-buffer sampling and is shared by a number of algorithms in the field. Our novel contributions are the use of a geometry shader to approximate the area of occluders and the adoption of a hemisphere discretization technique to classify the occluders according to their positions. To estimate coverage, we used a solid-angle approximation derived from our experiments with other algorithms and from some observations related to the lack of data inherent to image-based algorithms.

This type of approach, in which we evaluated the level of occlusion considering the direction from which coverage originated, storing the result in our "triangle buckets", allowed us to avoid over-occlusion and generally produced better quality results because of the implicit weighting of sample contributions. Quality is our primary concern, and we evaluated our results by comparing the structural similarity with off-line rendered images calculated through ray-tracing in Blender.

Ambient obscurance and, generically, global illumination approximations (for real-time rendering) are constantly improving, following the evolution of hardware, APIs, and the literature. Here, we share some ideas that we wish to explore in the near future. Our pyramid of "triangle buckets" does not only tell us how much a point is covered, but also from where (with a customizable level of precision). This may be exploited to achieve other types of results such as "directional occlusion" [15] and may be used to calculate direct lighting by using "bent normals", which are normals adjusted to consider the direction from

which more incoming light will potentially reach the surface (i.e., where there are no occluders).

The compute shaders, introduced in OpenGL 4.3, add another level of freedom in utilization of the GPU (in the direction of other general-purpose APIs that exploit GPUs and other parallel hardware, such as OpenCL and CUDA). Compute shaders operate differently from other shader stages: for example, they have no well-defined set of input values and no frequency of execution specified by the nature of the stage (once per vertex, once per fragment...). More efficient sampling may be key : our technique performs a number of texture fetch operations (particularly relevant to the efficiency of the technique) for each fragment, and a possible way of optimizing fetch operations is to use the GPU shared memory. Recent results in the SSAO field, based on CUDA implementations, recently appeared in [18] and [17] with interesting implications for SaSSAO.

Fig. 6: Lion head close up - max distance 0.5, angle bias 0.3. Up: SaSSAO, SSIM 92.80%, 35.90 fps - Down left: Blender - Down right: Alchemy, SSIM 92.71%, 22.26 fps.

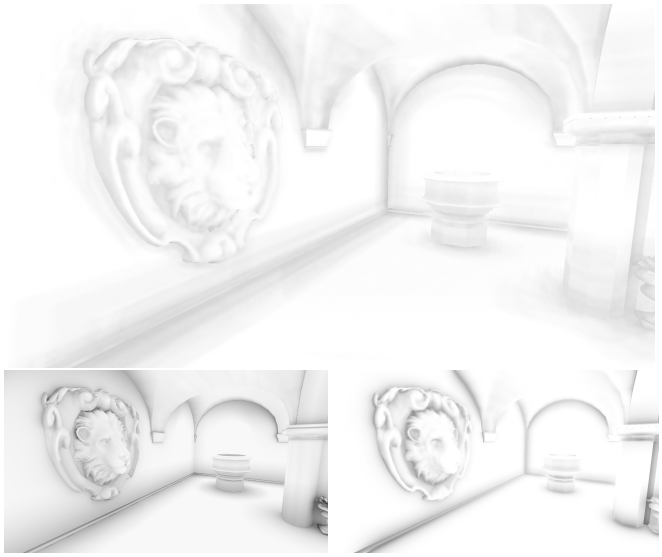

Fig. 7: Lion head close up - max distance 1.0, angle bias 0.3. Up: SaSSAO, SSIM 91.87%, 22.86 fps - Down left: Blender - Down right: Alchemy, SSIM 92.95%, 22.26 fps.

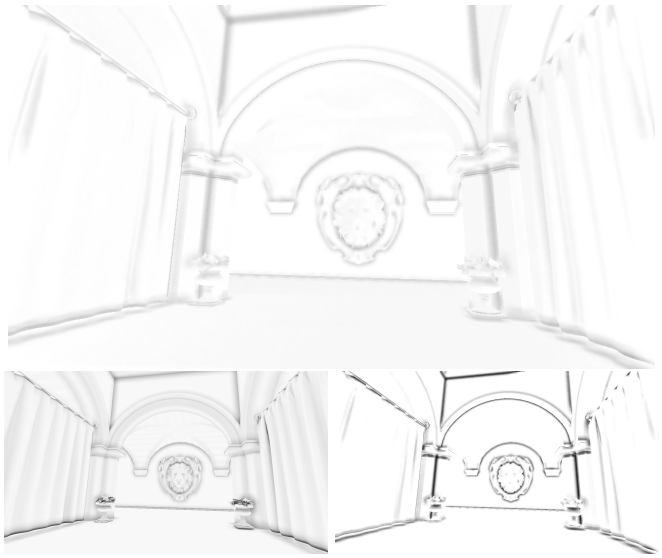

Fig. 8: Lion head and drapes - max distance 0.25, angle bias 0.6. Up: SaSSAO, SSIM 92.64%, 14.32 fps - Down left: Blender - Down right: Alchemy, SSIM 91.07%, 55.41 fps.

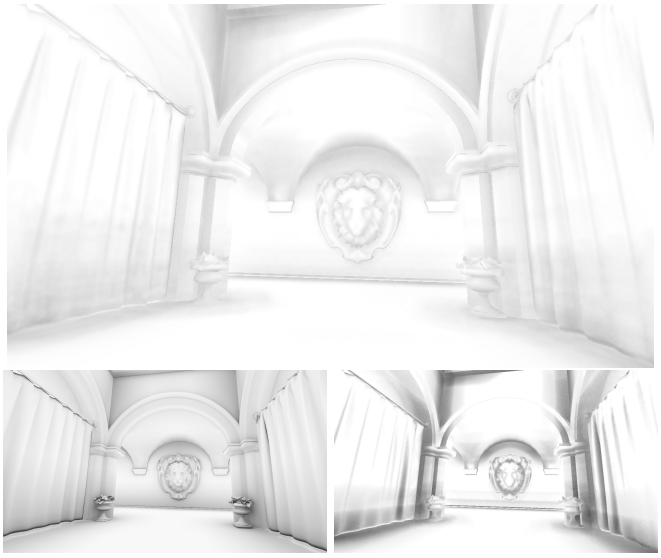

Fig. 9: Lion head and drapes - max distance 1.5, angle bias 0.4. Up: SaSSAO, SSIM 90.46%, 11.53 fps - Down left: Blender - Down right: Alchemy, SSIM 84.81%, 83.19 fps.

Fig. 10: Atrium (from top) - max distance 0.8, angle bias 0.3. Up: SaSSAO, SSIM 85.51%, 35.59 fps - Down left: Blender - Down right: Alchemy, SSIM 81.16%, 84.10 fps.

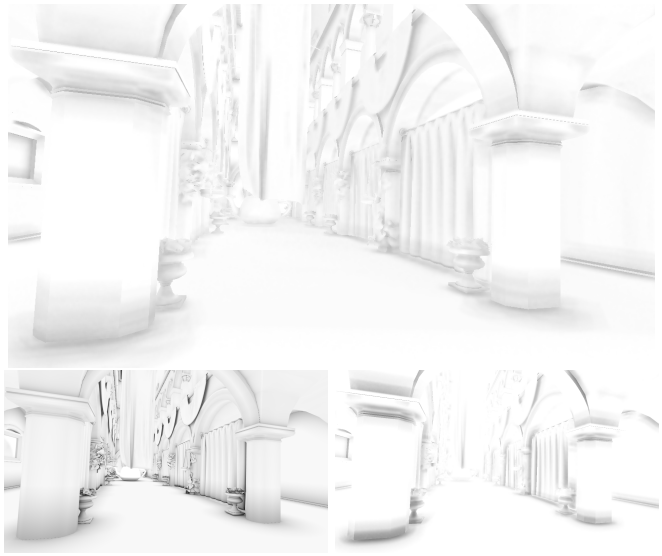

Fig. 11: Atrium - max distance 0.8, angle bias 0.3. Up: SaSSAO, SSIM 87.77%, 13.28 fps - Down left: Blender - Down right: Alchemy, SSIM 86.91%, 84.52 fps.

## References


1. Aalund, F.P.: A comparative study of screen-space ambient occlusion methods. Tech. rep., Technical University of Denmark (2013)
2. Bavoil, L., Sainz, M., Dimitrov, R.: Image-space horizon-based ambient occlusion. In: ACM SIGGRAPH 2008 talks. p. 22. ACM (2008)
3. Bunnell, M.: Dynamic ambient occlusion and indirect lighting. Gpu gems 2(2), 223–233 (2005)
4. Crytek: Atrium Sponza Palace, `www.crytek.com/cryengine/cryengine3/downloads`
5. Filion, D., McNaughton, R.: Effects & techniques. In: ACM SIGGRAPH 2008 Games. pp. 133–164. ACM (2008)
6. Gravås, L.O.: Image-space ambient obscurance in webgl. Tech. rep., Institutt for datateknikk og informasjonsvitenskap (2013)
7. Khanna, P., Slater, M., Mortensen, J., Yu, I.: A non-parametric guide for radiance sampling in global illumination. Computer Graphics, Imaging and Visualisation, 2007. CGIV'07 pp. 41–48 (2007)
8. Langer, M.S., Bülthoff, H.H.: Depth discrimination from shading under diffuse lighting. Perception 29(6), 649–660 (2000)
9. McGuire, M., Mara, M., Luebke, D.: Scalable ambient obscurance. In: Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics. pp. 97–103. Eurographics Association (2012)
10. McGuire, M., Osman, B., Bukowski, M., Hennessy, P.: The alchemy screen-space ambient obscurance algorithm. In: Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. pp. 25–32. ACM (2011)
11. Mittring, M.: Finding next gen: Cryengine 2. In: ACM SIGGRAPH 2007 courses. pp. 97–121. ACM (2007)

12. Oosterom, A.V., Strackee, J.: The solid angle of a plane triangle. IEEE Transactions on Biomedical Engineering BME-30(2), 125–126 (Feb 1983)
13. Pharr, M., Humphreys, G.: Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2010)
14. Ritschel, T., Dachsbacher, C., Grosch, T., Kautz, J.: The state of the art in interactive global illumination. Comput. Graph. Forum 31(1), 160–188 (Feb 2012)
15. Ritschel, T., Grosch, T., Seidel, H.P.: Approximating dynamic global illumination in image space. In: Proceedings of the 2009 symposium on Interactive 3D graphics and games. pp. 75–82. ACM (2009)
16. Shanmugam, P., Arikan, O.: Hardware accelerated ambient occlusion techniques on gpus. In: Proceedings of the 2007 symposium on Interactive 3D graphics and games. pp. 73–80. ACM (2007)
17. Timonen, V.: Line-Sweep Ambient Obscurance. Computer Graphics Forum (Proceedings of EGSR 2013) 32(4), 97–105 (2013)
18. Timonen, V.: Screen-space far-field ambient obscurance. In: Proceedings of the 5th High-Performance Graphics Conference. pp. 33–43. HPG '13, ACM, New York, NY, USA (2013)
19. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: From error visibility to structural similarity. Image Processing, IEEE Transactions on 13(4), 600–612 (2004)
20. Zhukov, S., Iones, A., Kronin, G.: An ambient light illumination model. In: Rendering Techniques' 98, pp. 45–55. Springer (1998)