Horizon-Split Ambient Occlusion

Rouslan Dimitrov Louis Bavoil Miguel Sainz
NVIDIA Corporation

Figure 1: *Ambient occlusion contribution with no additional shading on the David head model. (a) Our technique at 75 fps, (b) 30 fps with different parameters, (c) ray tracing with Mental Ray. The images are rendered at 800x600.*

Abstract

Ambient occlusion is a technique that computes the amount of light reaching a point on a diffuse surface based on its directly visible occluders. It gives perceptual clues of depth, curvature, and spatial proximity and thus is important for realistic rendering. Traditionally, ambient occlusion is calculated by integrating the visibility function over the normal-oriented hemisphere around any given surface point. In this paper we show this hemisphere can be partitioned into two regions by a horizon line defined by the surface in a local neighborhood of such point. We introduce an image-space algorithm for finding an approximation of this horizon and, furthermore, we provide an analytical closed form solution for the occlusion below the horizon, while the rest of the occlusion is computed by sampling based on a distribution to improve the convergence. The proposed ambient occlusion algorithm operates on the depth buffer of the scene being rendered and the associated per-pixel normal buffer. It can be implemented on graphics hardware in a pixel shader, independently of the scene geometry. We introduce heuristics to reduce artifacts due to the incompleteness of the input data and we include parameters to make the algorithm easy to customize for quality or performance purposes. We show that our technique can render high-quality ambient occlusion at interactive frame rates on current GPUs.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—;

Keywords: ambient occlusion, image space, ray tracing, graphics hardware

1 Introduction

Ambient occlusion can be expressed in terms of the rendering equation [Kajiya 1986] in the particular case where the incoming light intensity is uniform, the surface does not emit light and has a Lambertian BRDF. The light intensity from ambient occlusion at a surface point **P** with surface normal $\vec{n}$ is then:

$$A = \frac{1}{\pi}\int_{\Omega}(1 - V(\vec{\omega}))(\vec{n}\cdot\vec{\omega})d\omega \qquad (1)$$

V is the light visibility function over the normal-oriented unit hemisphere Ω, which returns 1 if a ray starting from **P** in direction $\vec{\omega}$ intersects an occluder and 0 otherwise. Unlike with global illumination, the ambient occlusion integral is not recursive, which makes it more affordable to compute.

A simple Monte-Carlo solution to Equation (1) is to compute this 2D integral by tracing rays in directions $\vec{\omega}$ through the hemisphere with a probability density function $PDF = \vec{n}\cdot\vec{\omega}$. Building on the concept of horizon occlusion [Rogers 1985], our approach splits the hemisphere into two parts by a horizon line of height $H(\theta)$. Rays that would normally be traced below the horizon are known to intersect an occluder so the intersection test for these rays can be skipped. More importantly, since we have the horizon line, we can compute the occlusion under the horizon much more efficiently, by sampling in 1D instead of 2D.

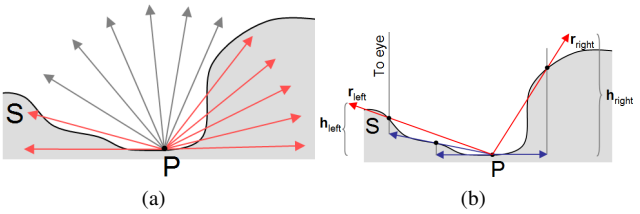


Figure 2: *(a) Red rays intersect the surface S in a local neighborhood of P. (b) We compute the height of the horizon by deflecting the horizon rays step by step towards the eye.*

We call horizon rays, the rays starting at the surface point **P** and tangent to the horizon. The horizon height $H(\theta)$ is the height at the tip of the ray in the normal direction, normalized between 0 and 1. To find the horizon rays, we propose a search along a curve which walks on the heightfield in direction θ. Figure 2 shows an overview of the algorithm in 2D. Horizon rays initially in the tangent plane of surface at **P** are deflected step by step towards the eye. The algorithm stops iterating when the length of the ray reaches a maximum range R.

We apply our method to the depth buffer of the rendered scene, which provides a descretized approximation of the visible geometry on the screen. As [Shanmugam and Arikan 2007] and [Mittring 2007] have shown, although this depth buffer information may be missing potential occluders, visually pleasant ambient occlusion can be computed from it.

This work was presented as a poster at the I3D'08 conference.

Equation 1 can be rewritten as:

$$A = \frac{1}{\pi} \int_{\theta=0}^{2\pi} \int_{z=H(\theta)}^{1} (1 - V(\vec{\omega}))(\vec{n} \cdot \vec{\omega}) d\omega \quad (2)$$

The ambient occlusion illumination A can be expressed in terms of the occlusion as:

$$A = 1 - \frac{1}{\pi} \left(\int_{\theta=0}^{2\pi} T(\theta) d\theta + \int_{\theta=0}^{2\pi} \int_{z=H(\theta)}^{1} V_{\vec{\omega}} (\vec{n} \cdot \vec{\omega}) d\omega \right) \quad (3)$$

where $T(\theta)$ is the horizon occlusion in direction θ. In fact, we derive that $T(\theta) = H(\theta)^2$, and thus for a given θ, the horizon occlusion term does not require any sampling of the hemisphere. This enables focusing the randomized sampling to a subset of the hemisphere, while still capturing the occlusion contribution of the non-sampled part. Figure 1 (a) shows an example where the horizon occlusion term only is a good approximation of the overall occlusion.

Our algorithm computes the terms in Equation 3 by computing the horizon height $H(\theta)$ and the visibility function $V(\vec{\omega})$ using a per-pixel normal and performing a linear search in the depth buffer. Our contributions include:

- We use the concept of horizon occlusion for optimizing the sampled area of the hemisphere Ω.
- We introduce a new algorithm for finding the horizon height $H(\theta)$ in a given direction, based on a depth buffer.
- We provide an analytical solution to compute the occlusion contribution of the horizon occlusion $T(\theta)$, which provides a lower bound to the ambient occlusion term.
- Our method works in image space and does not require any precomputation based on the scene geometry.

2 Related Work

Ambient Occlusion. The concept of ambient occlusion was pioneered by accessibility maps [Miller 1994] and obscurances [Zhukov et al. 1998a]. Obscurances replace the visibility function V by an attenuation term. Ambient occlusion as defined in Equation 1 appeared in [Langer and Bulthoff 1999]. It has now become a standard rendering technique as an alternative to global illumination [Landis 2002; Christensen 2003; Christensen et al. 2006]. In offline renderers, the visibility term V is typically computed by sampling the hemisphere using object-space ray tracer.

This ambient occlusion term from Equation 1 is an indirect lighting term, which needs to be weighted and added with direct lighting to produce a final color. It can be seen as an improvement to the flat ambient term from the OpenGL lighting equation. On the other hand, other techniques such as [Hegeman et al. 2006] and [Pharr and Green 2004] compute the following ambient shadow term:

$$A' = \frac{1}{2\pi} \int_{\Omega} (1 - V(\vec{\omega})) d\omega \quad (4)$$

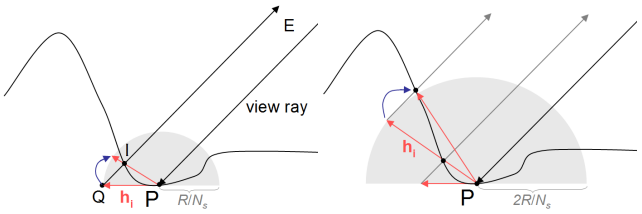
This term is the shadow from an environment light. Like for regular shadowing, it needs to be multiplied with the direct lighting term. In this paper, we compute A as defined in Equation 1.

Ambient occlusion terms can be computed using image-based algorithms. For instance, [Pharr and Green 2004] implement ambient occlusion on graphics hardware by averaging hard shadows from light sources distributed around the scene, using shadow mapping. This approach typically requires hundreds of point light sources to converge to a smooth solution. Obscurances [Zhukov et al. 1998a] with color bleeding can be computed based on casting bundles of rays in random directions, which can be implemented on graphics hardware using depth peeling from multiple viewpoints [Méndez et al. 2003]. Hardware-accelerated occlusion queries can also be used to compute ambient occlusion per face or per vertex. [Sattler et al. 2004; Franklin 2005].

In real-time graphics, for static scenes, ambient occlusion terms can be precomputed at the vertices or in light maps. Bent normals can be computed by averaging the direction of the unoccluded rays in the hemisphere [Landis 2002]. Ambient occlusion fields [Kontkanen and Laine 2005] render inter-object ambient occlusion, without self-occlusion. [Malmer et al. 2006] take a volumetric approach and store ambient occlusion values in a 3D grid and using linear interpolation. [Cadet and Lécussan 2007] store the ambient occlusion based on an octree. [Bunnell 2005], [Wassenius 2005] and [Hoberock and Jia 2007] approximate the geometry using form factors between disks to render ambient occlusion in real time. This disk-based approach may not be practical for complex scenes with arbitrary dynamic geometry. All of the techniques mentioned above require scene-dependent precomputation. [Hegeman et al. 2006] take a different approach and construct ad-hoc analytical formula for plausible ambient occlusion of trees.

Recently, [Mittring 2007] and [Shanmugam and Arikan 2007] have shown a new way of rendering ambient occlusion in screen space as a postprocessing pass based on a depth buffer from the eye's point of view. This is similar to soft shadow mapping based on backprojection, where the depth buffer is rendered from a light's point of view [Guennebaud et al. 2006; Schwarz and Stamminger 2007]. Crytek Gmb has presented the idea of using the information in a depth buffer [Mittring 2007] to compute ambient occlusion solely based on a depth buffer with no normal information. The technique recovers eye-space positions from the depth values in the depth buffer and samples a sphere around the point. Then it compares the depth of the samples with the depth of the point and computes an approximate ambient occlusion term based on these comparisons. [Shanmugam and Arikan 2007] accumulate solid angles occluded by neighboring pixels in screen space based on the pixel eye-space positions and normals stored in an ND-buffer. The efficiency of their image-space method relies on the fact that neighbor pixels in screen space are likely to come from objects that are close to each other in world space. To capture the occlusion from distant occluders, the occlusion from precomputed spheres is added to the image-space occlusion. To reduce under-occlusion artifacts, they use two ND-buffers: one for the front faces and one for the back faces. In this paper, we use a single ND-buffer. Their image-based method has over-occlusion issues because all the samples are assumed to be visible from the surface point.

Ray Marching in Depth Buffers. Although our algorithm internally works in eye space, it is similar to the tangent-space linear search, also known as line rasterization [Baboud and Décoret 2006; Amanatides and Woo 1987]. This linear search has been used in parallax occlusion mapping techniques [Brawley and Tatarchuk 2004; Oliveira and Policarpo 2005; Policarpo et al. 2005; Tatarchuk 2006] and we refer to it as ray marching. The idea of searching along a ray in a depth buffer has also been used image-space refraction [Davis and Wyman 2007; Oliveira and Brauwers 2007]. As mentioned in [Davis and Wyman 2007], tracing a ray in screen space can be implemented as an image-space operation by marching along the projected ray in image space. Our method can be seen as an optimization of ray marching based on an horizon line defined on the hemisphere.



(a) (b)

Figure 3: *Horizon Tracing. Two steps of the deflection process of an horizon ray* $\mathbf{h_i}$.

Horizon Culling. The idea of culling occluders below a horizon line has been used for occlusion culling on hierarchical terrains [Downs et al. 2001; Lloyd and Egbert 2002; Stewart 1997]. In this context, the horizon is the boundary between the sky and the terrain as seen from the point of view of the eye. Regions of the terrain that are completely below the horizon can be culled. As described by [Downs et al. 2001], floating horizons have also been used for plotting functions of the form $z = f(x,y)$ by processing the function from back to front and maintaining an upper and lower horizon for the function graph [Rogers 1985].

3 Our Algorithm

Our algorithm takes as input an ND-buffer storing linear depths and normals for the pixels on the screen. For each pixel, we compute an eye-space position of its corresponding surface point **P** and we integrate the ambient occlusion terms from Equation 3. We pick a set of random directions θ distributed around the normal $\vec{n}$ at point **P**. For each angle θ, the height of the horizon $H(\theta)$ is computed by marching in the depth buffer along a line projected onto the tangent plane in direction θ. Then, the horizon occlusion $T(\theta)$ is computed and additional normal rays are optionally traced in the part of the hemisphere above $H(\theta)$ to complete the sampling of the hemisphere. The algorithm is applied to every surface point **P** visible on the screen.

3.1 Horizon Tracing

For a given angle θ, the problem is to find the height $H(\theta)$ of the horizon seen from **P** within a certain radius of influence R around **P**. We start with a ray $\mathbf{h} = (\mathbf{P}, \vec{t})$ where $\vec{t}$ is a surface tangent at **P** in the direction θ, and we incrementally extend the length of this ray N_s times. At each step, the end point **Q** of the ray **h** is projected in screen space and its depth is compared with the depth of the corresponding point in the depth buffer. If the later is closer (Q is below the surface seen from the eye), and additional conditions are satisfied, then **h** is deflected towards the intersection point **I** and its direction is normalized and rescaled (see Figure 3). More precisely, the deflection process works as described in Figure 4. The horizon rays **h** are deflected only if **Q** is within the distance R from **P** and if the candidate horizon ray has a proper direction (ie. not inverted tangential component). This procedure is applied N_d times per pixel for randomly distributed angles θ distributed around the normal and the resulting heights define a piece-wise linear approximations of the horizon line $H(\theta)$ on the hemisphere Ω.

3.2 Horizon Occlusion

The horizon occlusion term from Equation 3 is:

$$O_T = \frac{1}{\pi}\int_{\theta=0}^{2\pi} T(\theta)d\theta$$

```
H(θ) :=
1  h ← T
2  for i ← 1..N_s
3      do Q ← P + i (R/N_s) h
4         if ISBELOW(Q, S)
5           then I ← INTERSECT(Q − E, S)
6                if LENGTH(I − P) < R and DOT(I − P, T) > 0
7                  then h ← NORMALIZE(PROJ_{n,t}(I − P))
8  return DOT(h, n)
```

Figure 4: *Pseudo code for computing* $H(\theta)$ *by deflecting a horizon ray. In each iteration, point* **Q** *is projected onto the screen and the surface S represented by the depth buffer is sampled. This sample is used to evalute the functions* ISBELOW *and* INTERSECT. *The function* $\text{PROJ}_{\vec{n},\vec{t}}$ *projects a vector on the plane defined by the normal* $\vec{n}$ *and the tangent* $\vec{t}$ *at* **P**.

$$= \frac{1}{\pi}\int_{\theta=0}^{2\pi}\int_{z=0}^{H(\theta)} (\vec{n}\cdot\vec{\omega})d\omega \qquad (5)$$

To perform the integration, we use cylindrical coordinates (r,θ,z), where $d\vec{w} = dz d\omega \hat{r}$ with $r = 1$. We partition Ω below H with adjacent vertical hemispherical lunes of height $H(\theta_i)$ subtending an angle $\alpha = \theta_{i+1} - \theta_i = 2\pi/N_d$ around the normal. We then approximate the occlusion O_T by:

$$\begin{aligned} O_T &= \frac{1}{\pi}\sum_{i=1}^{N_d}\int_0^{H(\theta_i)}\int_{\theta_i}^{\theta_{i+1}} (\vec{n}\cdot\vec{\omega})d\theta dz \\ &= \frac{\alpha}{\pi}\sum_{i=1}^{N_d}\int_0^{H(\theta_i)} (\vec{n}\cdot\vec{\omega})dz \\ &= \frac{2}{N_d}\sum_{i=1}^{N_d}\int_0^{H(\theta_i)} z dz \\ &= \frac{1}{N_d}\sum_{i=1}^{N_d} H^2(\theta_i)dz \end{aligned} \qquad (6)$$

Horizon occlusion typically produces good results for continuous surfaces with a single layer of depth complexity. In the next section, we describe a complementary algorithm to account for more complex occlusions such as inter-object occlusion.

3.3 Normal Occlusion

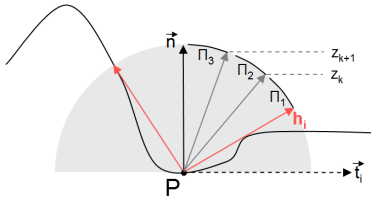


Figure 5: *For each horizon ray* h_i, $i = 1..N_d$, *we trace* N_n *normal rays (2 are shown in the figure), starting from P.*

We partition Ω above H with adjacent vertical stripes of quadrilateral patches delimited by the heights z_k (Figure 5).

The normal occlusion for a given direction $\vec{t_i}$ is:

$$O_{N,i}(z_k) = \frac{1}{\pi}\int_{\theta=\theta_i}^{\theta_{i+1}}\int_{z_k}^{z_{k+1}} V(\vec{\omega})(\vec{n}\cdot\vec{\omega})d\omega$$

$$
\begin{aligned}
&= \frac{\alpha}{\pi}\int_{z_k}^{z_{k+1}} V(\vec{\omega})(\vec{n}\cdot\vec{\omega})d\omega \\
&= \frac{2}{N_d}\int_{z_k}^{z_{k+1}} V(\vec{\omega})z d\omega \\
&= \frac{1}{N_d}(z_{k+1}^2 - z_k^2)V(\vec{\omega}) \qquad (7)
\end{aligned}
$$

Equation 7 is a valid approximation for any type of patch distribution. By picking a specific distribution, Equation 7 can be simplified so that each patch contributes the same amount of occlusion, which is more efficient. From a Monte-Carlo point of view, such a distribution is doing importance sampling using a cosine distribution in the upper part of the hemisphere. [Pharr and Humphreys 2004]. In this case,

$$z_{k+1}^2 - z_k^2 = c \qquad (8)$$

The occlusion contribution of all patches should add up to $1 - O_T$,

$$
\begin{aligned}
\sum_{k=1}^{N_n} O_{N,i}(z_k) &= \frac{1-O_T}{N_d} \\
\frac{N_n}{N_d}c &= \frac{1-O_T}{N_d} \\
c &= \frac{1-O_T}{N_n} \qquad (9)
\end{aligned}
$$

To compute the initial ray direction $\vec{r}_k$ towards patch Π_k, we need the polar angle ϕ_k, which relates to z_k as:

$$z_k = \cos(\phi_k)$$

Pluggin in Equation 8,

$$cos^2(\phi_{k+1}) - cos^2(\phi_k) = \frac{1-O_T}{N_n}$$

Since the terms on the left have a linear dependency,

$$cos^2(\phi_k) = \frac{1-O_T}{N_n}k$$

$$\phi_k = \arccos\left(\sqrt{\frac{1-O_T}{N_n}k}\right), \quad k = 0\ldots N_n - 1 \qquad (10)$$

And finally, the direction of r_k is:

$$\vec{r}_k = cos(\phi_k)\,\vec{n} + sin(\phi_k)\,\vec{t}_i$$

where t_i is the projection of the horizon ray h_i on the TB plane. After simplifying, our cosine distribution is:

$$\vec{r}_k = \sqrt{1 - k(1-O_T)/N_n}\,\vec{t}_i + \sqrt{k(1-O_T)/N_n}\,\vec{n}$$

Note that $\vec{r}_k$ is always above the horizon h_i. With this distribution, each patch contributes:

$$O_{N,i}(\vec{r}_k) = \frac{V(\vec{\omega})}{N_n N_d}$$

So the total normal occlusion is:

$$O_N = \sum_{i=1}^{N_d}\sum_{k=1}^{N_n}\frac{V(\vec{\omega})}{N_n N_d} \qquad (11)$$

For each patch Π_k, we determine $V(\vec{\omega})$ in the direction r_k by doing a linear search in the depth buffer in direction $\vec{\omega}$ with a fixed number of steps. After determining $V(\vec{\omega})$, we accumulate its occlusion contribution to the total occlusion O.

$R = 0.1$ $R = 0.2$ $R = 0.5$

Figure 6: *Effect of the radius of influence R on a scene.*

$N_d = 1$ $N_d = 4$ $N_d = 10$

Figure 7: *Effect of the number of initial directions N_d on a scene.*

3.4 Combining Horizon and Normal Occlusion

The final light intensity from ambient occlusion at P is:

$$A = 1 - (O_T + O_N) \qquad (12)$$

3.5 Linear Attenuation

At a distance R around objects, sharp occlusion boundaries may be visible because these occluders may strongly influence the ambient occlusion once they begin to be sampled. To soften these sharp edges, we weight the contribution of each ray by a linear attenuation function based on the normalized distance between P and the occluder.

3.6 Parameter Analysis

In this section, we give a description of the different parameters that control the performance and overall quality of the ambient occlusion estimation.

Radius of influence R defines the maximum distance of influence of any given occluder (Table 6). Increasing this radius has two side effects: occluder undersampling in the hemisphere (requiring more sampling rays to maintain image quality) and stronger ambient occlusion values due to the existence of more occluders (e.g. we can maximize the occlusion for the walls inside a closed room if the radius is large enough to include the opposing walls).

Number of tangent directions N_d is the number of rays around the normal used to estimate the horizon split line $H(\theta)$. The total number of traced rays is proportional to this parameter.

Number of normal rays N_n is the number of rays that will be used during ray marching. These rays are distributed on the subset of the hemisphere above the horizon line $H(\theta)$ determined during the tangent tracing pass.

Number of steps N_s is the number of steps that both the horizon tracing and the ray marching passes will evaluate on a per ray basis. This parameter affects perfomance directly and also the quality of the results for scenes with high geometric detail such as the Buddha dataset.

4 Implementation Details

We implemented our algorithm in screen-space as a fullscreen pass using a pixel shader that takes as input a linear depth buffer and a normal buffer. The depths are the z components of the eye-space positions and the normals are in eye space. This approach has the advantages of requiring no scene preprocessing.

4.1 Limitations in screen space

When working in screen space, we face certain limitations that directly affect the quality of the results. In particular, undersampling, data quantization and missing values can introduce artifacts that need to be addressed. We present in this section the problems we have encountered and how they can be solved.

4.1.1 Near-parallel view quantization

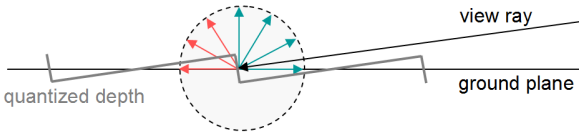


Figure 8: *Root of the banding artifacts for smooth surfaces in oblique angles.*

During the texture fetches of the depth values, it is possible to obtain erroneous values due to linear interpolation of depth discontinuities. In the general case we choose to use nearest pixel filtering to avoid this problem.

However point filtering can introduce banding artifacts. In particular, as shown in Figure 8, when evaluating a smooth surface quasi-perpendicular to the image plane (e.g. a ground plane on the scene), we can observe that many rays will erroneously intersect the surface (rays in red in the figure), introducing a view dependent variability of the ambient occlusion values.

To avoid these artifacts, we offset the initial pixel position P in the normal direction. If we consider the worst case when the view direction is parallel to the plane, the height of the staircase step in screen space is $u = 1/res$, with res being the screen resolution in pixels (Figure 9). Assuming the camera is at the origin and has a field of view ϕ.

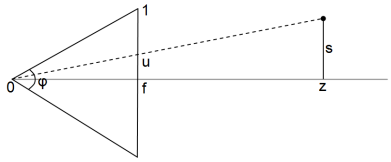


Figure 9: *Pinhole camera geometry.*

Considering the focal length as

$$f = \frac{1}{\tan(\frac{\phi}{2})}$$

By similar triangles, the size of the stair step s in eye space is:

$$s = \frac{uz}{f} = \frac{z}{f\ res}$$

Thus, we need to offset P to

$$P' = P + \frac{z}{f\ res}\vec{n} \tag{13}$$

We choose f and res from the image space axis where $f\ res$ is smaller.

4.1.2 Randomization

To hide banding artifacts from regular sampling, we use randomized rotations for the ray directions, and jittered step sizes for the linear searches. These randomized values are precalculated and stored in a tiled texture containing:

$$(\cos(\alpha), \sin(\alpha), \beta)$$

where $\alpha = 0..2\pi/N_d$, and $\beta \in (-1;1)$. From the first two components, we reconstruct a rotation matrix to apply to all initial horizon rays. The third component β is used to jitter the step size $s = R/N_s$ as follows:

$$\mathbf{P} + \vec{h} = \mathbf{P} + (i+\beta)\cdot s \cdot \frac{\vec{h}}{||\vec{h}||}$$

4.1.3 Depth Variance Based Blur

Depending on the amount of sampling directions and the use of randomization, the result of the ambient occlusion evaluation presents a certain degree of noise. This can be reduced by adding a postprocessing pass performing a Gaussian blur.

We choose to apply a 5x5 Gaussian blur, additionally weighting each sample by the inverse of its depth deviation from the kernel center, in order to preserve sharp object edges.

5 Experiments

In this section we present rendering results and analyse how the parameters of our method affect quality and performance. We also discuss the limitations of our method. We use four datasets: Buddha, Ring, Dragon, and David with about 800,000 triangles each. We rendered them using a Direct3D 10 renderer on a 2.4 GHz AMD Athlon64 X2 with an NVIDIA GeForce 8800 GTX graphics card and 2GB of RAM. The normal rendering pipeline consists of 3 main passes including the geometry pass that generates the depth and normal buffers, the ambient occlusion pass, and a depth-dependent blur on the ambient occlusion result.

For the David model in Figure 1(b), Table 1 breaks down the time of the individual passes. The horizon occlusion takes almost the same time as the normal occlusion with one ray. In this case, both terms have the same complexity in terms of number of samples ($N_d N_s$). The difference is due to the normal rays reusing the directions from the horizon rays. The cost the normal occlusion is proportional to the number of rays N_n.

Resolution	Geometry[ms]	HO[ms]	NO[ms]	Blur[ms]
640x480	4.02	7.16	4.67	0.71
800x600	4.40	10.55	7.24	1.33
1024x768	4.96	17.09	12.53	2.09
1280x1024	5.92	30.71	26.54	3.23
1600X1153	6.68	39.66	37.41	4.35

Table 1: *Break down of the performance for Figure 1(b). HO and NO stand for horizon and normal occlusion respectively.* $R = 0.2$, $N_d = 7$, $N_r = 1$, $N_s = 12$. *Num triangles:* 750,000.

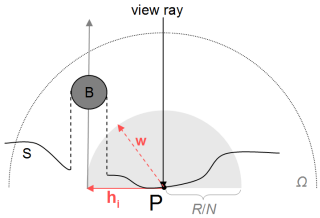


Figure 13: *Discontinuity / outlier problem. It is unclear whether B is connected to S.*

In Figure 11, we present different render qualities based on a fixed radius R and different sets of parameters. In (a) and (d), the algorithm is configured for performance and quality respectively and in (b) and (c), we show the difference from adding normal occlusion, which improves proximity occlusion.

As shown on Figures 11 and 14, our algorithm produces similar images to object-based ray tracing. For some scenes, the contribution of the horizon occlusion term in Equation 3 can dominate the final occlusion value. In this case, the ambient occlusion can be approximated by skipping the contribution of the normal rays. For example, Figures 10 (a) and (b) show the separate contributions of the horizon occlusion and the normal occlusion respectively. Excluding the contact clues, the horizon occlusion captures the relief which is dominant in this case. In general, normal occlusion is needed when there are occluders located in the normal direction relative to the surface point **P**.

Compared to Monte-Carlo ray marching, our implementation is more efficient because tracing one horizon ray has approximately the same cost as tracing one regular ray while carying the information from all the normal rays below it (Figure 2). The horizon occlusion converges faster because it is a close-form solution of the occlusion integral below the horizon, whereas ray marching performs a discrete approximation. In addition, horizon tracing walks on the surface while ray marching shoots rays blindly. Thus, as seen in Figure 12, ray marching requires more rays than horizon occlusion to produce similar images.

The method presented in this paper uses solely screen information to calculate ambient occlusion for all pixels on the screen. This introduces some inherent limitations due to the lack of information in the input data, which may cause underocclusion or overocclusion. For example, it is a common case that scenes present depth discontinuities when there are objects occluding others from the eye's point of view (Figure 13). The horizon estimation could overestimate the occlusion at the points where there is a depth discontinuity within the radius of influence. On the other hand, we could modify the algorithm to underestimate the horizon occlusion and deflect $\vec{h}_i$ only if the detected occluder is within a growing sphere centered at **P**. A more general solution to this problem could be to capture more that one depth layer using a depth peeling approach. For instance, having two depth layers, one with front faces and another with back faces would capture the closest occluders to the eye and improve the accuracy of the algorithm [Shanmugam and Arikan 2007]. However this would require the application to render the geometry a second time to fill the second layer.

Another limitation of our proposed method (and any screen-space method) is that we cannot reconstruct information that is not visible. For example when rotating a 3D model, sometimes popping artifacts can be seen. These occur when a geometric feature becomes visible or disappears from the view. This will be reflected as a sudden change on ambient occlusion on the geometry nearby. This problem can be addressed by using many layers of depth peeling or using multiple views at the additional cost of more geometry rendering passes.

6 Conclusions

In this paper, we have introduced a screen-space ambient occlusion algorithm that uses a combination of analytical solution and ray marching to calculate the ambient occlusion at interactive rates. Moreover the proposed method only requires per pixel depth and normal information as input data. We introduce the concept of hemisphere horizon split, that divides the hemisphere for ambient occlusion estimation into two parts. We describe an efficient search to find this split and an analytical formula to trivially evaluate occlusion contributions on the section of the hemisphere below it. Our method shows faster convergence compared to classic ray marching and shows less quantization of shades of gray for low number of traced rays. Since the proposed method operates as a post-process for a given scene it has a fixed cost independent of the geometry. The observed performance makes it suitable for integration in interactive and real-time rendering software using current and future graphics hardware. We have presented experiments showing the influence of the different parameters of the algorithm on both the quality and the performance of the generated images. We have also compared the images to the ones from a commercial ray tracer observing that similar levels of quality can be obtained at a fraction of the computational cost. Finally we have discussed the limitations of the proposed solution and our goal is to address these in the future following some of the proposed approaches.

Acknowledgements

Removed for anonymous submission We thank the Stanford mesh repository and the Michelangelo project for the dragon, buddha and David head meshes.

References

AMANATIDES, J., AND WOO, A. 1987. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*. Elsevier Science Publishers, Amsterdam, North-Holland, 3–10.

BABOUD, L., AND DÉCORET, X. 2006. Rendering geometry with relief textures. In *GI '06: Proceedings of Graphics Interface 2006*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 195–201.

BRAWLEY, Z., AND TATARCHUK, N. 2004. Parallax occlusion mapping: Self-shadowing, perspective-correct bump mapping using reverse height map tracing. In *ShaderX3: Advanced Rendering Techniques in DirectX and OpenGL*, W. Engel, Ed. Charles River Media, Cambridge, MA.

BUNNELL, M. 2005. Dynamic ambient occlusion and indirect lighting. In *GPU Gems 2*, 223–233.

CADET, G., AND LÉCUSSAN, B. 2007. Fast approximate ambient occlusion. In *SIGGRAPH '07: ACM SIGGRAPH 2007 posters*, ACM Press, New York, NY, USA, 191.

CHRISTENSEN, P., FONG, J., LAUR, D. M., AND BATALI, D. 2006. Ray tracing for the movie 'cars'.

CHRISTENSEN, P. H. 2003. Global illumination and all that. In *ACM SIGGRAPH Course 9 (RenderMan, Theory and Practice)*.

DAVIS, S. T., AND WYMAN, C. 2007. Interactive refractions with total internal reflection. In *GI '07: Proceedings of Graphics Interface 2007*, ACM Press, New York, NY, USA, 185–190.

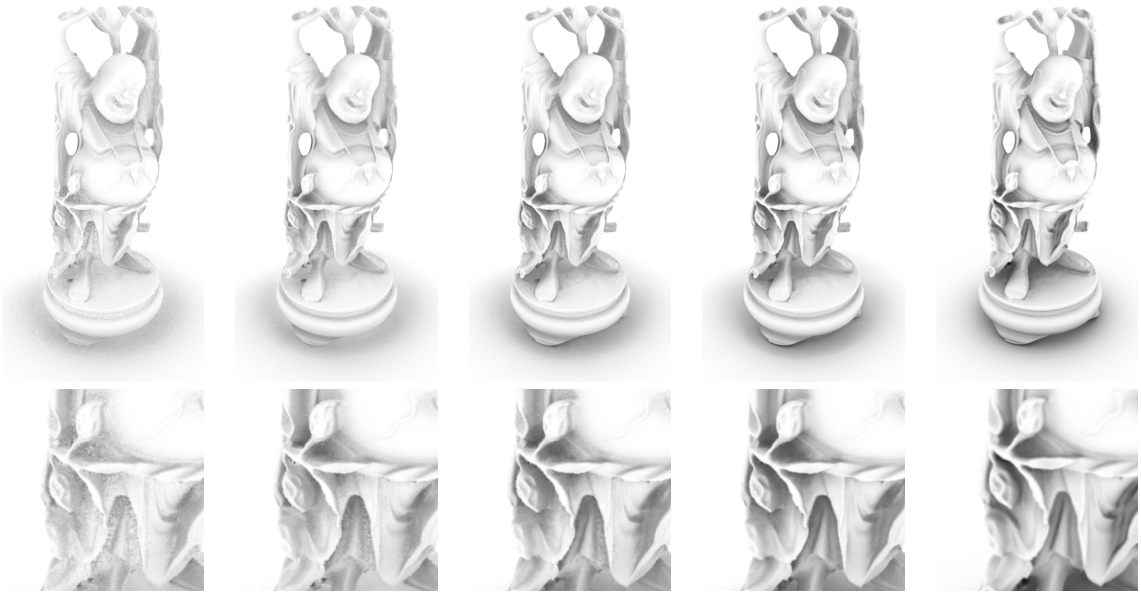
DOWNS, L., MÖLLER, T., AND SÉQUIN, C. H. 2001. Occlusion horizons for driving through urban scenery. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 121–124.

FRANKLIN, D. 2005. Hardware-based ambient occlusion. In *ShaderX 4*, 91–100.

2006. Gelato 2.1 technical reference. Tech. rep., NVIDIA.

(a) (b) (c)

Figure 10: *(a) Horizon occlusion contribution (b) Normal occlusion contribution (c) Final occlusion.*



(a) 58 fps (b) 29 fps (c) 18 fps (d) 5 fps (e) 8 min (ray tracer)

Figure 11: *Rendering results for the Buddha dataset with various sets of parameters in* 512*x*1024*. All images have the same radius of influence* $R = 0.3$ *and use linear attenuation. (a)* $N_d = 6$*,* $N_s = 9$*,* $N_n = 0$*. (b)* $N_d = 8$*,* $N_s = 18$*,* $N_n = 0$*. (c)* $N_d = 8$*,* $N_s = 18$*,* $N_n = 1$*. (d)* $N_d = 18$*,* $N_s = 25$*,* $N_n = 2$*. (e) Rendered with Mental Ray with 256 rays per pixel and adaptive 1-16x supersampling.*

GUENNEBAUD, G., BARTHE, L., AND PAULIN, M. 2006. Real-time soft shadow mapping by backprojection. In *Eurographics Symposium on Rendering*, 227–234.

GUENNEBAUD, G., BARTHE, L., AND PAULIN, M. 2007. High-quality adaptive soft shadow mapping. In *Eurographics*.

HEGEMAN, K., PREMOŽE, S., ASHIKHMIN, M., AND DRETTAKIS, G. 2006. Approximate ambient occlusion for trees. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 87–92.

HOBEROCK, J., AND JIA, Y. 2007. High-quality ambient occlusion. In *GPU Gems 3*, 257–274.

IONES, A., KRUPKIN, A., SBERT, M., AND ZHUKOV, S. 2003. Fast, realistic lighting for video games. *IEEE Comput. Graph. Appl. 23*, 3.

KAJIYA, J. T. 1986. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 143–150.

KONTKANEN, J., AND LAINE, S. 2005. Ambient occlusion fields. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 41–48.

LANDIS, H. 2002. Production ready global illumination. In *ACM SIGGRAPH Course 16: Renderman in Production*, L. Gritz, Ed.

LANGER, M. S., AND BULTHOFF, H. H. 1999. Perception of shape from shading on a cloudy day. Tech. Rep. 73, Max-Planck-Institut fur biologische Kybernetik.

LLOYD, B., AND EGBERT, P. 2002. Horizon occlusion culling for real-time rendering of hierarchical terrains. In *VIS '02: Proceedings of the conference on Visualization '02*, IEEE Computer Society, Washington, DC, USA, 403–410.

MALMER, M., MALMER, F., ASSARSSON, U., AND HOLZSCHUCH, N. 2006. Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools*.

MÉNDEZ, A., SBERT, M., AND CATÁ, J. 2003. Real-time obscurances with color bleeding. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, ACM Press, New York, NY, USA, 171–176.

MILLER, G. 1994. Efficient algorithms for local and global accessibility shading. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 319–326.

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

OLIVEIRA, M. M., AND BRAUWERS, M. 2007. Real-time refraction through deformable objects. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 89–96.

OLIVEIRA, M. M., AND POLICARPO, F. 2005. An efficient representation for surface details. Tech. Rep. RP-351, UFRGS.

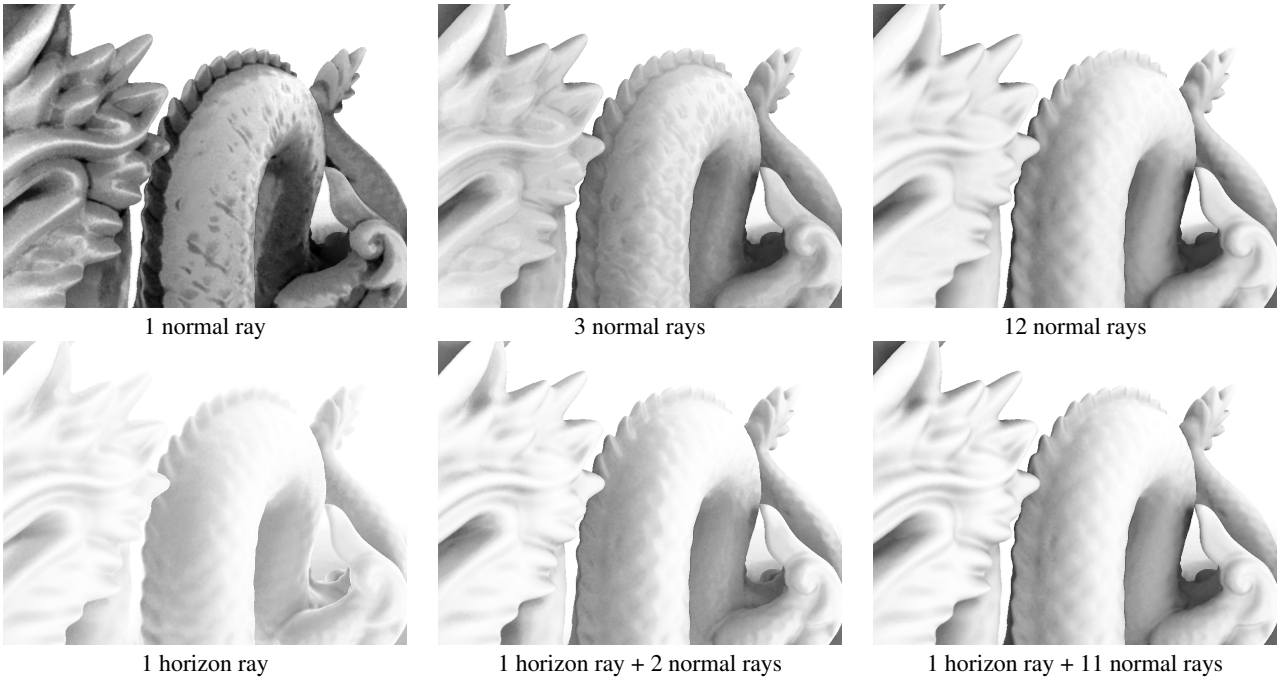


Figure 12: *(First row) Ray marching in the depth buffer. (Second row) Our algorithm. Ambient occlusion purely based on ray marching converges to a similar image as our algorithm, but at a much slower rate.*

(a) 60 fps (b) 19 fps (c) ray tracing.

Figure 14: *(a)* $N_d = 10$*,* $N_s = 6$*,* $N_n = 2$ *(b)* $N_d = 13$*,* $N_s = 8$*,* $N_n = 4$ *(c) Mental Ray with 256 samples per pixel.*

PHARR, M., AND GREEN, S. 2004. Ambient occlusion. In *GPU Gems*, 667–692.

PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

POLICARPO, F., OLIVEIRA, M. M., AND JO A. L. D. C. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM Press, New York, NY, USA, 935–935.

ROGERS, D. F. 1985. *Procedural elements for computer graphics*. McGraw-Hill, Inc., New York, NY, USA.

SATTLER, M., SARLETTE, R., ZACHMANN, G., AND KLEIN, R. 2004. Hardware-accelerated ambient occlusion computation. In *9th Int'l Fall Workshop VISION, MODELING, AND VISUALIZATION (VMV)*, 119–135.

SCHWARZ, M., AND STAMMINGER, M. 2007. Bitmask soft shadows. In *Eurographics*.

SHANMUGAM, P., AND ARIKAN, O. 2007. Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 73–80.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 527–536.

STEWART, A. J. 1997. Hierarchical visibility in terrains. In *Eurographics Rendering Workshop*.

TATARCHUK, N. 2006. Practical parallax occlusion mapping with approximate soft shadows for detailed surface rendering. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, ACM Press, New York, NY, USA, 81–112.

WASSENIUS, C., 2005. Accelerated ambient occlusion using spatial subdivision structures.

ZHUKOV, S., INOES, A., AND KRONIN, G. 1998. An ambient light illumination model. In *Rendering Techniques '98*, G. Drettakis and N. Max, Eds., Eurographics, 45–56.

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.