Noname manuscript No.
(will be inserted by the editor)

Multi-Resolution Screen-Space Ambient Occlusion

Thai-Duong Hoang · **Kok-Lim Low**

the date of receipt and acceptance should be inserted later

Abstract We present a new screen-space ambient occlusion (SSAO) algorithm that improves on the state-of-the-art SSAO methods in both speed and image quality. Our method computes ambient occlusion (AO) for multiple image resolutions and combines them to obtain the final high-resolution AO. It produces high-quality AO that includes details from small local occluders to low-frequency occlusions from large faraway occluders. This approach allows us to use very small sampling kernels at every resolution, and thereby achieve high performance without resorting to random sampling, and therefore our results do not suffer from noise and excessive blur, which are common of SSAO. We use bilateral upsampling to interpolate lower-resolution occlusion values to prevent blockiness and occlusion leaks. Compared to existing SSAO methods, our method produces results closer to ray-traced solutions, while running at comparable or higher frame rates than the fastest SSAO method.

Keywords ambient occlusion · multi-resolution · screen-space · bilateral upsampling · global illumination

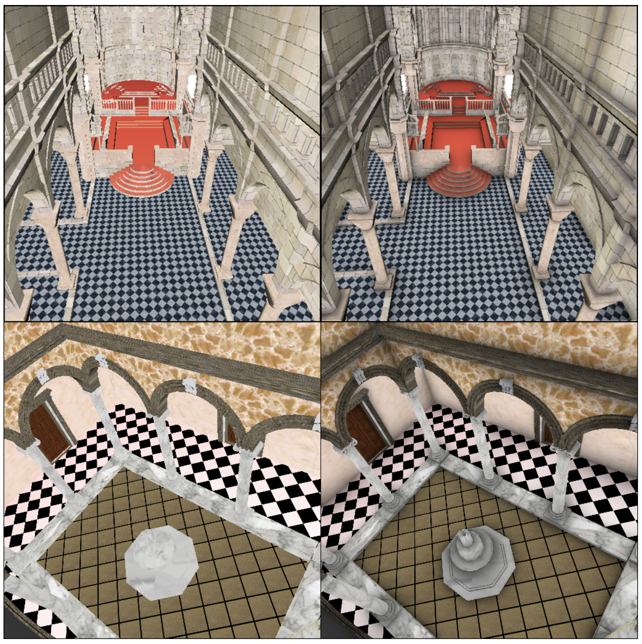
Fig. 1 (Left) Two scenes rendered without AO. (Right) The same scenes rendered with AO produced by our method (at more than 40 fps on NVIDIA GeForce 8600M GT at 512x512 pixels).

1 Introduction

Ambient occlusion (AO) is the lighting/shadowing effect under the direct illumination by a diffuse spherical light source surrounding the scene. The result is that concave areas such as creases or holes appear darker than exposed areas. AO is not a real-world phenomenon, since in reality, incoming light is rarely equal in all directions and inter-reflections occur between surfaces. Despite these, AO can add a significant degree of realism to a rendered image. It gives a sense of shape and depth in an otherwise "flat-looking" scene (see Fig. 1).

Several methods to compute AO exist. They differ in the extent to which accuracy is traded for speed. The most accurate methods often use Monte Carlo ray tracing in object space, but they are slow and only suitable for offline pre-computation. Real-time AO has recently become possible, mainly due to the advancement in programmable graphics hardware. A class of real-time methods, collectively known

Thai-Duong Hoang
Department of Computer Science
National University of Singapore
E-mail: duong@comp.nus.edu.sg

Kok-Lim Low
Department of Computer Science
National University of Singapore
E-mail: lowkl@comp.nus.edu.sg

as *screen-space ambient occlusion* (SSAO), trades AO accuracy for significant increase in performance. SSAO has become increasingly popular in games and other interactive applications that favor speed over accuracy. There exist many SSAO implementations that share the same core idea but differ in details, and their results can be vastly different from one another's. Current SSAO methods have no difficulty in producing local AO effects, such as the darkening of small creases, but are facing great performance challenges in producing more global AO effects caused by faraway occluders. Some of them try to get around the problem by random sampling, but the results are either noisy, or look blurry when low-pass filters are used to reduce the noise.

The contribution of this paper is a new SSAO algorithm, which we call *Multi-Resolution Screen-Space Ambient Occlusion* (MSSAO), that computes AO for multiple image resolutions and combines them to obtain the final high-resolution AO. Our method is able to produce high-quality AO that includes contributions by small local occluders to those by large faraway occluders. The approach is based on the principle that AO due to faraway occluders are low-frequency, thus can be computed in coarser resolutions; whereas AO due to nearby occluders are high-frequency, and thus must be computed in finer resolutions. This approach allows us to use very small sampling kernels at every resolution, and thereby achieve high performance without resorting to random sampling. In this paper, our method is compared with two state-of-the-art SSAO methods, and it is shown to produce results closer to ray-traced solutions, while running at comparable or higher frame rates than the fastest SSAO method.

2 Ambient Occlusion Theory

This section discusses the theoretical basis of AO. We assume that the incoming radiance is constant for all incoming directions. For simplicity, we also assume all surfaces are Lambertian. The equation for surface irradiance at point p with normal $\mathbf{n}$ is

$$E(p,\mathbf{n}) = L_A \int_{\Omega} v(p,\omega)\cos\theta d\omega, \tag{1}$$

where L_A is the incoming ambient radiance; Ω is the hemisphere above point p, representing all possible incoming directions; $v(p,\omega)$ is a binary visibility function that equals 0 if a ray cast from point p in direction ω is blocked, and equals 1 otherwise; θ is the angle between ω and $\mathbf{n}$; $d\omega$ is an infinitesimal solid angle along direction ω. We can also write Equation 1 as

$$E(p,\mathbf{n}) = L_A \pi k_A(p,\mathbf{n}), \tag{2}$$

where

$$k_A(p,\mathbf{n}) = \frac{1}{\pi}\int_{\Omega} v(p,\omega)\cos\theta d\omega. \tag{3}$$

k_A is the AO and its value ranges from 0 to 1. When k_A is 0, the point p is totally blocked from light; when it is 1, p is totally exposed. Although k_A is called "ambient occlusion", it actually corresponds to how much of the hemisphere above p is visible, or its "accessibility".

The above definition of AO is not useful for enclosed scenes, where it would be totally dark since $v(p,\omega)$ equals 0 everywhere. That is why in practice, the binary visibility function $v(p,\omega)$ is often replaced by an attenuation (or falloff) function $\rho(p,d)$ which varies smoothly from 1 to 0. With ρ, we can rewrite k_A as

$$k_A(p,\mathbf{n}) = 1 - \frac{1}{\pi}\int_{\Omega} \rho(p,d)\cos\theta d\omega. \tag{4}$$

$\rho(p,d)$ is a continuous function that depends on the distance d between p and the point where a ray cast from p in direction ω intersects some nearby geometry. As d increases from 0 to some manually-set value d_{max}, $\rho(p,d)$ decreases monotonically from 1 to 0. The use of ρ is somewhat empirical, but it is more useful than a binary visibility function.

3 Related Work

Here we briefly discuss existing AO methods that are targeted for real-time interactive frame rates on dynamic scenes, with a focus on SSAO.

3.1 Object-Space Methods

Bunnell [3] approximates a scene's objects by a hierarchy of disks. AO is calculated using approximated form factors between all pairs of disks. Further improvements have been achieved by [7] (less artifacts) and [4] (better accuracy). These methods require highly tessellated geometry, and cannot scale beyond simple scenes without sacrificing a lot of performance.

Reinbothe et al. [17] compute AO by ray-tracing in a voxelized representation of the scene instead of the original triangle mesh. Ray-tracing is slow for real-time applications, even when working with near-field voxels instead of triangles.

Ren et al. [18] and Sloan et al. [21] approximate occluders with spheres, and use spherical harmonics to analytically compute and store AO values. AO due to multiple spheres are accumulated by efficiently combining the corresponding spherical harmonics coefficients. Shanmugam et al. [20] use a similar approach with spheres and image-space splatting, but without spherical harmonics. These methods require a pre-processing step, and do not work for scenes with complex objects that cannot be approximated by spheres.

Kontkanen et al. [9] and Malmer et al. [13] compute an occlusion field around each occluder and store it in a cube map. During rendering, occlusion due to multiple objects is approximated by looking up and blending pre-computed values from different cube maps. Zhou et al. [25] propose a similar technique that uses either Haar wavelets or spherical harmonics instead of cube maps. AO fields require large memory storage and only work for semi-dynamic scenes composing of rigid objects. Self-occlusion and high-frequency occlusion are also ignored.

McGuire [14] analytically computes, for each screen pixel, occlusion caused by every triangle mesh in a scene. This method suffers from over-occlusion artifacts. Laine et al. [11] solve this problem by considering occlusion contributions from occluders coming from the same hemispherical direction only once. This idea is similar to hemispherical rasterization by [8], but the latter only works for self-shadowing objects. Analytical methods are slow, especially for big scenes with many triangles. Thus these methods are not yet suitable for real-time applications.

3.2 Screen-Space Methods

Screen-space methods use empirical models that darken a pixel by using its nearby pixels as occluders. Mittring [15] introduces one of the first known SSAO methods. This method samples 3D points inside a sphere centered at a shaded pixel, and determines how many of them are below the scene surface as seen from the eye by projecting the point samples into screen space. AO is defined as the ratio between the number of occluding samples and the total number of samples. Methods that improve on this idea are [5] (attenuation function, no self-occlusion), [19] (directional occlusion, one-bounce indirect illumination), [12] (line sampling, which is faster and more robust), and [23] (similar to the previous work, interleaved sampling).

Shanmugam et al. [20] and Fox et al. [6] sample directly in image-space instead of projecting 3D point samples. For a shaded pixel, neighboring pixels are randomly sampled and their corresponding object-space points are treated as micro-spheres occluders.

Bavoil et al. [2] compute AO by finding the horizon angle along each 2D direction from a shaded pixel. A horizon angle along a direction indicates how much the shaded pixel is occluded in that direction. AO is averaged from horizon angles in multiple 2D directions. HBAO produces higher-quality results compared to other SSAO methods since it is more analytical, but it is also much slower due to the use of ray marching to step along each direction ray.

Screen-space methods are fast, but suffer from some visual artifacts. Some of them are over-occlusion (since visibility is ignored), under-occlusion (since occluders' projections are either too small or missing on screen), noisy results (due to random sampling), blurry results (due to noise-reduction filters), and very local occlusion (due to small sampling kernels). Our method is able to overcome the noise, blur and local occlusion problems by using multiple resolutions. For the other problems, there are some proposed solutions, such as depth peeling [19], multiple cameras [1], and enlarged camera's field of view [1]. Those fixes can benefit any SSAO method, albeit at considerable performance costs. As such, this paper focuses only on the core ideas of SSAO and includes none of those extensions.

Observing that AO is mostly low-frequency, Sloan et al. [21] compute AO in a coarser resolution and upsample it using joint bilateral upsampling [10]. Bavoil et al. [1] use a similar technique, but also compute full-resolution AO to refine small details. However, as we have found, using only a single coarser resolution is insufficient to capture scene AO that often occurs at multiple different scales. Multi-resolution techniques are also proposed in [16] and [22], but for the purpose of computing one-bounce indirect illumination without visibility checking. Note that AO is not a special case or a sub-problem of indirect illumination.

4 Multi-Resolution SSAO Algorithm

Our algorithm uses a deferred shading framework with a typical g-buffer that stores eye-space positions and normals for each pixel. For each shaded pixel p, we treat its nearby pixels as occluders that may block light from reaching p. We sample the occluders directly from the g-buffer. For occluders further away from p, we can sample them at a coarser resolution than for occluders nearer to p. We assume that pixels close to p in screen space are also close-by in object space. This is not always true (the converse is always true though), but is very likely. Consequently, we use a low-resolution g-buffer to compute occlusion caused by faraway pixels and a high-resolution one for nearby pixels. In fact, we use multiple resolutions of g-buffer to capture AO at multiple scales. By using very small sampling kernels at each resolution, we are in effect sampling a large neighborhood around p. We do not need to resort to sparse random sampling in order to maintain high frame rates, and therefore our results are free of common SSAO artifacts such as noise and blur (see comparison in Fig. 2).

4.1 Overview

Our algorithm works by first rendering a scene's geometry to a g-buffer at the finest resolution. The g-buffer is then downsampled multiple times, similar to creating a mipmap. Then, an occlusion value is computed for every pixel in each resolution, by sampling other pixels (occluders) in its small 2D neighborhood. Finally, for each pixel in the finest resolution,

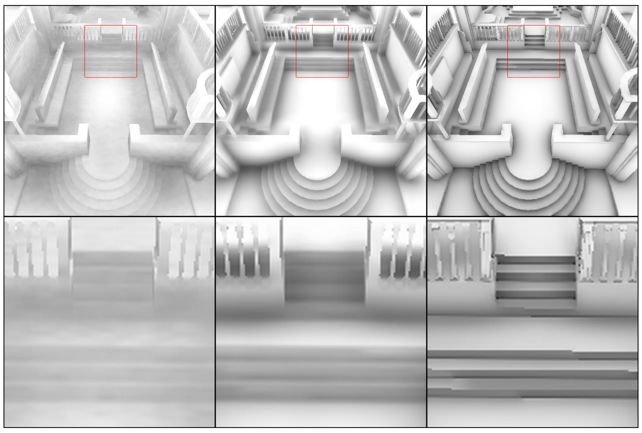
Fig. 2 (Left) Noise artifacts by Blizzard's method, (middle) blurry result by NVIDIA's HBAO, (right) result from our method. Our method produces no noise and preserves sharp edges and fine details better.

its AO value is computed by combining the corresponding occlusion values across multiple resolutions.

A naïve implementation that follows the basic idea above exactly would produce blocky results, since adjacent pixels in a resolution are grouped together in the next coarser resolution, thus sharing the same coarser-resolution occlusion value. To achieve smooth results, we use bilateral upsampling [10], which can avoid upsampling across large depth and normal differences.

In our implementation, AO is calculated from the coarsest resolution to the finest one using multiple rendering passes. In each pass, the occlusion value calculated for each pixel is combined with the occlusion value upsampled from the previous, lower resolution. The overall algorithm is described below.

```
Render scene into g-buffer.
Downsample g-buffer multiple times.
Start with the coarsest resolution.
Repeat until the finest resolution,
  For each pixel p in current resolution,
    Calculate its occlusion value AO1.
    If this is the coarsest resolution,
      Output AO1 (to be used in the next
          iteration).
    Else
      Blur result from previous resolution.
      Upsample blurred result to get AO2.
      Combine AO1 and AO2 to get AO3.
      Output AO3 (to be used in the next
          iteration).
Output AO3 as the final occlusion value.
```

Note that AO_1, AO_2, and AO_3 are per-pixel, per-resolution values. In our algorithm, AO_1 represents occlusion caused by neighboring pixels and AO_2 represents occlusion caused by farther-away pixels. They are combined together in every rendering pass (except the first) to get occlusion value AO_3. For example, suppose p is a pixel in some current resolution during the algorithm. In the next rendering pass (in a finer resolution), the previously computed value AO_3 of p will be upsampled and treated as AO_2 for the higher-resolution pixels near p (see the details in Section 4.5). Note that each AO_1 value is computed independently for all resolutions by sampling in a small 2D neighborhood. The upsampling and combining process ends when the finest resolution is reached, at which point AO_3 is the final AO value for the shaded pixel.

4.2 Downsampling

The downsampling process starts from the finest-resolution g-buffer, and produces a lower-resolution one with each rendering passes. In each pass, each pixel will combine the eye-space positions and normals of its corresponding four sub-pixels in the previous, finer resolution. Instead of using the typical mean value, we use the median in terms of depths. It is important to note that the normals used here are the polygons' normals, not interpolated vertex normals, as the latter would cause self-occlusion artifacts with low-tessellation scene models. The total number of resolutions depends on how "global" we want the AO to be. In practice we often use 5 levels.

The median is used instead of the mean because the former is more robust. That is, when a few numbers out of a set of numbers change sharply, the mean value generally fluctuates more than the median value. Using median helps our method achieve better temporal coherence. Consequently, our results suffer very little (most of the time unnoticeable) popping or shimmering artifacts between frames.

4.3 Occluder Sampling

To compute the occlusion value AO_1 in each resolution, we sample a small screen-space neighborhood around each shaded pixel. We determine the sampling kernel size in each resolution as follows. First, we set a maximum AO radius of influence in 3D object space. Let it be d_{max}, and it is the distance beyond which an occluder contributes no occlusion at all. This distance is then projected into screen space in the finest resolution (or the 0th resolution) so that we have a kernel radius $r_0(p)$ (in terms of number of pixels) for each pixel p. The kernel radius in the next coarser resolution should naturally be $r_1(p) = r_0(p)/2$, and the next should be $r_2(p) = r_1(p)/2 = r_0(p)/4$, and so on. For each shaded pixel p, $r_0(p)$ is calculated using the formula $r_0(p) = (sd_{max})/(2z\tan(\alpha/2))$, where s is the viewport's dimension in the finest resolution (assuming a square viewport), α is the camera's field-of-view angle, and z is the eye-space depth value of pixel p.

Note that the kernel size depends on the depth of each individual pixel because of perspective fore-shortening. To accurately reflect this, pixels nearer to the camera require larger kernel sizes in screen space. In practice, the values $r_0(p)$, $r_1(p)$, etc. can be large, thus we cap the radius to r_{max} pixels in any resolution, otherwise performance can drop very drastically when more pixels become closer to the camera (when the scene is zoomed in). Therefore, the final kernel radius at a pixel p at the ith level of resolution is $min(r_{max}, r_i(p))$. A typical value of r_{max} is 5 (a 11x11 kernel), except in the finest resolution, where it is capped at 2, mainly for performance reason. In coarser resolutions, we do not sample every pixel in a 11x11 kernel; instead we only sample every other pixel, effectively reducing the number of texel fetches from 81 to 25 per fragment. This is to take advantage of the blur pass that follows. Note that the kernel radius can be smaller than 1. In other words, we need not take any samples in some particular resolution if the kernel radius is smaller than a pixel.

4.4 Computing Ambient Occlusion

For each pixel p in a resolution, the next step after gathering N samples is to use them to compute the occlusion value AO_1. We use the following formula

$$AO_1(p) = \frac{1}{N}\sum_{i=1}^{N} \rho(p,d_i)\overline{\cos\theta_i}. \tag{5}$$

d_i is the distance between the ith occluder and the shaded point in object space, and θ_i is the angle between the shaded point's normal and the vector joining the shaded point and the ith occluder. The bar over the cosine term means its value is clamped to $[0,1]$. $\rho(p,d_i)$ is a falloff function that smoothly attenuates an occluder's contribution as it goes further away from the shaded point. If the nearby pixels correspond to uniform directions in the hemisphere above the shaded point, Equation 5 becomes a Monte Carlo approximation of Equation 3. Even though our formula is empirical, it is more robust than the original Crytek's formula [15] and those of the other methods based on it, such as Blizzard's [5]. That is, the AO computed by our formula has lower variance, and thus we need fewer samples to produce noise-free results.

The falloff function $\rho(p,d_i)$ must smoothly decreases from 1 to 0 as the distance d_i increases from 0 to some constant d_{max}. We use the following simple formula to compute ρ:

$$\rho(p,d_i) = 1 - min(1,(\frac{d_i}{d_{max}})^2). \tag{6}$$

We choose this quadratic falloff function because each occluder can be considered a very small sphere, and the solid angle subtended by it varies inversely to its squared distance from the 3D position of the shaded pixel.

4.5 Combining Occlusion Values

Now that we have computed AO_1, we must combine it with the occlusion values upsampled from the previous, coarser resolution, AO_2. To compute AO_2, we use a bilateral upsampling algorithm, which can avoid upsampling across large depth and normal discontinuities. The upsampling blends occlusion values from the four pixels in the previous resolution that are closest to the current-resolution pixel p (see Fig. 3). Besides the usual bilinear interpolation weights, the bilateral upsampling algorithm also uses both the depth and normal differences as weights. More specifically, for pixel p (with depth z and normal $\mathbf{n}$), the normal weight and depth weight of a low-resolution pixel p_i (with depth z_i and normal $\mathbf{n}_i$) are, respectively,

$$w_n = (\mathbf{n}\cdot\mathbf{n}_i)^{t_n}, \tag{7}$$

and

$$w_z = (\frac{1}{1+|z_i - z|})^{t_z}. \tag{8}$$

The powers t_n and t_z are to be interpreted as "tolerance" levels, and are dependent on a scene's size. If the scene is small, nearby pixels correspond to nearby 3D points, thus large depth differences must be tolerated less by using a larger power value. The aim is to balance between smooth interpolation of occlusion values and preservation of sharp details. We use $t_n = 8$ and $t_z = 16$ for all the example scenes shown in this paper. Fig. 4 clearly demonstrates the superior quality of bilateral upsampling over the other methods.

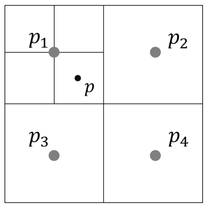

Fig. 3 In the upsampling step, each current-resolution pixel p will blend occlusion values from its four nearest lower-resolution pixels p_1, p_2, p_3, and p_4.

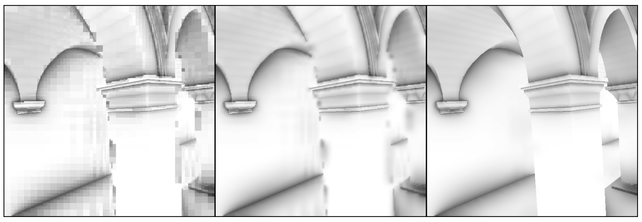

Fig. 4 (Left) Nearest neighbor upsampling, which is blocky, (middle) bilinear upsampling, which leaks occlusion, (right) bilateral upsampling, which is free of visible artifacts.

After upsampling, we have the faraway occlusion value AO_2. We must now combine it with the nearby occlusion value AO_1. We use a maximum function to combine them, i.e. $AO_3 = max(AO_1, AO_2)$ (note that higher occlusion value corresponds to darker shadow). In the implementation, we store AO_3 in a render target and use that as input texture in the next rendering pass, except in the final resolution, where we just output $1 - AO_3$ as the "accessibility" value for the shaded pixel.

The use of the maximum operator to combine occlusion values from multiple resolutions is based on the following observation. The occlusion value at any particular resolution is computed by considering all occluders within a neighborhood around the shaded pixel. These same occluders (in lower-resolution representations) are also included as a subset of occluders considered in the computation of occlusion values at all the coarser resolutions. The use of the maximum operator prevents the inner occluders from being multiply-counted when deriving the final AO value for the shaded pixel, and it also allows occlusion missed in any resolution to be picked up in other resolutions. In comparison, in most other SSAO methods, as the AO radius of influence increases, the final AO values become more incorrect. This is because, without visibility checking, further-away samples often incorrectly dilute occlusion caused by nearby ones. By retaining the maximum occlusion value across the resolutions, we can alleviate that ill-effect to some extent.

Since we retain the maximum occlusion value in each rendering pass, the final output may not look "clean". That is because neighboring pixels may have different maximum occlusion values computed in different resolutions. That problem can be solved by adding a blur pass just before the upsampling. The blurring method is essentially identical to bilateral upsampling. The only differences are that it is done in the same resolution (instead of cross-resolution), and uses a slightly larger kernel (3x3 pixels instead of 2x2 pixels). Fig. 5 illustrates the process of combining occlusion values across multiple resolutions. One can see that the bilateral upsampling has prevented much of the shadow leakage in the coarser resolutions.

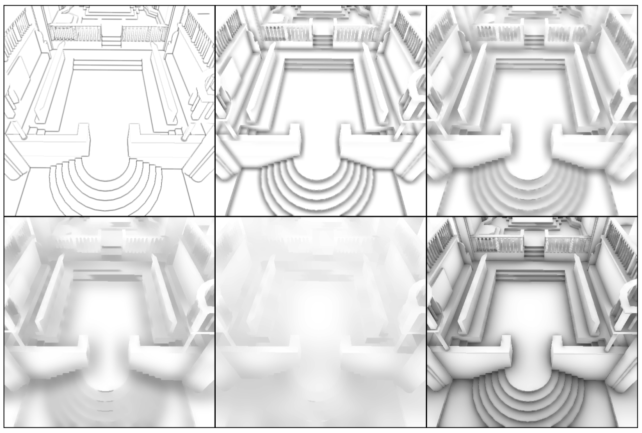
Fig. 5 Combining AO values across multiple resolutions to get the final AO value. The first 5 images are AO values computed at 5 different resolutions, from the finest to the coarsest. The last image (bottom-rightmost) is the final result. Note that the images shown here are outputs after blurring and upsampling.

5 Evaluations and Comparisons

We compare our method (MSSAO) with Blizzard's SSAO [5] and NVIDIA's HBAO [2] in both performance and visual quality. Blizzard's method is an enhanced version of Crytek's and is one of the fastest SSAO methods. On the other hand, NVIDIA's HBAO produces the best looking images among SSAO methods. Ray-traced images produced by the Blender software are also available as ground-truth references. In our experiments, we have also implemented and evaluated another recent SSAO method, Volumetric Ambient Occlusion [23]. As this method produced lower-quality results than HBAO and has poorer performance than Blizzard's SSAO, we decided to leave it out of the comparisons.

Tests are run on two machines, one with a NVIDIA GeForce 8600M GT, and the other with a more powerful NVIDIA GeForce 8800 GTS graphics card. All programs are configured to render at the final resolution of 512x512 pixels. Three test scenes are used: Sibenik Cathedral, Conference Room, and Sponza Atrium. These scenes are three of the standard benchmarking scenes used to evaluate global illumination algorithms. They feature geometry that is usually difficult for SSAO methods, such as sharp edges, holes, pillars, thin chair legs, small staircases, etc. The maximum AO radius of influence (d_{max}) is set to 2 units in all scenes and all methods. In all three scenes, our method uses 5 resolutions, with the maximum screen-space kernel radius (r_{max}) set to 5 in coarser resolutions and 2 in the finest resolution. The contrast and falloff parameters in the HBAO method have been carefully adjusted to best match the ground-truth results. It uses 16 screen-space directions and 8 ray-marching steps per direction. Blizzard' SSAO uses 32 samples per pixel. Blizzard's falloff function depends not only on the distance between an occluder and a shaded point, but also on the depth difference between an occluder and its projected pixel.

5.1 Qualitative Evaluations

Fig. 6 presents a comparison of visual quality among the three SSAO methods and the ground-truth references produced by Blender (images in the electronic version of this paper can be zoomed in for more details)[1]. Results show

[1] A bug in Blender has caused small dark spots in the rendered image of the Sibenik Cathedral model. These artifacts, however, do not affect the comparison.

that our method produces cleaner and sharper images that are closer to the ray-traced images. In all scenes, our method achieves a more natural attenuation of AO. Also, high-frequency geometric details such as sharp edges are preserved because no blur pass in the finest resolution is used. Blizzard's results are noisy while NVIDIA's are blurry and suffer from over-occlusion artifacts (see, for example, the contact shadows between the chairs and the ground in the Conference Room scene). In both methods, geometric details are not well-preserved (see the staircase in the Sibenik Cathedral scene, or the details on the chairs in the Conference Room scene, for example).

In Table 1, we also provide numerical comparison between the results using SSIM [24], which is a perceptual image comparison metric. Results from our method have the highest similarities to the ground-truth images.

	MSSAO	Blizzard	HBAO
Sibenik	85.43%	67.34%	78.04%
Conference	86.62%	78.82%	82.23%
Sponza	88.72%	74.96%	82.10%

Table 1 SSIM indices of the results from the three SSAO methods against the reference images. Each SSIM number is shown as the percentage of similarity between two images. Larger numbers are better.

Fig. 7 shows additional results from our method for different kinds of models, to show that our algorithm is suitable for a wide range of scenes. The final important thing to note is that our method generally does not have noticeable flickering/shimmering artifacts when the camera is moving, despite the use of multiple resolutions.

5.2 Performance

Table 2 shows the running times of all three SSAO methods for rendering a frame of the three test scenes. We measure the actual timings of the AO algorithms, so the geometry passes are ignored in all methods.

	GeForce 8600M GT		
	MSSAO	Blizzard	HBAO
Sibenik	19.2	24.9	92.8
Conference	19.5	22.2	90.4
Sponza	18.2	30.6	92.9
	GeForce 8800 GTS		
Sibenik	5.7	4.4	12.3
Conference	5.5	4.3	13.6
Sponza	5.8	4.9	14.0

Table 2 Runtime performance of the three SSAO methods, measured in millisecond (ms). Smaller numbers are better.

As polygon counts do not affect the complexity of SSAO algorithms, the differences in rendering speeds come mostly from the amount of computations and texture fetches per fragment. HBAO is the slowest method due to the fact that ray marching is done per pixel in many directions in screen space. On average, taking into account the occluder sampling, blurring, and upsampling, our algorithm fetches about 42 texels per fragment, while that number is 32 for Blizzard's method. For our method, we calculate the average number of texel fetches per fragment by adding the number of texel fetches across all resolutions, then divide it by the number of pixels in the highest resolution (512x512 in this case). On the 8600M GT, our method can achieve better performance since the kernel sizes are small and the sampling pattern is more cache-friendly. On the newer graphics card, Blizzard's method outperforms ours by a small margin. That is probably because of the GPU's bigger texture cache, so random sampling patterns and big kernel sizes do not hurt performance as much.

	GeForce 8800 GTS		
	MSSAO	Blizzard	HBAO
Sibenik	5.7	5.9	14.5
Conference	5.5	5.4	14.2
Sponza	5.8	6.4	15.2

Table 3 Runtime performance of the three SSAO methods with d_{max} = 4, measured in millisecond (ms). Smaller numbers are better.

However, our method can scale better with larger AO radius of influence (d_{max}). It is because doubling d_{max} in our method only requires adding one more level of resolution, while in the other methods it requires sampling in a screen-space region four-times as large. This is reflected in Table 3, when d_{max} is changed from 2 to 4. Our method maintains the same performance while Blizzard's SSAO and HBAO's performances degrade more. In general, we believe our method has achieved the high performance of the Blizzard's method and attained better image quality than that of NVIDIA's HBAO.

6 Discussion and Conclusion

Being a screen-space method, our method inherits the fundamental limitations of SSAO. The AO computed by our method is a coarse approximation of true AO. It is incorrect in places where occluders are hidden from the viewpoint — either outside the view frustum or behind the first layer of depth. There are workarounds, such as using various heuristics to "guess" the hidden occluders, but that does not solve the problem completely, and in many cases creates other artifacts. Depth peeling, multiple cameras, and enlarged field-of-view are some of the solutions already proposed. These extensions can improve accuracy to some extent, but they do not fundamentally change the core idea of SSAO. As long as insufficient data is given as input (e.g. only a g-buffer), artifacts are unavoidable.

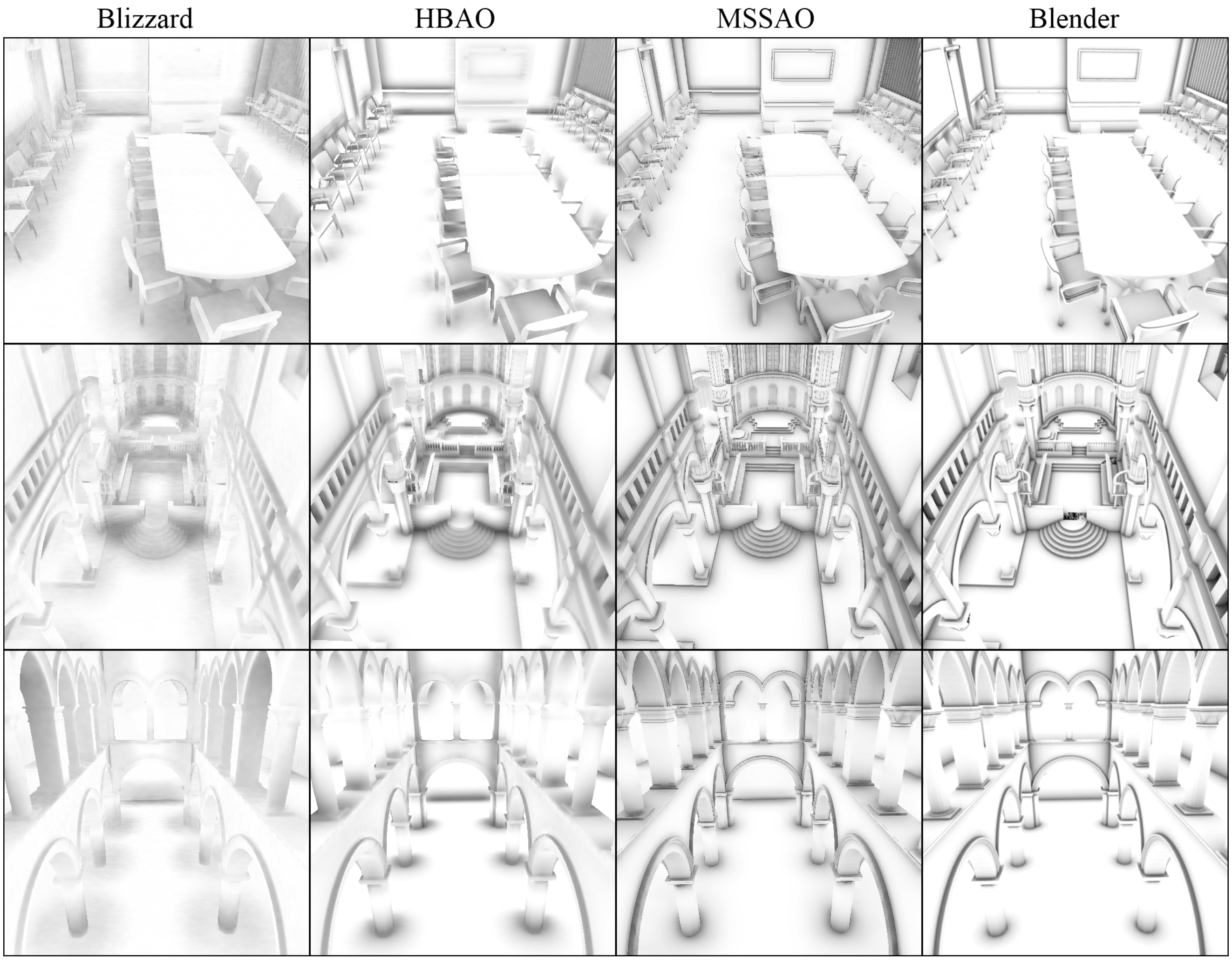

Fig. 6 Comparisons of AO results (without any other shading) for three test scenes. Scenes: (first row) Conference Room, (second row) Sibenik Cathedral, (third row) Sponza Atrium. Methods: (first column) Blizzard, (second column) HBAO, (third column) MSSAO, (fourth column) Blender. Images in the fourth column are the reference solutions.

Despite having some inherent limitations of SSAO, our method has improved on the current SSAO methods in a number of ways. Current SSAO methods are limited by the number and coherency of texture fetches, thus are limited to local occlusion. By sampling from multiple resolutions with small kernels in each, we can reduce the total number of texture fetches, and at the same time leverage the GPU's texture caching to make the algorithm much faster. That enables us to sample further without sacrificing much performance, thus capture more global AO effects. Together with a robust AO formula, our results are free of noise/blur even with small number of samples. Bilateral filtering and upsampling are used repeatedly in coarser resolutions to smoothly blend and interpolate occlusion values, thus avoiding artifacts from the use of multiple resolutions. Our multi-resolution approach is general and can be applied on top of other lower-level AO formulas besides the one described in this paper.

Acknowledgements The scene models used in this paper are from the following sources: Marko Dabrovic (Sibenik Cathedral, Sponza Atrium), Anat Grynberg (Conference Room), ompf.org's Scene Vault (DWC-51 Truck, Jagdpanther Tank, Kitchen), De Espona (Hebe Statue), Brian Curless (Dragon), Crytek (modified version of Sponza Atrium), elias_t (House), and Haerle Johannes (Oak Tree).

References

1. Bavoil, L., Sainz, M.: Multi-layer dual-resolution screen-space ambient occlusion. In: ACM SIGGRAPH 2009 Talks (2009)
2. Bavoil, L., Sainz, M., Dimitrov, R.: Image-space horizon-based ambient occlusion. In: ACM SIGGRAPH 2008 Talks (2008)
3. Bunnell, M.: Dynamic ambient occlusion and indirect lighting. In: GPU Gems 2, pp. 223–233. Addison-Wesley Professional (2005)
4. Christensen, P.: Point-based approximate color bleeding. Pixar Technical Memo #08-01 (2008)
5. Filion, D., McNaughton, R.: Effects & techniques. In: ACM SIGGRAPH 2008 Courses, pp. 133–164 (2008)
6. Fox, M., Compton, S.: Ambient occlusive crease shading. Game Developer Magazine (March 2008) (2008)

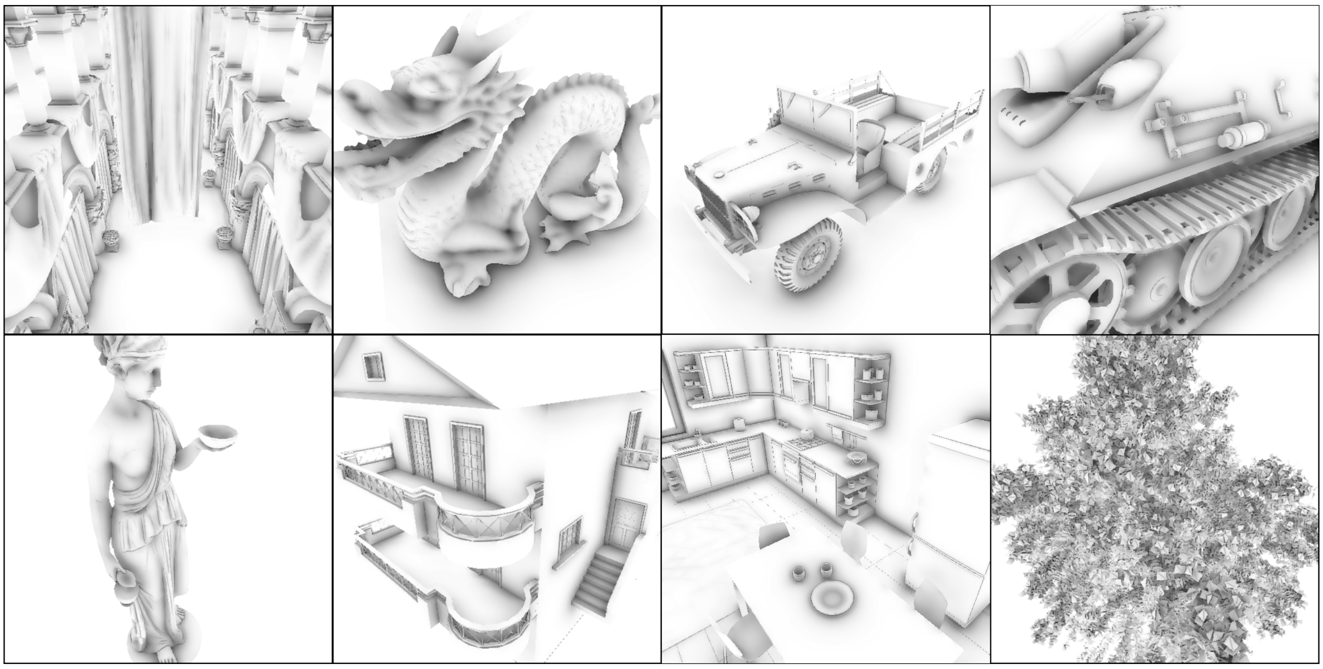
Fig. 7 Additional results from our method. Note that these images are all rendered at more than 40 fps on a NVIDIA GeForce 8600M GT.

7. Hoberock, J., Jia, Y.: High-quality ambient occlusion. In: GPU Gems 3, pp. 257–274. Addison-Wesley Professional (2007)
8. Kautz, J., Lehtinen, J., Aila, T.: Hemispherical rasterization for self-shadowing of dynamic objects. In: Proceedings of the Eurographics Symposium on Rendering 2004, pp. 179–184 (2004)
9. Kontkanen, J., Laine, S.: Ambient occlusion fields. In: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games, pp. 41–48 (2005)
10. Kopf, J., Cohen, M.F., Lischinski, D., Uyttendaele, M.: Joint bilateral upsampling. In: Proceedings of ACM SIGGRAPH 2007 (2007)
11. Laine, S., Karras, T.: Two methods for fast ray-cast ambient occlusion. Computer Graphics Forum **29**(4), 1325–1333 (2010)
12. Loos, B.J., Sloan, P.P.: Volumetric obscurance. In: Proceedings of the 2010 Symposium on Interactive 3D Graphics and Games, pp. 151–156 (2010)
13. Malmer, M., Malmer, F., Assarsson, U., Holzschuch, N.: Fast precomputed ambient occlusion for proximity shadows. Journal of Graphics Tools **12**(2), 59–71 (2007)
14. McGuire, M.: Ambient occlusion volumes. In: Proceedings of the Conference on High Performance Graphics, pp. 47–56 (2010)
15. Mittring, M.: Finding next gen: Cryengine 2. In: ACM SIGGRAPH 2007 Courses, pp. 97–121 (2007)
16. Nichols, G., Wyman, C.: Multiresolution splatting for indirect illumination. In: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, pp. 83–90 (2009)
17. Reinbothe, C., Boubekeur, T., Alexa, M.: Hybrid ambient occlusion. In: Eurographics 2009 Areas Papers (2009)
18. Ren, Z., Wang, R., Snyder, J., Zhou, K., Liu, X., Sun, B., Sloan, P.P., Bao, H., Peng, Q., Guo, B.: Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In: Proceedings of ACM SIGGRAPH 2006, pp. 977–986 (2006)
19. 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 (2009)
20. 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 (2007)
21. Sloan, P.P., Govindaraju, N.K., Nowrouzezahrai, D., Snyder, J.: Image-based proxy accumulation for real-time soft global illumination. In: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications, pp. 97–105 (2007)
22. Soler, C., Hoel, O., Rochet, F.: A deferred shading algorithm for real-time indirect illumination. In: ACM SIGGRAPH 2010 Talks, p. 18 (2010)
23. Szirmay-Kalos, L., Umenhoffer, T., Tóth, B., Szécsi, L., Sbert, M.: Volumetric ambient occlusion for real-time rendering and games. IEEE Computer Graphics and Applications **30**(1), 70–79 (2010)
24. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing **13**(4), 600–612 (2004)
25. Zhou, K., Hu, Y., Lin, S., Guo, B., Shum, H.Y.: Precomputed shadow fields for dynamic scenes. In: Proceedings of ACM SIGGRAPH 2005, pp. 1196–1201 (2005)