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

# Efficient Screen-Space Approach to High-Quality Multi-Scale 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 performance and quality. Our method computes ambient occlusion (AO) values at multiple image resolutions and combines them to obtain the final, high-resolution AO value for each image pixel. It produces high-quality AO that includes both high-frequency shadows due to nearby, occluding geometry and low-frequency shadows due to distant geometry. Our approach only needs to use very small sampling kernels at every resolution, thereby achieving high performance without resorting to random sampling. As a consequence, our results do not suffer from noise and excessive blur, which are common of other SSAO methods. Therefore, our method also avoid the expensive, final blur pass commonly used in other SSAO methods. The use of multiple resolutions also helps reduce errors that are caused by SSAO's inherent lack of visibility checking. Temporal incoherence caused by using coarse resolutions is solved with an optional temporal filtering pass. Our method produces results that are closer to ray-traced solutions than those of any existing SSAO method's, while running at similar or higher frame rates than the fastest ones.

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

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

## 1 Introduction

*Ambient occlusion* (AO) is the shadowing effect under the direct illumination of a uniform diffuse spherical light source surrounding the scene. Due to the occlusion of the incoming radiance from this light source, concave areas such as creases or holes will appear darker than exposed areas. AO is not a real-world phenomenon since in reality, incoming light is rarely equal in all directions and light inter-reflections occur between surfaces. Despite being artificial, 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). That is why AO is often used as a cheap alternative to more expensive global illumination solutions which also take into account light inter-reflections. In fact, AO is a standard technique used in CG movies and TV productions, which rely on it to make the rendered images more realistic. In computer games, real-time AO is also becoming more popular as commodity graphics hardware becomes more powerful.

Several methods to compute AO exist and they differ in the extent to which accuracy is traded for speed. The most accurate methods often use Monte Carlo ray tracing (see, for example, [15]), which means they are slow and are only suitable for offline pre-computations. AO results from such methods can be used for either offline or real-time rendering. However, using pre-computed AO results for real-time rendering imposes the limitation that the scene geometry must be static. Real-time AO computation, which often imposes fewer restrictions on the scene, has just recently become possible, mainly due to the advancement in programmable graphics hardware. It is useful for realistic real-time rendering of fully dynamic scenes, where AO cannot be pre-computed. A class of real-time methods, collectively known as *screen-space ambient occlusion* (SSAO), trades AO accuracy for significant increase in performance. Compared to other existing AO methods, SSAO works on all types of scenes, is significantly faster, and simpler to implement and integrate into existing rendering systems. In return, SSAO is far from accurate and also suffers from various quality issues. Due to its speed and simplicity, SSAO has become increasingly popular in 3D games and other interactive applications that favor speed over accuracy and quality. SSAO implementations often 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 difficulties in producing local AO effects, such as darkening of small creases, but are facing great performance challenges in producing more global AO effects, which are crucial for scene realism. The main reason is that SSAO relies on screen-space samples to compute AO. To produce global AO, a large number of samples must be taken in real-time, which quickly degrades performance. Most SSAO methods try to get around

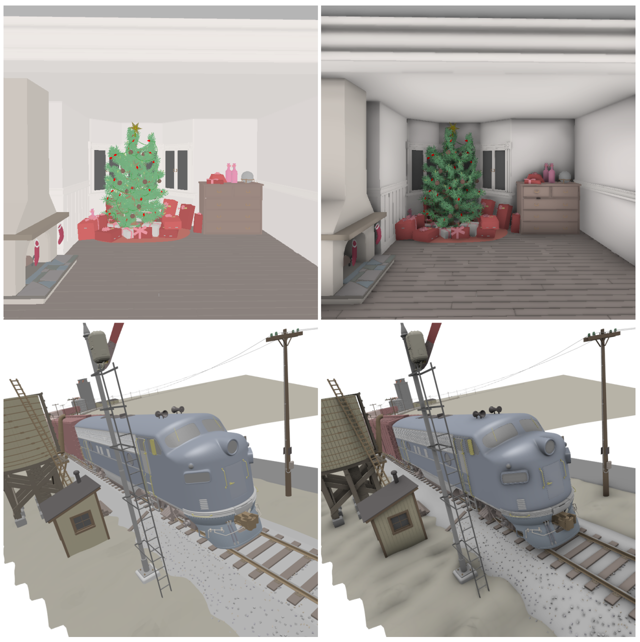

**Fig. 1** (Left) Two scenes rendered without AO. (Right) The same scenes rendered with AO produced by our method.

the problem by using random sampling, but the results either are 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 by combining partial AO values at multiple image resolutions. Our method can capture both sharp shadows from local occluders and smooth shadows from distant ones. It is based on the principle that AO due to faraway occluders are low-frequency, thus can be computed at coarser resolutions, whereas AO due to nearby occluders are high-frequency, and thus must be computed at finer resolutions. With this observation, we can use very small sampling kernels at each resolution and achieve high performance without random sampling. Moreover, by retaining the maximum AO value across all resolutions, we could compensate for the lack of visibility checking to some extent, and thus our results are more accurate than most other SSAO methods. In this paper, it is compared with three state-of-the-art SSAO methods, and is shown to produce much higher-quality results, while achieving performance comparable to the fastest one.

## 2 Ambient Occlusion Theory

This section discusses the theoretical basis of AO. We assume the reader is familiar with fundamental radiometry concepts such as solid angles, radiance, and irradiance. For a quick review of radiometry, we refer the reader to [5]. The definition of AO requires that the scene is being lit by only a uniformly diffuse, spherical light source surrounding it. In such a lighting environment, incoming radiance directly from the light source is constant for all incoming directions. For simplicity, we also assume all the surfaces are Lambertian, that is, they reflect incoming light equally in all directions. With all these assumptions, the equation for surface irradiance at a point $\mathbf{p}$ with normal $\mathbf{n}$ is

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

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

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

where

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

$k_A$ is defined to be the ambient occlusion value of point $\mathbf{p}$, and its value ranges from 0 to 1. When $k_A$ is 0, $\mathbf{p}$ is totally blocked from light; when it is 1, $\mathbf{p}$ is totally exposed. Although $k_A$ is called "ambient occlusion", it actually corresponds to how much of the hemisphere above $\mathbf{p}$ is visible, or its "accessibility". To avoid confusion, in this paper, we use the term AO to actually mean $1-k_A$, so a higher AO value means a lower intensity (or darker color).

The above definition of AO is not useful for enclosed scenes, where everything would be totally dark, since $v(\mathbf{p},\omega)$ equals 0 everywhere. That is why in practice, the binary visibility function $v(\mathbf{p},\omega)$ is often replaced by an attenuation (or falloff) function $\rho(\mathbf{p},d)$ which varies smoothly from 1 to 0 as $d$ increases [39]. With $\rho$, we can rewrite $k_A$ as

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

$\rho(\mathbf{p},d)$ is a continuous function that depends on the distance $d$ between $\mathbf{p}$ and the point where a ray cast from $\mathbf{p}$ in direction $\omega$ intersects some nearby geometry. As $d$ increases from 0 to some preset value $d_{max}$, $\rho(\mathbf{p},d)$ decreases monotonically from 1 to 0. Although the use of $\rho$ is empirical, it is more useful than a binary visibility function. In our method, Equation 4 with a quadratic falloff function are used as the basis for all AO computations. Technically, the definition of AO that uses a falloff function is called *Ambient obscurance* [39] to differentiate it from the old model of AO, but since the terms are used interchangeably in previous work, we use AO (which is the more established term) to mean ambient obscurance in this paper.

## 3 Related Work

Here we briefly discuss existing AO methods that are targeted for real-time or interactive rendering of 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 [9] (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. [27] 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. [28] and Sloan et al. [33] 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. [31] 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. [12] and Malmer et al. [17] 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. [38] 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 [19] 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. [14] 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 [10], but the latter only works for self shadowing objects. Analytical methods are slow, especially for big scenes with many triangles. These methods are thus not yet suitable for real-time applications.

### 3.2 Screen-Space Methods

Screen-space methods use empirical models that darken a pixel by treating its nearby pixels as potential occluders. Mittring [21] 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 [6] (attenuation function, no self-occlusion), [29] (directional occlusion, one-bounce indirect illumination), [16] (fast, low-variant sampling method called line sampling), and [36] (similar to [16]). McGuire et al. [20] uses a sampling technique similar to [16] but with a more robust AO formula to avoid black halos; their method also allows for more artistic control.

Shanmugam et al. [31] and Fox et al. [7] 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. Compared to the approach of projecting eye-space samples, sampling directly in screen space produce results that are less noisy, but also more biased, as screen-space directions do not necessarily correspond to 3D directions. Our method uses screen-space sampling since we have found that the trade-off is worth taking, knowing that SSAO's results in general are already quite biased (compared to ray-traced results, for example), and the real problem with it is the noise.

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 screen-space 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 differ mostly in their reformulations of the AO equation (Equation 4), and as a result, their sampling methods. Samples can be taken in a two-dimensional or three-dimensional space. Some methods take samples in three-dimensional spaces, resulting in high-variance approximations and noise [6, 21, 29]. Other methods take samples in two-dimensional spaces, whether that space is the screen-space itself [2, 7, 31], or some 2D subspace of the object space [16,20,36]. Fig. 2 show the various sampling schemes used in SSAO. Generally, the trend is shifting towards 2D sampling, because doing so produces low-noise results. One drawback is that 2D sampling patterns are more biased, but given that occlusion checking based on only the depth buffer is already inaccurate, biased sampling is not too big a prob-

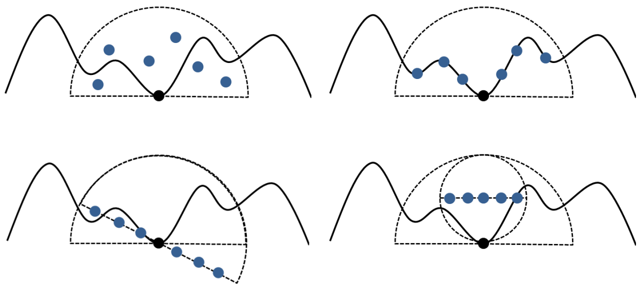

**Fig. 2** (Upper-left) The sampling scheme used in [6]. (Upper-right) The sampling scheme used in [31] and our method. (Lower-left) The sampling scheme used in [16]. (Lower-right) The sampling scheme used in [36].

lem as it may seem. Our method uses a sampling scheme similar to that of [31], but the actual patterns are more involved since we use multiple resolutions.

In general, screen-space methods are fast, but suffer from numerous 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. It can also crudely approximate visibility in a cheap way. For the other problems, there are some proposed solutions, such as depth peeling [29], 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.

### 3.3 Multi-Resolution Techniques

Observing that AO is mostly low-frequency, Sloan et al. [33] compute AO in a coarser resolution and upsample it using joint bilateral upsampling [13]. 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 (Fig. 3 shows that there are AO frequencies not captured using just two resolutions). Multi-resolution techniques are also proposed in [23–25] and [35], but for the purpose of computing one-bounce indirect illumination without visibility checking.

As capturing AO at different frequencies using different resolutions is natural, MSSAO can achieve better rendering speed and yet retain the quality of the results. As far as our knowledge extends, MSSAO is the first truly multi-resolution technique (more than two resolutions) that addresses the real-time AO problem concerning both performance and quality improvements.

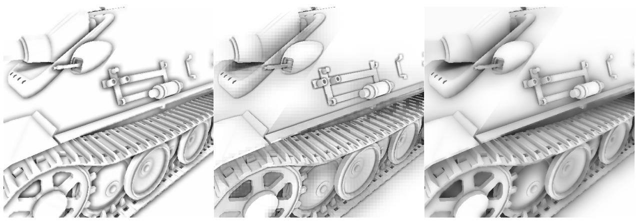

**Fig. 3** (Left) Result produced by using only two consecutive levels of resolution: 512x512 and 256x256 pixels. (Middle) Result produced by using only two very different levels of resolution: 64x64 and 512x512 pixels. (Right) Result obtained by using four levels of resolution: 64x64, 128x128, 256x256, and 512x512 pixels. The blocky artifacts in the middle image is not a major problem as it could be reduced with enough filtering. The most important thing to notice in this figure is that the left and the middle images both miss some of the shadow frequencies by using only two resolutions.

### 3.4 Temporal Coherence Methods

Methods that exploit temporal coherence are ones that reuse shading information from a previous frame and combine it with that of the current frame to either reduce temporal aliasing, improve image quality, or improve performance of a shading algorithm. The main idea behind these methods is that between consecutive frames, the shading information changes very little, so a large portion of it can be reused. A particular temporal coherence approach is image-space, real-time reverse reprojection [22, 30], in which a buffer is used to store previous frame's data, and current-frame's pixels are projected into that buffer to retrieve old data. That buffer is often called real-time the reprojection cache, or the history buffer. In the context of SSAO, reverse reprojection has been used in [18] for quality improvement purpose and in [34] for temporal aliasing reduction purpose. In our case, reverse projection is used as a filtering method to reduce flickering between consecutive frames.

## 4 Multi-Resolution SSAO Algorithm

Given an AO radius of influence $d_{max}$, we can define a hemisphere with radius $d_{max}$ centering at each shaded point $\mathbf{p}$. This hemisphere is on the tangent plane defined by $\mathbf{p}$'s normal, $\mathbf{n}$. At this point, it is important to be reminded that the discussions in this paper often refer to a shaded pixel as $p$ and its corresponding eye-space "point" as $\mathbf{p}$. Conceptually, our algorithm partitions the hemisphere with radius $d_{max}$ above $\mathbf{p}$ into a set of nested hemispheres. Each hemisphere is contained inside an outer, bigger hemisphere (Fig. 4). For each hemisphere, we compute an occlusion value for $\mathbf{p}$ due to occluders inside that hemisphere. We combine occlusion values using the maximum value across all hemispheres. The intuition here is that the combined AO value should obviously be more than or equal to the maximum partial AO value in some particular hemisphere. Keeping the

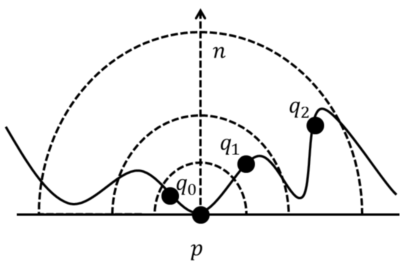

**Fig. 4** The hemisphere above a shaded point $\mathbf{p}$ is partitioned into several nested hemispheres. $\mathbf{q_0}, \mathbf{q_1}, \mathbf{q_2}$ represents occluders inside different hemispheres.

maximum value also serves as a crude visibility checking mechanism, as it prevents the lack of faraway occluders to dilute the occlusion due to nearby ones.

To speed up the computation, we notice that occlusion caused by occluders faraway from $\mathbf{p}$ need not be computed at the finest resolution. Thus, we relate each hemisphere to a resolution where bigger hemispheres correspond to coarser resolutions. For each hemisphere, we compute a partial AO value at its corresponding resolution. When computing AO for a hemisphere, samples must be taken inside the whole hemisphere, not just in the “difference” between two adjacent hemispheres. Computing AO for bigger hemispheres at coarser resolutions, however, introduces errors as pixels too close to $\mathbf{p}$ can be grouped together with $\mathbf{p}$ itself at coarser resolutions, causing the low-resolution AO values to miss contributions from nearby occluders. Therefore, besides retaining the maximum partial AO value, we also modulate it using the average AO value, avoiding the under-occlusion problem.

Our method is conceptually different from another idea that may seem similar at first, that is to distribute samples so that faraway occluders are sampled in coarser resolutions, effectively using fewer samples as the distance from the shaded point $\mathbf{p}$ increases. The latter approach has a major drawback, that is the AO values contributed by faraway occluders needed to be scaled (weighted) more, resulting in blocky artifacts. That problem can be solved with excessive low-pass filters, making the final results blurry. This is the approach used by [35] to compute screen-space, one-bounce color bleeding, and it is evident in their results that the indirect illumination is unusually low-frequency and the rendered images look too blurry.

Our algorithm uses a deferred shading framework with a typical g-buffer that stores eye-space positions and normals for each pixel. We refer the reader to [32] for a review of the deferred shading technique using g-buffer. For each shaded pixel $p$, we treat the eye-space points corresponding to its nearby pixels as occluders that may block light from reaching $\mathbf{p}$. Those occluders are sampled directly from the g-buffer. For occluders further away from $\mathbf{p}$, we can sample them at a coarser resolution than for occluders nearer to $\mathbf{p}$. We assume that pixels close to $p$ in screen space correspond to nearby geometry in object space. This is not always

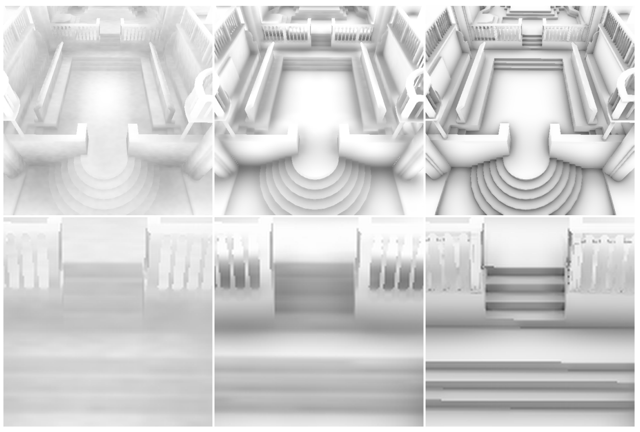

**Fig. 5** (Left) Noise artifacts by Blizzard’s method, (middle) blurry result by NVIDIA’s HBAO, (right) result from our method. Our method produces no noise and preserves sharp edges and fine details better.

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. 5).

## 4.1 Overview

Our algorithm first renders the scene’s geometry to a g-buffer at the finest resolution. The g-buffer stores per-pixel, eye-space coordinates and (normalized) normals. It is then downsampled multiple times, similar to creating a mipmap. Then, we compute an AO value for each pixel at each resolution, by sampling other pixels in its small 2D neighborhood. Finally, for each pixel at the finest resolution, its final AO value is computed by combining corresponding AO values across the resolutions. A naïve implementation that follows the idea above exactly would produce blocky artifacts, since adjacent pixels at a resolution are grouped together at coarser resolutions, thus sharing the same coarse AO values. To achieve smooth results, in our implementation, AO is calculated from the coarsest resolution to the finest one using multiple rendering passes. In each pass, the AO value calculated by sampling for each pixel is combined with an AO value upsampled from the previous, coarser resolution. We use bilateral upsampling [13] to avoid upsampling across large depth and normal differences. Also, to achieve cleaner results, we apply a low-pass bilateral filter to the coarser AO values right before the upsampling step. The overall algorithm is described in Table 4.1.

```
Render the scene into a g-buffer at (the finest) resolution $Res_1$;
for $i = 2$ to $n$ do
    Downsample the g-buffer from $Res_{i-1}$ to $Res_i$;
end for
for $i = n$ to 1 do
    for all pixel $p$ at resolution $Res_i$ do
        Sample $p$'s neighborhood to compute an AO value $AO_{near}$;
        if $i = n$ then
            $AO_{combined} \leftarrow AO_{near}$;
            Output $AO_{combined}$ as input to the next iteration;
        else
            Filter the result ($AO_{combined}$) from $Res_{i+1}$;
            Upsample filtered result to get $AO_{far}$;
            Combine $AO_{near}$ and $AO_{far}$ to obtain $AO_{combined}$;
            if $i > 1$ then
                Output $AO_{combined}$ as input to the next iteration;
            else
                Output $AO_{combined}$ as the final result;
            end if
        end if
    end for
end for
```

**Table 1** The overall algorithm.

Note that $AO_{near}$, $AO_{far}$, and $AO_{combined}$ are per-pixel, per-resolution values. In our algorithm, $AO_{near}$ represents AO caused by neighboring pixels and $AO_{far}$ represents upsampled AO caused by farther-away pixels. They are combined in every rendering pass (except the first) to obtain $AO_{combined}$. For example, suppose $p$ is some pixel in some current resolution during the algorithm. In the next rendering pass (at a finer resolution), the previously computed value $AO_{combined}$ of $p$ will be upsampled and treated as $AO_{far}$ for the higher-resolution pixels near $p$ (see the details in Section 4.5). Note that each $AO_{near}$ value is computed independently for all resolutions by sampling in a small screen-space neighborhood of the shaded pixel. The whole process ends when the finest resolution is reached, at which point $AO_{combined}$ 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 coarser-resolution one with each rendering pass. In each pass, every pixel will combine the eye-space coordinates and normals of its four corresponding sub-pixels at the previous, finer resolution. The total number of resolutions depends on how "global" we want the AO to be, which is often specified by a real number called the AO radius of influence. In practice we have found that using four or five resolutions balances between achieving a fairly far-reaching AO and avoiding artifacts caused by the lack of resolution in coarse buffers. Using more levels also gives few benefits since AO becomes more and more low-frequency and low-contrast at coarser resolutions.

```
for $i = 2$ to $n$ do
    for all pixel $p$ at resolution $Res_i$ do
        Obtain four corresponding sub-pixels from $Res_{i-1}$;
        Sort them so that $\mathbf{p}_0^z < \mathbf{p}_1^z < \mathbf{p}_2^z < \mathbf{p}_3^z$;
        if $\mathbf{p}_3^z - \mathbf{p}_0^z \le d_{threshold}$ then
            $\mathbf{p} \leftarrow (\mathbf{p}_1 + \mathbf{p}_2)/2$;
            $\mathbf{n} \leftarrow (\mathbf{n}_1 + \mathbf{n}_2)/2$;
        else
            $\mathbf{p} \leftarrow \mathbf{p}_1$;
            $\mathbf{n} \leftarrow \mathbf{n}_1$;
        end if
    end for
end for
```

**Table 2** The downsampling algorithm.

The most common way of combining high-resolution values is to average them. We have decided to use a more stable method, that is to keep the median value instead of the mean. We sort the four sub-pixels according to their eye-space depth values, pick the two pixels whose depth values are in the middle, and take the average of their eye-space coordinates. It is well known that median is more stable than mean, that is, on average, the median of a set of numbers fluctuates less than the mean when the numbers themselves vary. Therefore, using the median helps our method achieve better temporal coherence. However, we have found that using only the median has a significant drawback, as it does not preserve relative distances among eye-space points. As a result, points that are further away at finer resolutions become closer at coarser resolutions. That results in artifacts when occluders sometimes cast shadows on receivers outside their AO radius of influence. To avoid this situation, instead of the median, we keep one of the four sub-pixels' eye-space coordinates, but only when the maximum distance among the four sub-pixels' eye-space coordinates is large enough (larger than some preset value $d_{threshold}$). We sort the four corresponding z-values and pick the sub-pixel with the second smallest absolute z-value. The reason this downsampling scheme is not used in every case is that it is not as stable as keeping the median, and would create heavy shimmering or flickering artifacts during animations or camera movements.

Pixels' normals are downsampled similarly to eye-space coordinates. It is important to note that at the finest resolution, we use true polygons' normals, not interpolated vertex normals, as the latter would cause self-occlusion artifacts with sparsely-tessellated scene models. The downsampling algorithm is summarized in Table 2, where we use $\mathbf{p}_i^z$ to denote the eye-space depth value of point $\mathbf{p}_i$.

## 4.3 Occluder Sampling

To compute $AO_{near}$ for a shaded pixel $p$ at each resolution, we sample a small screen-space neighborhood around $p$.

The sampling kernel size is decided as follows. First, we set an AO radius of influence in eye space, $d_{max}$, denoting the distance to the shaded point beyond which an occluder contributes no occlusion at all. This distance is then projected into screen space at the finest resolution $Res_1$ so that we have a kernel radius $r_1(p)$ (in terms of number of pixels) for each pixel $p$. The maximum kernel radius for $p$ at the next coarser resolution should naturally be $r_2(p) = r_1(p)/2$, and the next should be $r_3(p) = r_1(p)/4$, and so on. Note that the kernel size depends on the depth of each individual pixel because of perspective fore-shortening, which cause pixels nearer to the camera to have larger kernel sizes in screen space. Since finer resolutions corresponds to smaller hemispheres, we do not use $r_i(p)$ directly as the kernel radius at resolution $Res_i$. Instead, we cap the radius to $r_{max}$ pixels at any resolution. Another reason for capping the kernel size is because the values $r_i(p)$ can be quite large when $p$ becomes closer to the camera (when the scene is zoomed in). In such situations, performance can drop drastically if we do not cap the kernels to some fixed size. A typical value of $r_{max}$ used in our implementation is 5, which corresponds to a 11x11-pixel kernel. In short, for each shaded pixel $p$ at resolution $Res_i$, the sampling kernel $R_i(p)$ is calculated using the following formulas

$$\begin{aligned} r_0(p) &= \frac{sd_{max}}{2z\tan(\alpha/2)} \\ r_i(p) &= r_0(p)/2^i \\ R_i(p) &= min(r_{max}, r_i(p)) \end{aligned} \tag{5}$$

where $s$ is the viewport's dimension at the finest resolution (assuming we have a square viewport), $\alpha$ is the camera's field-of-view angle, and $z$ is the eye-space depth value of pixel $p$.

Not every pixel in a 11x11-pixel kernel is sampled; instead we only sample every other pixel, effectively reducing the number of texel fetches from 121 to 36 per fragment (Fig. 6). This sampling pattern is similar to interleaved sampling [11], but without the randomness; it can take advantage of the followed-up blur pass that combines AO values in a 3x3-pixel neighborhood. Sometimes, when a more "global" AO is desired resulting in large kernel radii, sampling every other pixel can be slow. In that case, one can fix the number of samples and use a low-variant random sampling pattern such as Poisson disks. In fact, we sample using a 16-point Poisson disk for pixels at the finest resolution. No final blur pass is needed at this resolution since the kernel size is small. Finally, it is worth to note that Equation 5 implies that a kernel radius can be smaller than 1. In other words, for some pixels at a particular resolution, we need not take any samples if the corresponding kernel radius is smaller than the width of a pixel at that resolution.

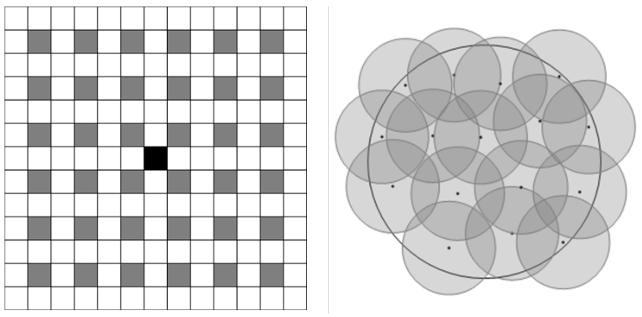
**Fig. 6** (Left) At all resolutions except the finest, we sample using an interleaved pattern that works well with a 3x3 low-pass filter afterwards. (Right) The 16-point Poisson disk pattern used at the finest resolution.

## 4.4 Computing Ambient Occlusion

For each pixel $p$ at a particular resolution, the next step after gathering $N$ samples is to use them to compute $AO_{near}$. We use the following formula

$$AO_{near}(\mathbf{p}) = \frac{1}{N}\sum_{i=1}^{N} \rho(\mathbf{p}, d_i)\overline{\mathbf{n} \cdot \widehat{\mathbf{q}_i - \mathbf{p}}}, \tag{6}$$

where $d_i$ is the eye-space distance between the $i$th occluder ($\mathbf{q}_i$) and $\mathbf{p}$, and $\theta_i$ is the angle between the $\mathbf{p}$'s normal and the vector joining $\mathbf{p}$ and $\mathbf{q}_i$. The bar over the cosine term means its value is clamped to $[0,1]$. $\rho(\mathbf{p}, d_i)$ is a falloff function that smoothly attenuates an occluder's contribution as it goes further away from $\mathbf{p}$. The falloff function $\rho(\mathbf{p}, d_i)$ 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 $\rho$:

$$\rho(\mathbf{p}, d_i) = 1 - min(1, (\frac{d_i}{d_{max}})^2). \tag{7}$$

We choose this quadratic falloff function because each occluder can be considered a very small sphere, and the solid angle subtended by that sphere varies inversely to its squared distance from $\mathbf{p}$.

If nearby pixels correspond to uniform directions in the tangent hemisphere above $\mathbf{p}$, Equation 6 comes close to a Monte Carlo approximation of Equation 3. This is rarely true in practice, so our AO results are often biased compared to, for example, ray-traced AO. However, the AO computed by our formula has low variance, and thus we need fewer samples to produce noise-free results compared to, for example, Crytek's [21] or Blizzard's [6] formulas (see Fig. 7). The main reason is that in those methods, the distribution of sample points takes into account both direction ($\theta_i$) and distance ($d_i$), so sampling is done in three-dimensional spaces. Whereas in our method, sample points are only distributed in a two-dimensional space (screen space), and the distance from a sample to the shaded point is part of the sample itself. Other recent SSAO methods have also exploited this

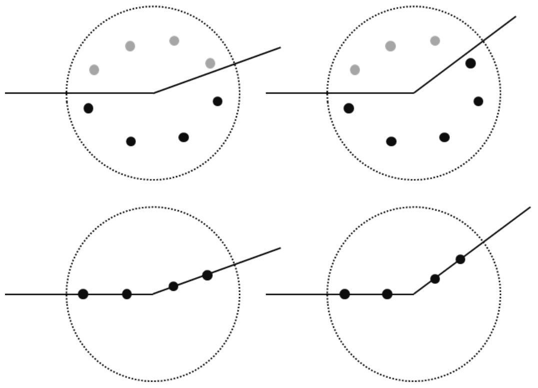

**Fig. 7** (Top) Crytek's method gives unstable occlusion value (from $4/8$ to $3/8$) with just a slight change in geometry. (Bottom) our cosine function varies smoothly as the geometry changes. As a result, we need very few samples for each shaded pixel. In this example, with 4 samples we are able to evaluate AO more accurately than Crytek's 8 samples. On the left, Crytek's method will give an occlusion value of $4/8$ or $1/2$ which corresponds (wrongly) to a totally flat surface according to their interpretation of AO.

```
for $i = n$ to 1 do
    for all pixel $p$ at resolution $Res_i$ do
        Calculate kernel size $R_i(p)$ using Equation 5;
        if $i \neq 1$ then
            Sample the neighborhood using Fig. 6 (left);
        else
            Sample the neighborhood using a Fig. 6 (right);
        end if
        Calculate $AO_{near}(\mathbf{p})$ using Equation 6;
    end for
end for
```

**Table 3** The sampling and computing AO algorithm.

idea of dimension-reduction to avoid noise [2,16,36]. Compare to those methods, our formula is cheaper to compute. An important thing to note is that we do not disregard non-blocking samples, such as those below the shaded point's hemisphere, since it is well-known that rejection sampling gives high-variance results [8]. Finally, Table 3 summarizes both the sampling and computing AO steps, since they are closely related.

## 4.5 Combining Occlusion Values

Now that we have computed $AO_{near}$, it must be combined with $AO_{far}$, which is the AO upsampled from a previous, coarser resolution. To compute $AO_{far}$, we use a bilateral upsampling algorithm which can avoid blurring across large depth and normal discontinuities. The upsampling blends AO values from the four low-resolution pixels that are closest to the current-resolution pixel $p$ (see Fig. 9). The weight used by the bilateral upsampling algorithm is a product of bilinear, depth, and normal differences. More specifically, with regards to 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,

$$w_{i,n} = (\frac{\mathbf{n} \cdot \mathbf{n}_i + 1}{2})^{t_n}, \tag{8}$$

and

$$w_{i,z} = (\frac{1}{1 + |z_i - z|})^{t_z}. \tag{9}$$

The powers $t_n$ and $t_z$ are to be interpreted as "tolerance" levels, and are dependent on the scene's size. If the scene is small, nearby pixels correspond to nearby eye-space "points", thus large depth differences must be tolerated less by using a larger power value. The aim is to balance between smooth interpolation of AO 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. 8 clearly demonstrates the superior quality of bilateral upsampling over the other upsampling methods.

After upsampling, we have the "distant" AO value $AO_{far}$. Now it must be combined with the "nearby" AO value $AO_{near}$. Since both the maximum AO and the average AO values across the resolutions are needed, we must combine $AO_{far}$ and $AO_{near}$ in different ways. Up until this point, $AO_{near}$, $AO_{far}$, and $AO_{combined}$ each is viewed as a single value. In fact, $AO_{near}$ is a tuple of two values, while $AO_{far}$ and $AO_{combined}$ each is a tuple of three values. Let us use $AO^{[j]}_{near}$ to mean the $j$th element of $AO_{near}$. We define $AO^{[1]}_{near}$ to be the value computed using Equation 6, $AO^{[2]}_{near}$ is $N$, the number of samples, also from the same Equation. For $AO_{combined}$, its element are computed using:

$$\begin{aligned} AO^{[1]}_{combined} &= max(AO^{[1]}_{near}/AO^{[2]}_{near}, AO^{[1]}_{far}), \\ AO^{[2]}_{combined} &= AO^{[1]}_{near} + AO^{[2]}_{far}, \\ AO^{[3]}_{combined} &= AO^{[2]}_{near} + AO^{[3]}_{far}. \end{aligned} \tag{10}$$

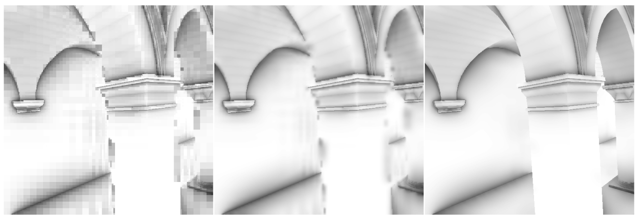

**Fig. 8** (Left) Nearest neighbor upsampling, which is blocky, (middle) bilinear upsampling, which leaks occlusion, (right) bilateral upsampling, which is free of visible artifacts.

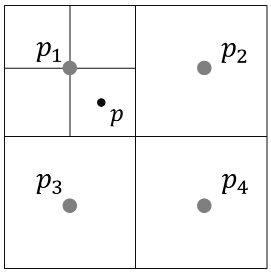

**Fig. 9** In the upsampling step, each current-resolution pixel $p$ will blend occlusion values from its four nearest lower-resolution pixels $p_1$, $p_2$, $p_3$, and $p_4$. The bilinear weights $w_b$ for $p_1$, $p_2$, $p_3$, and $p_4$ are $\frac{9}{16}$, $\frac{3}{16}$, $\frac{3}{16}$, and $\frac{1}{16}$ respectively.

As for $AO_{far}$, it is just an element-wise upsampled from $AO_{combine}$. The first element of $AO_{far}$ keeps track of the current maximum AO value, the second stores the unnormalized sum of all AO values, and the last keeps track of the total number of samples across all resolutions processed so far. We need to store the unnormalized sum and the total number of samples separately instead of just dividing the former by the latter and store one normalized AO value. The reason is that there maybe no samples at some particular resolutions, making it impossible to recover the correct average AO value by the time we reach the finest resolution if only the normalized value was kept. At the finest resolution, we combine both the maximum and average values using the formulas

$$\begin{aligned} AO_{max} &= max(AO_{near}^{[1]}/AO_{near}^{[2]}, AO_{far}^{[1]}), \\ AO_{average} &= (AO_{far}^{[2]} + AO_{near}^{[1]})/(AO_{far}^{[3]} + AO_{near}^{[2]}), \\ AO_{final} &= 1-(1-AO_{max})(1-AO_{average}), \end{aligned} \tag{11}$$

and just output $1-AO_{final}$ as the "accessibility" value for the shaded pixel. This value can be used directly to shade the pixel (like what is done in almost all examples shown in this paper), or used to modulate the pixel's ambient or diffuse intensity calculated using traditional shading methods such as Phong's.

In general, the use of the maximum operator to combine AO values from multiple resolutions prevents inner occluders from being multiply-counted, and it also allows occlusion missed at any resolution to be picked up at 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. Using only $AO_{max}$, however, is not enough, as a shaded point is often blocked by occluders in more than one resolution, thus the final AO value often is often bigger than $AO_{max}$. To account for this effect, we use the average AO value to modulate the maximum value in a way that ensures $0 \leq AO_{max} \leq AO_{final} \leq 1$ (Equation 11). Fig. 10 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.

Since we retain the maximum AO value in each rendering pass, the final output may not look "clean". That is because neighboring pixels may have different maximum AO values computed at different resolutions. That problem can be solved by adding a low-pass filter just before the upsampling. This blur is also needed to properly "combined" the AO values computed by sampling using an interleaved pattern. It allows us to sample sparsely at coarse resolutions and at the same time smooth out the final result. We use bilateral

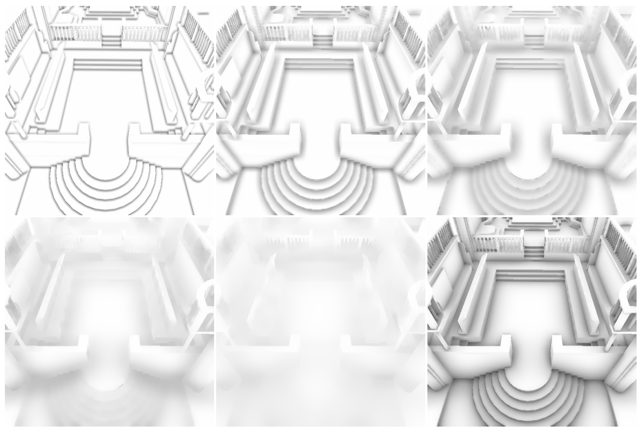
**Fig. 10** Combining AO values across multiple resolutions to get the final AO value. The first 5 images are AO values computed at 5 different resolutions, from the finest to the coarsest. The last image (bottom-rightmost) is the final result. Note that the images shown here are outputs after blurring and upsampling.

```
for $i = n-1$ to 1 do
  for all pixel $q$ with coordinate $\mathbf{t}$ at resolution $Res_{i+1}$ do
    for all pixel $q_j$ with coordinate $\mathbf{t}_j$ in a 3x3 neighborhood around $q$ do
      Compute screen-space distances $dx_j = |\mathbf{t}^x - \mathbf{t}_j^x|$ and $dy_j = |\mathbf{t}^y - \mathbf{t}_j^y|$;
      Compute a Gaussian weight using $w_{j,g} = 1/((dx_j+1)(dy_j+1))$;
    end for
    Sum the weighted AO values using $AO_{combined}(q) \leftarrow \sum_{j=1}^{9} w_{j,g} AO_{combined}(q_j)$;
  end for
  for all pixel $p$ at resolution $Res_i$ do
    Get the four pixels nearest to $p$ at resolution $Res_{i+1}$;
    for all low-resolution pixel $p_j (1 \leq j \leq 4)$ do
      Obtain a bilinear weight $w_{j,b}$ using Fig. 9;
      Compute a depth weight $w_{j,z}$ using Equation 9;
      Compute a normal weight $w_{j,n}$ using Equation 8;
      Compute a bilateral weight using $w_j = w_{j,b} w_{j,z} w_{j,n}$;
    end for
    Blend the four low-resolution AO values using $AO_{far}(p) = \sum_{j=1}^{4} w_j AO_{combined}(q_j)$
  end for
end for
```
**Table 4** The filtering and upsampling algorithms

filtering [26] which is similar to the bilateral upsampling algorithm, but with the bilinear weight replaced by a Gaussian one. The other differences are that it is done at the same resolution (instead of cross-resolution), and uses a slightly larger kernel (3x3 pixels instead of 2x2 pixels). The bilateral blurring and upsampling steps are summarized together in Table 4.

## 4.6 Temporal Filtering

The use of low-resolution g-buffers has an inherent limitation: their contents are much less coherent between consecutive frames. When the camera moves, high-resolution pixels can be “snapped” to different low-resolution pixels, creating sharp changes in low-resolution g-buffers. The results are shimmering, or flickering artifacts happening during animations or camera movements. Because of the way our down-sampling algorithm works, these effects happen mostly at the scene objects’ edges. For most scenes, the flickering is not too noticeable. Nevertheless, we can use a final rendering pass to eliminate all remaining temporal artifacts. To enhance the temporal coherence of MSSAO, we employ a technique called temporal reprojection, similar to what has been done in [22, 30]. For each pixel in the current frame, we reproject it into the screen space of the previous frame using two sets of view and projection matrices as follows

$$\begin{aligned} \mathbf{p} &= \mathbf{V}_{curr}^{-1}\mathbf{P}_{curr}^{-1}\mathbf{t}_{curr}, \\ \mathbf{t}_{prev} &= \mathbf{P}_{prev}\mathbf{V}_{prev}\mathbf{p}. \end{aligned} \tag{12}$$

$\mathbf{t}_{prev}$ and $\mathbf{t}_{curr}$ are the screen-space coordinates of the same eye-space “point” $\mathbf{p}$ in both the current and previous frames. $\mathbf{P}$ and $\mathbf{V}$ are the projection and view matrices accordingly. This equation is only applicable for static scenes where the geometry does not change. For dynamic scenes, one can store the optical flow $\mathbf{p}_{curr} - \mathbf{p}_{prev}$ in the frame buffer and use it to find the correct past-frame position of $\mathbf{p}$ before applying the view ($\mathbf{V}_{prev}$) and projection ($\mathbf{P}_{prev}$) transformations [18]. One difficulty arises when the point $\mathbf{p}$ is occluded from view by some other geometry, or outside the view frustum in the last frame. To check for occlusion, we sample the old frame buffer using $\mathbf{t}_{prev}$ to get the last-frame linear depth value ($z_{prev}$) at that screen-space position. If $z_{prev}$ and $\mathbf{p}^z$ are close enough, that is if

$$|1 - \frac{z_{prev}}{\mathbf{p}^z}| < \varepsilon, \tag{13}$$

then $\mathbf{p}$ is visible to the camera in the last frame. In that case, we sample the last-frame’s AO buffer using $\mathbf{t}_{prev}$ to obtain the current pixel’s old AO value $AO_{prev}$. Let $AO$ be the AO value computed using Equation 11, we combine it with $AO_{prev}$ using linear interpolation to obtain the final AO value for the current frame $AO_{curr}$ as follows

$$AO_{curr} = kAO_{prev} + (1-k)AO. \tag{14}$$

$k$ is a preset weight between 0 and 1. A higher $k$ means a more stable AO; in practice, that often results in a motion blur effect when the camera moves. A good value for $k$ will eliminate temporal aliasing, while at the same time minimizes motion-blurring. In our implementation, $k = 0.5$ is typically used. The temporal filtering algorithm is described by the pseudo-code in Table 5.

```
for all current-frame pixel $p$ with screen-space coordinate $\mathbf{t}_{curr}$ do
    Compute the corresponding eye-space point $\mathbf{p}$ using Equation 12;
    Project $\mathbf{p}$ to the previous frame's screen space (to obtain $\mathbf{t}_{prev}$) using Equation 12;
    if $\mathbf{t}_{prev}$ is outside the previous frame's g-buffer then
        $AO_{curr} \leftarrow AO$;
    else
        Obtain a linear depth value $z_{prev}$ at position $\mathbf{t}_{prev}$ in the previous frame's g-buffer;
        if the condition at (13) is true then
            $k \leftarrow 0.5$;
        else
            $k \leftarrow 0.0$;
        end if
        $AO_{curr} \leftarrow kAO_{prev} + (1-k)AO$;
    end if
end for
```

**Table 5** The temporal filtering algorithm.

## 5 Evaluations and Comparisons

We compare our method (MSSAO) with Blizzard’s SSAO [6], NVIDIA’s HBAO [2], and Volumetric AO (VAO) [36] in both performance and visual quality. These methods are chosen because among existing SSAO methods, Blizzard’s SSAO is one of the fastest, while HBAO produces the best looking images. VAO is somewhere in between the two. Ray-traced images produced by the Blender software are also available as ground-truth references. Tests are run on two machines with different graphics card, one with an NVIDIA GeForce 8600M GT, and the other with a more powerful NVIDIA GeForce 460M GTX. 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 four resolutions, with the maximum screen-space kernel radius ($r_{max}$) set to 5 in all resolutions. The threshold $d_{threshold}$ used in the down-sampling step is set to 1. 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’s SSAO and Volumetric AO each uses 32 samples per pixel.

### 5.1 Qualitative Evaluations

Fig. 11, presents comparisons of visual quality among the four SSAO methods and the ground-truth references pro-

| | Blizzard AO | HBAO | VAO | MSSAO |
|---|---|---|---|---|
| Sibenik | 67.34% | 78.04% | 78.09% | 85.62% |
| Conference | 78.82% | 82.23% | 80.77% | 86.66% |
| Sponza | 74.96% | 82.10% | 79.61% | 89.04% |

**Table 6** SSIM indices of the results from the four SSAO methods against the reference images. Each SSIM number is shown as the percentage of similarity between two images. Larger numbers are better.

duced by Blender (images in the electronic version of this paper can be zoomed in for more details)[1]. Results show 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 is used at the finest resolution. Blizzard's results are noisy while NVIDIA's are blurry and suffer from heavy over-occlusion artifacts (see, for example, the contact shadows between the chairs and the ground in the Conference Room scene). Results produced by Volumetric AO are free of noise but also heavily biased as lots of details are lost. In all methods except MSSAO, 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 6, we also provide numerical comparison between the results using the Structural SIMilarity (SSIM) index [37], which is a perceptual image comparison metric. According to SSIM, results from our method have the highest similarities to the ground-truth images.

Fig. 12 demonstrates another advantage of MSSAO over most existing SSAO methods. Since SSAO methods lack a mechanism to check for visibility (or ray-geometry intersections), when the AO radius of influence is increased, the (lack of) occlusion caused by distant occluders often wrongfully affects that caused by nearby occluders. The most common consequence is that with a large AO radius of influence, small, sharp details that can otherwise be captured using a smaller radius of influence, are gone. Our method does not suffer from this problem, as we capture and retain shadow frequencies at multiple scales, and use a better way to combine them.

Finally, Fig. 13 shows additional results from our method for different kinds of scene models, demonstrating that our algorithm is suitable for a wide range of scenes with various kinds of object. All of the images are rendered at 512x512 pixels at near 40 fps on a NVIDIA GeForce 8600M GT. They can be zoomed in for more details.

[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.

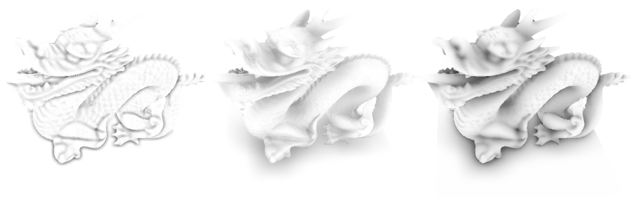

**Fig. 12** (Left) Result from a single-resolution SSAO method and a short AO radius of influence. (Middle) Result from a single-resolution SSAO method and a large AO radius of influence. (Right) Result from MSSAO with a large AO radius of influence. The left image lacks AO caused by distant occluders, the middle image lacks details such as the dragon's scales on its body, or the parts near its ear and its tail. The right image captures both nearby and distant AO.

| Step | Time taken |
|---|---|
| Geometry | 1.7 ms |
| Downsampling | 1.3 ms |
| Bilateral Filtering | 0.9 ms |
| Sampling | 4.0 ms |
| Temporal Filtering | 0.2 ms |

**Table 7** Break-down runtime performance of different steps in our method. Smaller numbers are better.

## 5.2 Performance Evaluations

In Table 7, we give detailed timings for each of the steps in our algorithm when rendering the Conference Room scene at 512x512 pixels on a GeForce 460M GTX graphics card. Note that the Geometry pass is independent of the method being use, although it is included to give the reader a more complete picture (e.g. from the table, one can deduce that the scene is rendered at about 123 fps). As expected, the sampling (and computing AO) step dominates the other steps in terms of time taken. As mentioned in Section 4.3, a faster sampling scheme such as a 16-point Poisson-disk pattern can be used if the AO radius of influence is large. In our case, $d_{max}$ is only 2 (and $r_{max}$ is 5) and using the interleaved pattern gives us the best quality without sacrificing performance.

Table 8 shows the running times of all three SSAO methods for rendering a frame of each of the three test scenes. We measure the actual timings of the AO algorithms, so the geometry passes are ignored in all methods.

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 the 8600M GT, our method can achieve better performance since our kernel sizes are small and the sampling pattern is more cache-friendly. On the newer graphics card, Blizzard's SSAO and Volumetric AO outperform ours by a very small margin. That is probably because of the GPU's bigger texture cache, so random sampling patterns and big

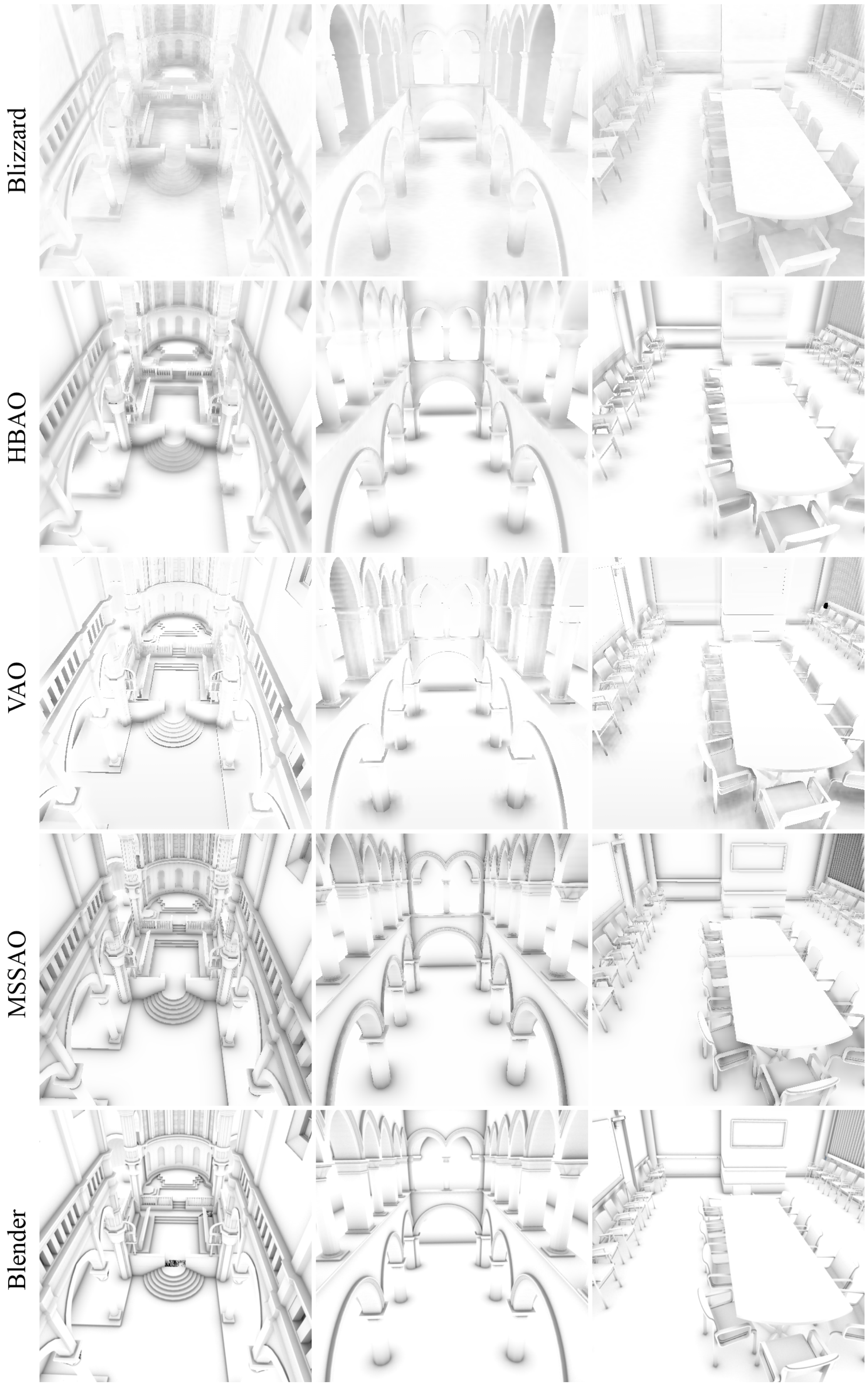

**Fig. 11** Comparison of AO results (without any other shading) produced by five methods for the three scenes. Blender's result is the reference solution.

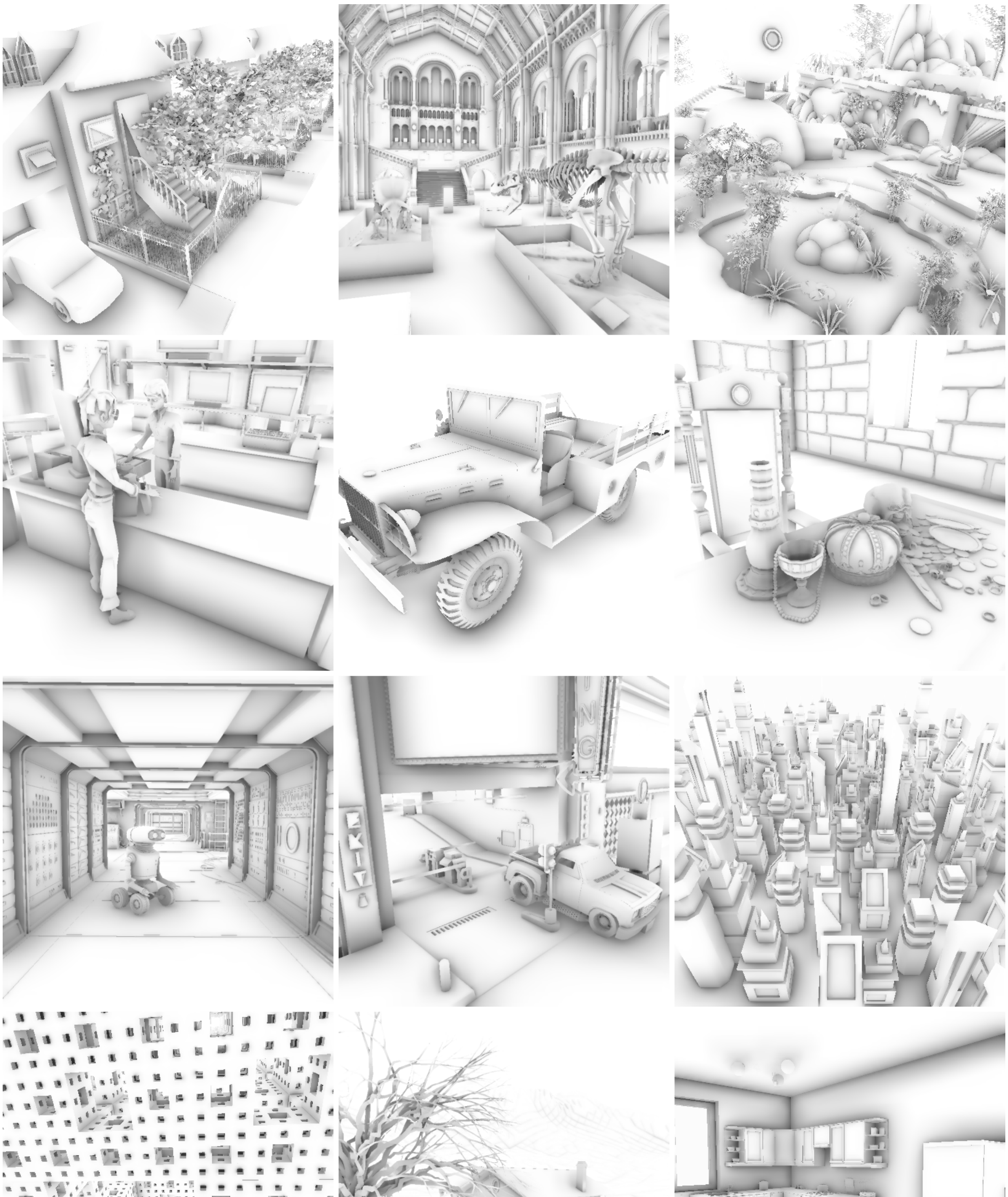

**Fig. 13** Additional results from our method.

| | GeForce 8600M GT | | | |
|---|---|---|---|---|
| | MSSAO | Blizzard | HBAO | VAO |
| Sibenik | 24.8 | 24.9 | 92.8 | 38.0 |
| Conference | 24.4 | 22.2 | 90.4 | 34.9 |
| Sponza | 24.1 | 30.6 | 92.9 | 39.5 |
| | GeForce 460M GTX | | | |
| Sibenik | 6.6 | 5.2 | 11.1 | 6.0 |
| Conference | 6.4 | 5.1 | 11.3 | 5.8 |
| Sponza | 6.5 | 5.1 | 11.2 | 6.0 |

**Table 8** Runtime performance of the four SSAO methods at 512x512 pixels, measured in millisecond (ms). Smaller numbers are better. Note that the running times of MSSAO are measured with the inclusion of temporal filtering.

| | GeForce 460M GTX | | | |
|---|---|---|---|---|
| | MSSAO | Blizzard | HBAO | VAO |
| Sibenik | 21.9 | 25.7 | 50.1 | 22.9 |
| Conference | 24.0 | 24.9 | 49.5 | 24.8 |
| Sponza | 22.2 | 28.9 | 54.3 | 24.0 |

**Table 9** Runtime performance of the four SSAO methods at 1024x1024 pixels, measured in millisecond (ms). Smaller numbers are better.

kernel sizes do not hurt performance as much. However, as we increase the resolution to 1024x1024 pixels, MSSAO outperforms the other methods again on the GeForce 460M GTX (see Table 9). One possible reason could be that our method does not perform an expensive final blur pass at the finest resolution like the others.

In general, we believe our method has achieved the high performance of Blizzard's SSAO and Volumetric AO, while attaining better image quality than that of all existing SSAO methods'.

### 5.3 Limitations

One of the limitations of our method is that it consumes more memory than existing SSAO methods. Each level of resolution (except the finest) makes use of four buffers, namely the positions, normals, AO, and filtered AO buffers. At the finest resolution we uses three buffers – positions, normals, and AO. A typical SSAO method uses only the three buffers (positions, normals, AO) at the finest resolution. Furthermore, each of our AO buffers contains three channels instead of one in order to "propagate" appropriate values up the levels of resolution. Assuming we use 4 resolutions, from 1024x1024 to 128x128, our method consumes 1.85 times as much memory as a typical SSAO method. In the context of this paper, where only AO is evaluated, this is not a problem with any decent graphics card. However, the effects of this increase in memory consumption may be noticeable when the method is used in a more realistic context, such as in a computer game, where memory is shared with other lighting and shading passes. We would like to mention that should memory bandwith become the bottleneck, further optimizations, such as reusing buffers, could be applied.

Our method also suffers from errors due to the use of coarse resolutions, even though the use of bilateral upsampling has reduced this effect greatly. This issue can be found in screen-space regions where there are lots of depth discontinuities. Such regions are often slightly over-occluded. The same problem also results in poor temporal coherence on thin geometry such as chair legs and tree leaves. However, it is worth to note that all these artifacts happen rarely and are not too noticeable.

## 6 Discussion and Conclusion

Being a screen-space method, our method inherits the fundamental limitations of SSAO. The AO computed by our method is only 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 they do not solve the problem completely, and in many cases create 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.

Despite having some inherent limitations of SSAO and some other minor drawbacks such as high memory usage and errors due to the lack of resolution, our method has improved on the state-of-the-art 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 leverage the GPU's texture caching to make the algorithm much faster. That enables us to sample further without sacrificing much performance, thus capturing more global AO effects. Together with a robust AO formula, our results are free of noise/blur even with a small number of samples. Bilateral filtering and upsampling are used repeatedly in coarser resolutions to smoothly blend and interpolate occlusion values to avoid artifacts from the use of multiple resolutions. The use of the maximum operator to combine occlusion values across multiple resolutions also helps counter the ill effects resulting from the lack of visibility checking in SSAO. As a result, our method captures multiple AO scales, which is not possible in other methods. Finally, 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** We thank the anonymous reviewers of CGI 2011 for their helpful comments and suggestions.

The scene models used in this paper are from the following sources: Marko Dabrovic (Sibenik Cathedral and Sponza Atrium), Anat Grynberg (Conference Room), ompf.org's Scene Vault (DWC-51 Truck, Jagdpanther Tank, and Kitchen), The Stanford 3D Scanning Repository (Dragon), Google 3D Warehouse (City by Scott, House Merlyn and Zona Residence by rubicundo, Residence by rubicundo2, and Menger Sponge by Duv), and 3dRender.com's Lighting Challenges (Natural History, The Shop Girls, The King's Treasure, Science Fiction, Christmas, The Local Train, Neon and Chrome, and Film Noir).

## 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. Dutre, P., Bekaert, P., Bala, K.: Advanced Global Illumination, 2nd edn. A K Peters/CRC Press (2005)
6. Filion, D., McNaughton, R.: Effects & techniques. In: ACM SIGGRAPH 2008 Courses, pp. 133–164 (2008)
7. Fox, M., Compton, S.: Ambient occlusive crease shading. Game Developer Magazine (March 2008) (2008)
8. Glassner, A.S.: Principles of Digital Image Synthesis. Morgan Kaufmann (1995)
9. Hoberock, J., Jia, Y.: High-quality ambient occlusion. In: GPU Gems 3, pp. 257–274. Addison-Wesley Professional (2007)
10. 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)
11. Keller, A., Heidrich, W.: Interleaved sampling. In: Rendering Techniques, pp. 269–276 (2001)
12. Kontkanen, J., Laine, S.: Ambient occlusion fields. In: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games, pp. 41–48 (2005)
13. Kopf, J., Cohen, M.F., Lischinski, D., Uyttendaele, M.: Joint bilateral upsampling. In: Proceedings of ACM SIGGRAPH 2007 (2007)
14. Laine, S., Karras, T.: Two methods for fast ray-cast ambient occlusion. Computer Graphics Forum **29**(4), 1325–1333 (2010)
15. Landis, H.: Production-ready global illumination. In: ACM SIGGRAPH 2002 Courses, pp. 331–338 (2002)
16. Loos, B.J., Sloan, P.P.: Volumetric obscurance. In: Proceedings of the 2010 Symposium on Interactive 3D Graphics and Games, pp. 151–156 (2010)
17. Malmer, M., Malmer, F., Assarsson, U., Holzschuch, N.: Fast precomputed ambient occlusion for proximity shadows. Journal of Graphics Tools **12**(2), 59–71 (2007)
18. Mattausch, O., Scherzer, D., Wimmer, M.: High-quality screen-space ambient occlusion using temporal coherence. Computer Graphics Forum **29**(8), 2492–2503 (2010)
19. McGuire, M.: Ambient occlusion volumes. In: Proceedings of the Conference on High Performance Graphics, pp. 47–56 (2010)
20. McGuire, M., Osman, B., Bukowski, M., Hennessy, P.: The alchemy screen-space ambient obscurance algorithm. In: High-Performance Graphics 2011 (2011)
21. Mittring, M.: Finding next gen: Cryengine 2. In: ACM SIGGRAPH 2007 Courses, pp. 97–121 (2007)
22. Nehab, D., Sander, P.V., Lawrence, J., Tatarchuk, N., Isidoro, J.R.: Accelerating real-time shading with reverse reprojection caching. In: Graphics Hardware (2007)
23. Nichols, G., Shopf, J., Wyman, C., Lensch, H.P.A., Sloan, P.P.: Hierarchical image-space radiosity for interactive global illumination. Computer Graphics Forum **28**(4), 1141–1149 (2009)
24. 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)
25. Nichols, G., Wyman, C.: Interactive indirect illumination using adaptive multiresolution splatting. IEEE Transactions on Visualization and Computer Graphics **16**, 729–741 (2010)
26. Paris, S., Kornprobst, P., Tumblin, J., Durand, F.: Bilateral filtering: Theory and applications. Foundations and Trends in Computer Graphics and Vision **4**(1), 1–73 (2009)
27. Reinbothe, C., Boubekeur, T., Alexa, M.: Hybrid ambient occlusion. In: Eurographics 2009 Areas Papers (2009)
28. 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)
29. 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)
30. Scherzer, D., Jeschke, S., Wimmer, M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence". In: Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering), pp. 45–50 (2007)
31. 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)
32. Shishkovtsov, O.: Deferred Shading in S.T.A.L.K.E.R., pp. 143–166. Addison-Wesley Professional (2005)
33. 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)
34. Smedberg, N., Wright, D.: Rendering techniques in gears of war 2. In: Game Developer Conference (2009)
35. Soler, C., Hoel, O., Rochet, F.: A deferred shading algorithm for real-time indirect illumination. In: ACM SIGGRAPH 2010 Talks, p. 18 (2010)
36. 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)
37. 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)
38. 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)
39. Zhukov, S., Inoes, A., Kronin, G.: An ambient light illumination model. In: Rendering Techniques '98, pp. 45–56 (1998)