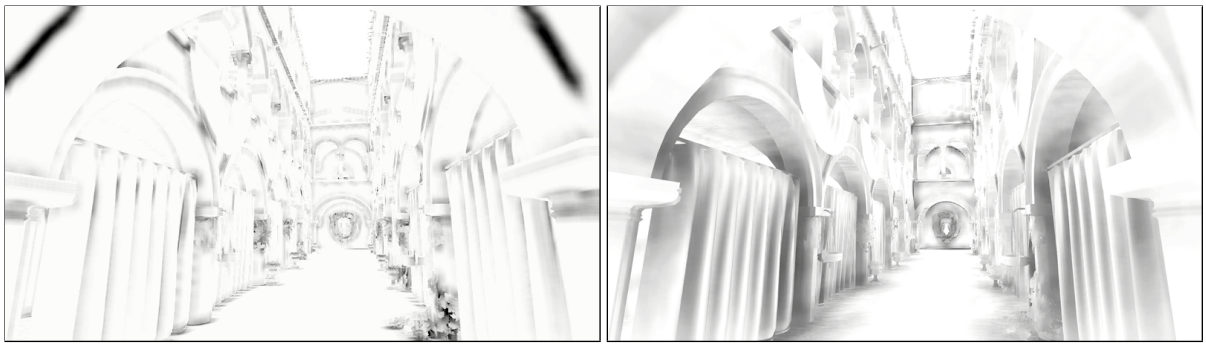
High Performance Graphics (2012)
C. Dachsbacher, J. Munkberg, and J. Pantaleoni (Editors)

Scalable Ambient Obscurance

Morgan McGuire[1,2], Michael Mara[1,2], and David Luebke[1]

[2]NVIDIA; [1]Williams College



Alchemy AO [MOBH11]: $r = 0.10$m, 1280×720 in 2.3 ms **New SAO Result**: $r = 1.5$m @ 1920×1080 in 2.3 ms

Figure 1: *Given a fixed time budget, our new algorithm substantially increases ambient obscurance resolution and quality over the 2011 AlchemyAO algorithm, which uses the same mathematical estimator. The resulting increased shading fidelity in this example allows the viewer to better discern shape in regions such as the archways and curtains.*

Abstract

This paper presents a set of architecture-aware performance and integration improvements for a recent screen-space ambient obscurance algorithm. These improvements collectively produce a $7\times$ *performance increase at* 2560×1600*, generalize the algorithm to both forward and deferred renderers, and eliminate the radius- and scene-dependence of the previous algorithm to provide a hard real-time guarantee of fixed execution time. The optimizations build on three strategies: pre-filter the depth buffer to maximize memory hierarchy efficiency; reduce total bandwidth by carefully reconstructing positions and normals at high precision from a depth buffer; and exploit low-level intra- and inter-thread techniques for parallel, floating-point architectures.*

1. Introduction

The Alchemy Ambient Obscurance algorithm by game studio Vicarious Visions [MOBH11] produces high quality screen-space ambient obscurance (AO). They described its design goals as: first, scaling down from then-current GPUs to the low-end Xbox 360, and second, integration with a deferred renderer at 1280×720 resolution. The AlchemyAO algorithm, like prior screen-space AO techniques, has received significant interest within the games industry.

This paper introduces a new variant of AlchemyAO for modern and future graphics architectures. We maintain the core mathematics of AlchemyAO but evaluate them with a new Scalable Ambient Obscurance (SAO) algorithm that improves on AlchemyAO in two ways. First, the new SAO algorithm requires as input only a standard depth buffer (i.e., it eliminates AlchemyAO's position and normal buffers), so it integrates with both deferred and forward renderers. Second, where AlchemyAO was designed to scale *down* to limited hardware at fixed resolution, our algorithm assumes modern hardware and scales *up* to high resolutions and world-space sampling radii. This addresses a known limitation of many previous screen-space AO methods, in which cache efficiency and thus net performance falls when gathering samples far from a pixel (shown in figure 5).

Figure 1 demonstrates the practical impact of our changes for real-time applications. Both images show the factor by which environment lighting will be attenuated due to occlusion at various scales. The left image shows the obscurance

result by the original algorithm. It exhibits the desirable features of AlchemyAO: grounded contact shadows, fine detail, and robustness to viewing direction. However, at the relatively modest 720p resolution, AlchemyAO can only sample a 0.1 m radius in 2.3 ms, limiting it to extremely local obscurance effects. The right image shows that even at higher 1080p resolution, the new SAO algorithm can sample a large 1.5 m radius in the same execution time, capturing more global occlusion.

As screen resolution increases, variance across pixels becomes less perceptible and bandwidth requirements increasingly limit performance. Holding the obscurance radius fixed and reducing filter quality provides a comparable visual result at the same execution time.

We also discuss approaches for maximizing precision of the depth buffer. This careful treatment of z precision enables us to discard the position and normal G-buffer of AlchemyAO and use a standard depth buffer instead, and then leads to our strategy of using fewer filter taps across a z-hierarchy to reduce bandwidth without sacrificing quality.

1.1. Related Work

Concurrent academic and industry work by Shanmugam and Arikan [SA07] and Mittring [Mit07] introduced physically-motivated screen-space ambient occlusion/obscurance for real-time rendering. Subsequent papers and presentations [Kaj09, Kap10, FM08, BS09, LS10, SKUT*10, MOBH11] improved performance and quality.

Mathematically, our technique combines the AlchemyAO [MOBH11] estimator, a rotating sample pattern [Kaj09], and a bilateral filter for reconstructing smooth AO from samples distributed over screen-space and the visible hemisphere [Kaj09, Kap10]. We also inherit the AlchemyAO estimator's treatment of the depth buffer as a thin shell (instead of an infinitely thick volume) for view robustness, which was first developed by Loos and Sloan [LS10].

AlchemyAO [MOBH11] makes three passes: noisy AO estimated from s samples at each pixel, and horizontal and bilateral blur respecting depth edges. Its AO estimator for camera-space point $C = (x_C, y_C, z_C)$ with normal $\hat{n}_C$ is:

$$A(C) = \max\left(0, 1 - \frac{2\sigma}{s} \cdot \sum_{i=1}^{s} \frac{\max(0, \vec{v}_i \cdot \hat{n}_C + z_C \beta)}{\vec{v}_i \cdot \vec{v}_i + \varepsilon}\right)^k \quad (1)$$

where constants σ, β, and k are chosen for aesthetics and ε is a small offset to avoid division by zero. The estimator relies on points $\{Q_1, \ldots, Q_s\}$ distributed on a ball about C, each of which yields a displacement $\vec{v}_i = Q_i - C$. We use their estimator unmodified within a new algorithmic structure that is both more generally applicable and exhibits better absolute and asymptotic performance.

Multiresolution AO (MAO) [HL12] computes AO at different output scales and then upsamples with joint-bilateral filtering before compositing. We also leverage this approach to improve cache coherence, but apply it to the z input instead of the AO output. To appreciate the difference, consider the obscurance cast onto a distant a receiver object. MAO gathers *onto* large receivers. Thin features and high frequency geometry can create flickering because they are undersampled. Our SAO gathers distant occlusion *from* large casters. This means that the shadows cast by thin objects fade out quickly (which one expects under area illumination), but that they receive shadowing at all scales. Because we always sample every object at every pixel, there is no undersampling and our approach is temporally robust.

2. Algorithm

Our algorithm takes as input a standard depth buffer and makes a series of passes over the full screen that result in an ambient visibility value $0 \leq A \leq 1$ at each pixel. The application's forward or deferred lighting pass then modulates environment an area illumination by A. The supplemental materials for this paper include GLSL implementation of all shaders. The input depth buffer is larger than the screen so that objects in a guard band slightly outside the frame can create obscurance. The following sections describe the purpose and optimizations of each pass.

2.1. High-Precision z Prepass

The input depth buffer typically arises from a depth-only prepass, for which modern GPUs are highly optimized. Most renderers perform such a pass to avoid later shading fragments that are ultimately occluded.

Precision for the depth buffer is key because we will derive all other values in our algorithm from it. Research and industry have carefully studied depth buffer formats to maximize the depth *discrimination precision* (e.g., [LJ99, AS06]), but given less attention to maximizing the *reconstruction accuracy* of camera-space points and normals from depth values. The latter problem involves the entire pipeline: creation of the modelview projection matrix on the host, the vertex transformation, hardware attribute interpolation, and storage in a depth buffer. Every arithmetic operation in that pipeline introduces error, so one must minimize operations and maximize the precision at which they are performed.

We observe that the following increase the accuracy of z values computed from a depth buffer on modern GPUs:

1. Compute the modelview-projection matrix at double precision on the host before casting to GPU single precision. This comprises three matrix products (projection, camera, and object), divisions, and trigonometric operations.

2. Choose $z_f = -\infty$ in the projection matrix [Smi83]. This reduces the number of floating point ALU operations required for the matrix product [UD12].

3. When using column-major matrices (the default in OpenGL), multiply vectors on the left ($\vec{v}' = \vec{v}^{\mathrm{T}}\mathbf{P}$) in the vertex shader. This saves about half a bit of precision.

Note that camera-space positions and normals stored in a G-buffer, such as those by AlchemyAO, also contain error. The rasterizer interpolates those across a triangle as $C/z, \hat{n}/z$, and then divides by z at each pixel, where z itself was interpolated as $1/z$ with limited fixed-point precision.

2.2. Hierarchical z Pass

This pass converts the hardware depth buffer value $0 \le d \le 1$ to a camera-space value $z < 0$ at each pixel by

$$z(d) = \frac{\mathbf{c}_0}{d \cdot \mathbf{c}_1 + \mathbf{c}_2}, \tag{2}$$

where z_n and z_f are the locations of the near and far planes, and constant array $\mathbf{c} = [z_\mathrm{n}, -1, +1]$ when $z_\mathrm{f} = -\infty$, and $\mathbf{c} = [z_\mathrm{n} z_\mathrm{f}, z_\mathrm{n} - z_\mathrm{f}, z_\mathrm{f}]$ otherwise. The pass then builds a MIP hierarchy for the z texture.

Each z value will later be read many times. MIP level 0 amortizes the division operation from equation 2 over those samples. The remaining MIP levels ensure that spatially-distributed samples taken in the following pass are read with high cache efficiency. Because a small region in each level will remain in cache, few reads will actually go to DRAM, resulting in high bandwidth and low latency. This addresses a scalability limitation of the original AlchemyAO that has also been observed in other screen-space AO methods.

2.3. Distributed AO Sample Pass

This pass distributes s samples on a half-ball about camera-space point C and normal n_C found at integer pixel location (x', y'). We recover C and $\hat{n}_C$ points from $z_C = z(x', y')$ by

$$(x_C, y_C) = z_C \cdot \left(\frac{1 - \mathbf{P}_{0,2}}{\mathbf{P}_{0,0}} - \frac{2(x' + \frac{1}{2})}{w \cdot \mathbf{P}_{0,0}}, \frac{1 + \mathbf{P}_{1,2}}{\mathbf{P}_{1,1}} - \frac{-2(y' + \frac{1}{2})}{h \cdot \mathbf{P}_{1,1}} \right) \tag{3}$$

$$\hat{n}_C = \mathrm{normalize}\left(\frac{\partial C}{\partial y'} \times \frac{\partial C}{\partial x'} \right) \tag{4}$$

for a $w \times h$ screen and projection matrix $\mathbf{P}$. Eqn. 3 inverts the projection matrix at a pixel to find its camera-space position C. Eqn. 4 then infers the camera-space face normal of the surface containing C from its screen-space gradient.

The world-space radius r of the ball corresponds to screen-space radius r'

$$r' = -r S' / z_C, \tag{5}$$

where S' pixel-size of a 1m object at $z = -1\mathrm{m}$ [MOBH11].

We place s direct samples in a spiral pattern, spatially varying the orientation about each pixel (figure 2). Sample i is taken at pixel $(x', y') + h_i \hat{u}_i$, where

$$\begin{aligned} \text{Let } \alpha_i &= \tfrac{1}{s}(i + 0.5) \\ h'_i &= r' \alpha_i; \;\; \theta_i = 2\pi \alpha_i \tau + \phi \qquad (6) \\ \hat{u}_i &= (\cos\theta_i, \sin\theta_i). \qquad (7) \end{aligned}$$

Constant τ is the number of turns around the circle made by the spiral, chosen to ensure equal angular distribution (we use $\tau = 7$ for $s = 9$). Angular offset ϕ is the random rotation angle at a pixel. We use AlchemyAO's XOR hash,

$$\phi = 30x' \wedge y' + 10x'y'. \tag{8}$$

Because MIP level m_i for sample i depends on h'_i and not a screen-space derivative, we explicitly compute:

$$\begin{aligned} m_i &= \lfloor \log_2(h'_i / q') \rfloor \qquad (9) \\ z_i &= z^{m_i}\left((x', y') + h_i \hat{u}_i)/2^{m_i}\right) \qquad (10) \end{aligned}$$

Constant q' is the screen-space radius increment at which we switch MIP levels; the optimal value depends on the resolution, number of GPU cores, and cache size. In our experiments, all $2^3 \le q' \le 2^5$ gave equivalent results. Lower values caused multiple taps at adjacent pixels to map to the same texel at a low MIP level, which amplified sample variance and manifested as temporal flicker. Higher values decreased performance because the working area no longer fit in cache. The implementation is efficient:

```
int m = clamp(findMSB(int(h)) − log_q, 0, MAX_MIP);
float z = texelFetch(zBuffer, ivec2(h∗u+xy) >> m, m).r;
```

We apply eqn. 3 to reconstruct each Q_i from z_i and then estimate its contribution to obscurance by eqn. 1.

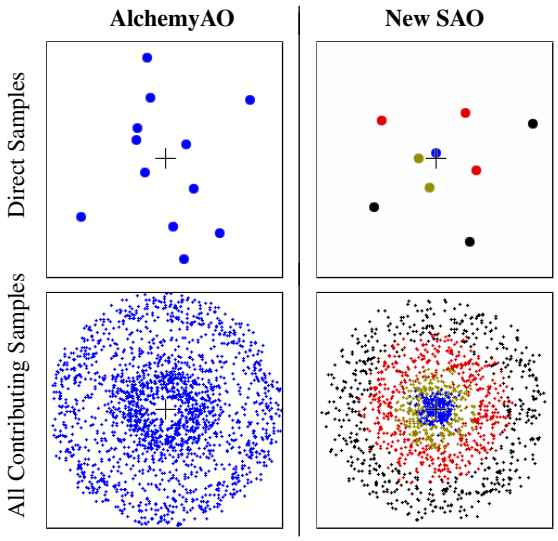


Figure 2: *The new algorithm achieves similar quality with fewer direct samples by improving the quality of the reconstruction and better distributing the samples. It further increases performance by taking samples at different hierarchical z levels. In this figure, the sample radius is much larger than the blur filter kernel. Top row: Points $\{Q_i\}$ directly sampled at one pixel. Bottom row: All points affecting one pixel after bilateral reconstruction. Left column: AAO with $12 \cdot (13 \times 13) = 2028$ taps. Right column: The new algorithm with $9 \cdot (2 \times 2) \cdot (7 \times 7) = 1764$ taps color-coded by hierarchy level.*

This pass concludes by applying a small 2×2 bilateral reconstruction filter, averaging A values to reduce variance where there is no significant depth discontinuity. It exploits the fact that GPUs process pixels within a 2×2 quad in parallel, by using screen-space derivative instructions to avoid the synchronization cost of inter-thread communication. See the supplemental material for implementation details.

The sampling pass as described has the advantage of a low, fixed runtime. The performance can be increased further, at the cost of variable runtime, by masking distant pixels such as the skybox. This is easily accomplished by invoking the AO pass as a shader on a full-screen quad placed at the appropriate depth.

To halve the number of texture fetch instructions and reduce input bandwidth per pixel in the next pass, at the end of the AO pass we pack A and z into an RGB8 texture as:

$$(R,G,B) \;=\; \left(A,\; \tfrac{1}{256}\mathrm{fix}\left(\tfrac{256\cdot z}{z_{\min}}\right),\; \mathrm{fract}\left(\tfrac{256\cdot z}{z_{\min}}\right)\right),$$

where $z_{\min}$ defines the farthest plane at which sharp features in AO are desired (-200m for our result figures).

2.4. Bilateral Reconstruction Passes

We reconstruct a piecewise-smooth AO solution from the raw sample buffer with two wide bilateral 1D filter passes, one horizontal and one vertical. Each uses seven taps with Gaussian weights modulated by linear depth difference scaled by an empirically chosen constant (we use 2^{11} for all examples here; increasing this constant increases sharpness while decreasing it reduces noise). We are able to space the taps with a stride of three pixels because the previous pass already filtered over 2×2 boxes. Our combined filters produce results comparable to previous work with slightly fewer taps in a slightly more uniform distribution (figure 2). As before, we invoke these passes with a full-screen rectangle at the far plane with a reverse depth test; this ensures that pixels at infinity do not contribute.

3. Results

3.1. Early z Precision

Figure 3 shows the accuracy of camera-space z values recovered from a depth buffer for combinations of formats and far plane positions. There is little accuracy difference between fixed and floating point formats because the process of computing the depth buffer values destroys more precision than the storage format. The worst-case error is on the order of 2mm.

Figure 4 shows that face normal recovery is accurate within 0.2°–better than RGB8 G-buffer normals in the worst case. At depth edges, normal recovery fails. One could estimate those normals from neighbors for features wider than one pixel. However, for AO there is no need to do so because the final bilateral blur eliminates single-pixel errors.

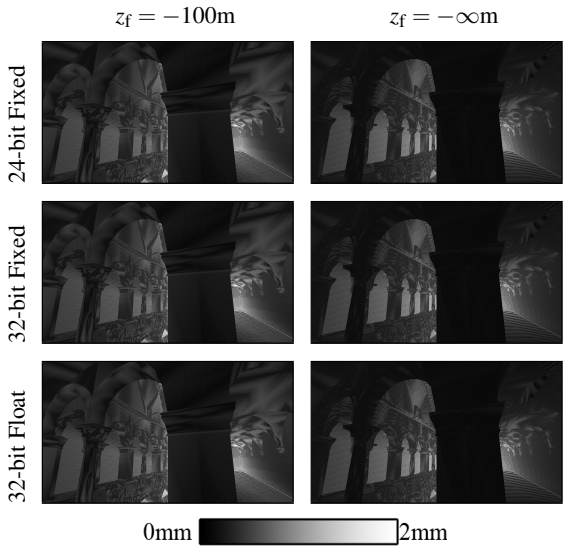


Figure 3: *An infinite far plane impacts recovered z accuracy more than depth buffer format at 24-bit or higher precision.*

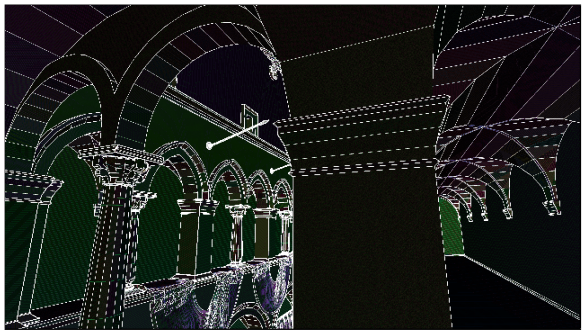


Figure 4: *Absolute face normal reconstruction difference, times 50. The peak error observed within a face was* 0.2°. *Results are undefined at depth discontinutities.*

3.2. Performance

SAO is a bandwidth-limited algorithm, so performance is proportional to bandwidth, which is in turn strongly driven by cache efficiency. Prefiltering the scene by computing a z hierarchy can be seen as a screen-space 2.5D analog of voxel LOD for cone tracing. It improves cache efficiency from an $O(\sqrt{r})$ miss-rate to a low-constant miss rate, as shown in figure 5. Results for that figure were measured on GeForce GTX 580 for the view in figure 6.

The top line is the [MOBH11] algorithm at $s = 12$, which exhibits the poor scaling noted in the literature. The center line is our algorithm without MIP-maps at $s = 9$ (which produces comparable image quality). Our bandwidth-reduction changes reduce the slope of the curve, for a 2.5× performance gain near the asymptote. The bottom line shows that our z hierarchy provides perfect scaling and another 3× for

	AlchemyAO	New SAO	
Memory traffic	293 B/pix	135 B/pix	**2.2**×
Time per pass:			
z^0 MIP	-	0.09 ms	
$z^1 \ldots z^n$ MIPs	-	0.18 ms	
Distributed AO	13.53 ms	1.33 ms	
Bilateral	2.57 ms	0.66 ms	
Total Time	16.10 ms	2.26 ms	**7.1**×

Table 2: *Memory traffic and timing breakdown at $r = 1.0$m, 1920×1080 on GeForce GTX 580 with a 190 pixel guard band, for the scene in figure 6.*

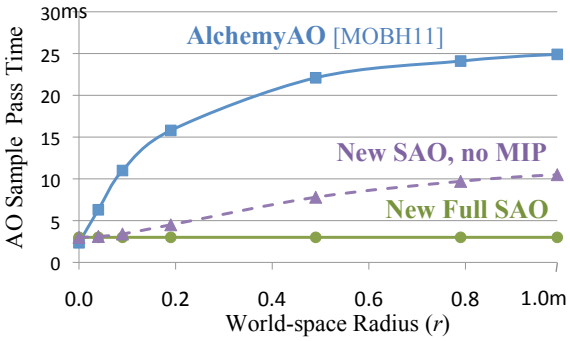


Figure 5: *Net performance for the Distributed AO Sample pass due to cache efficiency as a function of world-space sampling radius at 2560×1600.*

a net $7\times$ improvement. The overhead of the z reconstruction pass means that for very small radii (where the hierarchy is unused) the original algorithm may still outperform ours. Table 3 gives detailed performance per pass for SAO on GeForce GTX 680.

We motivated MIP-filtering by performance. What filter method gives the best image quality? Table 1 summarizes results on five filters. For each it gives the theoretical motivation, equation, and a representative result. We conclude that rotated grid subsampling is the best choice. It always produces values that were in the original scene is comparable to the quality baseline, and has the same performance as hardware MIP-map generation.

4. Discussion

SAO significantly improves the performance and quality of previous screen-space AO, particularly at high resolutions. This is important for forward-looking graphics applications: while the screen resolutions we evaluate (such as 2560×1600) were once considered exotic, such resolutions are rapidly becoming mainstream even on mobile platforms. The strong performance guarantee of SAO also represents an important pragmatic advance over AlchemyAO, for which cost increases with AO radius and proximity to the camera. Predictable performance is as essential as throughput for applications like games with strong real-time constraints.

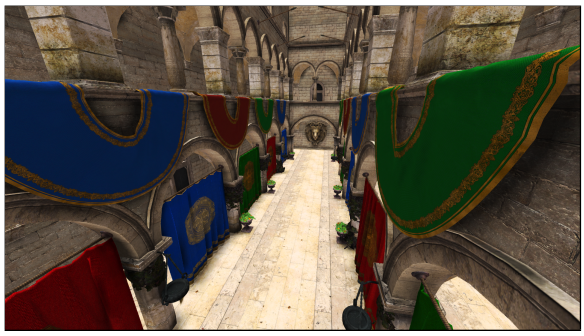


Figure 6: *Crytek Sponza with AO by our algorithm ($r = 0.5$).*

Figure 7: *Closeup of AO and final lit and textured image.*

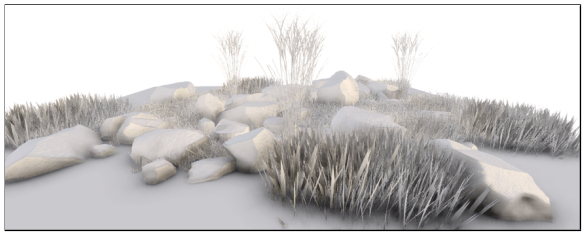


Figure 8: *SAO results on thin features, using the geometry from figure 12 from [MOBH11].*

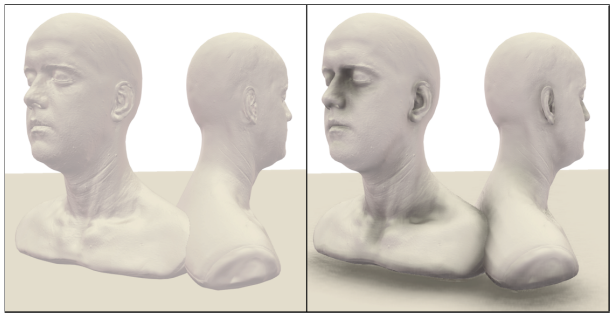


Figure 9: *Left: environment lighting only. Right: SAO brings out shape details critical to the perception of human facial features.*

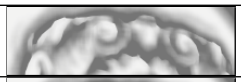
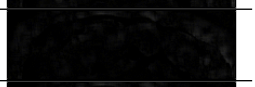
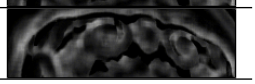
Filters	**Equation for $z^{m+1}(x',y')$**	**AO**	**Diff $\times 4$**
Quality Baseline *i.e., no MIP*	N.A.		
Rotated Grid Subsample *true scene values, no bias*	$z^m(2x'+(y'\&1^\wedge 1), 2y'+(x'\&1^\wedge 1))$		
Arithmetic Mean *preserve screen-space area*	$\frac{1}{4}\sum_{i,j} z^m(2x'+i, 2y'+j)$		
Harmonic Mean *preserve world-space area*	$\left[\sum_{i,j} 4/z^m(2x'+i, 2y'+j)\right]^{-1}$		
Min *conservative visibility*	$\min_{i,j} z^m(2x'+i, 2y'+j)$		
Max *conservative occlusion*	$\max_{i,j} z^m(2x'+i, 2y'+j)$		

Table 1: *AO quality resulting from different filters for creating level $m+1$ MIP map of the camera-space z buffer. & and $^\wedge$ are the C bitwise AND and XOR operators.*

Resolution	Guard Band Radius	Total Pixels (including guard)	Depth to Z	Z MIP	Sparse AO	Blur	Net AO	Time per Visible Pixel
1280×720	128	1,499,136	0.07	0.16	0.38	0.20	0.81 ms	0.879 ns
1920×1080	192	3,373,056	0.14	0.24	0.78	0.41	1.59 ms	0.767 ns
1920×1200	192	3,649,536	0.16	0.25	0.87	0.46	1.74 ms	0.755 ns
2560×1600	256	6,488,064	0.28	0.37	1.54	0.81	3.01 ms	0.735 ns

Table 3: *Time per pass (in milliseconds) measured with GL_TIMER_QUERY on GeForce GTX 680 at varying resolutions.*

Figure 10: *Scenes with architectural detail at many scales and both curved and linear features. All global illumination and shadowing in these images is due to SAO.*

One can vary the number of sample and blur taps to adjust run time vs. spatial and temporal noise. Our results are tuned for the constants that we expect applications to use on current GPUs, accepting some high-frequency temporal noise in AO that would typically be mitigated by texture details and special effects. On future hardware, or for scenes with low texture detail, we recommend increasing the number of sample and blur taps.

A limitation that SAO shares with other screen-space AO techniques is the need for a guard band on the viewport so that offscreen geometry contributes obscurance. This increases the memory footprint: a 5% guard band increases pixel count by 28% at 1920×1200. The impact on rendering time is minor, since only the very efficient z and hierarchical z passes affect guard band pixels. Color and intermediate buffer pixels in the guard band represent wasted memory. This suggests potential GPU architecture and API extensions, such as allowing the programmer to specify subrectangles or virtual memory pages of a render target that will actually be used and thus need to be physically allocated.

We also observe that some guard band pixels of the depth and hierarchical z buffers could be reclaimed once the buffers have been created, since fetches toward the outer edge of the guard band will always be taken from coarser levels. About half the memory devoted to guard band z data could theoretically be reclaimed, for example by invalidating the corresponding virtual memory pages.

Appendix: Updated Falloff (June 24, 2012)

Since the official publication of this paper in HPG2012, we've tuned our falloff function slightly and now advocate a function equivalent to:

$$A = \max\left(0, 1 - \frac{5}{r^6 S}\sum_{i=0}^{S-1} \max((r^2 - \vec{v}\cdot\vec{v})^3, 0)\max(\frac{\vec{v}\cdot\hat{n} - \beta}{\vec{v}\cdot\vec{v} + \varepsilon}, 0)\right)$$

for $\beta \approx 1$cm and $\varepsilon \approx 1$cm in human-scale scenes. This has the same general shape and form as the originally published falloff but lowers the contrast near sharp, dark corners. See the demo on our website for implementation details.

Acknowledgements

We thank Naty Hoffman (Activision Studio Central), Leonardo Zide (Treyarch), and Louis Bavoil (NVIDIA) for their input on this paper and implementation, Guedis Cardenas (Williams) for his work in the Williams graphics lab, and Michael Bukowski, Brian Osman, Padraic Hennessy and the rest of the team at Vicarious Visions for access to their original AlchemyAO source code and helping to tune the latest version for production use. Thanks to Eric Haines and Peter-Pike Sloan for comments on the paper itself.

References

[AS06] AKELEY K., SU J.: Minimum triangle separation for correct z-buffer occlusion. In *Graphics Hardware* (New York, NY, USA, 2006), GH '06, ACM, pp. 27–30. 2

[BS09] BAVOIL L., SAINZ M.: Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH 2009: Talks* (2009), ACM, pp. 1–1. 2

[FM08] FILION D., MCNAUGHTON R.: Effects & techniques. In *SIGGRAPH 2008 courses* (2008), ACM, pp. 133–164. 2

[HL12] HOANG T.-D., LOW K.-L.: Efficient screen-space approach to high-quality multiscale ambient occlusion. *Vis. Comput. 28*, 3 (Mar. 2012), 289–304. 2

[Kaj09] KAJALIN V.: Screen space ambient occlusion. In *ShaderX*^{7}, Engel W., (Ed.). Charles River Media, Mar 2009. 2

[Kap10] KAPLANYAN A.: CryENGINE 3: Reaching the speed of light. In *SIGGRAPH 2010 courses* (August 2010), ACM. 2

[LJ99] LAPIDOUS E., JIAO G.: Optimal depth buffer for low-cost graphics hardware. In *Graphics Hardware* (New York, NY, USA, 1999), HWWS '99, ACM, pp. 67–73. 2

[LS10] LOOS B. J., SLOAN P.-P.: Volumetric obscurance. In *Proceedings of I3D* (2010), ACM, pp. 151–156. 2

[Mit07] MITTRING M.: Finding next gen: CryEngine 2. In *SIGGRAPH 2007 courses* (2007), ACM, pp. 97–121. 2

[MOBH11] MCGUIRE M., OSMAN B., BUKOWSKI M., HENNESSY P.: The alchemy screen-space ambient obscurance algorithm. In *Proceedings of HPG* (Aug 2011). 1, 2, 3, 4, 5

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on GPUs. In *Proceedings of I3D* (2007), ACM, pp. 73–80. 2

[SKUT*10] SZIRMAY-KALOS L., UMENHOFFER T., TÓTH B., SZÉCSI L., SBERT M.: Volumetric ambient occlusion for real-time rendering and games. *IEEE CG&A 30*, 1 (2010), 70–79. 2

[Smi83] SMITH A. R.: The viewing transformation, 1983. 2

[UD12] UPCHURCH P., DESBRUN M.: Tightening the precision of perspective rendering. *JGT 16*, 1 (2012), 40–56. 2