Real-Time Ray-Traced Ambient Occlusion of Complex Scenes using Spatial Hashing

Pascal Gautron
NVIDIA

ABSTRACT

Ambient occlusion is often approximated in real-time using screen-space techniques, leading to visible artifacts. Raytracing provides a unique way to increase the rendering fidelity by accurately sampling the distance to the surrounding objects, but it introduces sampling noise. We propose a real-time ray-traced ambient occlusion technique in which noise is filtered in world space. Using extended spatial hashing for efficient storage, multiresolution AO evaluation and ad-hoc filtering, we demonstrate the usability of our technique as a production feature usable in CAD viewports with scenes comprising hundreds of millions of polygons.

CCS CONCEPTS

• **Computing methodologies** → **Ray tracing**.

ACM Reference Format:
Pascal Gautron. 2020. Real-Time Ray-Traced Ambient Occlusion of Complex Scenes using Spatial Hashing. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '20 Talks), August 17, 2020.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3388767.3407375

We introduce Hashed Raytraced Ambient Occlusion (HRTAO) for noise-free, raytraced ambient occlusion of complex CAD scenes. Unlike screen-space methods such as HBAO [Bavoil et al. 2008], we consider ambient occlusion as a world-space phenomenon for which spatial coherence must be processed in world space. Texture space shading techniques [Munkberg et al. 2016] take a similar path, by storing lighting in UV space. However, this requires the presence of low-distortion, paintable UVs. AO can also be cached at the geometry vertices, but artifacts appear with uneven tessellation.

Some other approaches use world-space data structures such as octrees [Krivanek and Gautron 2009]. Such approaches usually target offline rendering due to their inefficiency on graphics hardware.

Spatial hashing [Binder et al. 2018] is a GPU-friendly scheme for improved path tracing convergence for secondary bounces. We introduce a robust spatial hashing solution for ambient occlusion.

1 SPATIAL HASHING

The spatial hashing technique hashes world-space coordinates into two distinct 32-bit keys. The first is the index in the hash map where the lighting information will be stored. Since a 32-bit key could result in having unrelated points linked to the same hash index, a second hash function generates a checksum from the same coordinates. When a collision is detected, a linear search finds the next free space. Storage and access are then performed in (near) constant time, making this approach efficient for parallel execution. The hash map is reused across frames for temporal coherence.

Figure 1: Fixed-size hash cells lead to aliasing (left). Our adaptive cell sizing scheme adapts to any scene scale (middle, right). Vancouver model courtesy of CADMakers

2 MULTIRESOLUTION 8D HASH MAP

A 3D hash function is obtained from a 1D hash function h as follows:

$$H_{3D}(p) = h(p_z + h(p_y + h(p_x))) \tag{1}$$

Hashing neighboring 3D points in world space into a same hash key requires discretization, but using fixed steps leads to aliasing (Figure 1). As for regular textures, we use discrete levels of detail. We define a target feature size s_w in world space based on an approximate, user-defined feature size in pixels s_p, the camera aperture f, the resolution R and the distance d from the camera:

$$s_w = d \tan(\max(f/R_x, fR_x/R_y^2)s_p) \tag{2}$$

Our adaptive position discretization step s_{wd} is then:

$$s_{wd} = 2^{\lfloor \log_2(s_w/s_{min}) \rfloor} s_{min} \tag{3}$$

where s_{min} is an arbitrarily small value (e.g. 10^{-10}) defining the smallest possible feature in world space, which is used as a starting point on the log scale. We also embed s_{wd} in the hashing function, so that the hash map can efficiently store any number of LODs:

$$H_{4D}(p) = h(s_{wd} + h(p_z/s_{wd} + h(p_y/s_{wd} + h(p_x/s_{wd})))) \tag{4}$$

We also include surface normal in the hashing function:

$$H_{7D}(p, n) = h(\lfloor n_z s_{nd} \rfloor + h(\lfloor n_y s_{nd} \rfloor + h(\lfloor n_x s_{nd} \rfloor + H_{4D}(p)))) \tag{5}$$

where s_{nd} is an arbitrary discretization value, e.g. $s_{nd} = 3$.

The last dimension supports dynamic scenes, by embedding an animation frame identifier F incremented upon scene modifications:

$$H_{8D}(p, n, F) = h(F + H_{7D}(p, n)) \tag{6}$$

The shaded points are then hashed in different cells over time, hence avoiding ghosting and allowing us to lazily remove obsolete values.

SIGGRAPH '20 Talks, August 17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7971-7/20/08.
https://doi.org/10.1145/3388767.3407375

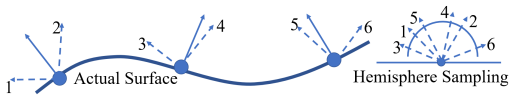

Figure 2: QMC sequence indices within a hash cell: sharing the sequence among the hit points optimizes sampling

3 COLLABORATIVE RAYTRACING

Computing ambient occlusion at a point requires sampling the surrounding hemisphere. Since we only store one AO value per hash cell, all points within a cell contribute to a same final value. Using a quasi-Monte-Carlo sequence [Keller 2013] with a single QMC sequence per hash cell we avoid tracing redundant rays, hence efficiently sampling the spatial and directional domains (Figure 2).

For each pixel we trace a number of rays from the corresponding hit point, and update its hash cell. This implicitly balances the workload towards cells with high visibility.

4 MULTIRESOLUTION AMBIENT OCCLUSION

Walking through a scene requires generating and using multiple levels of detail for a given point. We avoid redundancy by propagating the raytracing results using a bottom-up approach. For each point we allocate 4 cells: one at the target feature size, and 3 successive coarser sizes. The contribution of each ray is then propagated to the 3 coarser levels. Coarser LODs comprise more pixels, therefore inexpensively adding lower-resolution, converged AO values.

If the cell at the target feature size has an insufficient sample count we fetch additional samples from coarser levels, hence locally trading noise for spatial resolution.

5 ADAPTIVE FINAL FILTERING

We remove cell boundaries using an adaptive filtering technique based on the knowledge of the coarsest LOD used to compute the AO values. This LOD sets a higher bound on the spatial resolution, and hence determines the filtering kernel width. Our bilateral filter is based on the same criteria as the hashing function for consistency, and uses the idea of *a trous sampling* used by Schied et al. [2017].

6 RESULTS AND FUTURE WORK

We implemented HRTAO using the Vulkan API, and evaluated it by rendering 3 CAD scenes at 1080p resolution on an NVIDIA Quadro RTX8000: Vancouver (Figure 1), Endeavor and Servers (Figure 4). The timings in Table 1 are obtained by tracing 10 rays per pixel,

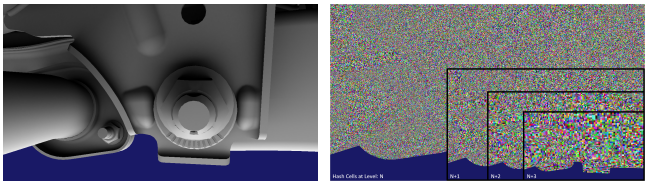

Figure 3: For a frame (left) we allocate hash cells for each pixel at 4 levels of detail (right, false colors hash indices)

Table 1: Performance at first frame and during navigation

Scene Name	Poly Count	Instance Count	HRTAO 0s (ms)	HRTAO nav. (ms)
Vancouver Exterior	60M	220K	8.3	2.0
Endeavor Interior	50M	92K	30.5	2.5
Servers	307M	30K	14.3	1.6

up to a maximum of 1000 samples per hash cell. This upper limit reduces the overall ray count and focuses on newly visible areas.

The cost of the HRTAO passes on the first frame is shown in Table 2. For a given scene the cost of finding the hash cells is roughly constant, as is the computation of per-pixel AO values. Depending on the workload, the raytracing and filtering times vary. The ray budget can be adjusted to match a target rendering speed.

HRTAO is a simple solution for noise-free raytraced ambient occlusion in massive scenes. Being world-space by nature, it avoids the pitfalls of screen-space occlusion and denoising techniques.Work in progress includes extensions to environment shadows, diffuse and glossy global illumination, and efficient multi-GPU implementation.

Table 2: Performance breakdown (ms) at the first frame

Scene Name	Cell Finder	Raytracing Update	Per-pixel AO	Filtering
Vancouver Exterior	1.4	6.1	0.4	0.4
Endeavor Interior	1.7	26.1	0.5	2.2
Servers	1.1	12.5	0.3	0.4

REFERENCES

Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. 2008. Image-Space Horizon-Based Ambient Occlusion. In *ACM SIGGRAPH 2008 Talks*. Article 22. https://doi.org/10.1145/1401032.1401061

Nikolaus Binder, Sascha Fricke, and Alexander Keller. 2018. Fast Path Space Filtering by Jittered Spatial Hashing. In *ACM SIGGRAPH 2018 Talks*. Article 71. https://doi.org/10.1145/3214745.3214806

Alexander Keller. 2013. Quasi-Monte Carlo Image Synthesis in a Nutshell. *Springer Proceedings in Mathematics and Statistics* 65 (01 2013), 213–249. https://doi.org/10.1007/978-3-642-41095-6_8

Jaroslav Krivanek and Pascal Gautron. 2009. Practical Global Illumination with Irradiance Caching. *Synthesis Lectures on Computer Graphics and Animation* 4, 1 (2009). https://doi.org/10.2200/S00180ED1V01Y200903CGR010

Jacob Munkberg, Jon Hasselgren, Petrik Clarberg, Magnus Andersson, and Tomas Akenine-Möller. 2016. Texture Space Caching and Reconstruction for Ray Tracing. *ACM Trans. Graph.* 35, 6, Article 249 (2016). https://doi.org/10.1145/2980179.2982407

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. In *Proceedings of High Performance Graphics*. Article 2. https://doi.org/10.1145/3105762.3105770

Figure 4: Endeavor (50M triangles) - Servers (307M triangles)