Correlation-Aware Semi-Analytic Visibility for Antialiased Rendering

Cyril Crassin
NVIDIA

Chris Wyman
NVIDIA

Morgan McGuire
NVIDIA

Aaron Lefohn
NVIDIA

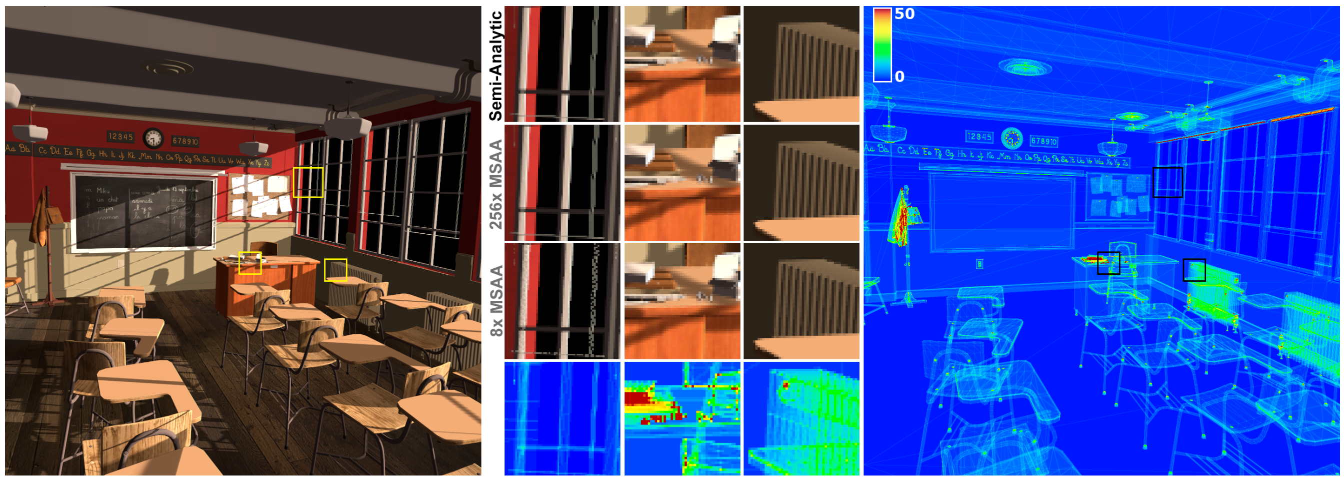


Figure 1: Our approach (left) accounts for geometry semi-analytically, approaching 256× **multisampled quality (see insets), despite aggregating contributions from up to 50 triangles per pixel. The heatmap (right) shows per-pixel fragment counts.**

ABSTRACT

Geometric aliasing is a persistent challenge for real-time rendering. Hardware multisampling remains limited to 8×, analytic coverage fails to capture correlated visibility samples, and spatial and temporal postfiltering primarily target edges of superpixel primitives.

We describe a novel semi-analytic representation of coverage designed to make progress on geometric antialiasing for subpixel primitives and pixels containing many edges while handling correlated subpixel coverage. Although not yet fast enough to deploy, it crosses three critical thresholds: image quality comparable to 256× MSAA, faster than 64× MSAA, and constant space per pixel.

CCS CONCEPTS

• **Computing methodologies** → **Visibility**; Rasterization;

KEYWORDS

Visibility, Filtering, Antialiasing

ACM Reference Format:
Cyril Crassin, Chris Wyman, Morgan McGuire, and Aaron Lefohn. 2018. Correlation-Aware Semi-Analytic Visibility for Antialiased Rendering. In *HPG '18: High-Performance Graphics, August 10–12, 2018, Vancouver, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3231578.3231584

1 INTRODUCTION

The aliasing artifacts of flicker, jaggies, and edge crawl in rendered images arise due to undersampling of geometry, materials, and shading. Important progress on reducing these artifacts has allowed the industry to produce useful real-time applications, but no current approach seems likely to solve the underlying aliasing problem while also scaling with scene complexity.

GPU-based supersampling is limited to eight samples per pixel, as adding samples linearly increases space and cost with diminishing returns. Analytic solutions have inherent performance limits [Auzinger et al. 2013; Catmull 1978] or restrict the geometry [Loop and Blinn 2005; Manson and Schaefer 2013]. The best temporal post-processing fails on subpixel features and overblurs large ones [Karis 2014] as it builds on aliased inputs. Prefiltering methods [Crassin et al. 2009; Lacewell et al. 2008; Neyret 1998; Qin et al. 2014] often filter poorly, especially for correlated visibility (e.g., objects lined up in depth), and require redesigning scene data structures.

Given aliasing's significance the difficulty existing techniques have with severely minified geometry, we believe it requires a fresh start, even if initial prototypes do not achieve the performance required by today's games. We isolate geometric aliasing and propose a radical approach, resurrecting the classic A-buffer and making it scalable. A-buffering has two logical parts: accumulate fragments, and then resolve them. A scalable (i.e., constant space and linear time) method must perform those simultanously as an on-line algorithm. The resolve step determines image quality, which is a prerequisite for relevance.

This short paper introduces a novel, scalable, constant-space, linear-time *resolve* step. For simplicity and flexibility, our strawman

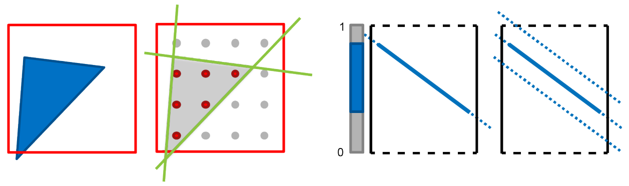


Figure 2: We represent geometry with (left) analytic coverage and filtered color, (center left) a sampled coverage mask, and (center right) a depth range and depth plane. After merging primitives, we store (right) a *depth slab* with an aggregate plane and bounds containing all merged primitives.

accumulation step presents fragments in depth order using an A-Buffer [Carpenter 1984]. Our results show achieved image quality comparable to 64 – 256× supersampling. We thus propose this as a promising, if initial, step to some ultimate visibility algorithm. Preprocessing to eliminate the A-buffer storage, fragment sort, and enable fully streaming triangle processing is important future work.

Our key differences from prior work include:

- We use a discrete fragment coverage mask, but *only* to determines if the our fragment is correlated, anti-correlated, or uncorrelated with existing pixel geometry,
- We also use an analytic coverage, allowing us to accumulate subpixel triangles finer than our coverage mask,
- Unlike prior techniques, our geometric representation for pixel aggregates allows us to merge interpenetrating primitives. We assume uniform geometric distribution where we are uncertain of correlation behavior.

2 ALGORITHM

Our resolve algorithm extends the ideas of discrete coverage masks, previously used for antialiasing [Crassin et al. 2015; Jouppi and Chang 1999; Kerzner and Salvi 2014; Wang et al. 2015], order-independent transparency [Wyman 2016], and fragment aggregation methods [Crassin et al. 2015; Salvi et al. 2011; Salvi and Vidimče 2012]. These representations alone: do not degrade gracefully when processing geometry finer than the coverage mask, decoupling depth and coverage often fails when triangles interpenetrate, and previous aggregates fail on correlated visibility. However, combining aspects of these algorithms helps address their limitations.

2.1 Geometric Representation

As shown in Figure 2, we represent each fragment with:

α: analytic scalar coverage computed from edge equations
M: coverage bit mask from table lookup [Waller et al. 2000]
S: oriented slab (initially just the triangle's plane) expressed as plane coefficients and thickness
$z_{\min}, z_{\max}$: depth range within the pixel
C: color, assumed to exhibit no shading or material aliasing

Recall that we target just geometric aliasing, combining fragment colors based on primitive visibility within the pixel. We store two coverage measures: our analytic value has exact magnitude while our mask encodes spatial distribution. Analytic coverage represents the normalized area of the triangle clipped to the pixels. When intergrating visibility, we assume this analytic area is uniformly distributed among the covered discrete mask samples.

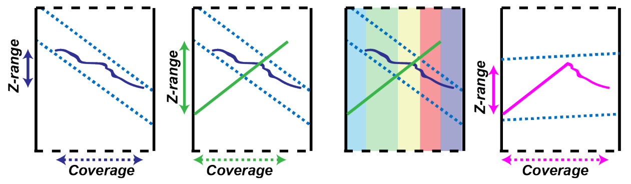


Figure 3: When merging, we start with an aggregate of prior fragments (left), with coverage, depth slab, and z-range. An incoming fragment (center left) has its own coverage, depth plane, and z-range. We identify regions of no overlap (blue), where the fragment is closer (green) or further (red) than the aggregate, or the two may interpenetrate (yellow). After applying our merge rules (right), we get a new aggregate.

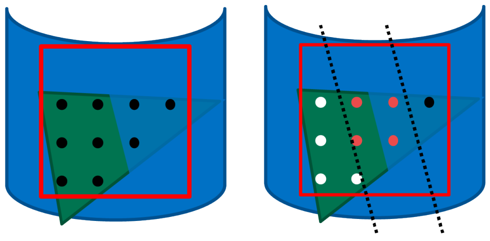


Figure 4: After identifying overlapping samples (left), we find those in front, behind, and interpenetrating our aggregate (right). To partition samples, we intersect the fragment *z*-plane with the front and back of the aggregate depth slab and project these intersections onto the pixel.

To achieve constant per-pixel storage, in anticipation of a future streaming algorithm, all fragments at a pixel are aggregated into a single representation as they pass through the resolve step. Thus, the fragment and the aggregate pixel representation are the same; we use the subscript 'a' to distinguish aggregate properties.

2.2 Merging a Fragment

The resolve operation simply merges each fragment in turn into a pixel's aggregate. The final pixel color is C'_a, the final aggregate color. The merge operation for a fragment comprises four steps:

1. Partition xy into fragment-distinct, aggregate-distinct, and overlapping subpixel regions by the coverage masks M
2. Within the overlapping xy, further partition into three z regions by comparing the slabs S and depth ranges: strictly in front (over), strictly behind (under), and [potentially]-interpenetrating
3. Separately merge the xy regions and the z-interpenetrating, z-over, and z-under fragment regions with the aggregate
4. Combine the independently-merged regions into an updated representation of the aggregate

Partition via coverage. We compute the xy-overlap between fragment i and the aggregate by bitwise-ANDing their coverage masks (M_i & M_a). The fragment-distinct region is (M_i & $\sim M_a$), the aggregate-distinct region is ($\sim M_i$ & M_a), and the uncovered region is ($\sim M_i$ & $\sim M_a$).

Partition overlap. In overlapping *xy*-regions, we further classify subpixel samples (see Figure 4). Samples can clearly lie in front or behind the aggregate. But fragment samples can also lie within the aggregate's depth slab, where we cannot identify which is closest. In that case, we assume an uniform distribution of aggregate geometry inside the slab, and we classifies those samples as *fuzzy*.

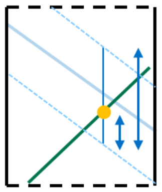
Merge separate regions. After partitioning our discrete samples, we blend their contributions together. Within each region, we compute the merged color as:

$$C'_{\mathrm{a}} = \frac{C_{\mathrm{a}} \cdot b_{\mathrm{a}} + C_i \cdot b_i}{b_{\mathrm{a}} + b_i},$$

For blending factor b, that we compute differently in each region:

- Only covered by the fragment: $b_i = 1$, $b_{\mathrm{a}} = 0$
- Only covered by the aggregate: $b_i = 0$, $b_{\mathrm{a}} = 1$
- Fragment over the aggregate: $b_i = \alpha_i^l$, $b_{\mathrm{a}} = \alpha_{\mathrm{a}}^l(1 - \alpha_i^l)$
- Aggregate over the fragment: $b_i = \alpha_i^l(1 - \alpha_{\mathrm{a}}^l)$, $b_{\mathrm{a}} = \alpha_{\mathrm{a}}^l$
- Interpenetrating regions: $b_i = \alpha_i^l(1 - \Delta)$, $b_{\mathrm{a}} = \alpha_{\mathrm{a}}^l\Delta$.

Here $\alpha^l = \alpha \times \frac{32}{|M|}$ represents the fragment- or aggregate- local analytic coverage value. It is the global α value of the fragment or aggregate normalized by the ratio of the coverage bits enabled in the associated mask M to the total coverage bits in our mask (we use a 32-bit coverage mask).



Δ is the degree of interpenetration, describing the relative fragment position within the aggregate's depth slab. We compute fragment i's centroid within the fuzzy region. We compute the triangle's depth and aggregate slab extents at this centroid. Then Δ represents a ratio of the relative distances, shown at right.

Combining regions. After aggregating in each region, we combine into our new aggregate using the regions' relative coverages:

$$C'_{\mathrm{a}} = \sum w_j C'_j, \qquad \alpha'_{\mathrm{a}} = \sum w_j \alpha'_j,$$

For j iterating over our 5 subpixel regions, and w_j being the ratio of coverage bits in a region to the total coverage bits in our mask.

Our new aggregate mask is $(M_i \mid M_{\mathrm{a}})$. For the new aggregate's depth slab, the plane is a final coverage-weighted blend of the prior planes. The slab extent depends on the maximal distance from the fragment to the combined plane (or the input slab extent, if larger).

2.2.1 Handling Tiny Triangles. When a triangle covers no samples in its bitmask, but has non-zero analytic area, we activate a single sample in the mask (closest to the triangle's center position), in order to locate the analytic coverage. To handle connected tiny triangles, we also assume anti-correlation over the overlapping regions (instead of de-correlation): $b_i = \alpha_i^l$, $b_{\mathrm{a}} = \min\{\alpha_{\mathrm{a}}^l, 1 - \alpha_i^l\}$

2.3 Rendering With This Representation

Above we described our subpixel aggregate and how to merge with incoming fragments. But the order of incoming fragments also matters. Since we aggregate into a scalar visibility, this behaves similar to order-independent transparency.

To simplify development, we started by assuming our fragments arrive in sorted order. We first create an A-buffer [Carpenter 1984], than traverse each pixel's fragments from front-to-back, sequentially merging each into our aggregate.

Obviously, this simplifies the problem. But order-independent transparency research has shown that multilayer techniques [Salvi and Vaidyanathan 2014] can closely approximate out-of-order primitives even when the merge operator relies on sorted geometry.

We have a promising initial multilayer prototype using multiple aggregates, but have not robustly addressed all corner cases or yet fully optimized performance bottlenecks.

3 RESULTS

We evaluate image quality on scenes exhibiting different characteristics in terms of geometric distributions. The classroom in Figure 1 exhibits heterogeneous density with some pixels containing over 50 fragments, due to highly-tessellated geometry. Often such fragments arise from connected triangles in a mesh, so they have similar depth ranges with perfect anti-correlation. We get comparable quality to 256× multisampling but render 7× faster, even with our naive accumulation and fragment sort. One of the hardest cases in this scene is the draw string on the window blinds, which is modeled as separate braided subpixel strands of thread. Absolute performance for this scene at 720*p* on Titan V is 30.2ms for our method, 198ms for 256× MSAA, 78ms for 128× MSAA, and 24ms for 32× MSAA. For fair comparisons, we implemented those high MSAA rates using a combination of super-resolution and 8× hardware MSAA.

Figure 5 (top) shows a uniformly high-density hairball, where all triangles are subpixel. The challenge here is capturing the net opacity versus the background. Hardware 8× MSAA fails for this; at least 64× supersampling is needed to converge.

The boat in Figure 5 (bottom) has thin details on the rigging and around windows, but more complex materials and shading than the hairball. It requires 256× supersampling for convergence of the shading on the braided ropes. Our accompanying video demonstrates our analytic integration provides extremely stable results under motion.

The total memory required for per-pixel aggregate storage is 32 bytes per pixel, which is less than an 8× MSAA buffer. In total, our analytic method approaches the quality of 256× MSAA with only the coverage mask requirement of 32×/memory requirement of 8× MSAA and runs faster than 64× MSAA. We therefore conclude that our method shows promise for tackling geometric aliasing under extreme minification.

4 DISCUSSION

This short paper demonstrates a promising new approach to solving the fundamental geometric aliasing problem for rasterization.

To enable fast iteration while developing our resolve step, we relied on an A-buffer rather than streaming fragments in primitive order. This imposed a significant performance overhead in exchange for simplifying aggregation due to a guarantee of ordered inputs. The next stage for geometric aliasing is to eliminate the A-buffer, either by making the aggregation process more robust to ordering or by prefiltering the primitives themselves *before* rasterization, effectively pushing the aggregation process into the geometric model as a new form of level of detail or coverage representation.

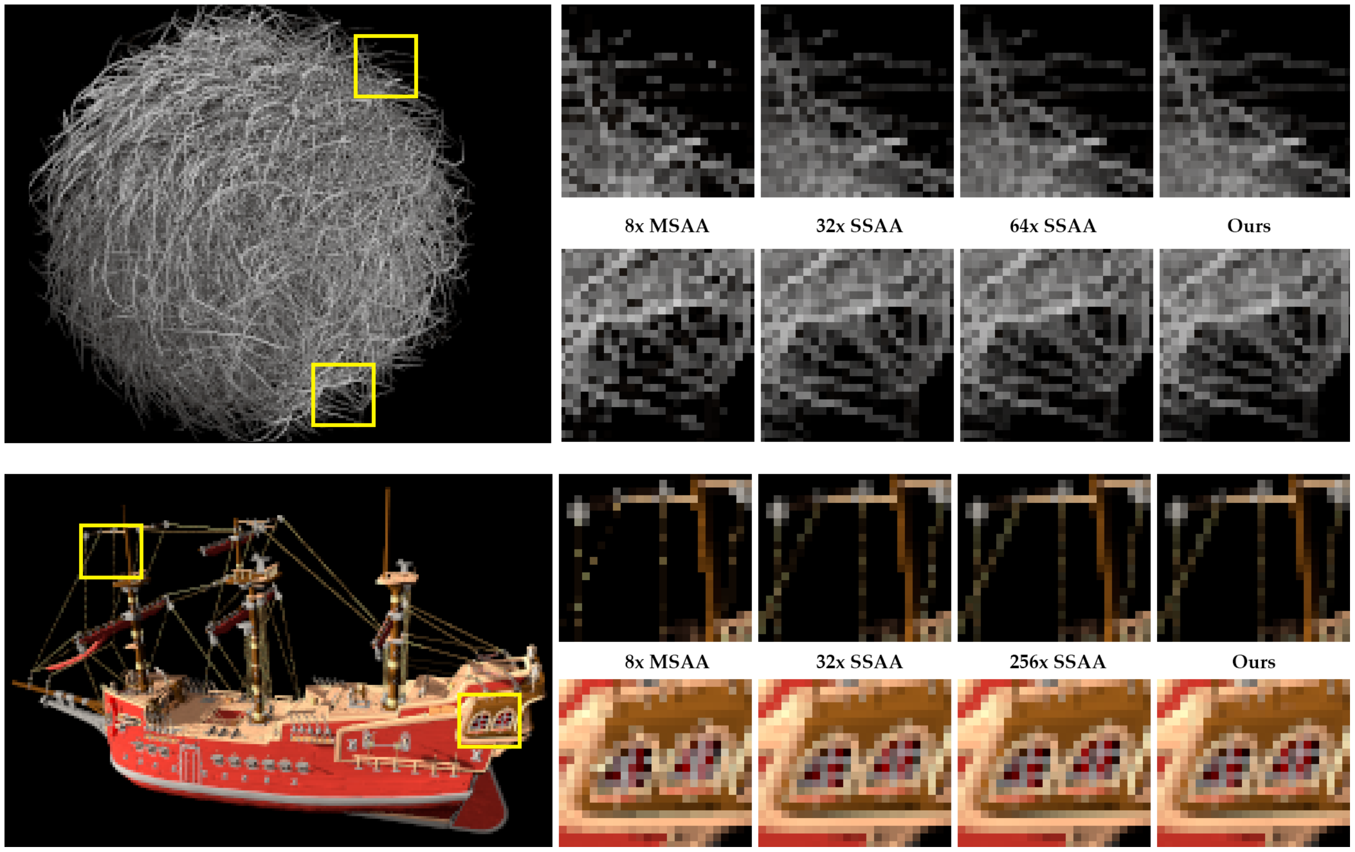


Figure 5: *Suzanne's Revenge* (courtesy of Greg Zaal, *blendswap.com*) and hairball rendered using our visibility representation, at low resolution to exacerbate aliasing. 8x MSAA is the best natively supported by GPUs, 32x MSAA has same coverage bit as our method, at 64x MSAA our prototype is already faster than multisampling, and 256x MSAA is our target quality.

Due to shader level of detail and MIP-maps, shading and material aliasing are less urgent problems than geometric aliasing. However, as all current solutions there are biased towards blur or energy loss, they also remain important open areas for future work.

While our representation using coverage masks helps address leaking due to correlations of coverage, it doesn't fully solve it, partly due to precision issues with the handling of intersections and partly due to the assumption of uniform distribution of analytic coverage inside covered sampled.

REFERENCES

Thomas Auzinger, Przemyslaw Musialski, Reinhold Preiner, and Michael Wimmer. 2013. Non-Sampled Anti-Aliasing. In *Proceedings Vision, Modeling and Visualization.* 169–176.

Loren Carpenter. 1984. The A-buffer, an antialiased hidden surface method. In *Proceedings of SIGGRAPH.* 103–108.

Edwin Catmull. 1978. A Hidden-surface Algorithm with Anti-aliasing. In *Proceedings of SIGGRAPH.* 6–11.

Cyril Crassin, Morgan McGuire, Kayvon Fatahalian, and Aaron Lefohn. 2015. Aggregate G-buffer Anti-aliasing. In *Symposium on Interactive 3D Graphics and Games.* 109–119.

Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. 2009. GigaVoxels : Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. In *Symposium on Interactive 3D Graphics and Games.*

Norman P. Jouppi and Chun-Fa Chang. 1999. Z3: An Economical Hardware Technique for High-quality Antialiasing and Transparency. In *Graphics Hardware.* 85–93.

Brian Karis. 2014. High-quality temporal supersampling. In *SIGGRAPH Course Notes: Advances in Real-Time Rendering in Games.*

E. Kerzner and M. Salvi. 2014. Streaming G-buffer Compression for Multi-sample Anti-aliasing. In *Proceedings of High Performance Graphics.* 1–7.

D. Lacewell, B. Burley, S. Boulos, and P. Shirley. 2008. Raytracing prefiltered occlusion for aggregate geometry. In *Symposium on Interactive Ray Tracing.* 19–26.

Charles Loop and Jim Blinn. 2005. Resolution independent curve rendering using programmable graphics hardware. *ACM Transsactions on Graphics* 24, 3 (2005), 1000–1009.

Josiah Manson and Scott Schaefer. 2013. Analytic Rasterization of Curves with Polynomial Filters. *Computer Graphics Forum* 32, 2 (2013), 499–507.

Fabrice Neyret. 1998. Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures. *IEEE Transactions on Visualization and Computer Graphics* 4, 1 (Jan. 1998), 55–70.

Hao Qin, Menglei Chai, Qiming Hou, Zhong Ren, and Kun Zhou. 2014. Cone Tracing for Furry Object Rendering. *IEEE Transactions on Visualization and Computer Graphics* 20, 8 (Aug. 2014), 1178–1188.

Marco Salvi, Jefferson Montgomery, and Aaron Lefohn. 2011. Adaptive Transparency. In *Proceedings of High Performance Graphics.* 119–126.

Marco Salvi and Karthik Vaidyanathan. 2014. Multi-Layer Alpha Blending. In *Proceedings of the Symposium on Interactive 3D Graphics and Games.* 151–158.

Marco Salvi and Kiril Vidimče. 2012. Surface Based Anti-aliasing. In *Symposium on Interactive 3D Graphics and Games.* 159–164.

Marcus D. Waller, Jon P. Ewins, Martin White, and Paul F. Lister. 2000. Efficient Coverage Mask Generation for Antialiasing. *IEEE Comput. Graph. Appl.* 20, 6 (Nov. 2000), 86–93.

Yuxiang Wang, Chris Wyman, Yong He, and Pradeep Sen. 2015. Decoupled Coverage Anti-aliasing. In *High-Performance Graphics.* 33–42.

Chris Wyman. 2016. Exploring and Expanding the Continuum of OIT Algorithms. In *High Performance Graphics.* 1–11.