Adaptive Temporal Antialiasing

Adam Marrs
NVIDIA

Josef Spjut
NVIDIA

Holger Gruen
NVIDIA

Rahul Sathe
NVIDIA

Morgan McGuire
NVIDIA

Figure 1: A modern house scene in Unreal Engine 4 with deferred shading, ray traced shadows, our adaptive temporal antialiasing technique, and a moving camera at 30 fps on NVIDIA Titan V. Zoomed details show raw 1 sample per pixel raster input, standard TAA output, a visualization of our segmentation mask, and our ATAA 8x result.

ABSTRACT

We introduce a pragmatic algorithm for real-time adaptive supersampling in games. It extends temporal antialiasing of rasterized images with adaptive ray tracing, and conforms to the constraints of a commercial game engine and today's GPU ray tracing APIs. The algorithm removes blurring and ghosting artifacts associated with standard temporal antialiasing and achieves quality approaching 16× supersampling of geometry, shading, and materials while staying within the 33ms frame budget required of most games.

CCS CONCEPTS

• **Computing methodologies** → **Ray tracing**;

KEYWORDS

adaptive sampling, supersampling, ray tracing

ACM Reference Format:
Adam Marrs, Josef Spjut, Holger Gruen, Rahul Sathe, and Morgan McGuire. 2018. Adaptive Temporal Antialiasing. In *HPG '18: High-Performance Graphics, August 10–12, 2018, Vancouver, Canada,* Anjul Patney and Matthias Niessner (Eds.). ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3231578.3231579

HPG '18, August 10–12, 2018, Vancouver, Canada

1 INTRODUCTION AND RELATED WORK

Aliasing of primary visible surfaces is one of the most fundamental and challenging limitations of computer graphics. Almost all rendering methods sample surfaces at points within pixels, and thus produce error when the points sampled are not representative of the pixel as a whole, that is, when primary surfaces are undersampled.

Analytic renderers could avoid the ray (under)-sampling problem, but despite some analytic solutions for limited cases [Auzinger et al. 2013] point samples from ray or raster intersections remain the only fully-developed approach for efficient rendering of complex geometry, materials, and shading. Aliasing due to undersampling manifests as jagged edges, spatial noise, and flickering (temporal noise). Attempts to conceal those errors by wider and more sophisticated reconstruction filters in space (e.g., MLAA [Reshetov 2009], FXAA [Lottes 2009]) and time (e.g., SMAA [Jimenez et al. 2012], TAA [Karis 2014]) convert those artifacts into blurring or ghosting.

Under a *fixed sample* count per pixel across an image, the only true solution to aliasing is to increase the sample density and band-limit the signal being sampled. Increasing density helps but does not solve the problem at rates affordable for real-time: supersampling (SSAA) incurs a cost linearly proportional to the number of samples while only increasing quality with the square root; multisampling (MSAA, CSAA, SBAA [Salvi and Vidimče 2012], SRAA [Chajdas et al. 2011]) samples geometry and materials and shading at varying rates to heuristically reduce the cost but also lowers quality; and aggregation (DCAA [Wang et al. 2015], AGAA [Crassin et al. 2016]) reduces cost even more aggressively but still limits quality at practical rates. Material prefiltering by mipmapping and its variants, level of detail for geometry, and shader level of detail can

improve undersampling by band-limiting the scene, but complicate rendering systems without fully solving the problem.

The standard in real-time rendering is to employ many of the aforementioned strategies simultaneously. Despite succeeding in many cases, these game-specific solutions require significant engineering complexity and careful hand-tuning of scenes by artists [Pedersen 2016; Pettineo 2015]. Since all these solutions depend on a *fixed* sampling count per pixel, an adversary can always place material, geometric, or shading features between samples to create unbounded error. Thus we consider the aliasing challenge open for real-time rendering. Offline ray-traced renderers have long employed highly *adaptive* sample counts to solve aliasing (e.g. Whitted's original paper [Whitted 1980]). More recently, Holländer et. al. [Holländer et al. 2013] aggressively identify pixels in need of antialiasing from coarse shading and high resolution geometry passes and achieve nearly identical results to SSAA with 10% reduced frame time despite cutting the number of shading samples in half. Importantly, rasterization requires processing all geometry, even if only a few pixels are affected.

We describe a new method for practical adaptive sampling in real time using a hybrid of ray tracing and rasterization. This method is enabled by the recently released DirectX Ray Tracing API (DXR) and NVIDIA RTX. Although hybrid algorithms have been proposed for heterogeneous architectures [Barringer and Akenine-Möller 2013], they have been impractical due to duplication of data structures between ray and raster APIs. DXR and RTX enable full interoperability between data structures and shaders for both types of rendering on the GPU across the full game engine for the first time.

We build on the common idea of adaptive sampling by showing how to *efficiently* combine these techniques for modern graphics hardware. We show how to leverage adaptive sampling within the context of temporal antialiasing to still amortize the cost of rasterized samples in time without creating blurring or ghosting.

2 PREVIOUS TEMPORAL ANTIALIASING

Temporal antialiasing [Karis 2014] (TAA) is fast and quite good in the cases it can handle, which is why it is the de facto standard for games today. TAA applies a subpixel shift to the image plane each frame and accumulates an exponentially-weighted moving average over previous frames, each of which was rendered with only one sample per pixel. On static scenes, this approaches the quality of full screen supersampling. For dynamic scenes, TAA 'reprojects' samples from the accumulated history buffer by offsetting texture fetches along per-pixel motion vectors generated by the rasterizer.

TAA fails in several cases. When new screen areas are disoccluded (revealed) by object motion, those are not represented in the history buffer, or are misrepresented by the motion vectors. Camera rotation and backwards translation also create thick disocclusions at the edges of the screen. Subpixel features such as wires and fine material details can slip between consecutive offset samples and thus are unrepresented by motion vectors in the next frame. Transparent surfaces create pixels at which the motion vectors from opaque objects do not match the total movement of represented objects. Finally, shadows and reflections do not move in the direction of the motion vectors of the surfaces that are shaded by them.

When TAA fails, it either produces ghosting (blurring due to integrating incorrect values) or reveals the original aliasing as jaggies, flicker, and noise. Standard TAA attempts to detect these cases by comparing the history sample to the local neighborhood of the corresponding pixel in the new frame. When they appear too different, TAA employs a variety of heuristics to clip, clamp, or interpolate in color space. As summarized by Salvi [2015], the best practices for these heuristics change frequently and no general purpose solution has previously been found.

3 A NEW ALGORITHM

We designed our method for compatibility with conventional game engines and to harness the strengths of TAA, while addressing its failures unequivocally and simply. The core idea is to run the base case of TAA on most pixels and then, rather than attempting to combat its failures with heuristics, output a conservative segmentation mask of where it will fail, and why. We then replace the complex heuristics of TAA at failure pixels with robust alternatives, adapting to the image content.

Figure 2 shows our algorithm in the context of Unreal Engine 4 (UE4) extended to support the DXR API. In the diagram, rectangular pictures represent visualizations of buffers and rounded rectangles represent shader passes. Not all intermediate buffers are shown. For example, where the previous frame's output feeds back as input to TAA, we do not show the associated ping-pong buffers.

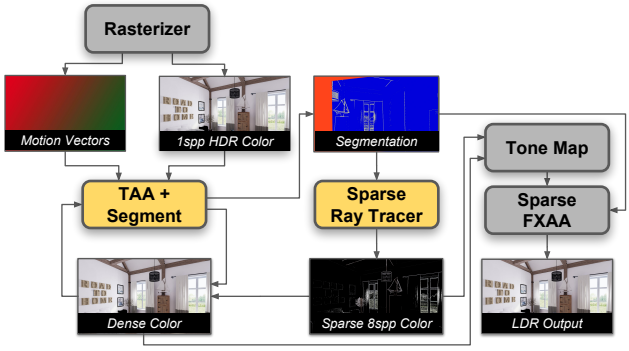

Figure 2: The data flow of our algorithm integrated into the UE4 rendering pipeline. The gray passes are only slightly modified. Most connections and the yellow passes are new. In the Segmentation mask, red pixels will use FXAA, blue pixels will use TAA, and yellow pixels will be ray traced

The image labeled 'Segmentation' in Figure 2 is a visualization of the segmentation mask. Red pixels identify disocclusions, especially around the edges of the screen where data from previous frames is not present. We process these areas with FXAA, since it has a low cost, requires no historical data, and runs on the low dynamic range post-tonemapped output to conserve memory bandwidth. By running FXAA only at disoccluded pixels, we further reduce its cost compared to full-screen applications; typically to less than 15% even for rapid object and camera movement. Blue pixels represent areas where our segmentation strategy determines the existing TAA result is acceptable. Finally, yellow pixels represent areas where our segmentation heuristic detects a high chance of

TAA failure. In practice, the segmentation mask is stored as two half-precision unsigned integer values packed into a single 32-bit memory resource. The first integer identifies a pixel's AA method (0 - FXAA, 1 - TAA, 2 - ATAA), and the second integer serves as a pixel classification history that stores if a pixel has received ray tracing via ATAA in previous frames.

The segmentation mask is generated during the full-screen TAA post-process pass in UE4. To detect TAA failures, we use a combination of criteria. First, motion vectors are inspected to determine if the the current pixel was previously occluded. The result of the motion vector comparison overrides all other criteria and can trigger an early exit in the detection process. Next, the pixel classification history is inspected to determine if a pixel has been recently ray traced. If so, the pixel will continue to be classified for ray tracing over the next few frames. This approach reduces flicker from rapid shifts of yellow pixels in the segmentation mask. Significant changes in a pixel's motion vectors will reset the pixel classification history. Pixel classification history can also trigger an early exit. Next, we compute the temporal change in luminance by inspecting a small neighborhood of pixels in the current and previous frames. Last, we find the change in depth values in a 3x3 pixel neighborhood of the current frame using an edge detecting Sobel filter, and compare the magnitude of the Sobel gradient against a threshold. Depth and temporal luminance are weighted equally.

At pixels marked for ATAA, we cast rays in either the 8×, 4×, or 2× MSAA *n*-rooks subpixel sampling patterns. Ray casts are spawned from a DXR Ray Generation Shader and do not use temporal jittering; however, we plan to explore this in future work. At ray hits we execute the full UE4 node-based material graph and shading pipeline, using identical HLSL code to the raster pipeline. Since forward-difference derivatives are not available in DXR Ray Generation Shaders, we treat them as infinite to force the highest resolution of textures. Thus, we rely on supersampling alone to address material aliasing (which is how most film renderers operate, for the highest quality); an alternative would be to use distance and orientation to analytically select a mipmap level, or to employ ray differentials [Christensen et al. 2003; Igehy 1999].

Since frames are almost always dominated by blue-classified (TAA) pixels, the cost of ray tracing is highly amortized and requires a ray budget far less than one sample per pixel. For example, we can adaptively employ 8× ray traced supersampling for 6% of the scene at a cost of fewer than 0.5 rays per pixel. The quality is then comparable to 8× supersampling everywhere; were it not, the boundaries between segmented regions would flicker in the final result due to different algorithms being employed.

4 RESULTS

We implemented the ATAA algorithm in Unreal Engine 4 and gathered results using Windows 10 v1803, Microsoft DXR, the NVIDIA RTX enabled 398.11 driver, and a NVIDIA Titan V GPU. To demonstrate the image quality achievable with ATAA, Figure 3 shows a comparison of ATAA and other common antialiasing algorithms used in games, zoomed to challenging areas of the scene.

The *No AA* row demonstrates the baseline aliasing that is expected from a single raster sample per pixel. The *FXAA* and *TAA* rows are the standard implementations available in UE4. *SSAA 4x*

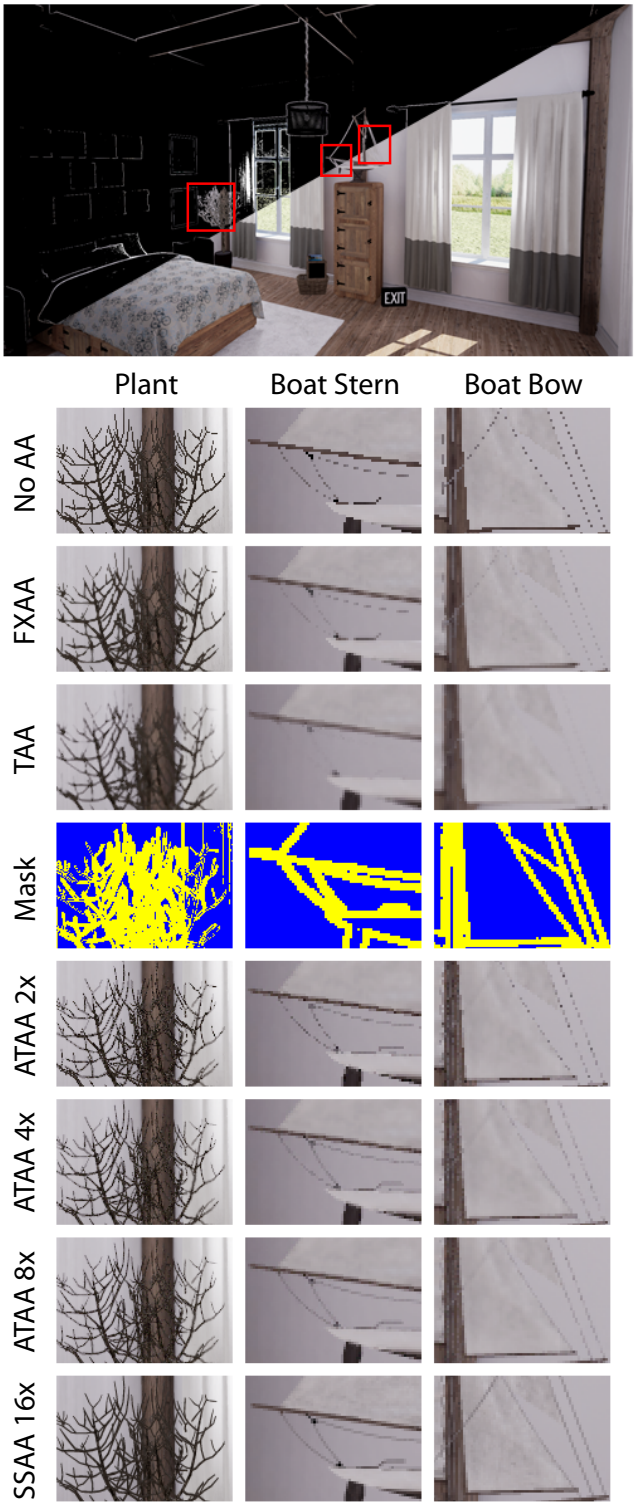

Figure 3: A zoomed comparison of ATAA with other common AA algorithms used in games. The diagonal cutout in the top image shows the contribution of ATAA's sparse ray tracing. Images captured at 1080p.

is 4× supersampling. We show the segmentation mask and three variations of ATAA with 2, 4, and 8 rays per pixel. Since the drawbacks of standard TAA are difficult to capture in still images, and all images in Figure 3 come from a stable converged frame, the supplemental video provides a more faithful comparison between standard TAA and ATAA in practice, including motion artifacts.

The 'Plant' inset column of Figure 3 shows regions where TAA misses or blurs out thin geometry that falls in the sub-pixel area between samples. ATAA's segmentation step identifies much of the region surrounding these tough areas and avoids undersampling by ray tracing. The 'Boat' inset columns demonstrate how ATAA produces antialiased curves from the boat geometry, and fills in areas where the sub-pixel geometry of the thin ropes are either missed (FXAA) or blurred until they are no longer visible (TAA). There are minor differences in ATAA's antialiased result caused by the material evaluation in DXR not correctly computing and evaluating texture mipmap level, while TAA uses screen-space derivatives provided by the rasterizer to compute mipmap level. Methods to perform mipmap level selection are well known; however, are difficult to implement in practice for arbitrary material graphs containing multiple dependent textures of varying resolution.

On a NVIDIA Titan V GPU at 1920×1080 resolution, ATAA runs in 18.4ms at 8× supersampling, 9.3ms at 4× supersampling, and 4.6ms at 2× supersampling for the image in Figure 1. This includes the creation of the standard TAA result, our segmentation mask, and adaptive ray tracing (including 1 shadow ray per light per primary ray). For the view shown in Figure 3, 107,881 pixels are selected for adaptive ray tracing, representing 5.2% of the total image resolution. The specific number of rays identified for antialiasing varies per frame according to the segmentation mask. In addition, the FXAA pass adds as much as 0.75ms when the whole frame is new, but in practice scales linearly down to 0 as fewer of the pixels are identified for FXAA in the mask. Under typical camera motion fewer than 5% of pixels are selected for FXAA. Our ATAA solution integrated successfully operates within the 33 millisecond frame budget for a typical UE4 frame across all settings. Operating within a total frame budget of 16ms, while also ray tracing 1spp shadows at screen resolution, is possible with the 2× and 4× ATAA variants. As DXR is an experimental feature of Windows 10 v1083, we are optimistic that performance will improve as the runtime and driver receive important release optimizations.

5 CONCLUSIONS

Primary surface aliasing is a cornerstone problem in computer graphics. The best known solution for offline rendering is adaptive supersampling. This was previously impractical for rasterization renderers in the context of complex materials and scenes because there was no way to efficiently rasterize sparse pixels. Even the most efficient GPU ray tracers required duplicated shaders and scene data. While DXR solves the technical challenge of combining rasterization and ray tracing, applying ray tracing to solve aliasing by supersampling was nontrivial: knowing *which* pixels to supersample when given only 1spp input, and reducing the cost to something that scales are not solved by naively ray tracing.

We have demonstrated a practical solution to this problem; so practical that it runs within a commercial game engine, operates in real-time even on first-generation real-time ray tracing commodity hardware and software, and connects to the full shader pipeline. Where film renderers choose pixels to adaptively supersample by first casting many rays per pixel, we instead amortize that cost over many frames by leveraging TAA's history buffer to detect aliasing. We further identify large, transient regions of aliasing due to disocclusions and employ post process FXAA there rather than expending rays. This hybrid strategy leverages advantages of the most sophisticated real-time antialiasing strategies but avoids their limitations. By feeding our supersampled results back into the TAA buffer, we also increase the probability that those pixels will not trigger supersampling on subsequent frames, further reducing cost.

Our method's performance is dominated by the ray trace. We cannot advocate it for immediate wide-spread deployment in games at current performance, but that is not concerning given that mainstream gaming GPUs have not yet appeared that support the DXR API. The real-time ray tracing ecosystem of drivers, GPUs, and algorithms must emerge together over the next few years.

ACKNOWLEDGMENTS

We wish to thank Ignacio Llamas, Edward Liu, and the entire ray tracing team at NVIDIA for their help and feedback.

REFERENCES

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

Rasmus Barringer and Tomas Akenine-Möller. 2013. A4: Asynchronous Adaptive Anti-aliasing Using Shared Memory. *ACM Trans. Graph.* 32, 4, Article 100 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2462015

Matthäus G. Chajdas, Morgan McGuire, and David Luebke. 2011. Subpixel Reconstruction Antialiasing for Deferred Shading. In *Symposium on Interactive 3D Graphics and Games (I3D '11)*. ACM, New York, NY, USA, 15–22 PAGE@7. https://doi.org/10.1145/1944745.1944748

Per H Christensen, David M Laur, Julia Fong, Wayne L Wooten, and Dana Batali. 2003. Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes. In *Computer Graphics Forum*, Vol. 22. Wiley Online Library, 543–552.

Cyril Crassin, Morgan Mcguire, Kayvon Fatahalian, and Aaron Lefohn. 2016. Aggregate G-Buffer Anti-Aliasing. 22 (06 2016), 1–1.

Matthias Holländer, Tamy Boubekeur, and Elmar Eisemann. 2013. Adaptive Supersampling for Deferred Anti-Aliasing. *Journal of Computer Graphics Techniques (JCGT)* 2, 1 (02 March 2013), 1–14. http://jcgt.org/published/0002/01/01/

Homan Igehy. 1999. Tracing ray differentials. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 179–186.

Jorge Jimenez, Jose I. Echevarria, Tiago Sousa, and Diego Gutierrez. 2012. SMAA: Enhanced Morphological Antialiasing. *Computer Graphics Forum (Proc. EUROGRAPHICS 2012)* 31, 2 (2012).

Brian Karis. 2014. High Quality Temporal Anti-Aliasing. (2014).

Tim Lottes. 2009. FXAA. (2009). http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf NVIDIA White Paper.

Lasse Jon Fuglsang Pedersen. 2016. Temporal Reprojection Anti-Aliasing in INSIDE. (2016).

Matt Pettineo. 2015. Rendering The Alternate History of The Order: 1886. (2015). SIGGRAPH Advances in Real-Time Rendering in Games Course.

Alexander Reshetov. 2009. Morphological Antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009 (HPG '09)*. ACM, New York, NY, USA, 109–116. https://doi.org/10.1145/1572769.1572787

Marco Salvi. 2015. Anti-Aliasing: Are We There Yet? (2015).

Marco Salvi and Kiril Vidimče. 2012. Surface Based Anti-aliasing. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '12)*. ACM, New York, NY, USA, 159–164. https://doi.org/10.1145/2159616.2159643

Yuxiang Wang, Chris Wyman, Yong He, and Pradeep Sen. 2015. Decoupled coverage anti-aliasing. (08 2015), 33-42 pages.

Turner Whitted. 1980. An Improved Illumination Model for Shaded Display. *Commun. ACM* 23, 6 (June 1980), 343–349. https://doi.org/10.1145/358876.358882