AXAA: Adaptive approXimate Anti-Aliasing

Jae-Ho Nah, Sunho Ki, Yeongkyu Lim, Jinhong Park, and Chulho Shin *
LG Electronics

Figure 1: *Anti-aliased images of the game Tera and the text SIGGRAPH. Compared to FXAA, AXAA improves text visibility, texture highlights, and geometry details. Additionally, AXAA smooths long edges as well as FXAA. In terms of performance, AXAA is up to 2.3× and 2.8× faster than CMAA and SMAA 1x, respectively. Image courtesy of Bluehole Studio.*

Abstract

Post-processing anti-aliasing algorithms are widely used now for real-time rendering because of their simplicity, performance, and suitability for deferred shading. Fast approximate anti-aliasing (FXAA) [Lottes 2009] is the fastest method among them, so many games support FXAA to get anti-aliased images. However, FXAA can easily lose texture details and text sharpness due to its excessive blurring.

To alleviate those problems of FXAA, we present adaptive approximate anti-aliasing (AXAA). Our approach adds three contributions to FXAA in order to avoid unnecessary filtering. First, we stop further anti-aliasing processes if the current pixel or its neighbors are judged as pixels on already filtered textures or fonts. Second, we try to maintain thin lines as much as possible in order to avoid blurring fonts and lines. Third, for higher performance, we adaptively set the search range of each pixel according to luma contrast. Our experiments show that AXAA provides significantly better image quality than FXAA, in terms of texture, text, and geometry details. Nevertheless, processing overhead of AXAA is still similar to that of FXAA.

Keywords: anti-aliasing

Concepts: •**Computing methodologies** → **Antialiasing; Image manipulation;** *Computational photography;*

*e-mail:jaeho.nah, sunho.ki, yk.lim, jinhong1.park, chulho.shin@lge.com

SIGGRAPH 2016 Posters, July 24-28, 2016, Anaheim, CA
ISBN: 978-1-4503-4371-8/16/07
DOI: http://dx.doi.org/10.1145/2945078.2945129

1 Post-Processing Anti-Aliasing and FXAA

Multi-sample anti-aliasing (MSAA) has been considered as a standard anti-aliasing technique. However, its high bandwidth requirements (especially in deferred shading) and performance penalty for multi-sampled frame buffers have resulted in an increased demand for alternative approaches. As a result, post-processing anti-aliasing (PPAA) is taking center stage as an attractive approach of MSAA. Because PPAA is an image-based solution that is independent of rendering pipelines, it is suitable for both forward and deferred rendering and is quite fast.

Morphological anti-aliasing (MLAA) introduced by Reshetov [2009] is one of the representative PPAA algorithms. MLAA detects discontinuities, classifies discontinued edges into Z, U, and L-shapes, and blends the edges with their neighborhood pixels. Enhanced subpixel morphological anti-aliasing (SMAA) [Jimenez et al. 2012] improves MLAA in various ways: accurate distance searches, local contrast adaption, extended patterns and geometric features detection, and combinations with temporal super-sampling and MSAA. Conservative morphological anti-aliasing (CMAA) [Davies 2014] takes a different approach; by separating locally dominant edges and long edge shapes, CMAA achieves both minimized shape distortion and smoothly anti-aliased edges.

In contrast to those multi-pass MLAA variations, fast approximate anti-aliasing (FXAA) [Lottes 2009] requires a single shader kernel (except for a simple luma calculation). Among various FXAA presets, the procedure of the FXAA 3.11 Quality preset is summarized as follows. First, the luma range of the current pixel and its neighbors is checked to extract a discontinued edge. Second, the dominant edge direction and the high contrast edge are determined using the neighborhood pixels. Third, both ends of the edge are searched. Finally, a sub-pixel offset is calculated and the texture coordinate is shifted by the offset for final blending. FXAA is usually faster than CMAA/SMAA, but its imperfect luma-based edge detection sometimes excessively blurs pixels and results in more blurry texts and textures than CMAA/SMAA.

2 Adaptive Approximate Anti-Aliasing

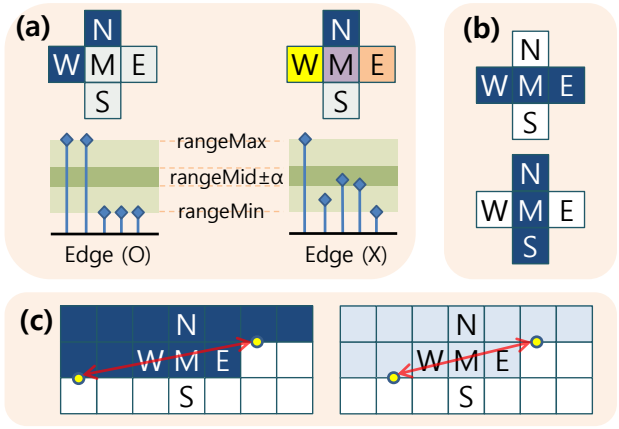

Figure 2: *Three main ideas of AXAA: (a) avoiding further processing of already filtered pixels, (b) maintaining contrast of thin lines, and (c) adaptive search ranges according to luma contrast.*

To increase image quality and performance of FXAA, we enhance FXAA in three ways. The core idea of our approach is to avoid unnecessary filtering of FXAA adaptively, so we call this approach adaptive approximate anti-aliasing (AXAA).

The detailed approach is described as follows. First, we establish an additional early-exit criterion to avoid duplicated anti-aliasing of already filtered pixels (Figure 2-(a)). This is possible by calculating additional median luma values (*rangeMid*); because image filtering blends neighbor pixels, the color (and luma) value of some pixel in an already filtered region is averaged over its neighbors. Thus, if we check whether the luma values of the current pixel or its neighbors are within the range of *rangeMid* $\pm\alpha$ or not, we can easily avoid blurring non-edge regions and increase visibility of textures and true-type fonts. Our setting of the alpha value is 10%.

Second, we conserve contrast of thin one-pixel-width lines because blurring thin lines dramatically decreases their visibility (Figure 2-(b)). Our approach is simple; if both differences between the current pixel and its two neighbor pixels in the dominant edge direction (*gradientN* and *gradientS* in the FXAA code) are higher than a specific value (currently 0.3), we set the sub-pixel offset to 0 in order to avoid bilinear filtering. This approach is especially effective for increasing visibility of small fonts and thin geometry.

Third, we adaptively set the search range of the current pixel according to its luma contrast (Figure 2-(c)). The wider the range (=more iterations) in FXAA, the higher the quality. However, search ranges in FXAA heavily affects performance because end-to-edge searches in FXAA are performed on all pixels that passed the early exit. We observe that if discontinued edges exist on low contrast regions, it is hard to distinguish the results between narrow and wide search ranges. Thus, we limit the search iteration according to luma contrast in order to increase performance. We limit the iteration count as follows:

$\max(dmin, dmax) \leq 0.1$: only 1 iteration
$\min(dmin, dmax) > 0.1$: 2+ iterations are available
$\min(dmin, dmax) > 0.3$: 3+ iterations are available,

where *dmin* and *dmax* are the differences between the current pixel's luma value and the minimum and maximum luma values in the region, respectively. The use of these adaptive search ranges compensates for overheads of the first and second approaches described above.

3 Experimental Results

Table 1: *Experimental results on laptop/mobile GPUs (unit: ms).*

Platform	Intel HD Graphics 4600	nVIDIA GeForce 840M	AMD Radeon 7650M	nVIDIA Tegra K1
Scene (API)	Tera (DirectX)			GameWorks FXAA (OpenGL)
SMAA 1x	10.6	4.2	9.6	5.9
CMAA	6.1	2.8	9.6	N/A
FXAA 3.11	5.3	2.5	4.2	2.0
AXAA	5.3	2.5	4.2	2.1

For our experiments, we used two scenes in Table 1: A scene in the game Tera (Figure 1) and the FXAA demo in nVIDIA GameWorks. We chose four different laptop/mobile platforms because PPAA costs are more important on low-end GPUs than on high-end GPUs. The fragment discard option for the FXAA/AXAA DirectX version was enabled, and the resolution was full HD. According to the results, AXAA achieves similar performance to FXAA, and it is 1.1×-2.3× and 1.7×-2.8× faster than CMAA and SMAA 1x, respectively. In terms of image quality, AXAA shows comparable results to CMAA and SMAA 1x, as illustrated in Figure 1. Thus, we believe that AXAA can be very attractive for laptop and mobile platforms.

4 Limitations and Future Work

The current parameters are optimal for our experimental scenes, so the current AXAA version may either miss jagged edges or blur non-edges. We would like to continuously investigate these difficult cases and improve image quality in the cases. Moreover, as other single-sample PPAA approaches, our approach does not properly handle sub-pixel problems and temporal aliasing. We think a combination with spatial multi-sampling and temporal super-sampling can be a solution, as with SMAA 4x.

Acknowledgments

We thank Leigh Davies, Timothy Lottes, and Jorge Jimenez for releasing their source codes. Additional images of Davies's synthetic tests [2014], Kishonti GFXBench Egypt, and Tera are included in the supplemental material for quality comparisons. We would like to thank Jin-Woo Kim and Byeongjun Choi for their advice to improve the quality of the poster.

References

DAVIES, L., 2014. Conservative morphological anti-aliasing (CMAA) - March 2014 update. Intel Technical Report, https://software.intel.com/en-us/articles/conservative-morphological-anti-aliasing-cmaa-update.

JIMENEZ, J., ECHEVARRIA, J. I., SOUSA, T., AND GUTIERREZ, D. 2012. SMAA: enhanced subpixel morphological antialiasing. *Computer Graphics Forum (EUROGRAPHICS 2012) 31*, 2pt1, 355–364.

LOTTES, T., 2009. FXAA. NVIDIA White Paper, http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf.

RESHETOV, A. 2009. Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009*, ACM, 109–116.