IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Edge Based Antialiasing: Quality without Compromise

Bineta Tresa Mathew

Department of Computer Science and Engineering, Birla Institute of Technology, Offshore Campus, RAK, UAE

Abstract

This paper discusses a new, more efficient technique for handling antialiasing of images. This technique is capable of overcoming the various problems experienced with current real-time antialiasing techniques. The various features associated with the new technique will be discussed in detail as well as how the new technique compares with existing techniques in real-time environments through tests in order to come to a conclusion regarding its performance capability. The purpose of this paper is to shed light on the advantages of this technique and prove that this technique is an efficient alternative to existing antialiasing techniques.

Keywords: AA, SSAA, MSAA, MLAA, FXAA.

Introduction

Antialiasing (AA) is the process whereby the jagged edges of a graphic can be smoothed to give a more pleasing and realistic appearance. The process in an intense processing technique and therefore faces many problems such as incorrect and dirty artifact production, flickering animations etc. which in turn require more processing power to fix. Many fixes such as increasing the sampling rates for example can help with clear image production but they are too costly to implement for real-time applications. Other fixes include development of completely new algorithms and complex shaders. However, AA has been a barrier when rendering images in real-time applications [1].

In its infant stages, the two most common go-to solutions were the *supersample antialising* (SSAA) and the *multisample antialising* (MSAA). These were heavily used in real-time applications especially video games. However, these techniques are not perfect. MSAA does not scale well when increasing the number of samples and is not completely compatible with modern environment rendering paradigms [2][3]. It exemplify this problem with numbers, MSAA 8x takes an average of 5.4 ms in modern video games with state of the art rendering engines (increasing to 7.7 ms on memory bandwidth intensive games) on a NVIDIA GeForce GTX 470. Memory consumption in this mode can be as high as 126MB and 316 MB, for forward and deferred rendering engines respectively, taking 12% and 30% of the rendering time of a mainstream GPU equipped with 1GB of memory. This problem is aggravated when HDR rendering is used, as the memory consumption and bandwidth increases even further.

Many alternatives approaches were considered but all of them had their own set of problems. The most prominent ones are:

- Most approaches do not deal with diagonal patterns, as they can recognize vertical and horizontal patterns.
- The aliasing effects on some shapes such as corners in text are clearly visible.
- Edge detection algorithms detect edges based on the difference in pixels without considering the surroundings of that edge.
- Subpixel features and Subpixel motions are not implemented/handled properly.

To deal with these issues the *Edge Based Antialiasing* (EBAA) was developed. This approach tackles each of the above mentioned problems separately, offering simple, modular solutions. First, for the edge detection algorithms, the number and type of edge patterns is increased. Second, multi/supersampling and temporal reprojection is combined with another technique called MLAA (which will be discussed in the coming section), so that the real subpixel features and subpixel motion can be handled effectively. Finally, for more accurate pattern classification, the edge detection algorithm is enhanced with better distance searches that include low contrast pixels as well.

The Root OF EBAA

The *Morphological Antialiasing* (MLAA) is a technique that estimates the pixel coverage of the original geometry [3]. To accurately rasterize an

antialiased triangle, the coverage area for each pixel inside the triangle must be calculated to blend it properly with the background. MLAA takes a non antialiased image and re-vectorizes the individual objects in the image pixel by pixel and then arranges the area with its neighbours. Figure 1 describes the process.

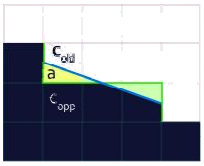

***Figure 1: MLAA first finds edges by looking for color discontinuities (green lines), and classifies them according to a series of pre-defined pattern shapes, which are then virtually re-vectorized* (blue line), *allowing to calculate the coverage areas* a *for the involved pixels. These areas are then used to blend with a neighbor. For example, the pixel* C_{opp} *fills the area* a *of the pixel* C_{old}: $c_{new} = (1-a) . c_{old} + a . c_{opp}$.**

Several MLAA antialising implementations appeared after the technique was published such as Jimenez's MLAA [3]. The algorithm has three passes: edge detection, pattern detection plus calculation of coverage areas, and final blending. Pattern detection is performed by locating the ends of an edge, and performing hardware bilinear filtering on them. Once the ends are reached, the algorithm looks at the crossing edges. These crossing edges are the perpendicular edges with respect to the direction of a search and are represented as the vertical green lines in Figure 1. With length and crossing edges information, the coverage area is retrieved, and used for the final blending. This MLAA implementation was chosen as the starting point for the EBAA algorithm which will be explained in detail in the coming sections.

The Various Features of EBAA

This section will detail the various features of EBAA and their implementation idea. EBAA is built on the MLAA implementation by Jimenez. First, the edge detection is improved by using general color information along with local contrast adaptation for recognizing and handling more edges. Then, as mentioned earlier, the number of patterns handled is extended and enhanced for a more reliable edge classification. Finally, MLAA is combined with multi/supersampling and temporal reprojection. Although, EBAA seems similar to MLAA, it performs much better and the images produced are much cleaner in terms of quality.

Edge Detection:

Edge detection is accomplished through the use of different information: RGB color, luma, depth, surface normal, object ID, or combinations of them. This means various methods will detect various types of edges. All the types can be combined and used but this puts an unnecessary toll on the performance. For EBAA, Luma was chosen for four reasons: first, MLAA processes edges that occur from color-based (either luma or RGB) discontinuities; otherwise artifacts may appear [2]. Second, unlike depths and normals, color information is always available for the algorithm. Third, it is compatible with shading aliasing. And fourth, it is much faster than RGB color while maintaining similar performance and similar results. For efficiency purposes, EBAA searches for edges at the top and left boundaries of each pixel, and uses other information from the neighbours.

Local Contrast Adaptation: To reduce artificial edges (caused due to local numerical differences) that reduce image quality, an adaptive threshold is performed which: a) prevent line searches from stopping at non-perceptually-visible crossing edges; and b) choose the dominant (higher contrast) edge when there are two parallel edges on a pixel.

Pattern Handling:

The EBAA's pattern detection can preserve sharp geometric features like corners, deal with diagonals and perform accurate distance searches.

Sharp Geometric Features: The re-vectorization of the image elements of MLAA tends to round corners on the image. Here the crossing edges used for the pattern detection are just one pixel long, and this makes it is not possible to distinguish a jagged edge from the actual corner of an object, which may therefore be wrongly processed.

To avoid this, two-pixel-long crossing edges are used instead; this helps in detecting the actual corners and ensure that there is no unnecessary processing of wrong features of an object. The degree of processing applied is defined by a rounding factor r, which scales the original coverage areas obtained by one-pixel-long crossing edges. The recommended range for r is [0:0-1:0].

Diagonal Patterns: The filter-based techniques available now search for patterns made of horizontal and vertical edge patterns. This translates into badly aliased results (in space and time) for diagonal lines (see Figure 2).

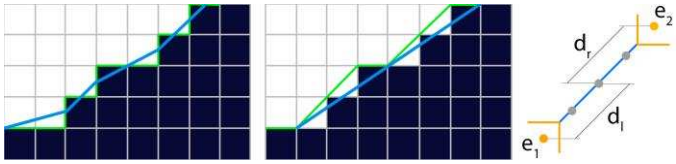

Figure 2: MLAA (left) and EBAA (center) re-vectorizations (blue lines) of near-45o diagonals. Thanks to the new handling of diagonal patterns (green lines), EBAA reconstructs the edge accurately. Right: The new approach requires just the same information as for the orthogonal case: distances dl and dr; and crossing edges e1 and e2 (right).

EBAA introduces a diagonal pattern detection that helps detecting the diagonal patterns. Here, a diagonal re-vectorization (Figure 2, center) is used, instead of the original orthogonal re-vectorizations that handle only horizontal and vertical edges (Figure 2, left). The mechanism uses a precomputed texture that takes the diagonal pattern, defined by the distances to both ends of the diagonal line and the diagonal crossing edges information (Figure 2, right); and outputs the accurate coverage areas.

Calculating diagonal coverage areas consists of the following steps, for both the top-left to bottom-right and the bottom-left to top-right diagonal cases:

- First, search for the diagonal distances dl and dr to the left and to right end of the diagonal lines.
- Then fetch the crossing edges e1 and e2.
- And finally use this input information (dl , dr, e1, e2), defining the specific diagonal pattern, to access the precomputed area texture, yielding the areas at and ab.

Accurate distances search: Obtaining accurate edge distances is the most important aspect of pattern classification.

Jimenez's MLAA makes extensive use of hardware interpolation (bilinear filtering) to identify patterns. Hardware bilinear filtering can be used as a way of fetching and encoding up to four different values with a single memory access (otherwise it would be necessary to perform one memory access per value to fetch). This is exploited to fetch two edges at once, allowing to partially reduce bandwidth usage. However, it does not check crossing edges during the search, which may lead to inaccuracies in pattern detection [3].

Unfortunately, fetching the crossing edges using the MLAA scheme requires two linearly filtered accesses per iteration, doubling the bandwidth usage. EBAA uses an approach similar to the MLAA approach but slightly alters it by generalizing the approach for two dimensional accesses which fetches four different values with a single memory access.

Jimenez's MLAA uses a linear interpolation of two binary values producing a single floating point value:

$$f_x(b_1, b_2, x) = x \cdot b_1 + (1 - x) \cdot b_2,$$

where *b1* and *b2* are two binary values, and x is the interpolation value. If $x \neq 0.5$, this produces a set of four unique values: {0, 1 – *x*, *x*, 1}. So, it is possible to find a decoding function f^{-1} that recovers the original *b1* and *b2* binary values. Instead EBAA performs bilinear interpolation of four binary values as follows:

$$f_{xy}(b, x, y) = f_x(b_1, b_2, x) \cdot y + f_x(b_3, b_4, x) \cdot (1 - y),$$

where y is the interpolation value in the second dimension. By choosing a value of $y = 0.5x$, it is possible to create a binary base that allows to encode a bilinear interpolation between four binary values into a single one, and still be able to recover the sixteen possible original values. EBAA uses this method to fetch the four *b1*, *b2*, *b3* and *b4* binary edge values.

Subpixel Rendering:

The MLAA algorithm uses a single sample per pixel. This leads to subsampling, thus making sure that the real subpixel features cannot be implemented (see Figure 3, no AA and MLAA). Having lower number of sample may reduce the bandwidth required but the more samples per pixel the algorithm has to process, the better the reconstruction of the antialiased image will be. The easiest method would be incorporating MSAA into MLAA. This method involves applying MSAA over each subsample group of MLAA separately and then averaging them together. However, this approach leads to blurry results (see Figure 3, MSAA 4x with MLAA). This is due to MLAA and MSAA making different assumptions about the coverage of the samples, so they cannot converge even when the samples per pixel count is increased.

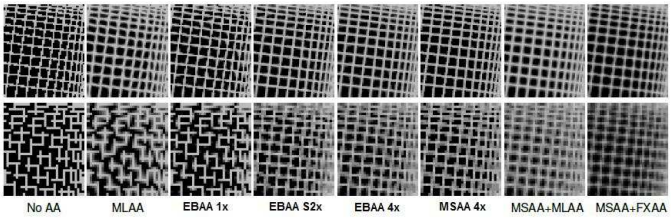

Figure 3: A difficult case for no AA, MLAA [3] and EBAA 1x: a white grid over a black background at mid-distance (top), prevents the reconstruction of accurate coverage; at a longer distance (bottom, zoomed in), the continuity of the grid is broken, preventing its recovery. Using extended patterns to deal with sharp geometric features and correct offsets allows for more accurate area estimation, making EBAA S2x and 4x converge to the MSAA 4x reference. Note how the naive application of MLAA over samples from MSAA 4x improves the connectivity of the grid, but blurring artifacts appear.

EBAA's solution here is to calculate the offset position of each subsample inside the pixel, in order to calculate their coverage areas accurately. Through this solution, when the different subsample groups are blended together, the average color at the center of the pixel is obtained as expected. Then, the only required change is to use different precomputed areas textures for each subsample position. This approach is general enough to handle additional samples coming from standard approaches like temporal supersampling and spatial multisampling, so several configurations are possible. In particular, the following modes are found to be the most interesting from a performance/quality perspective:

- EBAA 1x: includes accurate distance searches,

local contrast adaptation, sharp geometric features and diagonal pattern detection.

- EBAA S2x: includes all EBAA 1x features plus spatial multisampling.
- EBAA T2x: includes all EBAA 1x features plus temporal supersampling.
- EBAA 4x: includes all EBAA 1x features plus spatial and temporal multi/supersampling.

Figure 3 shows how EBAA 4x performs better to MSAA 4x. This makes EBAA better than simply combining MSAA and MLAA.

Temporal Reprojection:

While temporal supersampling helps efficiently render subpixel features, combining it with naive resolve approaches like linear blending results in very noticeable artifacts, commonly referred to as ghosting (see Figure 4, left). A better solution is to re-project instead the previous frames' subsamples into the current frame [3]. However, open regions still suffer from ghosting (see Figure 4, middle). To minimize this, EBAA weights the previous subsample by w, which depends on the difference in velocity with respect to the current subsample:

$$w = 0.5 \cdot max(0, 1 - K \cdot \sqrt{|\|v_c\| - \|v_p\||}),$$

where v_c and v_p are the velocity of current and previous frames, and K is a constant that determines how much EBAA attenuates previous frame according to velocity differences. Then, the final resolve is performed as follows:

$$c = (1.0 - w) \cdot c_c + w \cdot c_p.$$

where c is the final resolved color, c_c the color in current frame, and c_p the color in the previous frame. Such a solution robustly handles open regions but at the expense of no antialiasing on such regions (see Figure 4, right). However, the other components of EBAA will usually antialias these regions, effectively eliminating the problem.

Figure 4:* Left: *Using a naive resolve results in visible ghosting.* Middle: *Reprojection mitigates these artifacts but does not completely remove them.* Right: *The addition of velocity weighting allows to completely remove ghosting.

Results

Figure 5 shows a comparison of EBAA technique against MSAA, SSAA and MLAA. The screenshots are all 1080p images that have been captured from a system running on the AMD Radeon™ HD 7970 clocked at 1000 Mhz and Intel® Core™ i7 3770 clocked at 3500Mhz. As mentioned earlier, subpixel modes allow higher thresholds for edge detection, which lowers execution times without visible loss of image quality. The various games used for the test are: Assassin's Creed® Brotherhood, Borderlands® 2 and Crysis® 3 respectively.

Figure 5: EBAA 1x can produce results close to SSAA 16x, with EBAA T2x having a performance on par with the fastest MLAA implementation [3].The left image of the first row has MSAA only enabled. The left image of the first row has SSAA enabled. Finally in the third row, the left image has MLAA enabled. In all of the rows the right image has EBAA enabled. The improved edge/pattern detection allows to antialias difficult cases (first row). Subpixel features handling allows preserving connectivity and accurately representing distant objects (second row). The detection of sharp geometric features allows bettering reconstructing corners and intersections. Diagonal pattern detection allows accurate reconstruction of such shapes (third row).

As evident from the test results, EBAA not only solves limitations of just MLAA in particular, but of all postprocessing antialiasing filters in general. With respect to memory consumption, the most demanding in-game configuration requires only 38% of the memory used by MSAA 8x, resembling the results of SSAA 16x. Also evident is that EBAA is able to perform better than MLAA, while delivering superior overall quality, both in gradients and shading.

Conclusion

Through my personal experience and the various tests performed, I can conclude that this new technique tackles all the weak points remaining in filter-based antialiasing solutions. EBAA is a unique combination of a filter-based antialiasing technique with standard multi/supersampling approach and temporal reprojection. EBAA can be considered as a spiritual successor to MLAA as it is a combination of improved MLAA strategies and spatial and temporal multi/supersampling which accounts for a very robust solution, combining the different synergies for better fallbacks. EBAA delivers very accurate gradients, temporal stability and robustness, while introducing minimal overhead, making it an obvious choice for low-end configurations. We believe that through EBAA, developers can bring high quality antialiasing performance to mid-range GPUs, and highly recommend it to anyone who would like to increase their graphical fidelity without comprising memory bandwidth.

References

[1] J. Andersson, 5 Major Challenges in Interactive Rendering, ACM SIGGRAPH Courses (2010).
[2] J. Jimenez, D. Gutierrez, J. Yang, A. Reshetov, P. Demoreuille, T. Berghoff, C. Perthius, H. Yu, M. Mcguire, T. Lottes, H. Malan, E. Persson, D. Andreev, T. Sousa, Filtering Approaches For Real-Time Antialiasing, ACM SIGGRAPH Courses (2011).
[3] J. Jimenez, B. Masia, J. I. Echevarria, F. Navarro, D. Gutierrez, Practical Morphological Anti- Aliasing, GPU Pro 2, AK Peters Ltd., (2011).
[4] A. Reshetov, Morphological Antialiasing, High Performance Graphics (2009)