EUROGRAPHICS 2012 / P. Cignoni, T. Ertl
(Guest Editors)

*Volume 31 (2012), Number 2*

# SMAA: Enhanced Subpixel Morphological Antialiasing

Jorge Jimenez[1] Jose I. Echevarria[1] Tiago Sousa[2] Diego Gutierrez[1]

[1]Universidad de Zaragoza, Spain
[2]Crytek GmbH, Germany


**Figure 1:** *Example of SMAA 4x integrated in the* Crysis 2 *game. The insets show the differences between MLAA [JME*11], our novel SMAA T2x and 4x algorithms and MSAA 8x as reference. For 1080p frames, the average cost of SMAA T2x is 1.3 ms and 2.6 ms for SMAA 4x, measured on a NVIDIA GeForce GTX 470.*

**Abstract**

*We present a new image-based, post-processing antialiasing technique, which offers practical solutions to the common, open problems of existing filter-based real-time antialiasing algorithms. Some of the new features include local contrast analysis for more reliable edge detection, and a simple and effective way to handle sharp geometric features and diagonal lines. This, along with our accelerated and accurate pattern classification allows for a better reconstruction of silhouettes. Our method shows for the first time how to combine morphological antialiasing (MLAA) with additional multi/supersampling strategies (MSAA, SSAA) for accurate subpixel features, and how to couple it with temporal reprojection; always preserving the sharpness of the image. All these solutions combine synergies making for a very robust technique, yielding results of better overall quality than previous approaches while more closely converging to MSAA/SSAA references but maintaining extremely fast execution times. Additionally, we propose different presets to better fit the available resources or particular needs of each scenario.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing

## 1. Introduction

Aliasing is one of the longest-standing problems in computer graphics, producing clear artifacts in still images (spatial domain) and introducing flickering animations (temporal domain). While using higher sampling rates can ameliorate its effects, this approach is too expensive and thus not suitable for real-time applications. During the last few years we have seen great improvements in real-time rendering algorithms, from complex shaders to enhanced geometric detail by means of tessellation. However, aliasing remains one of the major stumbling blocks for trying to close the gap between off-line and real-time rendering [And10].

For more than a decade, *supersample antialiasing* (SSAA) and *multisample antialiasing* (MSAA) have been the gold standard antialiasing solutions in real-time applications and video games. However, MSAA does not scale well when increasing the number of samples and is not trivial to include in modern real-time rendering paradigms such as de-

ferred lighting/shading [JME*11, And11a, JGY*11]. To exemplify this problem with numbers, MSAA 8*x* 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 126 MB 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.

Recently, both industry and academia have begun to explore alternative approaches, where antialiasing is performed as a post-processing step [JGY*11]. The original *morphological antialiasing* (MLAA) method [Res09] gave birth to an explosion of real-time antialiasing techniques, rivaling in quality the results of MSAA and with a performance within the $[0.1 - 5]$ ms range. However, analyzing the current generation of filter-based antialiasing techniques, they all share at least some of the following problems:

- Most edge detection methods only take into account numerical differences between pixels, ignoring the fact that the surroundings of an edge also affect how humans perceive them.
- The original shape of the objects is not always preserved; an overall rounding of the corners is most of the times clearly visible in text, sharp corners and subpixel features.
- Most approaches are designed to handle horizontal or vertical patterns only, ignoring diagonals.
- Real subpixel features and subpixel motion are not properly handled.
- Specular and shading aliasing is not completely removed, especially when it happens at subpixel level.

Addressing all these issues while maintaining practical real-time performance poses a real challenge. We propose a novel post-process antialiasing technique, *Enhanced Subpixel Morphological Antialiasing* (SMAA). Our approach follows the divide-and-conquer paradigm, and tackles these complex problems separately, offering simple, modular solutions. First, we extend the number and type of edge patterns in order to keep sharp geometric features while processing also diagonal lines. Second, by adding multi/supersampling and temporal reprojection to morphological antialiasing, we are able to reconstruct real subpixel features and handle subpixel motion. Last, we introduce a robust edge detection that exploits local contrast along with accelerated yet precise distance searches for a more accurate pattern classification.

Given the modular nature of our approach, specific features can be enabled or disabled, adjusting to the needs of each particular scenario and hardware configuration. We propose four different modes, from the simplest to the more sophisticated version, which includes a novel combination of antialiasing as a post-process filter, and both spatial and temporal supersampling. This flexibility allows for direct, practical use of our technique even in current mainstream hardware. Furthermore, we have made public all the source code at `http://iryoku.com/smaa/`, including very exhaustive comments for both implementation and integration, to ensure both reproducibility and an easy and fast adoption of the technique.

## 2. Related Work

The simplest form of real-time antialiasing is *supersampling antialiasing* (SSAA), which involves rendering the scene at a higher resolution, then downsampling to the final resolution. It is also the basis of *multisampling antialiasing* (MSAA) [Ake93], where the color of a pixel is only calculated once instead of running at subsample frequencies. To display the scene, all samples are aggregated using some filter (a resolve operation). Although recent related techniques like *CSAA* [You06] and *EQAA* [AMD11] reduce bandwidth and storage costs by decoupling coverage from color, depth and stencil, these methods still inherit MSAA drawbacks.

The addition of new real-time rendering paradigms such as deferred shading [DWS*88, Har04, GPB04] and the lighting pre-pass [Eng08], along with current limitations in graphics hardware, have recently motivated a great amount of exciting new research in this field [JGY*11]. Most of the recent antialiasing solutions handle the aliasing problem as a post-process, devising filters that are applied over the final, aliased image, usually rendered at final display resolution. The basic idea is to find discontinuities on the image and to blur them in clever ways, in order to smooth the jagged edges. While the approach is not entirely new [Blo83, VO92, IK99], some advanced versions of it have been only recently applied in games [Shi05, Koo07, Sou07]. All these techniques alleviate the aliasing problem, although the sharp definition of the edges is obviously lost to a degree. More refined solutions like *directionally localized antialiasing* (DLAA) [And11b], use smarter blurs that produce very natural results and good temporal coherence. Nevertheless, these approaches still yield blurrier results than MSAA.

Other solutions, such as *morphological antialiasing* (MLAA) [Res09], try to estimate the pixel coverage of the original geometry based on the color discontinuities found in the final image. Reshetov's original work provides great results, but the proposed CPU implementation is not fast enough to be used in real-time. This triggered a number of real-time implementations that run on different hardware platforms, such as the GPU [BHD10, AMD10, JME*11], Playstation 3 SPUs and hybrid approaches that use both CPU and GPU [JGY*11, DP11]. *Topological reconstruction antialiasing* (TMLAA) [Bir11] uses topological information to recover subpixel features from the final image. However, this reconstruction can only fill one-pixel-sized holes, and it is not clear how well its assumptions work for animated sequences. *Fast approximate antialiasing* (FXAA) [Lot11] approaches the subpixel problem by simply attenuating such features, which enhances the perceived temporal stability.

However, its resulting images are still not at the quality level of standard methods like MSAA.

Deviating from pure image-based solutions, in the *distance-to-edge antialiasing* technique (DEAA) the forward rendering pass calculates and stores the distances of each pixel to near triangle edges with subpixel precision [JGY*11]. The post-process pass uses this information to derive blending coefficients. Similar in spirit, Persson's *GPAA* [Per11] and *GBAA* [JGY*11] use additional geometric information for coverage calculation. This produces almost perfect gradients with great temporal stability. However, working at final display resolution means they cannot handle subpixel features. Furthermore, they require either additional output buffers in the main pass or additional geometry passes. Providing better handling of subpixel features in deferred engines, *subpixel reconstruction antialiasing* (SRAA) [CML11] combines regular shading at final display resolution with supersampled geometry maps (normals and depth). Then, a super-resolution color image is built propagating the shaded samples over those maps; the resulting image is finally down-sampled again to final screen resolution. Despite bringing subpixel features to the table, they are based on heuristic estimations and the resulting gradients are in general of lower quality when compared with other approaches. *Directionally adaptive edge antialiasing* [IYP09] leverages MSAA subsample values for better gradient and color estimation. However, execution times are on the high side limiting the viability of the method to specific projects.

Finally, in very demanding realtime scenarios with complex shading and geometry, temporal antialiasing approaches have regained interest recently [NSL*07, YNS*09] [JGY*11] (see section *Anti-Aliasing Methods in CryENGINE 3*). The main idea is to distribute the cost of supersampling over contiguous frames. Our work also takes this aspect into account, handling subsamples via temporal reprojection. In a different context, the work of Yang and colleagues [YSLH11] aims at restoring jagged edges that occur after nonlinear image processing filters, for which they require that the original, alias-free image be available.

Table 1 in the supplementary material provides a detailed summary of the features supported for a representative selection of filter-based antialiasing techniques, including our work. This selection covers most of the recent major publications in the field, and includes all those for which implementations are available and are currently in use, in order to perform fair comparisons. It can be seen how each existing technique aims at solving a subset of all the problems involved, at the cost of leaving others out. In contrast, we provide a more holistic approach and systematically tackle all of them, while maintaining modularity by design.

## 3. Morphological Antialiasing

*Morphological antialiasing* (MLAA) [Res09], tries to estimate the pixel coverage of the original geometry. 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 (assuming a back-to-front rendering order). MLAA begins with an image without antialiasing (no coverage taken into account during rasterization), so it reverses the process by re-vectorizing the silhouettes, in order to estimate such coverage areas. Then, since the background cannot be known after rasterization, MLAA blends with a neighbor, assuming that its value is similar to the original background. Figure 2 describes this process; we refer the reader to the original publication for a more detailed explanation [Res09].

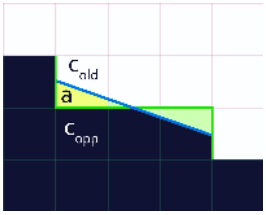


**Figure 2:** *MLAA first finds edges by looking for color discontinuities (green lines), and classifies them according to a series of pre-defined pattern shapes, which are then virtually re-vectorized (blue line), allowing to calculate the coverage areas* a *for the involved pixels. These areas are then used to blend with a neighbor. For example, the pixel* $C_{opp}$ *fills the area* a *of the pixel* $C_{old}$*:* $c_{new} = (1-a) \cdot c_{old} + a \cdot c_{opp}$.

Several morphological antialiasing implementations appeared after Reshetov's original paper [JGY*11]. Jimenez's MLAA [JME*11] is one of the fastest and most documented. Its key feature is the use of novel *texture structures* for great performance improvements. These textures are used to encode the location of the edges and coverage areas, as well as the precomputed areas for blending. The algorithm works in three passes: edge detection (which is performed using depth or luma information), pattern detection plus calculation of coverage areas, and final blending. Pattern detection is performed by searching both ends of an edge (*distance searching*), halving the necessary iterations by using hardware bilinear filtering. Once the ends are reached, the algorithm looks at the *crossing edges*, which provide a mechanism for straightforward pattern classification; these crossing edges are the perpendicular edges with respect to the direction of a search (see for example the vertical green lines in Figure 2). With length and crossing edges information, the coverage area is retrieved with a single access to a precomputed texture, and used for the final blending. Figure 3 exemplifies the different steps and components of the pipeline. We choose this MLAA implementation as a starting point for our algorithm, and refer the reader to the original publication for a more comprehensive description [JME*11].

## 4. SMAA: Features and Algorithm

In this section we present the core components of SMAA, their motivation and the main algorithmic ideas (see Fig-

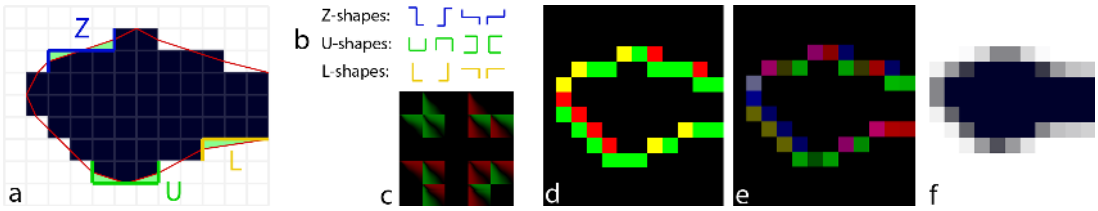


**Figure 3:** *MLAA overview. (a) Input image, with the intended approximation outlined by red lines and the coverage areas shown in green. (b) Predefined patterns in the original algorithm [Res09]. (c) Precomputed areas texture in Jimenez's GPU implementation [JME*11]. (d) Detected edges. (e) Calculated coverage areas. (f) Final blending. Our SMAA algorithm overhauls the whole pipeline by extending (b) and (c) for sharp geometric features and diagonals handling. Local contrast adaptation removes spurious edges in (d). Extended patterns detection and accurate searches improve accuracy in (e). SMAA can handle additional samples in (f) for accurate subpixel features and temporal supersampling.*

ure 4). We build on Jimenez's MLAA pipeline, improving or completely redefining every step. In particular, we improve edge detection by using color information with local contrast adaptation for cleaner edges. We extend the number of patterns handled for sharp geometric features preservation and diagonals processing. In a similar fashion, we enhance pattern handling with accurate and fast distance searches for a more reliable edge classification. Last, we show how morphological antialiasing can be accurately combined with multi/supersampling and temporal reprojection. Although our new technique shares some of the core ideas of MLAA, it constitutes a major overhaul in terms of quality and robustness (see Figure 3).

| | Temporal Stability | Shape reconstruction | Subpixel handling | Supersampling |
|---|---|---|---|---|
| Local contrast adaptation | ● | ● | | |
| Sharp geometric features detection | | ● | | |
| Diagonal patterns detection | ● | ● | | |
| Accurate distance searches | ● | ● | | |
| Temporal supersampling | ● | ● | ● | ● |
| Spatial multisampling | ● | ● | ● | |

**Figure 4:** *Overview of the key weaknesses of post-processing antialising filters (columns) and how the core elements of SMAA handle them (rows).*

### 4.1. Edge detection

Edge detection is critical in all AA filters, since each undetected edge will remain aliased on the final image. On the other hand, too many blurred edges can reduce the quality of the antialiased image, while imposing unnecessary performance penalties. Different information can be employed for edge detection: RGB color, luma, depth, surface normal, object ID... or combinations of them. We choose to use luma based on four observations: first, MLAA expects edges to come specifically from color-based (either luma or RGB) discontinuities; otherwise artifacts may appear [JGY*11] (see section *MLAA on the PS3*). Second, as opposed to depth and normals, color information is *always* available. Third, it can handle shading aliasing. And fourth, it is faster than RGB color while usually yielding similar results. For efficiency, we only search for edges at the top and left boundaries of each pixel, since the bottom and right ones can be retrieved from the neighbors.

**Local contrast adaptation**: The human visual system tends to mask low contrast edges in the presence of much higher contrasts in the surrounding area. Thus, a naive color edge detection based exclusively on local numerical differences will produce spurious edges (usually undetected by humans) that will affect pattern classification, downgrading image quality and temporal stability (see Figure 5, top). To avoid these spurious edges, we perform an adaptive double threshold which allows to: a) prevent line searches from stopping at non-perceptually-visible crossing edges; and b) choose the dominant (much higher contrast) edge when there are two parallel edges on a pixel (top-bottom, or left-right). This differs from previous approaches that take into account local contrast by simply checking the range of lumas found in the current pixel and its 4-neighborhood, and thus do not allow the notion of perceptual masking between edges [Lot11].

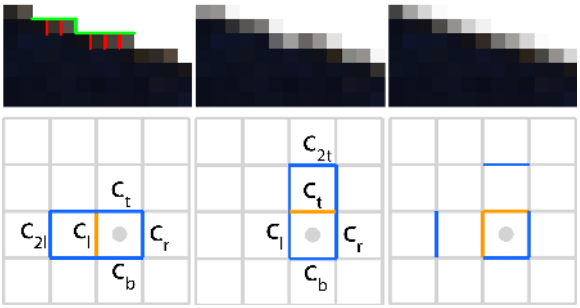


**Figure 5:** Top*: Dominant contrast in green edges should mask the spurious red crossing edges (left). Not taking this local contrast into account leads to artifacts (center). Our SMAA algorithm corrects them (right).* Bottom-left*: left boundary (orange) of a given pixel (marked with a dot) and surrounding candidate edges (blue) that may dominate it, making it non-visible for human viewers.* Bottom-middle*: top boundary scenario.* Bottom-right*: candidate surrounding edges actually calculated.*

Figure 5, bottom-left, shows the case for left edge (orange) of a given pixel (grey dot), plus the surrounding candidate edges (blue) that may *dominate* (mask) it. We calculate the maximum contrast $c_{max}$ for all these edges and compare it with the contrast for the left edge. If the latter is above a threshold of $0.5 \cdot c_{max}$ the edge is preserved; otherwise, it is ignored. The threshold was chosen empirically and provides good results in all our tests. The bottom-middle image shows the similar case for the top edge. Since computing all these edges involves too many memory accesses, we select a subset that yields satisfactory results (bottom-right).

For the case of the left boundary, a straightforward algorithm would calculate $e_l = |L - L_l| > T$, where $e_l$ is the boolean value that codes whether the edge is active, $L$ and $L_l$ represent luma values at the current and left pixels respectively, and $T$ is a given threshold (usually between 0.05 and 0.2). We refine this naive approach with an additional test that can be expressed as:

$$\begin{aligned} c_{max} &= max(c_t, c_r, c_b, c_l, c_{2l}) \\ e'_l &= e_l \wedge c_l > 0.5 \cdot c_{max} \end{aligned} \quad (1)$$

where $c_t, c_r, c_b, c_l, c_{2l}$ are the contrast deltas for the edges shown in Figure 5, and $e'_l$ represents the final boolean value (active or not) for the left edge boundary. The edge at the top boundary, $e'_t$, is calculated in a similar fashion.

### 4.2. Pattern handling

Our new pattern detection allows to preserve sharp geometric features like corners, deals with diagonals and enables accurate distance searches.

**Sharp geometric features**: The re-vectorization of silhouettes of MLAA tends to round corners on the image (see Figure 6, left). Given that the crossing edges used for pattern detection are just one pixel long, it is not possible to distinguish a jagged edge from the actual corner of an object, which may be wrongly processed.

To avoid this, we make the key observation that crossing edges in contour lines have a maximum size of one pixel, whereas for sharp corners this length will most likely be longer. We thus fetch two-pixel-long crossing edges instead; this allows to detect actual corners and apply a less aggressive processing, thus retaining more closely the true shape of the object (see Figure 6, right). 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 (blue lines in Figure 6, right). The recommended range for $r$ is $[0.0 - 1.0]$. For example, values of $r = 1.0$, 0.5 and 0.0 yield the blue, yellow and pink lines respectively.

For the (academic) case of an horizontal line, we modify Jimenez's MLAA coverage areas calculation as follows:

1. We perform the original pattern detection, using the regular crossing edges (red edges on Figure 6, right). This yields two areas $a_b$ and $a_t$ per pixel belonging to the pattern. $a_b$ is used to blend the bottom pixel $p_b$ with its top neighbor $p_t$, whilst $a_t$ is used to blend $p_t$ with $p_b$ (see [JME*11] for details).
2. We refine the areas $a_t$ and $a_b$ according to the following:

$$a'_t = \begin{cases} r \cdot a_t & \text{if } d_l < d_r \wedge e_1 \\ r \cdot a_t & \text{if } d_l \geq d_r \wedge e_2 \\ a_t & \text{otherwise} \end{cases} \quad (2)$$

$$a'_b = \begin{cases} r \cdot a_b & \text{if } d_l < d_r \wedge e_3 \\ r \cdot a_b & \text{if } d_l \geq d_r \wedge e_4 \\ a_b & \text{otherwise} \end{cases} \quad (3)$$

where $a'_t$ and $a'_b$ are the modified area values, $d_l$ and $d_r$ are the distances to the left and to the right of the line for the current pixel, and $e_i$ are booleans that indicate if an edge is active (see Figure 6).

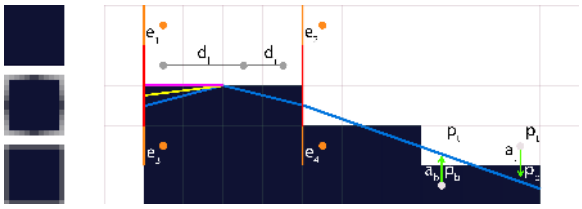

**Figure 6:** Left*: Comparison between no antialiasing (top), a regular MLAA approach (middle), and the SMAA results (bottom). Notice how SMAA keeps the original shape of the object much better, while MLAA tends to round its shape.* Right*: Corners have crossing edges of length at least two (see the second pixel column), while aliased contour lines have crossing edges of just one pixel in length (staircase towards the right). Fetching extended crossing edges (orange), in addition to regular edges (red), allows to discern between both cases, yielding a more accurate re-vectorization (pink), instead of rounding off corners (blue).*

**Diagonal patterns**: Most of the existing filter-based techniques search for patterns made exclusively of horizontal and vertical edges (orthogonal patterns). This translates into badly aliased results (in space *and* time) for diagonal lines (see Figure 7).

We introduce a novel diagonal pattern detection that allows to detect these scenarios. In these cases, a diagonal re-vectorization (Figure 7, center) is used to yield coverage ar-

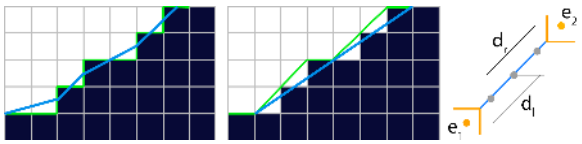

**Figure 7:** *MLAA (left) and SMAA (center) re-vectorizations (blue lines) of near-45° diagonals. Thanks to our handling of diagonal patterns (green lines), SMAA reconstructs the edge accurately. Right: our approach just requires the same information as for the orthogonal case: distances $d_l$ and $d_r$; and crossing edges $e_1$ and $e_2$ (right).*

eas, instead of the original orthogonal re-vectorizations (Figure 7, left). The mechanism developed to handle diagonal patterns is inspired by the orthogonal patterns handling of Jimenez's MLAA. We introduce a precomputed texture that takes as input the diagonal pattern, defined by the distances to both ends of the diagonal line and the diagonal crossing edges information (Figure 7, right); and outputs the accurate coverage areas. Figure 8 shows the possible diagonal patterns and their corresponding pre-calculated 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:

1. We search for the diagonal distances $d_l$ and $d_r$ to the left and and to right end of the diagonal lines.
2. We fetch the crossing edges $e_1$ and $e_2$.
3. We use this input information $(d_l, d_r, e_1, e_2)$, defining the specific diagonal pattern, to access the precomputed area texture, yielding the areas $a_t$ and $a_b$.

We perform this diagonal pattern detection before the orthogonal one in the coverage area calculation. If the diagonal pattern detection fails, we trigger the orthogonal detection. Otherwise, the areas produced by the diagonal pattern detection are used. This model allows to seamlessly perform the last blending step (step $f$ in Figure 3) in a symmetric way for both orthogonal and diagonal patterns, given the fact that the semantics of the produced areas $a_t$ and $a_b$ are the same in both cases.

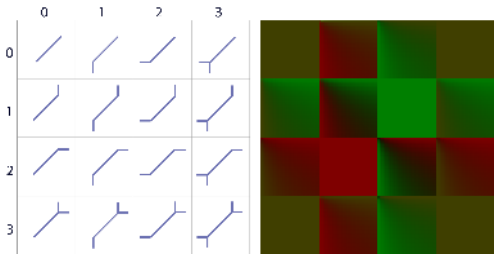


**Figure 8:** *Diagonal patterns map (left) and their precomputed area texture (right).*

**Accurate distances search**: Key to pattern detection and classification is obtaining accurate edge distances (lengths to both ends of the line). Jimenez's MLAA makes extensive use of hardware interpolation (bilinear filtering) to accelerate this process. 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 (see Figure 9, bottom-left). However, it does not check crossing edges during the search, which may lead to inaccuracies in pattern detection [JME*11].

Unfortunately, fetching the crossing edges in the search loop following their scheme would imply two linearly filtered accesses per iteration, doubling the bandwidth usage. We generalize the approach for *two dimensional* accesses, being able to fetch four different values with a single memory access (Figure 9, bottom-right).

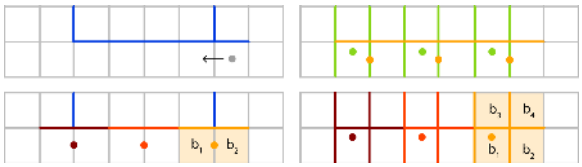


**Figure 9:** Top-left*: Example of a search to find the left end of a horizontal edge (starting position marked with a dot).* Top-right*: Hardware filtered accesses performed by Jimenez's MLAA (orange dots) just check the orange edges. SMAA bilinear accesses (green dots) are able to additionally check all the crossing edges (green). In this example, this will make the search stop when the first crossing edge is found, instead of finishing at the left end of the horizontal edge.* Bottom-left*: The different iterations of the search (represented by different colors) and the values fetched by Jimenez's MLAA. Note how it misses all the crossing edges (in blue).* Bottom-right*: The same iterations and the values fetched by SMAA. It can be seen how SMAA is able to check four different pixels (shaded pixels) with just a single memory access.*

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

$$f_x(b_1, b_2, x) = x \cdot b_1 + (1 - x) \cdot b_2, \tag{4}$$

where $b_1$ and $b_2$ are two binary values (either 0 or 1, given that the edges texture marks each edge as activated or not), 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 $b_1$ and $b_2$ binary values. Instead SMAA performs bilinear interpolation of four binary values as follows:

$$f_{xy}(b, x, y) = f_x(b_1, b_2, x) \cdot y + f_x(b_3, b_4, x) \cdot (1 - y), \tag{5}$$

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. We exploit this fact to fetch the four $b_1$, $b_2$, $b_3$ and $b_4$ binary edge values (see Figure 9, bottom-right). We refer the reader to the source code for the specific details of this feature.

### 4.3. Subpixel rendering

MLAA algorithms work with a single sample per pixel. This translates into subsampling, which makes it impossible to recover real subpixel features (see Figure 10, no AA and MLAA). Having more samples per pixel allows for a better reconstruction of the antialiased image. A naive extension would involve using MLAA in conjunction with MSAA,

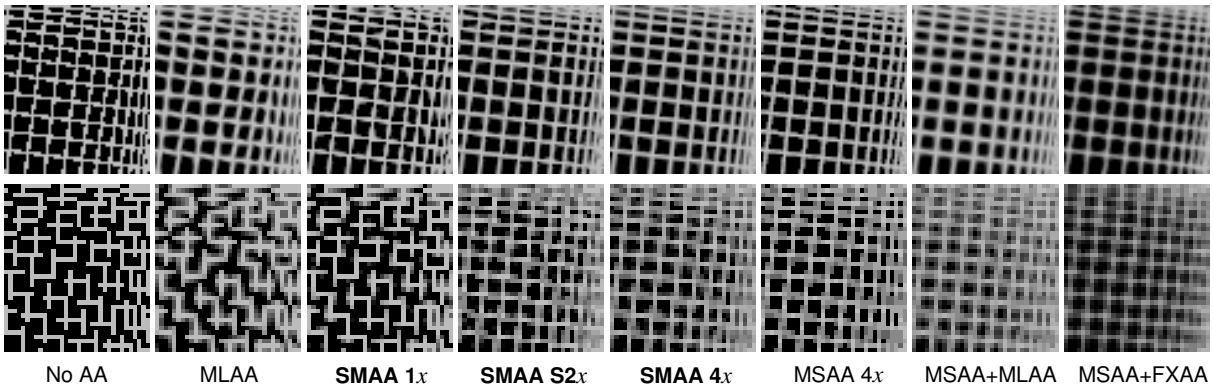


**Figure 10:** *A difficult case for no AA, MLAA [JME*11] and SMAA 1x: a white grid over a black background at mid-distance (top), prevents the reconstruction of accurate coverage; at a longer distance (bottom, zoomed in), the continuity of the grid is broken, preventing its recovery. Using extended patterns to deal with sharp geometric features and correct offsets allows for a more accurate area estimation, making SMAA 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. We have also performed the same test using FXAA 3.11 (preset 39, max. quality) since it is one of the most used MLAA-like solutions. Figures 3, 4 and 5 in the supplementary material extend this with additional MSAA modes, and AA filters applied pre and post resolve.*

applying it over each subsample group separately and then averaging them together. However, using such a simple approach leads to blurry results (see Figure 10, MSAA 4$x$ with MLAA). This is due to MLAA and MSAA making different assumptions about the coverage of the samples, so they cannot converge even increasing the samples per pixel count:

- MLAA is designed to work on the silhouettes of objects and not with thin lines and features, as found in distant objects with high-frequency details (see Figure 10, MLAA). As it can be seen, not taking into account sharp geometric features leads to blurry results (with unnatural glows). Thus, sharp geometric features detection (Subsection 4.2) is critical when applying MLAA over subpixel features. Ultimately, this allows for corners to be conservatively reconstructed, in order to allow multi/supersampling to reconstruct their real shape (see Figure 10, SMAA 1$x$; notice how non-silhouette features are ignored).
- Given that MLAA assumes the positions of all the samples to be at the center of the pixel for the revectorization, this simply does not work due to the under-/overestimation of the corresponding coverage areas, producing gradients that do not match in the resolve step, which translate into a blurry appearance of distant objects.

In addition to sharp features detection, our solution is to take into account the offset position of each subsample inside the pixel, in order to calculate properly their coverage areas. This way, when the different subsample groups are blended together, we obtain the average color at the center of the pixel (see Figure 11, left and middle). Then, the only required change to the pipeline 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, we have found the following modes to be the most interesting from a performance/quality perspective:

- SMAA 1$x$: includes accurate distance searches, local contrast adaptation, sharp geometric features and diagonal pattern detection.
- SMAA S2$x$: includes all SMAA 1$x$ features plus spatial multisampling.
- SMAA T2$x$: includes all SMAA 1$x$ features plus temporal supersampling.
- SMAA 4$x$: includes all SMAA 1$x$ features plus spatial and temporal multi/supersampling.

The SMAA 4$x$ mode requires to temporally jitter the SMAA S2$x$ mode, as shown in Figure 11 (right). Figure 10 shows how SMAA 4$x$ converges better to MSAA 4$x$ than simply combining MLAA with MSAA.

### 4.4. Temporal reprojection

While temporal supersampling allows to efficiently render subpixel features, coupling it with a naive resolve approach like linear blending results in very noticeable residual artifacts, commonly referred to as *ghosting* (see Figure 12, left).

A better solution is to re-project instead the previous frames' subsamples into the current frame [NSL*07, YNS*09] [JGY*11] (Section *Anti-Aliasing Methods in CryENGINE 3*). However, disoccluded regions (occluded regions in the previous frame now visible in the current frame) still suffer from residual artifacts (see Figure 12, middle). To minimize them, we weight the previous subsample by $w$, which depends on the difference in velocity with respect to the current subsample:

$$w = 0.5 \cdot max(0, 1 - K \cdot \sqrt{|\|v_c\| - \|v_p\||}), \qquad (6)$$

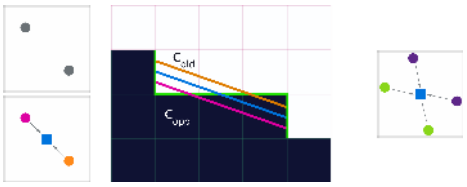

**Figure 11:** Left, top*: usual MSAA 2x pattern, with offsets at* $(-0.25, 0.25)$ *and* $(0.25, -0.25)$. Left, bottom*: For combining multi/supersampling with MLAA (SMAA S2x and T2x), we have to offset the area calculations so that the average between the subsamples on top and bottom (pink and orange) corresponds to the color at the center of the pixel (blue).* Middle*: MLAA area calculations are devised to estimate the re-vectorization at the center of the pixel (blue). For SMAA S2x and T2x, these areas must be offset by* $-0.25$ *(pink) and* $+0.25$ *(orange).* Right*: Example of combining four subsamples (SMAA 4x) coming from both spatial multisampling and temporal supersampling, using two jittered results of SMAA S2x (purple and green).*

where $v_c$ and $v_p$ are the velocity of current and previous frames, and $K$ is a constant that determines how much we attenuate previous frame according to velocity differences (we use a value of 30 for all our examples). Then, the final resolve is performed as follows:

$$c = (1.0 - w) \cdot c_c + w \cdot c_p. \tag{7}$$

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 disoccluded regions but at the expense of no antialiasing on such regions (see Figure 12, right). Nevertheless, the other components of our technique (either MLAA or spatial multisampling) will usually antialias these regions, effectively eliminating the problem.

A remaining problem of combining velocity weighting with a morphological strategy is that morphological antialiasing is actually blending pixels from both sides of the silhouette of an object at subpixel level. However, the velocity map remains aliased and so velocity is not propagated to the antialiased pixels, which leaves trails of blended pixels behind objects in motion. The solution is to apply SMAA also over the velocity buffer, in order to propagate velocities to the blended pixels. To efficiently perform this step, we coarsely store the velocity module in the alpha channel of the color buffer, so SMAA processes it for free.

**Figure 12:** 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.*

## 5. Results

Figure 13 shows a comparison of the subpixel modes of our technique against MLAA and SSAA 16x. Figures 1 and 2 from the supplementary material contain a more detailed comparison with a large number of selected antialiasing methods. We refer the reader to the supplementary material for additional examples both with still images and video. We recommend the digital version of the article for proper examination. Performance metrics are measured on a NVIDIA GeForce GTX 470 using 1080p images. Typical execution times for our technique are of 1.02 ms for SMAA 1$x$, 1.32 ms for SMAA T2$x$, 2.04 ms for SMAA S2$x$ and 2.34 ms for SMAA 4$x$. Subpixel modes allow higher thresholds for edge detection (see *better fallbacks* below), which lowers execution times without visible loss of image quality.

*Local contrast:* The first column of Figure 13 shows how a conventional edge detection approach (MLAA) usually fails to properly detect patterns in the presence of gradients. Note how our approach is able to detect and correctly antialias these difficult zones for smooth gradients.

*Diagonal pattern detection:* Our algorithm accurately reconstructs a perfectly straight diagonal line for the streetlamp silhouette. Traditional post-processing approaches generate aliasing artifacts, which can be clearly seen when looking at the image at native pixel resolution and in motion.

*Sharp geometric features:* Our technique manages to preserve the sharp corners in the base of the aerials (specially the one of the satellite dish), whereas most filter-based antialiasing techniques introduce some degree of roundness. This information is vital for multi/supersampling to reconstruct the accurate shape of an object. Also text present on textures or the user interface is better preserved.

*Accurate searches*: Blindly following edges without checking crossing edges at each step causes pattern detection to fail in some scenarios, as shown in the MLAA image. Our accurate search allows to enhance the antialiasing quality without increasing the number of memory accesses.

*Subpixel features*: Post-processing techniques using 1$x$ (final display resolution) color inputs are unable to reconstruct *accurate* subpixel features (which accounts for all selected techniques with the exception of SSAA), producing artifacts like spurious pixels, gaps in surfaces and distracting effects due to subsampling. In contrast, our SMAA T2$x$ mode is able to better preserve the connectivity of the lines, resembling more faithfully the results obtained with SSAA 16$x$. SMAA S2$x$ and 4$x$ modes also provide real subpixel features at the expenses of multisampling, an approach

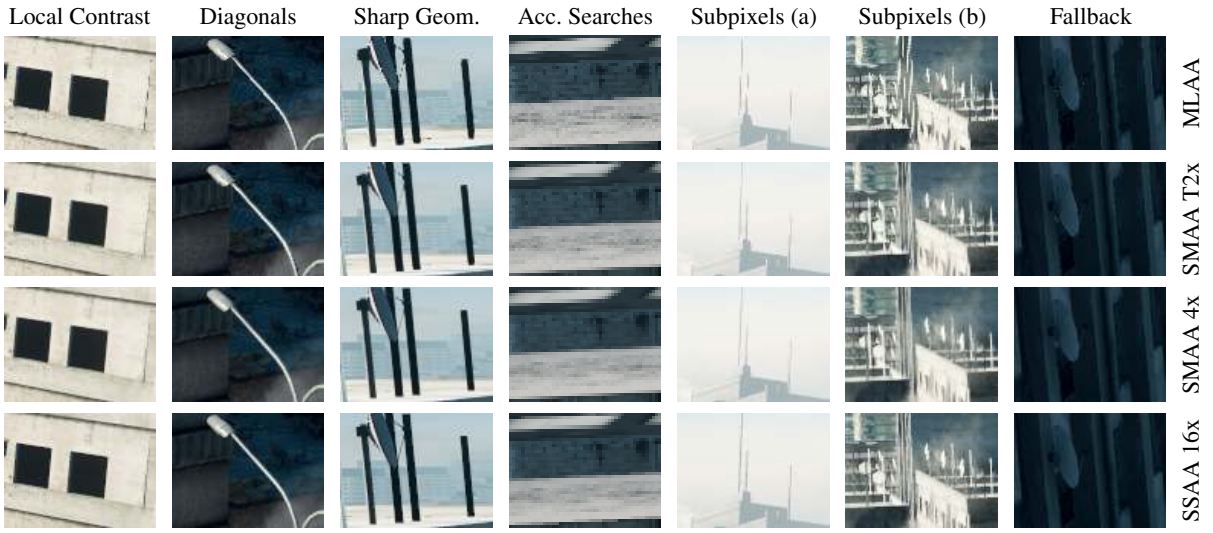


**Figure 13:** *SMAA can produce results close to SSAA 16x, with SMAA T2x having a performance on par with the fastest MLAA implementation [JME*11]. The improved edge/pattern detection allows to antialias difficult cases (first column). Diagonal pattern detection allows accurate reconstruction of such shapes (second column). The detection of sharp geometric features allows to better reconstruct corners and intersections (see second window in the first column, and bases of the aerials on the third column). Accurate searches allow to detect patterns in difficult scenarios (fourth column). Subpixel features handling allows to preserve connectivity and accurately represent distant objects (fifth and sixth columns). In zones with low-contrast edges, we fall back to MSAA 2x (seventh column), which provides good results and improves performance.*

with varying viability depending on the complexity of the shaders.

*Better fallbacks*: Subpixel SMAA modes not only allow actual handling of subpixel features, but also provide better fallbacks for additional robustness. If the morphological component of SMAA leaves any edge unprocessed, the MSAA component of S2$x$ and 4$x$ modes will back that up. If the temporal reprojection present in T2$x$ and 4$x$ modes fails due to changes in the occlusion of objects between frames, the morphological and MSAA components will reduce aliasing. And the possible shading aliasing of S2$x$ will be made up by temporal SSAA and MLAA in SMAA 4$x$, making it the most robust mode.

**Discussion**: Most of the features we have described solve limitations of not just MLAA in particular, but of *all* post-processing antialiasing filters in general. Performance-wise, in a forward rendering scenario SMAA 4$x$ and SMAA T2$x$ are about 1.46$x$ and 4.09$x$ faster than MSAA 8$x$ respectively (the first taking into account the required multisampling 2$x$ overhead). With respect to memory consumption, the most demanding configuration requires only 43% and 17% of the memory used by MSAA 8$x$, in a forward and deferred context respectively. Note that we are able to perform better than MSAA 8$x$, while delivering superior overall quality, both in gradients and shading, resembling more accurately the results of SSAA 16$x$. In the case of a deferred engine, using MSAA 8$x$ would incur an excessive drop of performance given the massive bandwidth required [And11a], along with the requirement of supersampling the edges at 8$x$.

In SMAA 1$x$ and T2$x$ modes, the execution times are within the same 1 ms ballpark as other solutions (see Table 1 in the supplementary material). The S2$x$ and 4$x$ modes are obviously more expensive due to multisampling (an average of 1.57 ms overhead for rendering at 2$x$, minus the resolve time), but they are still on-par with other techniques that handle subpixel features (SRAA and MSAA 8$x$), still yielding smoother results. Note that SRAA requires additional 4$x$ multisampled depth and/or normal maps and possibly two geometry passes; while our approach multisamples color information at 2$x$. In the case of a deferred renderer, our approach would require supersampling the edges; however, stencil-masked implementations allow for efficient performance.

The overhead introduced by each of our solutions is either negligible or very affordable. In particular, local contrast adaptation is 0.08 ms, the sharp geometric features detection and accurate distance searches take less than 0.01 ms; diagonals processing and temporal supersampling introduce a small overhead of 0.12 ms and 0.3 ms respectively. Spatial multisampling adds 1.02 ms for filtering the second sample, and an additional average of 1.57 ms to render the scene at 2$x$. The delta that SMAA 4$x$ adds on top of a 2$x$ forward-rendered scene is as little as 2.09 ms, making it an attractive option for any scenario that can afford such a small multisample count.

## 6. Conclusions

We have presented a technique that tackles all the weak points remaining in filter-based antialiasing solutions. We have shown for the first time how to combine a filter-based antialiasing technique with standard multi/supersampling approaches and temporal reprojection. This novel combination of improved MLAA strategies with spatial and temporal multi/supersampling accounts for a very robust solution, combining the different synergies for better fallbacks. SMAA 1*x* delivers very accurate gradients, temporal stability and robustness, while introducing minimal overhead, making it an obvious choice for low-end configurations. SMAA T2*x*, for little additional cost, offers a very attractive tradeoff for any kind of rendering engine (deferred or forward), avoiding 2*x* multisampling while still reconstructing subpixel detail. SMAA S2*x* and SMAA 4*x* are the best options regarding image quality. We believe that SMAA will finally enable deferred engines to match the antialiasing quality of forward rendering engines.

## 7. Acknowledgements

We thank the anonymous reviewers and the members of the Graphics and Imaging Lab for their valuable comments and suggestions. We also thank Stephen Hill, Jean-Francois St-Amour, Johan Andersson, Alex Fry, Naty Hoffman, Carsten Wenzel, Nick Kasyan, Pierre-Yves Donzallaz and Michael Kopietz for their help and support. This research has been funded by the European Commission, Seventh Framework Programme, through the projects GOLEM (Marie Curie IAPP, grant agreement no.: 251415) and VERVE (Information and Communication Technologies, grant agreement no.: 288914), and by the Spanish Ministry of Science and Technology (TIN2010-21543). Jorge Jimenez is also funded by a grant from the Gobierno de Aragón.

## References

[Ake93] AKELEY K.: Reality engine graphics. In *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (1993), pp. 109–116. 2

[AMD10] AMD: Morphological anti-aliasing, 2010. 2

[AMD11] AMD: *EQAA Modes for AMD 6900 Series Graphics Cards*. Tech. rep., AMD, 2011. 2

[And10] ANDERSSON J.: 5 major challenges in interactive rendering. In *ACM SIGGRAPH Courses* (2010). 1

[And11a] ANDERSSON J.: DirectX 11 Rendering in Battlefield 3. In *Game Developers Conference 2011* (2011). 2, 9

[And11b] ANDREEV D.: Directionally localized anti-aliasing (DLAA). In *Game Developers Conference 2011* (2011). 2

[BHD10] BIRI V., HERUBEL A., DEVERLY S.: Practical morphological antialiasing on the GPU. In *ACM SIGGRAPH 2010 Talks* (2010), SIGGRAPH '10, pp. 45:1–45:1. 2

[Bir11] BIRI V.: Morpological antialiasing and topological reconstruction. In *GRAPP 2011* (2011). 2

[Blo83] BLOOMENTHAL J.: Edge inference with applications to antialiasing. *SIGGRAPH Comput. Graph. 17* (1983), 157–162. 2

[CML11] CHAJDAS M. G., MCGUIRE M., LUEBKE D.: Subpixel reconstruction antialiasing for deferred shading. In *Symposium on Interactive 3D Graphics and Games* (2011), pp. 15–22. 3

[DP11] DE PEREYRA A.: *MLAA: Efficiently Moving Antialiasing from the GPU to the CPU*. Tech. rep., Intel, 2011. 2

[DWS*88] DEERING M., WINNER S., SCHEDIWY B., DUFFY C., HUNT N.: The triangle processor and normal vector shader: a vlsi system for high performance graphics. *SIGGRAPH Comput. Graph. 22* (June 1988), 21–30. 2

[Eng08] ENGEL W.: Light pre-pass renderer, 2008. 2

[GPB04] GELDREICH R., PRITCHARD M., BROOKS J.: Deferred lighting and shading. In *Game Developers Conference 2004* (2004). 2

[Har04] HARGREAVES S.: Deferred shading. In *Game Developers Conference 2004* (2004). 2

[IK99] ISSHIKI T., KUNIEDA H.: Efficient anti-aliasing algorithm for computer generated images. In *ISCAS (4)* (1999), pp. 532–535. 2

[IYP09] IOURCHA K., YANG J. C., POMIANOWSKI A.: A directionally adaptive edge anti-aliasing filter. In *Proceedings of the Conference on High Performance Graphics 2009* (2009), pp. 127–133. 3

[JGY*11] JIMENEZ J., GUTIERREZ D., YANG J., RESHETOV A., DEMOREUILLE P., BERGHOFF T., PERTHUIS C., YU H., MCGUIRE M., LOTTES T., MALAN H., PERSSON E., ANDREEV D., SOUSA T.: Filtering approaches for real-time antialiasing. In *ACM SIGGRAPH Courses* (2011). 2, 3, 4, 8

[JME*11] JIMENEZ J., MASIA B., ECHEVARRIA J. I., NAVARRO F., GUTIERREZ D.: Practical Morphological Anti-Aliasing. In *GPU Pro 2*. AK Peters Ltd., 2011, pp. 95–113. 1, 2, 3, 4, 5, 6, 7, 9

[Koo07] KOONCE R.: Deferred Shading in Tabula Rasa. In *GPU Gems 3*. Addison Wesley, 2007, pp. 429–457. 2

[Lot11] LOTTES T.: *FXAA*. Tech. rep., NVIDIA, 2011. 2, 4

[NSL*07] NEHAB D., SANDER P. V., LAWRENCE J., TATARCHUK N., ISIDORO J. R.: Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (2007), pp. 25–35. 3, 8

[Per11] PERSSON E.: Geometric post-process anti-aliasing, 2011. 3

[Res09] RESHETOV A.: Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009* (2009), pp. 109–116. 2, 3, 4

[Shi05] SHISHKOVTSOV O.: Deferred Shading in S.T.A.L.K.E.R. In *GPU Gems 2*. Addison Wesley, 2005, pp. 143–166. 2

[Sou07] SOUSA T.: Vegetation Procedural Animation and Shading in Crysis. In *GPU Gems 3*. Addison Wesley, 2007, pp. 373–385. 2

[VO92] VAN OVERVELD C. W. A. M.: Application of morphological filters to tackle discretization artifacts. *Vis. Comput. 8* (1992), 217–232. 2

[YNS*09] YANG L., NEHAB D., SANDER P. V., SITTHI-AMORN P., LAWRENCE J., HOPPE H.: Amortized supersampling. *ACM Trans. Graph. 28* (2009), 135:1–135:12. 3, 8

[You06] YOUNG P.: *Coverage Sampled Antialiasing*. Tech. rep., NVIDIA, 2006. 2

[YSLH11] YANG L., SANDER P. V., LAWRENCE J., HOPPE H.: Antialiasing recovery. *ACM Trans. Graph. 30* (2011). 3

EUROGRAPHICS 2012 / P. Cignoni, T. Ertl
(Guest Editors)

*Volume 31 (2012), Number 2*

# SMAA: Enhanced Subpixel Morphological Antialiasing Supplementary material

Jorge Jimenez[1] Jose I. Echevarria[1] Tiago Sousa[2] Diego Gutierrez[1]

[1]Universidad de Zaragoza, Spain
[2]Crytek GmbH, Germany

In the movie included, "*SMAA.mp4*", we show the behavior of the features of SMAA in motion. We compare against Jimenez's MLAA [JME*11], FXAA 3.11 [Lot11] (*preset 39* for the highest image quality), and CSAA 16*x*Q [You06] (in order to have as reference the MSAA hardware implementation with the best image quality). We also show clips of SMAA integrated in the commercial game engine "*CryENGINE 3*" from *Crytek*.

File "*Battlefield3.psd*" contains the full resolution images used in Figure 13 of the paper, Figure 1 and Figure 2 of this document. Each layer corresponds to a method for an easier and flexible comparison between them. We recommend to check them with software like Photoshop or GIMP, and swap between layers. Since MSAA was not available for this scene, we compare against different SSAA modes. Note that because Reshetov's criteria for edge detection and pattern handling can be enhanced [Res09], differences between his implementation and Jimenez's MLAA [JME*11] can be appreciated (the same happens with AMD's implementation [AMD10]). For example, while Reshetov utilizes RGB information for finding edges, Jimenez's MLAA uses luminance values. And while Reshetov takes into account the largest pattern a pixel belongs to, Jimenez's MLAA uses the shortest. However, Jimenez's threshold was tuned to be similar to Reshetov's. AMD's implementation cannot be customized, so it was applied with default settings. Additional screenshot comparisons can be found at the project website: `http://iryoku.com/smaa/`.

Figure 3 extends Figure 10 from the paper by comparing SMAA and MSAA modes with the same number of samples per pixel. It can be seen how SMAA gradually converges to the MSAA reference, in some cases surpassing the quality of MSAA modes with higher samples count.

Figure 4 demonstrates how offset positions must be taken into account for the area calculations when several samples per pixel are used. In this case, we test FXAA applied pre-resolve over each subsample coming from MSAA 4x in the same fashion we did with MLAA in Figure 10 from the paper. As can be seen, not taking into account the exact position of each sample leads to blurry results that do not converge to the MSAA reference. Additionally, we test FXAA and MLAA applied post-resolve (Figure 5) over a resolved MSAA 2*x* input. The results show how suboptimal (even incorrect) it is to apply these antialiasing filters over an already antialiased input.

Table 1 presents performance numbers and features sets of all the techniques used in our tests and comparisons.

Last, "*Code.zip*" includes the commented source code for the SMAA shader, and the Python code for generating the precalculated areas textures. The most up-to-date version and a demo application with some test examples can be found at `http://iryoku.com/smaa`.

## References

[AMD10] AMD: Morphological anti-aliasing, 2010. 1, 2, 4

[And11] ANDREEV D.: Directionally localized anti-aliasing (DLAA). In *Game Developers Conference 2011* (2011). 2, 4

[CML11] CHAJDAS M. G., MCGUIRE M., LUEBKE D.: Subpixel reconstruction antialiasing for deferred shading. In *Symposium on Interactive 3D Graphics and Games* (2011), pp. 15–22. 2, 4

[JME*11] JIMENEZ J., MASIA B., ECHEVARRIA J. I., NAVARRO F., GUTIERREZ D.: Practical Morphological Anti-Aliasing. In *GPU Pro 2*. AK Peters Ltd., 2011, pp. 95–113. 1, 2, 4

[Lot11] LOTTES T.: *FXAA*. Tech. rep., NVIDIA, 2011. 1, 2, 4

[Res09] RESHETOV A.: Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009* (2009), pp. 109–116. 1, 2, 4

[You06] YOUNG P.: *Coverage Sampled Antialiasing*. Tech. rep., NVIDIA, 2006. 1

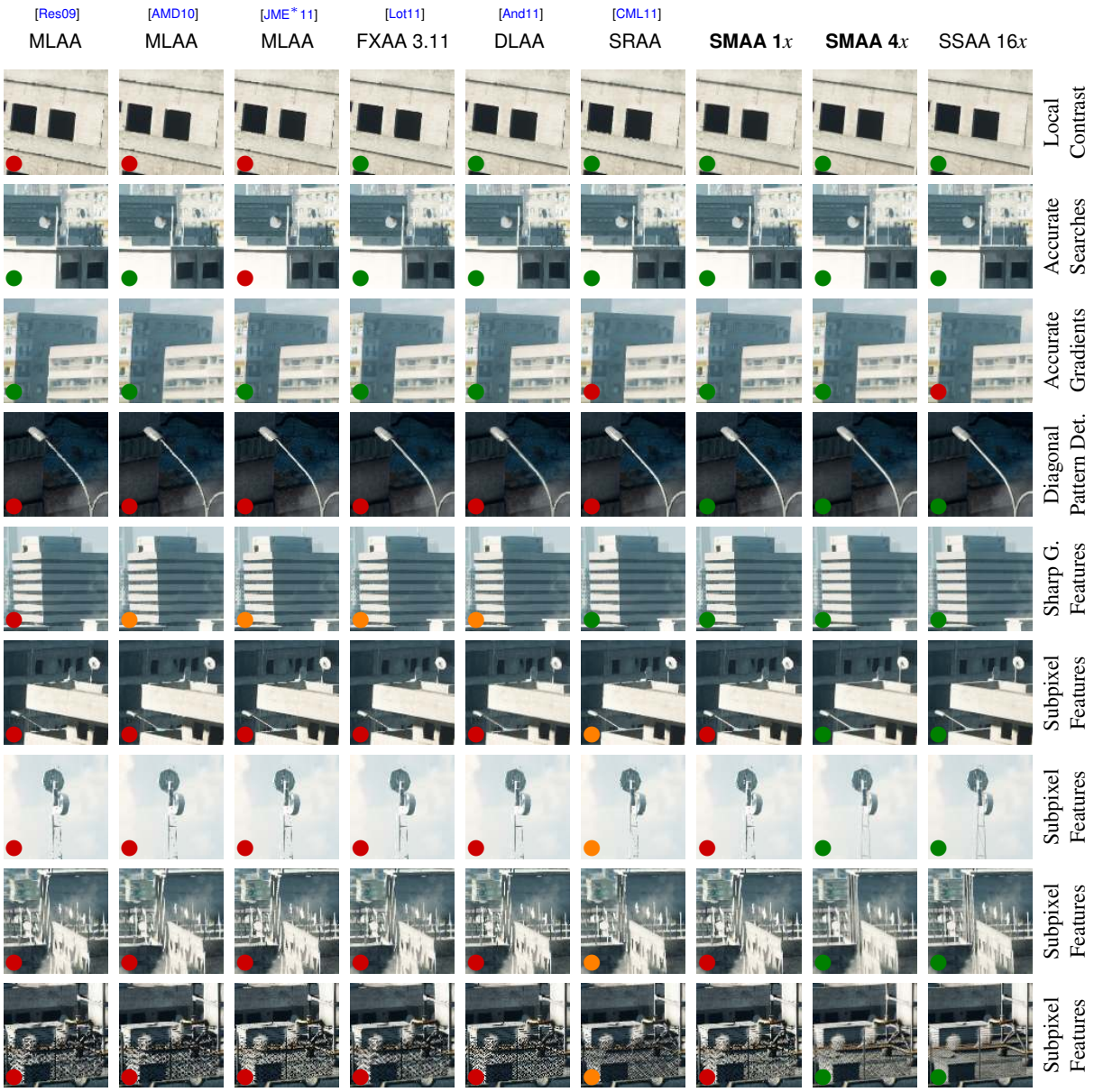


**Figure 1:** *Comparison of the features (rows) of our approach with a selection of anti-aliasing techniques (columns). To help the reader navigate through this image matrix, we have color-coded the performance of each method for each particular feature tested, although this is ultimately a subjective criterion. Green, orange and red dots mark accurate, regular and inaccurate handling of a feature. Gradients from SSAA 16x are hampered in this case because of the use of an ordered grid SSAA. In the case of FXAA, we used preset 39 (maximum quality). The accompanying Photoshop file also includes FXAA's default preset 12, SMAA 1x High (as included in this Figure) and Ultra (with lower threshold for edge detection and larger patterns handling). Zoom into the digital version of this paper or check the accompanying Photoshop files to see the details.*

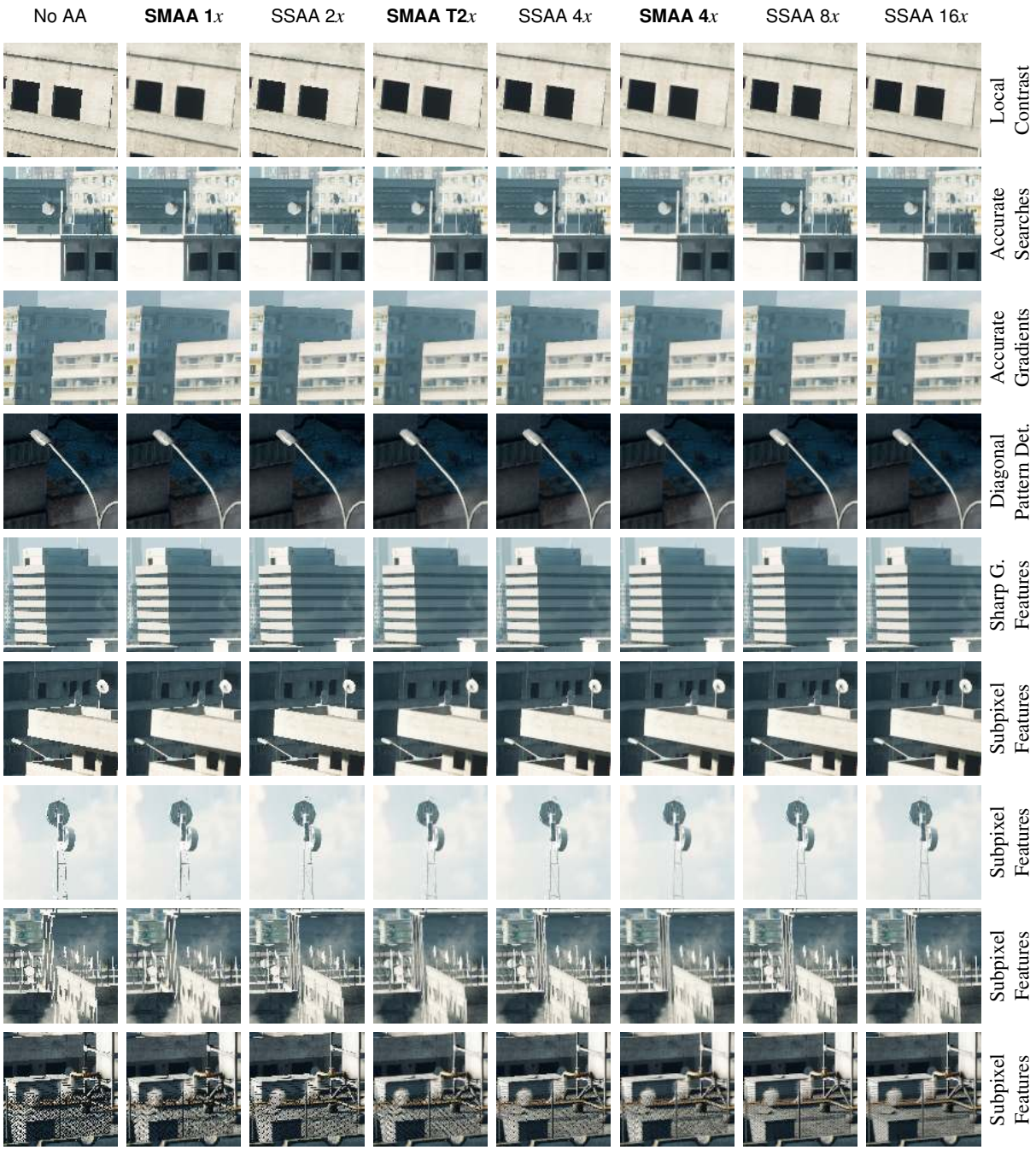


**Figure 2:** *Comparison of SMAA modes against ordered-grid SSAA. Zoom into the digital version of this paper or check the accompanying Photoshop files to see the details and comparison against all methods from Figure 1.*

**Table 1:** *Supported features for a selection of filter-based antialiasing techniques. Memory footprint in terms of: backbuffer size/depth buffer/additional render targets. Performance is given for 1080p on a NVIDIA GeForce GTX 470, with exception of: a) Reshetov's [Res09] CPU-based implementation, which is measured on a Core i7 2620M @ 2.7GHz; b) AMD's exclusive MLAA [AMD10], which is measured on a AMD Radeon HD 6870; and c) SRAA [CML11], whose times come from a GeForce GTX 480 (more powerful than the GeForce GTX 470). Please note that our SMAA T\S2x and 4x times include the resolves. MSAA performance numbers, calculated as the overhead over the same scenes without antialiasing, are 1.57 ms for MSAA 2x, 2.3 ms for MSAA 4x and 4.3 ms for MSAA 8x. Brute force SSAA overhead grows linearly with the number of samples, SSAA 16x takes an additional cost of 285 ms for the example in Figure 1 from the paper.*[1] *For fairness, we measured the 4x mode of our algorithm on the same GPU, yielding a performance of 1.82 ms.* [2] *We measured the times of FXAA 3.11 in default (12) and extreme (39) presets.*

| | [Res09] MLAA | [AMD10] MLAA | [JME* 11] MLAA | [Lot11] FXAA 3.11 | [And11] DLAA | [CML11] SRAA | SMAA 1x | SMAA S2x | SMAA T2x | SMAA 4x |
|---|---|---|---|---|---|---|---|---|---|---|
| Sharp geometric features | | | | | | yes | yes | yes | yes | yes |
| Diagonals | | | | | | | yes | yes | yes | yes |
| Subpixel features | | | | | | yes | | yes | yes | yes |
| Supersampled shading | | | | | | | | | yes | yes |
| Local contrast adaptation | | | | implicit | implicit | yes (n/a) | yes | yes | yes | yes |
| Accurate distance searches | yes | yes | | depends | yes | yes (n/a) | yes | yes | yes | yes |
| Accurate gradients* | yes | yes | yes | yes | yes | | yes | yes | yes | yes |
| Sharpness preservation* | medium | low | high | medium | medium | low | high | high | high | high |
| Ghosting-free | yes | yes | yes | yes | yes | yes | yes | | yes | |
| Input (color/depth) | 1x n/a | 1x n/a | 1x 1x | 1x n/a | 1x n/a | 1x 4x | 1x n/a | 2x n/a | 1x n/a | 2x n/a |
| Memory footprint | 1x/1x/n/a | 1x/1x/n/a | 1x/1x/1.5x | 1x/1x/0x | 1x/1x/1x | 1x/4x/0x | 1x/1x/2x | 2x/2x/2x | 1x/1x/4x | 2x/2x/4x |
| Performance | 350 ms | 6.6 ms[1] | 0.98 ms | 0.62 ms /0.83 ms[2] | 2.12 ms | 2.5 ms | 1.02 ms | 2.04 ms | 1.32 ms | 2.34 ms |

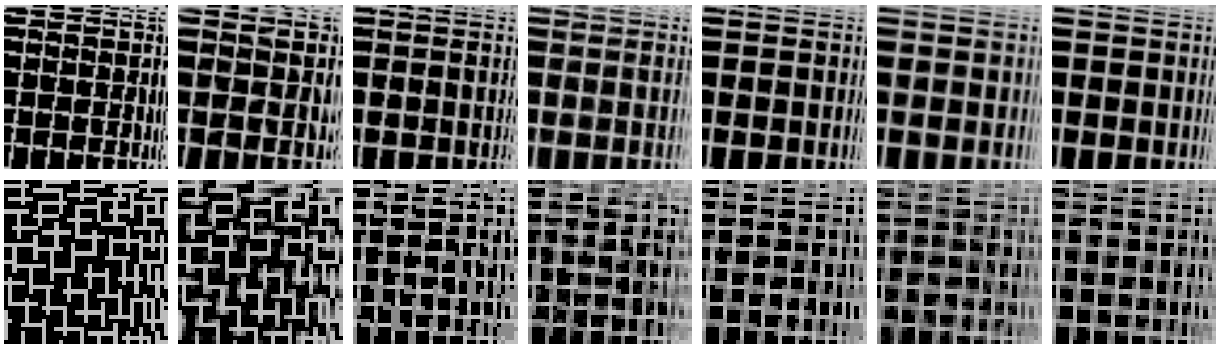


**Figure 3:** *Comparison between SMAA and MSAA modes with the same number of samples per pixel in a very challenging scenario for subpixel features, with MSAA 8x included as the highest quality MSAA reference. As can be seen, using the same amount of information, SMAA provides smoother gradients than its corresponding MSAA counterparts, in some cases surpassing the quality of MSAA modes with a higher number of samples. Performance numbers can be found in Table 1.*

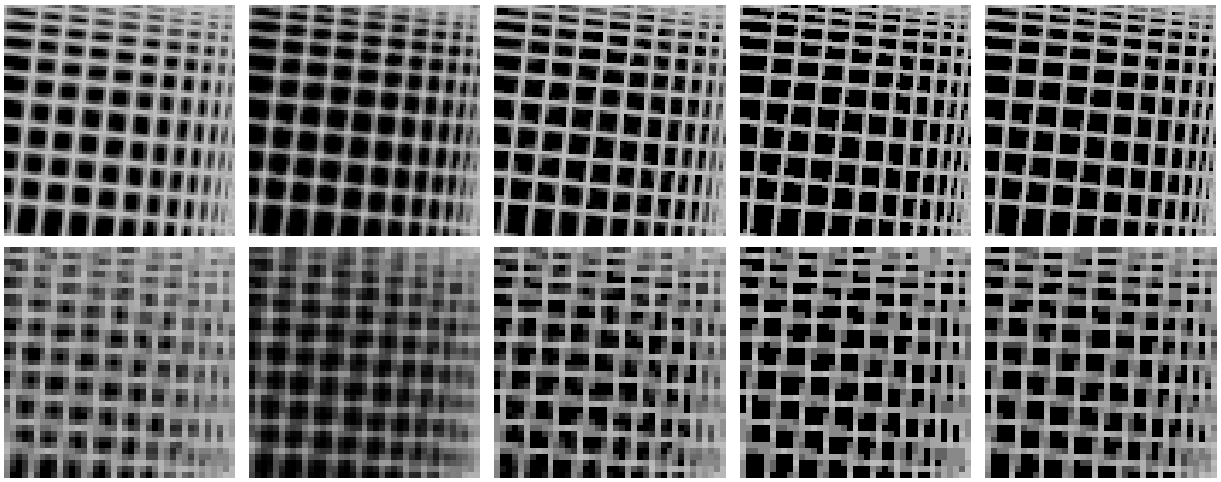


**Figure 4:** *AA filters applied pre-resolve to each sample of a MSAA 4x input. Unlike SMAA, FXAA and MLAA do not take into account the offset position of the additional samples, thus leading to blurry results when compared against the MSAA 4x and 8x references. FXAA 3.11 preset 39 (max. quality) and Jimenez's MLAA were used in this test.*

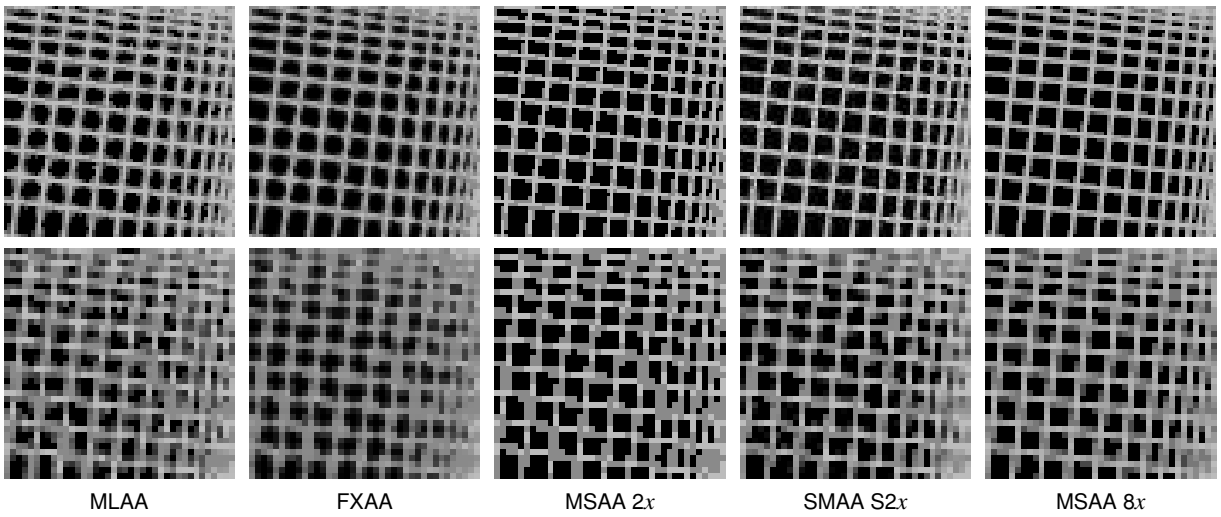


**Figure 5:** *AA filters applied post-resolve over a resolved MSAA 2x input. It can be seen that when FXAA or MLAA are not fed with clean edges, they introduce artifacts that make the final image look not to converge to the MSAA reference. SMAA S2x and MSAA 8x included as a higher quality reference. FXAA 3.11 preset 39 (max. quality) and Jimenez's MLAA were used in this test.*