Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Review:

Image anti-aliasing techniques for Internet visual media processing: a review*

Xu-dong JIANG[1], Bin SHENG[†‡1,2], Wei-yao LIN[3], Wei LU[4], Li-zhuang MA[1]

([1]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*)

([2]*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China*)

([3]*Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*)

([4]*Autodesk Software (China) Co., Ltd. Shanghai Branch, Shanghai 200122, China*)

[†]E-mail: shengbin@sjtu.edu.cn

Received Mar. 18, 2014; Revision accepted June 25, 2014; Crosschecked Aug. 19, 2014

Abstract: Anti-aliasing is a well-established technique in computer graphics that reduces the blocky or stair-wise appearance of pixels. This paper provides a comprehensive overview of the anti-aliasing techniques used in computer graphics, which can be classified into two categories: post-filtering based anti-aliasing and pre-filtering based anti-aliasing. We discuss post-filtering based anti-aliasing algorithms through classifying them into hardware anti-aliasing techniques and post-process techniques for deferred rendering. Comparisons are made among different methods to illustrate the strengths and weaknesses of every category. We also review the utilization of anti-aliasing techniques from the first category in different graphic processing units, i.e., different NVIDIA and AMD series. This review provides a guide that should allow researchers to position their work in this important research area, and new research problems are identified.

Key words: Anti-aliasing, Hardware, Post-filtering, Pre-filtering, Sampling
doi:10.1631/jzus.C1400100 **Document code:** A **CLC number:** TP391.41

1 Introduction

Pixels are the tiny square dots that make up an image on a computer screen and they are the smallest discrete elements used in graphics. Geometric shapes are sampled to generate pixels by rasterization, which is a major process in computer graphics. In general, shape boundaries constitute discontinuities in the signal, which introduce extremely high frequencies (Auzinger *et al*., 2012). A fundamental problem during the discretization process is that the signal cannot be reconstructed perfectly due to the limited sampling rate determined by the display resolution, according to the Nyquist-Shannon sampling theorem (Shannon, 1949). Thus, sharp edges often have a blocky or stair-wise appearance along the object silhouettes (Fig. 3b), which is known as aliasing. Aliasing effects may emerge in different situations, e.g., along the edges, silhouettes, or creases of objects, and very small objects can disappear between samples, or complex details of the features may be lost (Crow, 1977). Therefore, anti-aliasing is a well-established technique in computer graphics, which allows graphics to be more attractive by making the edges look smoother and less pixilated.

Various anti-aliasing techniques have been developed over the years, focusing on enhancing the visual quality and ensuring real-time rendering.

‡ Corresponding author

* Project supported by the National Basic Research Program (973) of China (No. 2011CB302203), the National Natural Science Foundation of China (Nos. 61202154, 61133009, and 61001146), the Shanghai Pujiang Program (No. 13PJ1404500), the Shanghai Science and Technology Commission (No. 13511505000), the Open Project Program of the National Laboratory of Pattern Recognition (Chinese Academy of Sciences), and the Open Project Program of the State Key Laboratory of CAD&CG, Zhejiang University, China (No. A1401)

Graphics hardware vendors have made significant efforts to compensate for aliasing artifacts by trading off the color accuracy against spatial continuity; i.e., multiple methods are supported in the hardware based on mixing weighted color samples (Young, 2007). In the academic domain, researchers have also proposed many algorithms to address different problems, e.g., estimating the color variance of pixels (Whitted, 1980; Amanatides, 1984; Heckbert and Hanrahan, 1984; Shinya *et al*., 1987; Thomas *et al*., 1989; Ohta and Maekawa, 1990; Genetti *et al*., 1998), observing color discontinuities at silhouette boundaries (Sander *et al*., 2001; Chan and Durand, 2005), recognizing certain patterns in images (Reshetov, 2009), and eliminating the problematic frequencies before rasterization (Crow, 1977; Catmull, 1978; Duff, 1989; McCool, 1995; Guenter and Tumblin, 1996; Manson and Schaefer, 2011; Auzinger *et al*., 2012; 2013a; 2013b).

In the following sections, anti-aliasing techniques are discussed based on two categories: post-filtering based anti-aliasing and pre-filtering based anti-aliasing. The former is further refined as anti-aliasing supported directly by hardware and post-process anti-aliasing techniques for deferred rendering. A detailed hierarchical structure of the anti-aliasing techniques can be seen in Fig. 1. Comparisons are made among different methods to illustrate the advantages and disadvantages of each category. These two different anti-aliasing pipeline systems are summarized in Fig. 2. In Fig. 2a, for example, the original image is anti-aliased initially using a pre-filtering method. Next, the signals are discretized by multiplying the filtered output by the sampling pattern obtained by rasterization. Finally, the image is reconstructed using an interpolation filter.

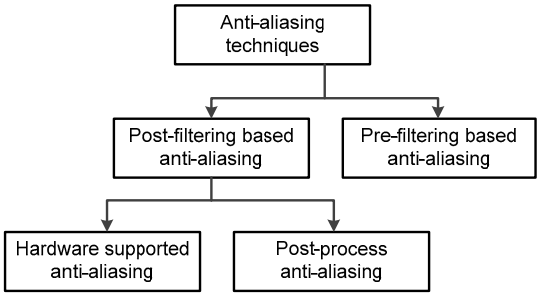

Fig. 1 A hierarchical overview of anti-aliasing techniques

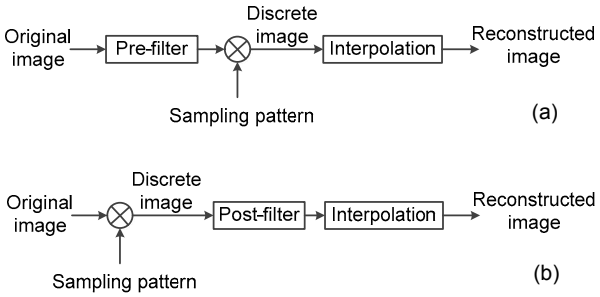

Fig. 2 Schematic of a pre-filtering (a) or post-filtering (b) anti-aliasing system

2 Post-filtering based anti-aliasing implementations

Different from pre-filtering based anti-aliasing, post-filtering based anti-aliasing methods are approximate sampling based solutions and are applied after sampling. In general, these methods can be classified into two categories: anti-aliasing supported directly by hardware, and anti-aliasing as a post-process for deferred rendering. Hardware supported anti-aliasing techniques have been integrated into graphic pipeline systems to provide a balance between visual quality and timing performance, while post-process methods can perfectly match the deferred rendering technique.

2.1 Anti-aliasing hardware implementations

Without anti-aliasing, a personal computer renders only the center of a pixel and applies a single color to the entire pixel. To improve visualization, hardware systems possess various integrated anti-aliasing techniques, where anti-aliasing is a hardware feature that is applied globally to the full screen.

2.1.1 Generic anti-aliasing implementations

Originally, the two main generic anti-aliasing methods are supersampling anti-aliasing (SSAA) and multisampling anti-aliasing (MSAA), which provide the basis for future improvements.

SSAA reduces aliasing artifacts by directly increasing the image sampling rate; i.e., instead of using the color detected at the center of a pixel, multiple color samples within the pixel are averaged (Leler, 1980). For example, if 4× SSAA is enabled on a

1280×720 monitor, the graphics card internally renders the frame at 2560×1440 and downsamples the image to a 1280×720 output resolution. Thus, edges, textures, and transparencies are all enhanced. In practice, this can be achieved either by computing N color samples per pixel or by rendering the scene in multiple passes with different subpixel offsets and then accumulating the results. At present, SSAA is the highest-quality anti-aliasing method available, but it is also the most expensive technique in terms of its processing and memory bandwidth requirements. Thus, most of the latest graphical processing units (GPUs) have stopped supporting SSAA to avoid performance degradation.

MSAA is a faster alternative to SSAA, supported by virtually all modern graphics processors and application programming interfaces (APIs). The basic principle of MSAA is to make a tradeoff between quality and workload, thereby minimizing the amount of extra processing while reducing the number of aliased edges as much as possible (Akeley, 1993). MSAA achieves this with two main techniques. The first technique is edge anti-aliasing, where the computer processes only extra samples of pixels at the edge of an object. A Z-test is performed before anti-aliasing is executed, where a difference in depth within a single pixel indicates that it contains the edge of an object and requires MSAA. The second technique reduces the sampling workload, where some calculations are performed only once per pixel, such as pixel shaders, texture lookups, and color sampling. Only the depth and stencil values are fully sampled, and they are then used to determine the optimal blend of colors between the object and the background.

Fig. 3 shows a comparison between the 4× MSAA (four samples per pixel) sampling pattern mode and the no anti-aliasing rendering mode. The original input image is shown in Fig. 3a. In Fig. 3c, there are four pixels, three of which contain the edge of a triangle and have been flagged because of differences in their Z-values. This means that MSAA needs to be performed. The four dots represent the positions where the MSAA samples were collected. Obviously, collecting more samples will increase the accuracy of the resulting color blend; i.e., 8× MSAA achieves better results than 4× MSAA and 4× MSAA is better than 2× MSAA. However, the test results obtained on GPUs show that using more than four anti-aliasing samples leads to diminishing returns. For example, 4× MSAA performs a much better job than 2× MSAA, but 8× MSAA does not provide a much better result than 4× MSAA. Fig. 4 highlights this point.

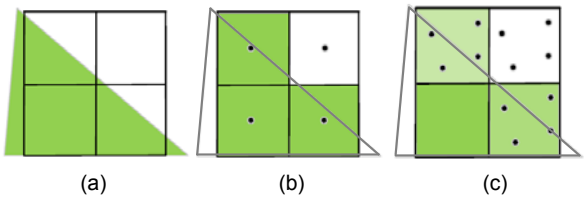

Fig. 3 Comparison of no anti-aliasing and multisampling anti-aliasing (MSAA) modes
(a) Input image; (b) No anti-aliasing rendering mode with sampling at the centroid of each pixel; (c) MSAA rendering mode with four samples per pixel

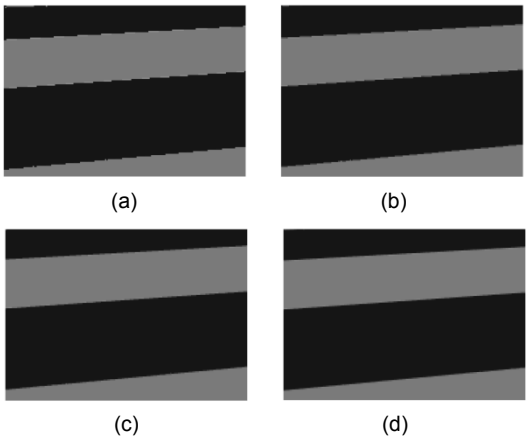

Fig. 4 Multisampling anti-aliasing (MSAA) results obtained with different sampling rates
(a) Original aliased image; (b) 2× MSAA result; (c) 4× MSAA result; (d) 8× MSAA result

2.1.2 Improved anti-aliasing implementations based on MSAA

MSAA is the de facto anti-aliasing method and provides a good balance between image quality and speed, although it still has some specific shortcomings. Thus, various improved anti-aliasing implementations have been developed based on MSAA.

Transparent texture technology is used widely to simulate highly detailed 3D objects, in which a 2D image is accompanied by a transparent alpha mask which improves the performance of the graphics system (Woligroski, 2011). However, MSAA does not

enhance the transparent texture quality in the same way as SSAA when it is used as an edge anti-aliasing technique. This means that transparent textures with aliased edges are not addressed by this method because there are no polygon edges inside. To overcome this problem, NVIDIA introduced a feature called transparency adaptive anti-aliasing (TAAA) based on two implementations, one using multisampling (TrMSAA) and the other using supersampling (TrSSAA) (Woligroski, 2011), which allows a choice between anti-aliasing the transparent texture at the same level as MSAA in the rest of the scene and supersampling the transparent texture with two, four, or eight samples. Theoretically, TrSSAA can work in almost every situation. However, test results show that it most often works in DirectX 10 and DirectX 11 game engines. TrMSAA, on the other hand, is limited to DirectX 9, and in practice it rarely works.

To further optimize MSAA, NVIDIA introduced a new technique called coverage sampling anti-aliasing (CSAA) (Young, 2007). Essentially, CSAA is a form of traditional MSAA with extra coverage samples, which resembles the operations of rendering with an oversized buffer and downfiltering by MSAA. The coverage samples determine whether a polygon is present at the location of the sample, which can be used to weight the calculation when blending the final color of the pixel. Coverage samples are relatively easy to gather; thus, CSAA produces anti-aliased images that rival the quality of 8× or 16× MSAA, but with only minimal effects on performance compared with standard (typically 4×) MSAA (Young, 2007). AMD (2011) offered enhanced quality anti-aliasing (EQAA), which is an identical mode to CSAA. However, the use of coverage samples provides a relatively limited increase in the overall anti-aliasing-based image quality. The use of more MSAA samples leads to a definite visual improvement, whereas more coverage samples may produce no increase in quality.

Custom filter anti-aliasing (CFAA) is a programmable method that gathers samples from within the pixel and outside the pixel edges (Woligroski, 2011). This means that samples can be collected within a circular radius of the pixel, unrestricted by the pixel boundaries, where the influence of the sample is also weighted based on its distance from the center of the pixel. As a customizable anti-aliasing mode, CFAA is performed by a shader, instead of in the raster operations (ROP), which also leads to performance concerns related to the use of shader resources.

All the techniques described above use spatial filters to preview aliasing. However, there is also a growing trend toward the use of temporal filters for attenuating aliasing. Temporal anti-aliasing (TXAA) is a film-style anti-aliasing technique that is integrated in NVIDIA graphic cards, designed specifically to reduce noticeable aliasing during movements (e.g., crawling and flickering). This technique is a mixture of a temporal filter, hardware anti-aliasing, and a custom computer graphics (CG) film-style anti-aliasing. To filter any given pixel on the screen, TXAA assesses the contribution of samples both inside and outside the pixel, as well as samples from previous frames to provide the highest quality filtering possibility. In contrast to methods such as fast approximate anti-aliasing (FXAA, Section 3.1), which attempt to maximize the performance at the expense of quality, TXAA attempts to maximize quality at the expense of performance.

2.2 Post-processing anti-aliasing implementations

The use of anti-aliasing in real-time applications such as games has been addressed mainly with MSAA or similar approaches such as CSAA, because these methods are practical and generally applicable for traditional forward rendering. However, as more and more rendering engines switch to a deferred shading model, the appeal of MSAA has been reduced significantly due to its added complexity and extra memory consumption (Deering *et al.*, 1988). Thus, industrial and academic researchers have explored alternative approaches to anti-aliasing as a post-processing step, which are well suited to deferred shading or fixed environments and can rival the performance of MSAA in real-time applications (Jimenez *et al.*, 2011a).

2.2.1 Morphological anti-aliasing techniques

The underlying concept of morphological anti-aliasing (MLAA) techniques involves: (1) detecting discontinuities, (2) calculating neighborhood weights, and (3) blending with the neighborhood. MLAA was developed by Intel Labs as an original post-processing anti-aliasing approach (Reshetov, 2009).

It processes the image buffer in three steps: finding discontinuities between pixels in a given image, identifying U-, Z-, and L-shaped patterns, and decomposing U/Z-shaped patterns into L-shaped patterns (Fig. 5) and then blending the colors in the neighborhood of each identified L-shaped pattern.

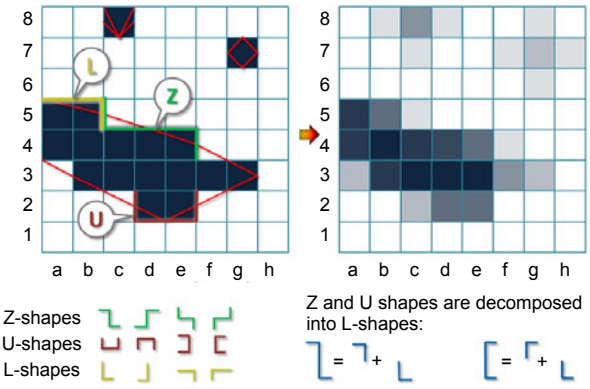

Fig. 5 Morphological anti-aliasing processing of an image with Z-, U-, and L-shaped patterns (Reshetov, 2009)

The original MLAA filter was the precursor of a generation of real-time anti-aliasing techniques, but it is nonlinear and requires deep branching and image-related knowledge, and may be inefficient in graphics hardware. Fortunately, Biri *et al*. (2010) developed an efficient MLAA scheme that runs flawlessly on medium range GPUs. In this technique, L-shaped edges are identified in the final image and samples are blended along the triangles formed. The value of the blended samples depends on the area of the trapezium covering each sample. A further evolution of MLAA for GPUs is known as Jimenez's MLAA (Jimenez *et al*., 2011b), which departs from the underlying concept by shifting to a paradigm that uses texture structures instead of lists. This method then handles all of the pattern types in a symmetric manner, thereby avoiding the need to decompose them into simpler patterns. In addition, the pre-computation of certain values into textures facilitates even faster implementations.

Based on Jimenez's MLAA, a novel post-processing anti-aliasing technique was developed by Jimenez *et al*. (2012), known as enhanced subpixel morphological anti-aliasing (SMAA). Basically, SMAA extends the number and type of edge patterns to process sharp geometric features and diagonal shapes by using an edge detection scheme that exploits local contrast features, as well as accelerated and precise distance searches. SMAA also addresses temporal aliasing by supplementing MLAA with additional multi/supersampling strategies, which are capable of reconstructing real subpixel features and handling subpixel motion in a comparable manner to 4× MSAA. However, the combination of supersampling and morphological methods leads to excessive blurring of shadows or textures and low resolution.

Andreev (2011) introduced directionally localized anti-aliasing (DLAA), which is a further simplification of MLAA and operates in a perceptual space, where the exact gradients produced by the specific areas covered may not be required. Instead of calculating weights, it uses vertical and horizontal blurring to produce gradients in the aliased edges. Gradients with different numbers of steps are used for two types of edges: short and long. Thus, DLAA totally eliminates pattern search rather than replacing it. The perception of edges is highly probabilistic in nature and it has high temporal stability, so this method can be implemented on both GPUs and synergistic processing units (SPUs).

Subpixel reconstruction anti-aliasing (SRAA) extends MLAA by adding buffers to facilitate the undersampling of information with subpixel features at a low cost (Chajdas *et al*., 2011). The key difference compared with MLAA is that SRAA calculates blending weights using a multisampled depth buffer based on continuous edge detection, which allows the blending of weights to be determined without expensive pattern searches. Discontinuities are found using a super-resolution buffer, which allows both pixel and subpixel aliasing to be resolved, thereby making SRAA more stable during animation. In general, SRAA performs best on deferred renderers because all of the information it requires is readily available in the geometry buffers (*G*-buffers). The object in the bottom left corner of Fig. 6 shows clearly that SRAA can blur aliasing information while maintaining the fidelity. SRAA can also be used with forward rendering if the information required to determine edges is generated. This is achieved typically as part of a Z-prepass. However, SRAA works only on geometric edges, which means that the aliasing artifacts on shading edges, textures, shadows, and specular highlight boundaries cannot be resolved.

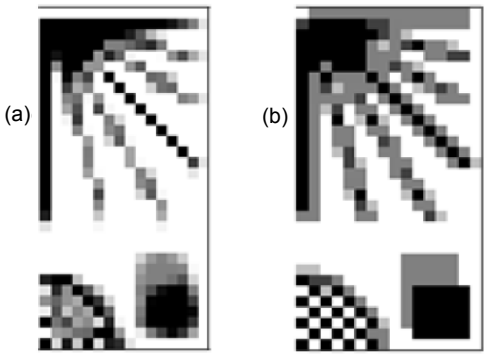

Fig. 6 Comparison of the rendering results obtained using morphological anti-aliasing (MLAA) and subpixel reconstruction anti-aliasing (SRAA)
(a) MLAA overblurs some edges because it lacks geometric information; (b) SRAA blurs alias information but maintains the fidelity (Chajdas *et al*., 2011)

Fast approximate anti-aliasing (FXAA) developed by NVIDIA (Lottes, 2011) is a cutting-edge method similar to MLAA, but it has the advantage of specific hardware features that maximize performance. FXAA ignores polygons and line edges, and simply analyzes the pixels on the screen. FXAA has two advantages: (1) it smoothens the jagged edges of all the pixels on the screen, including those inside alpha-blended textures and those produced by pixel shader effects; (2) it is very fast. Version 3 of the FXAA algorithm requires about 1.3 ms per frame on a $100 video card. Earlier versions were found to operate at double the speed compared with 4× MSAA, so enabling FXAA requires a modest 12% increase in the frame rate cost to obtain a considerable reduction in aliasing. The downside of FXAA is an increase in unwanted edge reduction inside textures or other areas.

2.2.2 Geometric anti-aliasing techniques

Geometric anti-aliasing methods are members of the MLAA family, focusing on anti-aliasing pixels intersected by precisely computed silhouette edges. Additional geometric information is used for coverage calculations and it can produce almost perfect gradients with high temporal stability.

Distance-to-edge anti-aliasing (DEAA) is an MLAA-like geometric method with anti-aliasing effects, which operates as a post-process (Malan, 2010). DEAA computes the distance-to-edge in the pixel shader during the beauty pass and writes blur hints alongside the color. It has different strengths and weaknesses compared with standard MLAA. The main advantage of DEAA is the capacity to produce very smooth edges with up to 256 different levels of blur, which avoids many of the temporal problems that affect MLAA techniques, such as crawling along rotating edges. The distance-to-edge is computed analytically, which means that there is no need to search for edges in the pixel's neighborhood, so the expensive local search step is skipped completely. The corresponding disadvantage is that DEAA requires extra space in the frame buffer to store the distance-to-edge information, which is usually 8 bits but preferably 32 bits.

Persson (2011a) developed two well-known geometric anti-aliasing algorithms. The first is geometric post-process anti-aliasing (GPAA), which uses geometry information to smooth geometric edges very accurately, without the temporal aliasing problems that affect MLAA or the need for the super-resolution buffer used by SRAA and traditional MSAA. In the post-processing step, the geometric edges are drawn as lines, and for each shaded pixel the shader computes the coverage that this pixel should have based on the primitive that shaded it. GPAA also checks the side of the line where the pixel center is located and blends the pixel with an up/down neighbor for a mostly horizontal line, or a left/right neighbor for a mostly vertical line. However, the need for a pre-processing step and potential scaling issues limit its attractiveness in real-time applications. To address these problems, an alternative approach called geometry buffer anti-aliasing (GBAA) has been developed (Persson, 2011b). During main scene rendering, the geometry information is usually stored in a separate buffer, whereas GBAA uses the channels available in an existing buffer. Anti-aliasing is then performed in the end as a fullscreen pass using the stored geometric information. GBAA eliminates the second geometry pass and allows anti-aliasing geometric edges as well as alpha-tested edges. A disadvantage is that GBAA requires additional memory for the geometry buffer.

In addition, Gjøl and Gjøl (2012) developed a smoothed lines anti-aliasing method where discontinuity edges are overdrawn for simple anti-aliasing objects on 3D-capable mobile phones, which can also be categorized as a geometric anti-aliasing technique.

Basically, this technique renders a smooth line on top of aliasing the primitive edges to cover the aliasing edge because line smoothing is widely available in hardware.

In general, geometric anti-aliasing techniques allow the rendering engine to use its knowledge of the underlying geometry to smooth edges, which greatly improves the image quality. These methods are very accurate, but they have limitations when applied to moderately complex scenes.

2.2.3 Other post-processing anti-aliasing techniques

A novel algorithm called resampling anti-aliasing (RSAA) provides an alternative approach, where filter coefficients are computed locally for each pixel by supersampling the geometry, while shading is performed only once per pixel (Reshetov, 2012). A geometric pre-pass is required to compute a bitmask for each pixel and this mask is used to fetch filter coefficients from a pre-computed lookup table in the post-processing stage. Thus, RSAA is restricted to a single color sample per pixel, but it allows multiple geometric subsamples (positions and normals). Thus, problems occur when more than two distinct surfaces overlap a pixel or there are insufficient valid samples in the neighborhood. The first situation typically corresponds to the intersection of two silhouettes when the background is also visible. Fortunately, this occurs less frequently than typical MLAA artifacts caused by mispredicting a single silhouette line. However, there is no solution to significant undersampling, except taking more samples during the runtime or pre-filtering in the off-time.

Lauritzen (2010) proposed an adaptive anti-aliasing technique that analyzes the multisampled *G*-buffer to discover planar features that can be shaded at the pixel frequency. This technique is fairly simple because it calculates the color value per sample for edges and per pixel for non-edges. Like other post-processing methods, this method requires silhouette detection and it marks the silhouettes in the stencil buffer. The anti-aliasing results obtained by this method are comparable to those obtained by SSAA, but they dispense with the redundant computations required by MSAA. Similarly, Salvi and Vidimče (2012) developed surface-based anti-aliasing (SBAA) as a real-time anti-aliasing method for deferred renders. SBAA decouples visibility from shading by aggregating point samples into larger to-be-shaded fragments, which significantly reduces the number of samples stored and shaded per pixel. Unlike the previous post-processing anti-aliasing techniques used in conjunction with deferred renders, SBAA resolves the visibility of subpixel features correctly, thereby minimizing spatial and temporal artifacts.

Lau (2003) presented a low-cost post-process anti-aliasing method that operates in the image domain. It is based on fitting curves to discontinuity edges extracted by an image edge detector. To reduce computation intensity, the algorithm pre-processes all possible edge patterns and fit curves, and then stores them into a lookup table. During runtime, it detects discontinuity edges by a simple Laplacian operator (Rosenfeld and Kak, 1982), and then constructs indices from local pixels of detected edges and their neighbors to obtain filtering information stored in the lookup table. The anti-aliasing is achieved by reshaping the discontinuity edges, according to the blending factors from the lookup table and the foreground and background pixel values. Due to low computation intensity and memory consumption, this method can be used in low-cost applications such as mobile applications. The main limitations of this method are with the edge detection and reshaping accuracy.

2.3 Discussion

Similar to Reshetov (2012), we generalize the post-filtering anti-aliasing techniques mentioned in previous sections in Table 1.

Graphic hardware vendors have made significant efforts to create suitable anti-aliasing images by integrating various anti-aliasing techniques into hardware. SSAA is the highest-quality anti-aliasing method but also the most expensive technique, while MSAA and enhanced MSAA modes make a tradeoff between quality and performance. However, test results have demonstrated that enhanced MSAA modes such as CSAA, EQAA, and CFAA fail to impress the user from the standpoint of image quality improvement. These proprietary modes can make slight improvements to visual quality but, realistically, their effects are barely noticeable compared with the basic MSAA mode upon which they are based. Aliasing effects on objects with texture transparencies are also

Table 1 Guestimate of sampling rates for post-filtering anti-aliasing techniques

Anti-aliasing method	Depth	Coverage	Geometry	Shading value	Storage	Bandwidth
No anti-aliasing	Once	N/A	Once	Once	Once	Once
SSAA	All	N/A	Once	Once	All	All
MSAA, CSAA, EQAA	Many	Many	Once	Once	Many	Some
MLAA, FXAA, SMAA 1×	Once	N/A	Once	Once	Some	Some
CFAA	N/A	Many	Once	Once	Many	Some
Geometric methods	N/A	N/A	∞	Once	Some	Some
DLAA	Once	N/A	Once	Once	Some	Some
SMAA 4×	Some	N/A	Some	Some	Some	Some
SRAA	Many	N/A	Many	Some	Many	Many
SBAA	Many	Many	Many	Once	Some	Some
RSAA	Many	Many	Many	Once	Some	Some

'Some' indicates the typical sampling rates required for anti-aliasing computations; 'Once', 'Many', and 'All' for a single sample per pixel, many subsamples per pixel, and most subsamples per pixel, respectively

not removed by MSAA, despite the availability of newer DirectX 10/11 techniques such as alpha-to-coverage; thus, there is a need for further transparent texture anti-aliasing methods. Adaptive anti-aliasing barely works with Radeon GPUs, but NVIDIA's transparent supersampling is relatively reliable with DirectX 10 and 11 games.

MLAA methods are highly economical in terms of conserving the bandwidth, and may be good options for deferred shading mode. They operate over a wide range of rasterization and ray tracing applications, and are even slightly effective in cleaning up aliasing artifacts that affect transparent textures (although not nearly as effective as a true texture transparency anti-aliasing method). On the contrary, geometric methods generate accuracy anti-aliasing effects by precisely computing silhouette edges. However, the cost of computation limits its application in moderately complex scenes. SMAA 4× handles temporal and spatial aliasing artifacts by a very accurate edge and pattern detection scheme, supplementing MLAA with temporal and spatial multi/super sampling. By this way, the method is about 1.8 times faster than MSAA 8×, and needs only 43% of the memory used by MSAA 8×. Unlike these, RSAA allows a significant bandwidth reduction in deferred rendering applications by restricting a single color sample per pixel, but with multiple geometric subsamples. In fact, RSAA provides higher anti-aliasing quality than any method in the MLAA family with a single color sample per pixel. However, there is no solution to significant undersampling, except taking more samples during the run-time or pre-filtering in the off-time.

3 Pre-filtering based anti-aliasing implementations

Pre-filtering based anti-aliasing techniques operate in the opposite manner to supersamping. The techniques change the spectrum of the input signal by attenuating all of the high-frequency components that might be aliased by low-pass filtering before sampling at pixel rates (Guenter and Tumblin, 1996). Theoretically, pre-filtering methods can completely eliminate aliasing artifacts whereas other anti-aliasing techniques can only reduce aliasing effects.

3.1 Analytic anti-aliasing techniques

The ideal method for analytic anti-aliasing would be to compute the Fourier transform of the input image multiplied by a filter function in the frequency domain, before performing the exact inverse Fourier transform (Duff, 1989). However, this method is impractical because the computational requirements are complex and expensive. According to the convolution theorem (Weisstein, 2014), the operation described above is equal to the convolution of an input image and a filter function in the temporal domain. Therefore, most analytic anti-aliasing algorithms evaluate the convolution integral at every point in the sampling grids using various methods, based on either quadrature or lookup table.

The first truly analytic anti-aliasing technique was proposed by Catmull (1978). This method clips polygons to the extent of each pixel and weights the contribution of each polygon to the pixel based on its clipped area. However, this method supports only constant shaded polygons and box filtering. Thus, aliasing artifacts still cannot be removed entirely. Feibush *et al*. (1980) first proposed an anti-aliasing technique that uses a 2D lookup table, which splits the input polygons into triangles and evaluates the convolution of the generated triangles with a filtering function. However, the lookup table becomes very complex if the filter function is not circularly symmetric. Catmull (1984) extended this method to temporal anti-aliasing and proposed an efficient algorithm to solve the hidden surface problem at the pixel level, which is necessary for pre-filtering anti-aliasing techniques. Duff (1989) described an exact convolution computation method for 2D anti-aliasing based on a simple polygon contour integral. Theoretically, the technique can support any filter model with polynomial approximations, as well as linear functions defined on the polytopes. However, this method is not easily extended to three dimensions because it requires potentially expensive clips of the boundaries of each pixel. This method was accelerated by Guenter and Tumblin (1996) by evaluating the convolution integral in one direction using a small 2D lookup table and sampling in the other direction. With the equivalent root mean square (RMS) error, this analytic technique is significantly faster than either uniform or jittered supersampling. Instead of convolution, Manson and Schaefer (2011) introduced a Haar wavelet-based anti-aliasing method for arbitrary polygons or fonts bounded by Bézier curves in two dimensions, as well as oriented triangle meshes in three dimensions. Basically, this method is equivalent to applying a box filter to the rasterized image. Unfortunately, this method is computationally intensive because it needs to determine the wavelet coefficients, and the computational demand increases rapidly with the resolution.

To obtain a closed-form solution, the analytic methods described above require that an integrable function be defined on the projected surface. Thus, only the geometries represented by linear functions or polynomials are suitable for analytic anti-aliasing techniques. A non-sampled anti-aliasing technique was introduced by Auzinger *et al*. (2013a) to avoid this restriction, supporting separable filter functions with arbitrary kernel sizes, as well as general shading methods such as texture mapping or complex illumination models. This method comprises three stages: (1) the relevant primitives that need to be considered for pre-filtering in each sample location are listed; (2) the contributions of the relevant primitives for each sample to the final sample color are computed by analytically evaluating the convolution; (3) the collected primitives are rasterized and the fragments are blended with the weights from the previous stage. When this approach is applied as an analytic visibility method (Auzinger *et al*., 2013b), it does not support intersecting primitives and it requires a consistently orientated input.

3.2 Discussion

Crow (1977) first proposed a pre-filtering method for anti-aliasing and most subsequent research in this field has focused on three areas: a closed-form solution to support arbitrary filter kernels, highly effective integral computation, and expansion of the filter functions. Pre-filtering techniques can be used to generate better anti-aliasing effects than post-filtering methods. However, all pre-filtering methods require analytic or piecewise analytic descriptions of the input image, as well as complex computations that consume most of the time when evaluating convolution integrals or wavelet coefficients. Anti-aliasing transparent textures and visibility determination for transparent objects are also major challenges for pre-filtering techniques. These issues further restrict the practical applications of these methods.

4 Analysis of anti-aliasing techniques

Hardware anti-aliasing techniques can take full advantage of GPUs to generate suitable anti-aliasing effects via their integration in a graphics pipeline system. Indeed, they are approximate sampling-based solutions, and thus the aliasing quality can be improved at the cost of an increased sampling rate. To balance the quality and performance, most hardware anti-aliasing techniques perform subsampling and filtering only at the silhouette edges of objects. Thus,

they are used widely in real-time applications such as games and animations. However, the deferred shading pipeline population makes hardware anti-aliasing techniques less effective because of the bandwidth overheads, where all of the sampling data has to be written to an output buffer before further processing.

Post-processing anti-aliasing techniques have a set of unique characteristics; e.g., they are independent of the rendering pipeline systems and highly economical in terms of conserving the bandwidth. Thus, they are highly suitable for deferred shading engines and ray tracing applications. Indeed, they detect silhouette edges based on discrete color (or depth) data and then blend the pixels around the found edges using a digital filter, which produces a plausible anti-aliasing effect. However, the quality is reduced in animation sequences because the silhouettes might be reconstructed in different ways in subsequent frames. These techniques can also have detrimental effects on image quality if small text is critical. Compared with hardware anti-aliasing techniques, the computational costs and the possibility of incomplete aliasing elimination are increased.

By contrast, pre-filtering methods remove high frequency spatial components before sampling to produce alias-free images. They are more economical in terms of bandwidth conservation because they eliminate the supersampling process. However, all pre-filtering techniques have the restriction that the input geometries need to be pre-filtered and a range of filter kernels should be employed. These techniques require analytic or piecewise analytic input images, as well as complexity convolution or wavelet computation. These disadvantages mean that they are rarely used in rasterization and practical applications. However, empirical research demonstrates that pre-filtering is more appropriate for high quality 2D images that contain lines, regions, and texts rather than images with 3D scenes. Table 2 shows a high level comparison of anti-aliasing techniques from perspectives of anti-aliasing effects, computational costs, input restrictions, and bandwidth.

5 Conclusions

Anti-aliasing has been an active research area in computer graphics for decades and numerous anti-aliasing techniques have been developed. This review classifies these anti-aliasing techniques into two categories: post-filtering based methods and pre-filtering based implementations. We discuss post-filtering based techniques through classifying them into hardware implementations and post-processing implementations. Hardware anti-aliasing techniques have been widely adopted by graphic hardware vendors and used in real-time applications. As they are deeply integrated into graphic pipeline systems, it is difficult to use them in deferred shading models. Thus, decoupling sampling from pipeline systems has become a new research topic. Reducing the sampling rate and improving anti-aliasing quality are also persistent research directions. Post-processing techniques are characterized by detecting silhouette edges from image data and performing highly effective filtering on pixels around the edges found. The research performed in recent years still centralizes on quickly detecting accurate silhouette edges through the combination of graphic and image-wise knowledge as well as reducing filtering computation costs. Pre-filtering-based methods stem from signal process theory. Most research performed focuses on three aspects: a closed-form solution to support arbitrary filter kernels, highly effective integral computation, and expansion of filter functions. Different strategies have been developed to aim at these directions. However, a common problem of these strategies is performance; these methods can hardly be used in real-time applications. Thus, there is large research space to attract the attention of researchers on reducing integral computation cost, expanding input scope and filter functions, and seeking a closed-form solution for arbitrary filter kernels.

It has been verified that several GPU-driven anti-aliasing approaches run in a few milliseconds on today's GPUs, which makes them practical for real-

Table 2 High-level comparison of anti-aliasing techniques

Anti-aliasing category	Anti-aliasing quality	Performance	Input restriction	Bandwidth
Hardware methods	Low	High	None	High
Post-processing methods	Medium	Medium	None	Low
Pre-filtering-based methods	High	Low	Analytic	Low

time rendering applications. They will also scale to take advantage of increased bandwidth and computation on future GPUs, e.g., GPGPU and new features of DirectX11. We believe that the next step could be combining ideas from hardware and the MLAA family to integrate heuristic shading weights with accurate geometric weights for higher quality than either can create alone in practice. Even further implementations could absorb the essences from pre- and post-filtering to construct approaches to achieve fine aliased images with either static 2D geometries or dynamic 3D scenes.

References

Akeley, K., 1993. Reality engine graphics. Proc. 20th Annual Conf. on Computer Graphics and Interactive Techniques, p.109-116. [doi:10.1145/166117.166131]

Amanatides, J., 1984. Ray tracing with cones. Proc. 11th Annual Conf. on Computer Graphics and Interactive Techniques, p.129-135. [doi:10.1145/800031.808589]

AMD, 2011. EQAA Modes for AMD 6900 Series Graphics Cards. Available from http://developer.amd.com/wordpress/media/2012/10/EQAA%2520Modes%2520for%2520AMD%2520HD%25206900%2520Series%2520Cards.pdf [Accessed on Mar. 1, 2014].

Andreev, D., 2011. Anti-aliasing from a different perspective. Game Developers Conf., p.1-55.

Auzinger, T., Guthe, M., Jeschke, S., 2012. Analytic anti-aliasing of linear functions on polytopes. *Comput. Graph. Forum*, **31**(2pt1):335-344. [doi:10.1111/j.1467-8659.2012.03012.x]

Auzinger, T., Musialski, P., Preiner, R., *et al*., 2013a. Non-sampled anti-aliasing. Proc. 18th Int. Workshop on Vision, Modeling and Visualization, p.169-176. [doi:10.2312/PE.VMV.VMV13.169-176]

Auzinger, T., Wimmer, M., Jescke, S., 2013b. Analytic visibility on the GPU. *Comput. Graph. Forum*, **32**(2pt4):409-418. [doi:10.1111/cgf.12061]

Biri, V., Herubel, A., Deverly, S., 2010. Practical morphological antialiasing on the GPU. Proc. ACM SIGGRAPH, Article 45. [doi:10.1145/1837026.1837085]

Catmull, E., 1978. A hidden-surface algorithm with anti-aliasing. *ACM SIGGRAPH Comput. Graph.*, **12**(3):6-11. [doi:10.1145/965139.807360]

Catmull, E., 1984. An analytic visible surface algorithm for independent pixel processing. *ACM SIGGRAPH Comput. Graph.*, **18**(3):109-115. [doi:10.1145/964965.808586]

Chajdas, M.G., McGuire, M., Luebke, D., 2011. Subpixel reconstruction antialiasing for deferred shading. Proc. Symp. on Interactive 3D Graphics and Games, p.15-22. [doi:10.1145/1944745.1944748]

Chan, E., Durand, F., 2005. Fast prefiltered lines. *GPU Gems*, **2**:345-359.

Crow, F.C., 1977. The aliasing problem in computer-generated shaded images. *Commun. ACM*, **20**(11):799-805. [doi:10.1145/359863.359869]

Deering, M., Winner, S., Schediwy, B., *et al.*, 1988. The triangle processor and normal vector shader: a VLSI system for high performance graphics. *ACM SIGGRAPH Comput. Graph.*, **22**(4):21-30. [doi:10.1145/378456.378468]

Duff, T., 1989. Polygon scan conversion by exact convolution. Proc. Int. Conf. on Raster Imaging and Digital Typography, p.154-168.

Feibush, E.A., Levoy, M., Cook, R.L., 1980. Synthetic texturing using digital filters. *ACM SIGGRAPH Comput. Graph.*, **14**(3):294-301. [doi:10.1145/965105.807507]

Genetti, J., Gordon, D., Williams, G., 1998. Adaptive supersampling in object space using pyramidal rays. *Comput. Graph. Forum*, **17**(1):29-54. [doi:10.1111/1467-8659.00214]

Gjøl, M., Gjøl, M., 2012. Inexpensive antialiasing of simple objects. *In*: Engel, W. (Ed.), GPU Pro 3. A.K. Peters Ltd., Natick, USA, p.169-178.

Guenter, B., Tumblin, J., 1996. Quadrature prefiltering for high quality antialiasing. *ACM Trans. Graph.*, **15**(4):332-353. [doi:10.1145/234535.234540]

Heckbert, P.S., Hanrahan, P., 1984. Beam tracing polygonal objects. *ACM SIGGRAPH Comput. Graph.*, **18**(3):119-127. [doi:10.1145/964965.808588]

Jimenez, J., Gutierrez, D., Yang, J., *et al.*, 2011a. Filtering approaches for real-time anti-aliasing. ACM SIGGRAPH Courses.

Jimenez, J., Masia, B., Echevarria, J.I., *et al.*, 2011b. Practical morphological anti-aliasing. *In*: Engel, W. (Ed.), GPU Pro 2. A.K. Peters Ltd., Natick, USA, p.95-114.

Jimenez, J., Echevarria, J.I., Sousa, T., *et al.*, 2012. SMAA: enhanced subpixel morphological antialiasing. *Comput. Graph. Forum*, **31**(2pt1):355-364. [doi:10.1111/j.1467-8659.2012.03014.x]

Lau, R.W.H., 2003. An efficient low-cost antialiasing method based on adaptive postfiltering. *IEEE Trans. Circ. Syst. Video Tech.*, **13**(3):247-256. [doi:10.1109/TCSVT.2003.809825]

Lauritzen, A., 2010. Deferred rendering for current and future rendering pipelines. SIGGRAPH Course: Beyond Programmable Shading, p.1-34.

Leler, W.J., 1980. Human vision, anti-aliasing, and the cheap 4000 line display. *ACM SIGGRAPH Comput. Graph.*, **14**(3):308-313. [doi:10.1145/965105.807509]

Lottes, T., 2011. FXAA. NVIDIA White Paper. Available from http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf [Accessed on Mar. 1, 2014].

Malan, H., 2010. Edge anti-aliasing by post-processing. *In*: Engel, W. (Ed.), GPU Pro. A.K. Peters Ltd., Natick, USA, p.265-289.

Manson, J., Schaefer, S., 2011. Wavelet rasterization. *Comput. Graph. Forum*, **30**(2):395-404. [doi:10.1111/j.1467-8659.2011.01887.x]

McCool, M.D., 1995. Analytic antialiasing with prism splines.

Proc. 22nd Annual Conf. on Computer Graphics and Interactive Techniques, p.429-436. [doi:10.1145/218380.218499]

Ohta, M., Maekawa, M., 1990. Ray-bound tracing for perfect and efficient anti-aliasing. *Vis. Comput.*, **6**(3):125-133. [doi:10.1007/BF01911004]

Persson, E., 2011a. Geometric Post-Process Anti-Aliasing. Available from http://www.humus.name/index.php?page=3D&ID=86 [Accessed on Mar. 1, 2014].

Persson, E., 2011b. Geometry Buffer Anti-Aliasing. Available from http://www.humus.name/index.php?page=3D&ID=87 [Accessed on Mar. 1, 2014].

Reshetov, A., 2009. Morphological antialiasing. Proc. Conf. on High Performance Graphics, p.109-116. [doi:10.1145/1572769.1572787]

Reshetov, A., 2012. Reducing aliasing artifacts through resampling. Proc. 4th ACM SIGGRAPH/Eurographics Conf. on High Performance Graphics, p.77-86. [doi:10.2312/EGGH/HPG12/077-086]

Rosenfeld, A., Kak, A.C., 1982. Digital Picture Processing (2nd Ed.). Morgan Kaufmann, Massachusetts, p.84-112.

Salvi, M., Vidimče, K., 2012. Surface based anti-aliasing. Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games, p.159-164. [doi:10.1145/2159616.2159643]

Sander, P.V., Hoppe, H., Snyder, J., *et al.*, 2001. Discontinuity edge overdraw. Proc. Symp. on Interactive 3D Graphics, p.167-174. [doi:10.1145/364338.364390]

Shannon, C.E., 1949. Communication in the presence of noise. *Proc. IRE*, **37**(1):10-21. [doi:10.1109/JRPROC.1949.232969]

Shinya, M., Takahashi, T., Naito, S., 1987. Principles and applications of pencil tracing. *ACM SIGGRAPH Comput. Graph.*, **21**(4):45-54. [doi:10.1145/37401.37408]

Thomas, D., Netravali, A.N., Fox, D.S., 1989. Anti-aliased ray tracing with covers. *Comput. Graph. Forum*, **8**(4):325-336. [doi:10.1111/j.1467-8659.1989.tb00514.x]

Weisstein, E.W., 2014. Convolution Theorem. Available from http://mathworld.wolfram.com/ConvolutionTheorem.html [Accessed on Mar. 1, 2014].

Whitted, T., 1980. An improved illumination model for shaded display. *Commun. ACM*, **23**(6):343-349. [doi:10.1145/358876.358882]

Woligroski, D., 2011. Anti-Aliasing Analysis, Part 1: Settings and Surprises. Tom's Hardware.

Young, P., 2007. Coverage Sampling Antialiasing. NVIDIA White Paper. Available from http://www.nvidia.com/object/coverage-sampled-aa.html [Accessed on Mar. 1, 2014].