DOI: 10.1111/cgf.12476
Pacific Graphics 2014
J. Keyser, Y. J. Kim, and P. Wonka
(Guest Editors)

Volume 33 (2014), Number 7

Sub-Pixel Anti-Aliasing Via Triangle-Based Geometry Reconstruction

Wenjun Du[1] Jieqing Feng[1,†] Baoguang Yang[2]

[1]State Key Lab of CAD&CG, Zhejiang University ({duwenjun, jqfeng}@cad.zju.edu.cn) † - Corresponding author
[2]Qualcomm Incoporated

Abstract

Anti-aliasing has recently been employed as a post-processing step to adapt to the deferred shading technique in real-time applications. Some of these existing algorithms store supersampling geometric information as geometric buffer (G-buffer) to detect and alleviate sub-pixel-level aliasing artifacts. However, the anti-aliasing filter based on sampled sub-pixel geometries only may introduce unfaithful shading information to the sub-pixel color in uniform-geometry regions, and large G-buffer will increase memory storage and fetch overheads. In this paper, we present a new Triangle-based Geometry Anti-Aliasing (TGAA) algorithm, to address these problems. The coverage triangle of each screen pixel is accessed, and then, the coverage information between the triangle and neighboring sub-pixels is stored in a screen-resolution bitmask, which allows the geometric information to be stored and accessed in an inexpensive manner. Using triangle-based geometry, TGAA can exclude irrelevant neighboring shading samples and achieve faithful anti-aliasing filtering. In addition, a morphological method of estimating the geometric edges in high-frequency geometry is incorporated into the TGAA's anti-aliasing filter to complement the algorithm. The implementation results demonstrate that the algorithm is efficient and scalable for generating high-quality anti-aliased images.

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

1. Introduction

Anti-aliasing is a technique to remove or alleviate aliasing artifacts and improve the visual quality of synthetized images. The classical approach, Super-Sampling Anti-Aliasing (SSAA), evaluates multiple sub-pixel shading samples and then compute thier average or weighted averages for each pixel. Another widely used method, Multi-Sampling Anti-Aliasing (MSAA) [Ake93], is a specific optimization of SSAA and performs notably better. The MSAA executes the fragment program once per pixel and supersamples the depth and stencil values. MSAA has long been the most popular anti-aliasing solution in various graphics applications.

Deferred shading [GPB04, Har04] is a powerful screen-space shading technique. It gathers data on the first shading pass and performs real shading composition on the second pass. The method can achieve complex illumination effects without a significant decrease in performance. Unfortunately, MSAA is not compatible with deferred shading because the real shading stage of deferred shading is performed in image space via geometric buffers (G-buffer). In such a context, MSAA will degenerates to SSAA and loses its performance advantages [Koo07, Shi05].

In recent years, new anti-aliasing algorithms have emerged that are performed as a post-processing step and are compatible with the deferred shading technique. The post-processing anti-aliasing algorithms can be classified into two categories based on whether geometric information is employed: the image-based approach and the geometry-assisted approach.

The image-based approach commonly detects and analyzes aliased features in a screen-resolution shading image and then performs heuristic filtering to alleviate the aliasing artifacts. These algorithms exhibit good performance in general, but they are subject to insufficient geometric infor-

† Chairman Eurographics Publications Board

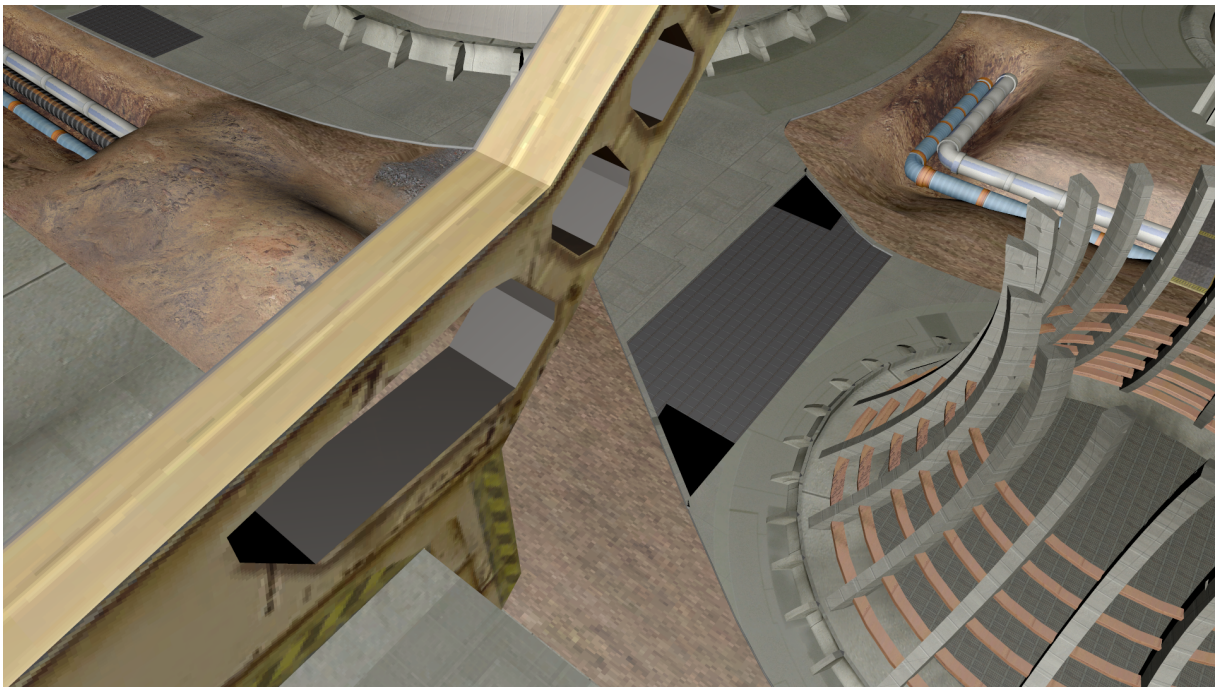

Figure 1: *Anti-aliasing result of TGAA in Powerplant (470K triangles),* 1920×1080*, 4.8 ms for the entire algorithm.*

mation and tend to overblur certain visual features or details in the image.

The geometry-assisted anti-aliasing approach is based on the fact that the shading costs are typically higher than the geometric computation costs [CML11]. The ideal color of the screen pixel can be seen as the average color of the sub-pixels, some geometry-assisted approach stores the pixel-level shading information and the sub-pixel-level geometric information, instead of the sub-pixel-level shading information in SSAA, to minimize the shading operations in the algorithm. These algorithms reconstruct the sub-pixel color according to the geometric similarity between this sub-pixel and its neighboring shading samples, and then achieve reasonable anti-aliasing color filtering. However, the high resolution G-buffer that represent super-sampling scene geometry will increase memory storage and access overheads.

As another geometry-assisted solution, [DYF08, LMS-G14] represent the sub-pixel-level scene geometry by the coverage triangle of screen pixels, which refers to the triangle closest to the viewpoint that overlaps the center of each screen pixel. Inspired by this, we present a Triangle-based Geometry Anti-Aliasing algorithm (TGAA), which has inexpensive G-buffers and faithful anti-aliasing filter. The sub-pixel-level geometries are represented by the coverage information between the coverage triangle and its neighboring sub-pixels for each screen pixel, and can be stored in a screen-resolution bitmask. The anti-aliasing filter uses information stored in the G-buffers to select neighboring shading samples and then reconstruct sub-pixel colors.

The main contributions of TGAA include the following:

- TGAA employs a triangle-based G-buffer generated from the original coverage triangle of each screen pixel and proposes a faithful geometric filter to alleviate aliasing.
- The triangle coverage information is stored in a screen-resolution texture representing sub-pixel-level geometry, and thus, the additional rendering pass for high resolution G-buffer generation is not required. Meanwhile, the small size storage as a bitmask significantly reduces the memory storage and access overheads of the algorithm.
- A morphological method is employed to estimate the anti-aliased color of the sub-pixel near thin geometric details.

2. Related Work

Aside from SSAA and MSAA, the subsequent algorithms, e.g., CSAA [YN06] and EQAA [AMD11], inherit the limitation of being incompatible with the deferred shading technique. Lauritzen [Lau10] combined tiled deferred shading with MSAA with detection of surface discontinuities. Holländer et al. [HBE13] improved the super-sampling aliasing framework to adapt the deferred shading context. Barringer et al. [BAM13] proposed asynchronous adaptive anti-aliasing using shared memory architecture between the GPU and CPU as an accelerated anti-aliasing solution.

In recent years, post-processing anti-aliasing algorithms have been proposed, and are well adapted to the deferred shading framework. Detailed surveys are given in [JGY*11]. Only the most relevant works are reviewed in this section.

2.1. Image-Based Anti-Aliasing Approach

Among the post-processing anti-aliasing algorithms, image-based approaches are widely used due to their simplicity and high efficiency. The input to an image-based anti-aliasing algorithm is a screen-resolution rasterized image. Various aliased features or details in the image are detected or estimated. An anisotropic filtering method is then designed and performed to alleviate these aliasing artifacts. In general, these algorithms can generate plausible results efficiently.

The directionally adaptive edge anti-aliasing filter [IYP09] is one of the pioneer works of post-processing anti-aliasing. For each pixel on the feature edge, this filter uses integration to estimate the weights of the adaptive filter, which are the coverages of the geometry on the pixel.

Morphological anti-aliasing (MLAA) [Res09] is a representative image-based anti-aliasing algorithms. It estimates the underlying edge properties by analyzing the color discontinuities. However, this method is implemented on the CPU using vector instructions and is not suitable for real-time GPU implementation. Jimenez's MLAA [JME*11] achieves high performance by fully exploiting hardware-supported texture operations, such as edge detection, area coverage estimation, precomputed blending patterns, and bilinear interpolation. Although it is regarded as successful image-based anti-aliasing algorithms, it still cannot detect or process sub-pixel-level aliasing artifacts well. In some cases, it may overblur features or details in the scene due to a lack of sub-pixel geometric information. Their subsequent work, enhanced subpixel morphological antialiasing (SMAA) [JESG12], solves the sub-pixel feature problem by combining MLAA with an additional multi/super-sampling step. In addition, this method improves such features as sharp edge estimation, pattern processing, and distance computation. However, certain aliasing artifacts are still omitted due to the limited number of precomputed patterns and lack of geometric information.

Directionally localized anti-aliasing (DLAA) [And11] is another image-based solution that blurs the feature edges along the estimated directions. Both vertical and horizontal blurs are adopted to produce a gradient effect along the aliased feature edges. Fast approximate anti-aliasing (FXAA) [Lot11] addresses sub-pixel features efficiently via fast feature detection and attenuation processes while sacrificing the anti-aliasing quality and accuracy to some extent. Reducing aliasing artifacts through resampling (RSAA) [Res12] adopts a high resolution sampling image to reconstruct the pixel color of the feature by computing the offsets of bilinear resampling. However, the method is not effective enough when more than two distinct surfaces cover a pixel or when there are a sufficient number of valid samples in its neighboring pixels.

2.2. Geometry-Assisted Anti-Aliasing Approach

Compared with image-based anti-aliasing approaches, geometry-assisted anti-aliasing algorithms consider additional sub-pixel level geometric information to estimate geometric features or details. As a result, they facilitate more accurate "feature pixel" filtering. However, they suffer from additional storage and memory accessing costs.

Distance-to-edge anti-aliasing (DEAA) [JGY*11] computes the distance from the pixel center to the nearest triangle edge at the sub-pixel precision and then uses the distance to estimate the filtering weight. Geometric post-process anti-aliasing (GPAA) [Per11a] and geometry buffer anti-aliasing (GBAA) [Per11b] algorithms compute feature pixel coverages using additional edge information. Unfortunately the three aforementioned algorithms still cannot handle sub-pixel-level aliasing artifacts correctly, as only screen resolution geometric information is adopted.

Subpixel reconstruction anti-aliasing (SRAA) [CML11] combines a screen-resolution shading image with additional supersampling position, depth or normal G-buffers and is capable of reconstructing sub-pixel-level geometric details. Furthermore, SRAA functions steadily for animated scenes. SRAA respects the original geometric boundaries more faithfully than image-based anti-aliasing approaches. Because the edge estimation of SRAA is based on pure sampled geometry, unfaithful filtering may occur when a sample has similar geometric attributes to its neighboring shading information. Furthermore, the storage and texture accessing costs of the high resolution G-buffer in SRAA are heavy, which is also a common problem for geometry-assisted anti-aliasing algorithm, particularly if the G-buffer is large.

3. Triangle-Based Anti-Aliasing Algorithm

3.1. Algorithm Overview

Employing sub-pixel geometry in anti-aliasing filtering introduces two main challenges. First, in some cases the sampled sub-pixel geometries (e.g., depth or normal) exhibit little geometric variation within the filtering region, and the filter may degenerate to a blurring process and introduce unfaithful shading information to sub-pixel color. The second challenge is the storage and access overheads of the large G-buffers.

To overcome these challenges, TGAA adopts triangle-based geometry as the basis of its anti-aliasing filtering, which can represent both sampled sub-pixel geometry and primitive information of the scene. Thus triangle-based geometry is more reliable for reconstructing the sub-pixel color, compared to considering the geometric similarity be-

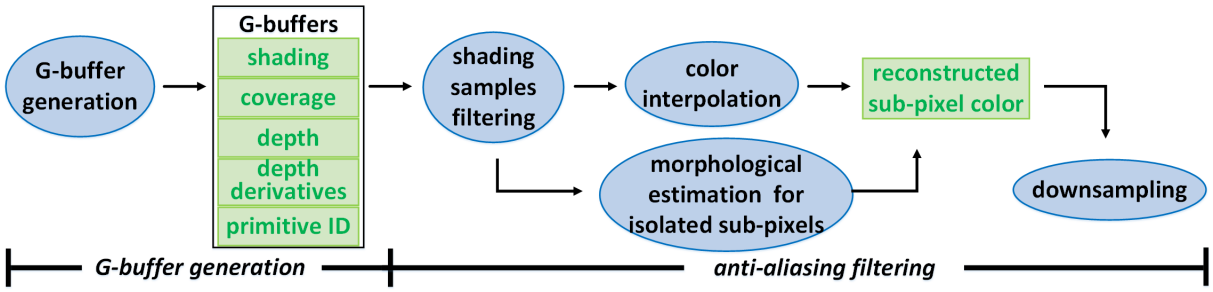

Figure 2: *Flowchart of the proposed algorithm in two passes, where rectangles denote data and ellipses denote procedure.*

tween sub-pixels alone. Meanwhile, triangle-based geometry can be stored in a small size texture to reduce the memory requirement (see details in section 3.2). The flowchart of the proposed algorithm is shown in Figure 2. TGAA includes two rendering passes: G-buffer generation and anti-aliasing filtering.

3.2. G-buffer generation

As described in Figure 2, the G-buffer generation pass generates the coverage information, the primitive ID information, the depth information accompanied by the shading information. The four generated information are all stored in screen resolution texture, while the coverage information can represent sub-pixel geometry.

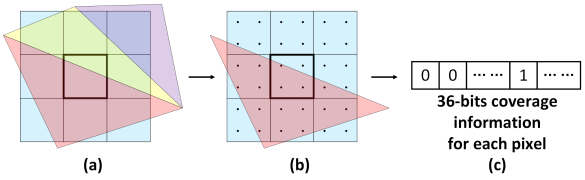

Figure 3: *(a) A screen pixel (center) and its 8 adjacent pixels with its surrounding triangle geometries. (b) There are 36 sub-pixels (black dots) for 4× anti-aliasing in this area. The coverage triangle (red) of the center pixel is accessed, and its coverage with the 36 neighboring sub-pixels is computed. (c) A 36-bit bitmask is generated as the sub-pixel geometric information.*

Coverage information: The sub-pixel-level geometry in TGAA is based on the coverage triangle of each pixel center, which can be retrieved from the geometry shader stage of the rendering pipeline. The coverage triangle can represent the original geometry of the scene and approximately reconstruct the real sub-pixel color with surrounding shading samples. Rather than directly storing the vertex coordinates of the coverage triangle, we can pre-compute the coverage information between the coverage triangle and its neighboring sub-pixels in this pass. Thus, the triangle-based geometric information is converted into a binary mask, and a large G-buffer is avoided.

TGAA supports different sub-pixel sampling rates or patterns and different filtering kernel of neighboring shading samples according to the application's requirements. Without loss of generality, assume that 4× TGAA is performed with an 8-neighbor pixel filtering kernel (Figure 3). For the coverage triangle of the current pixel, we will test whether the triangle covers 4×9 surrounding sub-pixels. The resulting bitmask will record this coverage information using 36 bits per pixel. This sub-pixel-level geometry has a smaller size and is less expensive to fetch compared to other sub-pixel anti-aliasing algorithms. For example, SRAA directly adopts $4 \times$ depth as the sub-pixel information: 128 bits per pixel. Moreover considering normal information of sub-pixel resolution, the G-buffer size will become 256 bits. We will perform a detailed G-buffer size comparison in Section 4.

Determining whether 36 neighboring sub-pixels lie inside the coverage triangle can be achieved by computing the barycentric coordinates of the sub-pixels, and this cost dominates the runtime of the entire pass. Instead of a brute-force computation, an incremental determination strategy [Pin88] is employed to avoid time-consuming matrix calculations and improve the performance. We choose one sub-pixel $p^*(x^*, y^*)$ between the 36 sub-pixels and compute its barycentric coordinates $(\omega_0, \omega_1, \omega_2)$. The coverage of the other 35 sub-pixels can be determined incrementally via Equation (1). In this manner, the bitmask for the edge pixel is determined efficiently.

$$\begin{cases} \omega_0(x,y) = \dfrac{\partial w_0}{\partial x^*}(x-x^*) + \dfrac{\partial w_0}{\partial y^*}(y-y^*) + \omega_0(x^*,y^*) \\ \omega_1(x,y) = \dfrac{\partial w_1}{\partial x^*}(x-x^*) + \dfrac{\partial w_1}{\partial y^*}(y-y^*) + \omega_1(x^*,y^*) \\ \omega_2(x,y) = 1 - \omega_0(x,y) - \omega_1(x,y) \end{cases} \tag{1}$$

Primitive ID and depth information: In addition to the coverage information, the primitive ID of the coverage triangle and the screen-resolution depth information are also

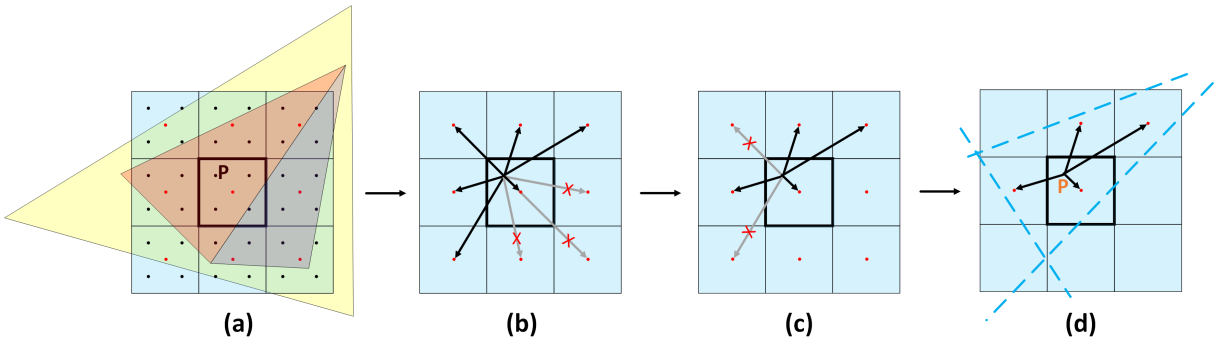

Figure 4: *An example of shading sample filtering for sub-pixel P. (a) Two foreground triangles (orange and grey) and one background triangle (yellow) near P. The red dots represent the shading samples stored in the G-buffer. (b) Exclusion of irrelevant shading samples via coverage information. (c) Exclusion of unfaithful shading samples via depth information and primitive ID information where multiple triangles (orange and yellow triangles) cover sub-pixel P. (d) 4 shading samples are identified. The blue dashed lines is the approximated illustration of the estimated sub-pixel edges near P.*

stored in this pass; these pieces of information can be acquired from the rendering pipeline. The information in the depth buffer includes the depth value of the pixel center and the partial deriviates of the depth (dz/dx and dz/dy) on the coverage triangle for each pixel.

The generated primitive ID information is stored in 32-bit format to meet the requirements of complex scenes. The depth buffer is also stored in 32-bit format, containing depth in 16 higher bits and depth deriviates in 16 lower bits (8 for the x-direction and 8 for the y-direction), respectively. The relatively low precision of the depth information is sufficient in our algorithm, because the depth information is only used to determine the relative positions of triangles. Furthermore, the primitive ID buffer is also used to estimate the color of "isolated" sub-pixels (see details in Section 3.3).

To summarize, 24-bit R8G8B8 format shading information, 36-bit coverage information, 32 bits depth information and 32-bit primitive ID information are generated in this pass. These 124 bits of data can be packed into a single 128-bit format texture. One advantage of TGAA is that the four types of information are all in screen resolution and can be stored in a single pass, whereas SRAA requires the screen resolution color information and high resolution geometric information using two passes. Furthermore, the relatively small G-buffer and compact storage require less memory requirement than the high resolution G-buffer in the algorithm.

3.3. Anti-Aliasing Filtering

In this stage, the anti-aliased color of each screen pixel is determined by sub-pixel-level filtering. As shown in Figure 2, this stage includes the following steps:

Shading sample filtering: In this step, the filter determines how the shading samples in the filtering kernel of 8-neighbor pixel area contribute to the sub-pixel color. Relevant shading samples are identified with the G-buffer information in the 8-neighbor pixels.

Figure 4 illustrates a typical example of shading sample filtering for sub-pixel P in TGAA. We will explain the two main steps of the filtering with this example.

1) Excluding irrelevant shading samples via coverage information. In this step, the coverage bitmasks stored in all 9 pixels are accessed. Accessing the 9 coverage bitmasks, we can check the coverage between P and the three triangles. In this way, we know the grey triangle and its three corresponding pixels (right, bottom right and bottom) are irrelevant (see Figure 4 (b)).

2) Excluding irrelevant shading samples via depth and primitive ID information when multiple candidate triangles cover sub-pixel P. Both the background yellow and the foreground orange triangles meet the condition of coverage, but the orange triangle is the actual coverage triangle of P that we want to identify. Computing the depth values of P on each candidate triangle can quickly eliminate those irrelevant triangles and identify the real coverage triangle since the actual coverage triangle is closest to the viewpoint. Thus, the computed depth values are compared and the smallest value is what we want to acquire (Figure 4 (c)).

Because of the possible floating point error in depth computation, a little threshold (e.g., 10-5) is used to avoid erroneous triangle elimination. If the difference of two computed depth values is smaller than the threshold, the two corresponding triangles can be considered identical. In this case, primitive ID is adopted as a final check, preventing the case that two triangles are different but have very close depth at P.

Color interpolation: The sub-pixel color can be interpolated once the identified neighboring shading samples are

determined. Whereas other algorithms compute the filter weight using a geometric similarity function, TGAA can take the 2-D distance between the sub-pixel and i shading samples as the filter weights (ω_i) since the actual filter is within the triangle. In this manner, the reconstructed sub-pixel color (c_{sub}) is estimated as distance-weighted average of shading sample colors (s_i), as denoted in Equation 2. The final pixel color can be obtained by averaging the reconstructed sub-pixel colors.

$$c_{sub} = \frac{\sum \omega_i \, s_i}{\sum \omega_i} \tag{2}$$

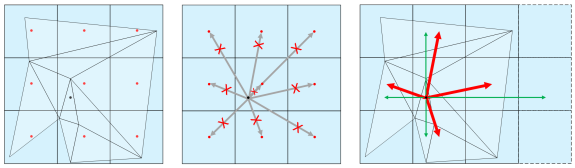

Figure 5: *(a) Isolated sub-pixel with high-frequency geometries. (b) Unavailable coverage information. (c) Illustration of morphological estimation. The four estimated lengthes of the edges (green arrows) determine the blending weights with neighboring pixels (red arrows).*

Morphological estimation for isolated sub-pixels: Based on the proposed filter above, we must find at least one coverage triangle, equivalently one or more corresponding shading samples, for each sub-pixel. However, if there is not any available triangle information for the sub-pixel, MLAA estimation strategy makes a good complement to deal with those isolated sub-pixels (see Figure 5 (a) and (b)). This issue is common for sub-pixel anti-aliasing algorithms, which has not been specified in related post-processing anti-aliasing studies.

The morphological anti-aliasing introduced by Reshetov [Res09] has been extended for GPU implementation [JME*11, JESG12]. The main concept is to detect the edge and compute the edge length in the horizontal and vertical directions using the color or depth information, and then blending the 4 neighboring color according to the edge length. TGAA inherits this strategy and only makes two modifications:

1) The primitive ID is employed to detect the length of geometric edge rather than color or depth. The shader increases the edge length by one pixel-size for each iteration, until the primitive IDs of the two subsequent pixels are different;

2) TGAA considers the sub-pixel position as the start point in edge length evaluation, which achieves more accurate length of the edge in four directions. The sub-pixel position should be consider while evaluating the length of the edge.

This additional step increases the overhead of the filter only sightly for two reasons. First, isolated sub-pixels occupy a very small proportion in the entire screen. In our testing scenarios, less than 0.1% of sub-pixels are isolated. Second, the computational cost of morphological anti-aliasing is mainly influenced by the number of iterations in the edge length computation step. This step typically requires 1∼5 iterations near the high-frequency geometries.

4. Implementation Results and Discussion

We have implemented the proposed anti-aliasing algorithm with DirectX 10.1 and HLSL shader mode 4.0 on a PC with Intel Core i5 760@2.8GHz, 16 GB of physical memory and NVIDIA GeForce GTX 780 GPU, 4 GB memory. The Windows 7 operation system was used. The testing screen resolutions varies from 1280×720 to 2560×1440, adapting from the mobile device to the PC display. The implementation results indicate that TGAA can generate high-quality anti-aliasing images, which approximate the reference SSAA images (Figure 6). In this paper, SRAA [CML11] and SMAA [JESG12] are selected as the representative geometry-assisted and image-based post-processing anti-aliasing algorithms, respectively, for detailed comparisons. Figure 7 shows a comparison of various algorithms and highlights two typical features. The generated images in this paper are provided without PDF compression in the supplemental files.

4.1. Comparison with SRAA

G-buffer size: One of the main advantages of TGAA is the small size G-buffer and low memory requirement. In our experiments, both TGAA and SRAA were implemented using full-screen geometry storage, with no storage compression techniques. The detailed G-buffer format and statistics of the G-buffer data size of the two algorithms are provided in Table 1. At the testing sub-pixel sampling rates of $4\times$ and $16\times$, the G-buffer size of TGAA is only 44.3% and 22.1% of SRAA, respectively, because the supersampling depth and normal G-buffer in SRAA are expensive to store and access. Remarkably, even the memory requirement of $16\times$ TGAA is lower than $4\times$ SRAA, which demonstrates that TGAA is more scalable with an increasing sub-pixel sampling rate.

Quality: TGAA is designed with a similar framework of SRAA; triangle-based geometry is used in TGAA for a more faithful anti-aliasing filtering. Figure 8 presents a typical example and compares SRAA and TGAA. The marked horizontal edge is located near the floor and the shortest building in the scene, and there is no clear variation in depth and normal for these pixels. In this case, SRAA's filter degrades to an unfaithful blur and introduces erroneous color in the filtering. TGAA consider only those reasonable shading sam-

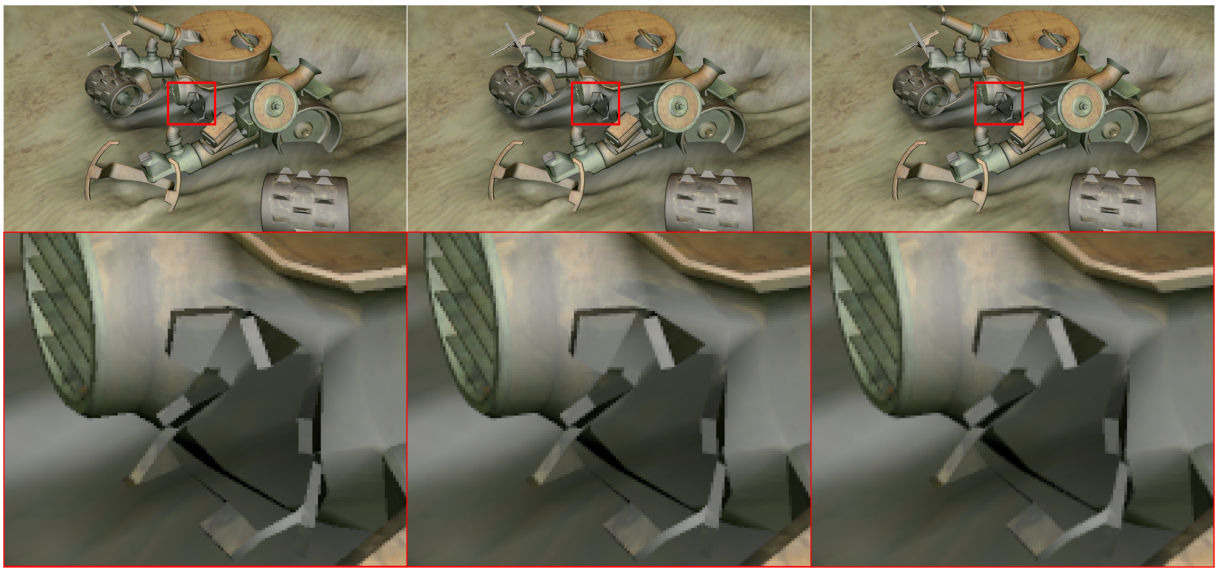

Figure 6: *Comparison of 4 × TGAA, 16 × TGAA and reference 64 × SSAA images in Tank (214K triangles).*

Table 1: *Comparison of the G-buffer data size (Mbytes) of SRAA and TGAA with various screen resolutions and sampling rates. The detailed G-buffer format of both algorithms are listed below (n represents the number of sub-pixels in each pixel).*

screen resolution	G-buffer size			
	TGAA		SRAA	
	4×	16×	4×	16×
1280 × 720	13.6	25.5	30.8	115.3
1600 × 900	21.3	39.9	48.1	180.0
1920 × 1080	30.7	57.4	69.3	259.4
2560 × 1440	54.5	102.1	123.2	461.1

TGAA: 24 bits color, $n\times$ 9 bits coverage, 32 bits primitive ID, and 32 bits depth
SRAA: 24 bits color, $n\times$ 32 bits depth and $n\times$ 32 bits normal

ples and avoids this artifact due to the triangle-based information.

We also measure the anti-aliasing quality for both TGAA and SRAA by comparing their color errors at sub-pixel level. The SSAA image with the same screen resolution and same sub-pixel sampling rate is used as the ground truth image. To quantitatively describe the sub-pixel color error of a given algorithm, we computed the difference between the corresponding sub-pixel color in the test image and the SSAA image. Then we added the 3 RGB channels of the color difference value to the measurement λ. The statistics are described in Table 2. The color error of the sub-pixels is distributed in different ranges. The statistics demonstrate that TGAA reconstructs the anti-aliasing color more faithfully than SRAA.

Performance: TGAA achieves real-time performance due to the simplicity of the algorithm framework and the low G-buffer access overhead. Table 3 and 4 detail the analysis of each step of the algorithm and the comparison between SRAA and TGAA under relatively low and high screen resolution.

These experimental results demonstrates that TGAA has little advantage of the performance with both 1280 × 720 and 2560 × 1440 resolution. In the G-buffer storage step, the main overhead of TGAA is the geometric processing in the geometry shader stage of the rendering pipeline, whereas high resolution rasterization dominates the performance of SRAA. TGAA display a slight advantage over the SRAA's two-pass rendering due to the simplicity of generating screen resolution G-buffers in a single pass. In the anti-aliasing filtering step, SRAA must access a relatively large G-buffer, but its bilateral filtering is straightforward and simple. However, the computational cost of selecting reasonable shading samples is higher in the filtering pass of TGAA, which leads to slightly lower performance on this pass compared to that of SRAA.

4.2. Comparison with SMAA

TGAA and the representative image-based anti-aliasing algorithm SMAA were also compared. Morphological methods tend to generate unfaithful anti-aliasing results near

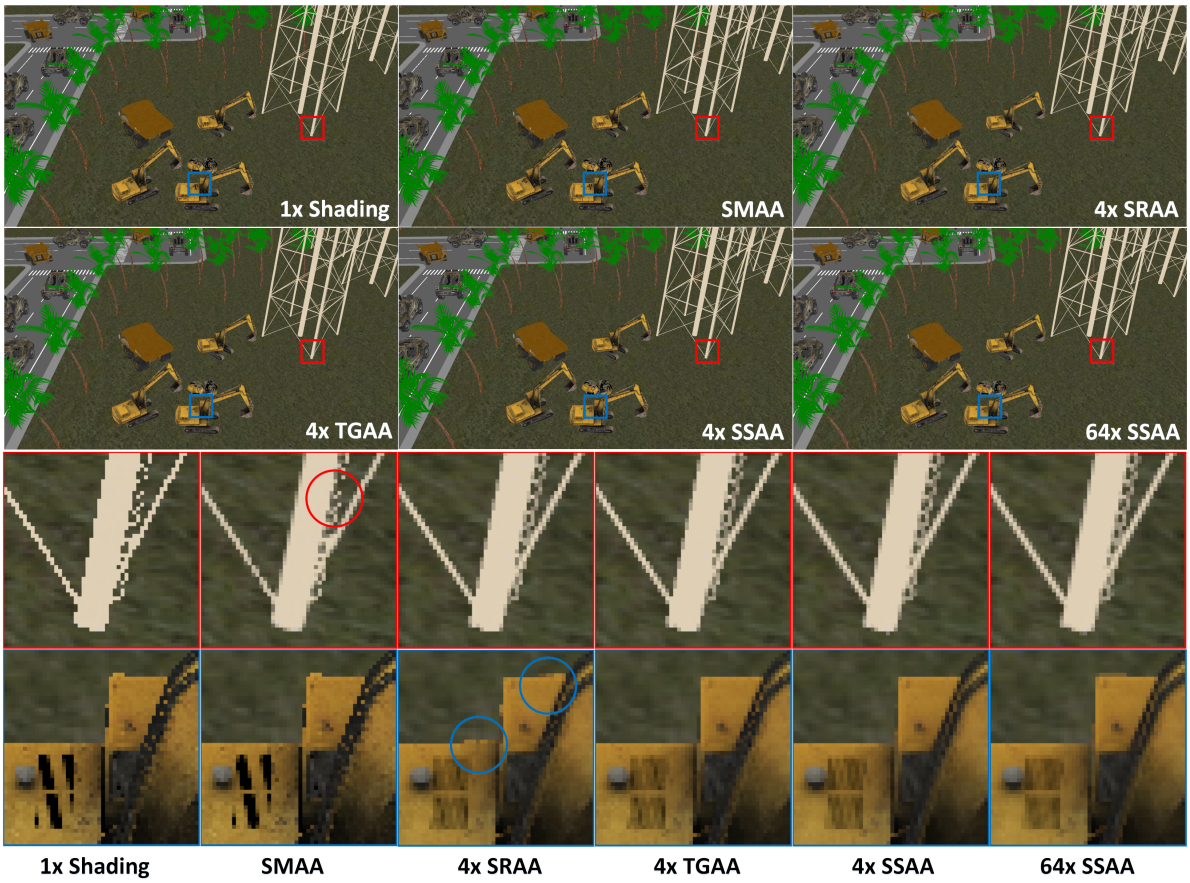

Figure 7: *Comparison of various algorithms in Traffic (958K triangles), 1280 × 720. Details in red square: SMAA blurs the 1 × shading image with a heuristic approach, whereas SRAA and TGAA can reconstruct sub-pixel-level edges and generate anti-aliasing images similar to the reference SSAA. Details in blue square: SRAA introduces unfaithful shading information to sub-pixel color filter in such region of uniform geometry.*

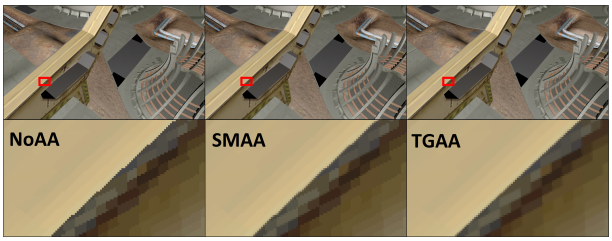

Figure 9: *Comparison of SMAA and TGAA in Powerplant (470K triangles).*

some geometric edges due to the heuristic edge estimation and the lack of the sub-pixel level geometry. As shown in Figure 9, TGAA provides smooth geometric edges in the anti-aliasing image, wheres SMAA produces segmented edge according to the aliased image. However, SMAA is more efficient than TGAA (see Table 5).

Table 2: *Percentage of sub-pixel in different intervals of the color error for 4 × SRAA and TGAA. The testing resolution is* 1280 × 720.

		$\lambda \leq 0.0001$	$\lambda \leq 0.001$	$\lambda \leq 0.01$
Building	SRAA	98.91%	99.10%	99.33%
	TGAA	99.92%	99.96%	99.98%
Tank	SRAA	96.51%	97.40%	98.79%
	TGAA	96.67%	98.22%	99.42%
Powerplant	SRAA	95.88%	97.25%	98.80%
	TGAA	95.98%	98.34%	99.32%
Traffic	SRAA	96.32%	97.38%	98.71%
	TGAA	96.61%	97.98%	99.25%

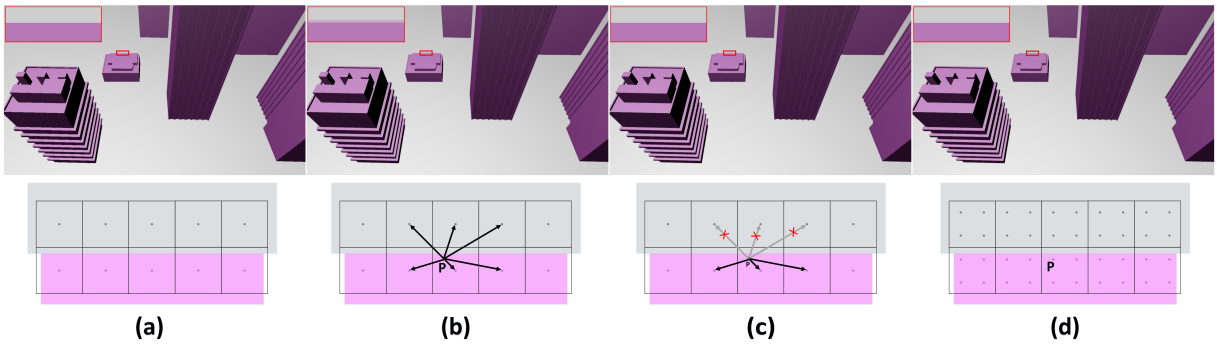

Figure 8: *Comparison of (a) 1 × shading, (b) 4 × SRAA, (c) 4 × TGAA and (d) reference 4 × SSAA images in Building (9K triangles).*

Table 3: *Runtime (milliseconds) comparison of SRAA and TGAA with* 1280 × 720 *screen resolution.*

Scene (#triangles)		shading generation	geo	AA filtering	total
Building (9, 000)	SRAA	0.5	1.2	1.6	3.3
	TGAA	1.2		1.9	3.1
Tank (214, 000)	SRAA	0.6	1.4	1.7	3.7
	TGAA	1.7		1.9	3.6
Powerplant (470, 000)	SRAA	0.6	1.5	1.7	3.8
	TGAA	1.7		1.9	3.8
Traffic (958, 000)	SRAA	0.6	1.8	2.0	4.4
	TGAA	2.2		2.0	4.2

Table 4: *Runtimes (milliseconds) comparison of SRAA and TGAA with* 2560 × 1440 *screen resolution.*

Scene (#triangles)		shading generation	geo	AA filtering	total
Building (9, 000)	SRAA	1.3	3.0	5.1	9.4
	TGAA	3.0		5.5	8.5
Tank (214, 000)	SRAA	1.5	3.2	5.4	10.1
	TGAA	3.6		5.7	9.3
Powerplant (470, 000)	SRAA	1.6	3.6	5.5	10.7
	TGAA	3.7		5.8	9.5
Traffic (958, 000)	SRAA	1.9	4.3	5.5	11.7
	TGAA	4.1		6.4	10.5

4.3. Limitation

The main limitation of TGAA is that some sub-pixels would fail in TGAA's filter, and then degenerate to MLAA and inherit MLAA's drawbacks. This fact are mainly lead by two reasons: 1) TGAA is based on the coverage bitmask information, which is a partial representation of the scene geometry. The loss of triangle geometry will lead fail cases. 2) TGAA's filter is strictly within the actual coverage triangle area to achieve faithful filtering. With this strategy, isolated sub-pixels have to adopt MLAA as a backup strategy.

The effect of this limitation can be observed, especially in scenes with pixel-sized triangles or features. Figure 10 demonstrates a typical Foliage scene with highly-tesselated triangles and pixel-sized geometric features. While generating smooth anti-aliasing quality for most geometric edges (red rectangle), some pixels are partly degenerate to morphological anti-aliasing result. It is hard to completely reconstruct those pixel-size or sub-pixel-size holes like the reference MSAA (blue rectangle).

Table 5: *Runtimes (milliseconds) comparison of SMAA and TGAA.*

	SMAA 1280 × 720	TGAA 1280 × 720	SMAA 2560 × 1440	TGAA 2560 × 1440
Building	2.5	3.1	7.0	8.5
Tank	2.7	3.6	7.4	9.3
Powerplant	2.7	3.8	7.9	9.5
Traffic	3.1	4.2	8.8	10.5

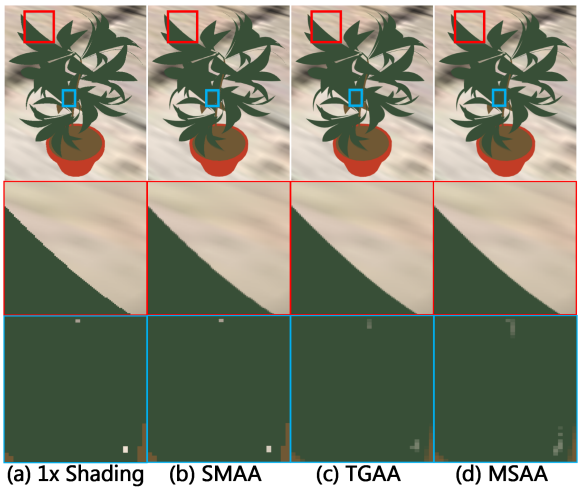

Figure 10: *Illustration of the limitation of TGAA in Foliage scene with highly-tesselated triangles and pixel-size features.*

Flickering can be expected in animated scene since two different anti-aliasing filters work together in TGAA. To totally address this problem, combining TGAA with a reliable temporal filter, e.g., temporal reprojection, could be considered as one solution. However, it is a new challenge to design such an anti-aliasing framework efficiently.

5. Conclusion and Future Work

We have presented a new post-processing geometry-assisted anti-aliasing algorithm termed TGAA. Using original triangle-based geometric information, TGAA can achieve real-time sub-pixel-level anti-aliasing. TGAA displays low memory requirements and faithful anti-aliasing quality compared to the state-of-the-art post-processing anti-aliasing algorithms.

As a post-processing geometry-assisted anti-aliasing algorithm, the proposed TGAA does not consider texture edges or shading edges. Image-based anti-aliasing approaches are more suitable for these artifacts. Our future plan is to design a hybrid approach that can process geometric aliasing and shading (or texture) aliasing robustly.

6. Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments. This work is supported by the National Natural Science Foundation of China under Grant Nos. 61170138 and 61472349.

References

[Ake93] AKELEY K.: Reality engine graphics. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), SIGGRAPH '93, pp. 109–116. 1

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

[And11] ANDREEV D.: Directionally localized anti-aliasing. In *Game Developers Conference* (2011), pp. 1–55. 3

[BAM13] BARRINGER R., AKENINE-MÖLLER T.: A4: Asynchronous adaptive anti-aliasing using shared memory. *ACM Trans. Graph. 32*, 4 (July 2013), 100:1–100:10. 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. 2, 3, 6

[DYF08] DAI Q., YANG B., FENG J.: Reconstructable geometry shadow maps. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (2008), pp. 4:1–4:1. 2

[GPB04] GELDREICH R., PRITCHARD M., BROOKS J.: Deferred lighting and shading. In *Game Developers Conference* (2004), pp. 1–21. 1

[Har04] HARGREAVES S.: Deferred shading. In *Game Developers Conference* (2004), pp. 1–32. 1

[HBE13] HOLLÄNDER M., BOUBEKEUR T., EISEMANN E.: Adaptive supersampling for deferred anti-aliasing. *Journal of Computer Graphics Techniques 2*, 1 (2013), 1–14. 2

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

[JESG12] JIMENEZ J., ECHEVARRIA J. I., SOUSA T., GUTIERREZ D.: SMAA: Enhanced subpixel morphological antialiasing. *Comp. Graph. Forum 31*, 2 (May 2012), 355–364. 3, 6

[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 anti-aliasing. In *ACM SIGGRAPH 2011 Course Notes* (2011), pp. 6:1–6:329. 3

[JME*11] JIMENEZ J., MASIA B., ECHEVARRIA J. I., NAVARRO F., GUTIERREZ D.: *Practical Morphological Anti-Aliasing*. GPU Pro 2. AK Peters Ltd., 2011, pp. 95–114. 3, 6

[Koo07] KOONCE R.: *Deferred Shading in Tabula Rasa*. GPU Gems 3. Addison Wesley, 2007, pp. 429–458. 1

[Lau10] LAURITZEN A.: *Deferred Rendering for Current and Future Rendering Pipelines*. Tech. rep., 2010. Beyond Programmable Shading course, SIGGRAPH 2010. 2

[LMSG14] LECOCQ P., MARVIE J.-E., SOURIMANT G., GAUTRON P.: Sub-pixel shadow mapping. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 2014* (2014), pp. 103–110. 2

[Lot11] LOTTES T.: *FXAA*. Tech. rep., NVIDIA, 2011. 3

[Per11a] PERSSON E.: *Geometric post-process anti-aliasing*. 2011. 3

[Per11b] PERSSON E.: *Geometry buffer anti-aliasing*. 2011. 3

[Pin88] PINEDA J.: A parallel algorithm for polygon rasterization. *SIGGRAPH Comput. Graph. 22*, 4 (June 1988), 17–20. 4

[Res09] RESHETOV A.: Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009* (2009), pp. 109–116. 3, 6

[Res12] RESHETOV A.: Reducing aliasing artifacts through resampling. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics conference on High-Performance Graphics* (2012), pp. 77–86. 3

[Shi05] SHISHKOVTSOV O.: *Deferred Shading in S.T.A.L.K.E.R.* GPU Gems 2. Addison Wesley, 2005, pp. 143–166. 1

[YN06] YOUNG P., NVIDIA: *Coverage Sampled Antialiasing*. Tech. rep., 2006. 2