Vision, Modeling, and Visualization (2013)
Michael Bronstein, Jean Favre, and Kai Hormann (Eds.)

Non-Sampled Anti-Aliasing

Thomas Auzinger, Przemyslaw Musialski, Reinhold Preiner, and Michael Wimmer

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Austria

Abstract

In this paper we present a parallel method for high-quality edge anti-aliasing in rasterization. In contrast to traditional graphics hardware methods, which rely on massive oversampling to combat aliasing issues, we evaluate a closed-form solution of the associated prefilter convolution. This enables the use of a wide range of filter functions with arbitrary kernel sizes, as well as general shading methods such as texture mapping or complex illumination models. Due to the use of analytic solutions, our results are exact in the mathematical sense and provide objective ground-truth for other anti-aliasing methods and enable the rigorous comparison of different models and filters. An efficient implementation on general purpose graphics hardware is discussed and several comparisons to existing techniques and of various filter functions are given.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism —Hidden line/surface removal I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism —Visible line/surface algorithms

1. Introduction

The common display and storage format of digital imagery is a discrete grid of color samples, i.e. a raster image. While being tremendously useful, it suffers from an inherent limitation: when converting an arbitrary continuous signal into this format, signal degradation occurs as the finite number of samples cannot store the full information content of the signal. This is known as *undersampling* and gives rise to a multitude of artifacts. Not only is part of the signal lost during the sampling process, but also some of the high-frequency details are introduced as low-frequency *aliases* of the data. This effect is known as *aliasing* and constitutes a major field of research in digital signal processing since day one.

In computer graphics, aliasing has many forms and can be classified into two main categories: spatial and temporal. The former is a general concern in image generation which we address in our work. The latter arises for temporal image sequences. In the process of generating raster images, there are, roughly spoken, two major sources of aliasing: (1) undersampling of the interaction of light with surface materials and (2) undersampling of the visibility of the object geometry. This is a consequence of the fact that the scene data is discretized by the rasterizer before shading or visibility computations are executed. Artifacts caused by (1) are, for example, low-frequency patterns in distant textures or missing highlights. (2) arises under various circumstances, e.g., along edges at silhouettes or creases of objects, very small objects that disappear between the samples, or complicated detail where features are lost [Cro77]. This is commonly referred to as *edge aliasing* and is an active research area in computer graphics since decades.

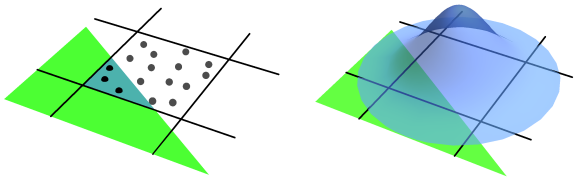

Figure 1: Left: Multisampling methods supersample the visibility function of the scene primitives and determine the contribution to the output pixel by the ratio of primitive hits. Right: An optimal edge anti-aliasing method would compute this contribution with an exact integral of a filter kernel with the visible area of the primitive.

Procedures that combat aliasing are called *anti-aliasing* and there are two general ways to do so: *supersampling* and

DOI: 10.2312/PE.VMV.VMV13.169-176

prefiltering. The first method evaluates the source signal at a larger amount of samples and then averages them down to the desired sample count (cf. Figure 1, left). This is also called post-anti-aliasing as the downfiltering happens after the sampling. This method has the conceptual limitation that it just increases the highest representable signal frequency, i.e. the Nyquist frequency, but is inadequate for highly frequent signals such as binary visibility. In practice, however, it often works well enough and is easily combinable with rasterization, thus widely used in various kinds of rendering settings and hardware architectures. Prefiltering goes the opposite way and eliminates the problematic frequencies before the sampling step. This is performed by convolving the source signal with a low-pass filter (cf. Figure 1, right). This method guarantees valid output and is conceptually correct in contrast to supersampling. Due to the complexity of the filter convolution, it is hardly used in rasterization.

As mentioned before, the two main sources of aliasing in rasterization are shading and visibility and the different anti-aliasing methods can be applied separately to both channels. In rasterization, *supersampling* usually refers to identical supersampling of both, while *multisampling* supersamples only the visibility. We present a different approach by employing prefiltering on the visibility channel (cf. Figure 1, right). In contrast to depth-buffering methods, we convolve the visibility signal of the projected scene with a suitable filter. This eliminates the undesired frequencies robustly from the source and produces near-perfect edge anti-aliasing. As all analytic approaches, this requires a closed-form solution to the prefiltering operation. In our case, we are able to obtain such a solution due to the mathematical simplicity of the visibility signal, which can be represented by a piecewise constant function.

While our design deviates significantly from traditional raster-based pipelines, it is still embarrassingly parallel and we present an implementation on general-purpose graphics hardware. As the main contribution of our work, we present a prototypic renderer, which generates raster images of scenes with general surface shading, such as textures and non-linear illumination models, that are exactly anti-aliased up to numerical precision of the graphics hardware. The output images can be considered as non-sampled anti-aliased (NSAA), since neither the filter nor the visibility is defined discreetly but completely analytically. As a result, our method provides a ground truth solution to edge anti-aliasing. It can serve as an objective reference for quantitative comparisons of various anti-aliasing approaches and can be used to evaluate the performance of different filters.

The paper is structured as follows: after a review of related work in Section 2 we describe or framework in Section 3 and present a discussion and results in Section 4. We conclude the work in Section 5.

2. Related Work

Before massive sampling became technologically viable with the z-buffer methodology [Cat74], a considerable body of work was created which deals with analytic visibility determination. Refer to Sutherland et al. [SSS73] for a survey of early methods for hidden-surface removal. Extensions to analytic hidden surface elimination were introduced by Weiler and Atherton [WA77], who clip polygonal regions against each other until trivial depth sorting is obtained, and by Franklin [Fra80], who uses a tiling and blocking faces. A number of further early works focused on parallelization and performance improvement of the analytic approach [CJ81, McK87]. Other improvements were made by Mulmuley [Mul89] and Sharir et al. [SO92] in order to base the run-time complexity on the actual number of intersections between the polygons.

The methods for visibility determination were developed hand-in-hand with filtering methods for anti-aliasing. Crow [Cro77] posed the problem in his seminal paper and a pre-filtering solution to it. Catmull was also aware of the importance of anti-aliasing and employed it very early [Cat78, Cat84]. An analytic algorithm that supports polynomial approximations of general filters and linear color functions was given by Duff [Duf89]. Further works developed a justified filtering methodology [Tur86, Mit87, MN88] and (semi) analytic solutions [McC95, GT96] for the reconstruction problem.

With the success of commodity graphics hardware, the research shifted to approximate sampling-based solutions. Integrated into the graphics pipeline, different strategies were developed to balance quality and performance requirement. Multisample anti-aliasing (MSAA) [AMHH08] performs supersampling of the projected scene visibility. For (partially) visible primitives, the shading is only executed once per pixel and a final blending combines the shading values of the supersamples. An optimization was introduced with coverage sampling anti-aliasing (CSAA) [You06]. A technique to supersample both visibility and shading independently of each other was introduced with *decoupled sampling* [RKLC*11] which is aimed at complex effects such as depth-of-field and motion blur.

Nonetheless, these methods do not mesh well with the nowadays popular deferred shading techniques [DWS*88], and, in recent years, screen-space post-processing approaches have been explored very intensively. They are based on edge-aware smoothing of the framebuffer and one prominent example is *morphological* anti-aliasing (MLAA) [Res09]. While originally introduced a decade ago, a revival of this field has produced many variants and implementations on graphics hardware [JGY*11, CML11, Lot11, JESG12].

Prefiltering-based anti-aliasing gained traction in the last few years. Manson and Schaefer [MS11] introduced wavelet rasterization that can rasterize polygons and parametric

curves, as well as a method for analytic rasterizing of 2D-shapes that have curved boundaries and linear color gradients [MS13]. Another set of works, presented by Auzinger et al., provides graphics hardware implementations for analytic anti-aliasing of linear shading [AGJ12] and exact hidden surface elimination [AWJ13].

These last methods are the most similar to our work but have a major limitation. Fully analytic rasterization requires an integrable shading function defined on the projected surface to obtain a closed-form solution. So far, this was only shown for linear functions and can be extended to polynomials. Common shading functions are non-linear due to perspective interpolation, normal vector normalization or trigonometric functions. In this work we sidestep this problem by analytically anti-aliasing only the visibility signal and thus allowing general shading.

3. Non-Sampled Anti-Aliasing

Our aim is to produce a perfectly edge anti-aliased raster image by analytically solving the convolution

$$w_P = \int \chi_P(x) W(s-x)\, dx \qquad (1)$$

of the *visible* projected area of a primitive P (given by its characteristic function χ_P) with a prefilter W at a given sample location s (usually a pixel center). The weight w_P is then used to blend the shading samples at s. In terms mulitsampling, this equates to taking an infinite amount of visibility samples x_i around s, and using their weighted sum $\sum_i W(x_i - s)$ for the final blending. Common multisampling techniques restrict their sample positions to a rectangular pixel region, whereas our method applies a radially symmetric prefilter of arbitrary extent, thereby allowing for unbiased reconstruction of the image signal.

Our proposed analytic pipeline consists of three main stages, depicted in Figure 2: *(1) Primitive gathering:* first, the relevant primitives that have to be considered for the prefiltering at each sample location are listed. These are all primitives that intersect the support of the filter kernel when placed at the individual pixel centers. *(2) Weight computation:* given the list of relevant primitives per sample, their contribution to the final sample color is computed by analytically evaluating the convolution given by equation 1. These weights are used in a final *(3) Blending* stage, where the primitives are rasterized and the fragments are blended with weights of the previous stage.

Our framework was developed with current graphics architecture in mind and efficiently utilizes the rasterization pipeline as well as the GPGPU capabilities of modern GPUs. Conceptually, our framework can accommodate arbitrary 2D primitives and every regular spacing of sample locations. In our implementation, however, we assume triangles as input and pixel centers as our sample locations. The individual

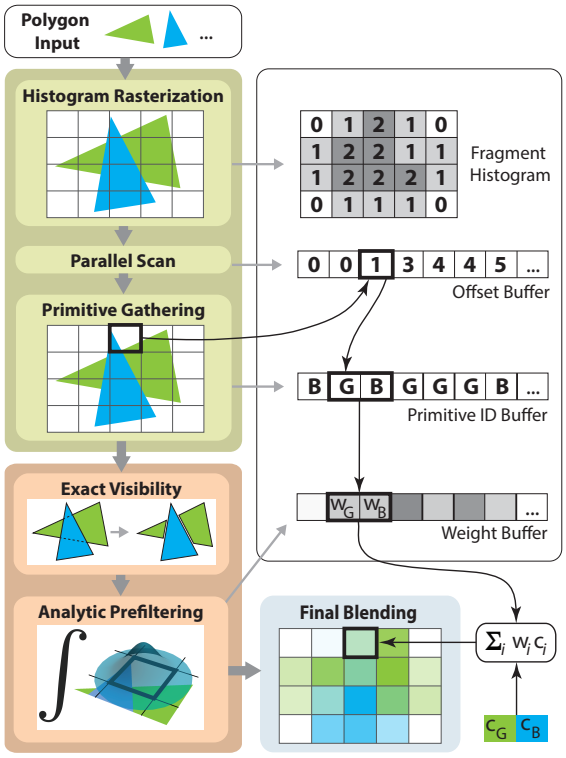

Figure 2: Overview of our NSAA pipeline and its three main stages: primitive gathering (green, Section 3.1), weight computation (red, Section 3.2), and final blending (blue, Section 3.3). Intermediate buffers are shown in gray.

stages of our pipeline, as shown in Figure 2, will be described in detail in the following sections.

3.1. Primitive Gathering

Convolution can be imagined as placing a prefilter at each sample location and evaluating the corresponding integral with each primitive (cf. Eq. 1). Mathematically, such a filter function should have infinite support to provide the best possible low pass performance and it would rapidly vanish with increasing distance from its center. As numerical precision is limited and minor changes in a filter are not visually perceivable, it is safe to place a cutoff at a certain distance from the center. In this way the influence region of the filter kernel around each sample location is limited and only close primitives are relevant. This considerably reduces the workload for large scenes or large output images as otherwise the number of possible pairings would grow with the product of primitive and sample count.

Given a set of input primitives, the first task of our algorithm is to list all the primitives that intersect the filter kernel of a given sample. As the number of intersecting primitives

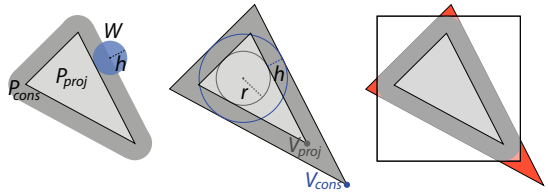

Figure 3: (a) For a given projected primitive P_{proj} we need to rasterize the conservative area P_{cons} in order to address all placements of the prefilter kernel where it intersects P_{proj}. (b) A triangle is conservatively enlarged by scaling it around its incenter according to its incircle radius r and the kernel radius h. (c) During rasterization, all dark red locations outside the minimum conservative box are discarded to reduce the number primitive IDs whose convolution evaluates to zero as they lie outside of P_{cons}. Note that the light red region still report superfluous IDs but are small compared to the triangle areas.

per sample can vary strongly depending on the scene, it is beneficial to store the per sample primitive lists in a linear, tightly packed buffer (cf. Figure 2). This *primitive ID buffer* is computed in two steps:

1. **List Offset Computation:** First, the required list sizes for each sample are computed. This is easily achieved by an initial scene rasterization pass, that accumulates a histogram of intersecting primitives using additive blending. In a following GPGPU pass, a *parallel scan* [HB10] over the linearized histogram buffer is performed to obtain a *list offset buffer*, which contains the index of each samples' first list element in the primitive ID buffer.
2. **Filling the Primitive ID Lists:** In a second scene rasterization pass, the list offsets are used to scatter the primitive ID of each primitive to the corresponding positions in the primitive ID buffer. As the histogram count above, this can be performed conflict-free for concurrently writing threads by employing atomic counters. For this task, we use an additional count buffer of the same length as the offset buffer. The buffer index of each primitive ID is then given as the sum of the offset and the count. The resulting primitive ID buffer is then transfered to the next stage for analytic visibility and weight computation.

Note that both rasterization passes are performed without shading or depth buffering. All primitive IDs are collected and visibility is computed at the next stage.

Conservative Rasterization. Due to the finite extent of our radial prefilters, primitives can be relevant for a given sample location without overlapping it, but intersecting only the prefilter support. We therefore compute a *conservative* coverage of all relevant sample locations when rasterizing a primitive during the histogram, primitive gathering, and the final blending stage. Similar to Hasselgren et al. [HAMO05], this is achieved by an image-space thickening of the primitives according to the kernel radius (cf. Figure 3a): in each rasterization pass, we make use of the geometry shader stage to first obtain the projection P_{proj} of an input triangle P, and then compute the required vertex offsets that produce a conservative triangle P_{cons}. Offsetting an image space vertex V_{proj} to its new conservative position V_{cons} basically equates to scaling its corresponding triangle around the center of its incircle by a factor of $\frac{r+h}{r}$, where r is the radius of the incircle, and h denotes the support radius of the filter kernel (cf. Figure 3b). The resulting enlarged primitive P_{cons} is then rasterized, and all its conservative fragments are processed accordingly. To reduce the number of unnecessarily evaluated fragments, which can become significant for acute triangles, we discard any fragments outside a conservative triangle clipping box in the pixel shader (cf. Figure 3c).

3.2. Analytic Weight Computation

In this stage we compute the exact contribution of all primitives to the given sample locations. As shown in Figure 2, it consists of two main steps. The analytic visibility step performs hidden surface elimination and outputs the visible parts of all primitives for the given view direction. This constitutes the exact visibility signal on which we apply prefiltering. Note that in traditional depth-buffering the visibility signal is (super)sampled directly, whereas we rely on a closed-form solution of the filter convolution integral to avoid any undersampling issues. Assuming triangular primitives, the visible regions are polygons and the convolution can be decomposed into a sum of integrals over the linear boundary segments of each polygon. For both the visibility and the prefiltering step we employ previous works and a summary of both is provided in the next paragraphs.

The hidden surface elimination is performed with the help of the analytic visibility method of Auzinger et al. [AWJ13]. It is a parallel algorithm that computes the boundaries of all visible regions of the scene primitives. This is achieved by taking the primitive geometry as input and determining the edge intersection between projected triangles. After a sorting of the intersections along their respective edges, occlusions are detected with the help of prefix sum computations. Similar to various scanline methods, this method propagates visibility information along the edges by counting both the start- and endpoints of occlusions and reporting the visible parts of the primitives. We also inherit this method's limitations and require non-intersecting and consistently orientated triangles as input to our implementation. The output of this step is a set of line segments which constitute the boundaries of all visible regions of all scene primitives. Note that this step does not depend on the output of the primitive gathering section of our pipeline and can run concurrently with it or on another device.

In the following prefiltering step, we exploit the geometric simplicity of the polygonal visible regions when evaluating integrals over them. Previous works on analytic filter convolution, such as the method of Manson and Schaefer [MS13] and Auzinger et al. [AGJ12], decompose the convolution over each visible region (cf. Eq. 1) into a sum of integrals over its boundary. Furthermore, they support linear shading functions which also contains our case of the constant characteristic function χ_P. Each summand of the convolution decomposition can be expressed in closed form and evaluated with the numerical precision of the hardware.

In contrast to these previous works, we do not employ full analytic shading as advanced shading effects cannot be solved in closed form. We use the two methods of Auzinger et al. [AGJ12, AWJ13] just for visibility prefiltering. This can be seen as a substitution for the depth-buffering in the traditional 3D graphics pipeline. Hidden surface elimination is performed once per frame and yields the aforementioned set of boundary line segments that is used as input to the prefiltering step together with the primitive ID buffer. As each buffer entry *id* is associated with a sample location s, we store the result of the filter convolution $\int \chi_{id}(x)W(s-x)\,dx$ at the corresponding entry of the weight buffer. This can be executed in parallel for all entries of the primitive ID buffer and allows efficient utilization of massively parallel hardware architectures. The filled weight buffer constitute the output of this step.

3.3. Final Blending

For each sample location s and all its corresponding primitives P_i, the previous stage outputs an analytically computed weight w_i that encodes the primitive's relative contribution to the final color of the sample (cf. Figure 2). We are left to compute a *shading value* c_i for each primitive. In traditional rasterization, shading is performed directly at the sample location (usually the pixel center or the centroid of contributing subsamples). Due to the positive extent of our prefilter, it cannot be guaranteed that the projected primitive overlaps the sample location. In this case we project the sample location onto the primitive, i.e. we take the point on the primitive with minimal distance to the sample location as our shading location.

By issuing another rasterization pass, we can now compute the shading value c_i at this point, for example via texturing or non-linear illumination and interpolation models. As each sample location gets contributions from various primitives, we employ conventional blend operations to aggregate the final sample color. Using a normalized prefilter kernel ensures that $\sum_i w_i \leq 1$ for all weights w_i associated with a sample location. $1-\sum_i w_i$ gives the weight of the background and is non-zero if the support of the local prefilter is not completely covered by the projected scene primitives. Thus, the final color of a sample is given by $\sum_i w_i c_i + (1-\sum_i w_i)c_B$ where c_B denotes the background color. Note that no depth buffering is used in this final rasterization step and that the scene visibility was already resolved by the weight computation stage which assigns zero weight to occluded primitives.

In this work we use a single shading value per primitive and sampling location. Supersampling of the shading channel is a promising future extensions of our framework for which the optimal sample positioning poses the biggest challenge. Current multisampling approaches place the shading sample inside the projected area of the primitive whereas the correct solution would require a placement inside the *visible* projected area. Otherwise it is possible to obtain shading values from hidden parts of the scene. For general triangular scenes, the visible regions of a single input primitive can already consist of several non-convex polygons and an efficient sampling of these regions would deliver the optimal anti-aliasing quality for rasterization. This would require fundamental changes to our pipeline, however, as the shading sample locations do not coincide any longer with the rasterization grid of common graphics hardware.

4. Results and Applications

We developed a prototype implementation of our method using the graphics API DirectX 11 and the GPGPU framework CUDA. The rasterization steps in both the *primitive gathering* and *final blending* stage of our pipeline use traditional rasterization with deactivated depth buffering. The parallel scan and the *weight computation* stage rely on the flexibility and parallel performance of general-purpose graphics hardware. In the following sections we provide an evaluation of the anti-aliasing performance of our implementation.

4.1. Evaluation

We use well established test pattern to asses the quality of our method. The first pattern is a zoneplate, which features concentric rings with decreasing width to introduce increasing frequencies at the outer rings. The result of our method using a Gaussian filter with a radius of two pixels is illustrated in Figure 4 top right. The color pattern is applied with texture mapping. As one can easily see, the result is free of undersampling artifacts due to the exact prefiltering even at the demanding outer rim. Furthermore, the use of radial filters avoids the introduction of anisotropy artifacts along the image diagonal.

As a second test case we employ a binary log grid to investigate the anti-aliasing quality of straight lines. Using the same prefilter as above, we notice no perceivable artifacts in the result shown in Figure 5 top right.

A simple test scene in Figure 7 shows the correct occlusion handling and a non-linear shading model that has no closed-form solution. By restricting our analytic prefiltering to the visibility signal, sample-based shading models such as texture mapping are supported by our framework.

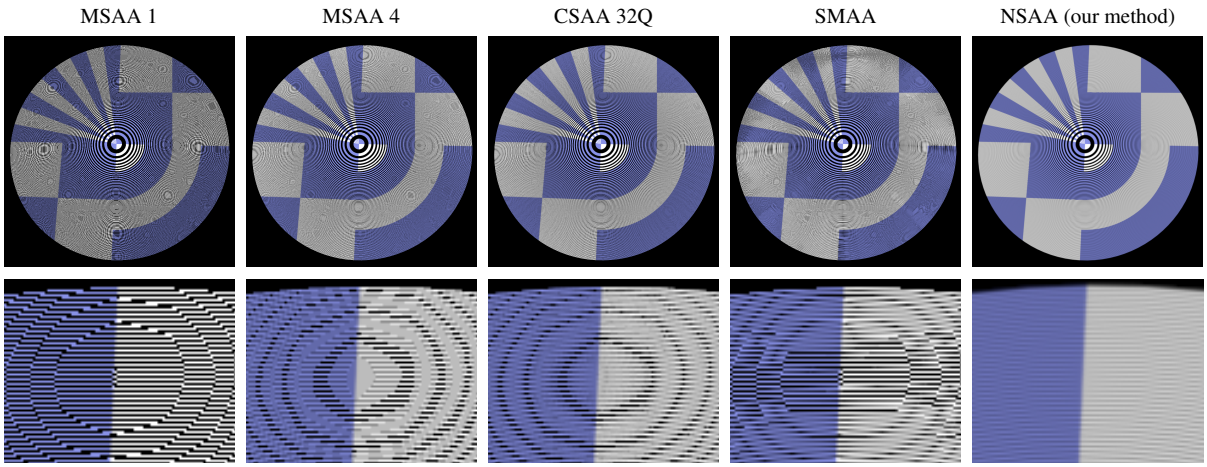

Figure 4: Comparison of common anti-aliasing techniques with our method (NSAA) using a Gaussian prefilter. The second row depicts a cutout of the top row zoneplate test pattern that was rasterized using the stated method.
Note that the performance of anti-aliasing methods is tied to the associated sampling characteristics. If the sampling is changed after applying anti-aliasing, artifacts in the form of excessive blurring or aliasing are introduced. We refer the reader to the electronic version of this paper to inspect the top row pattern close to their native resolution of 1024^2. The cutouts are added as a convenience for readers of the printed version.

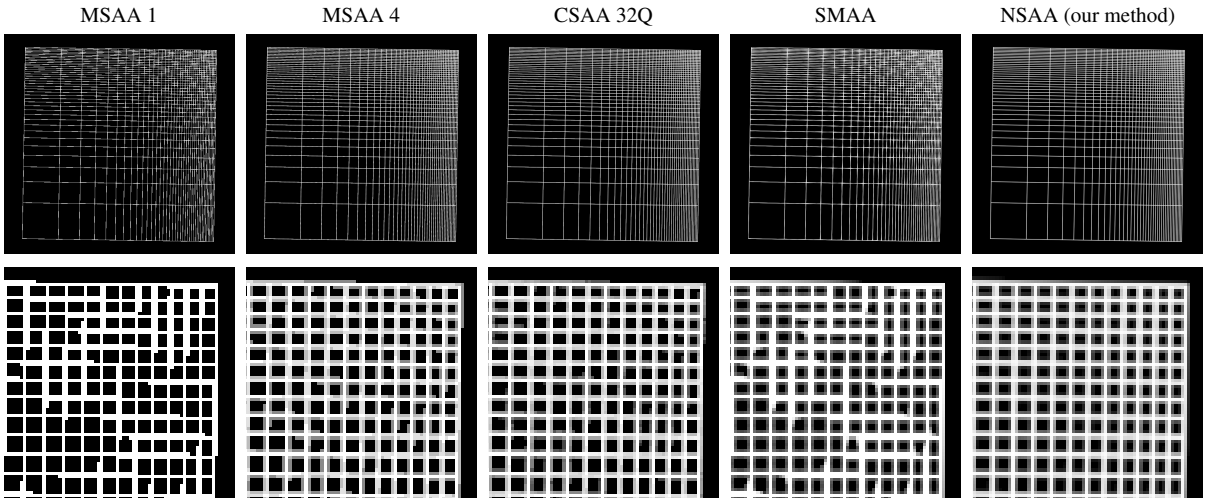

Figure 5: Comparison of common anti-aliasing techniques with our method (NSAA) using a Gaussian prefilter. The second row depicts a cutout of the top row log grid test pattern that was rasterized using the stated method. See Figure 4 for a further explanation of the two rows.

4.2. Ground Truth Generation

One possible application of our method is the objective comparison of approximative anti-aliasing methods. In the first and second row of Figures 4 and 5 we compare our output with widely used multi-sampling methods. MSAA1 to MSAA4 are sampling pattern that are specified by both the OpenGL and DirectX standards, while CSAA 32Q is the highest quality version of a NVidia specific variant. We also compare with SMAA, the post-processing method of Jimenez et al. [JESG12], which operates on the output image of a rendering system. As can be seen, all methods show various kinds of undersampling artifacts in the outer regions of the zoneplate or along the straight lines. As our method provides an analytically obtained reference, it can be considered as ground truth and any divergence from it can be quantified with a suitable error metric.

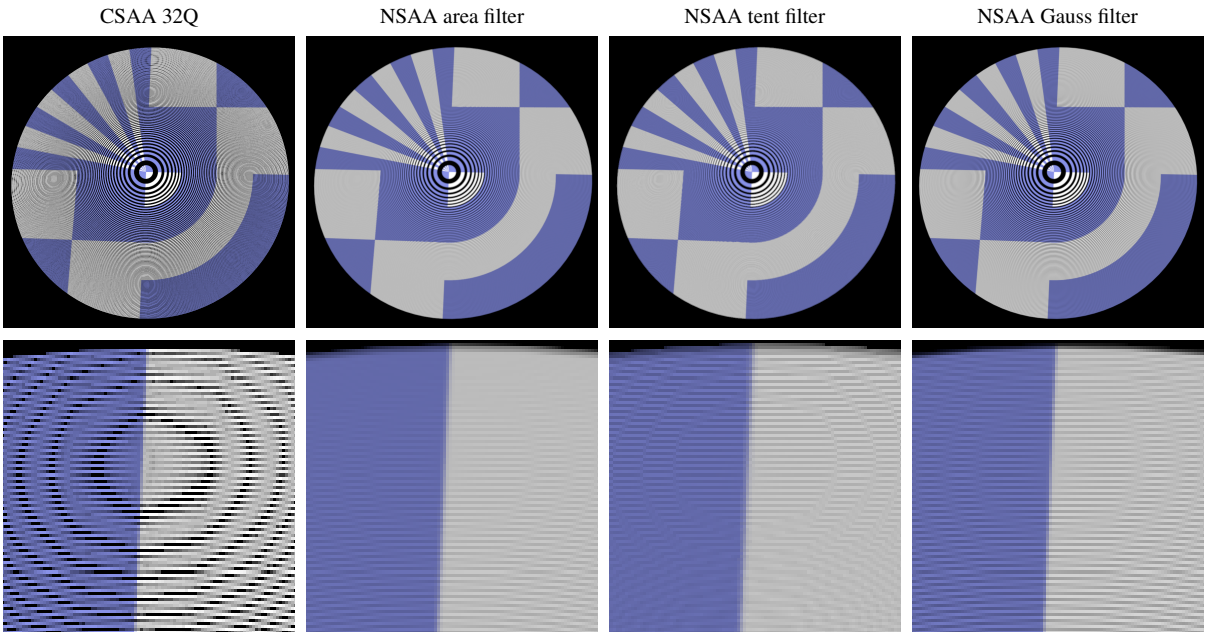

Figure 6: Comparison of different prefilters. Since our methods provides an exact evaluation of the filter convolution with the visibility signal, our results can be used to directly compare prefilters on general scenes. A comparison with CSAA 32Q is given to show that aliasing artifacts of common multisampling techniques would dominate the effects of different filter functions. We show the results for three radially symmetric filters each with a radius of two pixel: a constant area filter, a cone shaped tent filter and a Gaussian. See Figure 4 for a further explanation of the two rows.

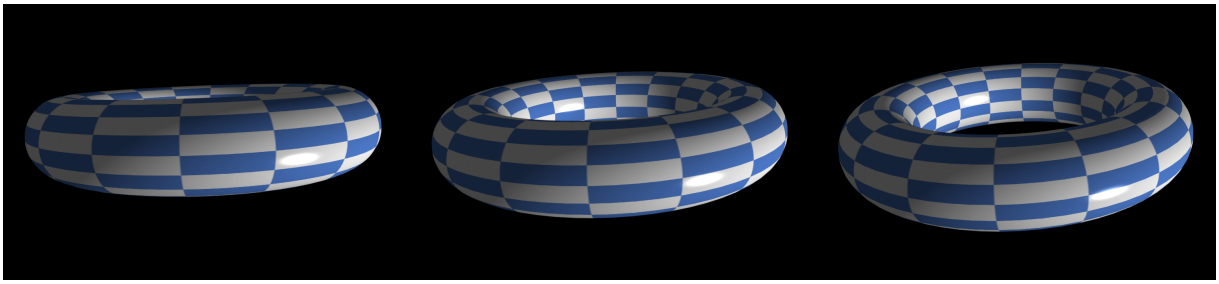

Figure 7: A sequence of torus models rasterized with our technique using a Gaussian prefilter of a radius of two pixels. Note that we correctly handle self occlusions of the torus geometry without using any depth buffering methodology. The silhouette exhibits high-quality anti-aliasing and we support general shading models such as texture mapping and Phong shading.

4.3. Performance

In this section we provide an overview of the performance characteristics of our method. For the presented test patterns in Figures 4-6, which consist of up to 20k triangles, we obtain near-interactive frame rates on a GeForce 680 with 4GB device memory. As expected, the weight computation stage is the bottleneck of our pipeline. More than 90% of the computation time is spent on the exact visibility and the analytic prefiltering, whereas less than 5% of the time is consumed by the rasterization-based sections of the pipeline. Due to the necessity of storing IDs and weights for all primitive fragments, the memory requirement are in the 100MB range for the aforementioned test scenes. A future extension of our framework that we are planning to develop is a tile-based rendering approach. For complex scenes, both the increased amount of geometry and textures would limit the available memory for our method. Using an adaptive screen-space tiling, depending on the available memory, would expand the capabilities of our framework to rasterize nearly arbitrary scenes. Even when processing only parts of a scene, we expect an adequate saturation of the processing elements of the graphics hardware.

5. Conclusion

We presented an analytic prefiltering framework to perform exact edge anti-aliasing with polynomial filter functions. An implementation on graphics hardware was described and its evaluation given. Furthermore, we showed the capability of our system to serve as a ground truth for the evaluation of sample-based anti-aliasing methods.

The main limitations of our work stem from the employed analytic visibility method, which does not support intersecting primitives and requires consistently orientated input. This is also the most promising avenue for future work as the analytic visibility method could be generalized to accommodate not only intersecting triangles but also transparency effects. The framework can also be extended to allow for separable filter to give a complete framework for the evaluation of prefilter performances.

References

[AGJ12] AUZINGER T., GUTHE M., JESCHKE S.: Analytic anti-aliasing of linear functions on polytopes. *Comp. Graph. Forum 31*, 2pt1 (May 2012), 335–344. 3, 5

[AMHH08] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008. 2

[AWJ13] AUZINGER T., WIMMER M., JESCKE S.: Analytic visibility on the GPU. *Comp. Graph. Forum 32*, 2pt4 (2013), 409–418. 3, 4, 5

[Cat74] CATMULL E. E.: *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, 1974. AAI7504786. 2

[Cat78] CATMULL E.: A hidden-surface algorithm with anti-aliasing. In *Proc. of SIGGRAPH '78* (New York, NY, USA, 1978), ACM, pp. 6–11. 2

[Cat84] CATMULL E.: An analytic visible surface algorithm for independent pixel processing. In *Proc. of SIGGRAPH '84* (New York, NY, USA, 1984), ACM, pp. 109–115. 2

[CJ81] CHANG P., JAIN R.: A multi-processor system for hidden-surface-removal. *SIGGRAPH Comput. Graph. 15*, 4 (Dec. 1981), 405–436. 2

[CML11] CHAJDAS M. G., MCGUIRE M., LUEBKE D.: Subpixel reconstruction antialiasing for deferred shading. In *Proc. of I3D '11* (New York, NY, USA, 2011), ACM, pp. 15–22. 2

[Cro77] CROW F. C.: The aliasing problem in computer-generated shaded images. *Commun. ACM 20*, 11 (Nov. 1977), 799–805. 1, 2

[Duf89] DUFF T.: Polygon scan conversion by exact convolution. *Proc. of Raster Imaging and Digital Typography 1989* (1989), 151–168. 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. In *Proc. of SIGGRAPH '88* (New York, NY, USA, 1988), ACM, pp. 21–30. 2

[Fra80] FRANKLIN W. R.: A linear time exact hidden surface algorithm. In *Proc. of SIGGRAPH '80* (New York, NY, USA, 1980), ACM, pp. 117–123. 2

[GT96] GUENTER B., TUMBLIN J.: Quadrature prefiltering for high quality antialiasing. *ACM Trans. Graph. 15*, 4 (Oct. 1996), 332–353. 2

[HAMO05] HASSELGREN J., AKENINE-MÖLLER T., OHLSSON L.: Conservative rasterization. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*, Pharr M., Fernando R., (Eds.). Addison-Wesley Professional, 2005. 4

[HB10] HOBEROCK J., BELL N.: Thrust: A parallel template library, 2010. Version 1.7.0. 4

[JESG12] JIMENEZ J., ECHEVARRIA J. I., SOUSA T., GUTIERREZ D.: Smaa: Enhanced subpixel morphological antialiasing. *Comp. Graph. Forum 31*, 2pt1 (May 2012), 355–364. 2, 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 Courses* (New York, NY, USA, 2011), ACM, pp. 6:1–6:329. 2

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

[McC95] MCCOOL M. D.: Analytic antialiasing with prism splines. In *Proc. of SIGGRAPH '95* (New York, NY, USA, 1995), ACM, pp. 429–436. 2

[McK87] MCKENNA M.: Worst-case optimal hidden-surface removal. *ACM Trans. Graph. 6*, 1 (Jan. 1987), 19–28. 2

[Mit87] MITCHELL D. P.: Generating antialiased images at low sampling densities. In *Proc. of SIGGRAPH '87* (New York, NY, USA, 1987), ACM, pp. 65–72. 2

[MN88] MITCHELL D. P., NETRAVALI A. N.: Reconstruction filters in computer-graphics. In *Proc. of SIGGRAPH '88* (New York, NY, USA, 1988), ACM, pp. 221–228. 2

[MS11] MANSON J., SCHAEFER S.: Wavelet rasterization. *Comp. Graph. Forum 30*, 2 (2011), 395–404. 2

[MS13] MANSON J., SCHAEFER S.: Analytic rasterization of curves with polynomial filters. *Comp. Graph. Forum 32*, 2pt4 (2013), 499–507. 3, 5

[Mul89] MULMULEY K.: An efficient algorithm for hidden surface removal. In *Proc. of SIGGRAPH '89* (New York, NY, USA, 1989), ACM, pp. 379–388. 2

[Res09] RESHETOV A.: Morphological antialiasing. In *Proc. of HPG '09* (New York, NY, USA, 2009), ACM, pp. 109–116. 2

[RKLC*11] RAGAN-KELLEY J., LEHTINEN J., CHEN J., DOGGETT M., DURAND F.: Decoupled sampling for graphics pipelines. *ACM Trans. Graph. 30*, 3 (May 2011), 17:1–17:17. 2

[SO92] SHARIR M., OVERMARS M. H.: A simple output-sensitive algorithm for hidden surface removal. *ACM Trans. Graph. 11*, 1 (Jan. 1992), 1–11. 2

[SSS73] SUTHERLAND I. E., SPROULL R. F., SCHUMACKER R. A.: Sorting and the hidden-surface problem. In *Proc. of AFIPS '73* (New York, NY, USA, 1973), ACM, pp. 685–693. 2

[Tur86] TURKOWSKI K.: Anti-aliasing in topological color spaces. In *Proc. of SIGGRAPH '86* (New York, NY, USA, 1986), ACM, pp. 307–314. 2

[WA77] WEILER K., ATHERTON P.: Hidden surface removal using polygon area sorting. In *Proc. of SIGGRAPH '77* (New York, NY, USA, 1977), ACM, pp. 214–222. 2

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