Sampled and Analytic Rasterization

Thomas Auzinger and Michael Wimmer
Vienna University of Technology

VMV2013
September 11–13, 2013
Lugano, Switzerland

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Introduction

In computer graphics, two main data formats are widely used:

- **Raster formats** (e.g. for image acquisition, processing and display)
- **Vector formats** (e.g. for text or object surfaces).

The term **rasterization** refers to:

- The **conversion** of vector information to a raster format
- A 3D **rendering technique** which uses intermediate raster buffers throughout the graphics pipeline.

We use both aspects in our work.

Rasterization is a fundamental task in computer graphics. The discrete nature of raster formats, however, limits their capability of retaining small details and special care has to be taken to avoid the introduction of **artifacts** during rasterization (see Section Anti-Aliasing on the right).

In the last years, we investigated how to enable **prefiltering** during rasterization to lift the limitations of supersampling approaches. In Table 1, a classification of anti-aliasing techniques is given together with our works. So far, we achieved:

- **Visibility and shading prefiltering** for simple shading models [2]
- **Visibility prefiltering** and sampled shading for general shading models [3]
- **Shading prefiltering** for 2D rendering [1], which is a special case of the full 3D variant [2].

In our approach, we replace the visibility and/or shading stage of the graphics pipeline to retain the vector character of the input data as long as possible (see Figure 1).

The new design still maps well to massively parallel hardware architectures and was implemented using **GPGPU**. An explanation of the two most relevant novel stages can be found in the Sections Analytic Visibility and Analytic Shading below.

		Visibility			
		Trivial	Sampled	Supersampled	Prefiltered
Shading	Sampled	Standard	Standard	MSAA, CSAA	NSAA [3]
	Supersampled	Font rasterization	—	SSAA, Decoupled	**Future work**
	Prefiltered	Analytic 2D [1]	—	—	Analytic 3D [2]

Table 1: Classification of anti-aliasing methods for rasterization. The traditional sampling based approaches have blue background whereas our works on prefiltering are underlaid with green.

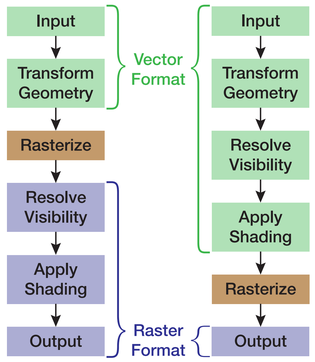


Figure 1: The standard (left) and our analytic (right) rasterization pipeline.

Anti-Aliasing

Rasterization of vector data to a raster output is equivalent to the **sampling** of a continuous input signal at discrete sample locations.

According to the **Nyquist-Shannon sampling theorem**, the density of the sample locations limits the finest details that are preserved by the sampling process. Finer details lead to **aliasing artifacts** in the sampled output (see Figure 2 on the right). Two main methods exist to combat these artifacts, i.e., to perform anti-aliasing:

- **Supersampling**
  This method samples the input data at a higher rate and then reduce the number of samples by averaging the supersamples with a suitable filter. Since the filtering happens after sampling, this is also called postfiltering. Note that this method still suffers from aliasing (see Figure 3 on the right).

- **Prefiltering**
  This methods smooths the data before the sampling with a suitable prefilter. This is the mathematically correct way to address aliasing issues (see Figure 4 on the right) but computationally expensive. We employ this method for analytic rasterization.

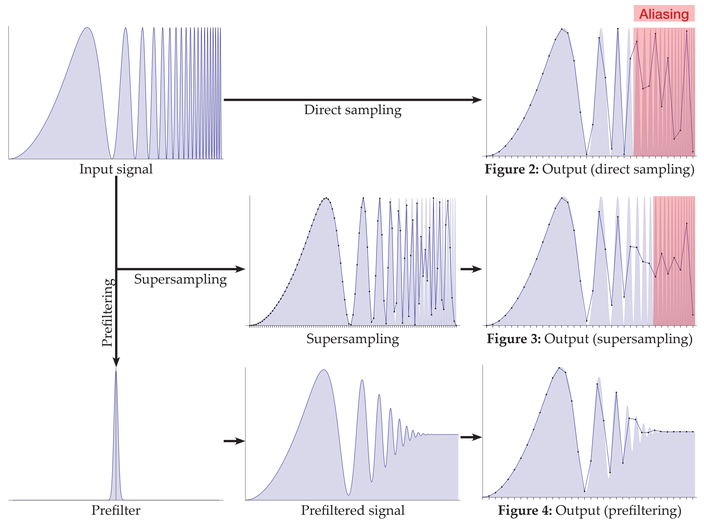


Figure 2: Output (direct sampling)

Figure 3: Output (supersampling)

Figure 4: Output (prefiltering)

Analytic Visibility

To enable **visibility prefiltering**, the visible parts of the projected scene primitives have to be determined exactly. We employ **parallel hidden surface elimination** for this task. The computations can be executed in parallel for all edges of the input triangles. This provides sufficient data parallelism to effectively use massively parallel hardware such as **GPUs**. The main stages of our analytic visibility method are described below.

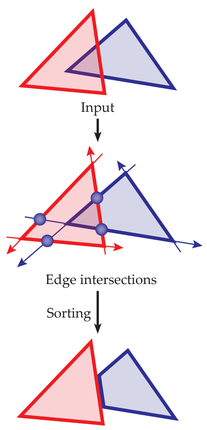
Intersection computation

The visibility of a triangle edge only changes at the borders of another triangle. For a given edge of a scene triangle, we thus compute all intersections with other triangles along the line that contains this edge.

```
Input:  E (set of all edges in the scene)
Output: I (set of all intersections)

for each edge e of E in parallel
  T ← triangles of local neighborhood
  for each triangle t of T in parallel
    o    ← get global memory offset
           // with atomics
    I[o] ← add intersections of e and t
  end for
end for
```

Pseudo-code of the intersection computation



Intersection sorting

Intersections are written concurrently and unordered into global memory, thus we perform a sorting step to order the intersection along each edge.

Hidden surface elimination

We walk along each edge and use the sorted intersections to count the number of triangles that occlude each intersection. A scan is used for this task. All visible edge segments, which are the output of this stage, are then stored in global memory. These segments constitute the **borders of all visible regions** of all projected scene primitives.

Analytic Shading

This stage computes the **convolution** of a function **with the prefilter** on all visible regions of the associated primitive. To perform full analytic rasterization, a simple shading function can be used [1, 2]. If the shading model cannot be evaluated in closed form, a constant function can be used to compute the convolution integral for analytic visibility prefiltering [3].

Integral computation

For the given sample locations (usually the pixel centers of the output image) the contribution of all triangles to them is computed. At each sample location a radial **prefilter** is placed and convolved with the function on the visible region of all triangles.

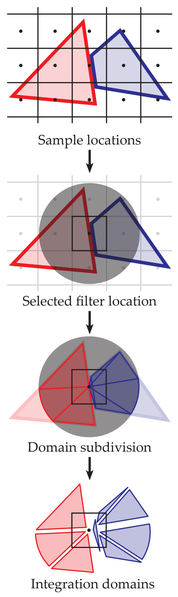
The visible region of each triangle is given by its boundary, i.e., the line segment output of the analytic visibility stage.
Each line segment and the sample location spans an **integration domain** that is clipped against the support of the filter over which is integrated.

Due to the linearity of integration, the integrations over the various domains can be done independently. This fact is exploited in our **parallel** implementation that is highly suitable for massively parallel hardware.

```
Input:  B (set of boundary segments of the
           visible regions of all triangles)
        S (set of sample locations)
Output: C (set of pixel colors)

for each sample location s of S in parallel
  L ← boundary segments of B of local tile
  for each segment l of L in parallel
    c    ← compute prefilter convolution of l
    C[s] ← add color c
  end for
end for
```

Pseudo-code of the integration computation

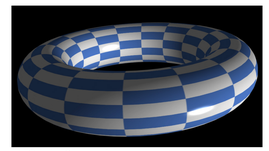


Results and Future Work

We gained **several key observation** through our research. The mathematically exact nature of prefiltering has higher compute and storage requirements when compared to massive supersampling. Since the actual raster conversion occurs at the end of the pipeline, all intermediate buffer sizes depend on the input complexity. Furthermore, exact geometrical computations and lengthy closed-form solution of the filter convolutions cause increased computational requirements. Prefiltering is in general 2–3 orders of magnitude **slower** than supersampling but results in a **ground truth** solution that is exact up to numerical precision (see Figure 5).

Only **simple shading** models are analytically integrable and sampling has to be used for more advanced effects.

In the **future**, we plan to combine **visibility prefiltering with shading supersampling** as shown in Table 1. This should provide the best achievable quality and set the ground truth reference for anti-aliasing methods. A main **challenge** is the correct sampling of the shading function on the complicated visible regions of the projected scene primitives.



NSAA [3] combines exact visibility prefiltering with advanced non-linear and discrete shading models such as texture mapping and Phong illumination.

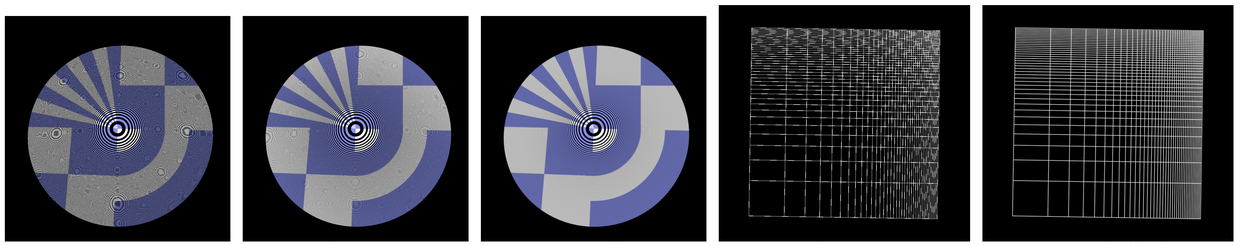


Figure 5: Comparison of various anti-aliasing methods. Our exact prefiltering approaches yield ground truth solutions to complex scenes with highly detailed geometry (left) or line rendering (right).

References

[1] T. Auzinger, M. Guthe, S. Jeschke, *Analytic Anti-Aliasing of Linear Functions on Polytopes*, Computer Graphics Forum (EG '12), 31(2):335–344, 2012

[2] T. Auzinger, M. Wimmer, S. Jeschke, *Analytic Visibility on the GPU*, Computer Graphics Forum (EG '13), 32(2):409–418, 2013

[3] T. Auzinger, P. Musialski, R. Preiner, M. Wimmer, *Non-Sampled Anti-Aliasing*, Proc. 18th Int. Workshop on Vision, Modeling and Visualization (VMV '13), 2013

Contact

Name: Thomas Auzinger

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

Email: thomas.auzinger@cg.tuwien.ac.at

LinkedIn: linkedin.com/in/thomasauzinger