Adaptive Supersampling for Deferred Anti-Aliasing

Matthias Holländer[1] Tamy Boubekeur[1] Elmar Eisemann[2]
[1]Telecom ParisTech - CNRS/LTCI [2]Delft University of Technology

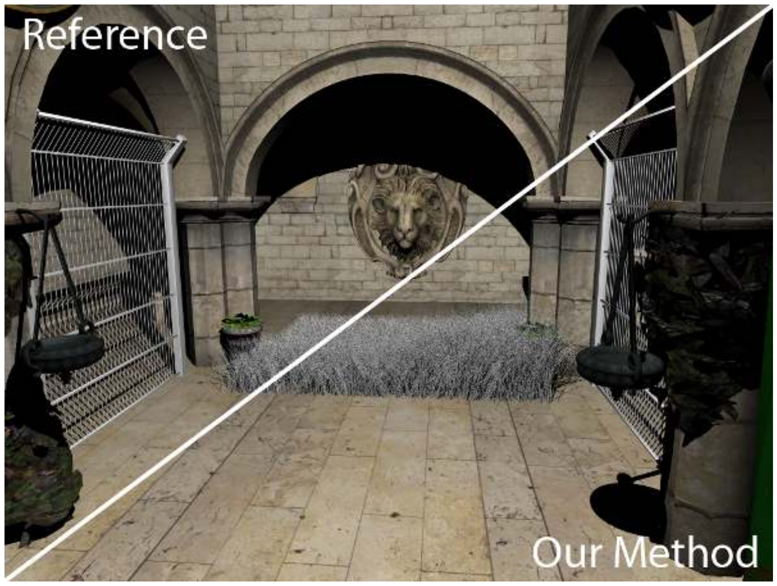

Figure 1. Left: Brute-force 4×4 supersampling result. Right: Comparable quality result produced using fewer operations by our adaptive shading of a 4×4 supersampled G-Buffer.

Abstract

We present a novel approach to perform anti-aliasing in a deferred-rendering context. Our approach is based on supersampling; the scene is rasterized into an enlarged geometry buffer, i.e., each pixel of the final image corresponds to a window of attributes within this buffer. For the final image, we sample this window adaptively based on different metrics accounting for geometry and appearance to derive the pixel shading. Further, we use anisotropic filtering to avoid texturing artifacts. Our approach concentrates the workload where needed and allows faster shading in various supersampling scenarios, especially when the shading cost per pixel is high.

1. Introduction

The process of applying lighting effects to the surface of an object is commonly called *shading*. Shading in real-time applications can be performed using a *forward-rendering* approach, which sends the scene's geometry to the GPU and performs lighting calculations for each rasterized pixel. During this process, the scene's information resides only temporarily in GPU memory. Consequently, shading has to be computed right away, even for pixels that might end up hidden. Performance falls when shading dominates the rendering cost. For complex scenes it is unavoidable that shading dominates, since, even under a simple shading algorithm, forward-rendering shading is necessarily linear in the number of light sources. It thus becomes arbitrarily expensive as the number of lights grows. To avoid shading hidden fragments, we populate the z-buffer with depth values by rendering the scene once without shading (i.e., an early-depth pass). In a subsequent pass, the geometry is transformed and rendered again, but only fragments surviving the depth test are shaded. Nonetheless, even then, the computations do not map perfectly to the GPU when triangles are pixel-sized or very small and an overhead is added.

Deferred shading is an alternative and better approach. Instead of lighting each fragment directly, a deferred-shading renderer writes each fragment's attributes to a geometry buffer (G-buffer). The attributes commonly include normals, positions, depth values, and material properties. Pixels of the final image are then shaded by making one pass per light over the regions of the screen affected by the light. Each pass makes use of the captured data in the G-buffer. This results in higher shading performance (provided the overhead of reading the values multiple times is significantly smaller than the cost of shading; a variation called *tiled deferred shading* addresses this).

Because a single G-buffer pixel will result in a single pixel in the final output, aliasing can become a severe problem under deferred shading. One can address this issue by supersampling. However, supersampling is expensive, because it shades substantially more samples than either forward shading with analytic or multisample anti-aliasing would, even in regions where no anti-aliasing would have been required.

In this paper, we propose a solution that is faster than brute-force supersampling and delivers high-quality results. Our method makes use of an adaptive sampling algorithm and a supersampled G-buffer, leading to an anti-aliasing solution for a deferred rendering context. This paper describes the key contributing elements of this method:

- adaptive sampling for deferred shading,
- two orthogonal metrics for driving that adaptation, and
- efficient anisotropic filtering for deferred shading.

2. Previous Work

For anti-aliasing (AA), multiple samples per pixels are needed. This implies many shading evaluations in forward rendering, e.g., MSAA (multi-sample AA) is a very common approach. Here, coverage (visiblity) is sampled finely, while shading is evaluated coarsely. Thibieroz [2009] showed that MSAA can be combined with deferred shading using a customized resolve operation to achieve plausible results. However, the stored samples are derived from MSAA samples, i.e., not all samples correspond to actual surface samples. Further, a fixed shading rate is used for edges instead of a variable one. Surface-based anti-aliasing (SBAA) [Salvi and Vidimče 2012] can be used to reduce storage costs in these scenarios by limiting the amount of per-pixel surface information considered for shading. This involves discarding primitives with small coverage to trade accuracy for a smaller memory footprint. A large body of work on real-time post-process anti-aliasing seeks to ameliorate image quality by adjusting the values in the color buffer after shading a low-resolution buffer. See Jimenez et al. [2011] for a survey of these methods. In contrast, we aim at providing an adaptive strategy for deferred shading to add samples where needed. Here, we will review the most recent methods that we believe are all currently in production use.

Fast approximate anti-aliasing (FXAA) [Lottes 2009], morphological anti-aliasing (MLAA) [Reshetov 2009], and directionally localized anti-aliasing (DLAA) [Andreev 2011] are purely RGB-based post-process filters and can be used to get rid of some type of aliasing artifacts. They resample the input texture with a slight offset for edge regions, thus effectively blurring the edges. Although this can remove some of the deficiencies, it can lead to an overall blur. Further, false positives for non-edge regions can appear, and the approach is not temporally coherent. Some of those filters are known to work in supersampling modes, but an academic description and the way samples are selected are, however, not available.

Subpixel reconstruction anti-aliasing (SRAA) [Chajdas et al. 2011] is similar to MLAA but relies on a super-resolution depth (and optional normal) buffer. Subpixel shading values are reconstructed using bilateral upsampling involving weights defined from neighboring pixels based on depth/normal information. In contrast to our method, they use a fixed subset of normal and depth values that is chosen blindly.

Valient's [2007] multi-sample anti-aliasing (MSAA) x2 averages the result of two samples. This leads to shading artifacts at boundaries of objects with different materials. Coverage-sampled anti-aliasing (CSAA) [Young 2007] cannot be combined with deferred shading at all.

Low-resolution rendering and careful upsampling can help to increase rendering speed. Yang et al. [2008], the authors render the image in low resolution and use bilateral upsampling for the final pass based on normal and depth information. A similar idea is known from the demo scene [Swoboda 2009] for multi-sample anti-aliasing.

For inferred lighting [Kircher and Lawrance 2009], an additional attribute combining a normal-group-id and an object-id is added to a low-resolution G-buffer. The unique id and depth are used to efficiently reject pixels from different objects and thus, non-edge-regions can be shaded with fewer samples. Nonetheless, aliasing artifacts can occur where brightly lit and dark surfaces meet due to the upsampling strategy. Additionally, the z-prepass optimization described above cannot be used because the initial depth buffer (first pass) is rendered at a lower resolution.

Finally, Lauritzen [2010] uses a tile-based deferred shading approach combined with quad-based light culling. Indeed, they offer an adaptive shading rate using normal and depth information with additional G-buffer storage cost for depth derivatives. However, this metric neglects shading discontinuities and does not offer anisotropic texture filtering.

3. Our Method

The idea of our algorithm is to produce a high-resolution G-buffer, but to perform shading only on a selected subset of these pixels. This subset is chosen via our metric that predicts where more samples are required.

At first, the scene's information is rendered into an enlarged G-buffer of size $(m \cdot w) \times (m \cdot h)$, where w and h are the width and height of the window, and $m \in \mathbb{N}$. Each pixel of the final image corresponds to a sampling window of size $m \times m$ in the large G-buffer.

Then, we shade one sample for each pixel and store the result in a separate buffer $\mathcal{S}$, which has, therefore, a resolution of $w \times h$.

Finally, we will derive the number of additional samples k that should be involved in shading a given pixel via two metrics. The first metric analyzes the contents of the G-buffer within a sampling window, and the second one looks at the actual shading values in $\mathcal{S}$.

3.1. Determining the Required Sample Number

For a sampling window of size $m \times m$, a maximum of $n = m^2$ samples per pixel is possible. Each metric has a user-defined constant c to scale the resulting value that should be adapted according to the scene. As aliasing occurs either at geometric discontinuities or discontinuities caused by shading, we propose the following metrics:

Geometry based metric. First, we compute the variance of the depth values in the sampling window for the corresponding pixel to deduce a first estimate of the required number of samples; a higher variance results in more samples:

$$k_d = c \times (\frac{\sum d_i^2}{n} - (\frac{\sum d_i}{n})^2).$$

Next, we investigate the normals and compute the cone spanned by them in the sampling window. We compute the average normal $\bar{n}$ from the sampling window and compare all other normals $\vec{n_{ij}}$ against it, effectively computing the conservative cone centered around the mean normal. The resulting number of samples is given by

$$k_n = c \times \sqrt{1.0 - (\min_{ij}(\text{abs}((\bar{n} \cdot \vec{n_{ij}}))))}.$$

The total sample count is

$$k = \max(k_d, k_n).$$

Shading-based metric. The second metric is based on the shading texture $\mathcal{S}$. For each pixel the range r of luminance values is computed by looking at the four direct

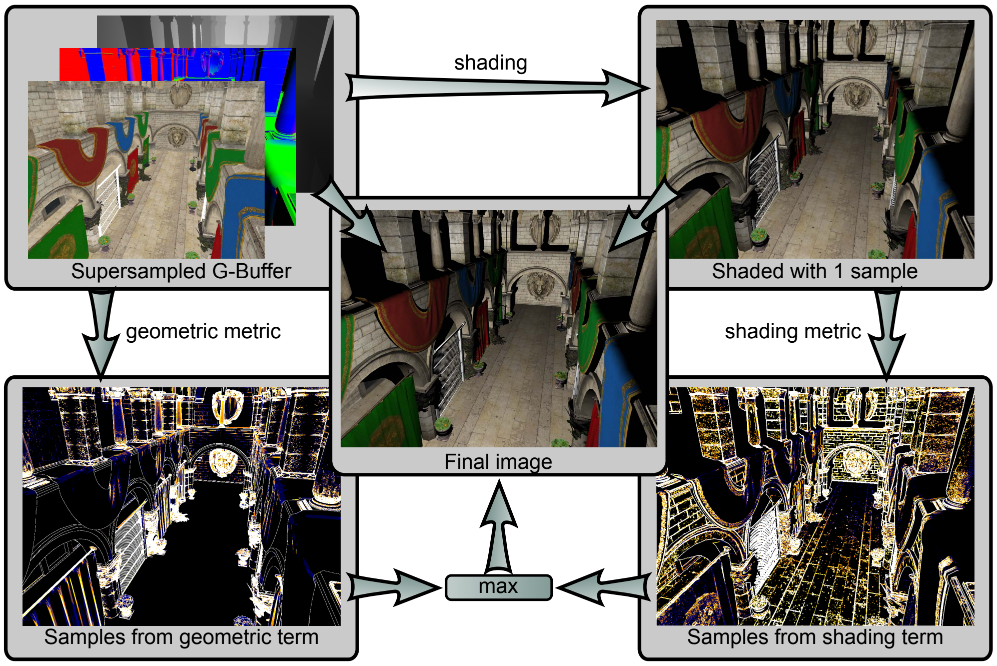

Figure 2. Overview of our method. Our deferred shading pass renders to an enlarged G-buffer, i.e., each pixel of the final image corresponds to a window of this buffer. Then, the G-buffer is evaluated using our geometric metric. The first sample for each pixel is shaded and our shading metric is applied. For the final image, we shade each pixel adaptively based on the result of our metrics and the previously rendered image with one sample per pixel.

neighboring pixels, resulting in $r = l_{\max} - l_{\min}$. The final sample count is

$$k_s = \begin{cases} 0 & \text{if } r < \max(t_{\text{minThreshold}}, l_{\max} * t_{\text{edgeThreshold}}), \\ c \times r, & \text{otherwise.} \end{cases}$$

This is similar to metrics found in RGB-filter-based AA methods such as FXAA [Lottes 2009].

The presented metrics are orthogonal to each other as they rely on different attributes of the scene. The result of the geometric metric and the shading metric are combined using a max filter. Notwithstanding, depending on the usage scenario (e.g., pure geometric aliasing), one could omit the shading metric. Furthermore, even other sampling methods based on the BRDF [Bagher et al. 2012] could be integrated, but the above solution is faster to compute and sufficed in all experiments.

3.2. Sampling the Window

Once the required sample count has been computed, we use k samples from the G-buffer of the pixel's corresponding sampling window to compute the final shading. We distribute the k samples according to an index matrix known from ordered dithering [Bayer 1973] (see Figure 3). These matrices are constructed such that the average distance between two consecutive numbers is maximized. Thus, we can ensure a good distribution of the samples that are taken from the supersampled G-buffer.

$$\begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix} \begin{pmatrix} 2 & 6 & 3 \\ 5 & 0 & 8 \\ 1 & 7 & 4 \end{pmatrix} \begin{pmatrix} 1 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 0 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

Figure 3. Index matrices for sizes $m = 2, 3, 4$ that are used to sample the large G-buffer.

4. Implementation

For our implementation, we chose a G-buffer format (Figure 4) that is similar to those found in games [Valient 2007]. Additionally, we store all textures of the scene in a 3D-array. This allows us to delay the albedo lookup until the final shading pass where texture aliasing artifacts can be removed by using anisotropic filtering.

We implemented our algorithm using multiple passes (see Listing 1). The first pass is a deferred shading pass where the destination G-buffer is m^2 times larger. One important element is that it will not store albedo-texture values but rather the corresponding texture-layer indices and texture coordinates.

The next pass computes the required number of samples per pixel by evaluating the sampling window with our metric. The final pass combines the result of the shade-first texture $\mathcal{S}$ and performs adaptive shading using the rest of the G-buffer; samples

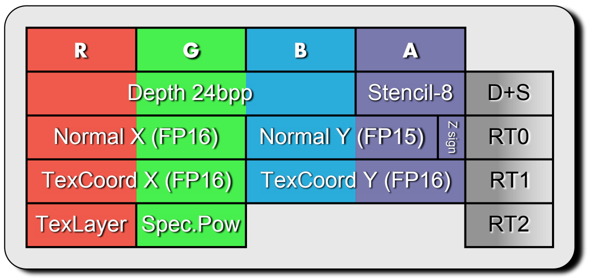

Figure 4. Our G-buffer format. Each channel has 8 bits.

```
GBuffer g = RenderToGBuffer();
Tex oneSample = ShadeFirstSample();

// evaluation using our two metrics
Tex samplecount = Evaluate( g, oneSample );

ShadeRestAdaptively( samplecount, oneSample, g );
```

Listing 1. Pseudocode for our algorithm.

are picked from the sampling window for each pixel according to the sample count and following the dither matrices (Figure 3).

4.1. Shading a Sample

Each sample that contributes to the final pixel is shaded by evaluating the BRDF and performing a texture lookup into the scene's array texture. Although our metric does incorporate shading, we have to avoid texture artifacts. By computing the derivatives of the texture coordinates in the G-buffer and taking into account the number of used samples in the window, we can use anisotropic texture lookups to better approximate the underlying texture footprint of the sample. The derivative computation is available through the `dFdx`/`dFdy` functions in GLSL. The anisotropic texture lookup is then performed using the `textureGrad` function.

Such a filtering would fail at the boundary of two different materials. Fortunately, we can detect these discontinuities because they usually appear at a high-order derivative in the texture layer/coordinates. The influence of the anisotropic texture lookups on the result are shown in Figure 5.

4.2. Evaluating Several Samples

There are several possibilities to evaluate multiple samples for a given pixel and our approach can employ several alternatives—the choice depends on the hardware spec-

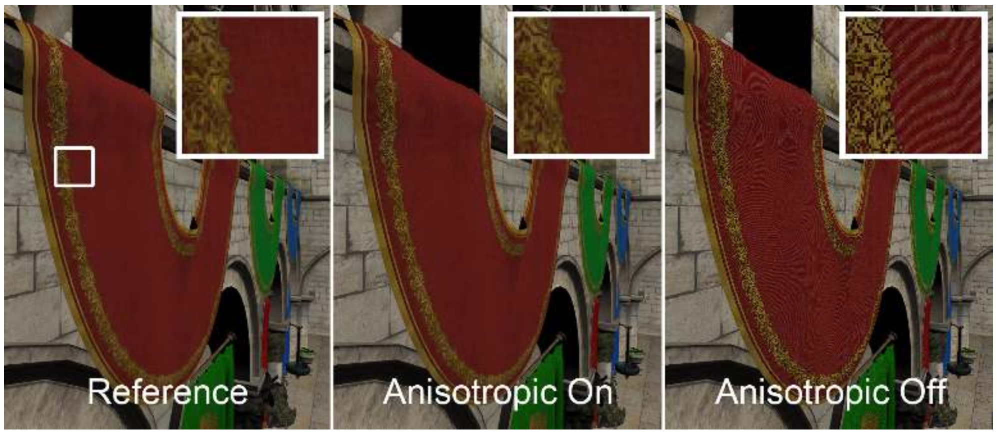

Figure 5. Left: Reference using full 3×3 supersampling with anisotropic filtering; Middle: our approach with anisotropic filtering; Right: our approach without anisotropic filtering.

ifications.

The easiest solution is to read out the number of needed samples and use a loop in a shader to evaluate and accumulate the result of the samples taken from the window. In practice, this often proved most efficient, despite potential divergence between the different threads.

Another solution is to write out the required number of needed samples per pixel into a stencil target. For the final shading pass, the stencil test is enabled and a full-screen quad is drawn for each possible sample count $k \in 2, \ldots, m^2$ where the stencil test discards all fragments different from k. This effectively groups all pixels with the same sample count during shading but requires the stencil mask to hold the correct number of samples and, thus, stencil-write capabilities from within a shader. At the time of writing, this extension has not been integrated into the OpenGL standard (`GL_ARB_shader_stencil_export`). Similarly, the sample count can be written to the depth buffer using `gl_FragDepth` followed by an evaluation using multiple screen-sized quads which are set to the corresponding depth and a depth comparison

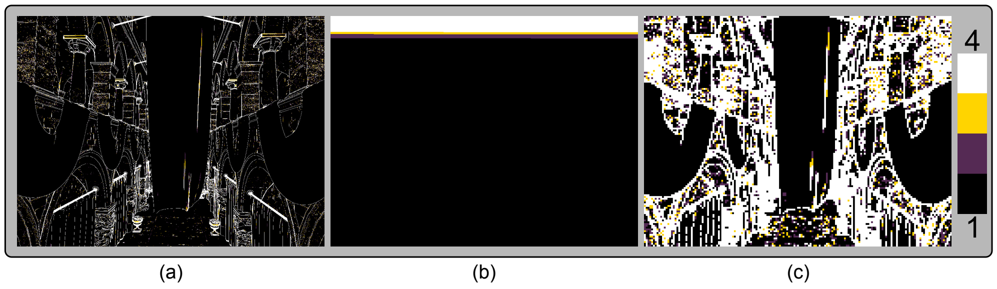

Figure 6. (a) Sample count per pixel as heatmap; (b) sorted sample counts (mostly zero); (c) low-resolution sample count using a max filter on a 6×6 window

with `GL_EQUAL`. However, this resulted in roughly 20% slower performance in all of our tests. The sample count values can benefit from *bucketing* by dynamically computing a histogram of the values and reducing the co-domain. Such a strategy, however, involves further investigation to avoid compromising quality and is left for future work.

We also experimented (see Figure 6(b)) with sorting the pixels according to their computed sample count, inspired by Hoberock et al. [2009]. Unfortunately, a group of pixels with the same sample count (e.g., all pixels with $k = 4$) does not necessarily have a high image-space coherence (see Figure 6(a) and Figure 7), so that texture accesses within this group are usually incoherent, thereby, limiting performance. Sorting samples was three times slower than our presented solution.

Also, we tried using a low-resolution sample count texture by applying a max-filter to the original sample count texture. This filter preserves high sample counts on edges that require AA. Now, each texel of this low-resolution sample count texture corresponds to a group of pixels in the image. Unfortunately, this increases the total number of samples drastically because high sample counts "leak" into regions that would only require a low number of samples (Figure 6(c)). Again, sorting the low-resolution sample count values did not improve performance (also roughly three times slower).

5. Results

In this section, we present the results of our algorithm. We compare our method to a reference solution that evaluates all n samples for all pixels with anisotropic filtering enabled. The reference solution and our algorithm are based on an ordered-grid su-

Supersampling	Method	ms/frame	stalled	MSE ($\times 10^{-5}$)	PSNR dB	Samples %
1×1	Default	12.91	28.91	–	–	–
	FXAA	13.28	29.24	–	–	–
2×2	Reference	26.0	78.7	–	–	100
	Ours	22.9	72.2	5.0	91.14	57.8
3×3	Reference	34.9	158.36	–	–	100
	Ours	30.69	143.77	3.08	93.24	53.40
4×4	Reference	57.48	317.76	–	–	100
	Ours	51.68	268.35	3.79	92.34	53.14

Table 1. Performance of our algorithm for the Sponza scene. The reference solution evaluates all samples for each pixel. PSNR (peak signal-to-noise ratio) and MSE (mean-squared error) given regarding the reference, similarly the percentage of samples shaded in contrast to the full evaluation is given. All timings are given in ms.

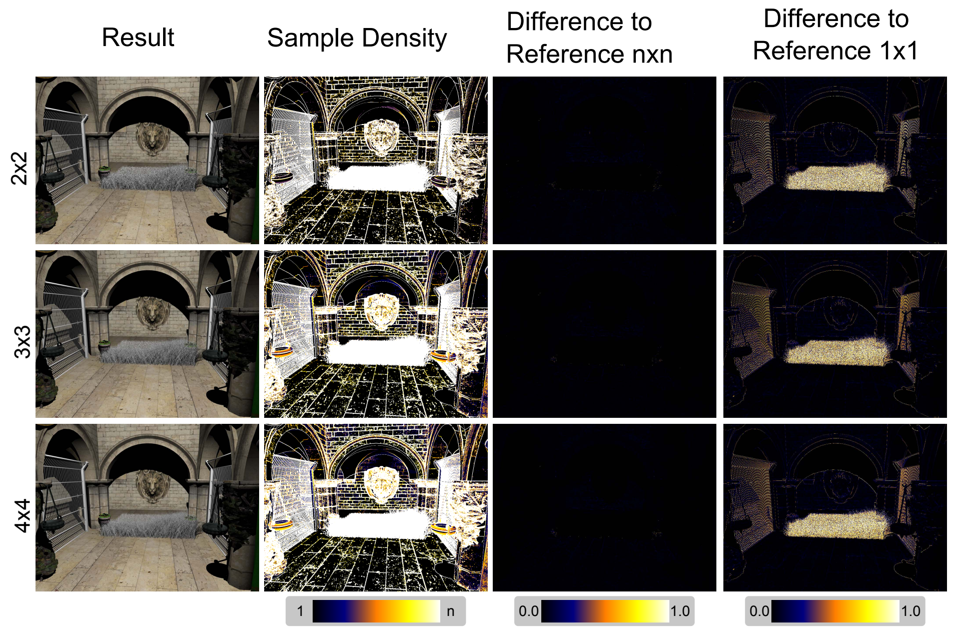

Figure 7. Our method at different supersampling rates. The third column shows the difference to the reference image at the same sampling rate whereas the last column compares to a 1×1 casual deferred shading. Both display the absolute error encoded as a heatmap. The difference from the $n \times n$ solution is nearly zero everywhere because although faster to produce than the reference solution, our results are nearly identical. Zoom into the figure to see the single-pixel differences.

persampling AA implementation. Performance results are illustrated in Table 1. For each version, we measured the performance of real-time shading as well as shading that stalls the fragment shader (per sample), thus, effectively simulating a heavy and complicated fragment shader as can be found in offline or interactive rendering.

We also calculated the mean-squared error (MSE), the peak signal-to-noise ratio (PSNR), and the percentage of samples in comparison to the reference solution. The timings were measured on an Intel Core i7 2.67 GHz with a GeForce GTX 480 with 1536 MB VRAM at a (application) resolution of 1024×768 for our OpenGL-based implementation. We used the Sponza scene with 280 K triangles and 194 K vertices for our tests.

The impact of our geometric and shading term are shown separately in Figure 8. Our geometric term also reduces the artifacts when small geometric details cannot be captured by our shading texture $\mathcal{S}$.

Our adaptive sampling algorithm outperforms the reference solution in all cases while keeping the quality at a similar level. Even for real-time shading scenarios, with low-cost fragment shaders, we can observe a gain in performance. Our metrics reduce roughly by half the required number of samples to shade. However, when shading only half of the samples, a performance boost of a factor of two is not possible due to the divergence of threads on the GPU and some incoherent texture accesses between neighboring pixels.

Our metric gives only a small error (see Figure 7) as it is sensitive to changes in depth and normal as well as shading discontinuities.

Overall, our algorithm proves to be particularly useful for supersampling scenarios, especially, when the shading cost per pixel is high (see Table 1 (stalled versions)).

6. Discussion and Conclusion

We presented a method to increase performance of supersampling anti-aliasing scenarios in the context of deferred shading. Our results are usually close to the reference but involve much less computation time. The proposed metrics provide a good trade-off between performance and accuracy with a controllable error.

In the future, we would like to design light-source aware metrics in order to improve shading near specular highlights, e.g., one could already combine our solution with a light pre-pass [Engel 2008]. Further, we would like to address thread divergence by using a tile-based approach similar to Lauritzen's work [2010]. Also, as is common for antialiasing methods, temporal coherence is an interesting problem, and we would like to try to compensate for such artifacts by averaging the sample count over several frames, in the spirit of Herzog et al. [2010].

7. Acknowledgements

This work has been partially funded by the French National Research Agency *MediaGPU* project, the REVERIE E.U. project, the *3D Life* European Network of Excellence and the Intel Visual Computing Insititute (IVCI) at Saarland University.

References

ANDREEV, D., 2011. DLAA. http://and.intercon.ru/releases/talks/dlaagdc2011/. 3

BAGHER, M., M., SOLER, C., SUBR, K., BELCOUR, L., AND HOLZSCHUCH, N. 2012. Interactive Rendering of Acquired Materials on Dynamic Geometry using Bandwidth Prediction. In *Proc. Symposium on Interactive 3D Graphics and Games (I3D)*, ACM, New York, NY, USA, 127–134. 6

BAYER, B. 1973. An Optimum Method for Two-Level Rendition of Continuous-Tone Pictures. In *IEEE International Conference on Communications, Volume 1*, IEEE, Washington, DC, 11–15. 6

CHAJDAS, M., MCGUIRE, M., AND LUEBKE, D. 2011. Subpixel Reconstruction Antialiasing for Deferred Shading. In *Proc. Symposium on Interactive 3D Graphics and Games (I3D)*, ACM, New York, NY, USA, 15–22. 3

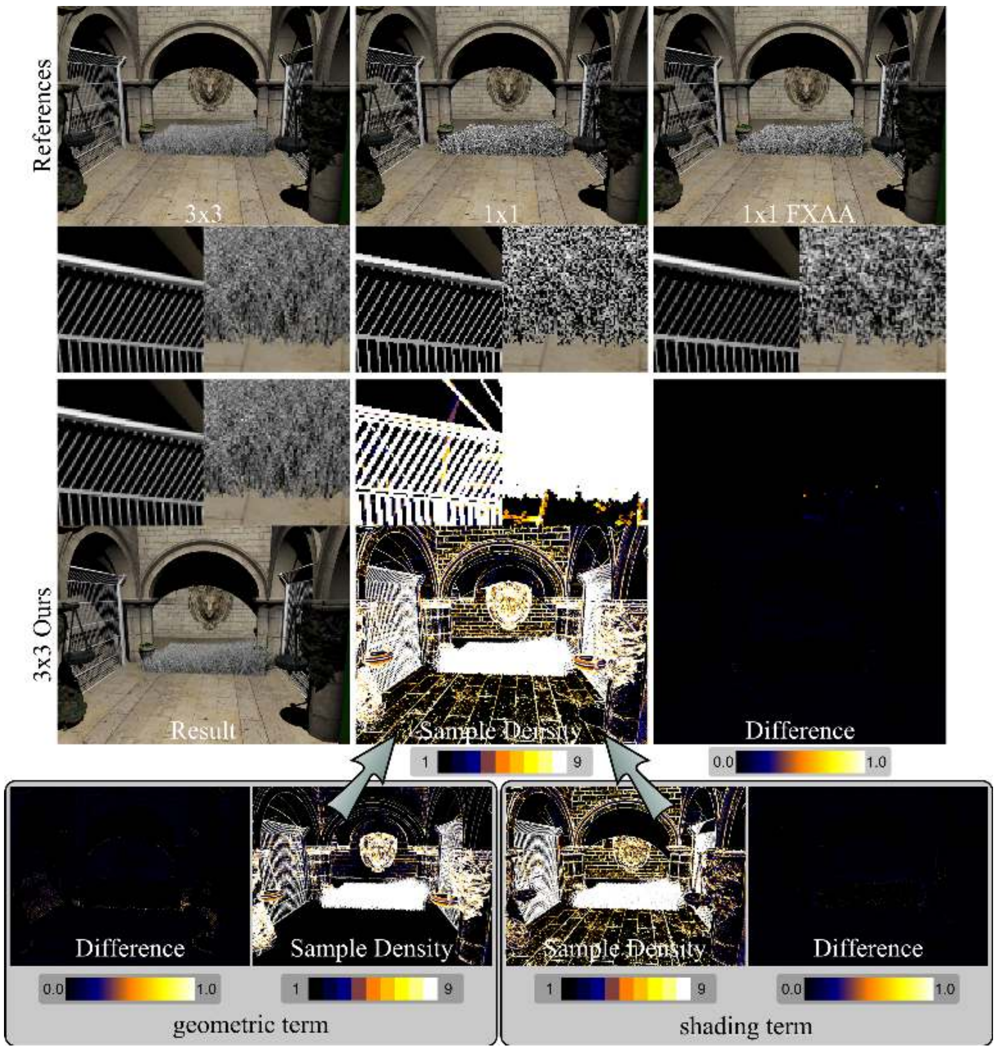

Figure 8. The importance of both the geometric and shading term; all difference images were computed with respect to the 3×3 supersampling reference solution. Top row left to right: 3×3 supersampling reference, deferred shading with a single sample, deferred shading with a single sample and FXAA post-processing (high quality settings). Middle row: our adaptive version, (final) sample density of our algorithm, difference image with respect to the 3×3 supersampling reference solution. Bottom row: sample density and difference for our geometric and shading term. Geometric features below the Nyquist frequency are not captured well by our shading term but are reduced using our geometric term. This is even more apparent when the camera is moving.

ENGEL, W., 2008. Diary of a Graphics Programmer: Light Pre-Pass Renderer. http://diaryofagraphicsprogrammer.blogspot.com/2008/03/light-pre-pass-renderer.html. 11

HERZOG, R., EISEMANN, E., MYSZKOWSKI, K., AND SEIDELM, H.-P. 2010. Spatio-

Temporal Upsampling on the GPU. In *Proc. Symposium on Interactive 3D Graphics and Games (I3D)*, ACM, Washington, DC, USA. 11

HOBEROCK, J., LU, V., JIA, Y., AND HART, J. 2009. Stream Compaction for Deferred Shading. In *Proc. High Performance Graphics*, ACM, New York, NY, USA, 173–180. 9

JIMENEZ, J., GUTIERREZ, D., YANG, J., RESHETOV, A., DEMOREUILLE, P., BERGHOFF, T., PERTHUIS, C., YU, H., MCGUIRE, M., LOTTES, T., ET AL. 2011. Filtering Approaches for Real-Time Anti-Aliasing. In *ACM SIGGRAPH 2011 Courses*, ACM, New York, NY, USA, 6:1–6:329. 3

KIRCHER, S., AND LAWRANCE, A. 2009. Inferred lighting: fast dynamic lighting and shadows for opaque and translucent objects. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, ACM, 39–45. 4

LAURITZEN, A. 2010. Deferred Rendering for Current and Future Rendering Pipelines. In *SIGGRAPH 2010 Course: Beyond Programmable Shading*, ACM, New York, NY, USA. 4, 11

LOTTES, T., 2009. FXAA. http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf. 3, 6

RESHETOV, A. 2009. Morphological Antialiasing. In *Proc. High Performance Graphics*, Eurographics, Aire-la-ville, Switzerland, 109–116. 3

SALVI, M., AND VIDIMČE, K. 2012. Surface Based Anti-Aliasing. In *ACM SIGGRAPH Symposium on Interactive 3D Rendering and Games, I3D*, ACM, New York, NY, USA, 159–164. 3

SWOBODA, M., 2009. Deferred Rendering in Frameranger. http://directtovideo.wordpress.com/2009/11/13/deferred-rendering-in-frameranger. 4

THIBIEROZ, N. 2009. Deferred Shading with Multisampling Anti-Aliasing in DirectX10. In *ShaderX7: Advanced Rendering Techniques*, Charles River Media, Hingham, MA, USA. 3

VALIENT, M. 2007. Deferred Rendering in Killzone 2. In *The Develop Conference and Expo*, http://www.dimension3.sk/2012/08/deferred-rendering-in-killzone-2/. 3, 6

YANG, L., SANDER, P., AND LAWRENCE, J. 2008. Geometry-Aware Framebuffer Level of Detail. In *Proceedings of the Nineteenth Eurographics Conference on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, vol. 27, 1183–1188. 3

YOUNG, P., 2007. Coverage-Sampled Antialiasing. http://developer.download.nvidia.com/assets/gamedev/docs/CSAA_Tutorial.pdf. 3

Author Contact Information

Matthias Holländer
Telecom ParisTech
Office C16
46, Rue Barrault
75013 Paris
hollaender@telecom-paristech.fr
http://perso.telecom-paristech.fr/~hollande/

Tamy Boubekeur
Telecom ParisTech
Office C13
46, Rue Barrault
75013 Paris
tamy.boubekeur@telecom-paristech.fr
http://perso.telecom-paristech.fr/~boubek/

Elmar Eisemann
TU Delft,
Computer Graphics & Visualization (Bldg. 36)
Mekelweg 4
2628 CD, Delft
E.Eisemann@tudelft.nl
http://graphics.tudelft.nl/~eisemann/

M. Holländer, T. Boubekeur, and E. Eisemann, Adaptive Supersampling for Deferred Anti-Aliasing, *Journal of Computer Graphics Techniques (JCGT)*, vol. 2, no. 1, 1–14, 2013
http://jcgt.org/published/0001/02/01/

Received:	2012-09-05		
Recommended:	2012-11-20	Corresponding Editor:	John Hable
Published:	2013-03-02	Editor-in-Chief:	Morgan McGuire

Updated 2013-10-15 to add a citation to Kircher and Lawrance's work. Thanks to Stephen Hill for the correction.