Surface Based Anti-Aliasing

Marco Salvi [*] Kiril Vidimče

Intel Corporation

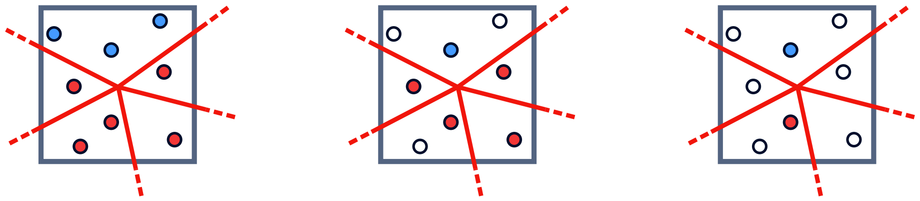

Figure 1: *From left to right: 8x super-sample anti-aliasing (SSAA), 8x multi-sample anti-aliasing (MSAA) and surface-based anti-aliasing (SBAA) with 8 visibility and 2 surface samples per pixel. The circles represent visibility samples, while the blue and red discs represent shading samples from two different surfaces. The four red primitives sharing the same vertex are part of the same foreground surface. Our MERGE2 algorithm exploits this configuration and shades only one sample for all four red primitives while reserving a second surface sample for the blue background surface. Unlike multi-sampling, SBAA based algorithms impose an upper bound on the number of captured, stored and shaded surfaces rather than primitives in each pixel, therefore significantly reducing storage and shading costs.*

Abstract

We present surface based anti-aliasing (SBAA), a new approach to real-time anti-aliasing for deferred renderers that improves the performance and lowers the memory requirements for anti-aliasing methods that sample sub-pixel visibility. We introduce a novel way of decoupling visibility determination from shading that, compared to previous multi-sampling based approaches, significantly reduces the number of samples stored and shaded per pixel. Unlike post-process anti-aliasing techniques used in conjunction with deferred renderers, SBAA correctly resolves visibility of sub-pixel features, minimizing spatial and temporal artifacts.

Keywords: anti-aliasing, deferred shading, multi-sampling

1 Introduction

Real-time applications based on deferred shading do not scale well with anti-aliasing methods that evaluate visibility at sub-pixel level since memory requirements and shading costs grow linearly with the number of per-pixel visibility samples. These limitations make it prohibitive to use deferred renders with the highest visibility rates currently supported on graphics hardware (e.g. 8 samples per pixel) and could entirely prevent using these popular software techniques on low power mobile devices or on future graphics processors supporting even higher visibility sampling rates.

The main contribution of this work is a novel way of decoupling visibility from shading that enables storing and shading a fixed number of samples per pixel. We do so by selecting and shading fragments that keep artifacts at a minimum and we demonstrate that our method can lower memory requirements by up to 65% while significantly reducing shading costs. Surface based anti-aliasing generates images that exhibit low levels of spatial and temporal aliasing. Moreover our algorithm is scalable and enables deferred shading to be used on future architectures that may support more visibility samples per pixel.

[*]e-mail: {marco.salvi, kiril.vidimce}@intel.com

2 Related Work

It is beyond the scope of this work to describe the numerous real-time anti-aliasing algorithms developed over the last few decades; we therefore limit our analysis to techniques most similar to ours.

Super-sampling anti-aliasing (SSAA) reduces aliasing artifacts by directly increasing the image sampling rate. In practice this can be done by either computing N color samples per pixel or by rendering the scene in multiple passes with different subpixel offsets and accumulating the result [Fuchs et al. 1985; Deering et al. 1988; Mammen 1989; Haeberli and Akeley 1990].

Because SSAA requires computing N samples per pixel, various methods have been developed to lower its shading cost. The defacto standard method supported by virtually all modern graphics processors and APIs is multi-sampling anti-aliasing (MSAA) [Akeley 1993]. This method reduces the cost of computing the final pixel color by decoupling visibility determination from shading and by limiting per-pixel shading to the rate of one sample per primitive while still evaluating visibility at full rate. Both SSAA and MSAA share the same storage requirements which grow linearly with the number of samples.

Coverage sampled anti-aliasing (CSAA) shades primitives once per pixel but unlike MSAA it decouples visibility and color data from coverage information [Young 2007]. The latter can be efficiently sampled at higher rates via hardware rasterization and stored in a compact form as bit masks, making it possible to approach the visual quality of high MSAA rates while lowering storage and shading costs. Since the final pixel color can depend on the geometric primitive submission order, CSAA can also be seen as a form of streaming, lossy color compression. Similarly to CSAA, the Z^3 algorithm [Jouppi and Chang 1999] also compresses color information by merging fragments as necessary in order to fit fragment data within a fixed amount of storage per pixel, while Lee et al. describe a modified A-buffer method that can merge fragments at list insertion time [Lee and Kim 2000]. We note that these three algorithms do not map well to deferred rendering pipelines because color data is required to guide the compression process. To reduce the cost of shading micropolygons in a GPU pipeline, Fatahalian et al. propose to merge pixel quads rasterized from adjacent primitives that have been generated from the same tessellated patch and that do not overlap in screen space [Fatahalian et al. 2010]. Quad-fragment merging is performed prior to shading and does not require color data, although it is applied to a small number of primitives buffered on-chip and it is therefore not directly applicable to deferred renderers. Moreover the merging scheme assumes that primitives are by definition part of the same surface making it infeasible to apply a similar technique to arbitrary (i.e. non-tessellated) geometry.

Several methods have been proposed to address the above limitations of using multi-sampling in conjunction with deferred renderers. Lightspeed decouples visibility samples from geometry samples (G-buffer) with an indirect frame-buffer which densely encodes variable-rate visibility information, enabling faster shading [Ragan-Kelley et al. 2007]. Initially developed for incremental relighting applications this method can be implemented in real-time on modern GPUs but it cannot alleviate MSAA memory requirements. Lauritzen significantly lowers shading costs by analyzing the multi-sampled G-buffer in order to discover planar features that can be shaded at pixel frequency but it does not address the often prohibitive amount of memory necessary to couple MSAA to deferred shading [Lauritzen 2010].

Image post-processing based anti-aliasing techniques such as morphological anti-aliasing [Reshetov 2009] can offer very high performance with low or no impact on memory requirements but they lack the robustness necessary to consistently generate images that do not exhibit significant spatial and temporal artifacts in the presence of sub-pixel features. For a survey of the most recent morphological and filtering based techniques see the course notes from Jimenez et al. [2011]. Lastly, sub-pixel reconstruction anti-aliasing significantly improves upon post-processing methods by evaluating visibility at super-resolution while limiting shading to one sample per pixel [Chajdas et al. 2011].

Surface based anti-aliasing (SBAA) is a technique that analyzes multi-sampled data such as depth, primitive ID, etc., to determine how many G-buffer samples need to be rendered, stored and shaded per pixel. SBAA efficiently decouples visibility from shading by aggregating point samples into larger to-be-shaded fragments and it is therefore similar in spirit to the work of Lauritzen [2010] and Fatahalian et al. [2010]. Unlike these techniques our work performs its analysis before rendering the G-buffer and can handle an arbitrary, but fixed a priori, number of G-buffer samples per pixel, thus reducing both storage and shading costs when compared to previous multi-sampling based methods. In particular our anti-aliasing scheme makes trade-offs similar to recent order-independent techniques like the k-buffer and adaptive transparency where image quality is exchanged for the benefits of fixed storage requirements per pixel [Bavoil et al. 2007; Salvi et al. 2011].

3 Algorithm

The starting point for our method is the standard multi-sampling anti-aliasing technique supported by modern GPUs. MSAA implementations typically compute the final color C of a pixel as sum of the color of the M fragments that cover it. The contribution of individual primitives is proportional to their fractional coverage w_i:

$$C = \sum_{i=1}^{M} w_i c_i \tag{1}$$

with $\sum w_i = 1$, where w_i is determined as the fraction of the N per-pixel visibility samples covered by the i_{th} geometric primitive. If M is equal to N, multi-sampling is equivalent to SSAA, although when $M < N$ this method effectively reduces the number of samples that need to be shaded at the cost of a less accurate reconstruction of the color of a primitive within a given pixel.

Software designed to decouple and defer shading by first rendering per-pixel geometric attributes to a G-buffer cannot easily take advantage of shading rate reduction provided by multi-sampling since at shading time the relation between geometry samples and the primitives they belong to is generally not known. Moreover the size of a typical G-buffer sample can be much larger than a color sample (e.g. 20-40 bytes) which can make the usage of deferred renderers at high resolutions and high multi-sampling rates impractical due to the very large storage and memory bandwidth requirements. For instance a 1080p image rendered with 8x MSAA and 32 byte G-buffer samples would require over 700 MBytes of graphics memory.

3.1 Surface Based Anti-Aliasing

We begin by establishing terminology. Renderers generate scenes that consist of a number of individual surfaces (or meshes) each of which consists of numerous primitives. Modern GPUs are designed around rasterization and rendering of triangles as the native primitive in the graphics pipeline.

Our method pushes multi-sampling one step further by observing that even when a pixel is covered by multiple primitives it is generally only covered by a very small number of distinct surfaces. In other words very often primitives covering a given pixel tend to be connected and to belong to the same surface. If the local curvature of a surface is low (e.g. close to a plane) then it is likely that its color within a pixel does not change significantly. In fact we note that texture sampling hardware automatically takes care of filtering out large color variations within a pixel introduced by arbitrary color textures, with the remaining higher frequency terms typically generated by lighting (e.g. specular terms).

Thus, we can further reduce the per-pixel shading rate by computing only one color sample per surface. The final pixel color is given by the sum of the contributions of K surface samples S_i:

$$C = \sum_{i=1}^{K} s_i c_i \tag{2}$$

with $\sum s_i = 1$, where s_i is the coverage associated to S_i. If we assign to each surface sample S_i a G-buffer sample and store only a small and fixed (e.g. 2 or 3) number of surfaces per pixel this approach also makes possible to significantly reduce the size of the frame-buffer for high visibility sampling rates, especially when used in conjunction with rendering techniques that have a high per sample storage cost, such as deferred shading. Figure 1 illustrates how our method operates with respect to SSAA and MSAA.

It is therefore possible to define a new class of surface based anti-aliasing algorithms for deferred shading which require the following steps:

1. Render opaque geometry via multi-sampling, capture depth, primitive IDs and other data

2. For each pixel analyze sample data, detect surfaces and output their primitive IDs and number of samples covered

3. Render opaque geometry via multi-sampling and capture G-buffer samples identified in the previous pass

4. Shade the G-buffer and output final anti-aliased image

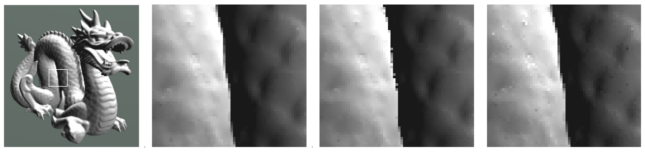

Figure 2: *Image quality comparison between TOP2 and MERGE2 coupled to 8x MSAA. From left to right: DRAGON scene and magnified details respectively rendered with our reference solution, TOP2 and MERGE2. The DRAGON mesh is extremely dense, generating an average of 6 primitives per pixel at 720p and it provides a clear failure case for the TOP algorithm which generates low quality results. On the other hand the MERGE algorithm can match the reference solution by only shading up to 2 surface samples per pixel.*

We note that a standard deferred renderer already requires rendering passes similar to steps 3 and 4, while some implementations also employs a so-called "z pre-pass" that can resemble step 1.

The key idea of our method lies in the per-pixel data analysis and surface detection performed in step 2. In particular we propose two simple and scalable surface detection algorithms that can discover surfaces by analyzing the sub-pixel data.

3.2 TOP Algorithm

As shown in Equation 1, computing the final color of a pixel may require accumulating the contribution of up to one primitive per visibility sample, although in most cases a primitive will cover multiple visibility samples per pixel, especially with high multi-sampling rates.

We can therefore define a simple mapping between a primitive and a surface and use primitive IDs to detect the primitives that cover the most samples in a given pixel. We call this algorithm TOPn, where n is the maximum number of per-pixel primitives and therefore G-buffer samples that we can detect and output.

Note that if a pixel is covered by at most n primitives, TOPn will generate the same results as multi-sampling, while in the remaining cases we can minimize the overall per pixel error by simply ignoring primitives with smaller coverage.

3.3 MERGE Algorithm

A typical failure case for TOP is exemplified by very dense geometry rendered with high multi-sampling rates. When the number of primitives covering a pixel converges to the number of visibility samples, no primitive tends to cover more samples than the others. As shown in Figure 2, selecting surfaces by choosing primitives with higher coverage doesn't yield optimal results.

When primitive coverage does not act as a good proxy to detect surfaces we need additional information to determine which samples can be mapped to a surface. To improve upon TOP we first select the primitive P with highest coverage and we then find the samples i whose normal $\hat{\mathbf{n}}_i$ is aligned to P by using the following test:

$$\hat{\mathbf{n}}_r \cdot \hat{\mathbf{n}}_i > cos(\alpha_\epsilon)$$

where $\hat{\mathbf{n}}_r$ is a reference normal drawn from a sample of P and α_ϵ is a small angle. While this simple test provides a new set of samples A that can be possibly identified as belonging to the same surface, it can also erroneously merge primitives which are aligned in space but that are not spatially correlated (e.g. very distant but spatially aligned background and foreground surfaces covering the same pixel).

To avoid false positives the MERGE algorithm determines whether the discrete depth distribution defined by A is not unimodal. In other words we want to determine if the depth distribution has more than one local maximum. To simplify computations we only detect one or two mode distributions. We do so by spatially subdividing the samples in A in three regions: two small and equally sized regions[1] at the extremities of the depth range defined by A and a third region at the center of it. If the center region is empty then all samples must be located inside the small regions at both ends of the depth range and thus we consider the distribution to be bimodal and samples in A are determined to not be part of the same surface. In this case MERGE behaves like TOP and outputs primitive IDs and sample count of the primitive with highest coverage. When the central region is not empty, the depth distribution is determined to be unimodal and we output the primitive ID of the reference normal $\hat{\mathbf{n}}_r$ and the number of samples in A.

In both cases we can generalize the MERGEn algorithm to any n number of surfaces by iteratively applying our algorithm n times only to the samples that have not been identified yet as part of some surface, until we either run out of samples or we have output the maximum number (n) of surface samples we are allowed to process.

4 Implementation

Our surface based anti-aliasing algorithms require a few simple modifications to a standard deferred render.

The DirectX 11 based implementation of TOPn first renders primitive IDs in a z pre-pass that may or may not be already part of a deferred rendering pipeline. A subsequent analysis pass loads per-sample primitive IDs for each pixel onto the GPU and reconstructs the sample count of each primitive. We do so by using the least significant bits of a primitive ID to address, increment and read back a per-primitive counter stored in on-chip memory. Due to the limited size of this memory we found that using only 7 bits guarantees good performance while minimizing artifacts due to address aliasing. Moreover since the most significant bits of the primitive IDs are not needed, this method enables rendering them to storage and bandwidth-efficient render targets (i.e. 8 bit per sample). At each iteration we discover a new surface and we output it to a 2 byte structure:

```
struct {
    unsigned char primitiveID;
    unsigned char sampleCount;
} SurfaceData;
```

for a maximum of n surfaces per pixel.

When rendering the G-buffer we bind the multi-sampled depth buffer and primitive ID buffer that got generated during step 1. We automatically process only fragments that match the ones already in the multi-sampled depth buffer and discard everything else. We then check if the primitive ID of the current fragment matches one of the surface primitive IDs computed in the analysis pass; if it does we output a new G-buffer sample for it, otherwise we discard the fragment. Note that in this rendering pass we do not bind a multi-sampled G-buffer as a render target; instead we shade at pixel frequency and output per-pixel data to a writable memory buffer. In the G-buffer shading pass we shade up to n G-buffer samples per pixel depending on how many distinct surfaces were found by the

[1]While the size of these regions is arbitrary we have found that the optimal size for best image quality is $\frac{1}{4}$ of the whole depth range.

	4x MSAA		8x MSAA			
Scene	TOP2	MERGE2	TOP2	TOP3	MERGE2	MERGE3
SPONZA	0.62	1.5	1.0	1.1	2.3	2.4
GRASS	0.64	1.6	1.0	1.1	2.4	2.6
POWERPLANT	0.64	1.6	1.0	1.1	2.4	2.5
DRAGON	0.60	1.2	0.92	1.0	2.2	2.5

Table 1: *SBAA analysis pass execution time in milliseconds for different scenes rendered at 720p as a function of multi-sampling rate, anti-aliasing method and number of G-Buffer samples.*

sample analysis shader. Finally the pixel color is determined by using Equation 2 where the fractional surface coverage s_i is computed as the ratio of the number of visibility samples covered by the i_{th} surface and the total number of samples covered by all detected surfaces.

To implement MERGEn we simply extend the z pre-pass to render per sample normals to a 4 byte representation and we modify the analysis pass to include the sample alignment and the depth unimodality tests discussed in Section 3.3. Note that the sample alignment test does not require very accurate normals and thus even a more compact representation can be used, e.g., one that combines normals and primitive IDs. This would further reduce storage and memory bandwidth costs.

5 Results

We analyze various configurations of TOP and MERGE supporting two G-Buffer samples per pixel with 4x MSAA and two or three G-buffer samples with 8x MSAA. Our algorithms are compared to a DirectX 11 based reference implementation of the deferred shading method recently introduced by Lauritzen [2010] and previously discussed in Section 2. For our tests we set α_ϵ to $\frac{\pi}{16}$.

To accurately probe the robustness of our surface based techniques we evaluate performance and image quality on scenes representative of modern workloads, selected to stress characteristics such as geometry density and complexity that can significantly impact the final image. All results are gathered at the resolution of 1280 x 720 on an ATI Radeon HD 5870 GPU with an Intel Core 2 quad-core CPU running Windows 7 64-bit.

5.1 Image Quality Analysis

In Figure 3 we qualitatively compare the reference solution with and without multi-sampling to images generated with our TOP and MERGE algorithms. For each scene we select an MSAA mode and the two anti-aliasing methods that yield the best image quality. If two or more techniques generate very similar images we choose the ones with the lowest computational and memory costs.

5.1.1 SPONZA Scene

This is an indoor scene with many smooth or even flat surfaces that do not form complex geometrical structures (floors, walls, arches, vases, etc.). In scenes like SPONZA, TOP2 generates results practically indistinguishable from 4x and 8x MSAA, therefore it is not necessary to resort to more complex algorithms which do not yield any image quality improvement.

5.1.2 GRASS Scene

This scene exhibits dense and high frequency geometry. Individual grass blades are modeled as single triangles defining well separated and distinct surfaces in space. Therefore in most cases MERGE won't be able to fuse fragments together and as discussed in Section 3.3 its behavior will automatically fall back to that of TOP with both methods generating very similar results. Still images rendered with TOP2 match very closely the reference implementation, although temporal aliasing is still visible to some degree when the camera is slowly moving through the scene. This issue can be addressed by adding a third G-Buffer sample per pixel. In particular the TOP3 configuration exhibits very similar amounts of temporal aliasing perceivable with standard 8x MSAA while significantly lowering rendering time and memory requirements.

5.1.3 POWERPLANT Scene

Similarly to the previous scene the POWERPLANT scene contains thin and complex geometrical structures that when rendered with 8x MSAA generate pixels covered by many primitives. Although unlike the GRASS scene in this case individual primitives are more likely to be connected and to define distinct surfaces. This effect is clearly visible in the magnified area in Figure 3 where MERGE2 can detect these surface and improve over TOP2 by generating an image very close to the reference method. It also exhibits very little to no temporal aliasing under slow camera movements.

5.2 Performance Analysis

The cost of analyzing per-sample data to generate IDs and coverage for the samples that require shading goes from 0.6 ms for TOP2 with 4x MSAA to 2.6 ms for MERGE3 and 8x MSAA (see Table 1) and it generally varies little across the tested scenes.

As mentioned in Section 4 there are additional rendering costs related to adding a z pre-pass to generate per-sample primitive IDs and/or normals. It is difficult to determine any performance pattern from this rendering pass alone as it can change significantly across scenes and can have positive repercussions on the subsequent passes due to the increased number of fragments which are more likely to be rejected before shading. We prefer instead to quote performance numbers for the entire frame to understand if despite these extra rendering costs our algorithms can still have a positive impact on performance. In particular we measure the rendering time with two different shading profiles: a simple profile that evaluates only a single directional light with diffuse and specular terms and a complex profile that shades up to 16 complex lights per G-buffer sample.

As evident from Table 2 our methods can significantly reduce the rendering time when compared to the reference solution running with 8x MSAA. Performance improvements are due to the often vastly reduced cost of shading given that we can strongly constrain the number of unique per-pixel shader invocations. We show the reduction of total shaded samples in Table 3. Unsurprisingly, scenes with very high-frequency geometry like GRASS benefit the most from our method reporting an up to 40% reduction in the total rendering time for TOP2 coupled to the complex lighting profile. In some cases SBAA can improve performance even when it shades moderately more samples than the reference implementation (e.g. the DRAGON scene rendered with TOPn and 8x MSAA). We attribute this to the fact that our method shades from one to three samples per pixel, exhibiting a more regular and SIMD-friendly behavior then Lauritzen [2010] that shades either one or eight samples per pixel. We also note that SBAA can negatively affect performance with 4x MSAA, e.g., when rendering the GRASS scene. In this particular case our method does not sufficiently reduce the shading cost to offset the additional computations it requires.

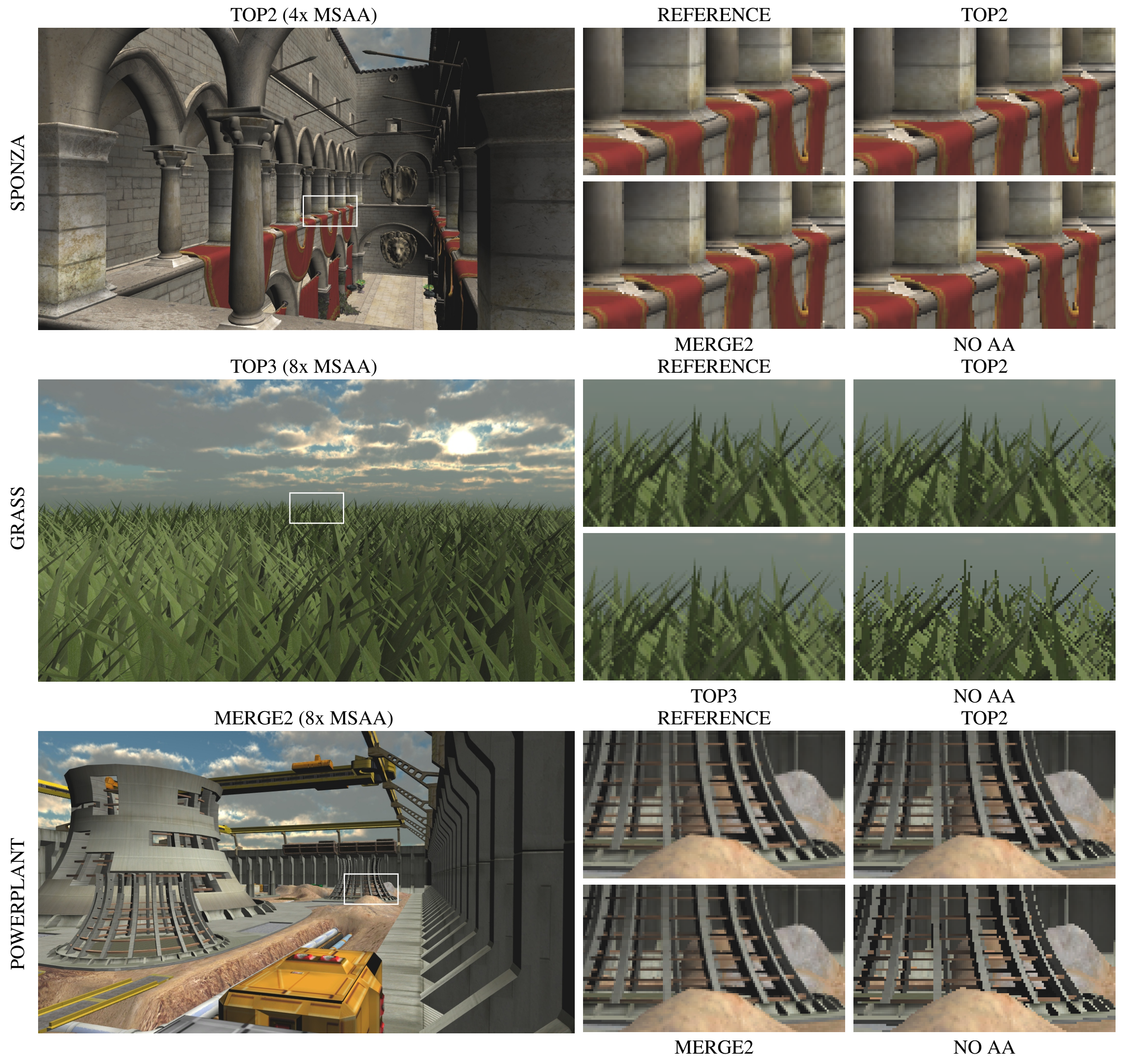

Figure 3: *We compare three scenes rendered with the Lauritzen [2010] 4x or 8x MSAA reference implementation to a few variations of the TOP and MERGE algorithms. Please note how our methods can generate images almost indistinguishable from the reference multi-sampling based solution*

	4x MSAA						8x MSAA									
Scene	REFER (ms)		TOP2 (%)		MERGE2 (%)		REFER (ms)		TOP2 (%)		TOP3 (%)		MERGE2 (%)		MERGE3 (%)	
SPONZA	6.0	8.7	20	32.2	38.3	42.5	10.5	13.0	-26.7	-6.9	-14.3	11.5	-11.4	-0.8	-0.1	14.6
GRASS	11.0	14.5	-13.6	-1.4	-0.1	10.3	16.9	21.2	-39.0	-28.3	-34.3	-17.5	-37.9	-27.8	-20.7	-7.5
POWERPLANT	5.8	8.5	20.7	34.1	41.4	41.2	10.4	13.3	-26.0	-9.0	-14.4	8.3	-11.5	-3.0	0.1	12.0
DRAGON	7.8	8.6	20.5	19.8	34.6	32.6	11.8	12.8	-16.1	-14.8	-11.0	- 7.0	-2.5	-3.1	2.5	3.9
MBytes	144.1		80.9 (-44%)		94.9 (-34%)		284.8		98.4 (-65%)		128.3 (-54%)		126.6 (-55%)		156.4 (-45%)	

Table 2: *Scene statistics. We report the frame time in milliseconds with simple and complex lighting (respectively left and right sub-columns) and percent variation in rendering time and memory requirements for the TOP and MERGE algorithms*

	4x MSAA			8x MSAA				
Scene	REFER (#)	TOP2	MERGE2	REFER (#)	TOP2	TOP3	MERGE2	MERGE3
SPONZA	1,041,972	3.14%	-5.25%	1,265,839	-13.46%	-9.25%	-23.22%	-22.16%
GRASS	1,000,996	5.38%	4.92%	1,987,385	-45.56%	-43.73%	-45.96%	-44.27%
POWERPLANT	1,114,806	-4.75%	-10.85%	1,447,662	-25.12%	-22.64%	-31.34%	-30.55%
DRAGON	944,970	9.13%	2.08%	999,965	3.37%	13.06%	-6.98%	-6.73%

Table 3: *Shading efficiency. We report number of total shaded samples for the reference Lauritzen [2010] rendering methods running at 4x and 8x MSAA and the percent variations in number of shaded samples for the TOP and MERGE algorithms.*

5.3 Memory analysis

Compared to the reference implementation our anti-aliasing methods can reduce memory needed for 4x and 8x MSAA respectively by up to 44% and 65% (see Table 2), despite requiring additional storage for data typically not necessary for deferred rendering (e.g. primitive IDs). By switching from 8x MSAA to TOP2 an application running at 1080p resolution can save ˜500 MBytes of graphics memory, which can be employed to store more textures and models.

6 Conclusion

We have presented a new class of anti-aliasing algorithms for deferred shading that can significantly improve performance and reduce memory requirements when compared to previous MSAA-based solutions.

Our methods are scalable and likely to be even more efficient when employed with high multi-sampling rates. In fact we expect SBAA-based techniques to be key in making deferred shading performance and memory requirements scale to 16x or even higher multi-sampling rates on future GPUs while still taking advantage of hardware-accelerated and power-efficient MSAA support.

In the future we would like to investigate improved surface detection schemes and architectural changes aimed at providing more efficient support for surface based anti-aliasing techniques.

Acknowledgments

We thank Sungkil Lee for the GRASS scene. Thanks to the the entire Advanced Rendering Technology team, Matt Pharr, Doug Binks and Larry Seiler at Intel for their contributions and support.

References

AKELEY, K. 1993. Reality engine graphics. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '93, 109–116.

BAVOIL, L., CALLAHAN, S. P., LEFOHN, A., COMBA, J. A. L. D., AND SILVA, C. T. 2007. Multi-fragment effects on the gpu using the k-buffer. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '07, 97–104.

CHAJDAS, M., MCGUIRE, M., AND LUEBKE, D. 2011. Subpixel reconstruction antialiasing. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press.

DEERING, M., WINNER, S., SCHEDIWY, B., DUFFY, C., AND HUNT, N. 1988. The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, 21–30.

FATAHALIAN, K., BOULOS, S., HEGARTY, J., AKELEY, K., MARK, W. R., MORETON, H., AND HANRAHAN, P. 2010. Reducing shading on gpus using quad-fragment merging. In *ACM SIGGRAPH 2010 papers*, ACM, New York, NY, USA, SIGGRAPH '10, 67:1–67:8.

FUCHS, H., GOLDFEATHER, J., HULTQUIST, J. P., SPACH, S., AUSTIN, J. D., BROOKS, JR., F. P., EYLES, J. G., AND POULTON, J. 1985. Fast Spheres, Shadows, Textures, Transparencies, and Imgage Enhancements in Pixel-Planes. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, ACM, vol. 19, 111–120.

HAEBERLI, P., AND AKELEY, K. 1990. The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, ACM, vol. 24, 309–318.

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., AND SOUSA, T. 2011. Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH Courses*.

JOUPPI, N. P., AND CHANG, C.-F. 1999. Z3: an economical hardware technique for high-quality antialiasing and transparency. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM, New York, NY, USA, HWWS '99, 85–93.

LAURITZEN, A. 2010. Deferred rendering for current and future rendering pipelines. Beyond Programmable Shading course, SIGGRAPH 2010. `http://bps10.idav.ucdavis.edu/`.

LEE, J.-A., AND KIM, L.-S. 2000. Single-pass full-screen hardware accelerated antialiasing. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM, New York, NY, USA, HWWS '00, 67–75.

MAMMEN, A. 1989. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl. 9*, 43–55.

RAGAN-KELLEY, J., KILPATRICK, C., SMITH, B. W., EPPS, D., GREEN, P., HERY, C., AND DURAND, F. 2007. The lightspeed automatic interactive lighting preview system. In *ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, SIGGRAPH '07.

RESHETOV, A. 2009. Morphological antialiasing. In *Proceedings of the 2009 ACM Symposium on High Performance Graphics*.

SALVI, M., MONTGOMERY, J., AND LEFOHN, A. 2011. Adaptive transparency. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, ACM, New York, NY, USA, HPG '11, 119–126.

YOUNG, P. 2007. Coverage sampled anti-aliasing. Tech. rep., NVIDIA Corporation.