Deferred Adaptive Compute Shading

Ian Mallett
University of Utah
ian@geometrian.com

Cem Yuksel
University of Utah
cem@cemyuksel.com

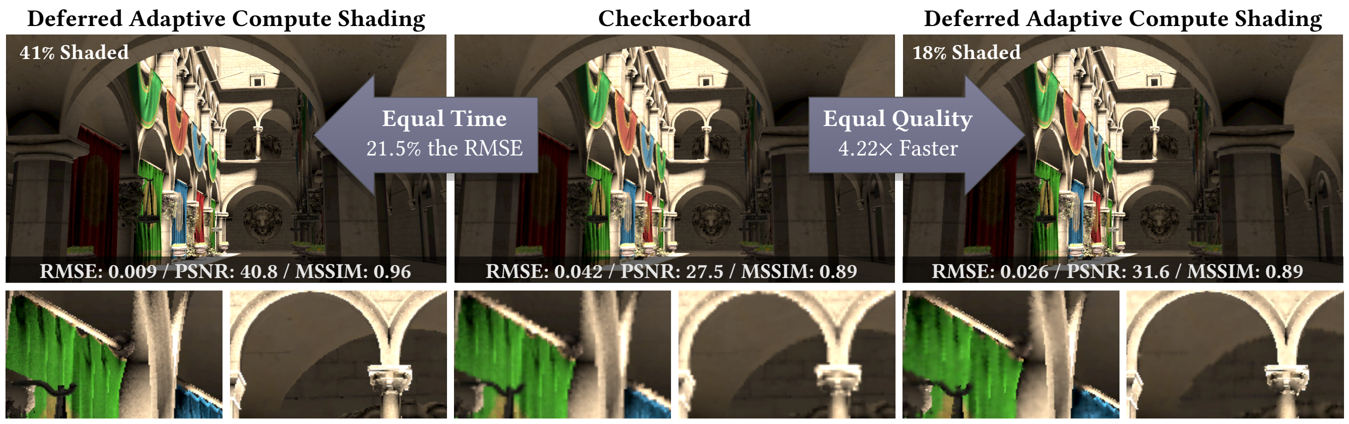
Deferred Adaptive Compute Shading
Checkerboard
Deferred Adaptive Compute Shading
41% Shaded
18% Shaded
Equal Time
21.5% the RMSE
Equal Quality
4.22× Faster
RMSE: 0.009 / PSNR: 40.8 / MSSIM: 0.96
RMSE: 0.042 / PSNR: 27.5 / MSSIM: 0.89
RMSE: 0.026 / PSNR: 31.6 / MSSIM: 0.89

Figure 1: *Comparison of Deferred Adaptive Compute Shading (DACS) to typical checkerboard methods.* ***Left:*** *given equal time, DACS produces only about a fifth the per-pixel error.* ***Right:*** *for equal MSSIM quality, DACS produces a result well over four times faster. Although numerically equal, the error in DACS is allocated into smoothly varying regions while silhouette edges are shaded at the full rate, resulting in a perceptually superior image (see zoomed views). Checkerboard rendering, which is non-adaptive, cannot do this.*

ABSTRACT

A primary advantage of deferred shading is eliminating wasted shading operations due to overdraw. We present a new algorithm that we call *Deferred Adaptive Compute Shading*, for providing further reduction in shading computations. Our method hierarchically shades the image while reducing the number of required shading operations to below one shading computation per pixel on average. We determine whether to shade a pixel or approximate it using previously shaded pixels around it, based on an estimate of the image variance at the pixel location. The algorithm is designed to dynamically reconfigure itself to achieve optimal warp coherence and measurable performance gain. We extensively evaluate our algorithm, demonstrating that it produces high-quality results and is robust and highly scalable while providing significant performance improvements in complex scenes.

CCS CONCEPTS

• **Computing methodologies → Rasterization**;

KEYWORDS

Deferred shading, adaptive shading, compute shader

ACM Reference Format:
Ian Mallett and Cem Yuksel. 2018. Deferred Adaptive Compute Shading. In *HPG '18: High-Performance Graphics, August 10–12, 2018, Vancouver, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3231578.3232160

1 INTRODUCTION

In an ordinary forward rasterizer, it is likely that fragments from some triangles that are ultimately behind other triangles (and therefore invisible) will be drawn first. Among other advantages, deferred shading [Saito and Takahashi 1990] provides a mechanism for eliminating this wasted shading cost, executing the (potentially expensive) fragment shader only once per pixel sample, avoiding additional fragment shading invocations due to overdrawing.

We propose reducing the shading rate further using a particular adaptive subdivision scheme. Our method determines whether to shade a pixel or else approximate its color using previously shaded pixels nearby. This way, we can dramatically reduce the number of shading invocations for parts of the image that have low estimated variance (e.g. regions of similar color), while preserving visual detail in high-variance areas (such as detailed textures or discontinuities in illumination or depth). This is particularly relevant with high-resolution displays that cause a considerable amount of per-pixel shading work [Clarberg et al. 2013]. We also describe a compute shader implementation that achieves full SIMD utilization and no additional divergence.

2 RELATED WORK

As graphics moves toward more-complex and computationally expensive shading algorithms involving realistic, physically based materials and a large number of light sources, deferred shading

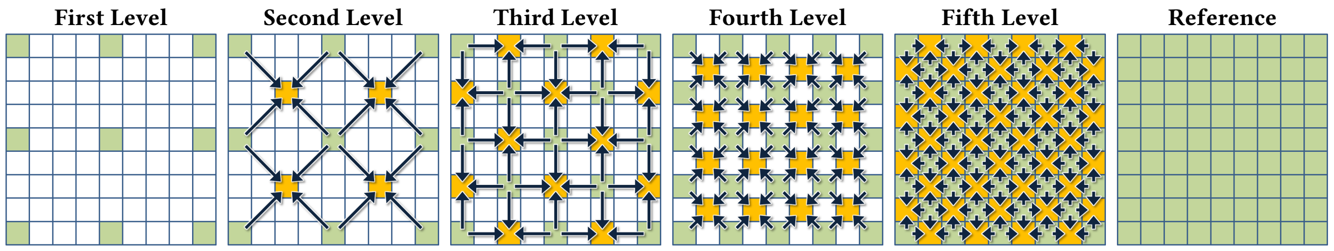

Figure 2: *Subdivision levels of our algorithm for an initial grid step size of* 4. *For each additional pixel added in each level (orange), we consider "neighbor" pixels from the previous level, as indicated by the arrows. Estimating the framebuffer's local variance from the G-buffer and these neighbor pixels, we decide whether to interpolate the new pixel from the neighbors, or else to shade it.*

[Saito and Takahashi 1990] has become nearly ubiquitous.

Decoupled shading [Ragan-Kelley et al. 2011] generalizes multisample antialiasing for separating shading rate from visibility. This can be implemented as a cache that simply reuses shading samples for "similar" visibility samples and can be implemented efficiently on the GPU [Liktor and Dachsbacher 2012] or speculative architectures [Clarberg et al. 2014, 2013]. However, there is a variety of work that separates shading and visibility for various purposes, and so can be considered forms of decoupled shading [Crassin et al. 2015; Kerzner and Salvi 2014]. Ideally, undersampling should be adaptive to the scene complexity. He et al. [2014] proposed a scheme that shades at a coarse rate, refining via scene information, such as known shadow edges. This approach only works for speculative hardware, and cannot be implemented on current GPUs. Vaidyanathan et al. [2014] described a speculative, forward-shading architecture that invokes the fragment shader at a coarse granularity, and then scatters the shading result to pixels while considering visibility samples. A different approach is to distribute sample locations in a perceptually-motivated fashion [Stengel et al. 2016], but a practical implementation on modern hardware is not described.

Recently, developers in different game studios have explored *checkerboard rendering* schemes for achieving various effects like antialiasing and temporal coherency [Mansouri 2016; Wihlidal 2017]. The checkerboard variant of Vlachos [2016] shares the same goal as our approach: undersampling the shading rate by skipping shading for some pixels. The skipped pixels are later filled in using a combination reconstruction filter and blur, implemented concurrently using optimized weighted samples. While this can be an effective approach, our tests show that our deferred adaptive compute shading algorithm can provide superior quality and performance.

3 ALGORITHM

Our Deferred Adaptive Compute Shading (DACS) algorithm partitions pixels in the framebuffer into multiple *subdivision levels*, which are progressively-denser subsets of pixels, as shown in Figure 2. In this example, the first level includes only every fourth pixel in a rectangular grid. Our processing begins with shading every pixel in the first level. Then, the pixels of the remaining levels are processed in order, starting with the second level. While processing these levels, we first estimate the variance by comparing the values of the four nearby pixels that belong to a previous level. If the four neighboring pixels are similar, according to a user-defined *similarity criterion*[1], we estimate the pixel value as the average of its neighbors; otherwise, we shade the pixel. This process continues hierarchically through the levels until all pixels have been assigned values. While the exact number of levels is variable we find that the initial step size of 4 shown in Figure 2 works well, and provides (potentially) up to $\approx 94\%$ reduction in shading. This particular selection order is inspired by the adaptive subdivision sampler of V-Ray [Chaos Group 2015]. The same fractal pattern was also used other ray tracing applications [Steinberger et al. 2012].

While it is easy to implement our algorithm naïvely, achieving effective warp utilization is non-trivial. Rasterized pixels are by-nature contiguous, and the simple approach of branching to shade or interpolate each pixel would create warp divergence. At a high level, our compute shader implementation separates the problem into two operations: *searching* for pixels that require shading by testing the shading criterion and *shading* the pixels so identified.

Warps begin in the search phase, in which each warp atomically increments a global pixel counter by k, the warp width. This corresponds to assigning a set of k pixels in the framebuffer to the warp. The warp then evaluates the shading criterion at each of these pixels. If the pixel can be interpolated from its neighbors, the interpolated result is stored immediately. If the pixel must be shaded, its coordinate is instead enqueued into the warp's *shading buffer*, a warp-local ring buffer of size $2k-1$. When the warp accumulates at least k pixel locations in this shading buffer, the whole warp switches to the shading phase. In the shading phase, a warp simply pops k pixels from its buffer, shades each, and then returns back to search mode. Because shading is deferred through the buffer, warps execute fully filled.

4 RESULTS

The Gradient, Step Functions, and Sinusoid tests in Figure 4 demonstrate low and high spatial frequencies at various angles. Low frequencies are typical of soft shadows and many shaded regions, such as walls or sky, while high frequencies are characteristic of texture features and depth-discontinuities.

We have also tested our algorithm using frames generated by Unreal Engine 4 [Epic Games 2014], with representative frames shown in Figure 3 along with quantitative error statistics in Table 1.

[1]Unless specified, in our results we shade a pixel if its neighbors have different material IDs or if the pre-gamma (i.e., lRGB) pixel values of the neighbors have a variance above a given threshold, which can be adjusted to vary quality. One could of course leverage additional information in the G-buffer, particularly depth and normals, to achieve potentially better results.

Table 1: *Quantitative Accuracy Comparisons of Deferred Adaptive Compute Shading in Simulated Unreal Engine 4 Scenes*

Scene Name	20% **Shading Rate**			50% **Shading Rate**			80% **Shading Rate**		
	RMSE	PSNR	MSSIM	RMSE	PSNR	MSSIM	RMSE	PSNR	MSSIM
Kite Demo Landscape	0.013434	37.450223	0.917924	0.006175	44.213449	0.976649	0.001992	54.152261	0.996919
Kite Demo Forest	0.014833	36.585668	0.933676	0.007348	42.700667	0.984564	0.003379	49.513014	0.992994
Xoio Berlin Flat	0.014190	37.059629	0.928814	0.006144	44.298021	0.974983	0.003052	50.316580	0.992241
Elvish Citadel	0.015064	36.508011	0.915203	0.007673	42.307247	0.966405	0.002751	51.219504	0.994643
Elemental Ice	0.011527	38.987657	0.948483	0.004451	47.111801	0.988349	0.001417	56.986239	0.998347
Elemental Lava	0.017941	34.989632	0.816162	0.010482	40.192873	0.910682	0.005681	46.208439	0.969872
Elemental Fire	0.007222	42.839877	0.978908	0.003622	49.077806	0.990305	0.002151	53.685692	0.996294

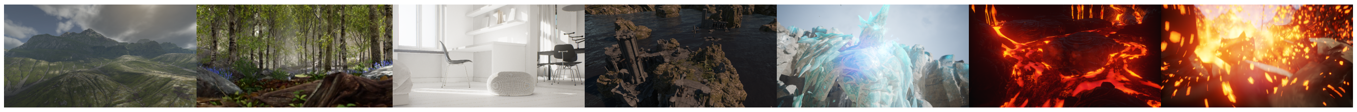

Figure 3: *Timing and representative reference frames for Unreal Engine 4 demo scenes on which we evaluated our algorithm. From left to right:* Kite Demo Landscape, Kite Demo Forest, Xoio Berlin Flat, Elvish Citadel, Elemental Ice, Elemental Lava, Elemental Fire.

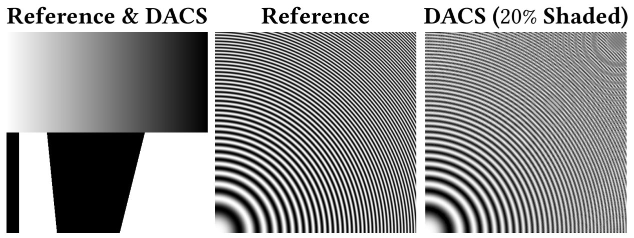

Figure 4: *Synthetic examples of spatial frequency.* ***Left:*** *The low frequencies in* Gradient *and infinite frequencies in* Step Functions *are reconstructed perfectly by DACS, even at low shading rates.* ***Center****: The* Sinusoid *example has many frequencies at all angles.* ***Right****: Reproduction of this texture using DACS. Note: this example may appear aliased or colored due to subpixel or perceptual issues.*

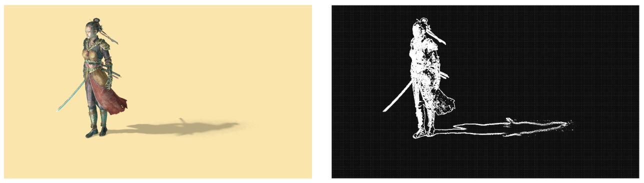

Figure 5: Nyra *model containing high-resolution textures and high geometric detail, rendered using diffuse image-based lighting and soft shadows [Fernando 2005]. Here, DACS shades* 11.0% *of pixels, affording a* 2.76× *speedup while maintaining a MSSIM of* 0.993.

Although the source frames from the game engine include aliasing artifacts and flickering, which are not improved by our adaptive shading approach, we see that the results produced by our algorithm are virtually indistinguishable from the reference in most cases.

The Crytek Sponza scene in Figure 1 compares our results to checkerboard rendering optimized for undersampling, as described by Vlachos [2016]: a stencil pattern of 2×2 pixels is rendered before the shading pass. This causes only alternating 2×2 quads to be emitted during fine raster, leading to half the shading cost. The missing pixels are then filled in by a combination reconstruction/blur filter, implemented using optimized texture samples. When we adjust the shading criterion such that DACS provides the same render time, we find that DACS has only 21.5% the RMSE as checkerboard rendering does. Meanwhile, when the shading criterion is adjusted to produce results with similar mean structural similarity (MSSIM) [Wang et al. 2004] values as checkerboard rendering, DACS produces its result 4.22× faster. We have observed a similar trend in all other scenes we tested.

Note that the qualitative nature of the error produced by DACS versus checkerboard rendering is very different. Checkerboard rendering is not adaptive and operates on the granularity of 2×2 quads. Thus, it has difficulty reconstructing fine silhouette edges and details. By contrast, DACS can reproduce fine edges, and most of its error manifests as overblurring in low-frequency regions.

The Nyra character scene (Figure 5) uses an 8K×8K shadow map. We test different numbers of texture look-up operations used by the PCSS algorithm for the blocker search phase, which generates distant, nonlocal, stochastic texture accesses, versus the shadow filtering phase, which produces more-localized texture accesses. We vary the number of texture look-up operations for both of these phases and compare the relative performance of DACS versus standard deferred shading in Figure 7. Obviously, increasing the number of texture look-up operations increases the render time in both algorithms. However, interestingly, the render time increases by a *smaller* amount with DACS, thereby resulting in a *larger speedup* factor as sampling complexity increases. We believe this is because in DACS, the reduced number of texture accesses overall increases texture cache performance, despite the added incoherency introduced by shading discontiguous pixels. This in turn leads to less penalty for texture accesses. This means that *as the number of texture lookups increases, DACS affords ever-larger performance advantages over ordinary deferred shading.*

Although thin textural detail is difficult, thin geometric detail is easier to handle. In the Windmill scene (Figure 6), we show an example of DACS handling very-thin geometry. Because the shading criterion used in DACS checks material ID, thin features are not

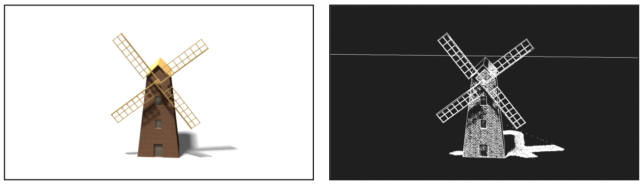

Figure 6: *WINDMILL scene demonstrating that thin geometry can be captured by means of a shading criterion that considers additional factors, like material ID.*

undersampled; DACS captures the thin features in the windmill's arms, even though they are often much finer than the initial shading levels. Aliasing only occurs when the object is small enough that the G-buffer cannot resolve it. For this view, the MSSIM is > 0.999 and the performance improvement is $1.90\times$.

5 DISCUSSION AND FUTURE WORK

The performance improvement of our method is correlated with its reduction in shading operations. In our tests we have used a very simple shading criterion, usually based on a simple comparison of material IDs and color variance. We found that this works well in practice, but a more-principled and/or perception-based heuristic could possibly provide better results.

While DACS provides an efficient implementation on current GPUs with high SIMD utilization, reordering the shading computation affects texture cache performance and the compute shader implementation leads to some overhead. In our tests we have observed that the savings in computation outweigh the overhead for scenes with high shading complexity and low shading rates.

By varying the shading criterion's threshold dynamically, it is possible to achieve a fine-grained performance/energy vs. render-quality tradeoff in realtime. This opens intriguing possibilities for future real-time graphics applications. For example, the shading criterion could vary temporally, based on the measured render time of the previous frame. Thus, the shading rate could be varied dynamically to achieve a constant framerate, allowing more control than with current LoD schemes. Also, render quality could be reduced to alleviate power consumption on mobile devices as energy or thermal considerations come into effect.

6 CONCLUSION

We have presented the Deferred Adaptive Compute Shading algorithm, an adaptive undersampling method that significantly reduces shading complexity while preserving high quality in practical scenarios. Overall, our results show that our approach can significantly reduce the amount of shading computation required to shade scenes, and this is borne out in a variety of scenes containing high geometric, textural, and shading detail, such as one would find in a realistic production environment.

ACKNOWLEDGMENTS

We thank Amit Prakash for his help with early data collection, Paul Tosca [2014], Morgan McGuire [2011], and Epic Games and XOIO [2014] for test models/scenes. This work was supported in part by NSF Grant #1409129.

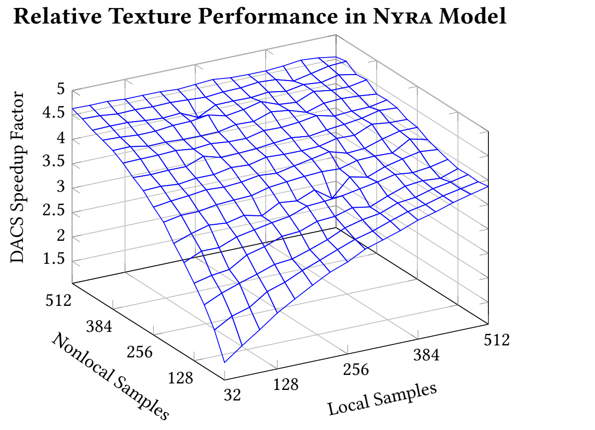

Figure 7: *Plot of relative performance of DACS compared to standard deferred shading, as a function of shadowmap "local" and "nonlocal" texture accesses for shadow computation with PCSS [Fernando 2005].*

REFERENCES

Chaos Group. 2015. V-Ray. http://docs.chaosgroup.com/pages/viewpage.action?pageId=7897184.

Petrik Clarberg, Robert Toth, Jon Hasselgren, Jim Nilsson, and Tomas Akenine-Möller. 2014. AMFS: adaptive multi-frequency shading for future graphics processors. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 141.

Petrik Clarberg, Robert Toth, and Jacob Munkberg. 2013. A Sort-based Deferred Shading Architecture for Decoupled Sampling. *ACM Trans. Graph.* 32, 4, Article 141 (July 2013), 10 pages.

Cyril Crassin, Morgan McGuire, Kayvon Fatahalian, and Aaron Lefohn. 2015. Aggregate G-buffer Anti-aliasing. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games (i3D '15).* 109–119.

Epic Games. 2014. Unreal Engine 4. http://www.unrealengine.com/.

Randima Fernando. 2005. Percentage-closer Soft Shadows. In *ACM SIGGRAPH 2005 Sketches (SIGGRAPH '05).* ACM, New York, NY, USA, Article 35.

Yong He, Yan Gu, and Kayvon Fatahalian. 2014. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Transactions on Graphics* 33, 4 (2014), Article–142.

Ethan Kerzner and Marco Salvi. 2014. Streaming G-Buffer Compression for Multi-Sample Anti-Aliasing. In *Proceedings of High Performance Graphics.* Eurographics Association, 1–7.

Gábor Liktor and Carsten Dachsbacher. 2012. Decoupled Deferred Shading for Hardware Rasterization. In *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.* 143–150.

Jalal Eddine El Mansouri. 2016. Rendering 'Rainbow Six | Siege'. http://www.gdcvault.com/play/1022990/Rendering-Rainbow-Six-Siege GDC.

Morgan McGuire. 2011. Computer Graphics Archive. http://graphics.cs.williams.edu/data.

Jonathan Ragan-Kelley, Jaakko Lehtinen, Jiawen Chen, Michael Doggett, and Frédo Durand. 2011. Decoupled Sampling for Graphics Pipelines. *ACM Trans. Graph.* 30, 3, Article 17 (May 2011), 17 pages.

Takafumi Saito and Tokiichiro Takahashi. 1990. Comprehensible Rendering of 3-D Shapes. *SIGGRAPH Comput. Graph.* 24, 4 (Sept. 1990), 197–206.

Markus Steinberger, Bernhard Kainz, Stefan Hauswiesner, Rostislav Khlebnikov, Denis Kalkofen, and Dieter Schmalstieg. 2012. Ray prioritization using stylization and visual saliency. *Computers & Graphics* 36, 6 (2012), 673 – 684.

Michael Stengel, Steve Grogorick, Martin Eisemann, and Marcus Magnor. 2016. Adaptive Image-Space Sampling for Gaze-Contingent Real-time Rendering. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 129–139.

Paul Tosca. 2014. Nyra. http://www.paultosca.com/newSite/nyra.html.

Karthik Vaidyanathan, Marco Salvi, Robert Toth, Tim Foley, Tomas Akenine-Möller, Jim Nilsson, Jacob Munkberg, Jon Hasselgren, Masamichi Sugihara, Petrik Clarberg, Tomasz Janczak, and Aaron Lefohn. 2014. Coarse Pixel Shading. In *Proceedings of High Performance Graphics.* Eurographics Association, 9–18.

Alex Vlachos. 2016. Advanced VR Rendering Performance. http://www.gdcvault.com/play/1023134/Advanced-VR-Rendering GDC.

Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on* 13, 4 (2004), 600–612.

Graham Wihlidal. 2017. 4K Checkerboard in Battlefield 1 and Mass Effect Andromeda. GDC.