Sample Distribution Shadow Maps

Andrew Lauritzen* Intel Corporation
Marco Salvi† Intel Corporation
Aaron Lefohn‡ Intel Corporation

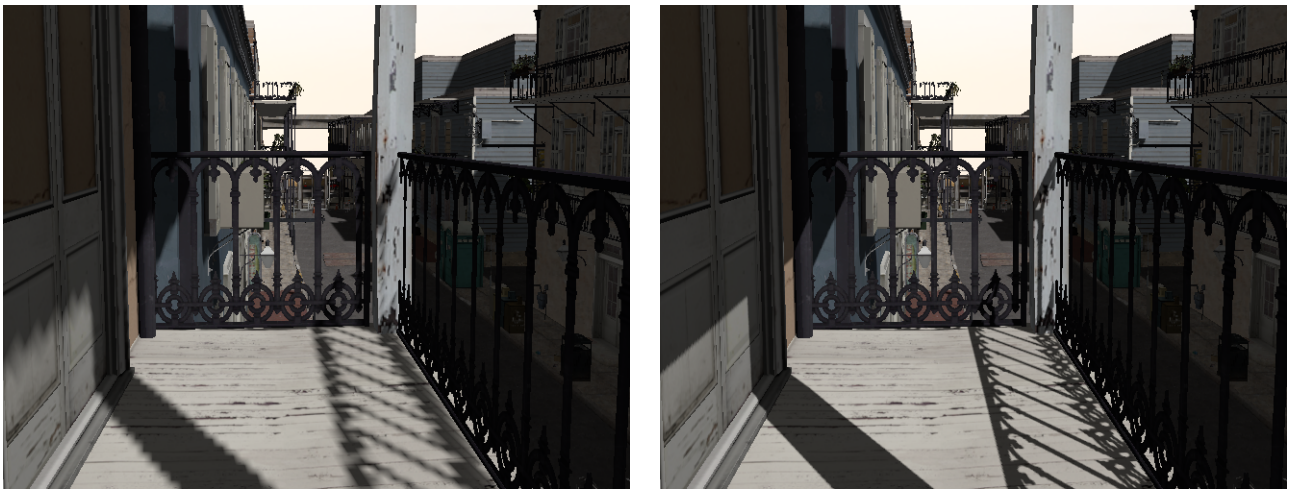


Figure 1: *A comparison of standard, scene-independent Parallel-Split Shadow Maps (PSSMs, left) with sample distribution shadow maps (SDSMs, right). Even with partitions tweaked to a specific camera view, PSSMs still produce lower quality shadows than SDSMs, which are fully automatic. Furthermore, the tighter SDSM shadow frusta enable more aggressive culling causing the algorithm to run faster than scene-independent schemes for complex geometry, such as this scene from Left for Dead 2 (courtesy of Valve Corporation).*

Abstract

This paper introduces Sample Distribution Shadow Maps (SDSMs), a new algorithm for hard and soft-edged shadows that greatly reduces undersampling, oversampling, and geometric aliasing errors compared to other shadow map techniques. SDSMs fall into the space between scene-dependent, variable-performance shadow algorithms and scene-independent, fixed-performance shadow algorithms. They provide a fully automated solution to shadow map aliasing by optimizing the placement and size of a fixed number of Z-partitions using the distribution of the light space samples required by the current frame. SDSMs build on the advantages of current state of the art techniques, including predictable performance and constant memory usage, while removing tedious and ultimately suboptimal parameter tuning. We compare SDSMs to Parallel-Split Shadow Maps (PSSMs, a state of the art Z-partitioning scheme) and show that SDSMs produce higher quality shadows. Finally, we demonstrate that SDSMs outperform PSSMs in a large 2009 game scene at high resolutions, making them suitable for games and other interactive applications.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: Z-partitioning, cascaded shadow maps, shadows, rendering

*e-mail:andrew.t.lauritzen@intel.com
†e-mail: marco.salvi@intel.com
‡e-mail:aaron.lefohn@intel.com

1 Introduction

Rendering high quality shadows efficiently is a difficult problem that continues to be the subject of a large body of research. Shadow mapping is the most common algorithm currently employed in games and other real-time rendering applications. Unfortunately, shadow maps suffer from a variety of aliasing problems due to both oversampling and undersampling.

Oversampling must be addressed by some form of resolution management. Although recent papers have demonstrated adaptive, scene-dependent shadow algorithms running at interactive rates, the variable memory and performance characteristics of these algorithms makes them unsuitable for games. To maintain consistent frame rates, games must design for worst-case performance and operate in a fixed amount of memory. Thus games typically forgo higher quality adaptive algorithms for scene-independent shadow map resolution management strategies. The most commonly used solution is Z-partitioning [Lloyd et al. 2006] (also known as cascaded shadow maps), which partitions the camera frustum and dedicates an independent shadow map for each. These partitions are placed either entirely manually by artists [Valient 2007] or via tunable heuristics [Zhang et al. 2006]. This hand tuning is time-consuming and at best produces a static partitioning that works well for some viewpoints, but poorly for others.

This paper introduces a shadow algorithm called *Sample Distri-*

bution Shadow Maps (SDSMs) which adaptively optimizes the shadow resolution every frame while retaining the fixed memory and performance characteristics required for real-time applications. SDSMs achieve superior quality shadows by analyzing the distribution of shadow map samples required to render the current view and automatically placing tightly-bounded, view-dependent Z-partitions.

Our main contributions are:

- a method for automatically partitioning the camera frustum based on the receiver sample distribution given by the depth buffer, and
- an algorithm to compute tightly-bounded light-space partition frusta from the sample distribution which greatly increases the quality of the shadows.

2 Related Work

Scene-dependent algorithms such as adaptive shadow maps [Fernando et al. 2001], alias-free shadow maps [Aila and Laine 2004], the irregular Z-buffer [Johnson et al. 2005] and resolution-matched shadow maps [Lefohn et al. 2007] build complex light-space data structures to evaluate the shadow receiver samples. These algorithms aim to guarantee shadow quality and thus degrade in performance and/or memory usage in difficult scenes which makes them undesirable for games. Additionally, the data structures used by these algorithms are not easily filtered, making it hard to address other forms of shadow aliasing.

Scene-independent algorithms typically attempt to warp or partition the shadow map to more closely match the generated samples to the samples required by the current view. The SDSM algorithm builds on Z-partitioning [Lloyd et al. 2006] (also known as cascaded shadow maps [Engel 2007]) but fully automates partition placement for the current view. Warping algorithms are orthogonal to our work, but since they complicate shadow map edge softening (they induce non-rectangular blur kernels) they are not employed in our approach.

Some work has been done on improving Z-partitioning by using geometric information from the scene [Zhang et al. 2007]. Unfortunately object-space methods are of limited use in complex scenes as the camera will often be inside at least one object bounding box, which prevents any tightening of the near plane, which is critical for achieving a good frustum partitioning [Lloyd et al. 2006]. Furthermore these techniques can at best eliminate some regions of empty space but they cannot efficiently take advantage of occluded regions in light space, which are more relevant than empty space in complex scenes. By examining the real sample distribution in image space we avoid issues with the granularity of object-space data structures (and indeed avoid the need for them at all) and automatically take advantage of both empty and occluded regions in light space.

3 Algorithm

In this section we outline the Sample Distribution Shadow Map (SDSM) algorithm.

SDSMs are inspired by Z-partitioning, but rather than rely on a static partitioning of the frustum, which can easily produce shadow maps with poor utilization (or none at all), we look at the actual light-space shadow map sample distribution and place the partitions to ensure good utilization of each. To compute this sample distribution, SDSMs use a camera-space depth buffer for the current frame together with a light projection matrix. For each depth buffer sample we reconstruct the world space position and project it into light space to produce the set of shadow samples that are required to render the scene.

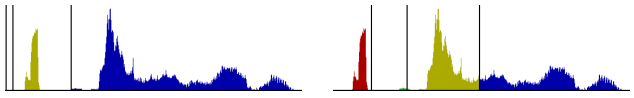


Figure 2: *Comparison of Z-partitioning schemes for a depth distribution (near is on the left, far on the right) from Left 4 Dead 2 with 4 partitions. The black, vertical bars indicate the split points between partitions and the colored distribution indicates which samples fall into which partitions (red, green, yellow, blue). Notice that the basic logarithmic Z-partitioning produces suboptimal results by placing partitions that contain no samples (red and green).*

Using this sample distribution, we automatically compute a partitioning of the depth range that efficiently covers just the required samples in the scene. This step eliminates the manual parameter tweaking that is required by other Z-partitioning schemes and can produce better solutions by continuously adapting the partitioning as the view and scene changes.

The second step of the algorithm uses the depth partitioning and the sample distribution to compute tight light-space partition frusta that cover only regions where samples are required. This step greatly increases the useful resolution of the shadow maps by focusing them on regions with samples and avoiding empty or regions that are occluded from the camera's point of view.

Overall the algorithm is easy to implement, but the results are compelling as we will demonstrate.

3.1 Depth Partitioning

Our simple partitioning strategy is based on placing partitions with minimal far-to-near ratios [Lloyd et al. 2006] to minimize the maximum resolution error over the entire frustum. Figure 2 shows the problem with using the basic logarithmic partitioning: two partitions (red and green) are entirely unused since no geometry falls into their respective depth ranges in this example.

Techniques such as Parallel-Split Shadow Maps (PSSMs) [Zhang et al. 2006] attempt to offset the problem by pushing more of the resolution away from the near plane with the justification that those regions are often unoccupied. Unfortunately, all such schemes introduce a parameter whose optimal value is related to the actual tightly-bounded near plane for the current view, forcing users to hand tune this parameter for their scene.

It is very important to increase the near plane distance to tightly bound the visible geometry in order to get a good partitioning, but scene-independent techniques do not know this distance [Lloyd 2007]. Fortunately, we know precisely what regions of the depth range are occupied from the sample distribution: we simply find the global minimum and maximum depths via a min/max reduction of the depth buffer.

Using these tight near and far planes, we partition the depth range logarithmically, so that the far-to-near ratio of each partition is constant. Figure 2 shows the result of SDSM logarithmic partitioning for the example distribution. Notice how the partitions have been shifted to cover the real sample distribution much more accurately than with basic logarithmic partitions.

3.2 Depth Partitioning Variants

In addition to the simple logarithmic scheme, we explored two more complex partitioning algorithms: adaptive logarithmic and K-means.

The adaptive logarithmic scheme is similar to logarithmic except it attempts to avoid gaps in the depth distribution by pushing partition boundaries to the end of the gaps. While this algorithm produces acceptable results in practice it is more expensive to compute since it requires knowledge of the full depth distribution (in the form of a depth histogram) rather than just the extrema. Computing a full depth histogram is somewhat more expensive than the single min/max reduction required by the simple logarithmic scheme. Unfortunately this variant of the algorithm is fairly opportunistic and view-dependent and in practice the additional cost of computing the histogram is not worth it.

While the logarithmic partitioning schemes aim to minimize the maximum error over the frustum, the K-means scheme tries to place partitions around clusters in the depth histogram, following the Z-partitioning principle that clusters in depth loosely imply clusters in light space. To this end, we search for K peaks in the depth histogram (also required by this variant), and place the partitions around them. To find these peaks, we use the weighted K-means algorithm (also known as Lloyd's algorithm) [Lloyd 1982].

Unfortunately while this algorithm typically gives reasonable results it has significant failure cases which make it impractical for robust interactive use. For instance, a few samples from a distant occluder are considered unimportant by the algorithm and hence are grouped with samples that might be very close to the viewer. While this is reasonable from a K-means point of view, these samples may be quite distant in light space and thus produce a poor partition. We were not able to acceptably resolve this problem via weighting the samples or clusters (based on light space distances or otherwise). Two-dimensional clustering in light space is probably required to combat this issue, but multi-dimensional optimization is significantly more expensive and often less robust than our one-dimensional solution. That said, it is an interesting direction for future research.

3.3 Tight Partition Frusta

Previous algorithms typically project the corners of each frustum partition into light-space and use those to define the shadow map frusta. We can do better by analyzing the receiver sample distribution.

In particular, the samples required for a given frustum partition often do not cover the entire projected footprint of the partition in light space, due to occlusion or empty space. If we restrict the frustum to tightly bound the samples in light space then it will be more compact, since any occluded or empty space around the samples will not be included. To exploit this, we do a light-space reduction over all of the samples that fall into each partition and compute a tight axis-aligned bounding box that we use for the partition frusta. For a larger performance cost we could compute oriented bounding boxes for each partition but we found that simply aligning the entire light frustum to the shadow map axes (i.e. we rotate light space so that the camera's Z axis aligns with the light-space X axis) captured the majority of the benefit at a negligible performance cost.

This is one of the most important steps in the SDSM algorithm. While the partitioning step automates what is typically a manual process, the tight partition frusta step can produce shadows whose quality far exceeds even hand-tuned cascades, as demonstrated in Figure 4.

Once these frusta are computed, the algorithms proceed as with standard Z-partitioning.

4 Implementation

We implemented SDSMs and PSSMs (a common Z-partitioning scheme for comparison) in DirectX 11. For shadow map filtering, we use exponential variance shadow maps [Lauritzen and McCool 2008] (EVSMs) which allows hardware multisample antialiasing (MSAA), mipmapping and anisotropic filtering of the shadows.

To keep the memory footprint of Z-partitioning low, we implemented the algorithms within a deferred rendering framework. This allows us to store only a single shadow partition at once, and iteratively apply the lighting contribution from the different partitions one by one. We also experimented with doing multiple partitions per pass to save framebuffer bandwidth but the difference in performance was negligible, and thus not worth the additional storage cost. In a forward rendering pipeline, a Z-only pass is required before applying shadowing to the scene. Forward renderers that defer evaluation of the shadowing term do not require any additional modification.

When using EVSM filtering we further reduce the storage overhead by rendering to a raw multisampled depth buffer and converting to the EVSM when resolving the buffer. Thus we never need to store the full, multisampled EVSM which can be quite large at 128 bits per texel.

All of the SDSM optimization algorithms and tight bounds determination are implemented in compute shaders. The standard logarithmic partitioning is implemented as a simple min/max reduction, followed by a similar segmented reduction to get the light-space tight partition frusta.

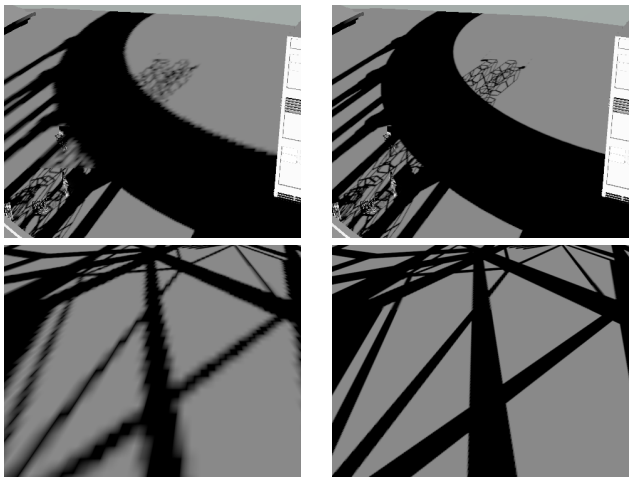
Because this algorithm consists of a few simple reductions, it can also be implemented in pixel shaders on previous graphics hardware at a similar cost via recursive downsampling. Due to limitations on how many values can be output from a pixel shader it may be necessary to perform the tight frusta reduction step in several passes (perhaps as many as one pass per partition). Dispite the additional passes, performance will be similar since the data flow is the same and reductions are entirely bottlenecked on memory operations.

For the partitioning variants, histogram generation is implemented in a compute shader using local and global atomic operations. We also implemented a version compatible with older hardware that uses vertex scattering and blending to generate the histogram [Scheuermann and Hensley 2007], but the performance of this version is highly data-dependent, ranging from two to twenty times slower than the compute shader implementation on the same graphics card. None of the variants that we tested ended up being sufficiently robust and compelling to warrant using them over the simple logarithmic scheme, which does not require a histogram.

An optional shadow edge softening amount is given in world space. The implementation scales this size to be consistent across all partitions and performs the blur as a separable prefilter on the shadow partitions.

5 Results

All results are gathered on an Intel Core i7 CPU with an ATI Radeon HD 5870 GPU. All PSSM results are hand-tuned to produce the best result possible for the given figure. Thus the quality advantage of SDSMs in these comparisons is coming mostly from the tight light-space partition frusta. In practice, PSSMs cannot be



PSSM SDSM (Logarithmic)

Figure 3: *Comparison of PSSM and SDSM (with logarithmic partitions) for two camera viewpoints. PSSM's scene-independent parameter is unable to handle both of these cases gracefully, while SDSM adapts automatically to both cases.*

hand-tuned for every viewpoint and thus compare even less favorably to SDSMs.

5.1 Resolution Management

Figure 3 demonstrates problems with PSSM for two different camera views. While PSSM cannot be tuned to look good for both, the SDSM algorithms do not require any hand tuning at all and automatically handle both cases well. Figure 1 shows another comparison. Notice that even in a single image, SDSMs produce better shadows across the entire camera frustum due to the tighter light space bounds.

The effect of tighter light space bounds is visualized in Figure 4. From the figure it is clear why SDSM results in higher quality shadows: the partitions it generates are simply much smaller because they take advantage of vast empty and occluded regions of light space. In this example, the buildings on either side of the road occlude the majority of the frustum and allocating shadow map space for those regions is a waste. This is not a constructed or uncommon case; large empty or occluded regions in light space are typical in most scenes and SDSM takes advantage of this reality to improve shadow quality.

5.2 Performance

The following table compares the performance of the SDSM algorithm with Z-partitioning (PSSMs in particular) using high quality settings: 4 partitions, each with 1024×1024 resolution and using full EVSM filtering including the cost of $4\times$ shadow MSAA and generating mipmaps for each partition. Results are given for two different screen resolutions and two different scenes. "Tower" is a small scene with ≈ 140 thousand polygons (pictured in Figure 3). Left 4 Dead 2 ("L4D2") is a large scene courtesy of Valve Corporation with ≈ 2.4 million polygons (pictured in Figure 1).

Full Frame Times in Milliseconds				
	1280×800		1920×1200	
	Tower	L4D2	Tower	L4D2
PSSM	6.2	12.6	7.5	15.1
SDSM (Reduce)	7.0	10.3	8.9	13.2
SDSM (Histogram)	7.2	10.5	9.6	13.6

From the small scene we can see that the overhead of the standard SDSM logarithmic analysis using the reduce path is 0.8 ms at 1280×800 and 1.4 ms at 1920×1200, while the variants that require histograms are slightly more expensive. In the large scene, however, SDSMs more than make up for this performance cost by producing tighter shadow frusta (see Figure 4). These tighter frusta allow more aggressive frustum culling during the shadow rendering phase, resulting in a significant net win for the SDSMs.

As an aside it is worth noting that some of the performance cost of SDSMs here is coming indirectly from increased shadow filtering load. Having more (useful) resolution in the shadow maps causes larger anisotropic filters which in turn increases the cost of doing a filtered shadow lookups. Disabling anisotropic filtering or using a shadow filtering algorithm that undersamples the texture footprint (for instance, most real-time implementations of percentage-closer filtering do not consider the texture coordinate derivatives at all) will show less of a performance hit with SDSMs. That said, this is not wasted work; good anisotropic filtering greatly improves the visual quality of shadows particularly when enough shadow resolution is available.

5.3 Limitations

SDSMs expose a limitation of the current graphics APIs when it comes to frustum culling in light space. Since the partition data is generated on the GPU which runs asynchronously, the CPU does not have access to the partition frusta when submitting the shadow partition rendering commands. Thus typical frustum culling cannot be applied. One solution is for the CPU to be conservative and render all objects that overlap any part of the camera frustum for every partition but this is expensive since in many cases partitions do not overlap significantly.

The best current solution in large, complex scenes is actually to stall on the CPU and wait for the GPU to finish generating the partition data, read it back, and continue rendering. The data itself is tiny (bounding boxes for a small number of partitions) so the only significant speed penalty is the stall. In practice, this hit is smaller than the conservative scheme and, as shown in Section 5.2, the tighter frusta of SDSM more than make up for this and all the other overheads of SDSM in even moderately complex scenes.

The ideal solution is for the GPU to do the frustum culling directly and submit the resulting rendering work to itself. In the future, we would like to see graphics APIs support this more general usage than the current model where the GPU is fed exclusively by the CPU.

Changing the shadow map projection each frame can potentially lead to temporal aliasing which is visible as shimmering or swimming shadow edges as the camera moves or rotates slowly. With conventional Z-partitioning this can be addressed by fixing the size of the shadow partitions in light space and quantizing their positions so that the shadow map samples always fall onto a fixed grid in world space. Unfortunately this is not an ideal solution as it is puts constrains on both the light (directional only) and the camera (cannot roll). Perhaps more importantly, fixing the size of the shadow map partitions in world space guarantees a suboptimal partitioning solution since by construction it cannot adapt to the shadow sample distribution.

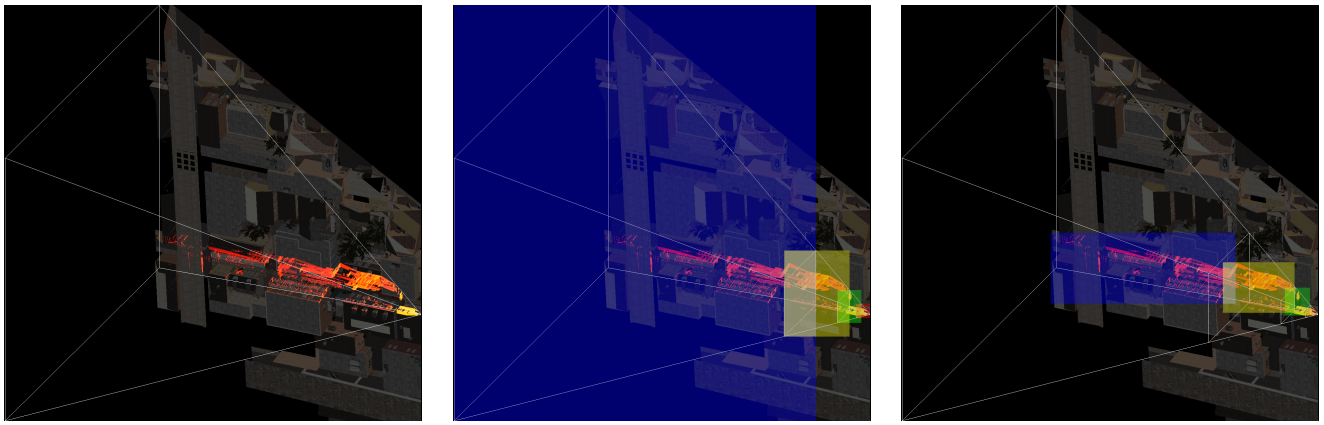


Figure 4: *Left: visualization of the sample distribution in light space resulting from the view in Figure 1. The camera frustum is shown as a white wireframe and the red/yellow regions show where shadow samples are required (i.e. that the camera can see) with yellower regions indicating a higher density of samples. Middle: the hand-tuned PSSM partitioning is reasonable but the light-space partition frusta (colored rectangles) are a poor fit to the actual samples since large sections of the camera frustum are occluded or empty. Right: in contrast, SDSM automatically produces a good partitioning and more importantly it computes frusta that tightly bound the required samples which results in higher shadow quality.*

The ideal solution to this problem is to achieve sub-pixel shadow resolution, which is generally achievable with SDSMs with each partitioning being approximately screen resolution or larger and good filtering. At this resolution or better, no temporal (or spatial) aliasing is visible. If getting sub-pixel shadow resolution is not possible or practical, a variant of the world space quantization scheme can be used. To allow resolution to increase or decrease without noticeably altering the texel alignment, partitions can be quantized to power-of-two sizes so that they always double or halve resolution. Then each partition can be snapped to a world-space grid as usual. While this scheme is more flexible and will still produce better results than purely static partitions it can waste a large fraction of the shadow map. Thus achieving sub-pixel shadow resolution should still be the preferred solution.

6 Conclusions and Future Work

In summary, we have shown that SDSMs provide a compelling shadowing solution for games or other applications with fixed performance and memory budgets. By analyzing the sample distribution required by the current view, SDSMs are able to automatically place good partitions without any hand tuning. We have analyzed the performance and quality results of SDSMs in a 2009 game scene and shown that the simple SDSM logarithmic partitioning scheme performs very well in practice. Finally, we have demonstrated that SDSMs benefit greatly from tight light-space frusta, enabling them to surpass the quality of hand-tuned Z-partitions in real scenes.

In the future, we would be interested in further exploring the space of algorithms that use knowledge of the sample distribution to produce high quality shadows within a fixed memory and performance budget. Algorithms that also attempt to address projective aliasing within these constraints are of particular interest.

Acknowledgements

We would like to thank Jason Mitchell and Wade Schin from Valve Corporation for providing the Left 4 Dead 2 scene, and Jeffery Williams for the Tower scene. Additionally, we would like to thank Nico Galoppo, Johan Andersson, Natalya Tatarchuk, Hao Chen, Greg Johnson, Matt Pharr, Craig Kolb, Kiril Vidimče and Tomas Akenine-Möller for their suggestions, feedback and support. Thanks also to our reviewers for helping us to improve the paper.

References

AILA, T., AND LAINE, S. 2004. Alias-free shadow maps. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering*, 161–166.

ENGEL, W. 2007. Cascaded shadow maps. In *ShaderX5: Advanced Rendering Techniques*, W. Engel, Ed. Charles River Media, 197–206.

FERNANDO, R., FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. 2001. Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 387–390.

JOHNSON, G. S., LEE, J., BURNS, C. A., AND MARK, W. R. 2005. The irregular Z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics 24*, 4, 1462–1482.

LAURITZEN, A., AND MCCOOL, M. 2008. Layered variance shadow maps. In *Proceedings of the 2008 Conference on Graphics Interface*, 139–146.

LEFOHN, A. E., SENGUPTA, S., AND OWENS, J. D. 2007. Resolution-matched shadow maps. *ACM Transactions on Graphics 26*, 4 (Oct.), 20:1–20:17.

LLOYD, B., TUFT, D., YOON, S., AND MANOCHA, D. 2006. Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006*, Eurographics Association, 215–226.

LLOYD, S. 1982. Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2 (Mar.), 129–137.

LLOYD, D. B. 2007. *Logarithmic Perspective Shadow Maps*. PhD thesis, Chapel Hill, NC, USA.

SCHEUERMANN, T., AND HENSLEY, J. 2007. Efficient histogram generation using scattering on gpus. In *I3D '07: Proceedings*

of the 2007 symposium on Interactive 3D graphics and games, ACM, New York, NY, USA, 33–37.

VALIENT, M. 2007. Deferred rendering in killzone 2. In *Proceedings of Develop Conference 2007*. http://www.guerrilla-games.com/publications/dr_kz2_rsx_dev07.pdf.

ZHANG, F., SUN, H., XU, L., AND LUN, L. K. 2006. Parallel-split shadow maps for large-scale virtual environments. In *VRCIA '06: Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications*, ACM, New York, NY, USA, 311–318.

ZHANG, F., SUN, H., AND NYMAN, O. 2007. Parallel-split shadow maps on programmable gpus. In *GPU Gems 3*, H. Nguyen, Ed. Addison Wesley, July, ch. 10, 203–237.