Fast Multi-View Soft Shadowing via Fragment Reprojection

Adam Marrs, Benjamin Watson, Christopher Healey

North Carolina State University

Figure 1: *Soft shadowing of a 100,000 polygon scene, using 64 depth maps of 512^2 resolution on an NVIDIA GTX Titan. Light samples are distributed over an area light source using random Poisson disc distribution. Left: depth maps generated by brute force rasterization in 4.85 ms. Right: depth maps generated by synchronized fragment reprojection in 0.48 ms.*

Abstract

This paper presents a practical approach for producing many unique depth maps of a scene in a single geometry pass without any additional rasterization. We use this method to accelerate the performance of soft shadowing algorithms for geometrically complex scenes. Physically accurate Monte-Carlo style soft shadowing algorithms typically require prohibitive amounts of rasterization to produce many depth maps from slightly differing viewpoints. Due to the similarity between viewpoints, much of the depth computation is redundant. We leverage this observation by rasterizing a single central depth map and then reprojecting central depth map fragments into different view spaces, producing new depth maps. Since fragment reprojection is a function of the number of fragments stored in the central depth map, performance is only weakly linked to scene geometry. This produces speeds comparable to Percentage Closer Soft Shadows (PCSS) and quality comparable to Monte-Carlo style brute force rasterization. Our method is more general and easier to use than PCSS, at the cost of higher memory usage.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

As a perceptually important visual effect, shadows have been an active research topic in the graphics community for several decades. Important visual cues encoded in shadows were identified as early as Renaissance painters and studied more recently by cognitive psychologists [DK96]. Shadows provide both static and dynamic visual clues about the location, shape, and arrangement of objects in an environment, as well as various characteristics about relevant light sources [MKD98].

Shadows are created by the complex interaction of light with objects and the environment. Light energy is constantly

absorbed and scattered as it collides with surfaces. Reproducing this behavior is especially challenging at real time speeds. Since the first shadow research was presented in the late 1970's [Cro77, Wil78] much research, along with hardware and software advancements, has been done to improve shadow rendering. Unfortunately, a comprehensive solution for producing accurate real time soft shadows in fully dynamic environments has yet to be found [SLSW13].

The primary task of soft shadowing algorithms is solving for the *visibility factor*: the proportion of an area light visible to each point on a receiving surface [ESAW11]. Accurately solving the visibility factor requires: 1) identifying geometry which occludes the light source as viewed from each point on a receiving surface and 2) determining how much of the light source each occluder obscures. While many clever approximations for the visibility factor exist [Fer05, DL06, GBP06, SSMW09], the most accurate rasterization-based methods produce a large number of depth maps to determine occlusion.

Modern graphics hardware is well tailored for efficiently generating a single depth map. Unfortunately, Monte-Carlo techniques require a large number of depth maps (64 - 256) to achieve acceptable quality [ESAW11]. Critically, the information stored in the various depth maps is often similar, if not identical, producing redundant computation. Compounding this problem, the caching structures of modern graphics processors are designed for throughput and are ineffective at keeping previous depth computations readily available.

We present a new approach, called Fragment Reprojection, which improves the performance of Monte-Carlo style soft shadowing algorithms by addressing the depth map generation bottleneck. Our method:

1. Leverages the similarity of information in the depth maps
2. Restructures depth computations to avoid rasterization
3. Harnesses GPGPU capabilities of recent GPU hardware
4. Produces all necessary depth maps without additional passes over the scene geometry

The performance of our approach is a function of the number of fragments stored in a single central depth map. This produces a performance curve that is weakly linked to scene geometry, yielding speeds comparable to Percentage Closer Soft Shadows (PCSS) and quality comparable to Monte-Carlo style brute force rasterization. Our approach is more general and easier to use than PCSS, at the cost of higher memory usage. Since our method performs rasterization only for the central view, some light leaking artifacts and slight umbra over-estimation exist (see Figure 1).

2. Related Work

Soft shadowing has been an active area of research for several decades. As such there are a multitude of approaches for solving the soft shadowing problem. We limit our related work discussion to the main categories of image-space soft shadowing techniques that inspired this work.

Monte-Carlo Sampling: The most accurate soft shadowing approaches employ sampling techniques which distribute randomly positioned samples across an area light. Each light sample is then used as an eye location and geometry between the eye and the receiver is rasterized [HH96]. The visibility factor is then calculated by performing standard shadow mapping for each depth map and accumulating the results on the graphics hardware. The visual quality is excellent, since occlusion is correctly represented for each light sample; however, the large amount of rasterization per frame is prohibitively slow for all but very small scenes [ESAW11]. Due to the extensive amount of sampling, these approaches are often referred to as 'brute force'.

Temporal Reprojection: Another category of soft shadow algorithms is built upon the insight that rendered frames are often similar, or even identical, over a period of time. This approach distributes rasterization work over many frames at the cost of requiring multiple frames to resolve the correct solution. The use of temporal reprojection was recently presented as a method to iteratively refine hard and soft shadows [SJW07, SSMW09]. Utilizing a history buffer, these approaches store shadow mapping results over a few frames, and reproject the results into the current frame. While temporal coherence techniques achieve real time speeds, the visual quality is limited to the initial unrefined solution when there is little similarity between frames.

Backprojection: Employing a more physically based approach, backprojection algorithms look toward an area light from each point on a receiver and evaluate how much of the light source is visible. They accomplish this using a depth map, rasterized from the center of an area light, as an approximation of occluding geometry. The depth map is sampled over an area to search for occluding fragments, which are then reprojected onto the area light. The proportion of the area light not covered by the reprojected fragments represents the amount of light energy reaching the current point on the receiver [GBP06]. For large penumbrae, the amount of depth map searching is rather high, requiring significant amounts of interaction with texture resources.

Adaptive Filtering: Filtering based algorithms perform shadow mapping and apply a variable amount of filtering to shadow edges to simulate penumbra and contact hardening. Adaptive filtering techniques often make major assumptions about the occluding geometry or ignore the geometry completely. For example, PCSS assumes there is only one parallel planar occluder in order to use the parallel planes equation for penumbra width estimation [Fer05]. While filtering approaches require significantly less memory, their approximations are incorrect for the majority of situations [ESAW11].

3. Algorithm

We propose an algorithm which improves the performance of Monte-Carlo sampling based approaches by generating many depth maps without additional geometry passes or rasterization. Since light sample views are typically very similar, the fragments of their associated depth buffers will also be similar. We take advantage of this similarity by reprojecting rasterized depth fragments of a single light view into many different light views, thus closely approximating rasterization output. Our algorithm is easy to implement and integrate since depth rasterization and shadow mapping are common features of nearly all real time rendering engines. The algorithm is as follows:

Step 1: Central Depth Rasterization
Render a depth map from the center of the area light.

Step 2: Fragment Reprojection
For every fragment of the central depth map:

- Construct the fragment's post projection 3D location
- Reproject the 3D location into another light sample view
- Store the reprojected fragment depth in a buffer
- Repeat for many views

Step 3: Shadow Mapping
Perform shadow mapping, averaging shadow comparisons from the generated depth maps.

3.1. Implementation

Previously, graphics hardware did not allow for the generalized access to computation and storage resources required to perform fragment reprojection. Specifically, threads dispatched for the specialized shader stages (vertex, pixel) could not write to arbitrary locations of dynamically selected memory resources. The introduction of the Geometry shader enabled the rasterizer to output to multiple render targets; however, the input order of primitives had to be retained [AMHH08]. In practical terms, this meant primitives were stored and sorted in an intermediate form - often in off-chip video memory. This severely limited the performance of Geometry shader code.

With the recent introduction of graphics hardware structures to support GPGPU computation, many of these roadblocks have been removed. The computation pipeline, exposed by APIs such as DirectCompute and CUDA, provides the general access to hardware resources we require. Parallel execution is defined by a new threading model and threads have access to new memory resources that support writing to arbitrary locations. Additionally, this model allows for more explicit control over how threads are created and executed by GPU processors.

In conversations with our NVIDIA collaborators, it was pointed out that the Pixel shader stage has also been given access to the new GPGPU memory resources. To evaluate the performance differences, we implemented fragment reprojection in both Compute and Pixel shaders.

3.1.1. Compute Shader Fragment Reprojection

By interleaving graphics tasks with the computation pipeline, we have implemented our algorithm in three passes: 1) rasterization of the central depth map, 2) fragment reprojection in the compute shader, and 3) a final eye pass to perform shadow mapping and shading.

Since the depth values produced by reprojection depend on the rasterized central depth fragments, a depth bias should not be added during central depth rasterization. Instead, it is added after fragment reprojection. Figure 2 shows self shadowing caused by incorrect depth biasing versus correct shading from post-reprojection bias.

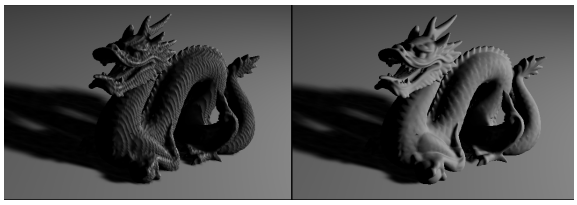


Figure 2: *Left: self shadowing due to adding depth bias before reprojection. Right: depth bias added after reprojection.*

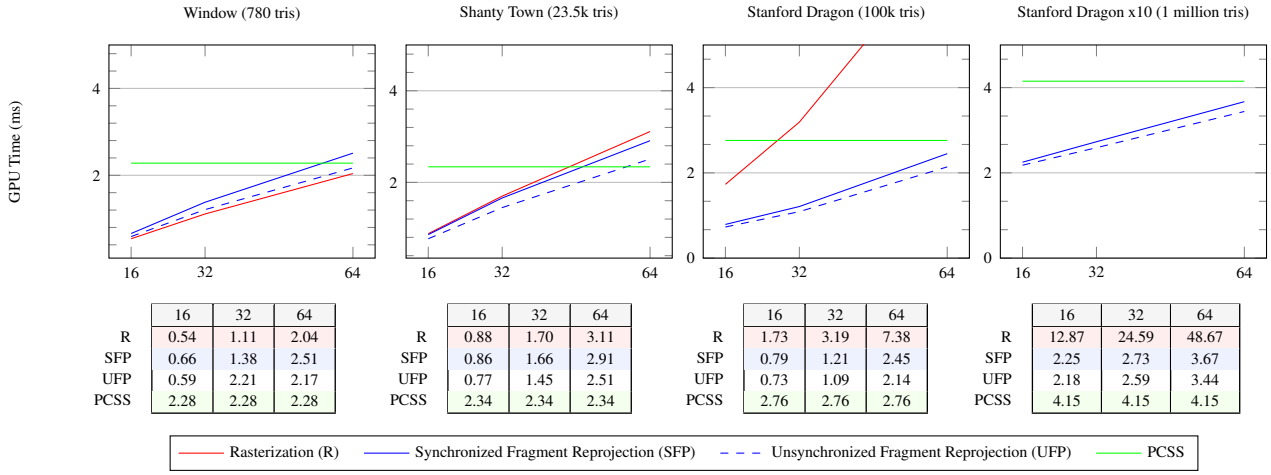
3.1.2. Pixel Shader Fragment Reprojection

The Pixel shader allows our algorithm to complete in two steps. Fragment reprojection occurs in the Pixel shader stage of our central depth rasterization step. In order to perform fragment reprojection efficiently, the correct depth should be resolved for each fragment before the associated Pixel shader executes. The graphics pipeline performs z-buffering after the Pixel shading stage; however, graphics APIs often include methods to expedite z-buffering. In Direct3D 11, this is enabled through a shader semantic called `[earlydepthstencil]`. Alternatively, Pixel shader fragment reprojection can be implemented in three steps, much like its Compute counterpart, instead using a full screen pass to trigger Pixel shaders for the Fragment Reprojection step.

3.2. Thread Synchronization

During fragment reprojection, many depth fragments may reproject to the same pixel location of a destination depth map. Since fragment reprojection threads are executed in parallel, a race condition will exist between shader threads attempting to write to the same location.

To evaluate the impact of unsynchronized threads, we implemented the algorithm both with and without thread synchronization. In the unsynchronized case, the last thread that attempts to write to the destination depth map will succeed.



	16	32	64
R	0.54	1.11	2.04
SFP	0.66	1.38	2.51
UFP	0.59	2.21	2.17
PCSS	2.28	2.28	2.28

	16	32	64
R	0.88	1.70	3.11
SFP	0.86	1.66	2.91
UFP	0.77	1.45	2.51
PCSS	2.34	2.34	2.34

	16	32	64
R	1.73	3.19	7.38
SFP	0.79	1.21	2.45
UFP	0.73	1.09	2.14
PCSS	2.76	2.76	2.76

	16	32	64
R	12.87	24.59	48.67
SFP	2.25	2.73	3.67
UFP	2.18	2.59	3.44
PCSS	4.15	4.15	4.15

Figure 3: *Performance comparison of brute force rasterization (red), synchronized (solid blue) and unsynchronized (dashed blue) fragment reprojection, and PCSS (green) for each of our test environments.*

In the synchronized case, so called "atomic operations" are used to ensure requests to read and write from memory are executed in the order they are received. To ensure the closest fragment is stored in each depth map, threads perform an atomic minimum operation on the reprojected depth and the value currently stored in memory. In this way, compute thread synchronization is analogous to z-buffering in the graphics pipeline.

4. Results

To evaluate our algorithm, we use two benchmarks: 1) brute force rasterization, representing high visual quality at the expense of low speed and 2) PCSS, representing lower quality at practical speeds. We chose PCSS as a practical performance benchmark due to its inclusion in recent high profile games [Bur13]. PCSS's performance is a function of the number of texture samples used in the occluder search and filtering steps, since it only uses a single light sample. We tuned the sampling amount for the best quality possible.

Our tests were run on an Intel i7-930 2.8GHz CPU, NVIDIA GTX Titan GPU, and Direct3D 11. We created four scenes with occluders of differing geometric complexity. The 'Window' is 780 triangles, the 'Shanty Town' is 23,500 triangles, and the 'Stanford Dragon' is 100,000 triangles. Our large test scene contains ten Stanford Dragons, for a total of 1 million triangles. We use Poisson disc random distribution for selecting area light sample locations, 32 bit depth buffers, and 2x2 hardware percentage closer filtering. The number of light samples is varied between 16, 32, and 64 samples.

All scene geometry fits within the GPU's L2 cache. This is a best case scenario for brute force rasterization, since only the first request for geometry will fetch from video memory. As a result, brute force rasterization exhibits a linear performance curve, where the number of rasterized triangles is inversely proportional to the speed of processing. In typical real time applications, this scenario is often not the case. The L2 cache will be filled and evicted many times, forcing more interaction with video memory.

4.1. Performance

As shown in Figure 3, the performance of both synchronized and unsynchronized fragment reprojection perform significantly better than best case brute force rasterization and PCSS when scene geometry is at least 100,000 triangles. Fragment reprojection begins to improve performance when scene geometry is at least 23,000 triangles. As the scene geometry becomes more complex, fragment reprojection is anywhere from 10× to 20× faster than best case rasterization. As demonstrated by the 1 million triangle scene, fragment reprojection's performance curve is only weakly dependent on geometric complexity and the number of light samples.

4.1.1. Pixel Shader Fragment Reprojection

Performance of fragment reprojection using the three pass Pixel shader is comparable to its Compute shader counterpart. When the depth complexity of a scene is high, Pixel shader reprojection is slower using the two pass `[early-depthstencil]` approach. It appears that expedited z-

Figure 4: *The visual difference between the brute force rasterization reference and synchronized fragment reprojection, using 64 samples and 512^2 resolution depth maps. Lighter areas indicate percentage difference between the images.*

buffering is not reliable, since Pixel shaders are being executed for fragments before z-buffering is complete. Additionally, the output merger would be a natural choice for performing synchronization of Pixel shader threads; however, the output merger is not compatible with GPGPU memory resources.

4.1.2. Synchronization and GPU Limitations

Synchronizing compute threads imposes a performance cost. Part of this cost is due to threads waiting for memory resources to be available during atomic operations. The bulk of the synchronization cost; however, comes from a limitation in the GPU hardware. Unfortunately, most modern GPUs only support atomic operations on integer format memory resources while the sampling and filtering hardware require floating point memory resources. Practically, this means if threads are synchronized we must either: 1) copy the integer resource to a floating point resource or 2) manually implement sampling and filtering of the integer memory resource in the final shading pass. Since manual filtering of integer resources is especially slow, and we can perform an optimized copy of GPU texture memory, we've found copying is a better option. Hardware supported floating point atomic operations would eliminate this unnecessary overhead from the synchronization cost.

4.2. Quality

The quality of fragment reprojection is very good. Support for large penumbrae is automatic and requires no tweaking. Figure 4 visualizes the differences between the brute force rasterization reference and synchronized fragment reprojection. Dark areas indiciate no difference, while lighter areas indicate the percentage of difference. We have also computed the normalized Root Mean Squared Error (RMSE). As shown in Figure 6, RMSE values for fragment reprojection are lower than PCSS for the Window and Stanford Dragon, but slightly higher for the Shanty Town due to light leaking artifacts.

4.2.1. Light Leaking

Fragment reprojection does not perform rasterization from each light sample's view. Consequently, depth maps produced during reprojection can both introduce newly occluded fragments and miss newly visible fragments that would have been generated during rasterization. As we observe with the Shanty Town, the inability to generate the exact fragments for light views in a high depth complexity scene lowers the achievable accuracy.

If light samples are positioned far enough away from the central light view, gaps are produced in the reprojected depth maps (Figure 5). These gaps are the cause of erroneously lit areas of shadow in the final image. To reduce these gaps, we adjust how the central light view is sampled. Increasing the resolution of the central light view, to ensure it is higher than the reprojection depth maps, generates additional fragments for each reprojection depth map pixel. These extra samples help fill depth map gaps.

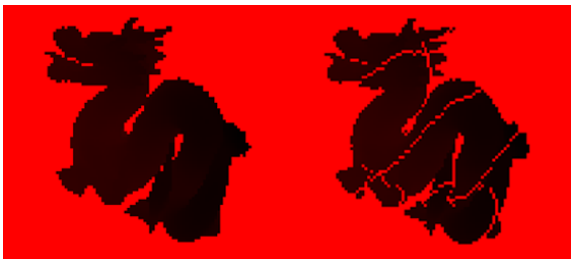


Figure 5: *Left: rasterized depth map. Right: depth map produced by fragment reprojection.*

These additional fragments are the primary cause of umbra over-estimation in the final image, which increases the

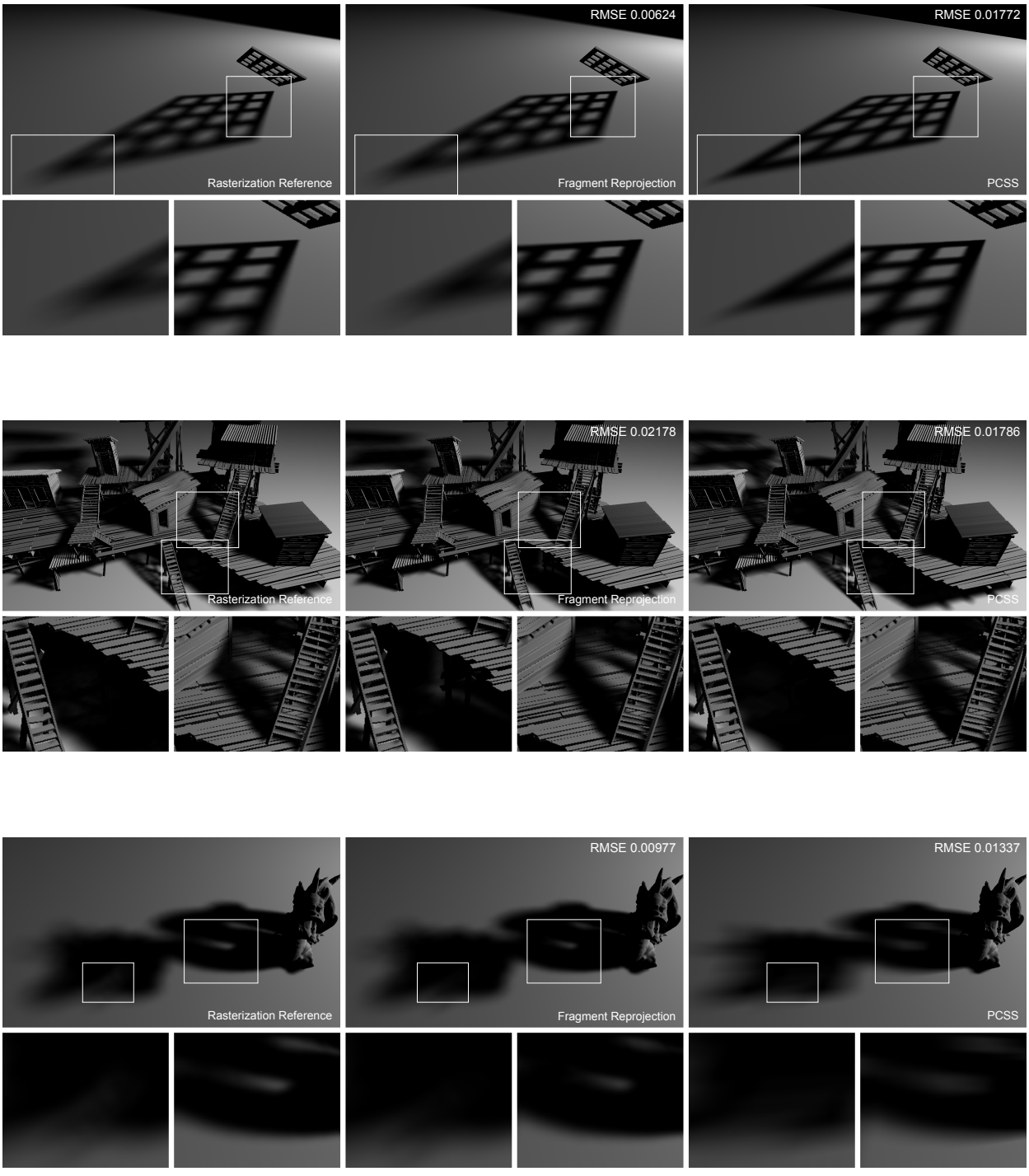


Figure 6: *Comparison of final shadowing. Top: Window, Middle: Shanty Town, Bottom: Stanford Dragon*

RMSE. However, it is well known that RMSE is not a good measure of perceived difference. As shown in Figure 6, visual differences caused by umbra over-estimation are less noticable than the filtering approach used in PCSS.

4.2.2. Temporal Aliasing

Increasing the resolution of the central light depth map increases the number of threads competing for the reprojection depth map memory resource. Increasing the depth complexity of the scene increases the probability that competing threads will store highly variable depth values. From this relationship we observe that thread synchronization is not necessary when scene depth complexity is low. In this case, the competition of threads for a memory resource is irrelevant, since each reprojected fragment's depth will be very similar. However, in scenes with high depth complexity, lack of synchronization creates objectionable temporal aliasing. Temporal aliasing is fixed by synchronizing threads and storing the closest fragment.

4.3. Limitations

The primary limitation of fragment reprojection is support for large area light sources. As an area light's size increases, light samples are positioned farther away from the central light view. This increases gaps in the reprojected depth maps, causing light leaking. Additionally, storing depth maps for every light view can use a large amount of memory. For 32-bit 512^2 resolution depth maps, each area light sample will use 1MB of memory. Due to GPU limitations, synchronization of compute threads using the copy-to-float approach doubles this cost. Fortunately, memory contraints are rapidly becoming less of a concern since modern game consoles and new GPUs include 3 to 8 GB of dedicated GPU memory.

5. Conclusions

We have presented a new, practical approach for accelerating the generation of many unique depth maps. We use this approach to improve the performance of Monte-Carlo style soft shadowing algorithms. By leveraging the similarity between depth maps, we avoid rasterization by reprojecting depth map fragments from a single rasterized view into many view spaces. The performance of our approach is a function of the number of fragments stored in the central depth map and produces a performance curve that is only weakly linked to scene geometry.

Fragment reprojection is easily integrated into existing rendering engines since it can be implemented with either Compute or Pixel shaders. Our contribution is an easier to use, more general method than PCSS with comparable speed and better quality. Our approach requires no tweaking and supports large penumbrae automatically, at the cost of higher memory usage.

5.1. Future Work

Since the quality of fragment reprojection depends heavily on the initial sampling of the scene, we are exploring more effective sampling methods to support reprojection. Improved initial scene sampling will reduce or eliminate light leaking artifacts and further improve quality. Given the performance savings of fragment reprojection, we are also interested in applying reprojection to other multi-view effects including omni-directional soft shadows and environment mapping.

Acknowledgments

We would like to thank NVIDIA for generously donating GTX Titan graphics hardware in support of this work. We thank David Luebke and Eric Enderton for their valuable thoughts and feedback. Special thanks to Vincent Argentina for the Shanty Town model.

References

[AMHH08] AKENINE-MOLLER T., HAINES E., HOFFMAN N.: *Real Time Rendering 3rd Edition*. 2008. 3

[Bur13] BURNES A.: Assassin's creed iv: Black flag graphics and performance guide, 11 2013. URL: http://www.geforce.com/whats-new/guides/. 4

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph. 11*, 2 (July 1977), 242–248. URL: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/965141.563901, doi:10.1145/965141.563901. 2

[DK96] DANIEL KERSTEN DAVID C. KNILL P. M. I. B.: Illusory motion from shadows, 1996. URL: http://www.nature.com/nature/journal/v379/n6560/full/379031a0.html, doi:10.1038/379031a0. 1

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006), I3D '06, ACM, pp. 161–165. URL: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/1111411.1111440, doi:10.1145/1111411.1111440. 2

[ESAW11] EISEMANN E., SCHWARZ M., ASSARSSON U., WIMMER M.: *Real-Time Shadows*. A.K. Peters, 2011. URL: http://www.cg.tuwien.ac.at/research/publications/2011/EISEMANN-2011-RTS/. 2

[Fer05] FERNANDO R.: Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches* (New York, NY, USA, 2005), SIGGRAPH '05, ACM. URL: http://doi.acm.org/10.1145/1187112.1187153, doi:10.1145/1187112.1187153. 2

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Real-time soft shadow mapping by backprojection. In *Proceedings of the 17th Eurographics conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2006), EGSR'06, Eurographics Association, pp. 227–234. URL: http://dx.doi.org/10.2312/EGWR/EGSR06/227-234, doi:10.2312/EGWR/EGSR06/227-234. 2

[HH96] HERF M., HECKBERT P. S.: Fast soft shadows. In *ACM SIGGRAPH 96 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '96* (New York, NY,

USA, 1996), SIGGRAPH '96, ACM, pp. 145–. URL: http://doi.acm.org/10.1145/253607.253870, doi:10.1145/253607.253870. 2

[MKD98] MAMASSIAN P., KNILL D., D K.: The perception of cast shadows. *Trends in Cognitive Sciences 2*, 8 (August 1998), 288–295. 1

[SJW07] SCHERZER D., JESCHKE S., WIMMER M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proceedings of the 18th Eurographics conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2007), EGSR'07, Eurographics Association, pp. 45–50. URL: http://dx.doi.org/10.2312/EGWR/EGSR07/045-050, doi:10.2312/EGWR/EGSR07/045-050. 2

[SLSW13] SCHWÄRZLER M., LUKSCH C., SCHERZER D., WIMMER M.: Fast percentage closer soft shadows using temporal coherence. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2013), I3D '13, ACM, pp. 79–86. URL: http://doi.acm.org/10.1145/2448196.2448209, doi:10.1145/2448196.2448209. 2

[SSMW09] SCHERZER D., SCHWÄRZLER M., MATTAUSCH O., WIMMER M.: Real-time soft shadows using temporal coherence. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II* (Berlin, Heidelberg, 2009), ISVC '09, Springer-Verlag, pp. 13–24. URL: http://dx.doi.org/10.1007/978-3-642-10520-3_2, doi:10.1007/978-3-642-10520-3_2. 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (Aug. 1978), 270–274. URL: http://doi.acm.org/10.1145/965139.807402, doi:10.1145/965139.807402. 2