Real-time Seamless Object Space Shading

Tianyu Li[1†] and Xiaoxin Guo[1†]

[1]TiMi L1 Studio, Tencent Games, China

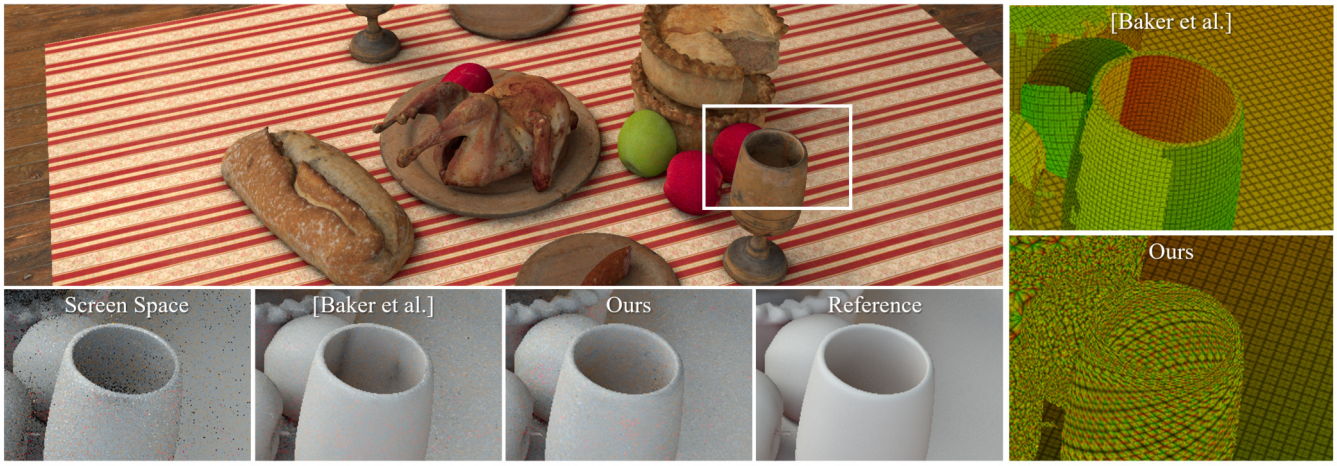


Figure 1: *A table full of food is entirely illuminated by indirect lighting. The images in the second row are rendered with different methods in 3 ms on an NVIDIA RTX 4090 GPU, from left to right: screen space ReSTIR GI estimator, object space ReSTIR GI estimator based on Baker et al. [BJ22]'s method, object space ReSTIR GI estimator based on our method. The right side column shows object parameterization visualization of different methods.*

Abstract

Object space shading remains a challenging problem in real-time rendering due to runtime overhead and object parameterization limitations. While the recently developed algorithm by Baker et al. [BJ22] enables high-performance real-time object space shading, it still suffers from seam artifacts. In this paper, we introduce an innovative object space shading system leveraging a virtualized per-halfedge texturing schema to obviate excessive shading and preclude texture seam artifacts. Moreover, we implement ReSTIR GI on our system (see Figure 1), removing the necessity of temporally reprojecting shading samples and improving the convergence of areas of disocclusion. Our system yields superior results in terms of both efficiency and visual fidelity.

CCS Concepts

• ***Computing methodologies*** → ***Rendering;***

1. Introduction

Real-time rendering demands high performance to achieve frame rates of 60 frames per second or greater. With the increasing adoption of ray tracing techniques, meeting these performance requirements has become more challenging. Temporal anti-aliasing [TDD*22] amortizes computations across multiple frames to improve efficiency but suffers from ghosting artifacts. Object space shading decouples visibility calculations from shading, enabling independent per-object shading frequency and superior anti-aliasing. However, the technique often requires greater memory consumption, limiting real-time rendering applications. Virtual texture based object space shading system, as introduced by Baker et al. [BJ22], avoids over-shading and does not require excessive memory de-

† Equal contributions

mands. However, being a UV-map-based solution, it inherently suffers from well-known seam problems [TYL17], limiting its application to spatiotemporal kernels. Our paper presents a virtualization approach for Htex [BD22] that eliminates seam artifacts. Furthermore, a ReSTIR global illumination solution is implemented within our object space pipeline, demonstrating improved stability because of obviating the need for re-projecting samples. Additionally, we have implemented persistent layers in our system to get a fallback reservoir from persistent layers during disocclusion scenarios. This innovation increases convergence with minimal impact on performance.

2. Related Work

Pixar [CCC87] proposed the original object shading framework, which subdivides visible polygons into micro polygons and then shades micro polygons.

In the domain of real-time object space shading, there are many ideas relying on UV maps [AHTA14; BJ22]. Andersson et al. [AHTA14] used a per-triangle culling method to determine the primitives to finally shading. Baker et al. [BJ22] developed a fine-grained 'virtual shadel allocation' strategy, whereby 8x8 texel blocks within the texture space—referred to as shadels—are dynamically flagged in a pre-Z pass, with memory subsequently allocated to these identified shadels. This approach avoids over-shading and introduces negligible runtime overhead. However, all these approaches rely on UV maps, which cannot ensure geometry locality during shading. As a result, spatial filtering is not practical.

To deploy spatiotemporal rendering algorithms in object space shading systems, seamless mesh parameterization methods are necessary. PTex [BL08], Mesh Colors [YKH10], and Htex [BD22] are all face-intrinsic implicit parameterization methods, among which Htex is the most GPU-friendly in implementation. PTex is designed for quad meshes and requires reading data from neighbor primitives for texture filtering, restricting its GPU adaption. Mesh Colors needs hardware extension for hardware-accelerated filtering. Htex can fit into any triangular mesh topology and works on common GPUs.

When it comes to ray tracing, reservoir-based resampled importance sampling emerges as a paramount focus in real-time ray tracing literature [BWP*20; OLK*21; LKB*22]. Its constant complexity and exponentially growing resampled samples perfectly fit into real-time usage. However, such methods need to efficiently accumulate temporal samples, resulting in inevitable ghosting and blurring. To solve this problem, Boksansky et al. [BJW21] employ a grid-based structure for reservoir resampling. However, the grid can only offer limited resolution under practical storage constraints, which consequently impairs the precision of the technique.

3. Algorithm Overview

The algorithm is summarized in Figure 2. Initially, our method involves preparing the layout information for packing all render textures of meshes into a single atlas texture. This step only needs to be evaluated once. Subsequently, the unique bits of visible shadels are marked in an occupancy map through a pre-Z pass. After shadel marking, the memory of shadels can be allocated via a sparse memory allocator based on occupancy information. Thereafter, a dedicated shading kernel operates on a per-shadel basis.

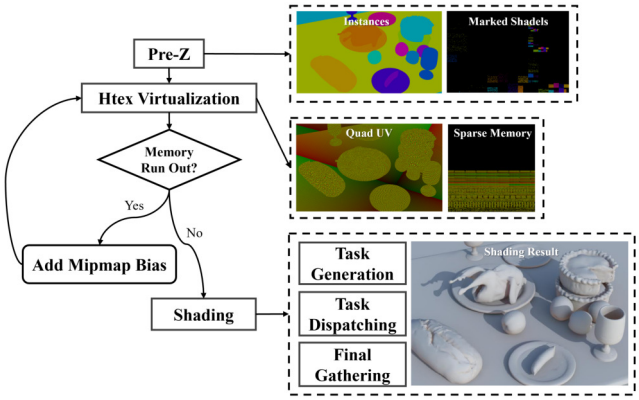


Figure 2: *Pipeline overview of our shading system.*

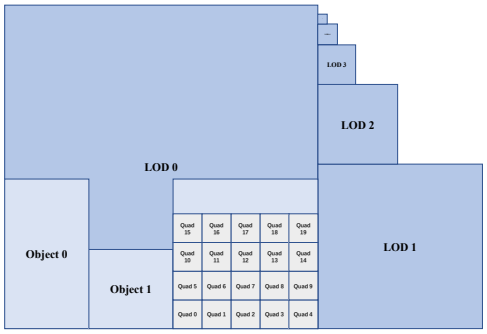


Figure 3: *Visualization of the atlas layout. All texture layers are packed into the same virtual texture. Higher levels are put on the bottom-left side of the remaining space.*

3.1. Htex Atlas Layout

The layout of our atlas can be seen in Figure 3. The textures associated with the intrinsic faces of all meshes across different levels are compactly packed into a single virtual texture. For each object, per-halfedge quad textures will form a rectangular texture depending on the number of faces and the texture quad size of each face. Quads are packed in a scan-line order. Hence we can compute the primitive index via atlas location. Subsequently, these individual rectangular textures, varying in size, are efficiently organized into a single texture using a 2D rectangular packing algorithm.

Ideally, quad textures should be sized differently based on the size of the quad. Using textures of the same size for all quads can lead to under-shading. We address this issue by allocating the largest possible quad atlas, ensuring that even the largest quads receive sufficient resolution in virtual texture space. In the subsequent pre-Z step, the mipmap levels for each shadel are calculated based on screen-space derivatives, thus preventing both over-shading and under-shading.

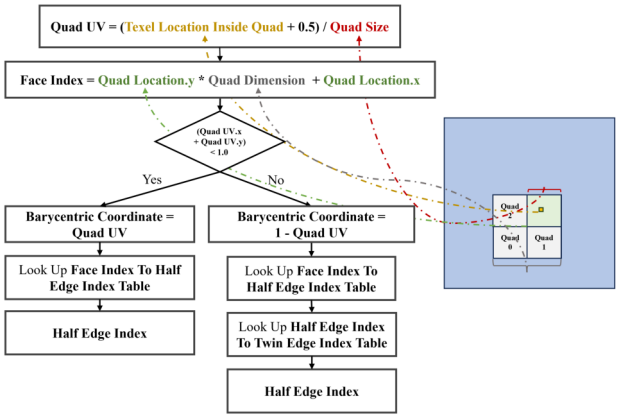


Figure 4: *A flowchart showing the process of preparing the shading context.*

3.2. Pre-Z and Shadel Marking

Similarly to the method of Baker et al. [BJ22], we mark visible shadels by set occupancy buffer during the pre-Z pass. For mipmapping, we need to compute gradients of Htex quad UV during this step so that we can derive correct mipmap levels and mark corresponding shadels.

3.3. Htex Virtualization

After successfully marking visible shadels, the subsequent step involves resolving the physical memory allocation for shadels. An allocation unit is a shadel group consisting of 8x8 shadel clusters arranged in a square grid. To efficiently allocate memory blocks for shadel groups, we employ a binned chunk allocator. If the memory pool runs out, an optional second memory allocation pass will be dispatched to add mipmap bias to shadel groups in mipmap layer 0.

3.4. Shading Pass

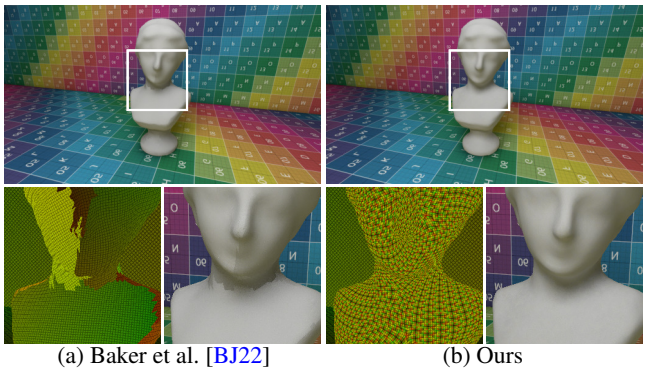
To dispatch the shading pass, we examine the occupancy map. If the corresponding occupancy bit is marked, the relevant shading task is enqueued into the task buffer.

For shading computation, we need to have the barycentric coordinate of the current triangle for vertex data reconstruction. We also need the current halfedge and the quad information for spatial neighbor searching. The process of the shading context preparation is summarized in Figure 4. We map an 8x8 thread group to a shadel to maximize data coherency.

After the shading pass, final gathering can be done at any time as long as users can provide primitive index and barycentric coordinate, which are necessary to construct the Htex sampling context.

4. Implementation and Results

We have implemented ReSTIR GI in our GPU-driven object space rendering system using Unity Engine [Haa14]. We did not implement bias correction for ReSTIR, so our results are biased.



(a) Baker et al. [BJ22]
Render time: 1.9 ms
1 sample per pixel

(b) Ours
Render time: 2.0 ms
1 sample per pixel

Figure 5: *Comparison of doing spatial resampling based on UV map (Baker et al. [BJ22]'s method) and adjacency information (our method). Notice that seam artifacts are inevitable when using a UV-based spatial kernel.*

All methods we compare against are also implemented within the same rendering framework, using the same functions and data structures when possible. The performance results are obtained on a computer with a 13900K CPU with 24 cores and 32 GB RAM, and an NVIDIA RTX 4090 GPU.

4.1. Spatial Searching

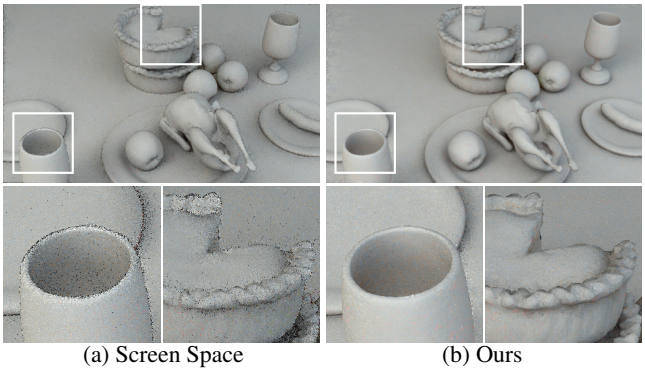
Spatial searching is essential for accessing broader reservoir samples from neighboring geometry. Baker et al. [BJ22]'s solution uses random walks on the UV map for this purpose. However, it is challenging for the UV map to be continuous in all areas, and this discontinuity in sample domains will cause discontinuity of bias and convergence, resulting in visible seams in rendering results. Our method leverages face-specific adjacency data to enable accurate spatial search, as illustrated in Figure 5, eliminating seam artifacts present in the method from Baker et al. [BJ22]

4.2. Elimination of Shading Sample Re-projection

By indexing samples with object space information, each shading sample at the same object space location uses a unique address. This eliminates the necessity for re-projecting spatiotemporal shading samples using motion vectors. As illustrated in Figure 6, in contrast to our solution, screen space methods often result in undersampling effects due to the unsuccessful re-projection of samples.

4.3. Persistent Layers

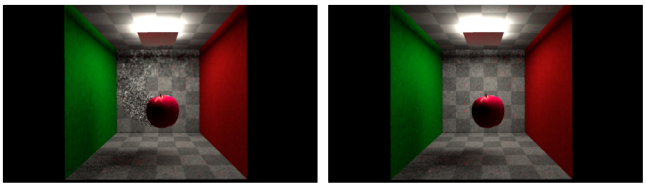
We implemented the finest level of mipmaps as a persistent layer, which is always present in physical memory. When spatial resampling fails to find a valid neighbor reservoir due to disocclusion, we fetch a reservoir from the persistent layer as a fallback reservoir. Figure 7 shows that employing fallback reservoirs is beneficial in enhancing the visual quality of disoccluded regions.



(a) Screen Space
Render time: 2.0 ms
1 sample per pixel

(b) Ours
Render time: 2.2 ms
1 sample per pixel

Figure 6: *Comparison of doing spatial resampling in screen space and texture space. We can find that for screen space solutions, the convergence is poorer at the rim of the cup and the crimped pie crust.*



(a) Without Persistent Layers
Render time: 2.1 ms
1 sample per pixel

(b) With Persistent Layers
Render time: 2.2 ms
1 sample per pixel

Figure 7: *Comparative impact of employing persistent layers on fallback reservoirs. An apple moves quickly along a circular path. The activation of persistent layers leads to a significant improvement in convergence efficiency in areas of disocclusion.*

5. Conclusion

We present a novel technique for using Htex texture mapping as shading space so that spatial kernels can be evaluated in a seam-free way. To minimize over-shading, we introduce virtualization to Htex. To validate our object space shading system, we implement ReSTIR GI on it. Our ReSTIR GI implementation does not require motion vector based screen space history reprojection so that samples can be streamed or accumulated stably. On top of that, combining fallback reservoirs from persistent layers helps us get better convergence in disoccluded regions.

6. Acknowledgments

We would like to thank our colleagues from TiMi L1 Studio, Jun Deng and Fei Ling, for their project support.

References

[AHTA14] ANDERSSON, M., HASSELGREN, J., TOTH, R., and AKENINE-MÖILER, T. "Adaptive Texture Space Shading for Stochastic Rendering". *Comput. Graph. Forum* 33.2 (May 2014), 341–350. ISSN: 0167-7055. DOI: 10.1111/cgf.12303. URL: https://doi.org/10.1111/cgf.12303 2.

[BD22] BARBIER, WILHEM and DUPUY, JONATHAN. "Htex: Per-Halfedge Texturing for Arbitrary Mesh Topologies". *Proc. ACM Comput. Graph. Interact. Tech.* 5.3 (July 2022). DOI: 10.1145/3543868. URL: https://doi.org/10.1145/3543868 2.

[BJ22] BAKER, DANIEL and JARZYNSKI, MARK. "Generalized Decoupled and Object Space Shading System". *Eurographics Symposium on Rendering*. Ed. by GHOSH, ABHIJEET and WEI, LI-YI. The Eurographics Association, 2022. ISBN: 978-3-03868-187-8. DOI: 10.2312/sr.20221163 1–3.

[BJW21] BOKSANSKY, JAKUB, JUKARAINEN, PAULA, and WYMAN, CHRIS. "Rendering Many Lights with Grid-Based Reservoirs". *Ray Tracing Gems II*. Ed. by MARRS, ADAM, SHIRLEY, PETER, and WALD, INGO. APress, Aug. 2021, 351–365. DOI: 10.1007/978-1-4842-7185-8_23 2.

[BL08] BURLEY, BRENT and LACEWELL, DYLAN. "Ptex: Per-Face Texture Mapping for Production Rendering". *Eurographics Symposium on Rendering 2008*. 2008, 1155–1164 2.

[BWP*20] BITTERLI, BENEDIKT, WYMAN, CHRIS, PHARR, MATT, et al. "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting". *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39.4 (July 2020). DOI: 10/gg8xc7 2.

[CCC87] COOK, ROBERT L., CARPENTER, LOREN, and CATMULL, EDWIN. "The Reyes Image Rendering Architecture". *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, 95–102. ISBN: 0897912276. DOI: 10.1145/37401.37414. URL: https://doi.org/10.1145/37401.37414 2.

[Haa14] HAAS, JOHN K. "A history of the unity game engine". (2014) 3.

[LKB*22] LIN, DAQI, KETTUNEN, MARKUS, BITTERLI, BENEDIKT, et al. "Generalized Resampled Importance Sampling: Foundations of ReSTIR". *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2022)* 41.4 (July 2022), 75:1–75:23. ISSN: 0730-0301. DOI: 10.1145/3528223.3530158. URL: https://doi.org/10.1145/3528223.3530158 2.

[OLK*21] OUYANG, Y., LIU, S., KETTUNEN, M., et al. "ReSTIR GI: Path Resampling for Real-Time Path Tracing". *Computer Graphics Forum* 40.8 (2021), 17–29. DOI: https://doi.org/10.1111/cgf.14378. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14378. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14378 2.

[TDD*22] TATARCHUK, NATALYA, DUPUY, JONATHAN, DELIOT, THOMAS, et al. "Advances in Real-Time Rendering in Games: Part I". *ACM SIGGRAPH 2022 Courses*. SIGGRAPH '22. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2022. ISBN: 9781450393621. DOI: 10.1145/3532720.3546895. URL: https://doi.org/10.1145/3532720.3546895 1.

[TYL17] TARINI, MARCO, YUKSEL, CEM, and LEFEBVRE, SYLVAIN. "Rethinking Texture Mapping". *ACM SIGGRAPH 2017 Courses*. SIGGRAPH '17. Los Angeles, California: Association for Computing Machinery, 2017. ISBN: 9781450350143. DOI: 10.1145/3084873.3084911. URL: https://doi.org/10.1145/3084873.3084911 2.

[YKH10] YUKSEL, CEM, KEYSER, JOHN, and HOUSE, DONALD H. "Mesh colors". *ACM Transactions on Graphics* 29.2 (2010), 15:1–15:11. ISSN: 0730-0301. DOI: 10.1145/1731047.1731053. URL: http://doi.acm.org/10.1145/1731047.1731053 2.