A Scalable Real-Time Many-Shadowed-Light Rendering System

Bo Li
racingpht@gmail.com
Warner Bros. Games
Montréal, Canada

Figure 1: Epic Game's SunTemple Rendered in Our System

ABSTRACT

In this paper, we present a new shadow rendering system, with a number of novel design to support large amount of shadowed lights in a large virtual environment, with real-time performance on mainstream GPUs.

CCS CONCEPTS

• **Computing methodologies** → **Rasterization**; *Image compression*; Vector / streaming algorithms.

KEYWORDS

rendering, compression, vector quantization, compute shader

ACM Reference Format:
Bo Li. 2019. A Scalable Real-Time Many-Shadowed-Light Rendering System. In *Proceedings of SIGGRAPH '19 Talks.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3306307.3328167

SIGGRAPH '19 Talks, July 28 - August 01, 2019, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6317-4/19/07.
https://doi.org/10.1145/3306307.3328167

1 INTRODUCTION

Supporting lots of shadow-casting lights in a virtual environment remains a challenging problem even with modern state of the art games engines and rendering systems[Eisemann et al. 2011]. Popular game engines such as UnrealEngine4 currently does not scale well beyond a dozen dynamic shadowed lights on current-gen consoles. Many AAA game engines also poses restrictions on how shadowed light should be placed due to technical limitations such as overlapping light channels and performance reasons. Baking methods such as light-map creates even more limitations in the lighting workflow, with either no dynamic interaction or handled with a huge performance impact. Furthermore, light-maps doesn't support ray-marching or other view-dependant volumetric effects.

In previous works[Doghramachi 2015][Olsson et al. 2014], algorithms were developed to support many shadowed lights. However, they are tested in idealized environments such as demo scenes, optimized for movable lights updating every frame. In our case however, we need to support production quality assets with more limited computation budget thus we can't afford updating all shadow buffers each frame. After all, fully dynamic light movements are rarely a major feature for games and visualization applications. We developed a new shadow rendering system specifically targeting the typical scenario that many stationary shadow-casting light with many static meshes, and dynamic objects moving around them. With a number of novel designs, our new system meets our performance and quality target.

2 OUR SYSTEM

2.1 Overview

One of the most important goal of our system is to cache static part of the shadow as much as possible. Each shadow-casting light needs to keep track of 3 major types of interactions: static meshes, vertex-animated meshes with static bounding box, and fully-dynamic movable meshes. With each of these interactions, different strategies such as update frequency, culling/fading distance and target resolution are applied respectively. Each light manages and updates 3 textures when necessary(Figure 3): static shadow buffer, dynamic shadow buffer with its Dirty-Mask texture. If there's no dynamic interaction, only static shadow buffer is allocated for the light. Otherwise, dynamic shadow buffer and its Dirty-Mask texture are generated in a single render-pass. Dirty-Mask textures are used to reduce shadow filtering cost by only filtering both static and dynamic shadow buffers where corresponding Dirty-Mask texels are set.

To improve scalability, a view-dependant updating frequency for each light is generated by heuristics based on screen-projected size, brightness, visibility and a user-defined resolution slider, so that CPU and GPU computational resources are allocated proportionally to its final image contribution in a time-sliced manner. Also, for each light, shadow maps are assigned in a pre-allocated pyramid-structured textures pool with increasing number of texture slots at lower resolutions, by a pre-defined pixel-texel ratio quality target. With this method, the amount of memory required are roughly determined by number of pixels multiplied by pixel-texel ratio and average number of overlapping shadows, regardless of scene complexity. Run-time texture allocations are avoided in most cases.

The shadow buffers are then packed into uniform buffers of flat texture descriptors arrays for fast GPU access indexed from shaders. The advantages of this packing over texture-atlas or texture-arrays are free of restrictions on texture size, memory locations, texture formats, etc. The GPU then performs screen-tile culling optimized for high depth complexity scenes and generating tile filtering tasks with shadow indices for further deferred-shadow process. Spot-light and Point-lights are processed in separate dispatch-indirect passes to optimze VGPR-limited shader-core occupancy. The results of shadow filtering then packed into a compact, optionally compressed buffer for further deferred-shading and lighting. Because of the flexible shadow buffer packing, other lighting effects such as single-pass local-shadowed volumetric fog are efficiently supported.

Two new compression algorithms are presented. First is variable-bit-rate shadow map compression to help removing static-shadow buffer generation pass. Second, a GPU-based TSVQ (Tree-Structured-Vector-Quantization) algorithm compresses deferred-shadow light masks, to support many overlapping lights per-pixel with minimal memory and bandwidth.

2.2 Depth Buffer Compression

In order to support as many static shadow buffers as possible with minimal memory footprint, we introduce an innovative GPU-based adaptive Quad-Tree compression with high throughput with a typical compression ratio of 30:1 with very slight quality losses. A macro-block size of 32x32 pixel is used to divide the buffer and depth planes are generated in the block to reduce floating-point errors. Blocks are then compressed independently in **groupshared** memory to minimize bandwidth consumption.

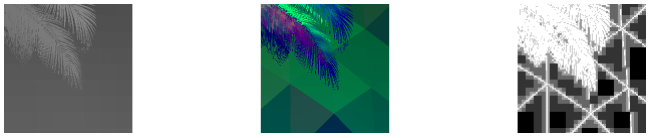

Figure 2: Adaptive Shadow Depth Compression. Left: Raw Depth, Middle: Depth Plane, Right: Compressed Tree

Figure 3: Example shadow buffers of flexible resolutions for a spot light with dynamic interactions. From Left to Right: Dirty-Mask(64x64), Static shadow buffer(512x512), Dynamic shadow buffer(2048x2048), Equivalent shadow buffer

2.3 Tree-Structured-Vector-Quantization for per-pixel shadow mask

Occupancy is key for high GPU performance, thus we choose Deferred-Shadow to separate Lighting and Shadowing passes. Traditionally Deferred-Shadow requires a lot of memory to store intermediate mask for each light channel per-pixel. We developed a novel GPU-based Vector-Quantization algorithm to compress the output. By heavily using **msad** and **LaneSwizzle** instructions, optimized into an unrolled branch-free inner loop, we achieves 0.75bit/pixel and high throughput on GPU shader cores.

2.4 Performance Results

While performance is highly data-dependant, with abovementioned design and optimizations, we achieved significant performance improvement over stock Unreal-Engine 4 by up to 10x in shadow rendering performance, and we are able to support over a thousand shadow-casting lights in a complex virtual environment in real-time.

ACKNOWLEDGMENTS

To Tech-Director Art Zaratsyan for supporting this talk and corrections. Thanks to Rendering-Lead Jimmy Béliveau for pushing optimization ideas further.

REFERENCES

Hawar Doghramachi. 2015. *GPU Pro 6: Advanced Rendering Techniques.* CRC Press.

Elmar Eisemann, Michael Schwarz, Ulf Assarsson, and Michael Wimmer. 2011. *Real-Time Shadows* (1st ed.). A. K. Peters, Ltd., Natick, MA, USA.

Ola Olsson, Markus Billeter, and Emil Persson. 2014. Efficient Real-Time Shading with Many Lights. In *SIGGRAPH Asia 2014 Courses (SA '14).* ACM, New York, NY, USA, Article 11, 310 pages. https://doi.org/10.1145/2659467.2659475