Sparse Shadow Tree

Kevin Myers*
Treyarch

Abstract

Lighting large outdoor scenes continues to present a challenge for realtime rendering. While techniques such as Parallel Split Shadow Maps [Zhang 2006] work well for a subset of the view frustum they fail to account for all shadowing in an outdoor scene. Lightmaps are often used as a fallback but require a unique UV parameterization and do not provide occlusion for moving objects. With deferred rendering the additional gbuffer overhead for unique parameterizations presents a challenge as well, making it costly to maintain the lightmap machinery in a modern pipeline.

Keywords: shadows, shadow compression, sparse tree

Concepts: •**Computing methodologies** → **Visibility;**

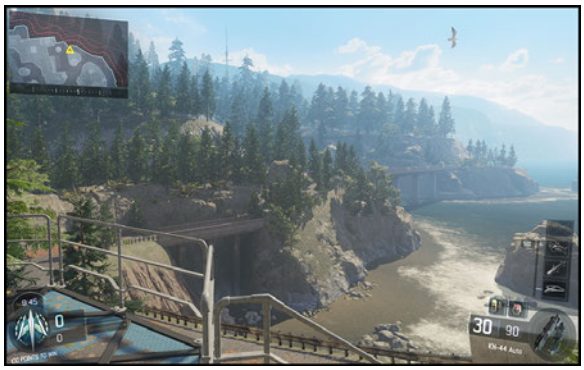

Figure 1: *Treyarch SST full world shadowing*

1 Motivation

We want a far shadowing solution that transitions seamlessly to a canonical split shadowing system for the mid-ground while providing occlusion for moving objects. In our research, we considered a voxel representation of visibility [Sintorn 2014] but determined it would be useful to preserve the original depth for use in other effects. Shadowmaps are ideal as they do not require a unique parametrization and store occluder depth. For these reasons we developed a shadowmap compression technology that compresses/decompresses depth images.

*e-mail:kmyers@treyarch.com

SIGGRAPH '16, July 24-28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4282-7/16/07
DOI: http://dx.doi.org/10.1145/2897839.2927418

2 Shadow Compression

We generate a global shadowmap on the GPU and then compresses it on the CPU via a recursive quad-tree process. Our method aims to convert the shadowmap into a hierarchy of planes. This hierarchy consists of a forest of sparse quad-trees where each quad-tree represents a tile of 128x128 shadow map. Compression occurs when a plane is able to approximate all the depth values in a given node. To facilitate higher compression ratios we optionally generate a second depth layer to optimize plane tolerances. Instead of merely biasing the shadow [Weiskopf 2003] this second layer improves compression by generating fewer and larger planes Figure 2.

2.1 Point Cloud Fitting

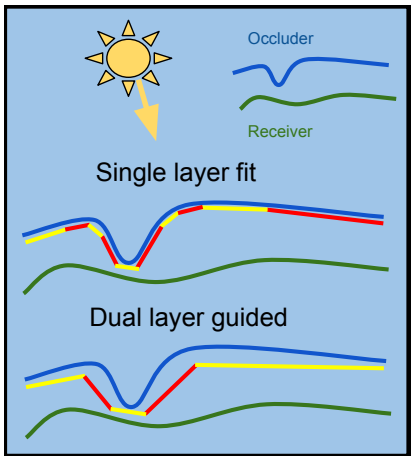

Figure 2: *Plane fitting*

At each level of the quad-tree our algorithm generates a 3D point cloud from the projected depths. We then attempt to fit a linear system to the point cloud such that solving the resulting plane equation reproduces depths that are biased within the bounds calculated by the two depth layers. If the fit is good enough the tree terminates otherwise we subdivide and continue the search.

2.2 Encoding

A simple encoding is important for efficient GPU decoding. To minimize costly branching, each node of the quad-tree encodes either a plane or the 4 relative offsets to child nodes.

	1st DWORD			2nd DWORD
	Control			
Parent node	0	0	Indices	Indices
62-bit plane	1	0	30-Bit z offset	16-bit x 16-bit y
30-bit plane	0	1	30-Bit z offset	Garbage

Figure 3: *Encoding*

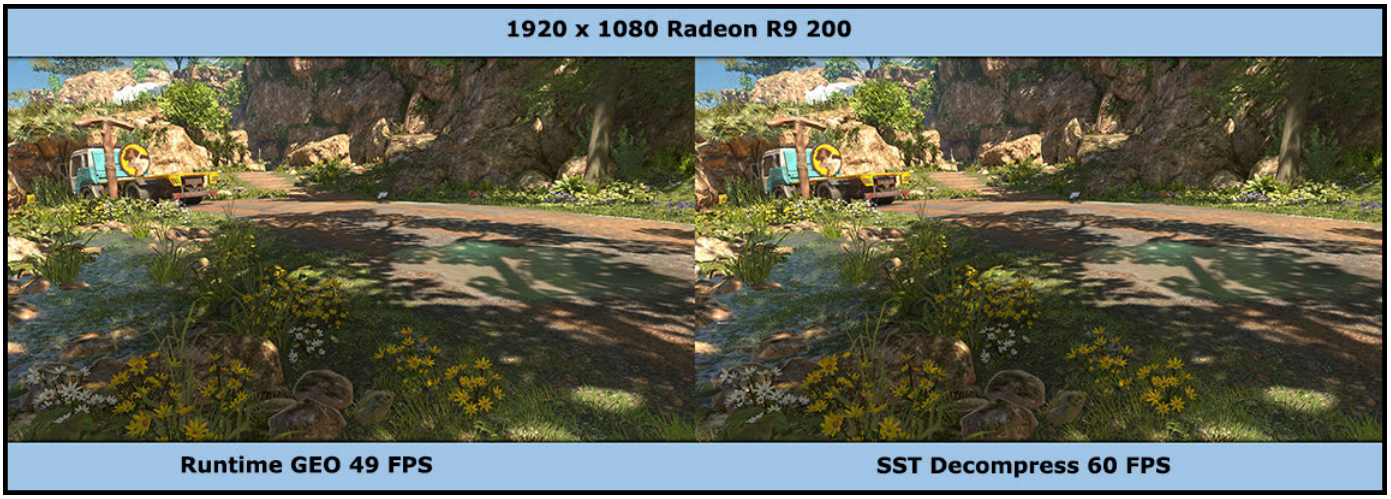

Figure 4: *Decompression perf*

Planes are always encoded at 62-bit precision with an additional control bit flagging planes that are uniform depth. When the plane is decoded the slope is ignored if the plane is flagged as uniform. This avoids divergence in the tree walk while halving the storage space of a plane with no slope. Relative offsets are either 13-bit or 10-bit depending upon their depth in the tree.

Since each 128x128 tree can be encoded in parallel, compression is very fast typically taking less than 1 minute on 6-core Intel x5690. As a post process, the individual quad-trees are gathered into a forest indexed by a 2D grid.

3 GPU Decompression

To facilitate filtering and provide a seamless transition to dynamic shadows we decompress all static occluders from SST into a parallel-split shadow map. Dynamic geo is then rendered over the decompressed depth. We directly read the SST when calculating lighting for all fragments that fall outside of the split distance by projecting shaded fragments onto the 2D grid that indexes the forest. We then use a morton code to quickly traverse the selected tree, retrieving the shadow depth.

3.1 Runtime Performance

Using SST to store static shadow casting geometry provides a significant performance improvement over runtime geometry processing. Typical improvements range from 10% to 20% from reduced vertex processing, no overdraw and bypassing the GPU ROP unit. Additionally on asynchronous compute platforms such as DX12 decompression can occur in parallel with graphics work as the decompression is handled by a compute shader.

Figure 5: *SST Shadows*

References

SINTORN, E., KAMPE, V., OLSSON, O., AND ASSARSSON, U. 2014. Compact precomputed voxelized shadows. *ACM Transactions on Graphics 33*, 150 (July).

WEISKOPF, D., AND ERTL, T. 2003. Shadow mapping based on dual depth layers. *Proceedings of Eurographics 3*, 53–60.

ZHANG, F., SUN, H., XU, L., AND LUN, L. K. 2006. Parallel-split shadow maps for large-scale virtual environments. *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications* (June), 311–318.