Iterative GPU Occlusion Culling with BVH

Gi Beom Lee
Sungkil Lee*
sungkil@skku.edu
Sungkyunkwan University
Suwon, South Korea

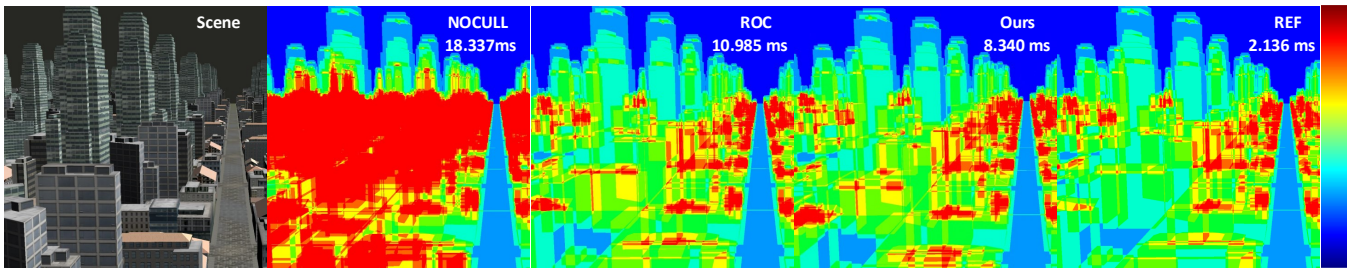

Figure 1: Comparison of our iterative occlusion culling (IOC) with no culling (NOCULL), raster occlusion culling (ROC), and precomputed ideal culling (REF). The blue-to-red color mapping indicates low-to-high fragment overdrawing. Culling efficiency of our IOC is lower than ROC, but its efficient batch culling eventually improves overall rendering performance.

CCS CONCEPTS

• **Computing methodologies** → **Visibility**.

KEYWORDS

real-time rendering, occlusion culling, bounding volume hierarchy

ACM Reference Format:
Gi Beom Lee and Sungkil Lee. 2020. Iterative GPU Occlusion Culling with BVH . In *Proceedings of High-Performance Graphics (Washington, D.C. '20).* ACM, New York, NY, USA, 2 pages. https://doi.org/xx.xxxx/xxxxxxx.xxxxxxx

1 BACKGROUND

Occlusion culling accelerates scene rendering by bypassing objects hidden by potentially visible occluders. Usually, what are seen in the past frame can be selected as the occluders, and their depth rendering is compared against the coarse bounding proxies of potential occludees. Occlusion query (OQ) in Graphics Processing Unit (GPU) is a typical technique. OQ counts the number of visible pixels to determine the rendering of the occludee, but its repeated read-back causes significant stalls in the rendering pipeline. The problem is alleviated using the scene hierarchy [3], but it does not scale well with a deeper hierarchy of massive scenes.

Raster occlusion culling (ROC) [1, 2] reduces the latency by directly marking visible objects during the rasterization, leveraged by early Z and unordered buffer access. Furthermore, ROC employs indirect multidraw commands to avoid GPU-to-CPU read-back, which significantly reduces the stalls. The technique is more efficient than the classical OQ, but does not scale well with a large number of objects due to the non-trivial amounts of fragment processing.

2 OUR APPROACH

In this poster, we present an iterative hierarchical ROC (IOC) technique that can scale up to massive scenes with higher geometry complexities. Unlike the original ROC, we do not handle individual objects, but use their hierarchical structures such as a Bounding Volume Hierarchy (BVH). The BVH is iteratively traversed from a moderate depth down to a deeper level (but not the bottom of the tree); see Figure 2. The interior nodes are occlusion-tested in batch. The granularity for the culling is coarser, but the light-weight occlusion test with fewer draw calls leads to a great speedup in the overall rendering performance.

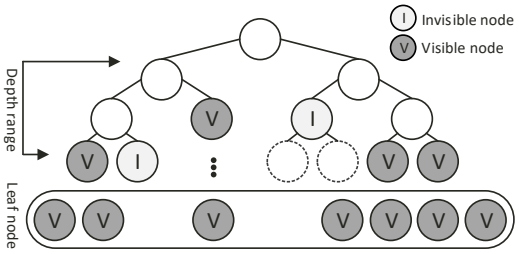

Figure 2: Iterative occlusion culling in the scene hierarchy.

*Corresponding author.

Washington, D.C. '20, July 16–18, 2020, Washington, D.C.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/xx.xxxx/xxxxxxx.xxxxxxx

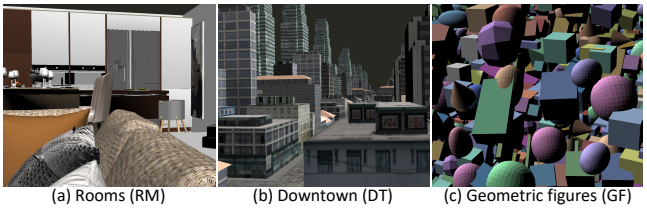
(a) Rooms (RM) (b) Downtown (DT) (c) Geometric figures (GF)

Figure 3: Three scenes used for experiments: rooms (2753 objects, 5k BVH nodes), downtown (31392 objects, 62k BVH nodes), and geometric figures (72000 objects, 143k BVH nodes). RM and DT model are provided through the courtesy of cgtrader.com.

The BVH for the occlusion culling is traversed in a top-to-bottom fashion as follows. We first pre-render the depth buffer of the occluders for early Z in the ROC, which are visible at the previous frame. Given a pair of starting and bottom depths, we perform ROC for the bounding boxes of all the interior nodes at the depth. If the node is culled, we stop the traversal for the node, and mark all the children in its subtrees as occluded in the visibility buffer (VB) defined in GPU. If the node is a leaf, we directly mark it as a potentially visible occludee in the VB. Otherwise, we tightly pack and enqueue them for the next occlusion test, and traverse deeper. The size of the queue is read back to the host memory to drive the indirect multidraw call of the next occlusion test. We iterate this down to the given bottom level. At the bottom level, all the children in the subtrees of the visible nodes are marked as potentially visible.

3 RESULTS

We implemented our solution on an Intel i7 machine with GTX 1080 Ti at 1920×1080. Three scenes (see Figure 3) are tested with animated camera sequences of 12s durations. The Rooms (RM) is a relatively simple scene. The Downtown (DT) is a wide-scale scene of a medium complexity, and the Geometric Figures (GF) is a scene of low-polygon objects with very high depth complexity. We compared out solution against the four existing techniques: no culling (NOCULL), pre-computed ideal culling (REF), view-frustum culling (VFC), and raster occlusion culling (ROC) [1, 2].

Table 1 compares our solution with ROC in terms of the culling cost. In the simple RM scene, the iterative test is not efficient as the ROC does. Our solution manifests itself in the large-scale scenes (DT and GF scenes). The hierarchical occlusion test significantly reduces the culling cost, while the coarse occlusion test adds only marginal rendering time for the occludees; ours is 4.5× and 7.6× faster than ROC for DT and GF scenes, respectively.

Figure 4 compares the overall performance, including culling as well as rendering, of our solution against NOCULL, REF, VFC, and ROC. In the RM scene, all the culling techniques does not show significant difference as expected. For DT and GF scenes, ours runs fastest with significant differences. In comparison, ROC runs even slower than VFC for DT, which results from the excessive rasterization overhead for the culling.

Table 1: Comparison of ours (IOC) with VFC and ROC, assessed in terms of the culling ratio and the culling cost for a short period (12s) of pre-defined camera animations.

scene		occlusion test (ms)	occludee rendering (ms)
RM	ROC	0.41	0.40
	IOC	0.61	0.40
DT	ROC	3.47	3.50
	IOC	0.78	3.58
GF	ROC	7.96	7.81
	IOC	1.05	8.04

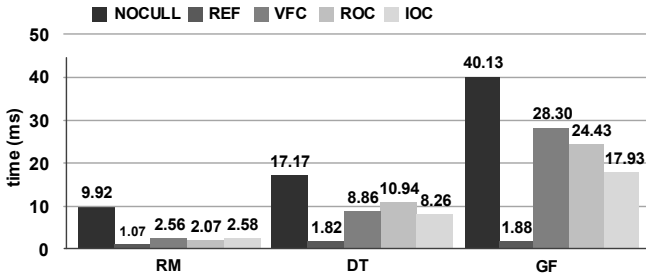

Figure 4: Performance comparison of rendering time for the pre-defined camera animations.

4 LIMITATIONS

One of the obvious limitations of our solution is the manual choice of the pair of the top and bottom levels in the hierarchy. This encourages further investigation on automatically finding a depth range for the iteration, reflecting the scene characteristics and statistics.

ACKNOWLEDGMENTS

This work was supported in part by the Midcareer and Advanced Convergence Technology Development (on Virtual Reality System for Personalized Mental Healthcare Contents) R&D programs through NRF grants funded by the Korea Government (MSIP) (Nos. 2019R1A2C2002449 and 2017M3C1B6070980).

REFERENCES

[1] Pierre Boudier and Cristoph Kubisch. 2015. GPU Driven Large Scene Rendering. In *Nvidia GPU Technology Conference*. 22–34.
[2] Christoph Kubisch and Markus Tavenrath. 2014. Opengl 4.4 scene rendering techniques. *NVIDIA Corporation* 5 (2014).
[3] Oliver Mattausch, Jiří Bittner, and Michael Wimmer. 2008. CHC++: Coherent hierarchical culling revisited. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 221–230.