Accelerating k^+-buffer using efficient fragment culling

Andreas A. Vasilakis* and Georgios Papaioannou†
Dept. of Informatics, Athens University of Economics & Business, Greece

1 Introduction

In the last decade, significant research has been conducted for addressing the problem of multi-fragment rendering from different perspectives. While the hardware-accelerated A-buffer is the dominant structure for holding multiple fragments via per-pixel linked lists, several variants have been proposed to alleviate the cost of excessive allocation and random access of video-memory. k-buffer is a widely-accepted A-buffer approximation, able to capture the *k-closest* to the viewer fragments, due to its reduced memory and computation requirements. To alleviate contention of distant fragments when rendering highly-complex scenes, k^+-buffer [Vasilakis and Fudos 2014] concurrently performs culling checks to efficiently discard fragments that are farther from all currently maintained fragments. Unfortunately, the fragment elimination process is performed inside the pixel shader execution, thus not exploiting the performance gain of hardware-accelerated early-Z culling. Furthermore, it depends on the fragment arrival order and requires the insertion of k fragments to start performing any culling tests.

2 Occupancy-based Fragment Culling

In this work, we investigate an efficient approach to treat fragment racing when computing k-nearest fragments. Based on the observation that knowing the depth position of the k-th fragment we can optimally find the k-closest ones, we introduce a novel order-independent fragment culling component, easily attached to the k^+-buffer pipeline. An additional rendering pass of the scene's geometry is initially employed to construct a per pixel binary fragment occupancy discretization. Then, the nearest depth of the k-th per pixel fragment is concurrently computed by performing bit counting operations and subsequently utilized to perform *early-z rejection* for the k^+-buffer construction process that follows. Any fragment with depth larger than this value will fail the depth test, avoiding the cost of its pixel shading execution. Note that no software modifications are required to the actual k^+-buffer implementation.

The depth range of each pixel p is divided into $B = 32 \cdot d$ uniform consecutive subintervals $[b_j, b_{j+1})$, where $b_j = p.near + \frac{j}{B}(p.far - p.near), j = 0, 1, \ldots, B-1$ and $d > k/32$ defines the depth space subdivision. A *depth-range* map is initially computed, containing for each pixel p the nearest ($p.near$) and farthest fragment ($p.far$) depth values from the camera. Then, a *depth occupancy* buffer is utilized to define a per-pixel bitmask B, whose entries indicate the presence of fragments in a subinterval. During the first geometry pass, the j-th bit of the depth map is set to 1 for each arriving fragment falling within the corresponding bucket via blending or atomic operations, depending on the hardware. Subsequently, a full-screen pass is performed to concurrently count the number of 1s in the bit-array for each pixel by adjusting *Brian Kernighan*'s algorithm. When the value of the counter reaches k, the current bucket's depth is returned to the Z-buffer. The average-case complexity of this pass is $O(k)$. Then, k^+-buffer is efficiently constructed by taking advantage of the early-z culling capabilities of the GPU. Note that in the case where more than one fragments are routed to the same subinterval and merged into one bit, our method is theoretically correct since we may have possibly stored the next larger fragment than k-th one.

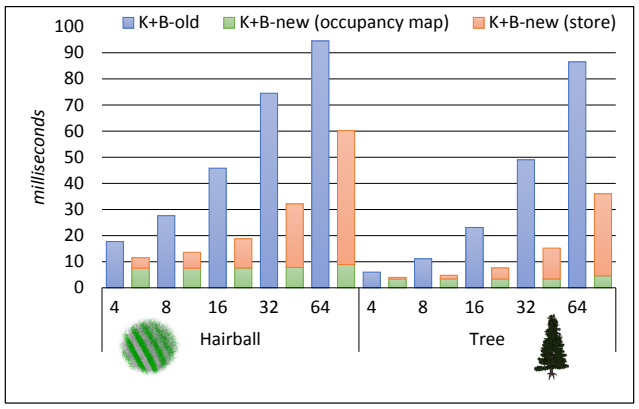

Figure 1: *Performance evaluation of the actual and modified k^+-buffer (K+B) under varying $k = 4, \ldots, 64$ values.*

3 Discussion

Figure 1 illustrates the performance increase when the proposed fragment clipping with $d = 32$ is enabled on the k^+-buffer. Despite the additional geometry passes needed, performance increases by 20% to 50%, when rendering the *hairball* (2.8M, 150) and *needle tree* (43.2T, 100) models (# triangles, average depth complexity) with a set of increasing $k = 4, \ldots, 64$ values at 1024^2 resolution on an NVIDIA GeForce GTX780 Ti. Without loss of generality, we may reuse occupancy buffer for capturing information of the actual k^+-buffer, since the k-th fragment computation is performed before the final rasterization. Thus, our extension requires specifically $(d - k + 3) \cdot 32$ bits additional per pixel storage. However, when moving to extreme screen or occupancy map resolutions, this cost is noticeable.

Acknowledgements. This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: ARISTEIA II-GLIDE (grant no.3712).

References

VASILAKIS, A. A., AND FUDOS, I. 2014. k^+-buffer: Fragment synchronized k-buffer. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, USA, I3D '14, 143–150.

*abasilak@aueb.gr
†gepap@aueb.gr

I3D 2015, February 27 – March 01, 2015, San Francisco, CA.

ACM 978-1-4503-3392-4/15/02 $15.00
http://dx.doi.org/10.1145/2699276.2721402