Accelerating k^+-buffer using Efficient Fragment Culling

Andreas A. Vasilakis and Georgios Papaioannou
{abasilak, gepap}@aueb.gr
Department of Informatics, Athens University of Economics & Business, Greece

Abstract. Visibility determination is a standard stage in the pipeline of numerous applications (from visualization to content creation tools) that require accurate processing of *out-of-order generated* fragments at interactive speeds. While the hardware-accelerated **A-buffer** [1] is the dominant structure for holding multiple fragments via per-pixel linked lists, ***k*-buffer** [2] is a widely-accepted 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** [3] concurrently performs culling checks to efficiently discard fragments that are farther from all currently maintained fragments. Inspired by ***fragment occupancy maps*** [4], we introduce an efficient fragment culling mechanism for accelerating k^+-buffer method.

k^+-buffer Fragment Culling Mechanism [3]

- Concurrently discards an incoming fragment that is farther from all currently maintained fragments (guided by the max element).

Limitations

1. Depends on the fragment arrival order, with no impact at the worst case scenario of fragments arriving in descending order.
2. Requires the k^+-buffer to be initially filled before it starts culling.
3. Fragment elimination is performed inside the pixel shader (not hardware-accelerated)

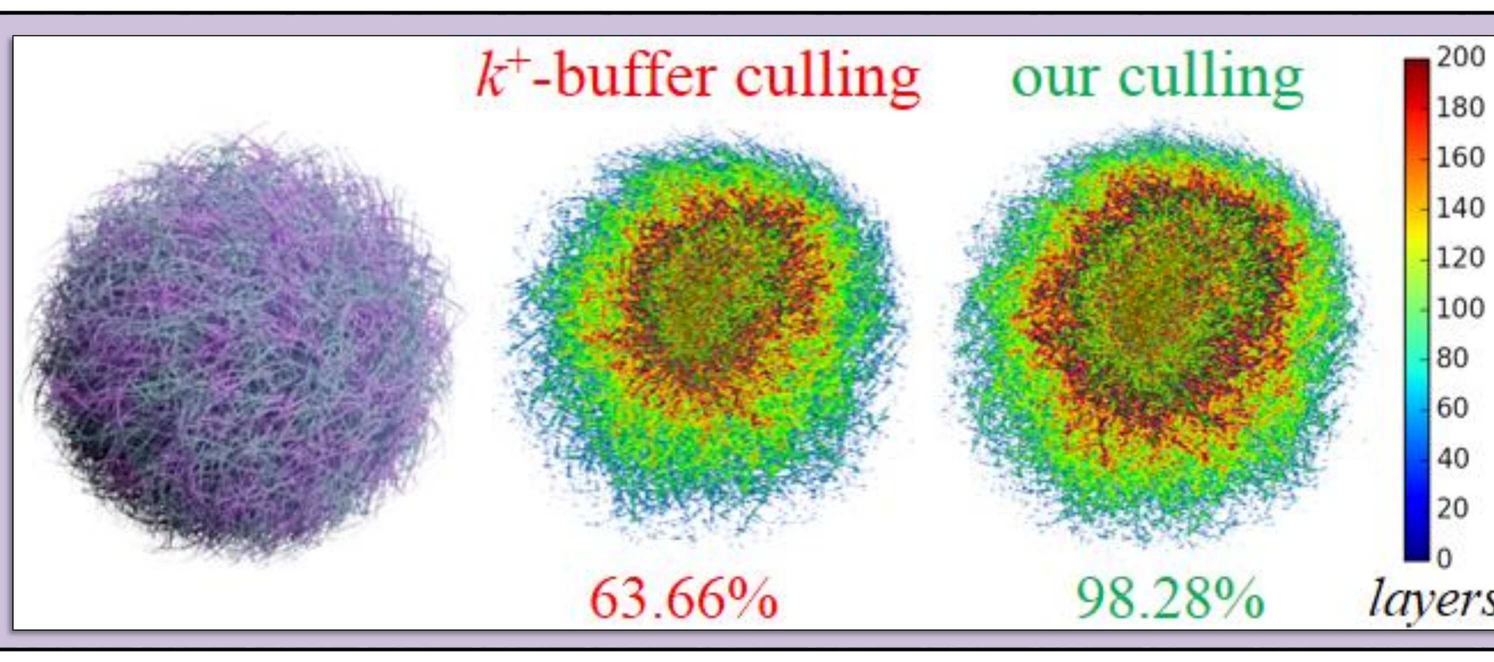


Figure 1. Notice the massive increase of fragments discarded, visualized as heatmap, of our culling mechanism (right) compared to its predecessor (left) when rendering the *Hairball* model (180 layers, k = 8).

Occupancy-based Fragment Culling Mechanism

- Performs ***early-z culling*** with ***k_a-th fragment*** per pixel, ***nearest largest to the actual k-th ($k_a \geq k$)***
1. Depth range is divided into B uniform consecutive subintervals [4].
2. Occupancy bitmask, indicates the presence of fragments in each subinterval [4].
3. Counts the number of 1s in bitmask until you reach k value ($O(k)$ time).
4. Efficiently discards fragments with larger depth value than the k_a-th fragment.

Algorithm

① {depth$_{MIN}$, depth$_{MAX}$} ← RenderBoundingBox (); [BLENDING or ATOMIC]
② occupancyMap ← RenderScene (depth$_{MIN}$, depth$_{MAX}$); [BLENDING or ATOMIC]
③ depth$_k$ ← FullScreenQuad (depth$_{MIN}$, depth$_{MAX}$, occupancyMap, k);
④ k^+-buffer ← RenderScene (k); [DEPTH_TEST(LEQUAL,depth$_k$)]
⑤ Final Image ← FullScreenQuad (k^+-buffer);

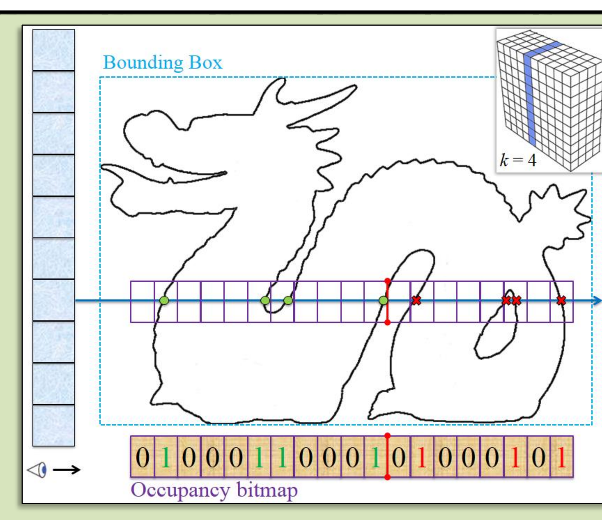
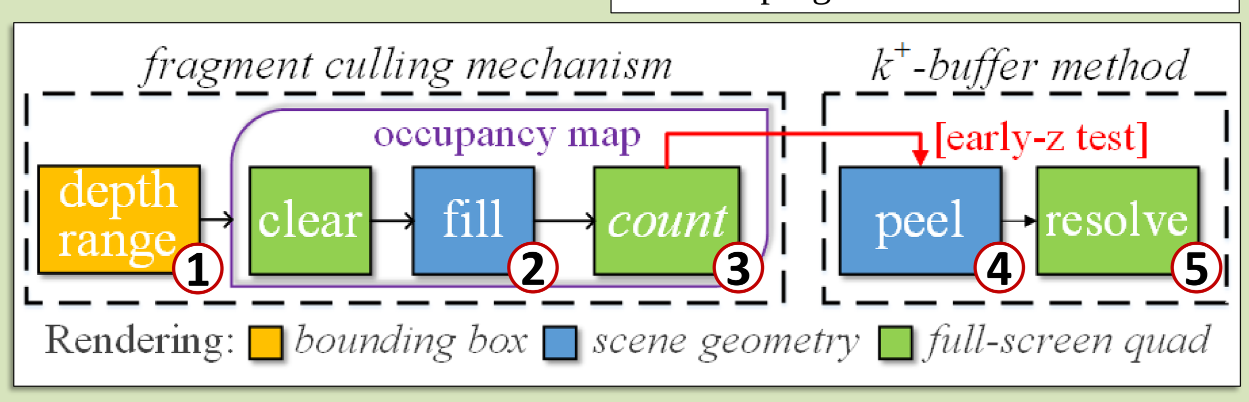


Figure 2. The fragment occupancy bitmask construction process of a column of a 4-buffer (highlighted with blue at top-right), when applied to the *dragon* model. Fragments with depth larger than the *k*-th fragment (red-colored line) are efficiently discarded.

Figure 3. Diagram of extending k^+-buffer pipeline. Each box represents a shader program.



Discussion

Results

Figure 4 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,…, 64 values at 1024^2 resolution on an NVIDIA GeForce GTX780 Ti.

Figure 5 illustrates the transparency results of an *ancient Greek temple* $(123K, 8)$ model for different values of $k = \{2,4,8,16\}$.

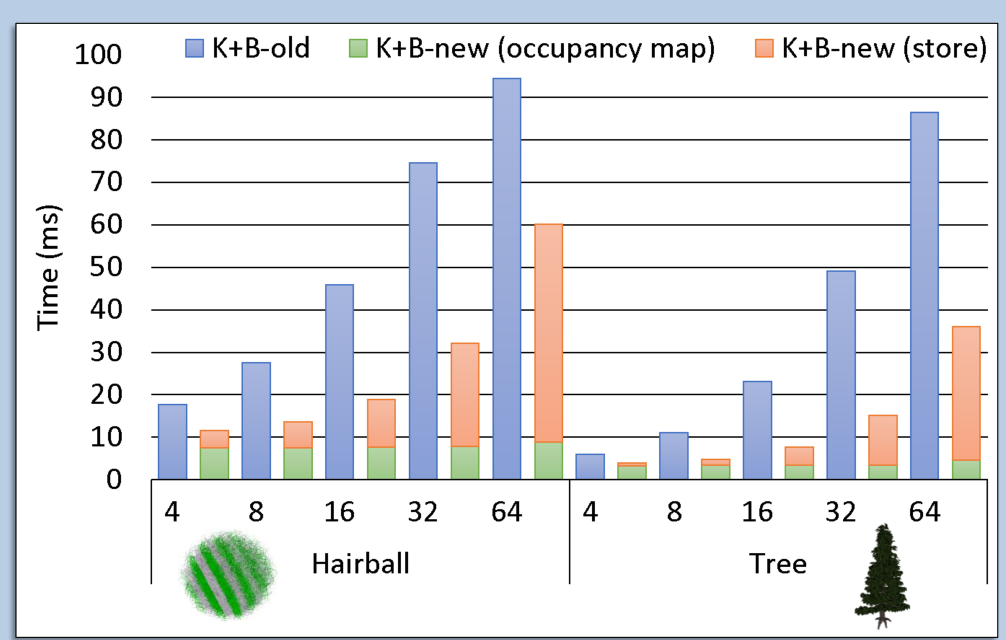


Figure 4. Performance evaluation of the actual and the modified k^+-buffer (**K+B**) under varying k values.

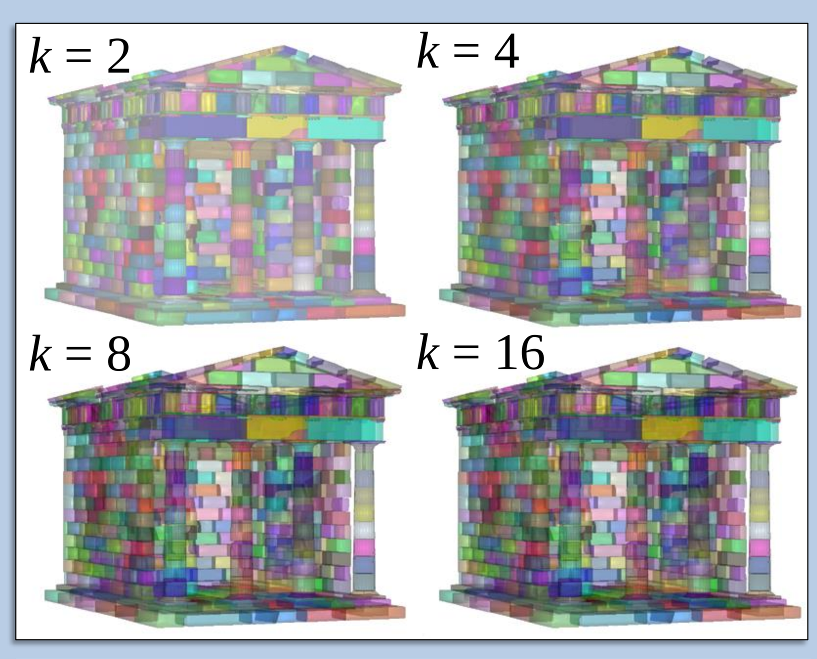


Figure 5. Transparency quality improvement by increasing k when rendering a *temple* model.

Advantages

- Works correctly even when fragments > 1 are routed to same bucket.
- Does not require any software modification of the actual k^+-buffer.

Limitations

- Works well only when the generated per-pixel fragments $n \gg k$.
- Requires additional per pixel storage for the fragment occupancy map.

Future Work

1. The idea can be easily extended to any other *k*-buffer alternative.
2. Memory-friendly representation can be implemented by reusing the occupancy buffer for storing color information of the actual *k*-buffer.
3. Replace bounding box with a better approximation (e.g. convex hull)

Accepted as Short Paper at EG 2015 [5]

References

[1] J. C. Yang, J. Hensley, H. Grün and N. Thibieroz. 2010. *Real-Time Concurrent Linked List Construction on the GPU*. Computer Graphics Forum, 29: 1297–1304.
[2] L. Bavoil, S. P. Callahan, A. Lefohn, J. L. D. Comba, and C. T. Silva. 2007. *Multi-fragment effects on the GPU using the k-buffer*. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '07), pages 97-104.
[3] A. A. Vasilakis and I. Fudos. 2014. *k^+-buffer: fragment synchronized k-buffer*. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '14), pages 143-150.
[4] F. Liu, M.-C. Huang, X.-H. Liu, and E.-H. Wu. 2009. *Efficient depth peeling via bucket sort*. In Proceedings of the Conference on High Performance Graphics (HPG '09), pages 51-57.
[5] A. A. Vasilakis and G. Papaioannou. 2015. *Improving k-buffer methods via Occupancy Maps*. In Proceedings of Eurographics 2015, Short Papers, Zurich, Switzerland, May 4-8, 2015.

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).