Improving *k*-buffer methods via Occupancy Maps

Andreas A. Vasilakis [†] and Georgios Papaioannou

Dept. of Informatics, Athens University of Economics & Business, Greece

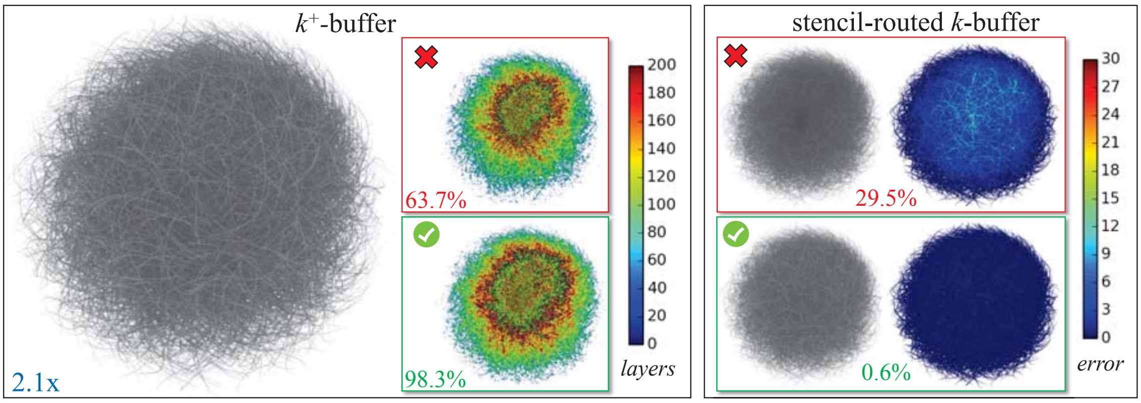

Figure 1: *(Left) k^+-buffer [VF14] is highly boosted by exploiting our fragment culling extension, when rendering the Hairball model (180 max layers, $k = 8$). Notice the massive increase of fragments discarded, visualized as heatmap, of our mechanism (bottom) compared to its predecessor (top). (Right) Observe the significant quality enhancement (see error heatmaps) of the original stencil-routed k-buffer [BM08] (top) when culling tests are enabled (bottom), as compared with the ground truth.*

Abstract

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 fragments, we introduce a novel fragment culling component by employing occupancy maps. Without any software-redesign, the proposed scheme can easily be attached at any k-buffer pipeline to efficiently perform early-z culling. Finally, we report on the efficiency, memory space, and robustness of the upgraded k-buffer alternatives providing comprehensive comparison results.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Hidden line/surface removal

1. Introduction

Multi-fragment visibility determination is a standard stage in developing numerous effects for interactive 3D games and graphics applications. Order independent transparency for deferred [BM08, HBT14] and forward rendering [MCTB11] can be considered typical examples of such applications that also comprise benchmark cases for testing multi-fragment processing frameworks for scenes with high depth complexity (e.g. flow data [GSM*14] or hair geometry [YYH*12]). Traditionally, a family of GPU-accelerated buffers is responsible on treating the problem of storing the out-of-order in-

[†] abasilak@aueb.gr

DOI: 10.2312/egsh.20151017

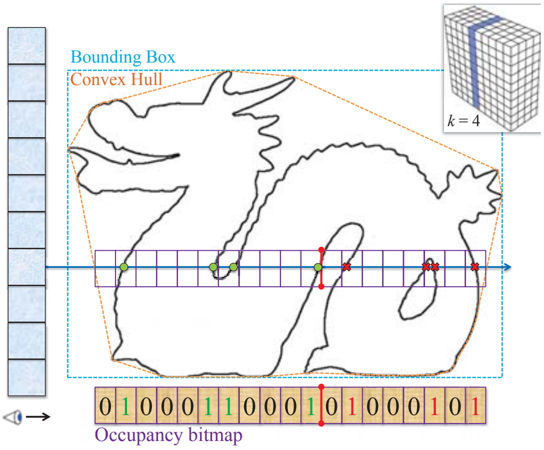

Figure 2: *The occupancy bitmap construction process of a column of a 4-buffer (highlighted with blue at top-right), when applied to the dragon model. Fragments with depth larger than the k-th fragment are efficiently discarded.*

coming fragments generated by the rasterization pipeline (see Fig. 2). While the GPU-accelerated A-buffer [YHGT10, VF12] is the dominant structures for holding multiple fragments via variable-length per-pixel structures, several alternatives have been proposed to alleviate the cost of excessive allocation and access of video-memory [MCTB11].

k-buffer [BCL*07] as well as its stencil-routed version [BM08] are widely-accepted A-buffer approximations, capable of capturing the k-*closest* fragments to the viewer by employing fixed-size per-pixel vectors on the GPU. Despite their reduced memory demands when compared to A-buffer solutions, both techniques suffer more or less from *read-modify-write hazards* (RMWH) caused when the generated fragments are inserted in arbitrary depth order. To this end, an abundance of k-buffer variants have been recently introduced aiming at eliminating the disturbing dotted or heavily speckled surface areas that result from the aforementioned problem, by performing atomic operations [MCTB13], constructing an auxiliary A-buffer [SML11], on the fly sorting fragment lists [YYH*12], or pixel synchronization mechanisms [Sal13, VF14]. However these modifications come with the cost of additional computation and memory requirements or the requirement of specialized hardware.

While exploiting hardware-accelerated pixel synchronization [Sal13, BH14] ensures accurate k-buffer construction, its performance degrades rapidly due to the heavy fragment contention of accessing the critical section when rendering highly-complex scenes. To this end, k^+-buffer [VF14] significantly alleviates fragment racing by concurrently performing efficient culling checks to discard fragments that are farther from all currently maintained fragments. Unfortunately, the proposed fragment rejection process has several limitations. First of all, the process initially requires the insertion of k fragments before it starts performing any culling tests. Secondly, it depends on the depth order of the incoming fragments, with no impact at the worst case scenario of fragments arriving in descending order. Last but not least, the actual fragment elimination is unfortunately performed inside the pixel shader execution, thus not exploiting the performance gain of hardware-accelerated *early-Z culling*.

In this work, we introduce an efficient order-independent fragment culling mechanism that can be easily used with any k-buffer framework to alleviate the aforementioned performance bottlenecks, increased memory footprint and rasterization artifacts. An additional rendering pass of the scene's geometry is initially employed to construct a per-pixel binary fragment occupancy discretization or *occupancy map* [LHLW09, SA09]. Then, the nearest depth of the k-th fragment for each pixel is concurrently computed by performing bit counting operations and subsequently utilized to perform early-z rejection for the k-buffer construction process that follows. As a result, fragments, whose depth is larger than the computed boundary, do not belong to the potential final result and eventually fail the depth test, avoiding their pixel shading execution cost. In essence, this process maximizes the number of fragments to be rejected and significantly reduces the total workload. An experimental evaluation is provided demonstrating the advantages of our work over the original k-buffer variants in terms of memory usage, performance cost and image quality.

2. Occupancy-based Fragment Culling

Ideally, when constructing a k-buffer, knowing the exact depth of the k-th fragment a priori allows us to insert all fragments with smaller or equal depth in constant time, discarding the rest (farther ones). In this work, we present an efficient approach for approximating the depth of the k_a-th fragment that is the nearest largest to the k-th one: $k_a \geq k$. Inspired by [LHLW09], a 32-bit unsigned integer 3D *fragment occupancy* array buffer is utilized to define a per-pixel bitmap, whose entries indicate the presence of fragments in each sub-interval. The depth range of each pixel p is divided into $S = 32 \cdot d$ uniform consecutive sub-intervals $[p.s_j, p.s_{j+1})$, where $p.s_j = p.z_n + \frac{j}{S}(p.z_f - p.z_n), j = 0, 1, \ldots, S-1$ and $d > k/32$ define the depth range subdivision. A *depth-range* map is initially computed, containing for each pixel p the nearest and farthest fragment depth values from the camera, $p.z_n$ and $p.z_f$, respectively [SA09]. However, this additional geometry pass is highly expensive especially for highly-tessellated scenes, so a *bounding box* [LHLW09] or a smaller representation as a *convex hull* may be instead chosen to approximate the geometry (see Fig. 2).

Initially, a **clear pass** initializes the occupancy buffer to zero. During the first geometry rasterization (**fill pass**), the

j-th bit of the occupancy map is set to 1 ($p.b_j = 1$) for each arriving fragment f falling within the corresponding sub-interval ($p.s_j \leq f.z < p.s_{j+1}$) via blending or atomic operations, depending on the hardware. A full-screen rendering stage follows (**count pass**) to concurrently count the number of 1s in the bit-array $p.b$ by adjusting the *Brian Kernighan*'s algorithm [Ker88]: count the number of times $p.b$ & $(p.b-1)$ is performed in a loop. When the counter reaches k, the algorithm halts and the current bucket's depth is returned to the Z-buffer ($O(k)$ time). Then, the k-buffer is efficiently constructed (**peel pass**) by taking advantage of the early-z culling capabilities of the GPU. Finally, a sorting process [LFS*14] is employed to reorder the captured fragments before generating the final image (**resolve pass**). Figure 3 illustrates how to modify any k-buffer pipeline in order to introduce our occupancy-based fragment culling.

Lemma 2.1 Given that more than one fragments are routed to the same subinterval, the foremost k fragments always succeeded to pass the culling mechanism.

Proof If $n_j \geq 0$ is the number of fragments falling into bucket j, $b_j = n_j > 0\ ?\ 1\ :\ 0$ and $N = \sum_j n_j > k$ is the total generated fragments at an arbitrary pixel, then exists an index i where $\sum_{j=0}^{i} b_j = k$. Let's assume that the statement is false, thus $\sum_{j=0}^{i} n_j < k \Leftrightarrow \sum_{j=0}^{i} n_j < \sum_{j=0}^{i} b_j$, which does not hold, since $n_j \geq b_j, \forall j \leq i$. □

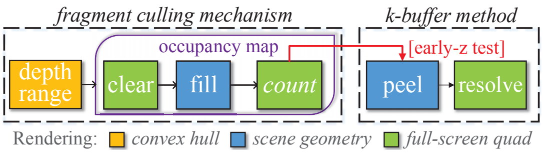

Figure 3: *The modified k-buffer pipeline.*

3. Results

We present an experimental analysis of our fragment culling mechanism extending a set of k-buffer realizations focusing on performance, memory requirements and image quality under different testing conditions (see also a comparative overview at Table 1). All experiments were conducted using a 1024^2 viewport with varying d, k on an NVIDIA GTX780 Ti with 3 GB of memory. We have also provided as supplementary material the shader code, including implementations for older and modern hardware, of the proposed components in the order shown in Figure 3.

Method	Perform.	Memory	Quality
[BCL*07]	↓	↑	×
[BM08]	↓	↑	↑
[SML11, YYH*12]	↑	↓	√
[MCTB13, Sal13, VF14]	↑	↑	√

Table 1: *Comprehensive comparison of augmented k-buffer approaches with fragment culling tests.*

Performance Analysis. Figure 4 shows how the performance exponentially improves when our fragment culling mechanism with $d = 32$ is exploited at k^+-buffer [VF14] when rendering three complex multilayer scenes with $k = \{4, 8, 16, 32\}$. Similar boost is observed for the rest variations [SML11, YYH*12, MCTB13, Sal13] (omitted for space reasons). Conversely, stencil-routed k-buffer [BM08] is significantly lessened from the additional geometry pass.

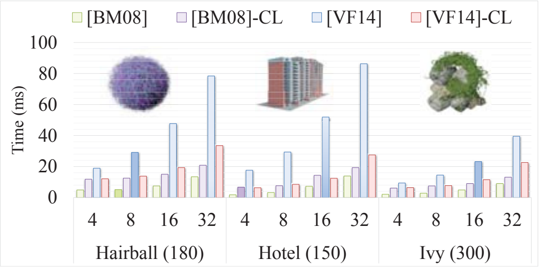

Figure 4: *Performance evaluation in ms of two representative k-buffer methods [BM08, VF14] without and with enabling our fragment culling on three scenes with high depth complexity (shown in brackets) and varying k.*

Figure 5 illustrates how the performance of *Hairball* rendering (see Fig. 1) scales when our fragment culling is exploited in k^+-buffer [VF14], under a set of increasing $d = \{1, \ldots, 64\}$ and fixed $k = 8$. We initially observe the small performance gain when the k^+-buffer's fragment culling is exploited. On the other hand, even with large depth subintervals the modified method is highly accelerated by the use of our technique. Note that when moving to higher values of d, computation of the culling layer is slightly increased. Finally, for $d = 32$, we performed culling testing inside the shader to show that moving from software- to hardware-implemented depth testing we achieved a 2× computation improvement.

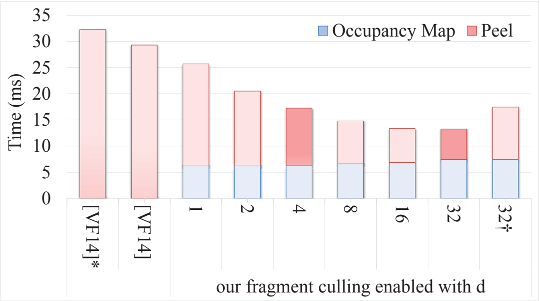

Figure 5: *Our mechanism speeds up the k^+-buffer [VF14] even when d remains at low levels. [VF14]* denotes a modified fragment culling-free k^+-buffer and † defines a shader-based fragment culling execution.*

Memory Allocation Analysis. Concerning memory demands, our mechanism requires specifically $(d-k+3)\cdot 32$-bit additional per-pixel storage (for $d > k$), when compared with the rest of memory-bounded k-buffer variants. Specifically, this extra memory overhead comes form the allocation of the near, far and k_a-th fragment depth textures and fragment occupancy buffer. A memory-friendly representation can also be implemented that reuses the fragment occupancy buffer for storing the color information of the actual k-buffer. On the other hand, the risk of a potential memory-overflow is avoided by highly reducing the stored fragments at the unbounded fragment linked lists [SML11, YYH*12]. For example, in the test scenario of Figure 1, we saved 67.4MB by storing only 1.72% extra unnecessary data.

Image Quality Analysis. Figure 1 shows the noticeable image quality improvement in the case of the *hairball* model, when fragment culling is enabled in the stencil-routed k-buffer [BM08]. Most of the visual information loss from RMWH is naturally recovered due to the reduced fragment racing. Note that recent k-buffer alternatives produce accurate results.

Limitations. First of all, the fragment rejection accuracy (and consequently the speed up) of the k-buffer construction is highly dependent on the occupancy map resolution; having no impact (which results at slow-down due to the extra pass) at the worst-case scenario of the front k fragments falling to the same bucket(s) due to z-fighting [VF13] or fragment distribution in small depth intervals. Secondly, the introduced extension is restricted to work for applications that aim to capture only the k-closest to the viewer fragments and not the “best” k ones [SML11]. Finally, RMWH of the original k-buffer [BCL*07] cannot be completely alleviated.

4. Conclusions

In this work, we have introduced a simple fragment culling process easily integrated to any k-buffer pipeline. The k-th fragment, well-approximated via fragment occupancy maps, defines the depth testing criterion. We expect that the included comparative experiments with respect to performance, memory usage and robustness will provide a useful guide for effectively addressing multi-fragment rendering on high-depth-complexity scenes.

Acknowledgments. 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

[BCL*07] BAVOIL L., CALLAHAN S. P., LEFOHN A., COMBA J. A. L. D., SILVA C. T.: Multi-fragment effects on the GPU using the *k*-buffer. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D ’07, ACM, pp. 97–104. 2, 3, 4

[BH14] BOLZ Z., HEYER M.: OpenGL extension: GL_NV_fragment_shader_interlock, 2014. 2

[BM08] BAVOIL L., MYERS K.: Deferred rendering using a stencil routed *k*-buffer. *ShaderX6: Advanced Rendering Techniques* (2008), 189–198. 1, 2, 3, 4

[GSM*14] GÜNTHER T., SCHULZE M., MARTINEZ ESTURO J., RÖSSL C., THEISEL H.: Opacity optimization for surfaces. *Computer Graphics Forum 33*, 3 (2014), 11–20. 1

[HBT14] HILLESLAND K. E., BILODEAU B., THIBIEROZ N.: Deferred shading for order-independent transparency. In *Proceedings of Eurographics 2014 Short Papers* (2014), EG ’14, The Eurographics Association, pp. 49–52. 1

[Ker88] KERNIGHAN B. W.: *The C Programming Language*, 2nd ed. Prentice Hall Professional Technic. Reference, 1988. 3

[LFS*14] LINDHOLM S., FALK M., SUNDÈN E., BOCK A., YNNERMAN A., ROPINSKI T.: Hybrid data visualization based on depth complexity histogram analysis. *Computer Graphics Forum* (2014). 3

[LHLW09] LIU F., HUANG M.-C., LIU X.-H., WU E.-H.: Efficient depth peeling via bucket sort. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG ’09, ACM, pp. 51–57. 2

[MCTB11] MAULE M., COMBA J. L., TORCHELSEN R. P., BASTOS R.: A survey of raster-based transparency techniques. *Computers & Graphics 35*, 6 (2011), 1023 – 1034. 1, 2

[MCTB13] MAULE M., COMBA J. A., TORCHELSEN R., BASTOS R.: Hybrid transparency. In *Proceedings of the 2013 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2013), I3D ’13, ACM, pp. 103–118. 2, 3

[SA09] SINTORN E., ASSARSSON U.: Hair self shadowing and transparency depth ordering using occupancy maps. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, USA, 2009), I3D ’09, ACM, pp. 67–74. 2

[Sal13] SALVI M.: Advances in real-time rendering in games: Pixel synchronization: Solving old graphics problems with new data structures. In *ACM SIGGRAPH 2013 Courses* (New York, NY, USA, 2013), SIGGRAPH ’13, ACM. 2, 3

[SML11] SALVI M., MONTGOMERY J., LEFOHN A.: Adaptive transparency. In *Proceedings of the 2011 Symposium on High Performance Graphics* (New York, NY, USA, 2011), HPG ’11, ACM, pp. 119–126. 2, 3, 4

[VF12] VASILAKIS A. A., FUDOS I.: S-buffer: Sparsity-aware multi-fragment rendering. In *Proceedings of Eurographics 2012 Short Papers* (Cagliari, Italy, 2012), EG ’12, pp. 101–104. 2

[VF13] VASILAKIS A. A., FUDOS I.: Depth-fighting aware methods for multifragment rendering. *IEEE Transactions on Visualization and Computer Graphics 19*, 6 (2013), 967–977. 4

[VF14] VASILAKIS A. A., FUDOS I.: k^+-buffer: Fragment synchronized k-buffer. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2014), I3D ’14, ACM, pp. 143–150. 1, 2, 3

[YHGT10] YANG J. C., HENSLEY J., GRUN H., THIBIEROZ N.: Real-time concurrent linked list construction on the GPU. *Computer Graphics Forum 29*, 4 (2010), 1297–1304. 2

[YYH*12] YU X., YANG J. C., HENSLEY J., HARADA T., YU J.: A framework for rendering complex scattering effects on hair. In *Proceedings of the 2012 symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D ’12, ACM, pp. 111–118. 1, 2, 3, 4