# Toward Efficient and Accurate Order-Independent Transparency

Ethan Kerzner* University of Iowa
Chris Wyman University of Iowa
Lee Butler US Army Research Laboratory
Christiaan Gribble SURVICE Engineering

## Abstract

Correctly rendering multi-layered transparent geometry requires accumulating contributions from multiple fragments per pixel. Dynamic A-buffers (e.g., Yang et al's [2010] per-pixel linked lists) achieve this by storing and sorting fragments on-the-fly. We introduce two improvements to recent GPU-based interactive A-buffer techniques. First, our redesigned algorithm uses fewer costly global atomic operations to construct linked lists. Second, we decouple visibility and shading to reduce memory demands of multi-fragment rendering.

**Keywords:** order independent transparency, real-time rendering

## 1 Introduction and Previous Work

Multi-fragment rendering has three per-pixel steps: identifying primitive visibility, shading, and accumulating final pixel color. Yang et al's [2010] per-pixel linked lists (PPLL) determine visibility in a single rendering pass, storing all fragments in a global buffer with a global atomic counter controlling write access. In contrast, the S-buffer [Vasilakis and Fudos 2012] uses two passes. The first obtains per-pixel fragment counts and allocates contiguous memory for each pixel's fragments. Memory requirements for both algorithms scale linearly with fragment count, but the S-buffer performs faster than PPLLs as it reduces global atomic contention during list creation and improves spatial coherence of fragment data when accumulating pixel colors.

## 2 Reduced Contention Per-Pixel Linked Lists

PPLLs use a global atomic counter to dynamically allocate a linked list node for each fragment. We propose reducing global contention by allocating memory per-primitive rather than per-fragment. Our *primitive-allocated linked lists* (PALLs) use a conservative bound of a primitive's fragment count to allocate memory, reducing fragment shader atomic contention. Fragment shaders then store visibility data inside this region using a linked list structure.

We implemented PALL in OpenGL. The reduced contention in PALL increases performance: when rendering $1.4 * 10^6$ fragments, PALL uses 4.95 ms per frame while PPLL uses 5.36 ms. Our supplementary material includes performance metrics that show PALL scales more efficiently than PPLL as total fragment count increases. However, PALL's conservative primitive bound increases memory usage.

## 3 The Compact A-Buffer

Existing interactive A-buffers store shading and visibility inside fragment lists. This saves per-primitive shading data repeatedly in multiple pixels. Decoupling storage of primitive and fragment data in our new *compact A-buffer* significantly reduces memory overhead. This approach resembles the decoupling proposed by Liktor and Dachsbacher's [2012] compact G-buffer.

Our compact A-buffer applies to either linked lists or the S-buffer. In both cases, rather than replicate per-primitive data in every fragment, we store a primitive ID with each fragment and create a single buffer of primitive data. When accumulating pixel color, we access this primitive data to shade each fragment. Although this adds a layer of indirection to shading computations, decoupling provides significant memory savings with only minor performance impact. Memory consumption still scales linearly with fragment count, but with a strictly lower constant factor as shown in Figure 1.

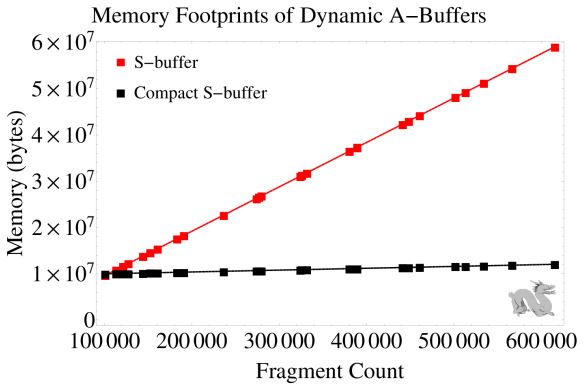


**Figure 1:** *Memory usage of regular and compact A-buffers, computed at $1024^2$ resolution with various fragment counts. When primitive count exceeds fragment count, our compact A-buffer has a larger memory footprint. However, our compact A-buffer scales more efficiently as average primitive size increases.*

## 4 Conclusion

We introduced two GPU-based A-buffer optimizations. The *primitive-allocated linked lists* reduces global fragment shader contention while computing primitive visibility, resulting in faster performance but increased memory usage. The *compact A-buffer* decouples storage of visibility and shading data, reducing memory demands at a small performance cost. Metrics and applications of these techniques are included in our supplementary material.

## References

LIKTOR, G., AND DACHSBACHER, C. 2012. Decoupled deferred shading for hardware rasterization. 143–150.

VASILAKIS, A., AND FUDOS, I. 2012. S-buffer: Sparsity-aware multi-fragment rendering. *Eurographics Symposium on Rendering*.

YANG, J., HENSLEY, J., GRUN, H., AND THIBIEROZ, N. 2010. Real-time concurrent linked list construction on the gpu. *Computer Graphics Forum 29*, 4, 1297–1304.

*e-mail:ethan-kerzner@uiowa.edu


ACM 978-1-4503-2261-4/13/07