Stochastic Layered Alpha Blending

Chris Wyman*
NVIDIA

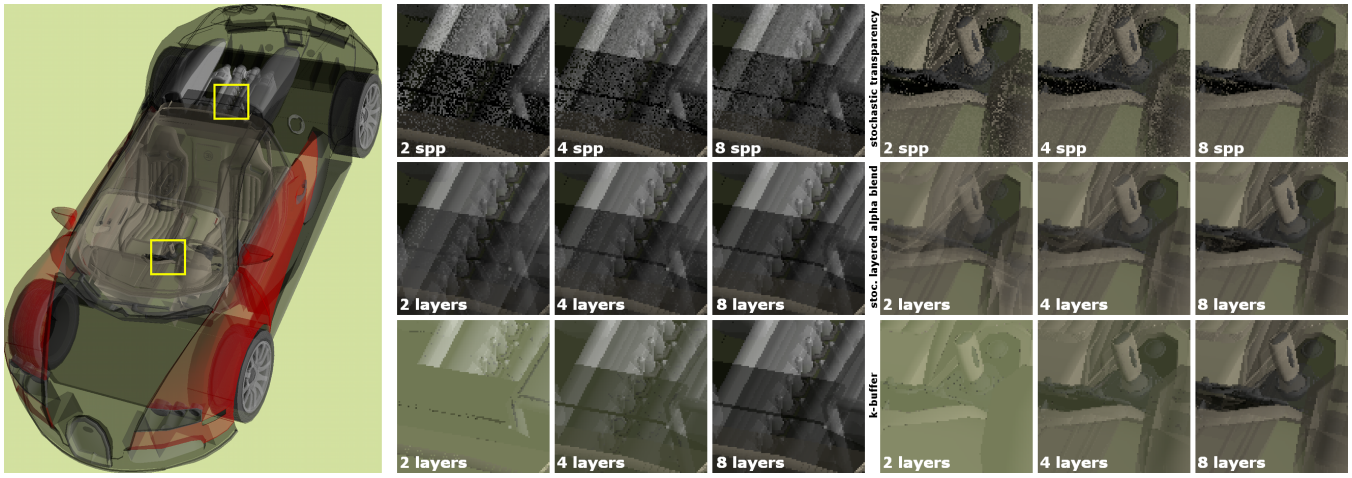

Figure 1: *Our* stochastic layered alpha blending *lies in the continuum of order-independent transparency techniques between stochastic transparency and k-buffering. Here we show a partially transparent model with around 20 depth layers. The insets compare (top) stochastic transparency, (middle) stochastic layered alpha blending, and (bottom) k-buffering with various number of samples. For stochastic layered alpha blending, we use continuous alpha with a number of virtual coverage bits (b in Equation 1) equal to twice the number of layers.*

Abstract

Researchers have long sought efficient techniques for order-independent transparency (OIT) in a rasterization pipeline, to avoid sorting geometry prior to render. Techniques like A-buffers, k-buffers, stochastic transparency, hybrid transparency, adaptive transparency, and multi-layer alpha blending all approach the problem slightly differently with different tradeoffs.

These OIT algorithms have many similarities, and our investigations allowed us to construct a continuum on which they lie. During this categorization, we identified various new algorithms including *stochastic layered alpha blending* (SLAB), which combines stochastic transparency's consistent and (optionally) unbiased convergence with the smaller memory footprint of k-buffers. Our approach can be seen as a stratified sampling technique for stochastic transparency, generating quality better than $32\times$ samples per pixel for roughly the cost and memory of $8\times$ stochastic samples. As with stochastic transparency, we can exchange noise for added bias; our algorithm provides an explicit parameter to trade noise for bias. At one end, this parameter gives results identical to stochastic transparency. On the other end, the results are identical to k-buffering.

Keywords: stochastic, order-independent transparency, alpha

Concepts: •**Computing methodologies** → **Visibility;**

*e-mail:chris.wyman@acm.org

SIGGRAPH 2016 Talks, July 24-28, 2016, Anaheim, CA
ISBN: 978-1-4503-4282-7/16/07
DOI: http://dx.doi.org/10.1145/2897839.2927423

1 Introduction

The classical A-buffer algorithm [Carpenter 1984] first stores a list of fragments affecting each pixel then accumulates their contributions during a resolve pass. To reduce memory consumption this list can be truncated (k-buffers [Bavoil et al. 2007]) or list elements can be merged due to space constraints. Adaptive transparency [Salvi et al. 2011], hybrid transparency [Maule et al. 2013], and multi-layer alpha blending [Salvi and Vaidyanathan 2014] provide a number of metrics for merging. Stochastic transparency [Enderton et al. 2010] *implicitly* builds per-pixel lists stochastically.

Unfortunately, these algorithms all have problems. A-buffers are expensive. Stochastic transparency converges slowly and relies on MSAA, limiting fast versions to 8 samples per pixel. K-buffers discard geometry. And when merging layers adaptively in a streaming, single pass process, the results can vary with geometry order (technically losing order independence).

2 Stochastic Layered Alpha Blending

We introduce *stochastic layered alpha blending* (SLAB), which stochastically inserts layers into a per-pixel list of fragments. This is different than k-buffers or prior multi-layer blending techniques which only discard layers when they run out of space. Like stochastic transparency, we might discard fragments that other layered blending techniques would keep, but we do this in an unbiased way. In fact, our probability of discarding fragments is identical to stochastic transparency's, though we compute our probability explicitly rather than implicitly via z-buffering.

Our motivation was straightforward: while stochastic transparency has never been described as using per-pixel lists of fragments, we observed it builds such lists implicitly. Figure 2 shows a simple $16\times$ MSAA pixel covered by 3 transparent fragments in turn. By definition no more than 16 surfaces can be stored per pixel, so like

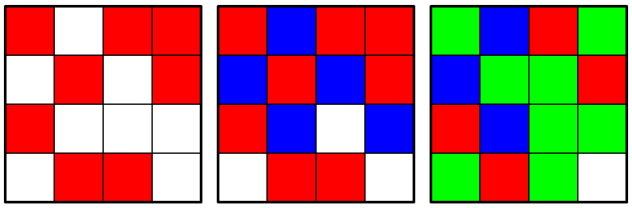

Figure 2: *Stochastic transparency with three fragments, $\alpha = 0.5$. (Left) When the red fragment is rasterized, it stochastically outputs its z into half the MSAA samples. (Center) The blue fragment also stochastically writes to half the MSAA samples, but some are occluded by the closer red fragment. (Right) The green fragment also writes to half the MSAA sample, but because it is in front of the other two fragments, it displaces blue and red fragments.*

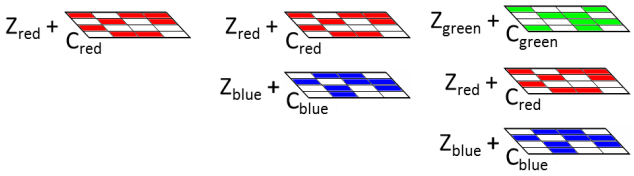

Figure 3: *Conceptually, stochastic layered alpha blending decouples storage of depth and stochastic subpixel coverage. Here, we use the same fragments and random subpixel sample from Figure 2. (Left to right) First red, then blue, then green are rasterized into our stochastic layered structure. Each fragment is stored in a layer containing the fragment depth and subpixel coverage. If we were limited to 2 layers per pixel, when the green fragment appeared the blue fragment would be discarded from our list.*

k-buffering it implicitly limits the number of fragments contributing to each pixel. However due to a coupling between depth and coverage, stochastic transparency redundantly stores information. In our example, since the green pixel is 50% transparent our final z-buffer stores its depth 8 times. If the red, green, and blue fragments came from our closest three surfaces, instead of being able to store 16 layers in this pixel we would be limited to 4 layers. In that case, why use 16 sample per pixel stochastic transparency rather than a 4 layer k-buffer?

Stochastic layered alpha blending decouples depth and coverage to avoid redundantly storing the depth of our layers. Each layer stores a depth and coverage mask, as shown in Figure 3. Since coverage samples no longer need to be explicitly tied to MSAA samples, layers can instead store depth and a continuous alpha, though this adds complexity to probability computations.

2.1 Algorithm

Consider a new incoming fragment F, a maximum of n layers per pixel, and an existing set of layers L_i for $i \in [1..m]$ (ordered by depth for illustration, i.e, L_m is furthest away).

If F's depth $\mathcal{Z}(F)$ is greater than $\mathcal{Z}(L_m)$ and $m = n$, discard F. Otherwise, we find L_j such than $\mathcal{Z}(L_j) < \mathcal{Z}(F) < \mathcal{Z}(L_{j+1})$. We then accumulate the coverage C_{acc} for layers 1 to j. If using a discrete coverage, this is the binary-or of C_i for $i \in [1..j]$. With a continuous alphas, $C_{acc} = 1 - \prod_{i=1}^{j}(1-\alpha_i)$. We probabilistically insert F between layers j and $j+1$ by picking a uniform random number $\xi \in [0..1]$ and inserting when $\xi < P_b(C_{acc}, C_F)$. P_b is the combinatoric probability that a b-bit coverage mask C_F, representing fragment F, will be visible given a b-bit occlusion mask C_{acc}. If using continuous alphas, both C_{acc} and C_F are discretized to integers in $[0..b]$. $(x)_y$ is the falling factorial, $x(x-1)...(x-y+1)$.

$$P_b(s,t) = \begin{cases} 1 - \frac{(s)_t}{(b)_t} = 1 - \frac{s!(b-t)!}{b!(s-t)!} & : t \leq s \\ 1 & : t > s \end{cases} \quad (1)$$

Equation 1 gives the probability for b stratified samples. A slightly different equation $(1 - (\frac{b-t}{b})^{(b-s)})$ gives the probability for b non-stratified samples. If, after insertion of F into our list, we have more than n layers, discard the furthest (i.e., L_m).

An interesting property is as we increase the number of coverage bits, i.e., $b \to \infty$, $P_b(s,t) \to 1$. So, with an infinite number of coverage bits per layer, stochastic layered alpha blending becomes k-buffering. Given n layers, b ranging from $[n..\infty]$ transitions between stochastic transparency and k-buffering.

As in stochastic transparency, the insertion pass can directly generate final color, or a second geometry pass can normalize the color and account for discarded surfaces in a biased manner. For easy comparison, our results for both techniques use Enderton et al.'s [2010] depth and alpha normalizations.

3 Results

We implemented a proof-of-concept of stochastic layered alpha blending in OpenGL, using the *NV_fragment_shader_interlock* extension to ensure insertion atomicity. Figure 1 shows an example, with insets comparing various settings for stochastic transparency, stochastic layered alpha blending, and k-buffers.

Optimizing our implementation is future work, but prototype performance is as follows (in Figure 1, 1920×1080, NVIDIA GeForce Titan X). With 4 (and 8) samples or layers: stochastic transparency 2.1 ms (2.5 ms), stochastic layered alpha blend 3.1 ms (4.3 ms), multi-layer alpha blend 5.5 ms (8.0 ms). For identical layer counts, our stochastic layered alpha blend is currently slower than stochastic transparency, due to fragment shader synchronization, but gives significantly less noise. Our k-buffer and stochastic layered alpha blending use the same code path (with different probability functions), so they have identical performance.

References

BAVOIL, L., CALLAHAN, S., LEFOHN, A., COMBA, J., AND SILVA, C. 2007. Multi-fragment effects on the gpu using the k-buffer. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 97–104.

CARPENTER, L. 1984. The a-buffer, an antialiased hidden surface method. In *Proceedings of SIGGRAPH*, 103–108.

ENDERTON, E., SINTORN, E., SHIRLEY, P., AND LUEBKE, D. 2010. Stochastic transparency. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 157–164.

MAULE, M., COMBA, J. A., TORCHELSEN, R., AND BASTOS, R. 2013. Hybrid transparency. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 103–118.

SALVI, M., AND VAIDYANATHAN, K. 2014. Multi-layer alpha blending. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 151–158.

SALVI, M., MONTGOMERY, J., AND LEFOHN, A. 2011. Adaptive transparency. In *Proceedings of High Performance Graphics*, 119–126.