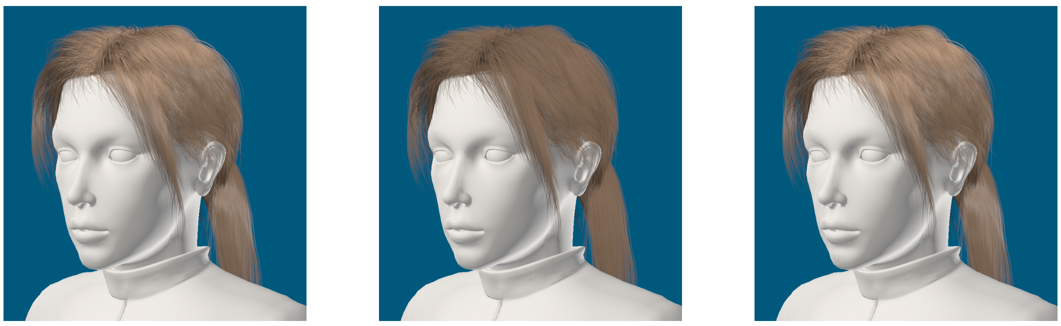
Deferred Shading for Order-Independent Transparency

K. E. Hillesland, B. Bilodeau and N. Thibieroz

Advanced Micro Devices, Inc.



(a) Deferred OIT. 2.8 ms (b) Low-cost Shading. 2.6 ms (c) Forward OIT. 4.9 ms

Figure 1: By deferring transparent fragment shading (a) we can identify fragments that do not influence the final pixel color significantly after blending and apply lower cost shading (b), giving us something very close to full shading of all fragments (c) but at much lower cost.

Abstract

Rendering many layers of transparency presents difficult challenges with respect to performance. Most previous work focused on the sorting problem, paying the full shading cost for all fragments. We present a method that defers shading until fragments can be classified as less important to the final pixel color, allowing us to switch to a lower-cost, approximate shading function. We apply this idea to TressFX*, which is a state-of-the-art hair rendering technique used in video game production. For hair rendering, we switched to low-quality shading in all but the front eight fragments per pixel. This gave us a 75% speedup without noticeable loss in visual quality.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shading, shadowing, visible line/surface algorithms

1. Introduction

Transparent objects are often difficult to render efficiently due to depth complexity. Hair and smoke, for example, can require blending tens to hundreds of layers. In the case of opaque objects, techniques such as occlusion culling and deferred shading are available to reduce shading cost. However, they rely on the fact that opaque objects completely obscure what is behind them. Because transparent objects, by definition, do not have this property, developers instead try to manage the cost of transparent-object shading by keeping the shading simple or rendering at lower resolution.

There has been some recent work in reducing the cost of sorting transparent objects, which is sometimes a significant cost when employing the standard blending functions. When the depth complexity is quite large, heuristics are often employed to identify which fragments are most important for the sake of saving sort cost, but all fragments incur the full cost of shading. Our work differs from previous work in that we tackle the shading cost issue. In fact, we adopt one of these previous approaches for saving sort cost, and improve performance further by substantially reducing shader cost. To achieve this, we need to defer shading of a fragment un-

DOI: 10.2312/egsh.20141012

til it can be classified as important or not. For those fragments identified as unimportant, we can save shading cost by switching to a lower-cost shading function.

2. Previous Work

Deferred shading stores shader inputs instead of color while rasterizing geometry, deferring the shading until the visible fragments have been identified. It leverages the assumption that only one fragment will be visible for each pixel sample; therefore, no more than one fragment need be shaded. This assumption is not true for transparent objects, so they are generally handled using a special forward pass in deferred renderers.

Achieving properly ordered blending requires sorting fragments by depth. One approach is to store all the fragments, and then sort and blend them. This can be accomplished in real time on modern graphics hardware by storing the fragments in per-pixel linked lists using atomic exchange and increment [YHGT10]. However, this method becomes prohibitive when depth complexity is in the hundreds.

A number of approaches have approximated full fragment sorting and blending, but the two closest to this work are a k-buffer implementation [YYH*12], which was simplified for *TressFX* [LaC13], and Hybrid Transparency [MCTB13]. They classify the k fragments closest to the eye in each pixel as the most important for sorting and blending. We do the same for sorting and blending, but also apply the heuristic for choosing where to apply shading cost.

We are effectively using shader level of detail to save shading cost. The main difference with previous work for shader level of detail is that we use it for fragments that contribute little to final pixel color in the context of transparency, rather than for low end hardware, or real-time constraints [OKS03, Pel05].

3. Method

Because of the nature of blending, some fragments in a pixel will contribute more to the final pixel color than others. We shade important fragments at high quality, and therefore high cost, and unimportant fragments using a lower-cost function.

We adapted the two-pass method of Yu et al. [YYH*12]. This is also the approach of *TressFX*, which has been used successfully in a game production environment [LaC13]. They classify the closest k fragments as being more important than the rest for sorting purposes. We refer to the front fragments as the *core* and the rest as the *tail*, adopting the terminology from Maule et al [MCTB13].

In *TressFX*, a geometry pass shades and collects fragments in per-pixel linked lists [YHGT10]. A second screen-space pass collects the front k fragments per-pixel in shader registers while blending tail fragments out of order. Once the front k elements have been collected, they are sorted and blended in order. This method requires only one pass over the scene geometry. In the next section, we will describe how this is applied to the specific case of hair rendering, in which a single geometry pass is important for performance.

Our method differs in that we defer shading to the second pass. As each fragment is identified as being part of the tail, we apply lower-cost shading, saving the higher-cost shading for the core. To defer the shading, we must store the shading inputs rather than final color in the linked lists. This can mean additional memory in a lot of cases, just as G-buffers tend to take more memory than final color in standard deferred rendering. In exchange, we get savings on shader cost, and the tradeoff of a deferred approach becomes more attractive as shading cost goes up. For the remainder of this paper, we refer to previous approaches as *forward* and to ours as *deferred*.

3.1. An Implementation for Hair

We apply this principle to the case of hair rendering, specifically the method used in *TressFX* [LaC13], including their approach to shading, shadowing and anti-aliasing. Our modification is to apply cheaper shading and shadowing to fragments in the tail. We chose to use the same k fragments for both sorted blending and high-quality shading, although this need not be the case. The model and viewpoint we used for analysis is shown in Figure 1.

We attenuate light according to how deep the hair fragment is in the hair volume relative to the light using a shadow map. Rather than classifying a fragment as in shadow when it is closer than the shadow map depth, we attenuate it according to the difference in depth using the following function.

$$S_n = \sum_{i=1}^{n} w(x_i, y_i)(1-\alpha)^{k\,d(x_i,y_i)} \tag{1}$$

where α is the transparency of the hair, d is the difference in depth between the hair fragment and the shadow map depth sampled at offset (x_i, y_i) and w is a Gaussian weight based on distance from the center of the kernel. Quality is controlled by the choice of n, which for our tests was 1 for lower cost shading in the tail and 25 for higher cost shading in the core. The high value reduces aliasing caused by the high-frequency nature of the hair geometry.

For shading, we use one of two functions, again dependent on the quality choice. For the first, we use a modified Kajiya-Kay function [Sch04].

$$R = (1 - S_n)\,(R_d + R_R + R_{TRT}) \tag{2}$$

The reflection factor R is computed from the shadow factor (S_n of Equation 1), a diffuse reflectance term (R_d) and two shifted specular terms (R_R and (R_{TRT}) as predicted by the Marschner model [MJC*03]. The diffuse term is the same as

Kajiya-Kay, and the two specular terms are shifted Kajiya-Kay specular terms.

For the *tail* fragments, we use only the diffuse term (R_d) with a single shadow tap (Equation 1 specialized for $n = 1$).

We also compute coverage analytically similarly to Yu, et al. [YYH*12] and factor this into the transparency term. Computing coverage in the first pass allows us to eliminate fragments with alpha less than $1/255$.

In general, the memory required to store shader inputs is greater than the memory required to store color; however, it is the same in this implementation. Both forward and deferred versions require storage of depth, and for a linked-list implementation, a next pointer. The forward version also requires storage of color plus alpha. At one byte per component, this is four bytes for color plus alpha. For the deferred version, we instead use those same four bytes to store the hair tangent (three bytes) and alpha (one byte), again for a total of four bytes.

3.2. Analysis Methodology

We evaluate performance and error relative to the forward version used in *TressFX*, which draws all fragments at high quality. We also look at two other variants.

In the first, we draw all layers with low-cost shading. This establishes a lower bound on quality and shader cost as k is reduced (to $k = 0$). We do not reduce k for sorting.

In the second, which we call *tailless*, we draw only the front k layers. We still need to process the entire list of fragments to find the front k, and we also still accumulate opacity of the tail fragments so the background is blended as if all the layers are there. This is the lowest bound on tail-shader cost, which is none at all.

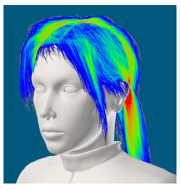
We primarily analyze performance and quality for $k = 8$ because we found this to be a good choice in terms of trade-off between quality and performance when considering the forward-rendering method. Also, it is the value used in publicly released game implementations [LaC13]. We analyze a single representative viewpoint for consistency. Timings are GPU time for each pass on an AMD Radeon HD 7970.

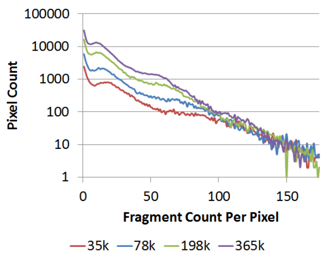
As we zoom in so that hair pixels cover more of the screen, the total load goes up, but the average depth complexity goes down. To analyze this effect, we vary the field of view. Total hair pixel coverage is 35k, 78k, 198k, and 365k for the four fields-of-view analyzed at 1920 x 1200. We refer to these views by their pixel coverage. We use the 198k case as the primary case (i.e., when we list only a single value).

For error, we use the root-mean-square error (RMSE) in linear color space, considering only hair pixels and weighting all channels equally. Measurements were taken for the 198k pixel view. These values are measured using 8 bits per-channel outputs, meaning they are in the range of 0 to 255 for each channel, or $\sqrt{3 \times 255^2} \approx 442$ is the upper bound on the RMSE.

4. Results

Because our method is designed to address performance in cases of transparency with high depth complexity, we begin with an analysis of the depth complexity for our test scene. Figure 2 shows the number of fragments per pixel for the case of 198k pixels with an average overdraw of about 20x. Other choices for field of view that we analyze cause this to vary from 18x to 30x, with 30x corresponding to the lowest pixel count.


(a) Visualizing fragments per pixel.



(b) Fragment histogram.

Figure 2: Fragments per pixel. (a) visualizes the case for 198k hair pixels, with grey = 1, blue = 8, green = 50, and red = 100 and above. (b) shows a histogram for the different zoom levels.

The deferred approach, with lower-quality shadowing and shading in the tail, introduces some error relative to the forward approach. At $k = 8$, the RMSE is 9. There are no noticeable artifacts relative to the forward approach, and it is difficult to distinguish between the two without being able to toggle directly between them. The tailless alternative results in an RMSE of 26 at $k = 8$. This tends to exacerbate sampling issues, introducing some objectionable noise. Finally, applying low-cost shading to all fragments results in 90 RMSE, mainly due to the lack of specular highlights and our failure to normalize to account for that loss, but the single-tap shadow filter also introduces a fair amount of artifacts. The point, however, is that even a poor low-cost shader in the tail gives us visual improvement.

Two figures illustrate performance characteristics. Figure 3 breaks down time for the forward and deferred approaches at various resolutions to illustrate how shading time is shifted from the first pass in the forward version to the second pass in the deferred version. We also included the tailless version to see the maximum available performance for the choice of the low-cost shading function.

Figure 4 shows the incremental cost of the tail fragments for the forward and deferred versions. The purpose of Figure 4 is to illustrate that the deferred approach gives an in-

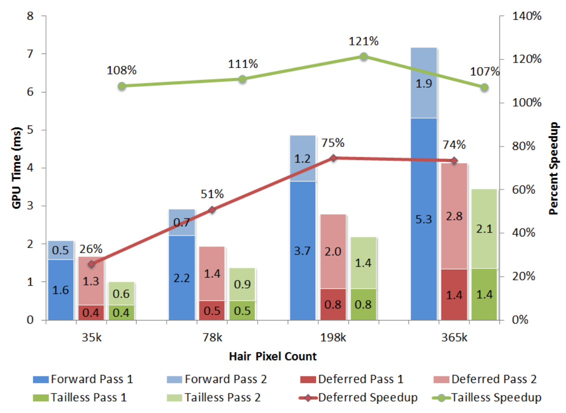


Figure 3: Comparison of forward and deferred performance at four different resolutions. These numbers are for the $k = 8$ case.

crease in fidelity for much lower cost and better performance scaling than the forward version.

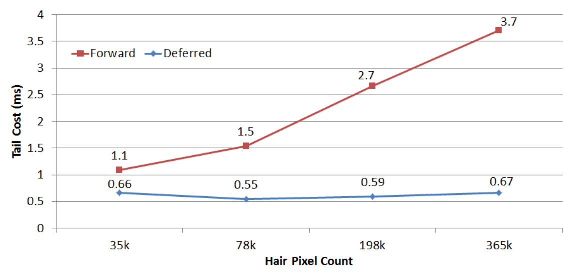


Figure 4: Additional cost of shading the tail.

Any additional cost of deferring the shading must be offset by the savings in switching to lower-cost shading in the tail. In this particular case, there is no increase in memory cost for storing tangents rather than colors in the linked list. In exchange, we were able to save both compute cost and memory bandwidth due to fewer shadow-map texture fetches.

Our method's speedup is limited by how much we can reduce shading cost in the tail. Reducing tail-shading cost to zero gives us the tailless case in Figure 3. This gives us an upper bound on speedup without changing k.

Our method's speedup is also limited by how much we can reduce k for shading. This, in turn, depends on the quality of the low-cost shading choice. The low-cost shading we chose shows noticeable quality loss at $k = 4$. However, reducing k from eight to zero only drops GPU time from 2.8 ms to 2.6 ms for the 198k case.

It should also be noted that although our method shows greater speedup as the relative load of fragment shading increases, it also starts to decrease at higher resolutions as the fraction of fragments in the tail decreases. This is indicated by the slight drop in the deferred speedup curve for the 365k case (Figure 3).

5. Conclusions and Future Work

Applying high-quality shading to the front fragments of each pixel while using a lower-cost shading for the rest provides a good trade-off between quality and performance. The shading cost is spent on the fragments that matter the most to the final image. For hair rendering, we achieved up to 75% speed-up over the forward version (Figure 3) while maintaining high-quality shading results.

For the future, we are interested in exploring other k-buffer implementations, as well as other transparency approximations such as Adaptive Transparency [SML11]. We would also like to look at other applications that exhibit high levels of transparent depth complexity, such as particles or atmospherics.

References

[LaC13] LaCroix J.: A Survivor Reborn: Tomb Raider on DX11. Game Developer Conference Advanced Visual Effecs with DirectX 11 Course Notes, March 2013. 2, 3

[MCTB13] Maule M., Comba J. a., Torchelsen R., Bastos R.: Hybrid Transparency. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2013), I3D '13, pp. 103–118. 2

[MJC*03] Marschner S. R., Jensen H. W., Cammarano M., Worley S., Hanrahan P.: Light scattering from human hair fibers. *ACM Transactions on Graphics 22* (2003), 780–791. 2

[OKS03] Olano M., Kuehne B., Simmons M.: Automatic shader level of detail. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2003), HWWS '03, pp. 7–14. 2

[Pel05] Pellacini F.: User-configurable automatic shader simplification. *ACM Trans. Graph. 24*, 3 (July 2005), 445–452. 2

[Sch04] Scheuermann T.: Practical real-time hair rendering and shading. In *ACM SIGGRAPH 2004 Sketches* (2004), SIGGRAPH '04, p. 147. 2

[SML11] Salvi M., Montgomery J., Lefohn A.: Adaptive Transparency. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (2011), HPG '11, pp. 119–126. 4

[YHGT10] Yang J. C., Hensley J., Grün H., Thibieroz N.: Real-time concurrent linked list construction on the GPU. In *Proceedings of the 21st Eurographics conference on Rendering* (2010), EGSR'10, pp. 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 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2012), I3D '12, pp. 111–118. 2, 3