Hybrid Transparency

Marilena Maule* UFRGS
João Comba UFRGS
Rafael Torchelsen UFFS
Rui Bastos NVIDIA

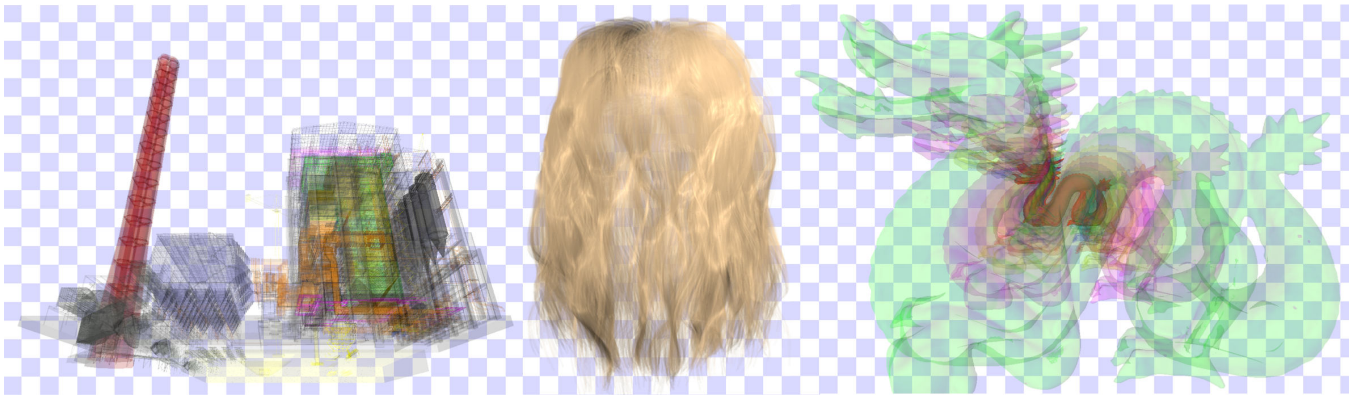


Figure 1: *Order-independent transparency (OIT) images generated in real-time using hybrid transparency. The maximum depth complexities are: 588 layers for the power plant, 671 layers for the hair, and 66 layers for the 11 dragons. Our hybrid approach closely approximates OIT by storing only* 16, 4, *and* 8 *of the transparent layers (left to right), combined to an approximation of the remaining transparent layers.*

Abstract

Hybrid transparency is an approach for real-time approximation of order-independent transparency. Our hybrid approach combines an accurate compositing, of a few core transparent layers, with a quick approximation, for the remaining layers. Its main advantage, the ability to operate in bounded memory without noticeable artifacts, enables its usage with high scene complexity and image resolution, which other approaches fail to handle. Hybrid transparency is suitable for highly-parallel execution, can be implemented in current GPUs and further improved, with minimal architecture changes. We present quality, memory, and performance analysis and comparisons which demonstrate that hybrid transparency is able to generate high-quality images at competitive frames rates and with the lowest memory consumption among comparable OIT techniques.

Keywords: order-independent transparency, visibility determination, real-time rendering

1 Introduction

Order-independent transparency (OIT) rendering has received significant research, but it is still considered an open problem. The main difficulties are memory consumption (usually unbounded) and processing cost (highly-varying with scene depth complexity). Correct rendering of scenes with transparent objects involves processing transparency layers in depth-sorted order—i.e., either front-to-back (FTB), which is how a viewer perceives a scene, or back-to-front (BTF), which is how light traverses a scene. The seminal work on alpha compositing in [Porter and Duff 1984] defines the algebra to combine two fragments, given a depth-order relation between them. The equations it defines are fundamental for proper compositing of transparent layers under the sorting requirement.

When looking at a scene with transparent objects, in FTB order, we notice that the objects behind a transparent layer get dimmed by the opacity of the layer. When looking through multiple transparent layers, the dimming of visibility accumulates successively at every layer. This leads to the conclusion that, after a given threshold, the remaining layers became so dimmed that make a very small contribution for the final blended color. This suggests splitting the transparent layers of a pixel in two classes: a core and a tail. The first transparent layers, close to the viewer, tend to have high impact on the color of a pixel and define the core of the blended fragments. Remaining layers (away from the viewer) have lower-and-lower impact, and define a tail of visible fragments.

Hybrid transparency (HT) is based on these observations, combining an accurate method for computing OIT on k core layers of each pixel and an approximate (but very fast) method for the remaining layers (tail). Both methods handle order-independent rasterization, but only the core requires sorting. HT presents high image quality and no visible artifacts (the viewer gets the correct notion of layers spatial relationship, without discontinuity artifacts, e.g. noise), as shown in Figure 1, but runs in bounded memory with two geometry passes. HT provides real-time frame rates, competitive with high-performance techniques which use unbounded memory—i.e. practical implementation of adaptive transparency [Salvi et al. 2011]. We summarize the main contributions of our work as follows:

- a practical method for OIT that can run in bounded memory in today's GPUs;
- rendering of larger transparent scenes at higher resolutions than current methods can handle, at competitive frame rates;
- no visually-noticeable artifacts (such as noise and flickering).

*e-mail: mmaule@inf.ufrgs.br

2 Background and Related Work

The intrinsic goal of OIT proposals is to correctly represent a visibility function that describes the contribution of layers along depth. The visibility of a fragment is defined by the multiplication of two components: (i) the opacity of the fragment and (ii) the transmittance of the accumulation of fragments in front of it. The algebra described in [Porter and Duff 1984], can be extended to compute the transmittance (τ) of a fragment i (from a depth-sorted list of fragments of a pixel) using the opacities (α) in front of it:

$$\tau_i = \begin{cases} 1 & \text{if } i = 1 \\ \prod_{j=1}^{i-1}(1-\alpha_j) & \text{if } i > 1. \end{cases} \tag{1}$$

Many approximate methods for rendering of transparent surfaces have been proposed to estimate the visibility function, which contains several terms (fragments), using only a subset of them. But, the removal of terms can lead to artifacts due to incorrect approximations of important terms. Another alternative to approximate transparency is to make assumptions about the transparent layers and to not sort them before blending. In this section, we will review only buffer-based techniques that solve transparency at fragment level, which implies in some storage of transparent layers for compositing. We refer the reader to the survey presented in [Maule et al. 2011] for further details and compendium of OIT techniques.

A common approach to handle the required visibility-sorted order is to store all transparent layers while rendering, and then sort and blend the fragments of each pixel as post-processing—i.e., A-buffer [Carpenter 1984; Yang et al. 2010]. This buffer-based approach produces correct images, but storing and processing all transparent fragments require unbounded memory, and lead to slow and highly-varying performance for complex scenes. As alternative, several approaches have been proposed to work on limited memory.

The OIT approaches cannot simply discard fragments when overflow occurs; this would lead to disturbing artifacts—e.g., noise and flickering. Rendering transparency using bounded memory requires strategies to approximate the visibility of layers that could not be stored. The Z^3 technique [Jouppi and Chang 1999] keeps three fragments (per pixel) in depth-sorted order, and combines the two closest fragments when overflow. The success of this approach is highly dependent of the incoming order of fragments because, after a merge of two fragments, no other fragment can be correctly inserted between them. The k-buffer [Bavoil et al. 2007] employs the same strategy, storing k fragments per pixel, but performs a coarse primitive sorting; so, fragments are generated in nearly-sorted order and the compositing is unlikely to introduce visibility errors.

Adaptive transparency (AT) [Salvi et al. 2011] approach approximates the transmittance function along depth using a fixed number of terms per pixel. When overflow occurs, the heuristic for removal of terms from the function aims at removing those that have smaller contributions to the pixel color, thus reducing the error of the approximation. In a second geometry pass, AT combines fragments in incoming (unsorted) order by consulting the transmittance function and their own opacity to determine their visibility. With this approach, several transparent layers can be handled using a fixed number of transmittance terms, and the error is constrained to less important terms. As proposed, AT requires critical sections at fragment level to handle parallel removal of terms from the transmittance function, which is not available on current GPUs. Alternatively, for validating the image quality, the authors used an unbounded-memory algorithm as support to store all transparency fragments, and post-processed them to emulate what the proposed AT algorithm should do.

Stochastic transparency [Enderton et al. 2010] presents an innovative approach to estimate visibility. The technique computes an alpha-to-coverage probability, used to fill individual samples in a multi-sampled texture, following the screen-door concept. With increasing number of samples, fragments are more likely to fill the percentage of samples associated to their correct visibilities. However, for a small number of samples, the method suffers from severe noise artifacts.

OIT techniques with best performance are those based on non-standard blending equations, thus requiring no sorting—i.e., no buffer and no more than one geometry pass. The weighted sums technique (WSum) [Meshkin 2007] was proposed for rendering of several particles, where most fragments have similar colors and opacities. The proposed blending equation ignores the visibility reduction that one fragment implies over those behind it, and compose all fragments weighted only by their own opacity. As a result, the pixel color saturates very quickly, producing too bright color artifacts. The weighted averages technique (WAvg) [Bavoil and Myers 2008] produces better results than WSum because it distributes the transmittance equally among the transparent fragments, avoiding saturation. Because WAvg does not account for the correct visibility reduction imposed by one layer over those behind, high opacity values generate very poor color approximation.

In order to approximate the correct contribution of fragments, HT evaluates the visibility of front-most fragments using an accurate method, because front-most fragments tend to be the most significant to the pixel color (this assumption is supported by the results presented in Section 4). The remaining fragments have little contribution individually, but altogether their contribution is significant to the pixel color, so they are not discarded but combined with a less accurate method.

3 Hybrid Transparency

In order to balance image quality, memory consumption, and performance, we developed the hybrid transparency technique (HT). This novel OIT approach divides the transparent layers in subsets defined by the importance of their contribution to the pixel color, and applies different algorithms to compose these subsets. Subsets with major importance are composed by accurate algorithms, which are slow due to the required sorting, while less important subsets of transparent fragments can be combined using less accurate, but faster, algorithms.

Parts of a scene that are seen through multiple transparent layers get dimmed by their accumulated opacity. This suggests that fragments away from the viewer are not as important as the closer ones, and their contribution can be approximated with less impact to the pixel color. Therefore, HT splits the fragments of a pixel in two subsets: core fragments, which are closer to the viewer, and tail fragments, which are seen by the viewer through the core, thus, getting dimmed and less important to the pixel color. The two subsets are combined with the opaque background to produce the final pixel color. Figure 2 illustrates the monotonically-decreasing behavior of transmittance along depth, which leaded us to the selection of the core and tail subsets.

3.1 The Core

The core is the subset of transparent fragments of a pixel that has main contribution to the pixel color, thus, deserving accurate blending. Leaded by the monotonically-decreasing behavior of the transmittance function along depth ([Salvi et al. 2011] and Fig. 2), we selected the front-most fragments to compose the core. Different high-quality algorithms can be used—e.g. depth peeling[Everitt 2001] or A-buffer[Carpenter 1984]—with minor modifications in order to select the core fragments.

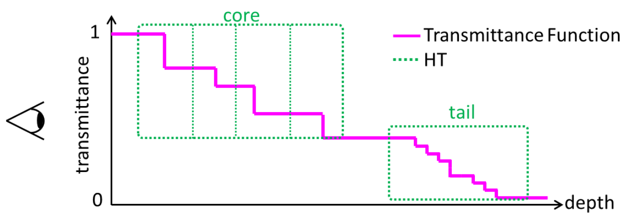


Figure 2: *Transparent fragments are weighted by the transmittance at their respective depth. The monotonically-decreasing behavior of the transmittance function indicates that fragments closer to the viewer tend to have major visibility, thus, composing a subset of core fragments which must be combined using an accurate algorithm. The same property lead us to conclude that fragments away from the viewer tend to have less contribution (visibility) and can be approximated with less prejudice to the pixel color.*

To extract k core fragments of a pixel, we use a truncated A-buffer (k-TAB), which allows capturing and blending the k closest fragments using bounded memory. The exact number of fragments to be handled by the core can be configured based on a memory budget, or based on information about the scene to be handled (image-quality requirement).

Using the **over** operator (from [Porter and Duff 1984]), we can define the visibility (v_i) of a core fragment i, using the opacities (α) of the depth-sorted core fragments, as

$$v_i = \begin{cases} \alpha_1 & \text{if } i = 1 \\ \alpha_i \times (1 - \sum_{j=1}^{i-1} v_j) & \text{if } i > 1. \end{cases} \tag{2}$$

Each color (C) of the k core fragments is weighted by the visibility of the fragment, and the sum of these weighted colors compose a new layer, which we call core layer (C_{core}), described by Equation 3. The opacity of the core layer (α_{core}) is the accumulation of visibilities from all the k fragments (Equation 4), and it is used to dim the visibility of the subsequent subset of fragments (the tail).

$$C_{core} = \sum_{i=1}^{k} (C_i \times v_i) \tag{3}$$

$$\alpha_{core} = \sum_{i=1}^{k} v_i \tag{4}$$

3.2 The Tail

The core will handle a fixed number of k fragments, which means that overflows are expected. If a pixel has n fragments to combine, all fragments not captured by the core ($n - k$) compose the tail subset. The tail algorithm must work on limited memory and still consider an unbounded number of fragments without overflowing. Two algorithms fit the requirements: weighted sum (WSum) [Meshkin 2007] and weighted averages (WAvg) [Bavoil and Myers 2008]. Both algorithms are fast, work in fixed memory and with only one geometry pass; because they do not perform sorting they are approximate.

Our choice was WAvg, because it provides a better color approximation [Bavoil and Myers 2008]. WAvg works in a single geometry pass by accumulating the colors of the fragments (C_i) weighted by their own opacities (α_i), and accumulating their opacities as well. Equation 5 describes the colors accumulation (C_{acc}), while Equation 6 describes the opacities accumulation (α_{acc}).

$$C_{acc} = \sum_{i=k+1}^{n} (C_i \times \alpha_i) \tag{5}$$

$$\alpha_{acc} = \sum_{i=k+1}^{n} \alpha_i \tag{6}$$

In a post-processing full-screen pass, the accumulated color is weighted by the accumulated opacity, producing the tail color (C_{tail}), as described by Equation 7. WAvg uses the counting of fragments to distribute the accumulated visibility equally among all the fragments of the pixel, composing the tail layer. This is done by dividing the accumulated opacity by the number of fragments ($t = n - k$), as in Equation 8.

$$C_{tail} = \frac{C_{acc}}{\alpha_{acc}} \tag{7}$$

$$\alpha_{avg} = \frac{\alpha_{acc}}{t} \tag{8}$$

Applying Equation 1, the transmittance of the composition (how much can be seen through the composition (τ_{tail})) is defined as

$$\tau_{tail} = (1 - \alpha_{avg})^t. \tag{9}$$

While the transmittance of the composition weights the next layer (background), the opacity (or absorptance: the amount of light blocked) of the composition (α_{tail}) weights the tail color. The opacity is the complement of the transmittance, so

$$\alpha_{tail} = 1 - \tau_{tail}. \tag{10}$$

3.3 Core, Tail, and Background Compositing

With the core and tail accumulated into two transparent layers, the **over** operator [Porter and Duff 1984] can be used to combine them, along with the background, or opaque layer. Each layer is dimmed by the layers in front of it. The tail contribution is limited by the transmittance remaining after the core compositing, and the background is weighted by the transmittance remaining after core and tail compositing.

The final color (C_{final}) accounts for the three contributions: (i) from the core (C_{core}), which is the first layer with full visibility ($v_{core} = 1$), (ii) from the tail (C_{tail}), which is dimmed by the core opacity and (iii) from the background (C_{bg}), weighted by the transmittance remaining after the compositing of the core and the tail (since $\alpha_{bg} = 1$). Equation 11 describe the final compositing.

$$C_{final} = C_{core} + (1 - \alpha_{core}) \times (C_{tail} + (1 - \alpha_{tail}) \times C_{bg}) \tag{11}$$

The HT core correctly estimate the visibility of its fragments. The HT tail correctly weights the fragments by their own opacities, but approximates their transmittance. Thus, HT approximates the transmittance function of a pixel as depicted in Figure 3.

3.4 Implementation Details

Hybrid transparency can be implemented on current GPUs using two geometry passes. The first pass extracts the k closest fragments and is followed by a full-screen pass that computes their visibilities. The second geometry pass does the shading and accumulates the fragments into the appropriate color buffer (core or tail). A final full-screen pass composites the core, tail, and background buffers.

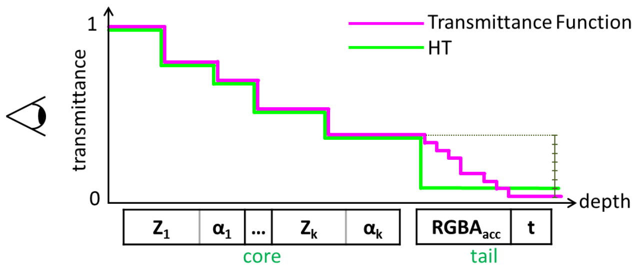


Figure 3: *HT approximation for the transmittance function: the core correctly evaluates the first part of the transmittance function. The remaining transmittance is equally distributed among the remaining terms.*

The first rendering pass is used only by the core, which collects the k front-most layers into a 3D texture that implements the k-TAB. Since the k-TAB is always kept sorted, the insertion of a fragment only requires to place itself in order. This is done by a single-pass bubble sort that swaps the fragments into their proper place using atomic operations. Today's GPUs do not provide atomic operations for more than one 32-bit word. Yet, 32 bits are enough to store a 24-bit depth and an 8-bit opacity (alpha) per fragment. After the first geometry pass, we draw a full-screen quadrilateral to create the corresponding per-pixel visibility function, which allows to correctly blend the k front-most fragments during the second geometry pass. The computation of the visibility function is done by traversing the list of k core fragments for each pixel and computing the visibility per depth, which is stored into another k-deep 3D texture (in order to acquire better numerical precision).

During the second geometry pass, fragments are shaded and forwarded to the core or tail processing, being composited in the respective color buffers. If the depth of a fragment is present in the k-TAB (created in the first rendering pass), the fragment is handled by the core; otherwise, it is handled by the tail. The last step combines the two transparent color buffers with the opaque one (or background). Algorithm 1 summarizes these steps—note that all steps are performed on the GPU.

Algorithm 1 Hybrid Transparency

```
1. Store: $< z, a >$
        1.1 Render Geometry
        1.2 Atomically keep the $k$ front-most fragments (core) in depth-sorted order into the k-TAB
2. Visibility Evaluation (core)
        2.1 Traverse the k-TAB, computing visibility at each depth and storing it into a 3D visibility buffer
3. Render: shade and compositing
        3.1 Render Geometry
        3.2 Composite the core fragments into color buffer $C_0$
        3.3 Accumulate overflowed fragments (tail) into color buffer $C_1$
4. Final Composite
        4.1 Composite color buffers and opaque background
```

4 Results and Discussion

We designed several tests to evaluate our proposal and check its viability for real-time OIT rendering, as well as created situations to stress-test our algorithm quality. HT is compared against the state-of-the-art OIT technique for visibility compression, the adaptive transparency (AT), which promisses to work in fixed memory when critical sections become available on the GPU, and is able to produce images with barely noticeable artifacts (very reduced noise and flickering) when compared to other known techniques—e.g. Z^3 and k-buffer. A reference image generated by a GPU implementation of the A-buffer is used to evaluate quantitative image quality. We also evaluate the core and tail components of HT independently, to demonstrate the advantages of the proposed hybrid combination.

Since one of the main advantages of our proposal is the ability to run in fixed memory, we evaluate buffer-overflow situations and their impact on image quality. Performance measurements are presented to demonstrate the practical application of HT for OIT renderin, along with a theoretical complexity analysis demonstrating its scalability with highly-parallel execution.

4.1 Memory Consumption Analysis

The memory consumption of HT depends on: (i) the screen dimensions (width W, and height H) and (ii) the core size (k). So, the total memory used can be described by:

$$TotalMemory = W \times H \times (2k + 2). \tag{12}$$

Each layer of the k-TAB stores a tuple $< z, \alpha >$, packed into a single word (4 bytes), and another word (4 bytes) for the accumulated visibility of each fragment. Two more words (8 bytes) per pixel are used to count and accumulate the color of all overflowed fragments.

The memory used by HT is comparable to the required for depth-peeling and WAvg, which depend mostly on the screen resolution, and little on the scene depth complexity—these are the most memory-economic techniques available. Unlike such techniques, HT has good performance and image quality simultaneously.

Memory consumption of HT is also comparable to the theoretical proposal of AT (which is supposed to work in bounded memory using critical sections). The actual AT implementation uses linked lists to hold an unbounded number of fragments (i.e. can run out of memory). Thus, the memory usage reported for AT in section 4.3 is the size actually used to store all fragments of the scene.

4.2 Image Quality Analysis

The images generated by HT are compared against a reference image (generated with a GPU implementation of the A-buffer) and images generated by AT and by HT components (k-TAB and WAvg). We used two distinct 3D scenes for practical image comparison: (i) the hair scene, same used in the original AT paper, and (ii) multiple instances of the Stanford dragon aligned. We also include a per-pixel error analysis of the color distance from the results of the methods to the reference image, with increasing memory (k).

Figure 4 shows the image quality evaluation for the hair model under different memory budgets. HT presents significant quality improvement when compared to the AT approach, using only 4 slots per pixel. We can also observe that, individually, the methods that compose the core (k-TAB) and the tail (WAvg) of HT are not able to produce the same image quality. With just 4 slots, the image generated with HT is barely distinguishable from the reference.

The dragon scene is composed of 22 instances of the dragon model, with challenging opacity configurations. The aligned instances of the model have the following opacities: 0.05, 0.1, 0.05, 0.1, 0.2, 0.9, 0.2, 0.1, 0.05, 0.1, 0.05 and its repetition. This situation generates front-most fragments that are less visible, due to their low opacities (and these are stored in the HT core), leaving fragments with higher contributions to be combined by the HT tail (less precise).

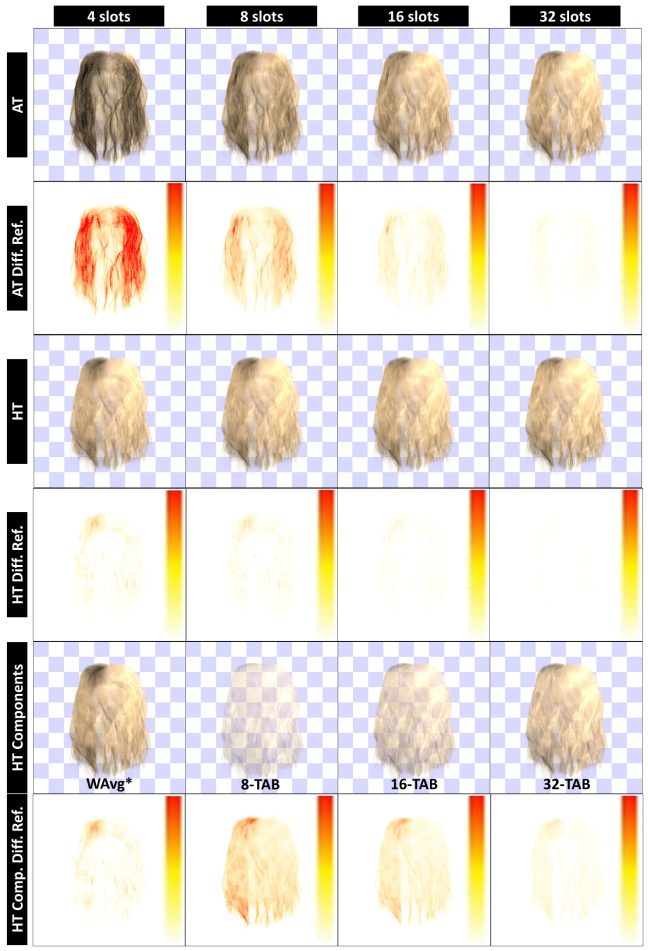


Figure 4: *Hair model with 15,000 strands and up to 663 transparent layers: columns with increasing number of slots used to approximate OIT (per pixel). Images generated by AT are presented in the first row. The second row displays the normalized distance to the reference image in color-coded mode. Third and fourth rows displays the same results for HT. The fifth row shows images generated by the HT components, and their corresponding error in the sixth row. WAvg does not permit parameterization, being unable to be improved with more memory. The TAB is presented with 8, 16 and 32 slots storing only the front-most fragments, discarding the remaining ones. (model dark.hair from Cem Yuksel)*

These opacities configuration is a difficult scenario for HT. However, as results in Figure 5 show, no artifact is visually noticeable, specially when compared to the disturbing artifacts generated by the other techniques. 4-TAB shows severe errors, because it is unable to store enough significant layers. WAvg errors are less noticeable, but the incorrect distribution of visibility caused loss of details. The AT results show an interesting shadow effect caused by the heuristic used to approximate transmittance on overflow.

Both AT and HT approximate the visibility function encoded by incoming fragments using fixed-size buffers. AT discards fragments with smaller contribution to the function by always underestimating transmittance approximations. HT assumes that the front-most fragments are the most important, so their transmittance are correctly evaluated, and the tail fragments equaly share the remaining transmittance. Figure 6 compares the approximation of the transmittance function for HT and AT.

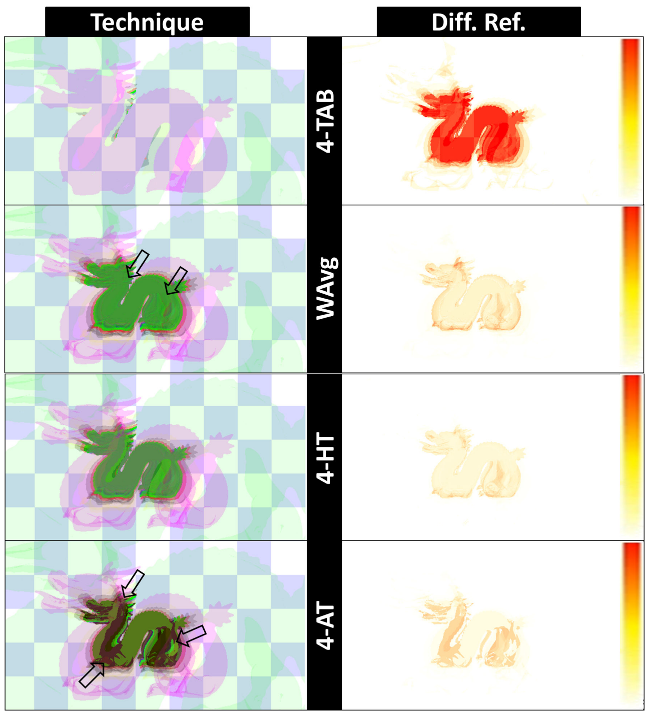


Figure 5: *22 aligned dragons: instances closer to the viewer have low opacities (HT core stores less important fragments), while instances farther away have high opacities (HT tail is compositing the most important terms). The first column presents images generated by each technique, followed by the color-coded error images in the second column. Even though this situation is challenging for HT, its error is smaller than the errors of other approaches, and HT visual quality is undoubtedly better.*

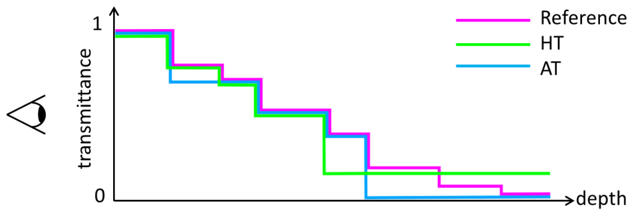
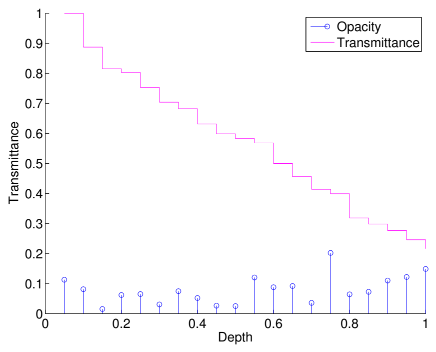


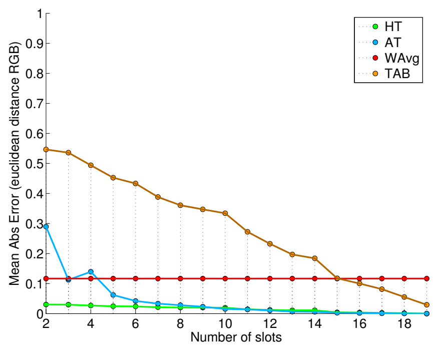
Figure 6: *HT and AT approximations of the transmittance function using a fixed number of slots. AT underestimates the transmittance for compressed nodes of the function, while HT correctly represents the first part of the function, and distributes equally the remaining transmittance.*

We present a second set of experiments with simulations of different alpha behaviors, measuring the color distance to the reference. The goal is to systematically explore scenes that can better assess the image quality generated by HT and other approaches. Results are reported in Figure 7 for four different distributions of alpha (opacity): random, uniformly increasing, HT worst case and constant; each producing different behaviors of the transmittance function (τ).

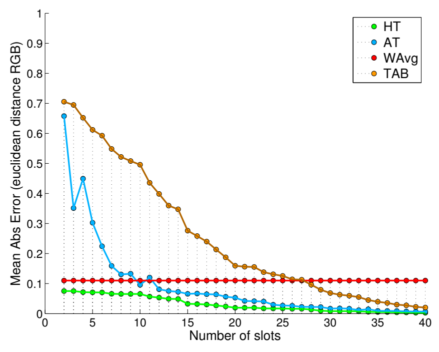
First, we create a visibility distribution with randomly chosen alpha values, which might appear in scenes that require mixed rendering effects (e.g. hair and smoke and foliage). Results of the random distribution show that, even for small memory budgets, HT presents the smallest errors.



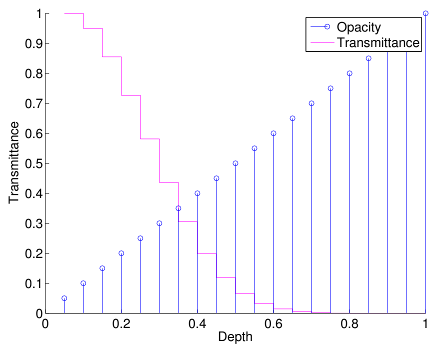
(a) ***Random*** *alpha distribution and τ function.*



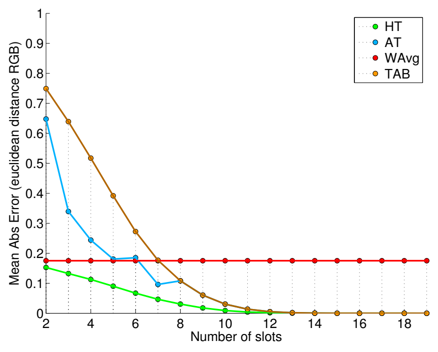
(b) *Error estimation for* ***random*** *alpha with 20 fragments.*



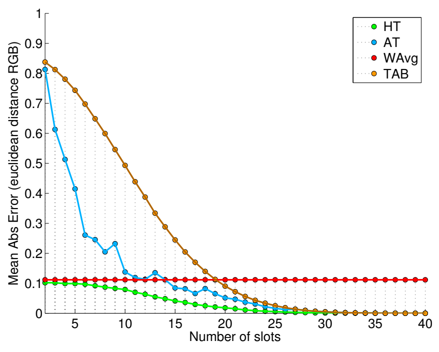
(c) *Error estimation for* ***random*** *alpha with 100 fragments.*



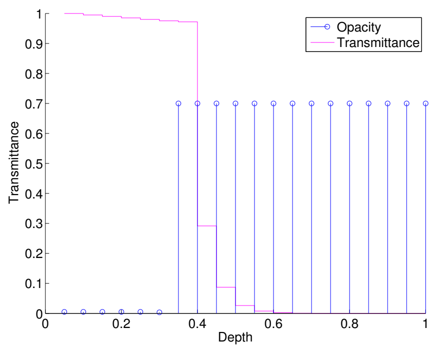
(d) ***Uniformly Increasing*** *alpha distribution and τ func.*



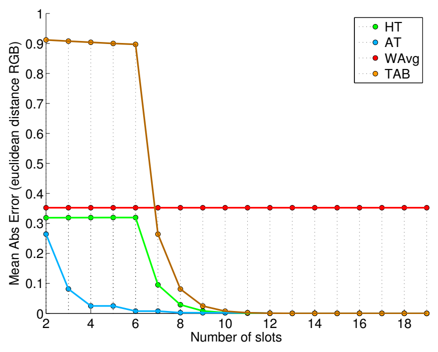
(e) *Error estim. for* ***uniform. increas.*** *alpha with 20 frags.*



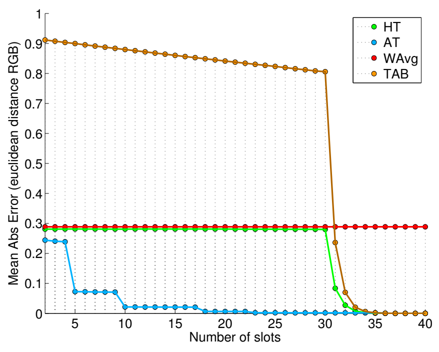
(f) *Error estim. for* ***uniform. increas.*** *alpha with 100 frags.*



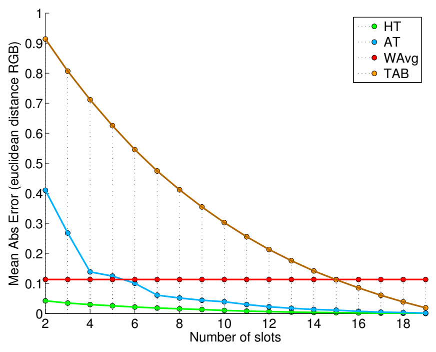
(g) ***HT worst case*** *alpha distribution and τ func.*



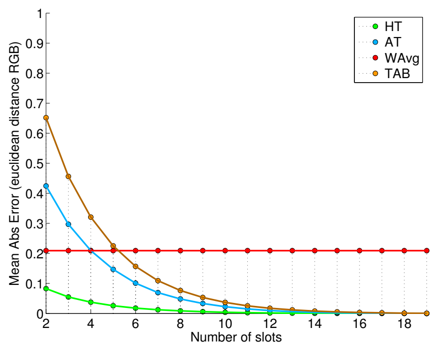
(h) *Error estimation for* ***HT worst case*** *alpha with 20 frags.*



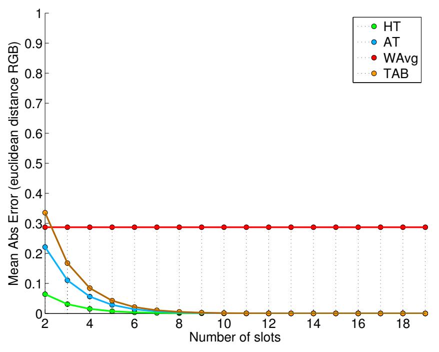
(i) *Error estimation for* ***HT worst case*** *alpha with 100 frags.*



(j) *Error estim. for* ***constant*** *alpha=0.1 with 20 frags.*



(k) *Error estim. for* ***constant*** *alpha=0.3 with 20 frags.*



(l) *Error estim. for* ***constant*** *alpha=0.5 with 20 frags.*

Figure 7: ***Error estimation*** *for different alpha distributions with increasing number of slots (k). We simulated the composition of fragments by the algorithms 100 times, each time with random colors, and plotted average errors. Figures 7a, 7d, 7g show the alpha behavior and how it changes transmittance along depth. We present evaluations for 20 fragments (Figures 7b, 7e, 7h), and for 100 fragments (Figures 7c, 7f, 7i). We also present the error behavior for constant alphas with 20 fragments (Figures 7j, 7k, 7l).*

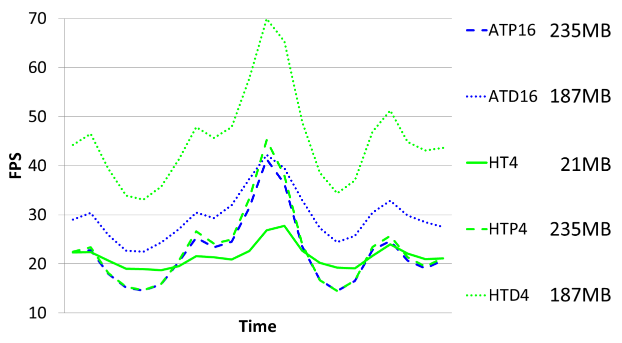


Figure 8: *Hair walkthrough FPS: Dotted lines denote techniques executed with unbounded memory support. HT4 (full line) runs in bounded memory. (15,000 strands, up to 671 transparent layers)*

The second distribution has alphas uniformly increasing along depth. It is interesting to note how TAB and HT behaviors resemble the transmittance function, but the hybrid aspect of HT greatly reduces the errors, while AT presents unpredictable behavior.

The third distribution is the worst scenario for HT, where no important layer is captured by the core. This alone would not cause significant errors, but combining it with the subsequent important layers (which is the WAvg worst case) can compromise image quality. This is the single configuration in our experiments where we were able to find that HT generates higher errors than AT.

Finally, we simulated constant alphas behaviors, usually present in scenes with particles or objects of the same material. Except from WAvg (which does not improve upon memory increase), all techniques display an exponential error decrease with an increasing number of slots.

In summary, the experiments reported in this section show that HT has several advantages over other methods with respect to image quality. For instance, HT does not present visible flickering (caused by different approximations in consecutive frames), neither loss of details (caused by incorrect evaluation of visibilities). HT is stable and does not produce noise artifacts. In fact, HT does not generate visually-noticeable artifacts.

4.3 Performance Analysis

HT was developed to sustain a high level of parallelism. The expected execution time is dominated by the first rendering pass, in which each thread performs up to k atomic swaps to keep an average of d fragments in depth-sorted order. The k swaps are performed in parallel, so, as in a pipeline of k stages and d instructions, the complexity order is $O(k + d)$.

The AT theoretical proposal uses critical sections to select the best samples to use in the transmittance function. However, critical sections are non-parallelizable blocks of instructions. The execution time of theoretical AT also would be dominated by the first rendering pass, with cost equivalent to the serialization of d blocks of instruction, with i instructions each, over k entries, for a total cost of $O(d \times i \times k)$.

Our practical experiments were performed using a GeForce GTX 580 GPU and an Intel Core i7 CPU running Windows 7 (x64). Since the actual implementation of AT needs unbounded memory support to run in current GPUs (originally the per-pixel linked lists, PLL), we included in our experiments another version of AT using a more economic unbounded algorithm (Dynamic Fragment Buffer (DFB) [Maule et al. 2012]), which enabled us to compare AT with HT for a massive model (the UNC Power Plant).

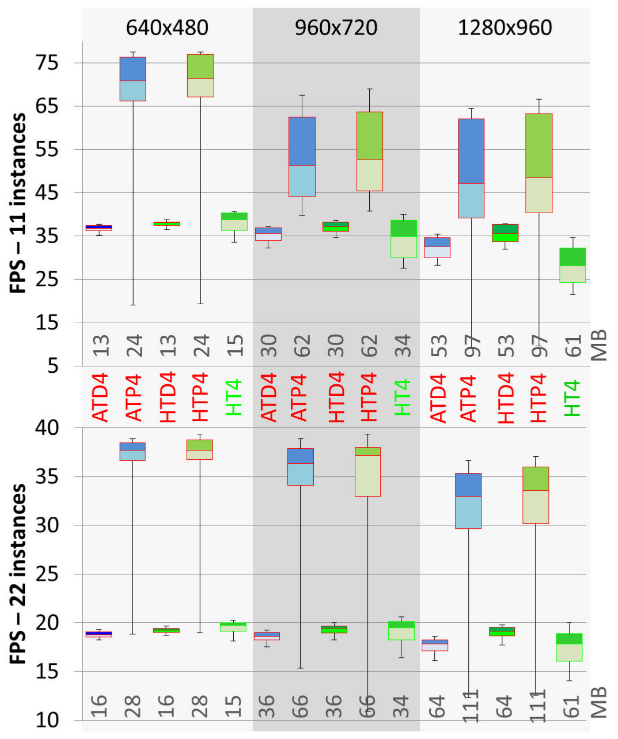


Figure 9: *Dragons variating resolution and number of instances. The red color denotes techniques executed with unbounded-memory support. HT4, in green, runs in bounded memory. (up to 132 transparent layers and 19,171,108 triangles)*

For fair comparisons with AT, resultss for our proposal (HT) were given using three implementation variants: (i) with unbounded-memory support from PLL (HTP), (ii) with unbounded-memory support from DFB (HTD), and (iii) the original bounded-memory approach (HT). The same support variations were used for AT, called ATP, when using the original PLL support, and ATD, when using the DFB support. The numbers following the techniques abbreviations denote the number of slots used to approximate the visibility function (k).

Figure 8 shows the performance results, along with the memory required by each technique, for rendering the hair model. The frames per second (FPS) of each method were measured for several frames during a walkthrough around the model. Dotted lines represent the emulated versions with unbounded-memory support, among which HTD4 presents outstanding performance. The fixed-memory HT4 not only has the lowest memory consumption (at least 8x smaller), but also has performance comparable to ATP16 and HTP4 running in unbounded memory.

A walkthrough around the scene with aligned instances of the Stanford dragon is used to test the techniques for different resolutions and number of instances. Figure 9 shows the performance results, and memory consumption, for increasing resolutions and two sets of instances. We display the FPS as boxplots to capture the performance variation along the camera path. Versions using PLL, with only one geometry step, are faster than HT and the unbounded versions using DFB, both requiring two geometry steps. Each dragon instance has 871,414 triangles and up to 6 transparent layers, which

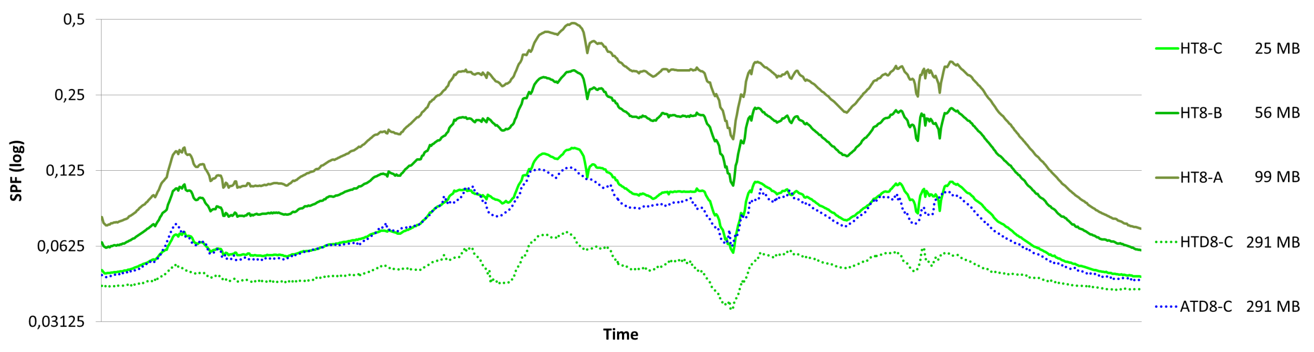


Figure 10: *Power Plant walkthrough SPF: Dotted lines denote techniques executed with unbounded-memory support. HT8's (full lines) run in fixed-size memory. The accompanying letters A, B and C indicate the screen resolution: A is for 1280x960, B is 960x720 and C is for 640x480. (the Power Plant scene has up to 588 transparent layers and 12,748,510 triangles)*

leads us to the conclusion that the performance for this scene is dominated by geometry processing.

A future single-pass HT (which needs 64-bit atomic updates available in GPU) is expected to present great performance improvement. In the other hand, a practical bounded-memory version of AT requires serialization of code chunks (through critical sections) to correctly compute the transmittance function update. For a massively-parallel architecture, as the GPUs, serializations mean great performance loss.

In our last experiment, we measured the seconds taken to generate each frame (SPF) along a walkthrough around the entire UNC Power Plant (12,748,510 triangles, up to 588 transparent layers). This model combines the two previous tests: several transparent layers and heavy geometry. The versions requiring unbounded memory (dotted lines) were able to run only for the lowest resolution, running out-of-memory for the bigger ones (PLL versions were unable to handle this model). HT is able to render bigger resolutions efficiently, because it has fixed and small memory footprint; its performance is comparable to the unbounded AT, while the unbounded HT is up 2 times faster.

5 Conclusion and Future Work

We introduced hybrid transparency, a novel OIT concept that can be extended combining different techniques to balance memory consumption, performance, and image quality. Our particular hybrid instance uses a truncated A-buffer (accurate) for the closest transparent layers and weighted averages (fast) for the remaining ones. HT is able to handle any number of transparent layers, operating in fixed memory on current GPU technology. Our tests demonstrate the feasibility of HT, in particular, its ability to handle complex scenes and high screen resolutions, which previous techniques could not handle with the same image quality. Performance results show that HT can be used in real-time applications with visually-imperceptible artifacts (no noise, and no flickering).

A significant improvement to the HT performance would be allowed if GPUs had support for 64-bit atomic operation. Also, with critical sections available in GPUs, a different number of slots per pixel would be possible, thus, storing the necessary number of fragments to reach an opacity threshold, providing quality guarantees.

As future work we plan to explore the use of many cores and tails interchanged along depth, in order to improve the image quality for the HT worst case, as well as explore possible combinations of HT with anti-aliasing techniques.

References

BAVOIL, L., AND MYERS, K., 2008. Order independent transparency with dual depth peeling. Technical report, NVIDIA.

BAVOIL, L., CALLAHAN, S. P., LEFOHN, A., COMBA, JO A. L. D., AND SILVA, C. T. 2007. Multi-fragment effects on the GPU using the k-buffer. In I3D '07: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, 97–104.

CARPENTER, L. 1984. The a-buffer, an antialiased hidden surface method. In SIGGRAPH '84: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, 103–108.

ENDERTON, E., SINTORN, E., SHIRLEY, P., AND LUEBKE, D. 2010. Stochastic transparency. In I3D '10: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 157–164.

EVERITT, C., 2001. Interactive order-independent transparency. Technical report, NVIDIA Corporation.

JOUPPI, N. P., AND CHANG, C.-F. 1999. Z3: an economical hardware technique for high-quality antialiasing and transparency. In HWWS '99: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, 85–93.

MAULE, M., COMBA, J. L., TORCHELSEN, R. P., AND BASTOS, R. 2011. A survey of raster-based transparency techniques. Computers & Graphics 35, 6, 1023 – 1034.

MAULE, M., COMBA, J. L., TORCHELSEN, R. P., AND BASTOS, R. 2012. Memory-efficient order-independent transparency with dynamic fragment buffer. In 25th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), 2012.

MESHKIN, H., 2007. Sort-independent alpha blending. Perpetual Entertainment, GDC Session.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. SIGGRAPH Comput. Graph. 18, 3, 253–259.

SALVI, M., MONTGOMERY, J., AND LEFOHN, A. 2011. Adaptive transparency. In Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics, ACM, New York, NY, USA, HPG '11, 119–126.

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