Neural Moment Transparency

G. Tsopouridis[1] and A. A. Vasilakis[2,3] and I. Fudos[1]

[1]Department of Computer Science and Engineering, University of Ioannina, Greece
[2]Department of Informatics, Athens University of Economics and Business, Greece
[3]Phasmatic, Greece

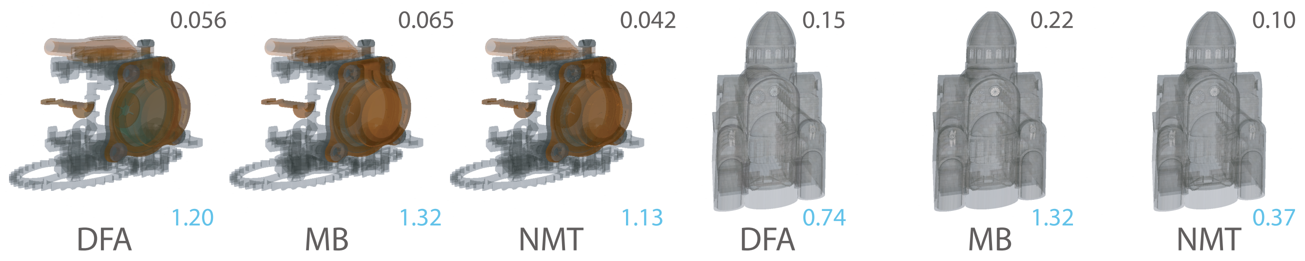

Figure 1: *Our approach qualitatively outperforms the competing methods according to the ꟻLIP (top right) and MSE (x10^{-4}, bottom right), achieving better quality results in scenes with varying depth complexity D, and opacity a. Left: D = 52, a = 0.15, Right: D = 31, a = 0.3.*

Abstract

We have developed a machine learning approach to efficiently compute per-fragment transmittance, using transmittance composed and accumulated with moment statistics, on a fragment shader. Our approach excels in achieving superior visual accuracy for computing order-independent transparency (OIT) in scenes with high depth complexity when compared to prior art.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ *Neural networks; Rasterization; Visibility;*

1. Introduction

Order Independent Transparency (OIT) is a widely used technique in computer graphics that enables the rendering of transparent objects in a scene while maintaining the correct light transmittance process [Wym16]. OIT techniques aim to solve this issue by ensuring that the correct ordering of transparent fragment samples is maintained [VVP20], even when the objects intersect with each other. Specifically, OIT is derived by

(i) sorting all n fragments f of a pixel p by their depth z: $[f_n, f_{n-1}, \dots, f_1]$, so that $z_n \le z_{n-1} \le \dots \le z_1$.
(ii) computing the pixel color $C_p = C_f(n)$ by the following recursive formula (also known as *over compositing operator* [PD84]):

$$C_f(i) = \begin{cases} C_{bg} & i = 0 \\ a_i C_i + (1 - a_i) C_f(i-1) & i = 1, \dots, n \end{cases} \quad (1)$$

where C_{bg} is the background color, C_i and a_i are the color and opacity of fragment f_i and $C_f(i)$ is the color in front of fragment f_i. Therefore, $C_f(n)$ is the color of pixel p.

Equation 1 can be rewritten in terms of the per fragment transmittance function $T(i)$ as follows:

$$C_p = C_{bg} T(0) + \sum_{i=1}^{n} a_i C_i T(i), \; T(i) = \prod_{k=i+1}^{n} (1 - a_k) \quad (2)$$

The first step requires storing all fragment information in a per pixel buffer and then sorting all fragments according to their depth value. This rendering pipeline is known as A-buffer [YHGT10] and is time and memory demanding [VVP20]. To approximate OIT with interactive performance and reasonable memory requirements several approaches have been used that extract per pixel information by blending information from all fragments in two or more rendering passes [Wym16].

In this paper we use the moment-based pipeline introduced by [MKKP18; Sha18], where an additive blending of power moments of absorbance is stored in a first pass followed by a second pass for the reconstruction of the per fragment transmittance. In our method, we replace the second pass with a pre-trained neural net-

DOI: 10.2312/egs.20241029

work that computes the transmittance based on input parameters such as the absorbance power moments, the number of fragments, and the normalized fragment depth. Our approach offers a 20% to 50% improvement in accuracy compared to power moments, with the trade-off of a slight performance overhead (around 15%). Our method is applicable to other absorbance/transmittance applications in the field of computer graphics.

2. Related Work

Blended transparency rendering. Real-time 3D applications, such as games, require satisfactory transparency results, all while adhering to stringent constraints on computation time costs [VVP20]. Blended OIT methods focus on portability and speed by adjusting the order-dependant over compositing operator [PD84] so that its arguments commute. *Weighted-sum alpha blending* [Mes07] and *Weighted-average alpha blending* [BM08] were the first to simplify the blending formulation. However, both methods come with poor transparency output on scenes with high opacity values. *Weighted-blended alpha blending* [MB13] color-weighted the operator to include depth-related information. The weights for each fragment are computed from a monotonic decreasing function. In general, this is the fastest way to convincingly approximate OIT, if the weight function is carefully tuned to match the scene's transparent content well.

Moment transparency rendering. [Sha18; MKKP18] is an innovative approach to real-time transparency rendering without relying on empirically determined weights. This approach uses moments, specifically a series of depth powers, to approximate the transmittance function with high visual quality. An advantage of using moments is that they are filterable, enabling efficient rendering at lower resolution, and facilitating reconstruction at a higher one. Nevertheless, the accuracy of the approximation is intricately linked to the type and quantity of moments employed. Utilizing a smaller number of moments leads to quicker performance but at the expense of accuracy. While, opting for a greater number of moments impedes performance and further escalates memory storage.

Deep transparency rendering. Neural networks have been used for correctly compositing transparent surfaces [TFV22; TVF24]. While these methods compute the overall OIT color on a per-pixel basis, our method explores a per-fragment moment-based transmittance inference step which leads to improved visual results.

3. Neural Moment Transparency

We utilize a neural network to directly calculate the transmittance for OIT on a per-fragment basis. Instead of reconstructing transmittance from moments as described in [MKKP18], we employ power moments to represent the transmittance of each fragment. These power moments, along with additional per-pixel information, serve as input features for our neural network, allowing us to enhance the transmittance approximation. As shown in Figure 2, we render transparent geometry twice: (i) first, to compute the stored transmittance using a moment function of all pixels (Moment Calculation) and other per-pixel statistics, and (ii) second, to estimate transmittance. Finally, we blend the color of all transparent surfaces (Transmittance Estimation).

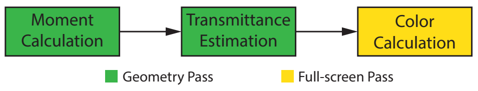

Figure 2: *Rendering flow diagram of our method.*

According to Equation 2, the transmittance T of each fragment f at depth z_f can be expressed as:

$$T(z_f) = \prod_{i=f+1}^{n} (1 - a_i) \tag{3}$$

To avoid explicit fragment ordering, moments are transformed to the log space, allowing alpha composition (Eq. 1) to be treated as an additive process. By doing so, the absorbance grows monotonically with the depth z_f and can be interpreted as cumulative distribution function of the finite measure. This allows for a compact representation based on the statistical moment theory. Thus, absorbance A [MKKP18] is defined as:

$$A(z_f) = -ln(T(z_f)) = \sum_{i=f+1}^{n} -ln(1 - a_i) \tag{4}$$

We choose four power moments $\mathbf{b}(z) = (1, z, z^2, z^3, z^4)$ as our moment generating function [MKKP18], allowing us to additively store each pixel transmittance as:

$$T(b) = \sum_{i=1}^{n} -ln(1 - a_i) * b(z_i),\ z_i \in [-1, 1] \tag{5}$$

The zeroth moment $b_0 = \sum_{i=1}^{n} -ln(1 - a_i)$, stores the total absorbance, from which we can derive the total transmittance $(exp(-b_0))$ of each pixel, and is used to blend the transparent surfaces with the background color C_{bg}.

We use the stored absorbance (Eq. 5) as an input to our neural network, along with the other input features (Sec. 3.1), to directly infer the per-fragment transmittance t_{pr} and finally the per-pixel blended color is computed by:

$$C_b = \sum_{i=1}^{n} a_i * C_i * t_{pr}(i) \tag{6}$$

With a_i, C_i, $t_{pr}(i)$, being the opacity, color and predicted transmittance of each fragment respectively. Finally, we compute the final color C_p by normalizing similarly to the original paper [MKKP18]:

$$C_p = \frac{1 - exp(-b_0)}{\sum_{i=1}^{n} a_i * t_{pr}(i)} C_b + exp(-b_0) C_{bg} \tag{7}$$

3.1. Network Architecture and Feature Selection

We have selected a compact fully connected multilayer perceptron to predict a per-fragment transmittance, given seven float input features (Fig. 3). Given the nature of the task, requiring a fast real-time performance, we require a highly compact neural network architecture as the inference is executed on a per-fragment basis. The neural network is composed of an input, an output, and two hidden layers (32 and 16 neurons) that use ReLU activation function. The output

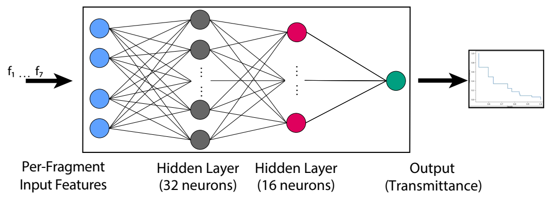
f₁ … f₇
Per-Fragment Input Features
Hidden Layer (32 neurons)
Hidden Layer (16 neurons)
Output (Transmittance)

Figure 3: *A visualization of our neural network that maps the input features f of each fragment into a transmittance.*

layer returns a float value $t_p \in [0,1]$ that represents the predicted fragment transmittance, using the *sigmoid* activation function. The neural network expects the following 7 per-fragment float features as input:

- Fragment depth (z) relative to maximum (p_M) and minimum (p_m) fragment pixel depths: $d = (e^z - e^{p_m})/(e^{p_M} - e^{p_m})$.
- Total number of pixel fragments.
- Accumulated pixel absorbance $T(b_0)$.
- Moment-based transmittance accumulated for each of the four generating functions (b_1, b_2, b_3, b_4).

The input features, including *power moments* and *relative fragment depths*, offer detailed depth and transmittance information throughout the pixel space. This enhances the ability of the neural net to effectively map these inputs to transmittance values. The normalization of all depth inputs to $[0,1]$ enhances the generalization across various scene configurations, particularly in scenarios with closely aligned depth values. The *total number of fragments* and *accumulated absorbance* guide the distribution of transmittance among individual fragments.

Our method aims to enhance visual quality without sacrificing real-time performance. To achieve this, we use per-fragment inferences and employ a compact neural network architecture for real-time processing in complex scenes.

3.2. Training Dataset

Our neural network underwent supervised learning, where per-fragment input features were mapped to transmittance values. We created a randomly generated synthetic training dataset, incorporating diverse scenarios by generating random pixels with varying numbers of fragments (ranging from 2 to 32), fragment depth ranges (0.3 to 1.0), and values (0.1 to 0.5). The precise transmittance (training target) was computed using Equation 1.

To construct a robust synthetic training dataset we created 250K pairs of per-fragment inputs and the corresponding target per-fragment exact transmittance values. This dataset encompasses a wide range of inputs, ensuring that our neural network can generalize effectively across various scene configurations. The inclusion of varying depth and opacity complexities is crucial for the networks ability to accurately determine transmittance values. For training, Mean-Squared Error (MSE) is used as a loss function and Adam with a learning rate of 0.001 as the optimizer. We have trained our network for 1000 epochs using a batch size of 8192 using TensorFlow in around three hours.

4. Implementation

Our method requires two geometry passes and a fullscreen pass (Fig. 2), (i) a geometry pass for *Moment Calculation*, that computes and stores pixel transmittance using four power moments, total absorbance (b_0), and min/max pixel depths, (ii) a geometry pass *Transmittance Estimation* that uses the input features and computes the per-fragment transmittance using the neural network, and (iii) a full-screen pass for *Color Calculation*, that computes the final OIT color (Eq. 7). Our method utilizes four textures:

- a 32-bit float RGBA texture to store color multiplied by its transmittance and opacity.
- a 32-bit float RGBA texture to accumulate moments.
- a 32-bit float RG texture to accumulate the total absorbance (b_0) and the normalization value.
- a 16-bit float RG texture to store the minimum and maximum per pixel depths.

For a comprehensive experimental evaluation of Neural Moment Transparency (NMT) (Sec. 5), we implemented both Moment-Based Order-Independent Transparency (MB) [Sha18; MKKP18] and Deep and Fast Order-Independent Transparency (DFA) [TVF24]. Our approach demonstrates comparable memory requirements (352 bits per-pixel), which is in line with competing methods (DFA: 352 bits per pixel, MB: 320 bits per pixel). Despite this, our method showcases superior visual effects (Fig. 4) with respect to ground-truth images produced by an A-buffer (AB) [YHGT10] while maintaining real-time performance (Tab. 1). All methods were implemented in OpenGL 4.6, using interlocks and atomic operations where needed, with similar texture sizes of 32-bit floats. Shader source code of this work is available online. [†]

5. Experimental Evaluation

We present an experimental evaluation of NMT as compared to DFA [TVF24], and MB [MKKP18; Sha18] with regards to performance and image quality. We have performed experiments on several scenes, with varying depth complexity D *(maximum per-pixel fragments)* and opacity using a 1920×1080 viewport for all experiments using an NVIDIA RTX 2080 Super GPU.

As our method predicts a per-fragment transmittance, it is scene independent and it is expected to work in all contexts. In general, our method outperforms the competing methods in terms of quality while maintaining real-time performance.

Quality. Our method provides the best image quality in terms of ꟻLIP [ANA*20] mean error compared to the aforementioned competing methods (Fig. 1, 4). *MB* produces the lowest quality images, which could be attributed to its transmittance reconstruction inaccuracies. We used four power moments as our test scenario, however several variants (number of power moments, trigonometric moments) could be used to further improve visual quality with tradeoffs in performance and memory requirements. *DFA* further improves image quality due to its ability to directly approximate pixel RGB color, using input features that cover the entire pixel depth and provide color information. It also uses the exact color

[†] https://github.com/gtsopus/NMT

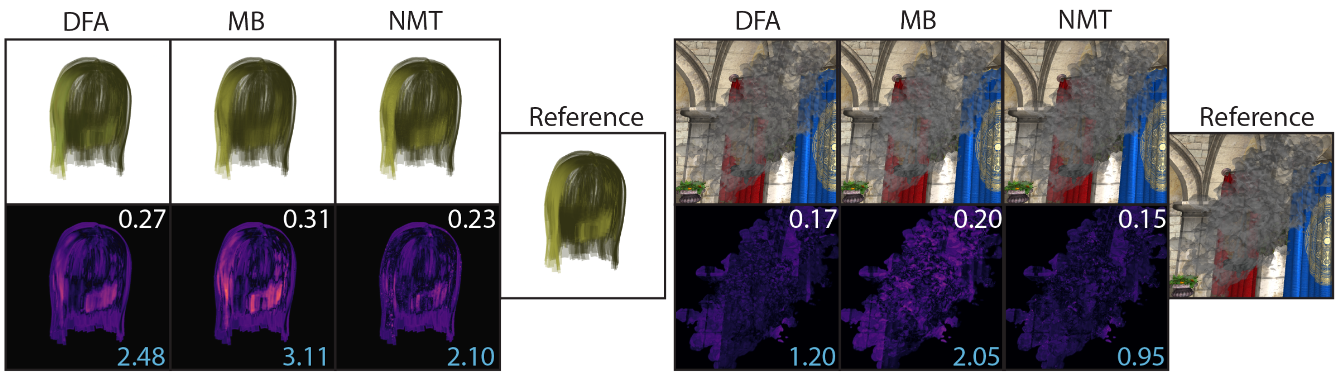

Figure 4: *Based on ꟻLIP (top right) and MSE (x10^{-4}, bottom right), our method achieves better visual quality, by improving transmittance approximation, in scenes with varying depth complexity D and opacity a. Left: D = 171, a = 0.2, Right: D = 24, a = 0.3.*

of the two closest fragments to improve image quality. Finally, our *NMT* outperformed the aforementioned approaches due to its ability to better infer fragment transmittance using a neural network. The neural network input features provide information regarding transmittance/absorbance (moments), and fragment depth allowing the neural network to detect the space positioning of each fragment and assign the appropriate transmittance value.

Performance. As illustrated in Table 1, our method achieves real-time performance in scenes of varying depth complexity, with only a minor performance overhead that depends on the number of fragments and the inference time of the neural network (Fig. 2).

Table 1: *NMT exhibits a slightly decreased performance in scenes with depth complexity D. Moment Calculation pass timings in parentheses. Timings are in ms.*

Scene (D)	AB	DFA	MB	NMT
Sibenik (31)	15.0	3.0	4.3	5.2 (3.9)
Engine (52)	5.3	1.3	2.0	2.3 (1.3)
Hair (172)	17.2	3.3	6.7	8.0 (6.4)
Smoke (24)	7.9	2.6	3.9	4.6 (3.5)

6. Conclusions

Our method employs a neural network to directly infer the transmittance of each pixel allowing us to effectively render approximate OIT in real time. Our method produces superior results compared to the approximate MB method [MKKP18] utilizing four power moments. This is attributed to our neural network-based approach, which numerically refines the final result by incorporating additional features. As a future research direction, we aim to investigate the behavior of a variety of other moment generating functions, such as trigonometric moments, to further increase the visual quality with a potentially small overhead in computation time and memory. Finally, we plan to leverage neural networks to address the rendering challenges associated with various effects (e.g., shadow mapping).

References

[ANA*20] Andersson, Pontus, Nilsson, Jim, Akenine-Möller, Tomas, et al. “ꟻLIP: A Difference Evaluator for Alternating Images”. *Proc. ACM Comput. Graph. Interact. Tech.* 3.2 (Aug. 2020). doi: 10.1145/3406183 3.

[BM08] Bavoil, Louis and Myers, Kevin. “Order Independent Transparency with Dual Depth Peeling”. 2008 2.

[MB13] McGuire, Morgan and Bavoil, Louis. “Weighted Blended Order-Independent Transparency”. *JCGT* 2.2 (Dec. 2013), 122–141. url: http://jcgt.org/published/0002/02/09/ 2.

[Mes07] Meshkin, Houman. “Sort-independent Alpha Blending”. *GDC Talk* 2.4 (2007) 2.

[MKKP18] Münstermann, Cedrick, Krumpen, Stefan, Klein, Reinhard, and Peters, Christoph. “Moment-Based Order-Independent Transparency”. *Proc. ACM Comput. Graph. Interact. Tech.* 1.1 (July 2018). doi: 10.1145/3203206 1–4.

[PD84] Porter, Thomas and Duff, Tom. “Compositing Digital Images”. *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’84. New York, NY, USA: ACM, 1984, 253–259. doi: 10.1145/800031.808606 1, 2.

[Sha18] Sharpe, Brian. “Moment Transparency”. *High-Performance Graphics*. Vancouver, British Columbia, Canada: ACM, 2018, 1–4. doi: 10.1145/3231578.3231585 1–3.

[TFV22] Tsopouridis, Grigoris, Fudos, Ioannis, and Vasilakis, Andreas A. “Deep Hybrid Order-Independent Transparency”. *The Visual Computer* 38.9 (2022), 3289–3300. doi: 10.1007/s00371-022-02562-7 2.

[TVF24] Tsopouridis, Grigoris, Vasilakis, Andreas A., and Fudos, Ioannis. “Deep and Fast Approximate Order Independent Transparency”. *Computer Graphics Forum* (2024), e15071. doi: 10.1111/cgf.15071 2, 3.

[VVP20] Vasilakis, Andreas A., Vardis, Kostas, and Papaioannou, Georgios. “A Survey of Multifragment Rendering”. *Computer Graphics Forum* 39.2 (2020), 623–642. doi: 10.1111/cgf.14019 1, 2.

[Wym16] Wyman, Chris. “Exploring and Expanding the Continuum of OIT Algorithms”. *Proceedings of High Performance Graphics*. HPG ’16. Dublin, Ireland: Eurographics Association, 2016, 1–11. doi: 10.2312/hpg.20161187 1.

[YHGT10] Yang, Jason C., Hensley, Justin, Grün, Holger, and Thibieroz, Nicolas. “Real-Time Concurrent Linked List Construction on the GPU”. *Computer Graphics Forum* 29.4 (2010), 1297–1304. doi: 10.1111/j.1467-8659.2010.01725.x 1, 3.