DOI: 10.1111/cgf.15071

COMPUTER GRAPHICS *forum*
Volume 43 (2024), number 6, e15071

Deep and Fast Approximate Order Independent Transparency

Grigoris Tsopouridis,[1] Andreas A. Vasilakis[2,3] and Ioannis Fudos[1]

[1]Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece
g.tsopouridis@uoi.gr, fudos@cs.uoi.gr
[2]Department of Informatics, Athens University of Economics and Business, Athens, Greece
abasilak@aueb.gr
[3]Phasmatic, Corfu, Greece

Abstract

We present a machine learning approach for efficiently computing order independent transparency (OIT) by deploying a light weight neural network implemented fully on shaders. Our method is fast, requires a small constant amount of memory (depends only on the screen resolution and not on the number of triangles or transparent layers), is more accurate as compared to previous approximate methods, works for every scene without setup and is portable to all platforms running even with commodity GPUs. Our method requires a rendering pass to extract all features that are subsequently used to predict the overall OIT pixel colour with a pre-trained neural network. We provide a comparative experimental evaluation and shader source code of all methods for reproduction of the experiments.

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

CCS Concepts: •Computing methodologies → 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 visual information [MCTB11, Wym16]. Transparent objects in a scene, such as glass, water, or smoke, can often overlap, resulting in incorrect or unexpected rendering outputs. 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 of a pixel p by their depth z: $[f_n, f_{n-1}, \ldots, f_1]$, so that $z_n \leq z_{n-1} \leq \cdots \leq z_1$.
(ii) computing the pixel colour $C_p = C_f(n)$ by the following recursive formula (also known as *over compositing operator* [PD84]):

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

where C_b is the background colour, C_i and a_i are the colour and opacity of fragment f_i and $C_f(i)$ is the colour in front of fragment f_i. Therefore, $C_f(n)$ is the colour of pixel p.

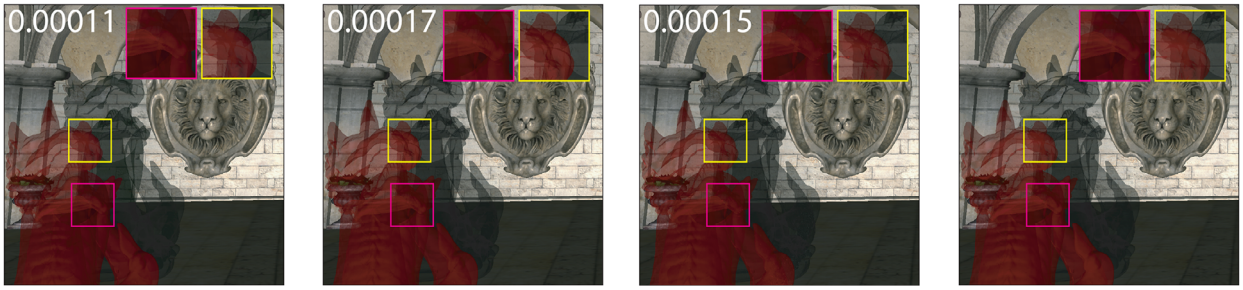
The first step requires to store all fragment information in a per pixel buffer and then to sort all fragments according to their depth value. To achieve real-time performance several algorithms have been used that maintain a memory space per pixel, where information regarding *all* fragments for this pixel is stored. This rendering pipeline is known as A-buffer [Car84] and can be implemented either by allocating a predetermined memory portion per pixel that can accommodate all fragments [LHLW10] which is fast but memory demanding, or by creating per pixel linked-lists [YHGT10, VF12], which is slower but allocates exactly the required memory.

To approximate OIT with interactive performance and reasonable memory requirements the following categories of approaches have been explored:

(a) *average OIT* approaches [Mes07, BM08] utilize only global average information such as average opacity and average colour. They work on the pixel level, are fast but lack accuracy compared to k-buffer and blended approaches.
(b) *blended OIT* approaches [MB13, MKKP18, Sha18, FEE21] use information which is passed on by each fragment and then is averaged or summed at the pixel level. The processing by each fragment is conducted via a blending function. They are

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd. This is an open access article under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org



(a) Deep and Fast Approximate OIT (b) Weighted Blended OIT (c) Moment Based OIT (d) Ground Truth

Figure 1: *Our deep learning approach (a) produces visually convincing transparency results with respect to ground-truth images produced by an A-buffer [YHGT10] (d), qualitatively outperforming previous approximate methods [MB13, MKKP18] (b,c) using MSE as a metric.*

fast and provide good accuracy in the expense of a small processing overhead per fragment.

(c) *k-buffer and multifragment* OIT approaches use a k-subset selection of all generated fragments [Wym16, VPF15, VVPM17] or the entire set of fragments [SML11]. The k value can be fixed, variable or adaptive and may refer to k closest or a random subset of fragments. They sacrifice performance and memory to offer high quality working with relatively advanced GPUs.

(d) *hybrid OIT* approaches [MCTB13, SV14, TFV22] combine k-buffer approaches with average or blended OIT approaches for the rest of the fragments that are not part of the k-buffer set (also known as the *tail* part).

In this work, we present a deep learning approach to robustly compute OIT that outperforms all previous blended and average OIT approaches. Our Deep and Fast Approximate OIT (DFAOIT) receives as input features, global per pixel measures such as average colour and opacity. Using this input, a pre-trained neural network outputs the final pixel transparency colour at once. The neural network inference has been implemented to run efficiently on a fragment shader, without requirements for state of the art graphics hardware.

The approach introduced in this work makes the following technical contributions:

- To our knowledge, this is the first approach to use deep learning for approximating OIT with global per pixel measures.
- Our approach offers comparable performance with approximate average or blended OIT approaches and offers $20\% - 80\%$ better accuracy (using the mean square pixel colour error) compared to the most competent previous approaches.
- We offer a cross-platform implementation for OIT rendering, where no hardware that guarantees a deterministic pipeline order is required.
- Our method requires no parameters to be determined or tuned by the user before rendering.
- It can also be used for improving the tail part of hybrid OIT approaches.

The structure of the rest of this paper is as follows. Section 2 provides comparison with state of the art. Section 3 outlines our DFAOIT method and Section 4 offers implementation details. Section 5 conducts a thorough comparative experimental evaluation. Finally, Section 6 offers conclusions and future research directions.

2. Related Work

2.1. Rendering transparency

Many games and simulations use transparent objects like glass, leaves, hair or fog to add realism and depth to their environments [MJ09, MET*18, SSNK21]. Existing techniques often require separating opaque and transparent geometry into separate rendering passes and the respective output buffers are merged together to form the final blended colour [SBF15]. Traditionally, a standard set of compositing operators is applied to combine transparent surfaces [PD84]. However, this process is challenging since it requires an order-dependent processing on a per-geometry basis (per frame) to achieve the desired blending result [HCNS18]. Moreover, an inaccurate solution in cases of intersecting or circularly overlapping geometry is also possible. Advances in graphics hardware have fuelled an explosion of research and development in the field of OIT [MCTB11, Wym16] to approximate alpha compositing in screen space without explicit geometry ordering. Without loss of generality, these methods can be classified into two board categories: (i) *blended* (categories (b) and (c) of Section 1) and (ii) *multifragment* transparency rendering methods (categories (a) and (d) of Section 1). Despite the recent trends of neural rendering [TFT*20], and its application to enhance screen-space global illumination [NAM*17, TF17, ZXL*20, DNSD22] and subsurface scattering [VKJ19, LHW21], to our knowledge we are the first to exploit deep learning for efficiently computing OIT effects.

2.2. Blended transparency rendering

Real-time 3D applications, such as games, require acceptable transparency results within a limited computation time cost while respecting the current graphics rendering engine pipeline. Blended OIT methods focus on portability and speed by adjusting the over compositing operator (Equation 1) [PD84] so that its arguments commute. *Weighted-sum alpha blending* [Mes07] was the first to introduce blended OIT by redefining the blending formulation as

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

follows:

$$C_p = \sum_{i=1}^{n} C_i a_i + C_0(1 - \sum_{i=1}^{n} a_i) \quad (2)$$

While the alpha blending equation was simplified in order to ignore the order-dependent terms, this method produces unacceptable quality (overly dark or bright colour) for high opacity values. Following the same idea, *Weighted-average alpha blending* [BM08] offers a better OIT approximation by distributing the transmittance equally among the transparent fragments:

$$C_p = C_{avg}(1 - (1 - a_{avg})^n) + C_0(1 - a_{avg})^n \quad (3)$$

$$C_{avg} = \frac{\sum_{i=1}^{n} C_i a_i}{\sum_{i=1}^{n} a_i}, \text{ and } a_{avg} = \frac{\sum_{i=1}^{n} a_i}{n} \quad (4)$$

The blending average method overcomes colour saturation exhibited in the previous method, but comes with poor transparency output on scenes with high opacity values (e.g. avoids to properly support invisible surfaces). *Weighted-blended alpha blending* [MB13] extended the previous operator to compute the exact coverage of the background, which also improves the colour term. To create an occlusion cue between transparent surfaces, the operator was further colour-weighted to include depth information. The weights for each fragment f_i are computed from a monotonic decreasing function $w(z_i, a_i)$ (several proposed in the original paper). The operator can be formulated as:

$$C_p = C_{wavg} a_{net} + C_0(1 - a_{net}) \quad (5)$$

$$C_{wavg} = \frac{\sum_{i=1}^{n} C_i a_i w(z_i, a_i)}{\sum_{i=1}^{n} a_i w(z_i, a_i)}, \text{ and } a_{net} = 1 - \prod_{i=1}^{n}(1 - a_i) \quad (6)$$

In all cases, weights have to be carefully tuned for each scene in order to provide plausible results. Recently, Friederichs *et al.* extended the previous algorithm via a layering approach [FEE21]. They divided the view frustum into multiple depth intervals on which the weighted-blended OIT algorithm is applied separately leading to more robust depth cues and improved colour precision. However, it requires excessive memory demands due to linked lists construction [YHGT10] and it struggles to deliver good performance when a high number of bins is required.

Moment transparency [Sha18, MKKP18] achieves better results in real-time by replacing the empirically determined weights $w(z_i, a_i)$ with *moments* to capture and reconstruct the transmittance function. Several variants (number of power moments, overestimation, low-precision buffers) were also introduced on each with trade-offs between quality, performance, and robustness.

2.3. Multifragment transparency rendering

There has been a significant increase in the number of techniques that use multiple fragments to create realistic transparency [VVP20]. These methods have the advantage of compositing more geometry layers, by preserving more information from the fragment sampling domain. The multifragment process can be classified into two categories based on the maximum number of fragment samples per pixel captured in a single iteration: (i) *depth-peeling* and (ii) *A-buffer*.

Depth-peeling methods iteratively extract fragment information in a depth-sorted manner. Depending on the specific algorithm, each rendering iteration extracts a batch of fragments with a guaranteed depth order. This process continues until a sufficient visual quality has been achieved or all fragments have been extracted. The most prominent method is probably the *front-to-back depth-peeling* [Eve01], which involves rendering the geometry multiple times and extracting a *single* fragment layer in ascending depth order for each iteration. While several extensions have been introduced to speed up the fragment peeling process by extracting multiple fragments per iteration [BM08, LHLW09, VF13, MMNL16], depth-peeling methods fail to behave interactively in complex environments since the geometry is repeatedly rasterized multiple times. By exploiting more fragment storage for fewer rendering passes, *multilayer depth peeling* (also well-known as *k-buffer* [BCL*07] methods) can be considered as the most preferred framework for optimal fragment k-subset selection. These approaches accumulate transparent geometry with the help of a fixed number of layers [SML11, SV14, Wym16]. To approximate the ground-truth OIT solution, a fragment merging strategy is performed in a streaming fashion. The original methods required specialized hardware modifications in pixel synchronization [Sal13, VF14] that were made available later on, limiting the portability of these methods. Finally, Maule *et al.* [MCTB13] introduced a *hybrid transparency (HT)* method, in which the closest transparent layers are accurately composited using a k-buffer, while the remaining ones are quickly approximated using blended transparency techniques. This allows for a balance of accuracy and efficiency in the compositing process. Recently, Tsopouridis *et al.* improved this work by dynamically distributing more fragments in pixels that contribute more to the visual result guided by a deep learning prediction mechanism [TFV22].

Historically, *A-buffer* method has been the best solution for handling multiple transparent layers of rasterized geometry [Car84]. It uses multiple per-pixel depth-order fragment linked lists to accurately simulate transparency on the GPU [YHGT10, VF12]. However, one of the main drawbacks of A-buffer-based solutions is their high memory requirements. Because the number of fragments generated during the rendering process is unpredictable, the memory needed to store them can quickly become overwhelming, making A-buffer unsuitable for interactive applications on constrained rendering devices [VVP20].

Finally, Sitzmann *et al.* [STH*19] introduce a deep learning approach for creating a persistent 3D feature embedding of a scene for view synthesis without having to explicitly model its geometry. This method supports occlusion but not transparency. In this respect, it may be worthwhile to explore if there are appropriate extensions that can provide deep representations with embedded OIT information.

3. Deep and Fast Approximate OIT

To derive a formula for approximating the OIT colour $C_f(n)$ of the entire pixel p, by average fragment quantities, we expand

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

Equation (1):

$$C_f(n) = a_n C_n + (1 - a_n) a_{n-1} C_{n-1} + \dots + a_1 C_1 \prod_{i=2}^{n} (1 - a_i) + C_b \prod_{i=1}^{n} (1 - a_i) \quad (7)$$

and then we derive the approximation:

$$\widehat{C}_f(n) = h(a_{avg}, C_{avg}, C_{acc}, g(n, 1)) + C_b \prod_{i=1}^{n} (1 - a_i) \quad (8)$$

with $g(i, k) = a_i C_i + (1 - a_i) a_{i-k} C_{i-k}$, where C_{avg} and a_{avg} is the average colour and average opacity of the farthest $n - 2$ fragments of pixel p, C_{acc} is the sum of the premultiplied alpha colours $\sum_{i=1}^{n} a_i C_i$. Finally, $g(n, 1)$ is the combined colour of the first two fragments and h is a function that approximates the sum of terms

$$\sum_{i=1}^{n} a_i C_i \prod_{k=i+1}^{n} (1 - a_k)$$

For example, for $n = 4$ where we have four fragments, we may use the following function $h(x, y, z, w) = w + (1 - x)^2 (z - 2xy - x^2 y)$. The following shows that this choice for function h provides a valid approximation (the parts that are being approximated are highlighted by the corresponding colour):

$$\begin{aligned}
\widehat{C}_f(4) = \; & a_4 C4 + (1 - a_4) a_3 C_3 + \\
& (1 - a_{avg})^2 (a_4 C_4 + a_3 C_3 + a_2 C_2 + a_1 C_1 \\
& - 2 a_{avg} C_{avg} - a_{avg}^2 C_{avg}) \\
& + (1 - a_4)(1 - a_3)(1 - a_2)(1 - a_1) C_b \\
\approx \; & a_4 C_4 + (1 - a_4) a_3 C_3 + \\
& (1 - a_4)(1 - a_3)(a_4 C_4 + a_3 C_3 + a_2 C_2 + a_1 C_1 \\
& - a_4 C_4 - a_3 C_3 - a_2 a_1 C_1) \\
& + (1 - a_4)(1 - a_3)(1 - a_2)(1 - a_1) C_b \\
= \; & C_f(4)
\end{aligned}$$

The fact that there is a valid non trivial approximation for $n = 4$ substantiates the following:

- it is evidence that there may be a function h, that given the four inputs a_{avg}, C_{avg}, C_{acc}, $g(n, 1)$, which approximates the combined colour of all fragments (without the background) for any n.
- since the approximation function h for $n = 4$ is a non trivial polynomial function of degree 5, it is reasonable to use a neural network to determine function h for any n.

Therefore, we select as initial candidate input features the number of fragments, the average opacity and colour of the farthest $n - 2$ fragments a_{avg}, C_{avg}, the sum of C_{acc}, and the combined colour of the first two fragments $g(n, 1)$. The term of the background colour can always be added at the final step using the exact formula: $C_b \prod_{i=1}^{n} (1 - a_i)$.

In the rest of this section, we describe the training and inference phases of our deep and fast OIT technique. We introduce a 'one network for one effect' approach for training the network, that maps the per-pixel inputs to an OIT RGB colour. We first describe the neural network architecture (Section 3.1) along with the selection of its input features and training parameters. Then, for the training phase (Section 3.2), we explain how we acquire the necessary screen-space training data for the supervised learning process. Finally, we discuss the limitations of various network architectures and input features that are not utilized by our method (Section 3.3).

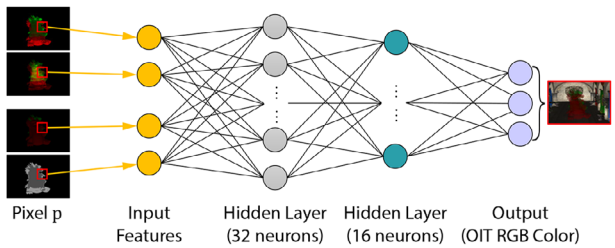


Figure 2: *A visualization of our neural network that maps the input features of pixel p into an OIT RGB colour output.*

3.1. Network architecture and feature selection

We have selected a compact fully connected multilayer perceptron to predict per-pixel OIT RGB colours, given 10 float input features (Figure 2). The neural network is composed of an input, an output, and two hidden layers. The hidden layers are composed of 32 and 16 neurons respectively, and use ReLU as their activation function. The output layer returns three float values $C_p = (r_p, g_p, b_p) \in [0, 1]$ that represents the predicted OIT RGB colour, using the sigmoid activation function. For training the network, Mean-Squared Error (MSE) is used as a loss function and Adam [KB14] with a learning rate of 0.001 as an optimizer. Finally, we train our network for 3000 epochs using a batch size of 4096. All input features are in the range of [0,1], except the accumulated premultiplied colour.

The neural network expects the following 10 per-pixel float features as input:

- Correct OIT RGB colour of the two front fragments: $g(n, 1) = a_n C_n + (1 - a_n) a_{n-1} C_{n-1}$.
- Average opacity, excluding the front two fragments: $a_{avg} = \sum_{i=1}^{n-2} a_i / n$.
- Average RGB colour, excluding the front two fragments: $C_{avg} = \sum_{i=1}^{n-2} C_i / n$.
- Accumulated premultiplied alpha RGB colour: $C_{acc} = \sum_{i=1}^{n} a_i C_i$.

In most cases, the larger the number of other fragments between a fragment and the camera, the smaller impact it has on the final colour since it is being attenuated significantly (Equations 1 and 7).

Consequently, the input features must provide colour information throughout the whole depth of a pixel to enable the neural network to map them to an OIT colour. The exact OIT colour of the first two fragments is vital to the final OIT colour and it allows the neural network to better approximate the exact OIT and retain finer details. The average colour C_{avg} and opacity α_{avg} provide the contribution of the remaining fragments to the final colour, playing a complementary role to $g(n, 1)$. Finally, the accumulated alpha-multiplied colour C_{acc} gives a general insight regarding the colour and opacity

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

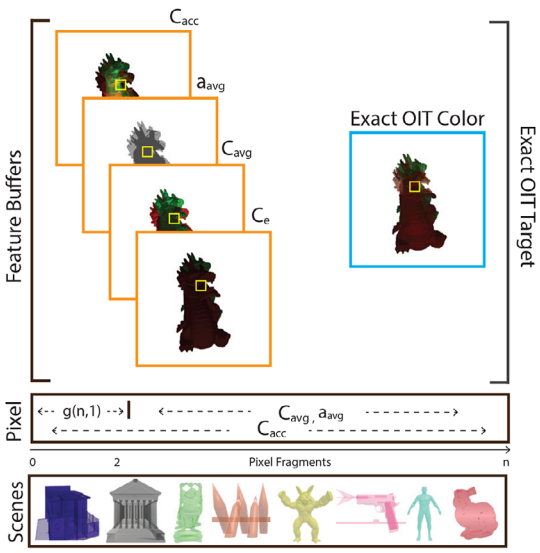


Figure 3: *Visualizing image training pairs. The per-pixel input features (small yellow squares) are generated using rasterization (top). The targets can be generated correctly by A-buffer [YHGT10] (cyan frame) using the training scenes at the (bottom). The input features (orange frames at the top) aim to gather colour information throughout the entire pixel (middle).*

of the pixel. All these features assist in determining function h of Equation (8).

3.2. Training dataset

We trained our neural network using supervised learning, mapping the per-pixel input features to an RGB colour. The training examples are generated by exporting the information from all generated fragments (e.g. an A-buffer [YHGT10]) and computing the input features as well as the exact OIT RGB colour (target). To achieve this, we developed a dataset containing pairs of per-pixel inputs and the corresponding per-pixel exact OIT RGB colour (Figure 3). The dataset comprises input features from various fully transparent scenes with differing depth complexity (10 to 50 fragments). Each scene features a single multi-coloured 3D object with a random per-pixel opacity $\alpha \in [0.01, 0.85]$, ensuring that our network can generalize over a wide range of opacity values. We utilized eight different scenes, resulting in a total of 5 million training examples. Our neural network is scene agnostic, as it is trained on a per-pixel basis using only depth and colour information, and is not dependent on scene geometries. This establishes a robust generalization capability, as evidenced by the clear divergence in colour and depth complexities between the training (Figure 3) and testing scenes (Figure 9, and Figure 10).

The training dataset plays a major role in determining the final OIT result, as examples with different colours, depth complexities and opacities must be included. Our method required a short training time of approximately 3 hours using a NVIDIA RTX 2080 Super GPU.

Table 1: *Comparing* DFAOIT$_k$ *variants with different closest fragment counts* ($k = 2, 4, 8$) *as input features, showcasing the tradeoff between performance and quality.*

Method	DFAOIT$_2$	DFAOIT$_4$	DFAOIT$_8$
Engine (36)			
MSE	0.00026	0.00014	0.00011
Time (ms)	2.9	6.1	14.2
Memory (bits per pixel)	352	480	736
Sibenik (21)			
MSE	0.0010	0.00057	0.00038
Time (ms)	3	5.7	12.4
Memory (bits per pixel)	352	480	736

3.3. Ablation study

As our method targets real-time performance and relatively low memory requirements, we were limited in terms of network architectures and input features. In this section, we discuss the limitations of various other network architectures and features. Regarding network architectures, we discarded the employment of CNN early on, as while using a CNN may be beneficial in terms of quality due to its ability to correlate neighbouring pixel information, it is quite demanding in terms of computations and memory even when modern hardware is exploited [DNSD22] and hard to implement in a fragment shader. Additionally, we observed that using larger hidden layers (up to 256 neurons) slightly improve overall accuracy, but are not worth the performance and memory overhead. For example, doubling the layer size (64 and 32 neurons) only improved the quality by 6%, but doubled the weights memory cost, as they need to be stored in a shader, and inference time.

Neural network inputs should be fast and easy to compute (even by commodity graphics hardware). In a way, our method splits pixel space into two parts, the exact front OIT part C_e, and the supplementary approximation tail part C_{acc}, C_{avg}, a_{avg}. To improve quality, we can better represent colour changes in pixel space by further splitting the depth of the pixel into three or four (or more) depth buckets (inspired by [FEE21]), with the first bucket containing exact OIT information, and the remaining buckets containing average colour and opacity, each bucket should cover an increasingly larger depth partition, as farthest fragments do not contribute much to the final colour. While this could potentially increase quality, it is a challenge to find optimal ways to split space, and it greatly increases memory and performance overhead, deeming it unsuitable for our method.

Additionally, it is possible to include the exact colour of more than two fragments. We consider this unsuitable for our method as it significantly increases memory and computational requirements. However, this enables our method to provide diverse options, allowing users to finely tune the trade-off between performance and quality. Notably, higher k values yield superior quality, particularly in scenes with greater depth complexity (Table 1).

We experimented with integrating an existing, approximate OIT method [Weighted Average (WAVG)], as an additional input feature. However, this led to an increase in computational and memory demands, encompassing both the storage for WAVG and the

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

supplementary feature storage, without yielding a significant enhancement in visual quality.

Incorporating pixel depth metrics (mean, minimum, maximum) and pixel transmittance data as input features did not yield an enhancement in the quality of our predictions. This outcome could be attributed to the inherent nature of per-pixel colour approximation methods, such as ours. Nonetheless, it is important to note that these features may prove valuable in scenarios that employ a per-fragment colour approximation approach. In such instances, the integration of both local (pixel-level) and global (scene-level) features could potentially address the issue, without introducing a significant performance overhead as these features might offer valuable insights for the ordering of fragments within the scene. In the future, one could possibly employ machine learning for deriving a per-fragment *colour weight* function $f(z_i, C_i)$, better representing the occlusion of objects [MB13].

Finally, we have experimented with alternative functions such as leaky RELU and ELU. The ELU did not yield promising result, while the leaky RELU offers in some cases a small accuracy improvement (around 5%) with proper adjustment of the negative slope. However, the leaky RELU needs more experimentation and tuning before it is deployed on a universal shader.

4. Implementation

Our method requires two passes (Figure 4), (i) a geometry pass for *Feature Extraction*, that computes and stores the neural network input features, and (ii) a full-screen pass for *OIT Colour Inference*, that uses the input features and the stored weights and biases of the neural network to compute the predicted RGB OIT colour. *Feature Extraction* accumulates the colour, premultiplied colour, opacity as well as the colour and opacity of the first two fragments. We extract the two closest to the camera fragments using a custom depth comparison implementation with interlocks, however other, more advanced ways can be employed such as k-buffer [BCL*07, VF14], depth-peeling [Eve01] or by utilizing minimum separation and temporal information [MMNL16].

The neural network approximates the colour C_t of transparent objects, excluding the contribution of the background colour C_b, which is accounted for in *OIT Colour Inference* using the background product. Thus, the final colour C_p is computed as follows:

$$C_p = C_t + C_b \prod_{i=1}^{n} (1 - a_i) \tag{9}$$

Our method requires four textures (Figure 5):

- an RGB 32-bit unsigned integer texture that contains the number of fragments per pixel, which is used to compute the average colours and opacity, and the depths of the two closest fragments.
- an RGBA texture, using 16-bits per value, that accumulates colour and alpha values.
- an RGBA texture, using 16-bits per value, that accumulates colours multiplied by their opacities ($a_i C_i$) and the background product $\prod_{i=1}^{n}(1 - a_i)$.
- a two-layered RGBA texture that uses 16-bits per value, storing the colours of the two closest fragments.

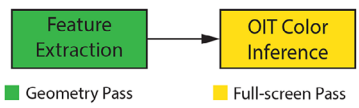


Figure 4: *Rendering flow diagram of our two-stage method.*

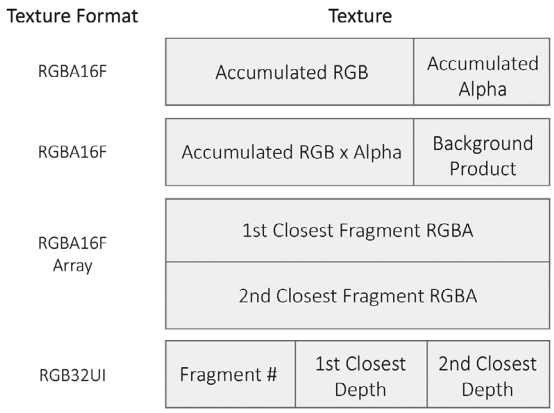


Figure 5: *Texture contents and sizes of DFAOIT.*

To provide a fair comparative experimental evaluation (Section 5), we implemented Weighted Sum (WSUM) [Mes07] using an RGBA texture with 16 bits per value, that accumulates colour and alpha values. Both Weighted Blended (WBOIT) [MB13] and WAVG [BM08] allocate two textures, an RGBA accumulation texture (32-bits per value) that accumulates colour, and a 32-bit unsigned int texture that stores the background product ($\prod_{i=1}^{n}(1 - a_i)$, WBOIT) and the number of fragments per pixel (WAVG). HT [MCTB13] requires a two (or four, for $k = 4$) layered 16-bit RGBA texture to store the closest fragment colours and opacities, a 32-bit RGBA texture to accumulate colours and opacities which is used for the tail colour computation and, a 32-bit RG texture that stores the background coefficient and the number of per pixel fragments. Moment Based Order-Independent Transparency (MBOIT) [Sha18, MKKP18], requires a 32-bit RGBA texture to store blended colour, a 32-bit RGBA texture to accumulate moments and, a 32-bit RG texture to accumulate the total absorbance and the normalization value. Our method sacrifices more memory budget (352 bits per pixel), compared to the competition (WBOIT, WAVG: 160 bits per pixel - WSUM: 64 bits per pixel - HT ($k = 2$), MBOIT: 320 bits per pixel), for providing superior visual effects with real-time performance (Section 5). However, enhancing HT with more layers $k = 4$, requires 448 bits per pixel.

All methods were implemented in OpenGL 4.6, using interlocks and atomic operations where needed and similar texture sizes of 32-bit float or unsigned integer per value. Note that all methods, including ours, can be easily ported to older OpenGL versions, OpenGL ES and WebGL via core GPU-accelerated *frame buffer blending operations*, thus supporting older GPUs and special

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

purpose software platforms. In the low-end hardware variant of our method, depth-peeling [Eve01] is first used to extract the two closest fragments to the camera and an additional geometry pass is then performed to count the per-pixel fragments and accumulate the colour and opacity neural network input features. Nevertheless, we observed that for our method there is no performance gain or loss when the low-end hardware variant is used. Shader source code of this work, for both low- and high-end graphics hardware, is available online (https://github.com/gtsopus/DFA).

5. Experimental Evaluation

We present an experimental evaluation of DFAOIT as compared to WBOIT [MB13], WAVG [BM08], WSUM [Mes07], HT [MCTB13] and, MBOIT [Sha18] with regards to performance and image quality. We have performed experiments on several scenes, with varying colours, depth complexity and opacity using a 1920×1080 viewport for all experiments except those of Table 3. All experiments were conducted on a system with NVIDIA RTX 2080 Super GPU. The training dataset does not include any of the test scenes.

We have conducted experiments using several scenes that differ with regard to colour and depth complexity *D (maximum per-pixel fragments)*, resulting in diverse input features. As our method follows the 'one network for one effect approach' it is expected to work in every environment without the need of any further fine-tuning, similar to WAVG, HT and WSUM. However, WBOITs [MB13] weights may require fine-tuning, a tedious process that has a significant impact on results. For this reason, for WBOIT we chose one of the author's recommended depth weights: $w(z, a) = a \cdot \max(10^{-2}, 3 \cdot 10^3 (1 - z)^3)$, that fits the requirements, and tuned to derive better results for tested scenes and camera setup. In general, our method outperforms the competing methods in terms of quality while maintaining real-time performance. The ground truth images were generated using a per-pixel linked-lists A-buffer implementation [YHGT10]. Our method was also implemented using blending operations, offering similar results.

Table 2: *DFAOIT shows slightly improved performance compared to prior art in scenes with varying depth complexity D.*

Scene (D)	HT2	HT4	WBOIT	DFAOIT	MBOIT
Dragons (24)	1.3	2.4	1.6	1.7	2.3
Hairball (120)	27.3	41.4	29.4	32.1	45.6
Car (40)	1.9	3.2	2.1	2.5	3.7
Engine (36)	2.5	3.7	2.6	2.9	3.8
Salle de Bain (30)	1.6	3.1	1.8	2.2	2.8
Sibenik (21)	2.0	3.6	2.7	3.0	3.9
Hair (75)	1.4	2.5	0.97	1.55	1.85

WSUM and WAVG perform better but are excluded due to accuracy issues. Timings are measured in ms.

Table 3: *Benchmarks of our method split into feature extraction and inference times (Figure 4).*

Scene (D)	1280×720	1920×1080	2880×1620
Sponza (40)	$1.6 + 1.7 = 3.3$	$3.0 + 2.3 = 5.3$	$4.4 + 3.1 = 7.5$
Sibenik (21)	$1.1 + 1.0 = 2.1$	$1.7 + 1.4 = 3.1$	$2.3 + 2.0 = 4.3$
Engine (36)	$1.1 + 0.8 = 1.9$	$1.7 + 1.2 = 2.9$	$2.5 + 1.6 = 4.1$

DFAOIT performs well even on high resolutions and depth complex scenes (D). Timings are in ms.

5.1. Performance

As illustrated in Table 2, our method performs slightly slower than WBOIT, offering a middle ground between WBOIT and MBOIT, achieving real-time performance in scenes of varying depth complexity, with only a minor performance overhead that depends on the collection of the network input features and the inference time of the neural network (Figure 4). Our method scales and performs well in multiple resolutions (Table 3), with the feature extraction stage depending also on the number of scene fragments and, the neural network inference time depending on the number of transparent pixels.

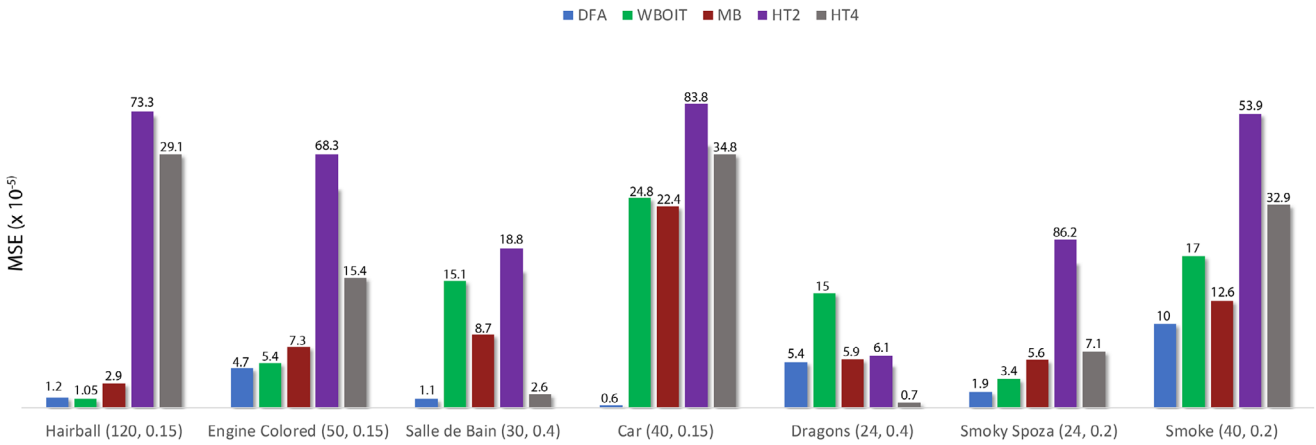


Figure 6: *MSE measurements of the OIT approaches on several scenes with varying depth complexity D, and opacity a. Our method outperforms all methods in most cases, offering better image quality.*

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

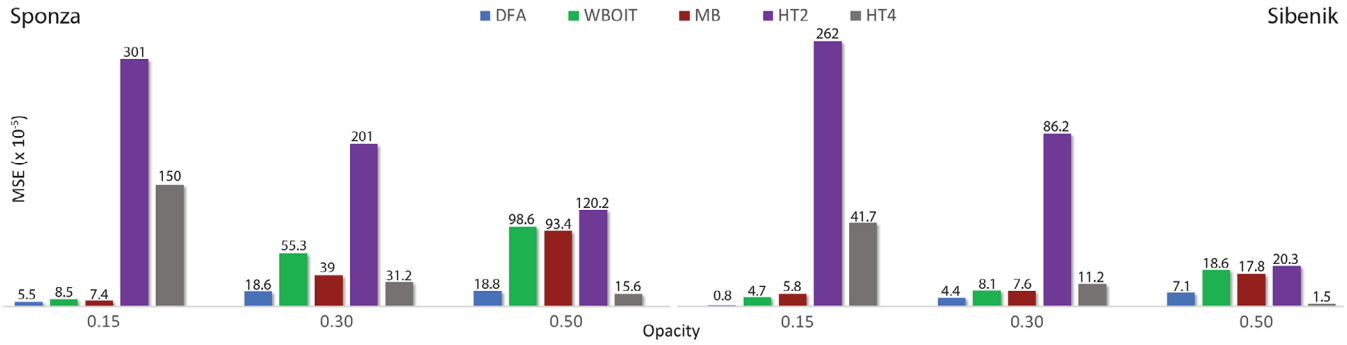


Figure 7: *MSE measurements of the different OIT approaches on two scenes (left: Sponza with $D = 40$, right: Sibenik with $D = 21$) with varying opacity values. We observe that our method yields a better result independent from the opacity values in most cases.*

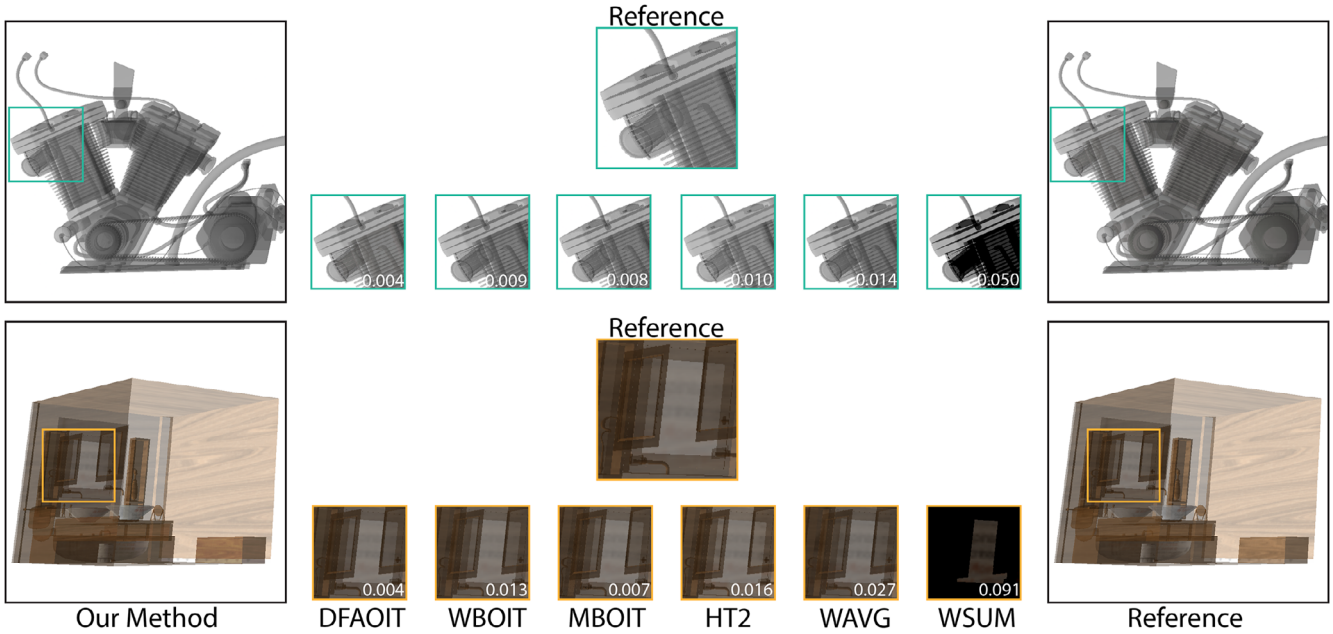


Figure 8: *A quality evaluation between different approaches using ∇LIP error. Our method achieves better quality by retaining higher transmittance detail.*

5.2. Quality

Our method provides the best image quality in terms of MSE compared to the aforementioned competing average (WAVG) and earlier blending OIT approaches (WBOIT) and compared to modern blending (MBOIT) and hybrid (HT) approaches (Figures 1, 6, 7). *WSUM* produces the lowest quality images as the alpha blending equation is over-simplified and ignores the order-dependent terms. For several cases, *WSUM* and *WAVG* yield MSE that is much larger than DFAOIT and therefore were not included in the corresponding figures. However, WAVG offers a better OIT quality as it distributes the transmittance equally among the transparent fragments. It alleviates the colour saturation exhibited by *WSUM* and uses the exact transmittance product for the background, but comes with poor transparency results, as it fails to take into account object occlusion.

HT increases quality by combining both exact and approximate OIT methods. It computes the exact OIT colour of the k closest fragments and approximates the OIT colour (WAVG) for the remaining fragments. We used HT with $k = 2$ (HT2) and $k = 4$ (HT4) as our testing scenarios. Similar to HT2, our method captures the exact colour of the first two fragments, and offers the best trade-off between performance and quality. This aims to show that our method does not just blend the exact and approximate parts, but inferes a better OIT approximation. For HT4, the accuracy of DFAOIT is still much better in most cases, but the performance of HT degrades, becoming much worse than our method (Table 2).

WBOIT further improves visual quality by extending the blending method by using weights, that decrease with the distance to the camera, for the colours. Weights create an occlusion cue between transparent surfaces and work as an occlusion estimator. However,

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

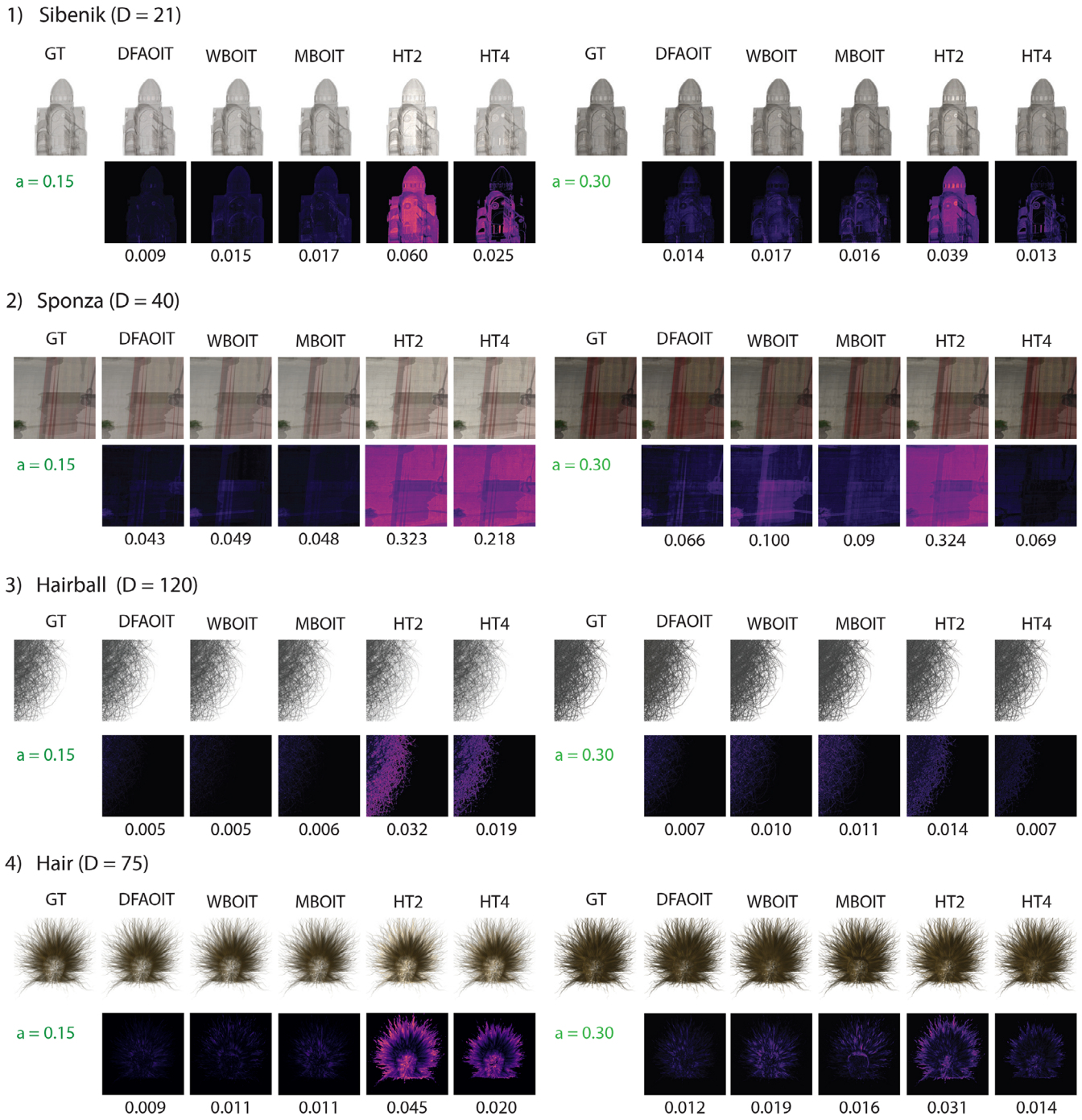


Figure 9: *A quality comparison between the different approaches. An error map (generated using the FLIP tool [ANAM*20]) is computed between each method and the ground truth (GT). Our method displays better OIT results compared to the other methods.*

WBOIT weights are scene and camera specific and may require fine-tuning, a cumbersome process that has a significant impact on the final colour, and does not always provide optimal results.

MBOIT achieves better results by replacing the empirically determined weights of WBOIT with an approximated transmittance, that is captured and reconstructed using moments. We use four power moments as our test scenario, however several variants (number of power moments, trigonometric moments) were also introduced by their authors, each with trade-offs between quality, performance, and memory. The four moments version used here provides comparable times and memory requirements as our method.

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

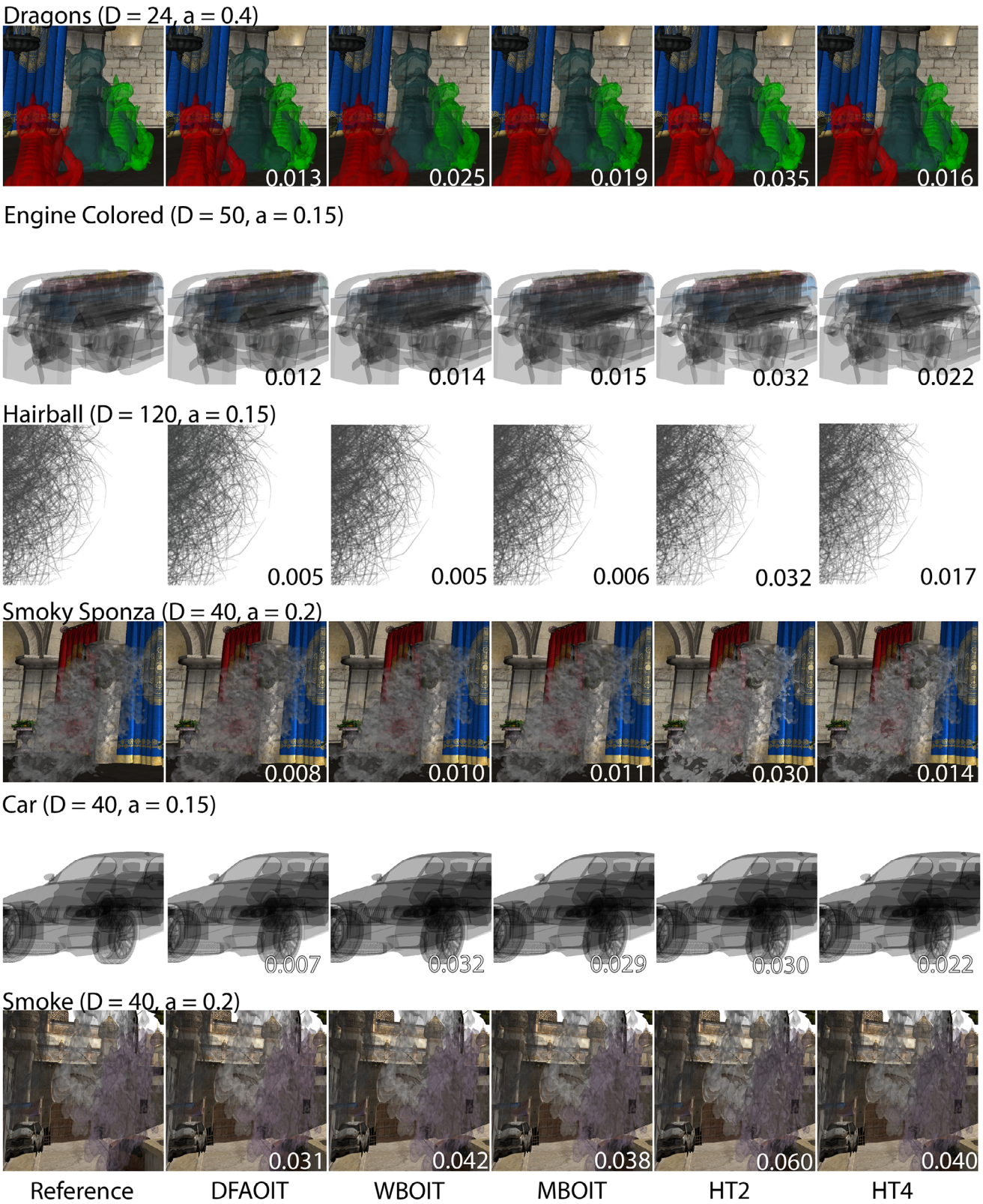


Figure 10: *A quality comparison between the different approaches when compared with the ground truth (GT). Our method offers better OIT quality on various complex scenes based on ϒLIP error.*

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

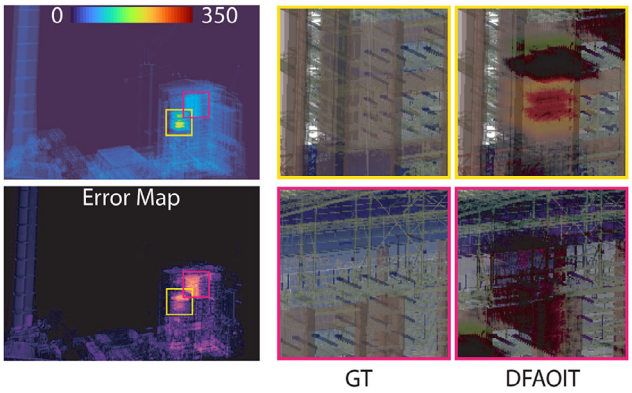


Figure 11: *Our method fails to yield correct results compared to the ground truth (GT) in areas with high depth complexity, as noted on the fragment distribution (top) and FLIP error map (bottom).*

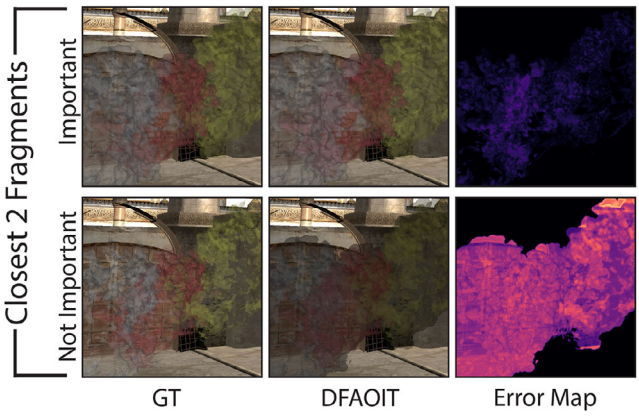


Figure 12: *Our method shows significantly higher errors compared to the ground truth (GT) when the closest two fragments are not important (very low opacity), as noted in the FLIP error maps.*

Finally, *DFAOIT* outperforms the aforementioned approaches due to its ability to determine a more sophisticated colour composition, as the colour blending is performed by the neural network. The neural network input features cover the entire pixel space and provide colour and opacity information. The exact colour of the two closest fragments improves quality and enables our method to retain better details (Figures 1 and 8), while the remaining input features improve the colour approximation of the remaining $n-2$ fragments. Our method outperforms the competing methods in increasingly higher opacity scenes, as the closest fragment contributions become increasingly more important (Figures 7 and 8).

Our experimental evaluation demonstrates that our method outperforms the competing methods in a plethora of testing scenarios with a variety of scenes with different opacities and depth complexities, in terms of MSE as shown in Figures 6 and 7, including more complex scenes (Engine Coloured D $=$ 52, Hairball D $=$ 120).

Additionally, Figures 9 and 10 provide qualitative and quantitative results that prove that our method provides better accuracy in terms of both the FLIP measure [ANAM*20] and the MSE metric, as well as substantiating that our method offers a better approximation of the colour of the remaining fragments compared to HT.

We have included a supplementary video demonstration showing the real-time behaviour of our DFAOIT, as well as two competing methods (WBOIT, MBOIT). The video illustrates that our method does not exhibit view-dependent artifacts during camera movement, and it demonstrates the efficiency of our approach in rendering high depth-complexity scenes with quality similar to the ground truth A-buffer result.

5.3. Limitations

Our approach involves a neural network with high adaptability to a wide range of colours and scene complexities. However, we have observed that in highly complex scenes, our method may struggle to generate accurate outcomes, resulting in incorrect or overly vibrant colours (Figure 11). This limitation is linked to specific input features of the neural network, particularly those related to the entire pixel space and the premultiplied colour accumulation. We propose that normalizing this value to [0,1], incorporating the opacity of the closest fragments as a new input, and conducting extensive training of the neural network on highly complex scenes will effectively resolve this issue.

In scenes where the nearest fragments have very low opacity (Figure 12), our method shows a significantly higher error, failing to produce plausible results. This discrepancy is due to the optimization of the neural network training set and input features for scenes where the nearest fragments have a significant impact on the final outcome. To address this limitation, we propose incorporating the average opacity of the first two fragments as an additional input feature. This will enable the network to distinguish and adapt to scenarios where the first two fragments are not significant. Additionally, enhancing the training dataset by including similar scenes and providing additional information about the opacity of the first two fragments to the neural network can further improve performance.

6. Conclusions

Our method employs machine learning and can be used to approximately render OIT in real time with high accuracy. The shader source code provided can be adapted for rendering transparent scenes in games, computer aided design systems, visualization platforms and, virtual reality applications.

Without loss of generality, the neural network can be trained with perceptual image measures such as FLIP [ANAM*20] or with other transparency techniques besides OIT. As a future research direction, we will investigate a neural network on a per-fragment basis that will convey fragment information in the sense of the blended OIT techniques. This approach may reduce the error at the expense of a small overhead in computation time and memory. Additionally, adding feature descriptors that capture more local information, similar to the ones in [MKKP18], could improve the results further in cases with high depth complexity.

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

Finally, a potentially practical approach in scenarios involving the rendering of specific scenes (e.g. Video Games) could be to overfit the neural network using the test scenes. While this may reduce the generalization ability of our method, it effectively encodes the reference quality results (A-Buffer) on the neural network weights, resulting in much better quality.

Acknowledgements

3D models, except the "Engine" scene, were downloaded from Morgan McGuire's Computer Graphics Archive [McG17]. Hair model downloaded from sketchfab. We acknowledge support of this work by the project "Dioni: Computing Infrastructure for Big-Data Processing and Analysis." (MIS No. 5047222) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund). The publication of the article in OA mode was financially supported by HEAL-Link.

Conflict of Interest Statement

All authors declare that they have no conflict of interest.

References

[ANAM*20] Andersson P., Nilsson J., Akenine-Möller T., Oskarsson M., Åström K., Fairchild M. D.: ꟻLIP: A difference evaluator for alternating images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 3*, *2* (August 2020), Article 15, pp. 1–23. https://doi.org/10.1145/3406183

[BCL*07] Bavoil L., Callahan S. P., Lefohn A., Comba J. L. D., Silva C. T.: Multi-fragment effects on the GPU using the k-buffer. In Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (2007), I3D '07, ACM, pp. 97–104. https://doi.org/10.1145/1230100.1230117

[BM08] Bavoil L., Myers K.: Order independent transparency with dual depth peeling, Technical Report, NVIDIA Corporation (2008). https://developer.download.nvidia.com/SDK/10/opengl/src/dual_depth_peeling/doc/DualDepthPeeling.pdf

[Car84] Carpenter L.: The a-buffer, an antialiased hidden surface method. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, (New York, January 1984), vol. *18*, pp. 103–108. https://doi.org/10.1145/964965.808585

[DNSD22] Datta S., Nowrouzezahrai D., Schied C., Dong Z.: Neural shadow mapping. In *ACM SIGGRAPH 2022 Conference Proceedings*, (New York, 2022), pp. 1-9. https://doi.org/10.1145/3528233.3530700

[Eve01] Everitt C.: *Interactive Order-Independent Transparency*. Technical Report, NVIDIA Corporation (2001). https://developer.download.nvidia.com/assets/gamedev/docs/OrderIndependentTransparency.pdf

[FEE21] Friederichs F., Eisemann M., Eisemann E.: Layered weighted blended order-independent transparency. In Proceedings of Graphics Interface 2021 (2021), GI 2021, Canadian Information Processing Society, pp. 196–202. https://doi.org/10.20380/GI2021.22

[HCNS18] Han S., Chen G., Nehab D., Sander P. V.: In-depth buffers. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, *1* (July 2018). https://doi.org/10.1145/3203194

[KB14] Kingma D. P., Ba J.: Adam: A method for stochastic optimization, 2014. https://doi.org/10.48550/ARXIV.1412.6980

[LHLW09] Liu F., Huang M.-C., Liu X.-H., Wu E.-H.: Efficient depth peeling via Bucket Sort. In Proceedings of the Conference on High Performance Graphics 2009 (2009), ACM, pp. 51–57. https://doi.org/10.1145/1572769.1572779

[LHLW10] Liu F., Huang M.-C., Liu X.-H., Wu E.-H.: Freepipe: A programmable parallel rendering architecture for efficient multi-fragment effects. In Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (2010), ACM, pp. 75–82. https://doi.org/10.1145/1730804.1730817

[LHW21] Leonard L., Höhlein K., Westermann R.: Learning multiple-scattering solutions for sphere-tracing of volumetric subsurface effects. *Computer Graphics Forum 40*, 2 (2021), 165–178. https://doi.org/10.1111/cgf.142623

[MB13] McGuire M., Bavoil L.: Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques (JCGT) 2*, *2* (December 2013), 122–141. http://jcgt.org/published/0002/02/09/

[McG17] McGuire M.: Computer graphics archive. https://casual-effects.com/data (July 2017).

[MCTB11] Maule M., Comba J. L., Torchelsen R. P., Bastos R.: A survey of raster-based transparency techniques. *Computers & Graphics 35*, *6* (2011), 1023–1034. https://doi.org/10.1016/j.cag.2011.07.006

[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), ACM, pp. 103–118. https://doi.org/10.1145/2448196.2448212

[Mes07] Meshkin H.: Sort-independent alpha blending. *GDC Talk 2*, *4* (2007). https://www.gdcvault.com/play/535/Sort-Independent-Alpha

[MET*18] Martin T., Engel W., Thibieroz N., Yang J., Lacroix J.: Tressfx: Advanced real-time hair rendering. In *GPU Pro 360 Guide to Lighting*. AK Peters/CRC Press (2018), pp. 281–298.

[MJ09] Moore J., Jefferies D.: Covering new ground: Foliage rendering in pure. In *SIGGRAPH 2009 Advances in Real-Time Rendering in 3D Graphics and Games Course*. 2009.

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

[MKKP18] Münstermann C., Krumpen S., Klein R., Peters C.: Moment-based order-independent transparency. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, *1* (July 2018). https://doi.org/10.1145/3203206

[MMNL16] Mara M., McGuire M., Nowrouzezahrai D., Luebke D.: Deep G-buffers for Stable Global Illumination Approximation. In *Proceedings of High Performance Graphics* (2016), HPG '16, Eurographics Association, pp. 87–98. https://doi.org/10.2312/hpg.20161195

[NAM*17] Nalbach O., Arabadzhiyska E., Mehta D., Seidel H.-P., Ritschel T.: Deep shading: Convolutional neural networks for screen space shading. *Computer Graphics Forum 36*, *4* (2017), 65–78. https://doi.org/10.1111/cgf.13225

[PD84] Porter T., Duff T.: Compositing digital images. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (1984), SIGGRAPH '84, ACM, pp. 253–259. https://doi.org/10.1145/800031.808606

[Sal13] Salvi M.: Advances in real-time rendering in games: Pixel synchronization: Solving old graphics problems with new data structures. In *ACM SIGGRAPH 2013 Courses* (2013), SIGGRAPH '13, ACM.

[SBF15] Schollmeyer A., Babanin A., Froehlich B.: Order-independent transparency for programmable deferred shading pipelines. *Computer Graphics Forum 34*, *7* (2015), 67–76. https://doi.org/10.1111/cgf.12746

[Sha18] Sharpe B.: Moment transparency. In *Proceedings of the Conference on High-Performance Graphics* (2018), ACM. https://doi.org/10.1145/3231578.3231585

[SML11] Salvi M., Montgomery J., Lefohn A.: Adaptive Transparency. In *Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics* (2011), ACM. https://doi.org/10.1145/2018323.2018342

[SSNK21] Sikachev P., Slega S., Nowakowski K., Kowalczyk K.: Transparency rendering in cyberpunk 2077. In *ACM SIGGRAPH 2021 Talks* (2021), pp. 1–2.

[STH*19] Sitzmann V., Thies J., Heide F., Nießner M., Wetzstein G., Zollhöfer M.: Deepvoxels: Learning persistent 3d feature embeddings. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019), IEEE Computer Society, pp. 2432–2441. https://doi.org/10.1109/CVPR.2019.00254

[SV14] Salvi M., Vaidyanathan K.: Multi-layer alpha blending. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2014), I3D '14, ACM, pp. 151–158. https://doi.org/10.1145/2556700.2556705

[TF17] Thomas M. M., Forbes A. G.: Deep illumination: Approximating dynamic global illumination with generative adversarial network. CoRR (2017). https://doi.org/10.48550/arXiv.1710.09834

[TFT*20] Tewari A., Fried O., Thies J., Sitzmann V., Lombardi S., Sunkavalli K., Martin-Brualla R., Simon T., Saragih J., Nießner M., Pandey R., Fanello S., Wetzstein G., Zhu J.-Y., Theobalt C., Agrawala M., Shechtman E., Goldman D. B., Zollhöfer M.: Star on neural rendering. *Computer Graphics Forum 39*, *2* (2020), 701–727. https://doi.org/10.1111/cgf.14022

[TFV22] Tsopouridis G., Fudos I., Vasilakis A.-A.: Deep hybrid order-independent transparency. *The Visual Computer 38*, *9* (2022), 3289–3300. https://doi.org/10.1007/s00371-022-02562-7

[VF12] Vasilakis A., Fudos I.: S-buffer: Sparsity-aware multifragment rendering. In *Eurographics (Short Papers)* (2012), The Eurographics Association, pp. 101–104. https://doi.org/10.2312/conf/EG2012/short/101-104

[VF13] Vasilakis A. A., Fudos I.: Depth-fighting aware methods for multifragment rendering. *IEEE Transactions on Visualization and Computer Graphics 19*, *6* (June 2013), 967–977. https://doi.org/10.1109/TVCG.2012.300

[VF14] Vasilakis A. A., Fudos I.: k^+-buffer: Fragment synchronized k-buffer. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2014), I3D '14, ACM, pp. 143–150. https://doi.org/10.1145/2556700.2556702

[VKJ19] Vicini D., Koltun V., Jakob W.: A learned shape-adaptive subsurface scattering model. *Transactions on Graphics (Proceedings of SIGGRAPH) 38*, *4* (July 2019), 15. https://doi.org/10.1145/3306346.3322974

[VPF15] Vasilakis A. A., Papaioannou G., Fudos I.: k^+-buffer: An efficient, memory-friendly and dynamic k-buffer framework. *IEEE Transactions on Visualization and Computer Graphics 21*, *6* (June 2015), 688–700. https://doi.org/10.1109/TVCG.2015.2417581

[VVP20] Vasilakis A. A., Vardis K., Papaioannou G.: A survey of multifragment rendering. *Computer Graphics Forum 39*, *2* (2020), 623–642. https://doi.org/10.1111/cgf.14019

[VVPM17] Vasilakis A. A., Vardis K., Papaioannou G., Moustakas K.: Variable k-buffer using importance maps. In *Proceedings of the European Association for Computer Graphics: Short Papers* (2017), EG '17, Eurographics Association, pp. 21–24. https://doi.org/10.2312/egsh.20171005

[Wym16] Wyman C.: Exploring and expanding the continuum of oit algorithms. In *Proceedings of High Performance Graphics* (2016), HPG '16, Eurographics Association, pp. 1–11. https://doi.org/10.2312/hpg.20161187

[YHGT10] Yang J. C., Hensley J., Grün H., Thibieroz N.: Real-time concurrent linked list construction on the gpu.

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

Computer Graphics Forum 29, *4* (2010), 1297–1304. https://doi.org/10.1111/j.1467-8659.2010.01725.x

[ZXL*20] Zhang D., Xian C., Luo G., Xiong Y., Han C.: Deepao: Efficient screen space ambient occlusion generation via deep network. *IEEE Access 8* (2020), 64434–64441. https://doi.org/10.1109/ACCESS.2020.2984771

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supporting Information

© 2024 The Authors. *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.