DOI: 10.1111/cgf.14409
Pacific Graphics 2021
E. Eisemann, K. Singh, and F.-L Zhang
(Guest Editors)

Volume 40 (2021), Number 7

Z-Thickness Blending: Effective Fragment Merging for Multi-Fragment Rendering

Dongjoon Kim[1] and Heewon Kye*[2]

[1]School of Information Convergence, College of Software and Convergence, Kwangwoon University, S.Korea
[2]Division of Computer Engineering, Hansung University, S.Korea.

Abstract

An effective fragment merging technique is presented in this study that addresses multi-fragment problems, including fragment overflow and z-fighting, and provides visual effects that are beneficial for various screen-space rendering algorithms. The proposed method merges locally adjacent fragments along the viewing direction to resolve the aforementioned problems based on cost-effective multi-layer representation and coplanar blending. We introduce a z-thickness model based on the radiosity spreading from the viewing z-direction. Moreover, we present the fragment-merging schemes and rules for determining the visibility of the merged fragments based on the proposed z-thickness model. The proposed method is targeted at multi-fragment rendering that handles individual fragments (e.g., k-buffer) instead of representing the fragments as an approximated transmittance function. In addition, our method provides a smooth visibility transition across overlapping fragments, resulting in visual advantages in various visualization applications. In this paper, we demonstrate the advantages of the proposed method through several screen-space rendering applications.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Rendering;***

1. Introduction

The multi-fragment rendering technique [VVP20], which encompasses various algorithms to process a set of fragments of a single image, operates on multiple fragments at the same pixel location. Specifically, this technique performs two rendering processes in a common graphics processing units (GPU) pipeline (i.e., raster-based graphics), including the storage of three-dimensional (3D) geometric information into framebuffers and the generation of final images by exploiting the stored geometric information (i.e., screen-space information). The memory access capabilities of modern graphics hardware allow multi-fragment rendering to utilize screen-space geometric information, which helps increase the demand for screen-space or deferred rendering algorithms [KG13, MY18] in real-time such as order-independent-transparency (OIT) [VPF15, MKKP18], dynamic photorealistic rendering techniques [BKKB13, MMNL14, FHSS18], and interactive scientific visualizations (e.g., multivariate surface data representations [RASS16] or hybrid data representations for the fusion of multiple volumetric and potentially transparent surface data [LFS*15]).

Several studies developed various approaches for storing the bounded number of fragments using the k-buffer algorithm [BCL*07], which is an efficient GPU-based fragment level sorting algorithm and is widely-accepted A-buffer approximations [Car84]. This algorithm is capable of capturing k number of fragments using a fixed-size GPU buffer and ensuring the correct occlusion cues within the depth range of k-front fragments. However, it suffers from fragment overflow, which occurs in complex scenes, resulting in incorrect occlusion and flickering artifacts.

Various approximation approaches have been proposed in previous studies to address the aforementioned problem with fragment overflow. A representative approach encodes multiple fragments into the limited buffer memory based on the transmittance function by heuristic [SML11, MM17] or theoretical method [MKKP18]. This approach is reported to be effective in OIT applications [Eve01] by providing plausible rendering images [MB13, MKKP18], and it can inherently resolve coplanar fragments that result in z-fighting and unproductive storage required for redundant depth layers. However, the representations of correct geometry intersections, which are important factors in certain visualization tools, are limited when the correct visibility and depth of each surface layer are not properly stored. In addition, the problems of coplanarity and incorrect geometry intersections are highlighted in Figure 1.

In this paper, we propose a novel fragment merging method that introduces a new surface concept and the optical model-based vis-

* corresponding author

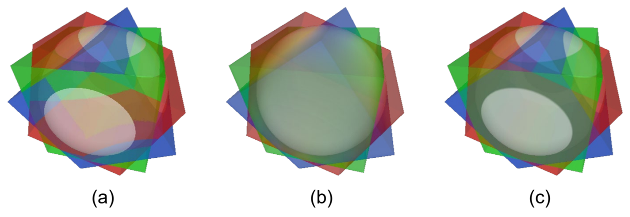


Figure 1: *A white sphere and three cubes (red, green, and blue) with intersections and coplanar surfaces. Transparency rendering results using (a) individually stored fragments without handling coplanarity, (b) parameterized transmittance function, and (c) individually stored fragments with coplanarity handling.*

ibility decision operations. The proposed method can effectively mitigate various problems such as fragment overflow and z-fighting based on the cost-effective multi-layer representation and coplanar blending. An optical model is applied to the fragments using the *z-thickness model* where the virtual z-thickness is assigned along the z-direction. This optical model can theoretically represent the visibility of an arbitrary interval inside a homogeneous medium (i.e., a fragment using the z-thickness model), and it has lesser computational cost than the other methods. The advantages of the z-thickness model are as follows. First, it provides a cost-effective multi-layer approximation that locally merges adjacent overlapping fragments along the z-direction and maintains separated fragments far away, capturing abundant screen-space geometry information in a limited buffer capacity. Second, it provides smooth visibility transitions, which provide visual advantages in various visualization applications, by considering the degree of the partial overlap between fragments. The primary contributions of this study can be summarized as follows.

- Z-fighting handling. Merging the adjacent fragments, whose z-thickness values are determined based on the z-depth resolution (Section 4.1.3), allows for a natural blending of coplanar layers.
- A cost-effective multi-layer representation with respect to memory. Local merging of adjacent fragments along the z-direction while maintaining separated fragments far from each other captures the abundant screen-space geometry information in a limited buffer capacity (Sections 3 and 5.1).
- An efficient integration of the proposed method using two competitive raster-based implementations, namely the bounded and unbounded memory usage approaches (Section 4).
- Visual effects applicable to various visualization applications such as point set surfel rendering, comparative visualization, ghosted illustration, and volume/polygon hybrid rendering (Sections 5.2 and 5.3).
- Scalability to multi-layered screen-space rendering algorithms. Leveraging the abundant screen-space geometry information enables more precise rendering in the screen-space global illumination effects on a transparent complex scene (Section 5.4).

In a recent survey on multi-fragment rendering [VVP20], the authors emphasized the need to develop encoding and decoding techniques to represent more fragment information with limited buffer capacity. They also discussed further directions for various visualization effects based on multi-fragment rendering. In this paper, we demonstrate that the proposed method can be leveraged in such directions by introducing a novel z-thickness model and a set of concepts that can be applied to a variety of screen-space-based rendering algorithms.

2. Related Work

Several approaches have been reported to efficiently achieve multi-fragment rendering [VVP20], where handling a large number of fragments is the primary challenge. In GPU raster-based graphics, such fragments concurrently enter the same pixel, resulting in heavy contention and depth ordering problems [Cra10, MCTB11].

Several GPU-accelerated buffers have been introduced to address this problem. A buffer [Car84] is the first method to capture all fragments by constructing linked lists per pixel in a single rasterization pass. Specifically, the fragments in per-pixel linked lists in the A-buffer algorithm are post-sorted based on their depth values. However, the requirement for unbounded memory when using these types of algorithms limits their practical applications. Yang et al. [YHGT10] introduced a practical GPU-accelerated A-buffer implementation using atomic memory operations built in the GPU. To alleviate their read-modify-write hazards, Maule et al. [MCTB12] presented a memory-efficient buffer scheme based on GPU-accelerated A-buffer techniques. This is the preeminent method for multi-fragment rendering, despite the unbounded memory required for valid results.

Bavoil et al. [BCL*07] suggested a k-buffer algorithm to alleviate the burden of excessive allocation and access to GPU memory. Their algorithm is designed to capture the k foremost fragments by employing a fixed-size buffer memory per pixel on the GPU and minimizing read-modify-write hazards due to incoming fragments concurrently mapping onto the same pixel (i.e., data contention or races). Several studies have developed many k-buffer variants [MCTB11] by performing multiple passes [LWXW09] or exploiting the GPU-accelerated pixel synchronization operations [Sal13]. Vasilakis et al. [VPF15] developed an efficient and memory-friendly k-buffer algorithm called k^+*-buffer*, which uses a dynamic k value decision scheme and GPU-optimized implementation techniques.

As mentioned previously, fragment overflow is the most challenging problem in multi-fragment rendering based on bounded memory approaches such as k-buffer. Discarding the overflowing fragments is simple and widely accepted in many k-buffer methods but causes geometric information loss, resulting in unpleasant images when rendering complex scenes. The OIT is a typical multi-fragment rendering application in which fragment overflow affects the image quality degradation. An alternative OIT approach is the blended OIT algorithm that performs blending over fragments without sorting them [Mes07, BM08]. Stochastic transparency [ESSL10] presents a transparency effect by a stochastic representation of transmittance in a blended OIT manner. McGuire and Bavoil [MB13] developed a weighted blending operator in which the visibility contribution was determined based on the opacity and depth of the fragment to improve their incorrect alpha coverage and color. Their blending model is a transmittance function simply parameterized by the opacity and depth of per-pixel fragments.

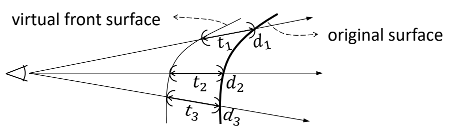


Figure 2: *Illustration of the z-thickness model in which $d_{1,2,3}$ the depth value along each ray and $t_{1,2,3}$ indicates the z-thickness value at each depth.*

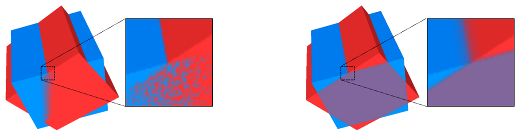


Figure 3: *Z-fighting handling. Two superimposed cubes (red and blue) are rendered. The left image shows the z-fighting problem. The right image shows the consistent and continuous blending.*

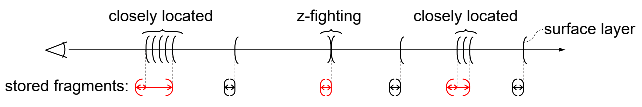


Figure 4: *Illustration of how the fragments coming from coplanar or closely located surface layers are stored into the buffer memory. The interval of the brackets indicates the z-thickness. The red bracket indicates the merged fragment.*

However, there are limitations in the determination of the correct visibility of intersecting surface layers due to the lack of consideration for the discontinuous change in the transmittance function at the depth of the layer. The adaptive transparency algorithm [SML11] approximates the incoming fragments as the transmittance function along the view direction using a fixed number of terms. However, the heuristic removal of transmittance terms results in inconsistent transparency across pixels. Münstermann et al. [MKKP18] introduced a theoretical approximation of the original transmittance function based on the moment. Recently, Friederichs et al. [FEE20] employed a layered concept that locally blends geometries at even intervals along the viewing direction to improve the weighted blending operator [MB13]. They demonstrated that the local blending approach yielded better OIT results. However, these approaches do not provide correct intersections in visually distinguishable geometries that generally occur in the first few semi-transparent layers.

The strategy that preserves the foremost fragments and approximates the farthest fragments can be a simple but powerful alternative to this problem. Marilena et al. [MCTB13] introduced the core-tail concept, preserving the foremost fragments in a sorted order while accumulating the farther fragments into the tail fragment. Moreover, Salvi and Vaidyanathan [SV14] utilized a core-tail scheme with opacity encoding. However, the lack of consideration of overlapping fragments results in poor images.

On the other hand, coplanarity of the fragments is also a typical problem in multi-fragment rendering. Binary z-test-based visibility decisions with floating-point round-off errors cause a z-fighting problem for coplanar layers. Transmittance-function-based approaches [WGER05, MB13, MKKP18] inherently address this problem. However, they lack a precise depth representation. In buffer-based approaches that store individual fragments, special care for coplanar primitives [VF12] can suppress speckling artifacts but require additional capacities and operations for extracting coplanar fragments separately. Kim et al. [KKL*16] proposed an intermixing technique to handle this problem in the context of image-based blending, but it required an impractical number of rendering passes to display multiple surface layers. Duff [Duf17] presented deep compositing operators that handle the overlap of volumetric elements such as clouds and smoke, which is similar to our blending operators.

3. Method

It is important to represent and restore abundant geometric information with a limited number of fragments in a multi-fragment rendering. For example, many screen-space rendering algorithms [BCL*07, BKKB13, MMNL14, FHSS18] use a limited number of fragments to calculate the geometric interactions for dynamic rendering effects in real-time.

Therefore, we introduce the *z-thickness model* and present the fragment merging schemes based on this model. The proposed method facilitates local merging of adjacent overlapping fragments along the z-direction while maintaining separated fragments far away from each other, capturing abundant screen-space geometry information with a limited number of fragments. Moreover, the proposed method considers the degree of partial overlap between fragments, resulting in visual effects that can be beneficial for a variety of visualization applications.

Note that a fragment has an opacity (i.e., alpha) and color information. In this paper, we use the color information as the alpha-multiplied color, and the term *visibility* represents the opacity (i.e., alpha) and alpha-multiplied color.

3.1. *Z*-Thickness Surface Model

The z-thickness model is designed by assigning a z-directional thickness to each fragment (refer to Figure 2). The application of the z-thickness to a surface simulates the surface as a translucent band whose boundary is placed before the z-thickness length from the surface. The z-thickness model leads to the virtual overlap between fragments whose depth locations are close to each other. The virtual overlap provides the following advantages. First, it handles the z-fighting problem by displaying continuous blending visibility instead of unpleasant speckling and noisy dotted visibility (refer to Figure 3). Second, it stores surface layers far enough from each other instead of storing all overlapped surface layers close to each other (refer to Figure 4). This allows screen-space algorithms that use the bounded memory capacity to perform effectively in terms of memory efficiency and geometric representation.

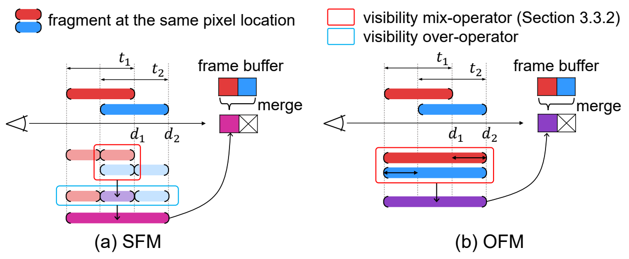
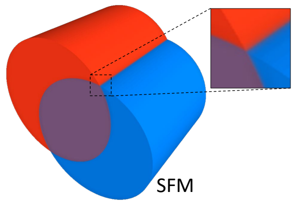


Figure 5: *Fragment merging based on the z-thickness model. (a) smooth fragment merging (SFM) and (b) order-independent fragment merging (OFM). d_n and t_n (n = 1, 2) indicate depth and thickness values of each incoming fragment, respectively.*



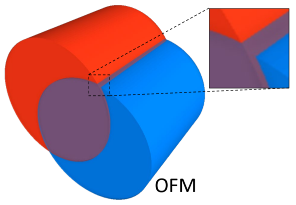


Figure 6: *Visual impact of the SFM and OFM. The z-thickness value is set to 0.02 times the cylinder height.*

3.2. Fragment Merging

We introduce two schemes for merging fragments based on the proposed z-thickness model: *smooth fragment merging* (SFM) and *order-independent fragment merging* (OFM). Figure 5 shows these fragment-merging schemes. The SFM divides overlapping fragments by the z-directional boundaries of each fragment and then composites the visibilities of the subdivided fragments. The OFM merges the partially overlapping fragments into one fragment whose visibility is determined without considering the fragment subdivision, thereby allowing an order-independent visibility decision.

It would be better to use the SFM instead of OFM when the order of fragments is sorted in depth order (e.g., fragments from the ray tracing or resolve pass of multi-fragment algorithm) because the SFM provides more elaborate visibility than the OFM. However, in the raster-based graphics pipeline, the fragment merging performed in the k-buffer algorithm, which uses bounded memory capacity, should use the OFM because the order of fragments coming into the shader is not deterministic. A detailed discussion is provided in Section 4.

Fragment merging can be thought of as an approximation model that merges N adjacent fragments along a ray into one fragment. The merged fragment is defined by (i) the maximum depth value of the adjacent fragments and (ii) the z-thickness value that covers the adjacent fragments.

3.2.1. Smooth fragment merging (SFM)

The SFM provides smooth visibility transitions by considering the degree of partial overlap between fragments. This is accomplished by subdividing and compositing the fragments and their visibilities (refer to Figure 5(a)). The visibility decision by subdividing and compositing the fragments is described in Section 3.3. A primary assumption to provide consistent visibility in the SFM is that the depth order of the fragments coming onto the same pixel location must be sorted. Otherwise, the SFM will yield inconsistent visibility across pixels and during an animation because the visibility is calculated by the *over operator* [PD84] (i.e., front-to-back blending operator).

Up to three subdivided fragments are generated when two fragments partially overlap each other. In the SFM, we merge the subdivided fragments into one fragment whose visibility is determined by applying the over-operator, providing smooth visibility transitions.

3.2.2. Order-independent fragment merging (OFM)

In certain applications where only limited memory resources are allowed, bounded memory must be used throughout the entire rendering pipeline. For example, general k-buffer algorithms [BCL*07, MCTB11, VPF15] use the bounded memory that stores k fragments. The k-buffer algorithm can efficiently select the k-foremost fragments among the unsorted fragments that concurrently enter the same pixel using the GPU built-in atomic operators during the store pass. Fragment merging can also be easily applied to the store pass based on the k-buffer algorithm, and it gives the k fragments rich per-pixel geometric information. Therefore, we suggest the OFM depicted in Figure 5(b) to determine the visibility of the merged fragment regardless of both the order of the incoming fragments and their precedence in depth. The order-independent visibility decision of the merged fragment is accomplished using the *mix-operator* described in Section 3.3.2.

3.2.3. Visual impact of SFM and OFM

The SFM considers the degree of partial overlap between fragments and provides smooth visibility transitions. On the other hand, the OFM provides monotonous visibility transitions by considering only whether the fragments overlap but ensures visibility decisions regardless of both the order of incoming fragments and their precedence in depth. Figure 6 highlights the visual impact of overlapping surface layers with respect to OFM and SFM. The SFM provides a continuous color change with respect to the degree of overlap around pixels that represent surface intersection, resulting in an anti-aliasing effect.

3.3. Visibility Decision

In this section, we present the optical operators that divide and composite the visibility of overlapping fragments based on the proposed z-thickness model.

3.3.1. Visibility subdivision

We assume that each fragment based on our z-thickness model represents a homogeneous medium, enabling the fragment visibility to be subdivided with respect to the length inside the medium along the viewing z-direction. Note that this operator is only available for the SFM.

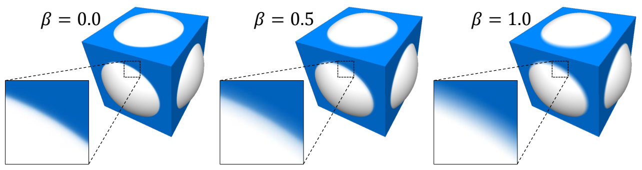


Figure 7: *Visual impact of the SFM with respect to* β *of* (4)*. A bit large z-thickness value (0.05 times the sphere radius) is used to highlight the visual impact.*

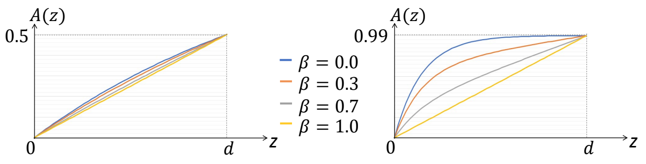


Figure 8: *Plots of opacities of* (4) *according to different* β *values and fragment opacities (0.5 and 0.99).*

Fragments can be divided into several subdivided fragments. The merged visibility of these subdivided fragments must be identical to that of the original fragment. The extinction coefficient τ is constant inside a fragment that is filled with the homogeneous medium. Therefore, the opacity accumulated up to depth z inside the fragment can be defined as

$$A(z) = 1 - e^{-\tau z} \tag{1}$$

If the accumulated opacity of the medium with thickness d is given as $A_d = A(d)$, (1) can be rewritten as

$$A(z) = 1 - (1 - A_d)^{z/d} \qquad (0 \le z \le d) \tag{2}$$

Based on the simple emission-absorption optical model [Max95], the accumulated color $C(z)$ at depth z inside the medium with visibility $(C(d), A(d))$ is calculated as

$$C(z) = C_d \frac{A(z)}{A_d} \tag{3}$$

(3) is used to determine the visibility of the front subdivided fragment. It can also determine the visibility of the back-subdivided fragment using the thickness of the subdivided medium instead of depth z.

The linear approximation of (2), $A(z) \approx \frac{z}{d} A_d$, provides smoother visibility transitions at the intersection of the geometries (refer to Figure 7). Moreover, β is used as a control parameter to adjust the visual effect between the smooth visibility transition and strictly optical visibility:

$$A(z) = \beta \frac{z}{d} A_d + (1-\beta)(1 - (1 - A_d)^{z/d}) \qquad (0 \le \beta \le 1) \tag{4}$$

Figure 8 shows a plot of (4) with respect to the depth inside the fragment. The effect of β is insignificant if the opacity is low, while β can be used as a control to make the inner structure visible if the opacity is high (i.e., close to 1). Figure 9 shows the ghosted effect, which makes the interior objects appear with low contrast based on β, providing a depth cue.

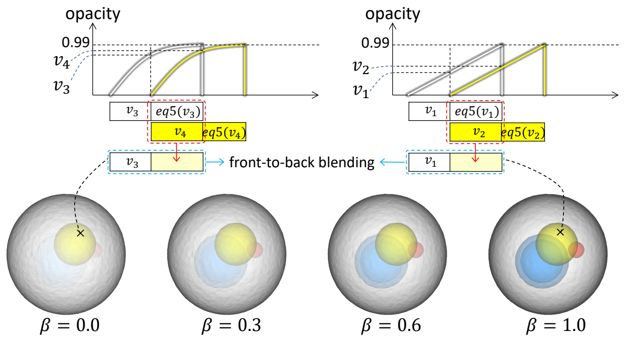


Figure 9: *Illustration of the ghosted effect using the visual impact of the SFM with respect to* β*. Three colored spheres inside a large white sphere are rendered. A sufficiently large z-thickness value (the radius of the white sphere) is used to reveal the interior spheres. All surfaces have an opacity of 0.99.*

The visibility of the back subdivided fragment (C_b, A_b) must be determined to ensure that the accumulated visibility across the subdivided fragments is identical to the visibility of the original fragment since the visibility of the subdivided fragments using the approximation in (4) does not restore the visibility of the original fragment. This is achieved by substituting (3) and (4) into the over operator (i.e., front-to-back blending operator) of two sequential visibilities given as

$$(C_b, A_b) = \frac{(C_d, A_d) - (C(z), A(z))}{(1 - A(z))} \tag{5}$$

3.3.2. Mix operator

Given completely overlapping fragments, individual fragments that overlap each other contribute equally to the final visibility of the merged fragment, regardless of the order in which the visibility is received for compositing. Therefore, we introduc a *mix-operator* that accumulates the opacity and color separately.

First, the accumulating opacity A_{acc} is calculated based on the accumulated effect of incoming opacity A_i:

$$A_{acc} = 1 - \prod_i (1 - A_i) \tag{6}$$

The accumulated color c_{acc} is calculated by normalizing the sum of the participating color c_i (not opacity-weighted color) with respect to incoming opacity A_i, which can be represented by the visibility color C_i (opacity-weighted color). The accumulation of the visibility color C_{acc} is then obtained as

$$c_{acc} = \frac{\sum_i c_i A_i}{\sum_i A_i} = \frac{\sum_i C_i}{\sum_i A_i}, \qquad C_{acc} = c_{acc} A_{acc} \tag{7}$$

The order of the incoming opacity does not affect accumulation opacity. This enables order-independent visibility compositing if the opacity summation $A_{sum} = \sum_i A_i$ is given. The formula can be expressed as a recurrent extension of (6) and (7):

$$\begin{aligned} \widetilde{A}_{acc} &= 1 - (1 - A_{acc})(1 - A_{new}), \\ \widetilde{C}_{acc} &= \frac{\frac{C_{acc}}{A_{acc}} A_{sum} + C_{new}}{A_{sum} + A_{new}} \widetilde{A}_{acc} \end{aligned} \tag{8}$$

where $(\widetilde{C}_{acc}, \widetilde{A}_{acc})$ and (C_{new}, A_{new}) are the new accumulating visibility and the new incoming visibility, respectively.

In the proposed method, this operator is applied to the red rectangles in Figure 5 in which A_{sum} is assigned as an element of a fragment.

4. Implementation

The proposed method can be easily applied to any GPU-based rendering pipeline, such as ray-tracing and raster-based graphics, providing that they handle individual fragments. We apply the proposed method to multi-fragment rendering based on raster-based graphics, which is widely supported by general GPUs and provides efficient performance. Generally, multi-fragment rendering based on the raster graphics pipeline requires two sequential rendering passes, namely the *store pass* and *resolve pass*. The store pass captures fragments from the same pixel in a streaming, while the resolve pass sorts the fragments stored in the framebuffer and then performs additional screen-space rendering operations. In this section, we introduce GPU-based implementations for efficient integration of the proposed method using two competitive raster-based approaches, namely a limited memory usage approach (Section 4.2) and an unlimited memory usage approach (Section 4.3).

4.1. Common operations

First, we introduce the common techniques used in both implementation approaches before discussing the detailed implementation.

4.1.1. Fragment merging

The OFM can be directly implemented because of its order-independent property and the simple approximation that does not consider fragment subdivision. In addition, the SFM is easy to implement since it sequentially merges two overlapping fragments sorted in depth order. Specifically, we output N fragments into M fragments ($M \leq N$) by merging the overlapping fragments when using the resolve pass. This has the linear computational complexity O(N) was used in our implementation, as shown in Algorithm 1.

The following rules should be considered when integrating the proposed fragment merging with the multi-fragment rendering based on the raster graphics. First, the OFM should be used when performing the fragment merging in the store pass because the order of fragments coming into the shader is not deterministic. On the other hand, there is no need to consider fragment merging in the resolve pass because the adjacent fragments are already merged once the OFM is performed in the store pass. Second, it would be better to use SFM rather than OFM in the resolve pass because (i) it provides more elaborate visibility than the OFM and (ii) the resolve pass allows the fragments to be sorted in depth order. Algorithm 1 presents the integration of the SFM into the resolve pass.

Algorithm 1 Output fragments (resolve pass)

```
1:  $F_{in}[N] = f_1, f_2, ..., f_N$                ▷ $N$ fragments sorted in depth order
2:  $F_{out}[M]$                                   ▷ maximum $M$ fragments for output
3:  $f_i = F_{in}[0]$ , $c := 0$                      ▷ $c$ counts valid output fragments
4:  for $i$:=0; $i < N$; $i := i+1$ do
5:      $f_{next} := null$ , $f_{merge} := null$
6:      if $i+1 < N$ then
7:          $f_{next} := F_{in}[i+1]$
8:          if $f_i$ and $f_{next}$ overlap then
9:              $f_{merge}$ := SFM($f_i$, $f_{next}$)        ▷ see Section 3.2.1
10:     if $f_{merge}$ is $null$ then
11:         if $c < M-1$ then
12:             $F_{out}[c] := f_i$ , $c := c+1$
13:             $f_i := f_{next}$
14:         else if $f_{next}$ is not $null$ then          ▷ tail-handling case
15:             $f_i.v$ := OVER($f_i.v$, $f_{next}.v$) ▷ over operator,.v: rgba
16:             $f_i.t := f_{next}.z - f_i.z + f_i.t$         ▷ .t: z-thickness
17:             $f_i.z := f_{next}.z$               ▷ .z: depth (original surface)
18:             $f_i.a := f_i.a + f_{next}.a$          ▷ .a: $A_{sum}$ (refer to (8))
19:     else
20:         $F_i := f_{merge}$
21: if $f_i$ is not $null$ then
22:     $F_{out}[c] := f_i$ , $c := c+1$
23: (optional) over operator for $F_{out}[0,...,c-1]$'s visibilities (OIT)
24: (optional) store $F_{out}$ for the screen-space rendering pass
```

4.1.2. Tail-handling

The multi-fragment rendering, which uses a limited number of fragments, inherently suffers from fragment overflow in complex scenes although the proposed method can effectively mitigate the fragment overflow problem. We employ *tail handling* [MCTB13, Kub14], which merges the fragments further away than k-front fragments into one fragment (i.e., tail fragment), to provide plausible rendering images. We use different operators based on the store pass, where the order of fragments is not deterministic, and resolve pass, where the fragments are sorted in depth order, to determine the visibility of the tail fragment. The store pass uses the mix operator (refer to Section 3.3.2), while the resolve pass uses the over operator (refer to the 15th line in Algorithm 1).

4.1.3. Z-thickness value determination

We consider a z-value precision determined by a single-precision floating-point, which provides 24-bit precision, to address the z-fighting problem. In addition, we consider the floating-point z-buffer behavior, which is nonlinear, indicating that the precision of z is proportional to the reciprocal of the z value. In other words, there is a lot of precision close to the eye and considerably lesser precision in the distance. The near and far clip planes of the view frustum affect the z-precision at various ranges in the graphics pipeline. The smallest depth separation that can resolve at depth z of the view frustum range is called the *z-resolution* of the z buffer, which is calculated as [Bak]:

$$P(z) = \frac{b}{\frac{b}{z} - \frac{1}{2^n}} - z, \qquad (b = \frac{z_n z_f}{z_n - z_f}) \tag{9}$$

where z_n is the depth value of the near clip plane, z_f is the depth value of the far clip plans, and n is the number of precision bits (conventionally 24 in the single-precision floating-point). We set the z-thickness value at z to two times the z-resolution, $2 \times P(z)$, to provide sufficient z-resolution for z-fighting handling. Note that our implementation uses $2 \times P(z)$ in the normal multi-fragment rendering. Moreover, the z-thickness value can be manually adjusted by the user based on the visualization purpose and geometry complexity.

4.2. Bounded memory usage approach

The k-buffer algorithm is a conventional approach that uses a bounded memory capacity (in Algorithm 1, $M = N = k$, k is normally set to 4 or 8 [Kub14]) in the entire rendering pipeline. However, it suffers from data races during the store pass as it performs read-modify-write operations of fragments. Therefore, we use a pixel synchronization technique [Sal13] in which other fragments coming towards the same pixel must wait for the processing of an occupying fragment to end. In our implementation, we use atomic processing using a *rasterizer-ordered-view* supported in Direct3D 11.3 or higher. Furthermore, we use the k^+buffer techniques [VPF15], namely the *max array buffer* and the *fragment culling*, to optimize the implementation. The max array buffer works by assigning a slot to the fragment with the maximum depth value prior to the tail in a stream, and fragment culling conditionally performs memory read/write to efficiently replace the k-front fragments. We also assign a slot to the max array buffer to handle the tail fragment.

4.3. Unbounded memory usage approach

The store pass suffers from data races of the incoming fragments. Therefore, we employ the dynamic framebuffer approach [MCTB12], which uses only an *atomic count* instruction to avoid data races efficiently, because the k-buffer algorithm requires a heavy pixel synchronization task. In addition, this approach requires the count pass, where the atomic count instruction counts the number of fragments per pixel before the store pass and enables the store pass to store the incoming fragments into the framebuffer without the read-modify overhead. Although it requires an unbounded memory (we employ the preallocated chunk that can store a maximum of 1024 per-pixel fragments, that is, $N = 1024$ in Algorithm 1 and $M = 4$ or 8), it provides a smooth visibility transition across the overlapping geometries and results in correct OIT images with considerably faster performance than the k-buffer algorithm.

5. Experiments

The proposed fragment merging (SFM and OFM) can be applied to any rendering method that handles individual fragments, providing a bounded number of fragments to effectively perform screen-space rendering algorithms in terms of memory efficiency and geometric representation. In our experiments, we set the bounded number to 8 ($M = 8$ in Algorithm 1), satisfying both performance and quality [Kub14]. We considered the following techniques to demonstrate the validity of the proposed method:

- The dynamic fragment buffer (DFB) [MCTB12] is a preeminent GPU buffer-based method that can handle all fragments with the fastest rendering performance. However, this method requires unbounded memory. The SFM should be used when applying the proposed method to the DFB (DFB+SFM) based on the integration rules discussed on Section 4.1.1.

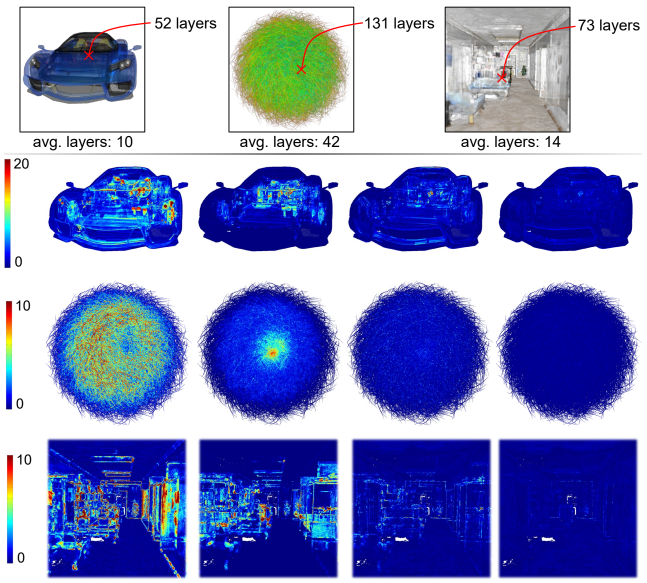


Figure 10: *Heatmap-coded differences between the image generated using the dynamic frame buffer (DFB) [MCTB12], which is used as a reference, against the images using the Moment-based transmittance approximation (MBT) [MKKP18], static k^+ buffer (SKB) [VPF15], SKB+OFM, and DFB+SFM (from left to right). The sportscar data contains 1075 mesh objects, calling 1075 draw functions. The hairball data is a single mesh object containing high frequency details. The floor data is a dense point set, which is displayed by large size surfels.*

- The k^+-buffer [VPF15] is an efficient k-buffer approach that uses both static and dynamic k values with an optimal GPU implementation. In our experiments, we employed the static version of the k^+-buffer (SKB). The OFM should be used when applying the proposed method to the SKB (SKB+OFM) because fragment merging is performed in the store pass based on the integration rules presented in Section 4.1.1.
- The moment-based transmittance approximation (MBT) [MKKP18] theoretically encodes all fragments along a ray into the transmittance function defined by a small number of variables. This enables high rendering speeds while providing plausible OIT results unless geometry intersections are considered. The proposed fragment merging method is not applied to this method because it does not handle individual fragments.

All experiments were implemented using shader programs, leveraging the GPU raster-based graphics pipeline. The experiments were conducted on a computer with an Intel Core i77700 3.6 GHz CPU with 32 GB of RAM, an NVIDIA GeForce GTX 1080 GPU, and a Windows 10 x64 operating system.

5.1. Order Independent Transparency

Various comparisons were conducted to prove the validity of the multi-layer representation based on the proposed method in terms of the OIT performance (i.e., improved image quality) of the approximation model. The DFB provides the reference image, while

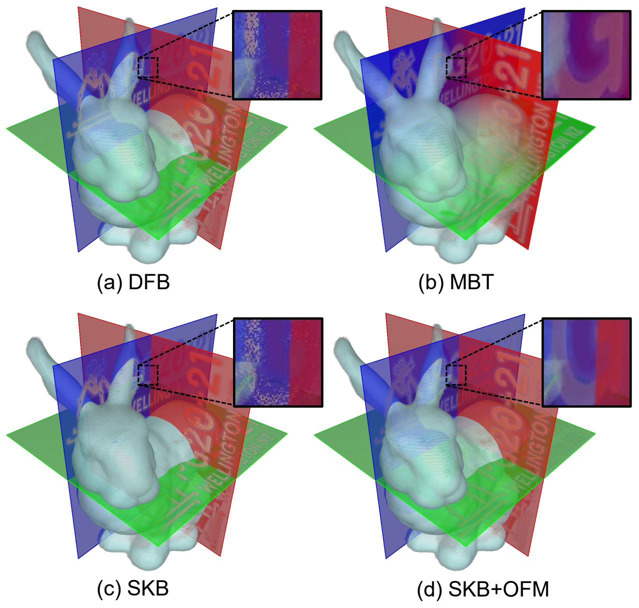


Figure 11: *Distinguishable occlusion test with transparency. A point set of bunny model is represented by large size surfels, resulting in a lot of overlapping layers. The intersecting planes are superimposed on the bunny model. The text primitives are coplanar with the intersecting planes.*

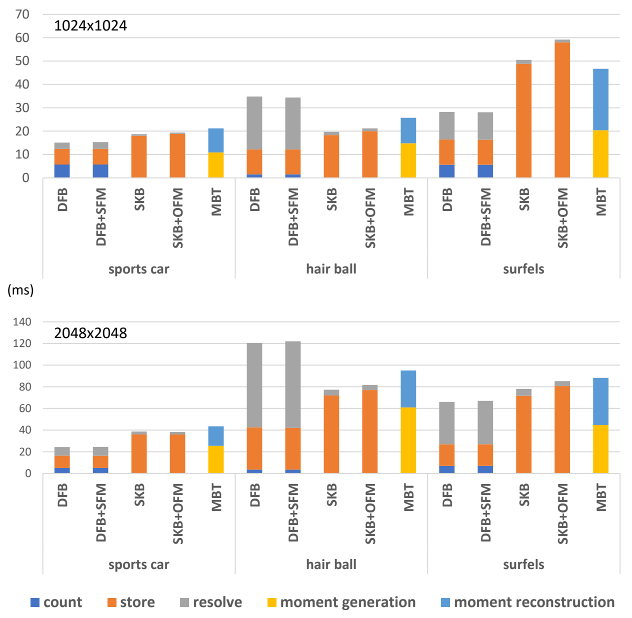


Figure 12: *Performance evaluation, in milliseconds, at image resolutions of* 1024×1024 *and* 2048×2048 *for all methods.*

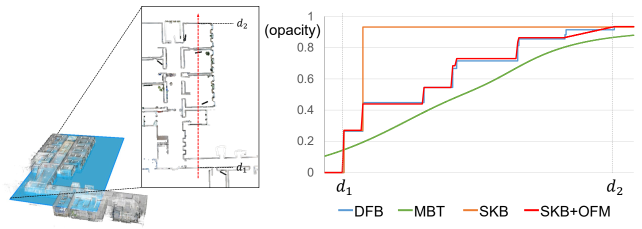


Figure 13: *Absorbance functions (right) along a ray (red dotted arrow) of the scene. The scene is from the floor data (full view) used in Figure 10. The viewing ray passes through 52 layers of overlapping surfels and 10 separated geometry layers in space when rendering is performed using the ray as a viewing direction.*

the SKB and MBT provide the approximated OIT images. Furthermore, we used a z-thickness value of $100 \times P(z)$ (refer to (9)) to demonstrate the effect of merging adjacent fragments along the z direction on the OIT. Fragment merging was applied differently in SKB and DFB. The OFM was applied to the store pass of SKB, and the SFM was applied to the resolve pass of DFB.

5.1.1. Visual analysis

Figure 10 shows the results of the comparisons rendered with high transparency. It is apparent from the results that the proposed method improves OIT results when only a limited memory is available (see the results obtained by SKB and SKB+OFM). Moreover, the proposed method provides OIT results with a negligible loss of quality when a limited number of fragments from all fragments (see the results obtained by DFB+SFM) was generated. This implies that local merging of adjacent fragments along the z-direction can well encode the entire fragments with a limited number of fragments.

Figure 11 shows that the proposed method is effective in rendering a complex scene that contains overlapping geometries. Surfels [PZVBG00] from a point set surface produce densely adjacent layers along the z-direction when using a large surfel size to cover the point set surface. Therefore, the SKB suffers from fragment overflow early in the depth layers, producing incorrect occlusion cues (refer to Figure 11(c)). An approximation approach to the transmittance function, MBT, does not capture correct depth values, resulting in indistinguishable intersections (refer to Figure 11(b)). We applied the OFM with the default z-thickness value, $2 \times P(z)$, to the SKB to show that the proposed method can mitigate such problems.

5.1.2. Performance analysis

We evaluated the computational overhead of OIT methods that are previously discussed by measuring rendering passes, such as the count pass, store pass, and resolve pass. Specifically, the count pass is required for the DFB to count the number of per-pixel fragments and generate an offset table that efficiently maps per-pixel fragments to the GPU memory. Sorting the stored fragments in depth order in the resolve pass of the DFB and SKB is performed using the insertion sort or shell sort. The insertion sort is used when the number of sorting fragments was less than 16; otherwise, the shell sort was chosen [VPF15].

Figure 12 shows the execution times of each rendering pass of comparison methods for the scenes in Figure 10. The computational

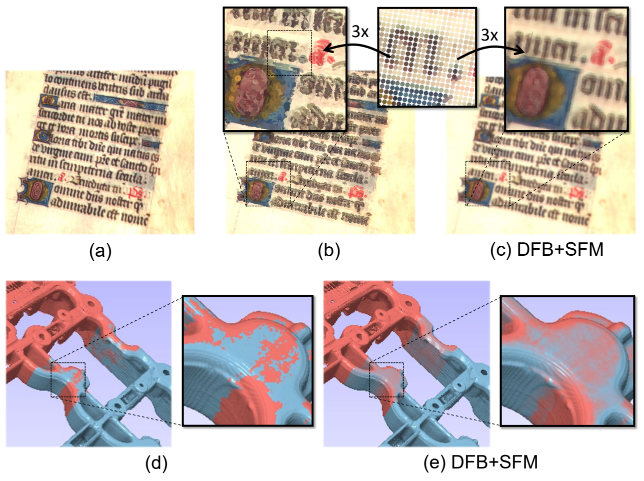


Figure 14: *Local depth blending effects of the SFM. The top row images show a manuscipt-textured mesh model. (a) A reference image rendered from the polygons of the mesh model. (b)(c) Images rendered from large surfels representing the point set of the mesh model. The middle image among the zoomed images of (b) and (c) highlights the point resolution (surfels scaled to 3× the size of the original resolution sufels). The bottom images show the aligned geometries containing partially overlapping surfaces. (a)(b)(d) Images rendered using foremost surface rendering, which uses a binary z-depth test.*

overhead of the SFM performed in the resolve pass of the DFB was negligible compared to the sorting overhead. The merged fragments can change the order and number of read-write-modify operations, affecting the performance, in the pixel synchronization when storing the k-front fragments from the incoming and read-back fragments. However, the performance loss was insignificant compared to the visual benefit of the proposed method.

We showed that the local merge of adjacent fragments based on the proposed method could properly represent the multi-layered geometry information by comparing the absorbance functions (i.e., $1-$transmittance) obtained from the comparison methods. Figure 13 shows the absorbance functions along the z-direction at a pixel of an image where densely adjacent surfels are rendered as overlapping layers. We used the absorbance function obtained from the combination of SKB+OFM, which handles the fragments that concurrently and randomly come onto the shader, to evaluate the robustness of the local merging based on the proposed method. It enables the representation of physically separated geometries apparent in the absorbance function of DFB with a limited number of fragments.

5.2. Varying Z-Thickness Effects

We demonstrated that the proposed method (SFM) offers visual advantages in special-purpose non-photorealistic rendering by varying the z-thickness values. Furthermore, β was set to 1 because a smooth visibility transition is an important factor.

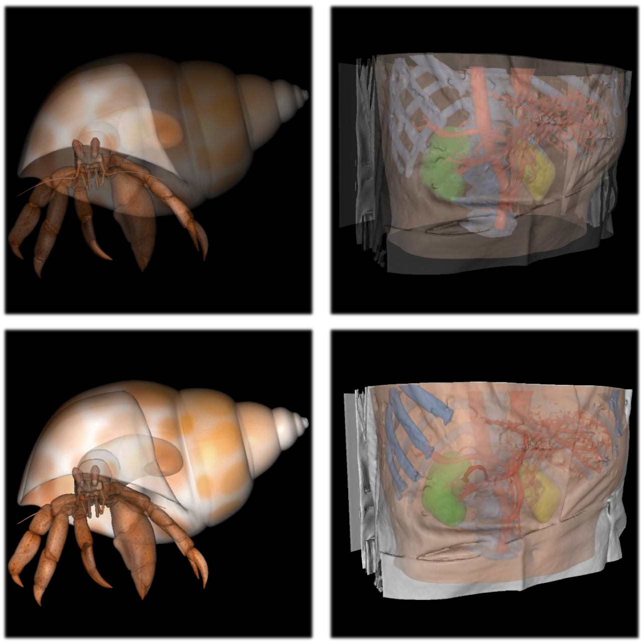


Figure 15: *Ghosted illustration. The upper images show transparency rendering (alpha = 0.3) using DFB. The bottom images show the ghosted effect obtained by our method (DFB+SFM).*

5.2.1. Local depth blending

SFM allows the blending of images based on the depth order and degree of overlap of the fragments along the viewing direction. The common method of rendering a point-set surface without surface reconstruction is the point set rendering using surfels. A straightforward way to minimize holes is to use large surfels, which results in speckling artifacts due to z-fighting occurring at densely overlapping points. The SFM effectively resolved this problem. The top images in Figure 14 highlight this. We chose a z-thickness value of $10 \times P(z)$.

Comparative visualization provides a distance cue of the local geometry for users to recognize the differences in shapes and sizes [BBF*11]. Conventionally, color mapping based on the differences in shapes and sizes is a widely used visualization technique. In this study, the proposed method provides local distance cues with a smooth visibility transition in overlapping areas instead of using color mapping, as shown in the bottom images of Figure 14. In the figure, the stronger the red color indicates that more red surface is at the front, and vice versa. On the other hand, the closer the blending colors are, the closer the two surfaces are located. This is accomplished using SFM. We set the z-thickness value as the maximum difference to display.

5.2.2. Ghosted illustration

The ghosted illustration is an effective tool for simultaneously visualizing interior and exterior structures while preserving clear shape cues. In particular, the traditional approach to realize ghosted illustration is based on a series of composition stages that modulate the

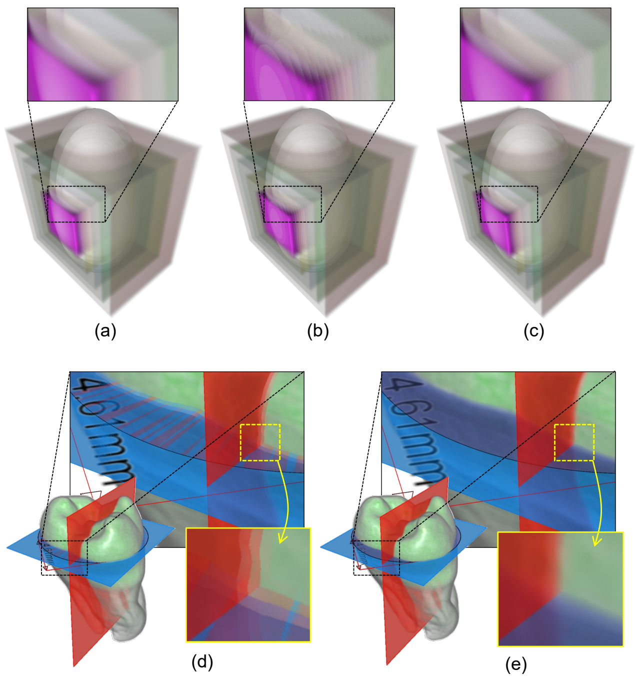


Figure 16: *Hybrid rendering. Synthetic 3D models with multiple transparent thin polygonal surfaces (ellipsoids) and thick volumetric surfaces (cubes) are rendered using (a) 100 times supersampling without SFM, (b) normal sampling without SFM, and (c) normal sampling with SFM. A real-world volume data with superimposing planes is rendered using normal sampling (d) without and (e) with SFM.*

transparency factors of exterior surfaces [BRV*10]. The smooth visibility transition based on the SFM offers ghosted effects by adjusting the visibility contrast of the occluding or occluded geometries with respect to transparency and depth. Examples are shown in Figure 15. To reveal the interior geometry, we applied a large z-thickness value to the fragments to make them overlap with each other instead of mitigating geometry occlusions by transparency.

5.3. Hybrid Rendering with Volumetric Object

We successfully demonstrated the applicability of the proposed method for the hybrid rendering of multiple volumetric and polygonal surfaces with transparency. Generally, polygon rendering is performed prior to volume rendering. Moreover, the sorted and merged fragments in the resolve pass should be stored back into the framebuffer similar to the screen-space rendering algorithms. The SFM in the volume-rendering pass is applied to the read-back fragments and volume samples at every sample step of the ray-casting algorithm [KW03].

The geometric intersections between thin transparent layers (polygonal surfaces) and thick translucent layers (volumetric surfaces) are shown in Figure 16. Figure 16(b) shows the banding artifact caused by discontinuous sampling. The SFM yields a high-quality image similar to the supersampling result because a

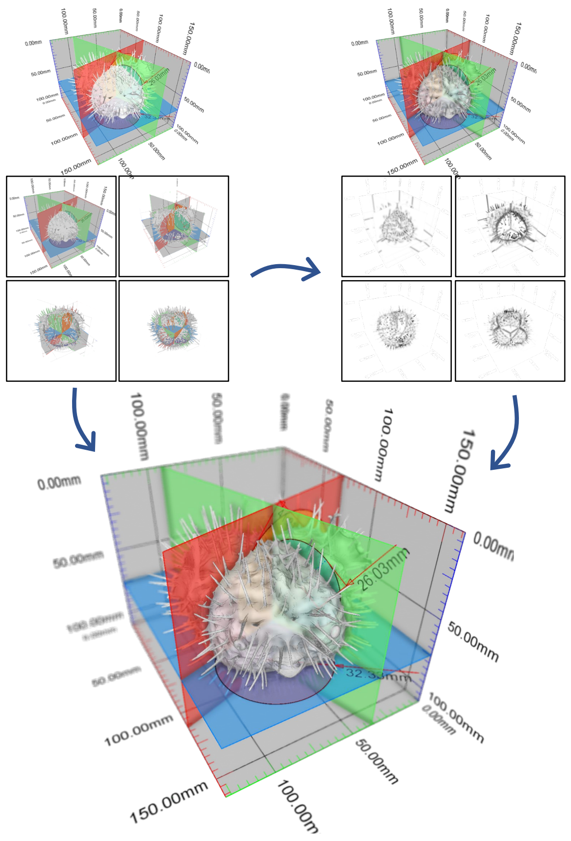


Figure 17: *Example of the multi-layered screen-space rendering of the dynamic ambient occlusion and depth-of-field effects. The multi-layered representation of merged fragments (top-left) is used for the dynamic ambient occlusion (top-right). A semi-transparent scene (bottom) with the ambient occlusion and depth-of-field effects is rendered using the multi-layered representation of the merged fragments and ambient occlusion values.*

smooth visibility transition of the SFM reduces such jaggies. Figures 16(d)(e) highlight the visual effect of the SFM in real-world hybrid rendering cases.

5.4. Extension to Screen-space Rendering

Fragments that represent the screen-space geometry must be stored in the frame buffer for the screen-space rendering. Subsequently, it reads the stored fragments and calculates the geometric interactions to produce the rendering effects. A single-layer representation in the screen space is not sufficient for geometric interactions, resulting in problematic rendered images. Various multi-layered screen-space rendering approaches [BS09, YWSM13] have been developed to address this problem.

The proposed method enables screen-space rendering using cost-effective multi-layer information, as demonstrated by the OIT. However, this can result in having distorted geometry that causes artifacts in screen-space rendering. Therefore, we calculated the geometric occlusion of two surface boundaries, namely the virtual front boundary where the shape of the original geometry is maintained and the back boundary whose depth value is equal to the depth value of the original geometry, based on the z-thickness model (refer to Figure 2) at the merged fragments with z-thickness values greater than $10 \times P(z)$.

To prove that the proposed method is valid in such multi-layered screen-space rendering algorithms, the fragments based on proposed method were applied to the dynamic ambient occlusion (AO) and depth-of-field (DOF) effects performed in the screen space. Figure 17 shows that screen-space rendering effects can be successfully applied to a scene containing semi-transparent primitives based on the proposed method. For this, we first applied the merged fragments to the multi-layered horizon-based ambient occlusion [BS09] by assigning the computed ambient occlusion values to each fragment. Subsequently, we applied the screen-space ray-tracing algorithm [LES09] to the fragment layers for a depth-of-field effect. Note that the geometry interaction must be calculated on the geometry configured by both the front and back surface boundaries of the z-thickness model for merged fragments with large z-thickness values.

6. Conclusions

In this paper, we presented a novel fragment-merging technique that merges partially overlapping fragments into one based on the proposed z-thickness model. We demonstrated that the proposed technique can be easily applied to multi-fragment rendering that handles individual fragments, such as DFB and SKB. We also presented the following advantages based on the proposed method. First, it mitigates the early problems on fragment overflow from which the SKB algorithm suffers, improving the OIT performance in terms of speed and quality. Second, it effectively suppress z-fighting problems by blending coplanar and coplanar-like surfaces. Third, it allows a smooth visibility transition (SFM), providing varying z-thickness effects that are useful in rendering overlapping and intersecting geometries. Finally, it was successfully applied to multi-layered screen-space rendering while delivering a wide depth range of screen-space geometry.

Moreover, we presented two approaches (i.e., DFB+SFM and SKB+OFM), which are based on the raster graphics pipeline, used for implementing the proposed method. These approaches were found to be efficient based on various experimental studies, including the comparison between the earlier models of the presented approaches (i.e., DFB and SKB) and a recent competitive implementation (i.e., MBT). Furthermore, we implemented a method to determine the z-thickness value based on the z-resolution of view frustum, which can be effectively address the problems in coplanarity such as the z-fighting and storage waste of coplanar-like fragments. Furthermore, our implementation allowed users to adjust the z-thickness values for special-purpose visual effects.

Although the proposed z-thickness model enables effective fragment merging, two additional information for a fragment (i.e., z-thickness value (4 bytes float-type) and accumulated opacity (4 bytes float-type)) result in unnecessary memory usage in scenes where overlapping layers are not critical.

Future studies will focus on the determination of the visibility of the merged fragment and its subdivided fragment. The proposed z-thickness model in this study may be combined with a higher-order approximation [MKKP18] of merging fragments, providing a more elaborate visibility determination of the subdivided fragment. Furthermore, an attention-based level-of-detail approach may be applied when determining the z-thickness values with respect to the pixel area. This reduces the fragment storage in areas that are not noticed by an observer [LKC08]. An approach of estimating scene complexity in screen space may be used to selectively apply the proposed method to pixels where complex gemometries generate overlapping fragments.

Acknowledgements

We thank the reviewers for their insightful comments and recommendations pertaining to proofreading. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1F1A1063740) and by the Research Grant of Kwangwoon University in 2021 (Dongjoon Kim). This research was also financially supported by Hansung University (Heewon Kye).

References

[Bak] BAKER S.: Learning to love your z-buffer. `https://sjbaker.org/steve/omniv/love_your_z_buffer.html`. Accessed: 2021-06-01. 6

[BBF*11] BUSKING S., BOTHA C. P., FERRARINI L., MILLES J., POST F. H.: Image-based rendering of intersecting surfaces for dynamic comparative visualization. *The visual computer 27*, 5 (2011), 347–363. 9

[BCL*07] BAVOIL L., CALLAHAN S. P., LEFOHN A., COMBA J. L., 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), pp. 97–104. 1, 2, 3, 4

[BKKB13] BAUER F., KNUTH M., KUIJPER A., BENDER J.: Screen-space ambient occlusion using a-buffer techniques. In *2013 International Conference on Computer-Aided Design and Computer Graphics* (2013), IEEE, pp. 140–147. 1, 3

[BM08] BAVOIL L., MYERS K.: Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK* (2008), 1–12. 2

[BRV*10] BRUCKNER S., RAUTEK P., VIOLA I., ROBERTS M., SOUSA M. C., GRÖLLER M. E.: Hybrid visibility compositing and masking for illustrative rendering. *Computers & Graphics 34*, 4 (2010), 361–369. 10

[BS09] BAVOIL L., SAINZ M.: Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH 2009: Talks*. 2009, pp. 1–1. 10, 11

[Car84] CARPENTER L.: The a-buffer, an antialiased hidden surface method. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (1984), pp. 103–108. 1, 2

[Cra10] CRASSIN C.: Opengl 4.0+ abuffer v2. 0: Linked lists of fragment pages. *Personal Blog, July* (2010). 2

[Duf17] DUFF T.: Deep compositing using lie algebras. *ACM Transactions on Graphics (TOG) 36*, 3 (2017), 1–12. 3

[ESSL10] ENDERTON E., SINTORN E., SHIRLEY P., LUEBKE D.: Stochastic transparency. *IEEE transactions on visualization and computer graphics 17*, 8 (2010), 1036–1047. 2

[Eve01] EVERITT C.: Interactive order-independent transparency. *White paper, nVIDIA 2*, 6 (2001), 7. 1

[FEE20] FRIEDERICHS F., EISEMANN E., EISEMANN E.: Layered weighted blended order-independent transparency. In *Graphics Interface 2021* (2020). 3

[FHSS18] FRANKE L., HOFMANN N., STAMMINGER M., SELGRAD K.: Multi-layer depth of field rendering with tiled splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 1 (2018), 1–17. 1, 3

[KG13] KARIS B., GAMES E.: Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice 4*, 3 (2013). 1

[KKL*16] KIM D.-J., KIM B., LEE J., SHIN J., KIM K. W., SHIN Y.-G.: High-quality slab-based intermixing method for fusion rendering of multiple medical objects. *Computer methods and programs in biomedicine 123* (2016), 27–42. 3

[Kub14] KUBISCH C.: Order independent transparency in opengl 4. x. In *Proceedings of the 2014 GPU Technology Conference* (2014), vol. 3. 6, 7

[KW03] KRUGER J., WESTERMANN R.: Acceleration techniques for gpu-based volume rendering. In *IEEE Visualization, 2003. VIS 2003.* (2003), IEEE, pp. 287–292. 10

[LES09] LEE S., EISEMANN E., SEIDEL H.-P.: Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics (TOG) 28*, 5 (2009), 1–6. 11

[LFS*15] LINDHOLM S., FALK M., SUNDÉN E., BOCK A., YNNERMAN A., ROPINSKI T.: Hybrid data visualization based on depth complexity histogram analysis. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 74–85. 1

[LKC08] LEE S., KIM G. J., CHOI S.: Real-time tracking of visually attended objects in virtual environments and its application to lod. *IEEE Transactions on Visualization and Computer Graphics 15*, 1 (2008), 6–19. 11

[LWXW09] LIU B., WEI L.-Y., XU Y.-Q., WU E.: Multi-layer depth peeling via fragment sort. In *2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics* (2009), IEEE, pp. 452–456. 2

[Max95] MAX N.: Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (1995), 99–108. 5

[MB13] MCGUIRE M., BAVOIL L.: Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques* (2013). 1, 2, 3

[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. 2, 4

[MCTB12] MAULE M., COMBA J. L., TORCHELSEN R., BASTOS R.: Memory-efficient order-independent transparency with dynamic fragment buffer. In *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images* (2012), IEEE, pp. 134–141. 2, 7

[MCTB13] MAULE M., COMBA J., TORCHELSEN R., BASTOS R.: Hybrid transparency. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2013), pp. 103–118. 3, 6

[Mes07] MESHKIN H.: Sort-independent alpha blending. *GDC Talk* (2007). 2

[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 (2018), 1–20. 1, 3, 7, 11

[MM17] MCGUIRE M., MARA M.: Phenomenological transparency. *IEEE transactions on visualization and computer graphics 23*, 5 (2017), 1465–1478. 1

[MMNL14] MARA M., MCGUIRE M., NOWROUZEZAHRAI D., LUEBKE D.: Fast global illumination approximations on deep g-buffers. *NVIDIA Corporation 2*, 4 (2014). 1, 3

[MY18] MALLETT I., YUKSEL C.: Deferred adaptive compute shading. In *Proceedings of the Conference on High-Performance Graphics* (2018), pp. 1–4. 1

[PD84] PORTER T., DUFF T.: Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (1984), pp. 253–259. 4

[PZVBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 335–342. 8

[RASS16] ROCHA A., ALIM U., SILVA J. D., SOUSA M. C.: Decal-maps: Real-time layering of decals on surfaces for multivariate visualization. *IEEE transactions on visualization and computer graphics 23*, 1 (2016), 821–830. 1

[Sal13] SALVI M.: Advances in real-time rendering in games: Pixel synchronization: Solving old graphics problems with new data structures. In *ACM SIGGRAPH* (2013). 2, 7

[SML11] SALVI M., MONTGOMERY J., LEFOHN A.: Adaptive transparency. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (2011), pp. 119–126. 1, 3

[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), pp. 151–158. 3

[VF12] VASILAKIS A.-A., FUDOS I.: Depth-fighting aware methods for multifragment rendering. *IEEE transactions on visualization and computer graphics 19*, 6 (2012), 967–977. 3

[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 (2015), 688–700. 1, 2, 4, 7, 8

[VVP20] VASILAKIS A., VARDIS K., PAPAIOANNOU G.: A survey of multifragment rendering. *STAR 39*, 2 (2020). 1, 2

[WGER05] WEXLER D., GRITZ L., ENDERTON E., RICE J.: Gpu-accelerated high-quality hidden surface removal. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2005), pp. 7–14. 3

[YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 1297–1304. 2

[YWSM13] YU K., WU S., SHENG B., MA L.: Layered depth-of-field rendering using color spreading. In *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry* (2013), pp. 77–82. 10