Real-Time Motion Blur Using Multi-Layer Motion Vectors

Donghyun Lee, Hyeoksu Kwon and Kyoungsu Oh *

Abstract: Traditional methods for motion blur, often relying on a single layer, deviate from the correct colors. We propose a multilayer rendering method that closely approximates the motion blur effect. Our approach stores motion vectors for each pixel, divides these vectors into multiple sample points, and performs a backward search from the current pixel. The color at a sample point is sampled if it shares the same motion vector as its origin. This procedure repeats across layers, with only the nearest color values sampled for depth testing. The average color sampled at each point becomes that of the motion blur. Our experimental results indicate that our method significantly reduces the color deviation commonly found in traditional approaches, achieving structural similarity index measures (SSIM) of 0.8 and 0.92, which represent substantial improvements over the accumulation method.

Keywords: real-time rendering; motion blur; image processing

1. Introduction

Motion blur is an effect created by the movement of objects between frames, blurring them in the direction of movement to produce realistic images. When this effect is applied to videos, the motion of objects is expressed in the image of each frame, enhancing realism. In general, sequences that contain a moderate amount of motion blur are perceived as natural. However, rendering motion blurred image requires significant computations, unlike cameras that generate motion blur automatically. As a result, much research has been done to reduce the resources required to generate motion blur. Although existing methods can achieve fast rendering speeds through post-processing, they generally rely on a single layer for motion blur generation [1–7]. If an object in the first layer moves, it cannot produce a color mixed with objects in the later layers. Another method to create motion blur is the accumulation method, which uses multiple rendering iterations with different times in a single frame to generate images that are subsequently averaged [8,9]. Because this method uses an actual rendering process, the motion blur effect is correctly simulated. However, this method introduces artifacts at low rendering iterations and performs slowly at high rendering iterations. Another method to create motion blur outputs a single image with motion blur using images from the previous frame and the current frame with a trained AI model [10]. However, it requires training data and can cause artifacts in real-time rendering environments with many objects.

Our proposed method employs multilayer motion vectors alongside color and depth images to calculate object movement, achieving a high-speed correct motion blur effect, as shown in Figure 1.

The motion vector is the difference in the position of each pixel between the previous and current frames, as shown in Figure 2.

All pixels between $P_{T-1}$ and $P_T$ in Figure 2 are averaged to obtain the motion-blurred color of pixel $P_T$. Each pixel searches for colors that pass through it, and the motion-blurred color is calculated by averaging all these colors.
Figure 1. (a) Scene with moving billiard balls (b) Scene with moving cars.

Figure 2. Motion vector corresponding to the movement of pixel $P_{T-1}$ to $P_T$ between frames $T-1$ and $T$.

This paper is organized as follows. Section 2 describes how motion blur is generated with multilayered images. Section 3 explains the generation of multilayered images, and Section 4 analyzes the rendering speed of each method. Section 5 discusses the research results and examines potential future directions of research. Section 6 concludes the study.

2. Motion Blur with Multilayer Images

Figure 3 shows three key steps of our method: multi-layer image rendering [11–14], motion blur using multi-layer image, and denoising with pixel reliability.
In this section, we explain how to compute motion blur using multi-layer image, which consists of three steps, as shown in Figure 3. The details of this process are described below.

2.1. Multi-Layer Image Rendering

We render scenes to generate multilayer images composed of color, depth, and motion vector maps. We do depth peeling rendering or stencil-routed rendering. Details will be given in Section 3.

2.2. Motion Blur Generation Using Multi-Layer Image

The process starts by drawing a full-screen quad using a multilayer image to compute the blurred color of each pixel. This process involves calculating the search area where motion vectors are likely to be found, defending the motion vector that passes through the current pixel, and then sampling the pixel in the opposite direction of the motion [15]. The motion blur color is then calculated from the sampled points.

2.2.1. Calculating Search Area

To calculate the motion-blurred color of a pixel we should find pixels that pass through the current pixel between $T - 1$ and $T$. Rather than checking all pixels, we calculate the search area based on the average and standard deviation of motion vectors. If a mipmap level $n$ is read from a pixel, it is an average of $2^n \times 2^n$ pixels centered on that pixel [16,17]. The standard deviation is then calculated as the square root of the difference between the average of the squared motion vector and the square of the average of the motion vector. In Figure 4, we illustrate the search area (blue box) generated according to the motion vector map.

![Figure 4. Search area generated according to motion vector map.](image)

2.2.2. Finding Motion Vector

The approximated search area can be used to find the motion vectors that pass through the current pixel according to the pseudocode shown in Figure 5, with three nested loops. Our algorithm uses nested loops to save the vector at a pixel if it corresponds to the current search path; otherwise, it continues the search using the backward motion vector path and using backward search to find the motion vector passing through the current pixel, as shown in Figure 6.
Figure 5. Pseudocode to determine the motion vector passing through the current pixel (P).

Figure 6. Find the motion vector passing through the current pixel (P) using the search area and backward search.

2.2.3. Calculating Motion Blur Effect

The motion vector obtained in the previous step is used to calculate the color of the motion blur effect according to the pseudocode shown in Figure 7. For each sample-point time, the motion vector is used to obtain the positions of neighboring pixels. If more than one sample point is sampled simultaneously, the color of the sample point closest to the camera is used. The average of all sample point colors is the color of the motion blur.

Figure 7. Pseudocode to calculate motion blur color and success rate using the motion vector.

2.3. Denoising with Pixel Reliability

Because the motion vector search is a stochastic method, a small number of iterations may incur noise. We therefore implemented a denoising phase by adding pixel reliability
to the Edge-Avoiding À-Trous filter [18–20]. As the success rate from the previous step, pixel reliability, defined as the ratio of successful sample-point times, can be used to minimize noise.

2.3.1. Edge-Avoiding À-Trous Filter

The Edge-Avoiding À-Trous filter uses the color and depth of neighbor pixels to get the denoised color of a pixel. If the color or depth difference of a neighbor pixel is large, the weight of it is low. As a result, noises are removed while edges are preserved. Additionally, pixels can be classified as noise and removed, as shown in Figure 8.

2.3.2. Pixel Reliability

In the previous step, we calculate by averaging the color of pixels corresponding to different times. As our algorithm is stochastic, we may fail to find pixels corresponding to some times. The ratio of found samples over the total number of times is pixel reliability. By incorporating pixel reliability as a weight in the Edge-Avoiding À-Trous filter, low-reliability pixels can be classified as noise and removed, as shown in Figure 8.

Figure 8. Noise filtering results (a) Unfiltered (b) Edge-Avoiding À-Trous-filtered (c) Filtered by reliability.

3. Multilayer Images

Rendering motion blur with a single layer often fails to apply the effect correctly. This limitation is shown in Figure 9a. The blue ball is moving from left to right. For pixel P for time T0, T1, and T2, the corresponding pixels are on the blue ball. But for T3, T4, and T5, the corresponding pixel is P on the second layer. Our multilayer approach, illustrated in Figure 9b, allows points that are missed with a single layer to be captured in a second layer.

Figure 9. Color sampling of points Tn opposite of the motion vector found at the current pixel using a single layer (a) a single layer and (b) multiple layers.
We used depth peeling rendering or stencil-routed rendering to get multi-layer images. In this section, we will explain both methods.

3.1. Depth Peeling by Comparison with Previous Layer Depth

Scenes are rendered multiple times across multiple layers through iterated depth comparison and rendering [11, 12]. In each pass, polygons closer than the previous depth are killed, and the closest surviving polygon is stored. Figures 10a and 11 show multi-layer rendering via depth peeling by comparison with previous layer depth.

![Figure 10](image1.png)

**Figure 10.** Simple pseudocode to multi-layer rendering (a) Depth peeling by comparison with previous layer depth and (b) Depth peeling by stencil-routed rendering.

![Figure 11](image2.png)

**Figure 11.** Depth peeling by comparison with previous layer depth pipeline.

3.2. Depth Peeling by Stencil-Routed Rendering

Stencil-routed rendering preserves color only in pixels with the same stencil value as the comparison value and decreases the stencil value of the pixel where the color is stored by 1 [13, 14]. Subsequently, multilayer images are generated by depth-wise sorting after rendering all objects. Figures 10b and 12 show multi-layer rendering via depth peeling by stencil-routed rendering.
3.2.1. Polygon Loss Problem

Modern GPUs, which use multiple sampling points per pixel, can experience a loss of polygon information if a polygon covers only a few of these sampling points. This condition leads to unchanged stencil values across different renders, resulting in fewer layers, as shown in Figure 13.

3.2.2. Comparing Average Depth of Sample Points

The first layer with the highest sample count is replaced with the image generated by the general rendering process. However, some pixels use information from the lost pixels to calculate the color of the motion blur. In situations where the polygon covers fewer sample points, resulting in ‘lost pixels’; these are identified by their higher average depth compared to the surrounding, non-lost pixels. These lost pixels can be detected by comparing the depth of the current pixel with the average depth of its neighbors. Figure 14 shows the reliability filter performance of the depth comparison method. In Figure 14a, we did not apply the depth comparison method and saw clear artifacts. However, in Figure 14b, we applied the depth comparison method and saw that the artifacts disappeared.
Figure 14. Reliability filter performance (a) without depth comparison and (b) with depth comparison.

4. Experimental Analysis

The accumulation method and the two rendering methods were experimentally analyzed in a Direct3D 11 (Intel i9-10900K, Samsung 128 GB RAM, Nvidia RTX 3080) environment. The accumulation method renders a single blurred image from rendering iterations within a frame.

4.1. Samples per Frame

As shown in Table 1, we analyzed performance by varying the number of samples per frame in scenes with 860 K triangles, noting only a minor decrease in frames per second (FPS) with increasing samples.

Table 1. FPS to the sample per frame.

<table>
<thead>
<tr>
<th>Measured Variable</th>
<th>Sample per Frame</th>
<th>Frame per Second (FPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth Peeling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Stencil Routing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>134</td>
<td></td>
</tr>
</tbody>
</table>

4.2. Triangles per Scene

Table 2 compares the speed of two rendering methods for several scenes with different numbers of triangles.

Table 2. FPS to the number of triangles.

<table>
<thead>
<tr>
<th>Measured Variable</th>
<th>Num of Triangles per Scene</th>
<th>Frame per Second (FPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth Peeling</td>
<td>520 K Triangles/Scene</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>860 K Triangles/Scene</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>1160 K Triangles/Scene</td>
<td>51</td>
</tr>
<tr>
<td>Stencil-Routed</td>
<td>520 K Triangles/Scene</td>
<td>184</td>
</tr>
<tr>
<td></td>
<td>860 K Triangles/Scene</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>1160 K Triangles/Scene</td>
<td>116</td>
</tr>
</tbody>
</table>

4.3. Comparing Motion Blur via SSIM

We evaluated the perceptual quality of images using the structural similarity index measure (SSIM) [21]. The three methods were subjected to the same denoising process, and the SSIM calculations were averaged over 10 randomly selected images from the same scene with triangles at 860 K. As shown in Table 3, both the depth peeling and stencil-routed methods achieved SSIM values above 0.93.
Table 3. Comparison in terms of SSIM.

<table>
<thead>
<tr>
<th>Measured Variable</th>
<th>Samples per Frame</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth Peeling</td>
<td>16</td>
<td>0.950942</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>0.959914</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>0.964858</td>
</tr>
<tr>
<td>Stencil-Routed</td>
<td>16</td>
<td>0.932637</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>0.933672</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>0.934416</td>
</tr>
</tbody>
</table>

4.4. Comparing Motion Blur Image Quality

Figure 15 compares the image quality of three rendering methods. We chose the accumulation method as our reference because it is the closest to the correct motion blur. The checkerboard pattern on the front of the car is not clear in scenes with 16 samples per frame but is clear in scenes with 32 samples per frame.

![Figure 15. Comparing image quality of the rendered scene by each rendering method and multi-layer rendering method.](image)

4.5. Comparing Motion Blur by Number of Layers

Figure 16 shows the impact of varying layer counts on motion blur quality. Notably, a single layer does not adequately convey motion blur, while multiple layers significantly enhance the effect.
5. Discussion

In this section, we compare previously proposed motion blur generation algorithms with our method and discuss potential improvements and future work.

The previously proposed methods can be broadly categorized into accumulation buffers, the use of AI, and post-processing. First, the accumulation method creates the motion blur effect by repeating the rendering in a single frame. This method produces the most ideal motion blur effect because it repeats the rendering. However, this method takes a long time to create the motion blur effect because it needs to go through many rendering iterations to reduce artifacts, which means it is not suitable for real-time rendering environments.

Second, using AI is a fast way to create motion blur effects. However, it requires a large amount of data to train the agent, and it consumes a lot of resources to create the data. It can also produce artifacts in real-time rendering environments with many objects and rapid changes. Also, because the agent cannot accurately explore the processes it outputs, it takes a long time to correct errors when they occur.

Finally, post-processing methods that use information from the current frame and previous frames to create motion blur effects are the most popular. Single-layer post-processing is a popular method, but if objects in the first layer move, it is not possible to create a motion blur effect that blends with objects in later layers. Many algorithms have been developed and proposed to deal with this problem, but they have limitations because they do not use the information provided by the real layer.

Our method uses depth filling and stencil routing techniques to accurately capture multiple objects moving in different directions and at different speeds. It also uses motion vectors from multiple layers to reproduce blur close to the answer. This allows us to approximate the colors and details of the original scene more accurately, which are often lost in traditional methods. We have shown that our approach achieves higher SSIM values, indicating superior image quality, and maintains high frame rates, confirming its practical application in real-time rendering.

In future work, we will investigate ways to improve the motion vector detection process and streamline the multilayer rendering process. If the efficiency of the motion vector detection process is increased, the denoising process can be eliminated. Also, due to the nature of our method, which uses information from multiple layers, resources are essentially consumed in drawing multiple layers. In the case of depth filling, for example, rendering must be repeated in the same frame to get as many layers as needed. Stencil routes are rendered once, but there is a process of sorting by depth, and additional denoising techniques must be used. By studying and applying these improvements, you can expect to increase the efficiency and speed of motion blur generation.
6. Conclusions

We are using multilayer images and motion vectors, and we have developed a method that significantly enhances the quality and speed of motion blur effects. This approach matches the original colors more accurately and maintains high FPS even in complex scenes. We compared the SSIM index to the accumulation method and found a result of 0.92 for depth peeling and 0.8 for the stencil-routed method. The stencil-routed method runs the depth-based denoising process one more time due to the polygon loss problem. This is where the SSIM difference between the two methods comes from. Our results are expected to enable high-speed and high-quality motion-blurring effects in real-time rendering.

Author Contributions: Conceptualization, K.O.; methodology, D.L. and H.K.; software, D.L. and H.K.; validation, K.O.; formal analysis, K.O.; investigation, D.L.; resources, D.L.; data curation, H.K.; writing—original draft preparation, H.K.; writing—review and editing, K.O.; visualization, D.L.; supervision, K.O.; project administration, K.O.; funding acquisition, K.O. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Innovative Human Resource Development for Local Intellectualization program through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (IITP-2024-RS-2022-00156360).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References


Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.