Real-Time Lens Blur Effects and Focus Control: Supplementary Material

Sungkil Lee[1] Elmar Eisemann[1,2] Hans-Peter Seidel[1]
[1]Max-Planck-Institut für Informatik [2] Télécom ParisTech/CNRS-LTCI/Saarland University

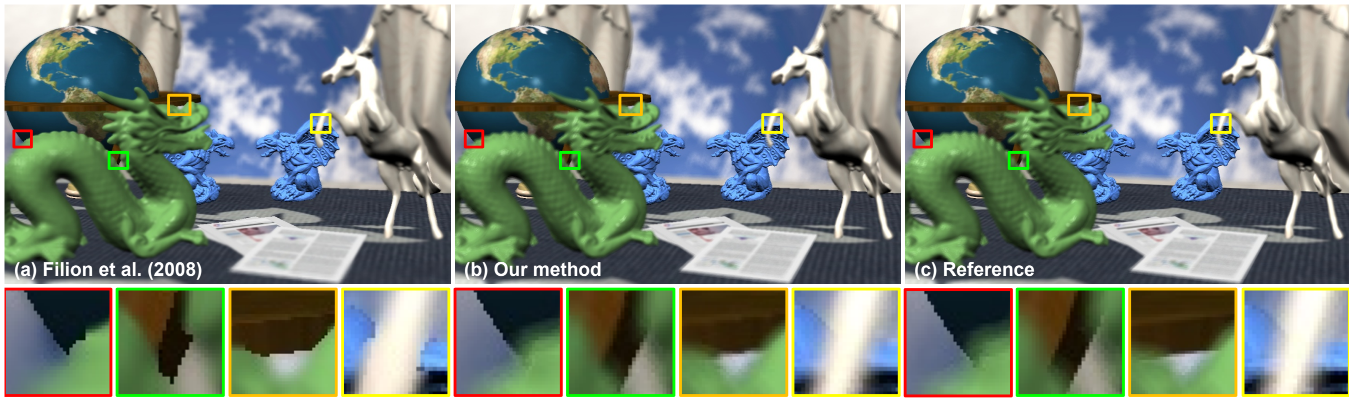

Figure 1: Comparison between a single-image method (a), our method (b), and a reference method [Haeberli and Akeley 1990] (c).

This document is intended to give further insight into our algorithm. We show a comparison between our algorithm with very efficient single-image methods, discuss the precision of our depth peeling, as well as the memory consumption of our solution. Finally, we address antialiasing to further improve the quality of output images.

1 Comparison with Single-Image Method

Methods used in game engines [Hammon 2007; Filion and McNaughton 2008] often sacrifice quality for efficiency. Usually, a single color and depth texture undergoes a filtering process to approximate visibility and depth of field (DOF) effects. For filtering a variation of mipmaps ensures high performance (e.g., 1-5 ms), even on low-end platforms.

Because a standard blur would result in leaking from in-focus to out-of-focus regions, the depth buffer is used to adapt the filtering process. However, despite these measures, missing scene information, especially in the presence of blurred foreground objects, can make the approaches fail frequently. Convincing effects can only be achieved for small blur kernels.

Both mentioned game techniques use non-physical lens models, which makes a fair comparison difficult. In Figure 1, we tried to match the blur level of the single-image approaches. Due to the comparably small blur, our solution only needs 32 rays and 2 layers to produce a high-quality output. Nonetheless, even with these settings, our method is still slower than a filtering solution. Our approach needs 14 ms per image (of which 3 ms are used by the raytracing process) which is approximately twice as slow as [Filion and McNaughton 2008] with 6 ms (5.4 ms for rendering, 0.6 for postprocessing). Nevertheless, artifacts appear in the postprocessing methods, even for such a small blur radius. Our solution delivers a convincing outcome and can treat larger blur kernels gracefully. The same findings hold when comparing our method to another recent mipmap-based solution [Lee et al. 2009b]. This method shares the idea of the former approaches, but matches a physical lens behavior more closely.

Overall, the postprocessing methods can achieve reasonable results for small kernels and achieve a very high performance. Nonetheless, they cannot recover hidden foreground surfaces (see the insets in the figure) which can lead to artifacts and restrict achievable blur.

2 Precision of Our Depth Peeling

Our peeling method allows us to perform an artifact-free rendering in the sense that rays do not miss surfaces. Our approach exploits the ambiguity of an image- and geometry-based representation which implies that we can extend the single-pixel umbra by half a pixel to each side without introducing artifacts. It is not possible to safely extend the umbra region further. Figure 2 illustrates the result of an extended-umbra peeling beyond the valid extent. These artifacts do not need to be pronounced and mostly appear at depth continuities and, further, these artifacts can almost disappear

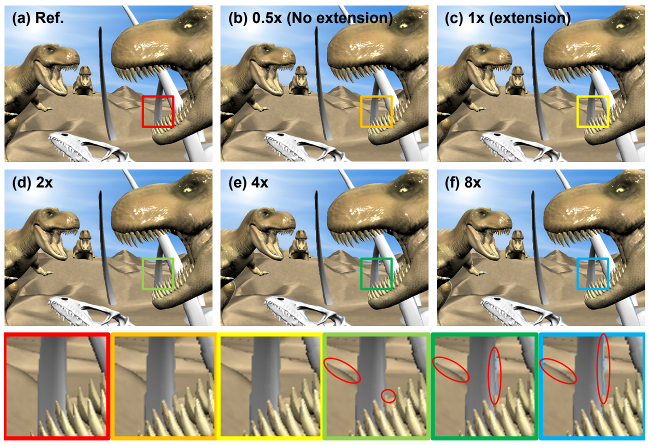

Figure 2: Comparison of the raytraced images using different thresholds for depth peeling. Our strategy, umbra culling (b) and its extension to neighboring pixels (c) show that no rays are missing, very close to the reference rendering (a). More extensions (e,d,f) reveal leaking yet small amounts of rays around geometry edges.

when the rays are integrated to produce the final image. This makes our approach relatively robust to this artifact and one could choose more aggressive settings (e.g., 4 pixels), although we did not make use of this possibility.

The major quality difference with respect to the accumulation buffer method (that moves the camera) is that the peeling is done only once. Hence, the surface sampling does not change for each lens ray. In practice, the effect is somewhat comparable to anti-aliasing, but the difference is much more subtle because the DOF blur kernel hides potential differences.

3 Memory Consumption

Our memory consumption is comparably low. Let N be number of layers, and S be the maximum footprint size of the lens rays. By using 16-bit floating-point depth, the peeled layers consume $N \times$ (3 (RGB) + 2 (depth)) = 5 bytes. The N-Buffers store the minimum and maximum depths, hence, 4 bytes. Four layers share one N-Buffer and we compute it from a downsampled version of the original buffer.

Let M be the downsampling exponent for the N-Buffers; for example, $M = 3$ denotes an image of 1/8 of the original one on each side. Then, the cost per original pixel is: $4 \cdot {}^{N}\!/\!_{4} \cdot \left({}^{(\log_2 S - M)}\!/\!_{4^M}\right)$. Finally, we rely on a mipmap to fill up the gap between N-Buffers and original resolution resulting in $4 \cdot {}^{N}\!/\!_{4} \cdot \sum_1^M {}^{1}\!/\!_{4^i}$. Thus, the total cost is:

$$5N + N\left(\frac{\log_2 S - M}{4^M}\right) + N \cdot \sum_1^M \frac{1}{4^i}.$$

For M=3, N=8, S=256, 32 MB are used for a 1024×768. In practice, $N = 4$ is sufficient for our approach, leading to 16 MB. Due to the strong blur, [Lee et al. 2009a] need 16 layers for the scenes depicted in the paper, resulting in 72 MB of memory. Further, their slower ray-tracing performance usually implies the need for a supplementary per-layer mipmapping. Consequently, the consumption increases to 92 MB, while still not matching the quality of our 16 MB result.

4 Multisampled Antialiasing

Multisampled Antialiasing (MSAA) is an interesting feature, but for DOF rendering only needed in the focused region. By clipping the scene, one can render only the focused part with MSAA—the remaining parts are blurred and MSAA is unnecessary. We can then compute a coverage value in each pixel [Lee2009a] using the alpha channel. During DOF rendering, we can look up the current pixel's corresponding MSAA pixel (there is only one because the MSAA part is focused). It is then possible to integrate the blending in our ray-tracing step and achieve an anti-aliased outcome. Figure 3 compares an image produced by this method to the image rendered without MSAA.

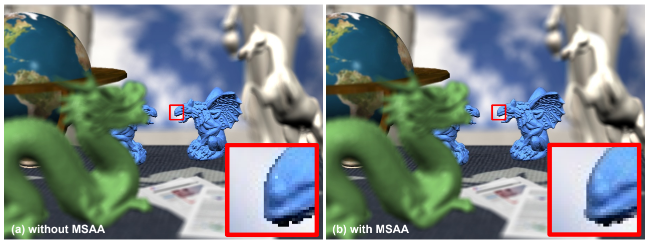

Figure 3: Comparison between images generated without MSAA (a) and with MSAA (b).

5 Curvature of Field Using Geometric Lens

Finally, we demonstrate one more example using curvature of field, not shown in the paper due to space limitations. In Figure 4, we imitate the appearance of a photo shot by a lens like LensBaby$^{\text{TM}}$. The figure shows the illusion of still velocity generated by our geometric lens model.

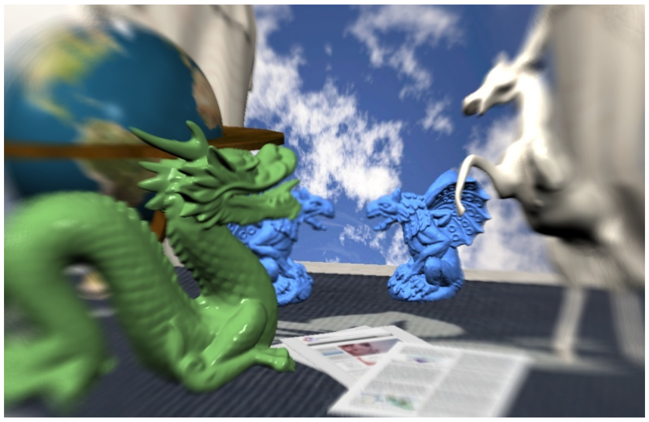

Figure 4: Curvature of field rendered by our method.

Acknowledgements

The 3d models are provided in the courtesy of the Stanford 3D Scanning Repository, AIM@Shape, INRIA GAMMA, TurboSquid, and Taschen Books (500 3D Objects).

References

FILION, D., AND MCNAUGHTON, R. 2008. Starcraft II effects & techniques. In *ACM SIGGRAPH 2008 classes*, 133–164.

HAEBERLI, P., AND AKELEY, K. 1990. The accumulation buffer: Hardware support for high-quality rendering. *Proc. ACM SIGGRAPH*, 309–318.

HAMMON, JR., E. 2007. Practical post-process depth of field. In *GPU Gems 3*, H. Nguyen, Ed. Addison-Wesley, ch. 28, 583–606.

LEE, S., EISEMANN, E., AND SEIDEL, H.-P. 2009. Depth-of-Field Rendering with Multiview Synthesis. *ACM Trans. Graph. 28*, 5, 134:1–6.

LEE, S., KIM, G. J., AND CHOI, S. 2009. Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation. *IEEE Trans. Vis. and Computer Graphics 15*, 3, 453–464.