Real-time Depth of Field using Multi-Layer Filtering

Kai Selgrad* Christian Reintges* Dominik Penk* Pascal Wagner* Marc Stamminger*
Computer Graphics Group, University of Erlangen-Nuremberg

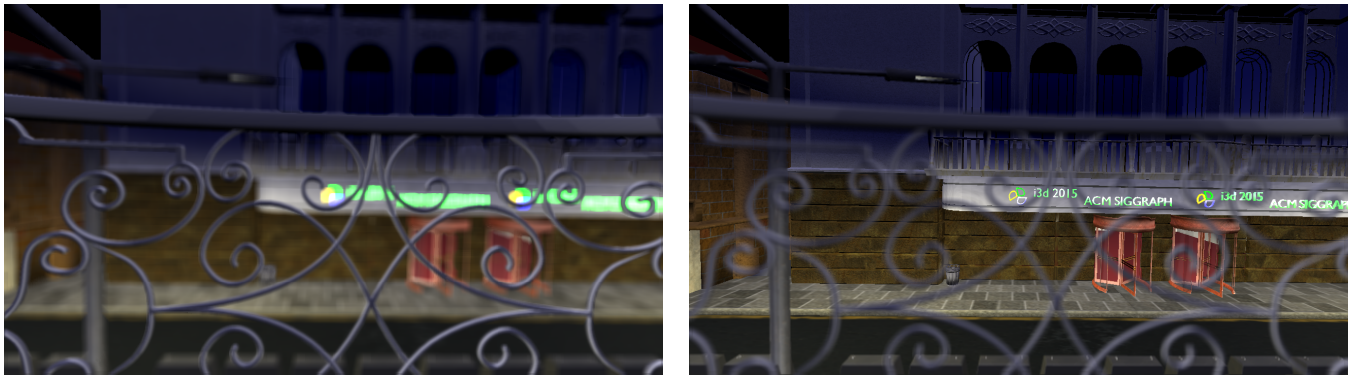

Figure 1: *Our novel approach to rendering depth of field in real-time provides pleasant and plausible results (left), also for complicated cases where out-of-focus geometry in the near-field would occlude important scene geometry (right).*

Abstract

We present a novel technique for rendering depth of field that addresses difficult overlap cases, such as close, but out-of-focus, geometry in the near-field. Such scene configurations are not managed well by state-of-the-art post-processing approaches since essential information is missing due to occlusion.

Our proposed algorithm renders the scene from a single camera position and computes a layered image using a single pass by constructing per-pixel lists. These lists can be filtered progressively to generate differently blurred representations of the scene. We show how this structure can be exploited to generate depth of field in real-time, even in complicated scene constellations.

CR Categories: I.3.3 [Computer Graphics]: Image Generation

Keywords: depth of field, rendering, real-time, layers

1 Introduction

Depth of field is a visually important effect, and a frequently used artistic element in photography and movies. It is caused by the finite size of the lens of a camera (or of the eye) that gathers light. The lens focuses incident light on the film (or the retina), but it does so only for light rays originating from a certain distance, the focus depth. Light rays emanating from before or behind this focus depth become blurred. From the view of a pixel, its eye rays are scattered over the entire lens, and they converge on the focus plane. Before or behind the plane they span the so-called *circle of confusion*, the world space size of which varies linearly with the distance to the focus plane. A detailed description of the effect can be found in Riguer et al. [2003].

A simulation of depth of field requires an additional integral over the 2D-range of the lens, which makes computation expensive. Thus, for real-time rendering, approximate methods have been developed and integrated successfully in state-of-the-art rendering engines.

Most of these approaches apply a spatially varying blur filter as a post-process to the final high dynamic range image, just before tone-mapping. Even if the results often look convincing, there are cases that such approaches cannot handle correctly. An example is shown in Figure 1 where a railing in the foreground is blurred by depth of field, making parts of the background visible that cannot be seen in a sharp image. The effect is evident in objects close to the viewer, since the blur becomes particularly strong there. Any approach based on a sharp input image cannot generate this effect, and the results quickly appear misrepresented for these cases.

In this paper, we propose a novel approach to rendering depth of field in real-time that addresses this effect correctly. It takes a multi-layer image, obtained by gathering fragments in per-pixel linked lists [Yang et al. 2010] as input. Building this data structure is computationally more expensive than a simple, single-layer G-buffer, but it contains all the information necessary for generating the effect described above properly. Note that this representation also holds all data required to render transparent objects, and, therefore, our method integrates transparency seamlessly.

Next, we build a mip-map-like filter pyramid on this multi-layer image by means of a particular process that only merges fragments of similar depth. As a result, we obtain a sequence of multi-layer images, filtered by increasing filter sizes. In recent work [Selgrad et al. 2014], we presented the same data structure, yet for quickly computing soft shadows.

For final rendering, we compose the filtered layered depth images by only using fragments with a filter size that corresponds to their required circle of confusion. Our algorithm generates the inward-blur effect correctly, achieving good results even for strong blur in the foreground.

*firstname.lastname@fau.de

Blurring the layered depth images is computationally more expensive than a simple post-processing filter, but we achieve real-time performance even at HD resolution using a number of optimizations that will be described in this paper.

2 Related Work

Depth of field is a result from the finite aperture of real-world optical systems. Simulating depth of field requires an integral of the light field incident to the lens over the opened range of the aperture.

In offline-rendering, this integral can be combined with the integral over the pixel's area (anti-aliasing), over time (motion blur), over area lights, over light frequency (spectral rendering) and others. Since all integrals require a large number of samples per pixel regardless, depth of field can be added almost seamlessly. Nevertheless, it is worthwhile to examine this integral further to optimize the number of required samples (e.g. [Lehtinen et al. 2011; Belcour et al. 2013; Lei and Hughes 2013]), to replace the point samples with line samples [Tzeng et al. 2012], or to include even more difficult lens effects [Hullin et al. 2012]. Such approaches are, however, still far from real-time.

In real-time, hardware-based rendering environments, integration is more difficult. The likely oldest approach is to use the accumulation buffer to average a number of renderings from slightly shifted view points [Haeberli and Akeley 1990]. Each of these images corresponds to a single sample on the lens. Generating images without ghosting requires many such render passes, and is, therefore, too inefficient for real-time applications.

Other approaches take a pinhole image, potentially with multiple layers, and blur the image by splatting each pixels' contributions to their neighbors [Krivanek et al. 2003; Lee et al. 2008; Kosloff et al. 2009]. As a result, every pixel receives a number of transparent contributions that must be blended correctly, which requires sorting. A variant is blurring using heat diffusion [Kass et al. 2006], which is fast but mainly used for previews in movie productions due to the limited quality.

Whereas splatting can be regarded as a scattering approach (distributing information from a pixel to its neighbors), approaches that gather the information for a pixel of interest from their neighborhood are better suited for hardware rendering. The typical approach that is also used in many state-of-the-art rendering engines is, instead, to apply a filtering procedure to a rasterized, sharp image assuming a pinhole camera as a post-processing step. For each (sub-)pixel the size of the circle of confusion is computed and an image-space filter with this size applied. To avoid filtering over silhouettes, where the filter size changes abruptly, a bilateral filter can be applied that only includes fragments of similar depth or that excludes further fragments [Riguer et al. 2003]. The filtering step for these approaches still requires a significant amount of time, firstly, because filter kernels can become larger (hundreds of pixels), and secondly because the varying filter size makes the filter non-separable.

Yet, with some optimizations, decent real-time results are possible. However, none of these post-processing approaches using a single-layer input image can properly address the effect of depth of field that makes parts of the scene occluded in the pinhole view visible. This effect is particularly strong for nearby-objects and can make thin structures completely transparent.

This transparency effect can be simulated by subdividing the scene into depth layers that are then filtered separately [Schedl and Wimmer 2012]. Even if the method is optimized for GPUs, no real-time rates are reported for reasonable resolutions. A real-time method that can also generate transparent, nearby objects is the method by Lee et al. [2009]. In their approach, lens blur is simulated similarly to offline-rendering by sampling the lens with a number of rays. The method reaches real-time by rendering the scene to a number of layers, where each layer is represented as a height field. With this representation, it is possible to efficiently intersect nearby lens rays with a binary search on the height fields. Yet, difficult setups (glossy highlights, large, near-field blur) require a large number of rays to achieve smooth results, which directly affect computation time.

All of the above methods, as well as our approach, use a simple box, circle, or Gaussian blur. Yet, the lens blur of real-world cameras has a more interesting shape, called "Bokeh". The Bokeh is the image of a point light out of focus. It is different for all lenses and apertures, and it usually reveals the shape of the aperture. Some depth of field methods focus on the ability to reproduce this Bokeh effect [McIntosh et al. 2012].

3 Algorithm Overview

Our algorithm encompasses three phases:

Collection of Scene Fragments In this step, the scene is rendered and all fragments within a pixel are gathered in per-pixel linked lists [Yang et al. 2010] that are recomputed on the fly for every frame.

Filtering The previously computed fragment lists are then filtered and merged via a filtering scheme by mip-mapping and stack-filtering the lists similarly to Selgrad et al. [2014]. During the filtering step (Section 5) silhouette edges, i.e. partial coverage, translate to reduced opacity in the levels filtered more strongly.

Accumulation Finally, the filtered lists are accumulated per pixel, where from each level only those fragments are used that have a filter size corresponding to the current circle of confusion.

In the following sections we describe these steps in detail.

4 Data Structure Generation

The base level of our multi-layer hierarchy is generated with a single rendering pass of the scene geometry. In this pass each front-facing fragment is shaded in the pixel shader and collected in a fragment list of the corresponding screen pixel using the technique by Yang et al. [2010]. The list entries encompass the fragments' color, opacity and depth.

The per-fragment information is compactly stored in a three-component vector using half-precision, floating point values for color and opacity and a single precision value for depth.

To facilitate further processing of the collected data the lists are then sorted by increasing depth values and stored compactly in the form of consecutive arrays. This is achieved by a scanning pass [Sengupta et al. 2007] applied to the list-length counters (computed during the collection phase) before the actual sorting operation. The sort operation can thus be implemented to directly sort into the newly indexed arrays.

While sorting it is possible to impose an upper limit on the number of fragments per pixel. On the base layer, we keep the closest three or four fragments, because only in rare and particular cases (such as scenes with many semi-transparent objects) can more fragments (on the base layer) contribute to the final image. See Section 7 on how this parameter affects rendering performance. Note that we collect fragments regardless of their alpha values, which is in contrast

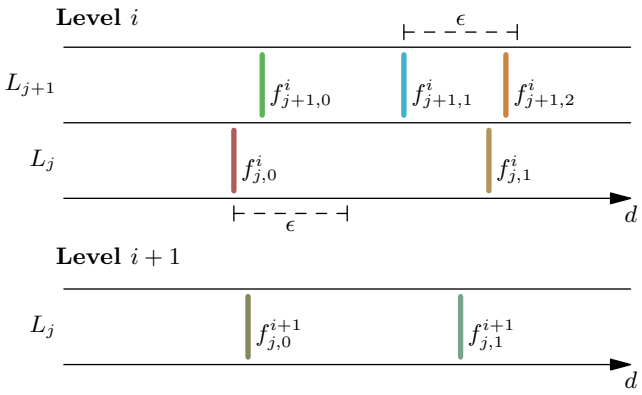

Figure 2: *While filtering from level i to $i+1$ fragments from the source lists are merged if they are of similar depth. In the example shown the fragments $f^i_{j,0}$ and $f^i_{j+1,0}$ are merged into $f^{i+1}_{j,0}$, and the fragments $f^i_{j+1,1}$, $f^i_{j,1}$ and $f^i_{j+1,2}$ are merged into $f^{i+1}_{j,1}$.*

to using per-pixel linked lists for order independent-transparency where no fragments beyond the first opaque fragment are collected.

5 Filtering Levels

A key aspect of our method is the incremental filtering of the base level to obtain multi-layer images with increasing filter size.

5.1 Filtering via Mip-Mapping

The most simple and efficient, albeit lower-quality, filtering method is to downsample the fragment lists similarly to mip-mapping. This is achieved by filtering neighboring fragment lists where we merge the 2×2 lists from level i to produce a single list of level $i+1$, generating a new, filtered multi-layer image with half the resolution.

This is accomplished in a manner similar to merge sort's combination step (which generates a single sorted list from sorted sub-lists). At level i the fragment closest to the camera, $f^i_{j,0}$ from list j is selected. This corresponds to the red fragment shown in Figure 2. The fragment's data is used to initialize the fragment $f^{i+1}_{j,0}$ that will be generated for the filtered level. Further fragments are accumulated in $f^{i+1}_{j,0}$ (e.g. $f^i_{j+1,0}$, the green fragment in Figure 2) until a fragment farther from $f^i_{j,0}$ than the depth threshold ϵ is encountered (e.g. $f^i_{j+1,1}$, the blue fragment in Figure 2). At this point the accumulated fragment is emitted, a new fragment, $f^{i+1}_{j,1}$, to be generated for the filtered level is initialized and accumulation proceeds as described.

Merging close-by fragments (instead of a simple sorting step) prevents the lists from becoming excessively long in higher levels. While merging, two cases must be considered. Firstly, fragments from the same list (e.g. $f^i_{j+1,1}$ and $f^i_{j+1,2}$, i.e. the blue and orange fragments in Figure 2) are generally semi-transparent occluders for the following ones. Therefore, we employ alpha blending using the over operator, where the merged fragment's color and opacity (c' and α', respectively) are computed from the fragments a and b by $\alpha' = 1-(1-\alpha_a)(1-\alpha_b)$, $c' = \frac{1}{\alpha'}\left(\alpha_a c_a + (1-\alpha_a)\alpha_b c_b\right)$. This can be done front-to-back, i.e. in traversal order.

Secondly, fragments from different per-pixel lists are known not to overlap; therefore, additive alpha blending is employed to compute the color and opacity of the final fragment, based on the intermediate data accumulated as described above, i.e. α'_1, α'_2, α'_3 and α'_4,

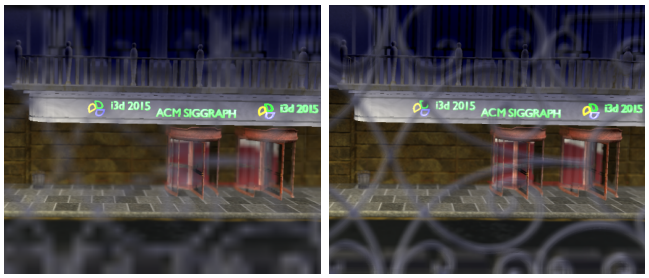

Figure 3: *Filtering multiple levels using the mip-mapping scheme only exhibits strong artifacts, especially for blur in the near field (left). The use of a filter stack preserves the fine structure while still filtering it appropriately (right).*

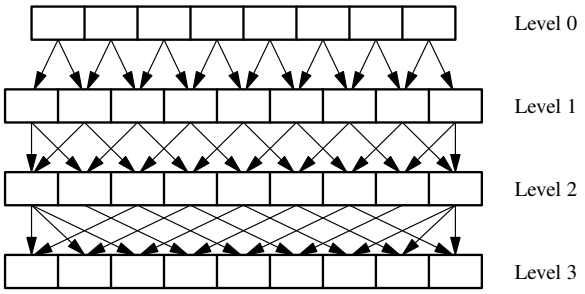

Figure 4: *Data flow for the computation of a filter stack. A uniform filter size is applied per level by extending the border cells where appropriate (i.e. by clamping).*

where empty sublists set $\alpha'_j = 0$. The combined fragment's opacity is then $\alpha_{\text{merged}} = \frac{1}{4}(\alpha'_1 + \ldots + \alpha'_4)$ and the resulting color is given by $c_{\text{merged}} = \frac{\alpha'_1 c'_1 + \ldots + \alpha'_4 c'_4}{4\alpha_{\text{merged}}}$. This combination ensures that opaque surfaces remain opaque at higher levels and, together with the first step, yields semi-transparent fragments on silhouette edges where not all lists contain fragments in the given range.

5.2 Filtering via Stack and Y-Map

Obviously, using a mip-mapping scheme leads to severe downsampling for larger filter sizes, and thus to a reduced quality of the filtered data as exemplified in Figure 3. Such artifacts are especially noticeable when there is strong blur in the near-field, i.e. when a very coarse filter level is applied close to the camera.

These artifacts mainly stem from the fact that we use a simple box-filter. The use of more sophisticated filters, e.g. an À-Trous filter kernel [Dammertz et al. 2010] or a Gaussian, is, in principle, possible. However, the larger kernel size would result in a drastic increase in computation time, and as filtering time is a limiting factor in our method (see Section 7), we opted for a combination of mip-mapping and filter stack, as described in the following. This choice minimizes downsampling artifacts and facilitates maintaining the very efficient 2×2 box-filter.

With a filter stack, too, an exponentially increasing filter size is applied to the image, however without a reduction of the image's size (the image size actually increases by 1 in both dimensions, as can be seen in the illustration of the filter pattern in Figure 4). Figure 3 shows a comparison between mip-mapped and filter stack-based blur using our algorithm; mip-mapping clearly exhibits the underlying pixel structure whereas the stack filtered version is much smoother.

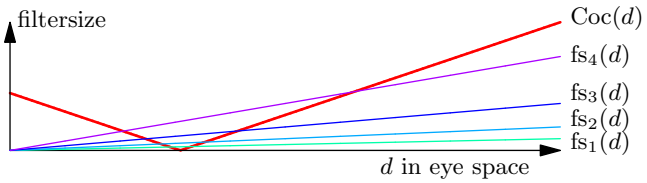

Figure 5: *The circle of confusion (red) and the filter sizes represented by different levels of our hierarchy (increasing levels from bottom to top).*

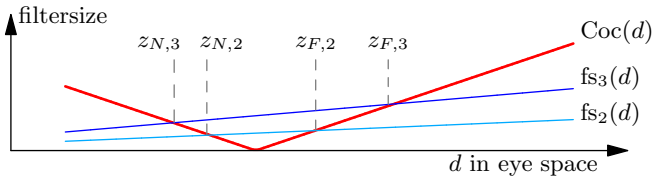

Figure 6: *Two filtered levels with entry and exit borders determined by the circle of confusion. Level 2 is applied between $z_{N,3}$ and $z_{N,2}$ as well as between $z_{F,2}$ and $z_{F,3}$. Figure 5 shows the complete example without level borders.*

The downside of using a filter stack is that, with higher levels, the resolution does not decrease while the length of the per-pixel lists potentially increases, leading to a much more computationally intensive filter. In our approach we trade image quality for performance with a structure called Y-map [Schwarz and Stamminger 2008] that combines mip-mapping with a filter stack. This is done in such a manner that initially a certain number of mip-mapping (resolution reducing) steps are performed before we switch to a filter stack. This way, the performance of the filtering as a whole can be controlled by the number of mip-mapping steps applied, as this reduces the amount of work while stack filtering. See Section 7 for an analysis of this parameter.

6 Accumulation

The data structure described thus far is very general; its applicability to soft shadows is described in Selgrad et al. [2014]. In this section, we describe how this data structure can be used to implement a feature-rich, real-time depth of field effect. Section 6.1 shows how we traverse our data structure to gather fragment information and Section 6.2 details how the targeted effect can be exploited in the filtering step (already described for the general case in Section 5).

6.1 Traversal

Using the previously generated and filtered levels, we can now compute the final output image by a composition of those. As each level of our hierarchy holds differently filtered version of the scene, we traverse all layers and accumulate those fragments for which the circle of confusion matches the filter size. This can be done efficiently by traversing and accumulating fragments from sub-intervals of all levels, guided by the circle of confusion. To avoid visible discontinuities in filter size between levels, we interpolate accordingly.

The size of the circle of confusion increases with the distance to the plane in focus as illustrated in Figure 5. A key component of our method is the traversal of the fragment lists with filter sizes matching the circle of confusion at the depth in question. Figure 5 shows the filter sizes for a given camera configuration in eye space, in relation to the circle of confusion. It can be seen that the eye space fragment size of the levels, $\mathrm{fs}_S(d)$, increases with distance to the camera, whereas the size of the circle of confusion first decreases, then increases again. This explains why we usually find particularly strong blurring in the near-field (i.e. in front of the focus plane).

To facilitate an efficient traversal, we compute the intersections of the circle of confusion size with the filter sizes in eye space. The intersection in the near-field, z_N, denotes the depth at which the level representing the next finer filter size should be used, until the next intersection. Consequently, the intersection in the far-field, z_F, determines the depth at which a filter size is no longer appropriate. See Figure 6 for an illustration of the level borders. Note, however, that according to this description, the intersections of the highest level can safely be disregarded as there is no level above it to transition from or to.

The circle of confusion $\mathrm{Coc}(z)$ and the filter size of level S $\mathrm{fs}_S(z)$ can be described as

$$\begin{aligned} \mathrm{Coc}(z) &= F|z - z_f| \\ \mathrm{fs}_S(z) &= 2^S G z \end{aligned}$$

where z_f is the distance to the focal plane, F a factor including the constant camera parameters (see Demers [2004] for more details) and G a factor describing the camera's field of view.

Based on this the borders of level S, $z_{N,S}$ and $z_{F,S}$, can be computed by solving $\mathrm{Coc}(z) = \mathrm{fs}_S(z)$, yielding

$$\begin{aligned} z_{N,S} &= \frac{z_f}{1 + (2^S G)/F}, \\ z_{F,S} &= \frac{z_f}{1 - (2^S G)/F}. \end{aligned}$$

To simplify the traversal kernel, we compute a table linking the border values to the level the traversal will transition into:

$$\mathrm{T} = \{(z_{N,n-1}, n-1), \ldots (z_{N,0}, 0), (z_{F,0}, 1), \ldots (z_{F,n-1}, n)\}.$$

This table is computed at the start of every frame and facilitates straightforward level selection in the traversal kernel. Our traversal is started with the most strongly filtered level n and accumulates fragments from it until the first level border is encountered, after which we switch to the specified level and continue with the traversal accordingly.

The resulting output color is computed by using alpha blending on the encountered fragments' color and opacity values in the same way as described for merging fragments in Section 5.1. Figure 7 shows the contribution of a number of levels to the final image.

6.2 Fragment List Truncation

While we generate the increasingly filtered levels of our hierarchy we exploit a very simple observation, which, as described in Section 7, impacts rendering performance. As can be seen in Figure 6, there are certain intervals at each level for which no fragment information will be required during its traversal. These intervals, for filter size 2^S, are $z < z_{N,S}$, $z_{N,S-1} < z < z_{F,S-1}$ and $z > z_{F,S}$, with the exception of the base level.

Figure 5 shows that the middle interval of a level, i.e. $z_{N,S-1} < z < z_{F,S-1}$, is, as described, neither used during traversal, nor required to construct the following level. Therefore, we skip this interval during our incremental filtering to produce shorter lists which

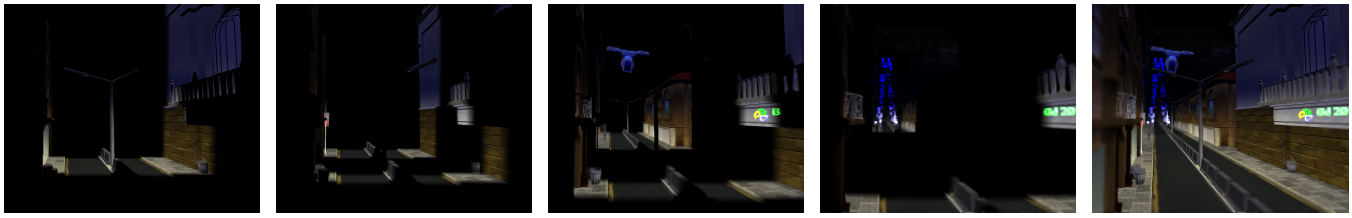

Figure 7: *Contribution of levels 0 through 3 (in reading order) to the final image (right).*

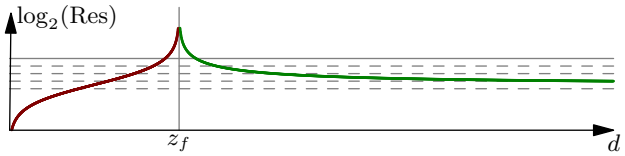

Figure 8: *Resolution required to capture the circle of confusion over a range of depth values. The red curve is z_N, starting from z_f the green curve is z_F. The solid gray line shows the base level resolution used, the dashed lines indicate progressively doubled filter sizes.*

are faster to traverse, but even more importantly (see Section 7) speed up filtering itself.

Filtering work can be reduced further by skipping the far-field of levels for which this interval will not be requested. Figure 8 shows that for a certain level, determined by the camera parameters, the fragment size catches up with the size of the circle of confusion and the required filter size plateaus.

7 Results

The focus of our evaluation is the performance of our method in the more complicated overlap case and to show that our algorithm runs in real-time on standard graphics hardware.

Table 1 lists detailed times for the rendering shown in Figure 1 where the focused text is seen through the out-of-focus railing. It can be seen that the rendering time is mostly dominated by the filtering process (due to a heavy use of stack filtering), and not by the traversal of all filtered levels. As described in Section 5.2, filtering work can be reduced by computing more mip-mapping layers before switching to a filter stack. The effect of adding a single mip-mapping layer is shown in the second column. A visual comparison of the results given in the second and third column of Table 1 is shown in Figure 3. We find that the difference in rendering time between these two is very meritable regarding the improvement in image quality, while rendering performance is much improved with respect to the setting of the first column. Further rendering times for the scenes shown in Figures 7, 9 and 10 are listed in Table 2.

To evaluate the influence of the number of fragments kept in the collection phase (see Section 4), as well as the impact of fragment list truncation (as described in Section 6.2) we compare rendering time for a scene with very high depth complexity (see Figure 10) in Table 3. We note that both parameters increase overall performance, where reducing the initial list length yields a 4.5% to 8.9% gain in performance while fragment list truncation increases performance by 15.3% up to 23.8%, resulting in a combined increase of 29.4%.

Regarding image quality, we compare our results to the approach by Lee et al. [2009] when run with sufficient lens samples. A comparison for the difficult overlap case is shown in Figure 11 where the

Filter configuration	1 / 3	2 / 2	4 / 0
Collect	4.14	4.14	4.14
Filter	10.19	5.38	4.26
Accumulate	5.76	5.45	4.68
Sum	20.09	14.97	13.08
Frames/sec	49.8	66.8	76.5

Table 1: *Detailed times (in ms) for the scene shown in Figure 1 with the text in focus, rendered at 720p. The filter configurations are, in order, one step using mip-mapping, followed by three stack filtering steps, two steps of both filters and lastly, using mip-mapping only. The results for the same scene with the railing in focus are comparable. Times were measured on a GeForce GTX 780.*

Filter configuration	Street	Sponza	Grass
Collect	6.64	6.33	11.22
Filter	13.87	8.51	11.42
Accumulate	1.81	3.21	3.36
Sum	22.32	17.46	26.00
Frames/sec	44.8	57.3	38.5

Table 2: *Times (in ms) for the scenes shown in Figures 7, 9 and 10. More detailed information on the times of the last column are presented in Table 3. Times were measured on a GeForce GTX 780 at 720p.*

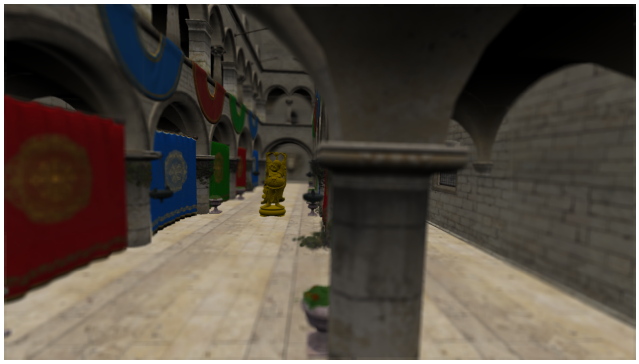

Figure 9: *Rendering times for the Sponza scene are listed in Table 2.*

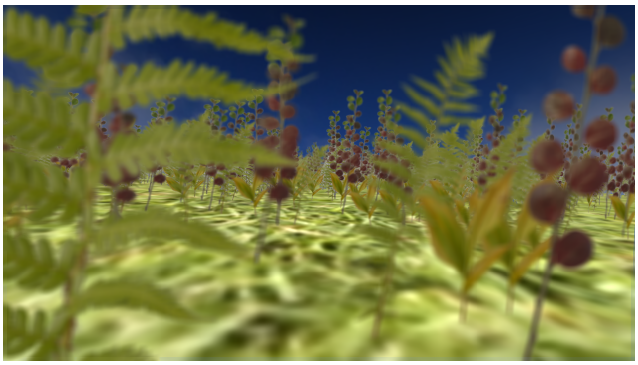

Figure 10: *A scene with high depth complexity. The plants are modeled by alpha-mapped quads, which our approach supports seamlessly. Detailed rendering times can be found in Table 3.*

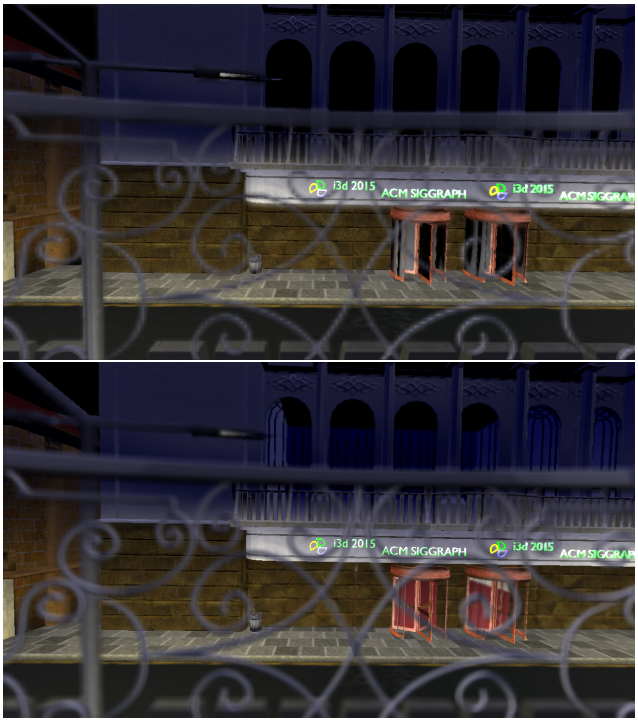

Figure 11: *A comparison of our approach (top, 49.8 frames/sec) to the reference depth of field blur according to Lee et al. [2009] (bottom, 128 samples per pixel, 15.8 frames/sec). Note how fine structures in the front reveal the in-focus text behind.*

	Full Lists		Truncated Lists	
	$N = 16$	$N = 4$	$N = 16$	$N = 4$
Collect	11.71	11.22	11.71	11.22
Filter	21.94	17.41	14.11	11.42
Accumulate	3.54	3.57	3.37	3.36
Sum	33.65	32.20	29.19	26.00
Frames/sec	29.7	31.1	34.3	38.5

Table 3: *Times (in ms) comparing the impact of fragment list truncation and reduction of N, the number of fragments collected in the first phase of our algorithm for the scene shown in Figure 10. Rendering resolution was 720p using one mip-mapping step followed by three stack filtering steps, measured on a GeForce GTX 780.*

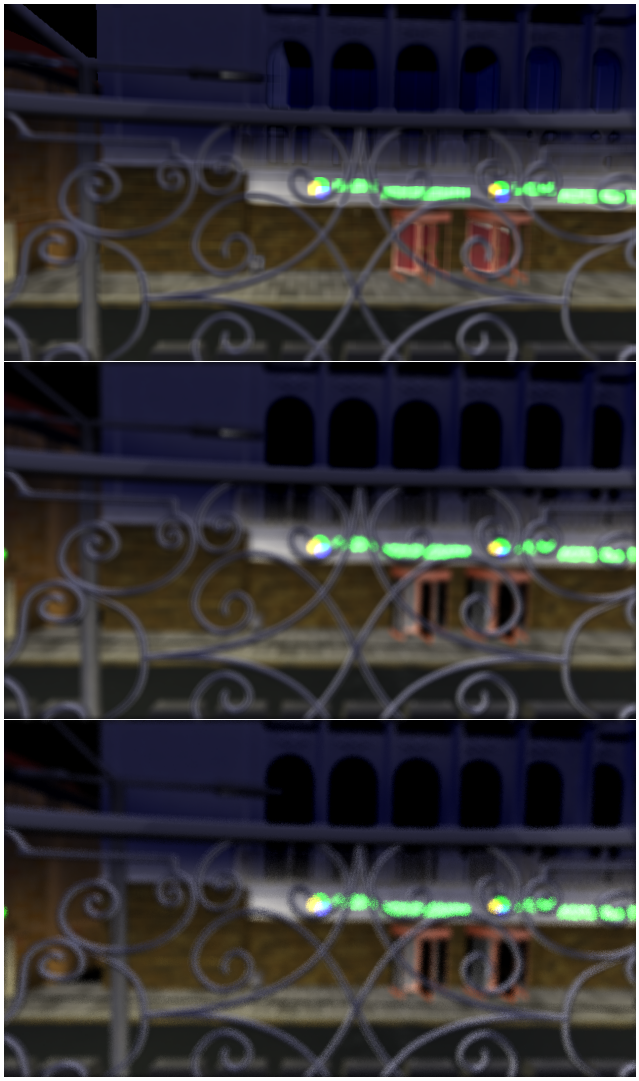

Figure 12: *We compare our approach (top, 52.6 frames/sec) to the reference depth of field blur by Lee et al. [2009] (center, 256 samples per pixel). The blurred neon text shines through the out-of-focus railing. With too few samples per pixel, ray tracing based methods exhibit point-sampling artifacts (bottom, 16 samples).*

text shines though the railing. Figure 12 shows how a bright background shines through thin, out-of-focus objects. As can be seen, our algorithm (top) computes images very close to the reference solution (center). Note that our approach does not suffer from point-sampling artifacts as ray tracing based methods do (e.g. [Lee et al. 2009]) as exemplified in Figure 12 (bottom). Our approach also seamlessly integrates semi-transparent surfaces such as the glass elements, which are not easily addressed by other (layered) methods.

The subtle artifact close to the blurred logo in Figure 12 is a consequence of the chosen blending mode (we generally apply additive blending to avoid light leaks at grazing angles). However, in this situation three depth layers (the railing, text area and wall) are combined and the opacites of the first two layers are slightly overestimated such that the farthest layer does not contribute anymore. In fact, blending with the over-operator removes the artifact, but exhibits problems in other cases. The artifact can also be removed by relaxing the ϵ parameter (see Section 5) so that the two farther layers are merged during filtering.

8 Conclusion

This paper presents an efficient data structure and its application for rendering a plausible depth of field effect in real-time, even with complicated configurations. We showed how our method approaches ray traced results, and that it avoids errors from point-sampling as encountered with such techniques when an insufficient amount of samples is used. Furthermore, our method is easily scalable by means of a set of efficient parameters that allow quality vs performance trade-offs.

We believe that further optimization of our method is possible, e.g. by adding (partial) binary search to the accumulation step, as well as by further, per-level post-processing of the generated filter levels.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive and thorough feedback, and gratefully acknowledge the generous funding by the German Research Foundation (GRK 1773).

References

BELCOUR, L., SOLER, C., SUBR, K., HOLZSCHUCH, N., AND DURAND, F. 2013. 5d covariance tracing for efficient defocus and motion blur. *ACM Trans. Graph. 32*, 3, 31:1–31:18.

DAMMERTZ, H., SEWTZ, D., HANIKA, J., AND LENSCH, H. 2010. Edge-avoiding a-trous wavelet transform for fast global illumination filtering. In *Proc. High Performance Graphics 2010*, 67–75.

DEMERS, J. 2004. Depth of field: A survey of techniques. In *GPU Gems*, R. Fernando, Ed. Pearson Higher Education.

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

HULLIN, M. B., HANIKA, J., AND HEIDRICH, W. 2012. Polynomial optics: A construction kit for efficient ray-tracing of lens systems. *Computer Graphics Forum 31*, 4, 1375–1383.

KASS, M., LEFOHN, A., AND OWENS, J. 2006. Interactive depth of field using simulated diffusion on a gpu. *Pixar Animation Studios Tech Report 2*, 1–8.

KOSLOFF, T. J., TAO, M. W., AND BARSKY, B. A. 2009. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. In *Proceedings of Graphics Interface 2009*, Canadian Information Processing Society, 39–46.

KRIVANEK, J., ZARA, J., AND BOUATOUCH, K. 2003. Fast depth of field rendering with surface splatting. In *Computer Graphics International, 2003. Proceedings*, IEEE, 196–201.

LEE, S., KIM, G. J., AND CHOI, S. 2008. Real-time depth-of-field rendering using point splatting on per-pixel layers. *Computer Graphics Forum 27*, 7, 1955–1962.

LEE, S., EISEMANN, E., AND SEIDEL, H.-P. 2009. Depth-of-field rendering with multiview synthesis. *ACM Trans. Graph. (Proc. of SIGGRAPH Asia) 28*, 5.

LEHTINEN, J., AILA, T., CHEN, J., LAINE, S., AND DURAND, F. 2011. Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4, 55:1–55:12.

LEI, K., AND HUGHES, J. F. 2013. Approximate depth of field effects using few samples per pixel. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, 119–128.

MCINTOSH, L., RIECKE, B. E., AND DIPAOLA, S. 2012. Efficiently simulating the bokeh of polygonal apertures in a post-process depth of field shader. *Comp. Graph. Forum 31*, 6, 1810–1822.

RIGUER, G., TATARCHUK, N., AND ISIDORO, J. R. 2003. Real-time depth of field simulation. In *ShaderX2: Shader Programming Tips and Tricks with DirectX 9.0*, W. Engel, Ed. Wordware, Plano, Texas.

SCHEDL, D., AND WIMMER, M. 2012. A layered depth-of-field method for solving partial occlusion. *Journal of WSCG 20*, 3 (6), 239–246.

SCHWARZ, M., AND STAMMINGER, M. 2008. Quality scalability of soft shadow mapping. In *Proc. Graphics Interface*, 147–154.

SELGRAD, K., DACHSBACHER, C., MEYER, Q., AND STAMMINGER, M. 2014. Filtering multi-layer shadow maps for accurate soft shadows. *Computer Graphics Forum*, doi: 10.1111/cgf.12506.

SENGUPTA, S., HARRIS, M., ZHANG, Y., AND OWENS, J. D. 2007. Scan primitives for GPU computing. In *Proc. Symposium on Graphics Hardware*, 97–106.

TZENG, S., PATNEY, A., DAVIDSON, A., EBEIDA, M. S., MITCHELL, S. A., AND OWENS, J. D. 2012. High-quality parallel depth-of-field using line samples. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, Eurographics Association, 23–31.

YANG, J. C., HENSLEY, J., GRÜN, H., AND THIBIEROZ, N. 2010. Real-time concurrent linked list construction on the GPU. *Computer Graphics Forum (Proc. EGSR) 29*, 4, 1297–1304.