Real-Time Lens Blur Effects and Focus Control

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

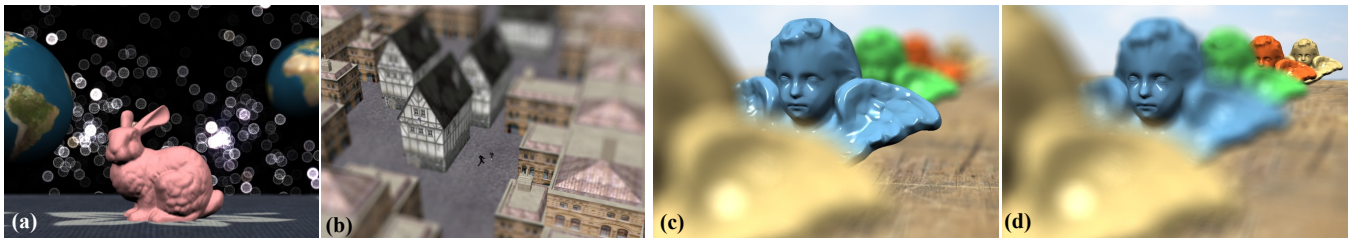

Figure 1: *Example images rendered in real time by our method. We achieve near-accurate depth-of-field effects, including lens aberrations (e.g., spherical aberration, (a)). The efficiency of our method makes it well-suited for artistic purposes and we support complex simulations like tilt-shift photography (b). Further, our system offers an intuitive control of depth of field and we extend the physical model (c) to achieve an expressive, yet convincing result (d) (here, the background statues stay focused).*

Abstract

We present a novel rendering system for defocus blur and lens effects. It supports physically-based rendering and outperforms previous approaches by involving a novel GPU-based tracing method. Our solution achieves more precision than competing real-time solutions and our results are mostly indistinguishable from offline rendering. Our method is also more general and can integrate advanced simulations, such as simple geometric lens models enabling various lens aberration effects. These latter is crucial for realism, but are often employed in artistic contexts, too. We show that available artistic lenses can be simulated by our method. In this spirit, our work introduces an intuitive control over depth-of-field effects. The physical basis is crucial as a starting point to enable new artistic renderings based on a generalized focal surface to emphasize particular elements in the scene while retaining a realistic look. Our real-time solution provides realistic, as well as plausible expressive results.

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

1 Introduction

Real cameras have an aperture through which light falls on an image plane containing receptors to register an image. For a sharp image, a small aperture is preferable, but then less light would hit these sensors and diffraction becomes an issue. Using a larger aperture in combination with a lens, 3D points at a certain *focal distance* are projected to a single point on the sensors, while other points map to a *circle of confusion* (COC) [Potmesil and Chakravarty 1981]. The latter effect leads to blur and only within a certain distance range, the *depth of field* (DOF), the image is crisp. DOF dramatically improves photorealism and depth perception [Mather 1996]. It also has become an important aspect for semantic purposes by drawing attention to certain elements while maintaining a realistic look.

This paper presents an efficient solution to approximate the image-capturing process by considering not only aperture but also aspects of the lens interaction itself. We approximate optical aberrations, which is a unique feature for real-time approaches. Sometimes considered an artifact, they are crucial for realism and allow us to reproduce many features often employed in artistic photos (Figure 1(a)). To achieve these effects our algorithm needs a certain generality that is also underlined by support for specialized configurations, e.g., tilt-shift photography where lens and image plane no longer align (Figure 1(b)). Our work enables us to interactively explore this large variety of possibilities and even outperforms competing methods. Our goal is to enable artists and designers to enhance, emphasize, and layout a scene or animation to better match their intentions. In this context, efficiency is an important aspect, but also controllability. We propose an intuitive interaction for physical and non-physical effects. In particular, we are concerned with focus, which is the most crucial component. Our interface enables even novice users to produce convincing results (Figure 1(d)).

Precisely, the contributions of our paper are as follows:

- An efficient algorithm for DOF and lens blur effects;
- An interactive and intuitive focus control system;
- A generalized method for *expressive* DOF rendering.

The rest of this paper is structured as follows. We review previous work (Section 2), before discussing our lens model and rendering algorithm (Section 3). Many optical aberrations come directly from the lens simulation and we motivate their use (Section 4). We illustrate our focus-control method and extend it to expressive rendering (Section 5). Finally, we discuss and present performance results (Section 6), before concluding (Section 7).

2 Previous Work

Many techniques exist to generate focal imagery in graphics, but the results were often of low quality or far from real time. The lack of high-quality interactive rendering methods might be one of the reasons why little work addressed focus manipulation, despite the increased awareness that previsualization and control are crucial for productions and instant feedback is central in this context. DOF is well-suited for abstraction: one can guide perception, enhance

*email: slee/hpseidel@mpi-inf.mpg.de, eisemann@telecom-paristech.fr

ACM Reference Format
Lee, S., Eisemann, E., Seidel, H. 2010. Real-Time Lens Blur Effects and Focus Control. *ACM Trans. Graph. 29,* 4, Article 65 (July 2010), 7 pages. DOI = 10.1145/1778765.1778802 http://doi.acm.org/10.1145/1778765.1778802.

and emphasize areas of interest, reduce the complexity of a scene (making it more understandable), or achieve dramatic appearances beyond physical boundaries. Previously, interactive solutions usually failed to reproduce realistic results, provided only a small range of parameters with limited possibilities, and generalizations lacked plausibility. We obtain a more general near-accurate simulation and a physical basis for artistic effects.

DOF Rendering Most real-time DOF methods postprocess a single image shot from the center of the lens. Filters are used to approximate COCs at each pixel [Rokita 1996; Riguer et al. 2003; Hensley et al. 2005; Bertalmío et al. 2004; Hammon 2007; Zhou et al. 2007; Kosloff et al. 2009; Lee et al. 2009b] leading to high performance but also artifacts such as *intensity leakage* (color bleeding in the background) or *depth discontinuity*. Anisotropic filtering [Bertalmío et al. 2004; Lee et al. 2009b] or spreading [Kosloff et al. 2009] can address this issue partially. Alternative scatter methods [Potmesil and Chakravarty 1981] transform pixels into COC sprites, but the necessary back-to-front sorting of the sprites makes it applicable mostly to offline rendering [Demers 2004].

In general, a single image misses information about surfaces hidden from the lens center which has a large impact on the final image. One way of hallucinating missing geometries is to split the single image into depth layers and extend the colors on each layer into the hidden areas [Barsky et al. 2002; Kass et al. 2006; Kraus and Strengert 2007; Kosloff and Barsky 2007]. However, such extrapolation does not reflect true scene information and can lead to overly-blurred and incorrect results, especially for out-of-focus foreground elements. Further, fusing layers via alpha blending is a coarse approximation. Such approaches work only for separate objects. Usually, discretization artifacts can only be mitigated with image processing [Barsky et al. 2005], information duplication [Kraus and Strengert 2007], or depth variation [Lee et al. 2008].

Multiview accumulation can treat visibility correctly, via ray tracing [Cook et al. 1984] or rasterization [Haeberli and Akeley 1990]. Since each scene rendering induces a heavy cost, these methods are usually inappropriate for real-time use. The accumulation buffer method [Haeberli and Akeley 1990] further forces several constraints on ray directions, making it difficult to extend it to general lens models. A recent method [Lee et al. 2009a] combines both elements. A single render step derives a layered representation on which an image-based raytracer is executed. The algorithm is efficient and achieves quality comparable to accurate methods. However, for our expressive scenario where highly anisotropic blur can occur, it shows reduced performance. Many layers are needed to bound errors, increasing memory consumption and making artifacts more common. Our approach works in the same spirit but is more efficient (even for standard lenses), is better adapted to the expressive purpose, scales better for smaller amounts of layers, and incorporates advanced lens effects that no other real-time solution provides.

User Control, Semantics, and Generalized DOF Creating images from 3D models imposes challenges. Instead of directly interacting with the appearance of an object, as is the case for painting, final results are defined indirectly via rendering parameters. With the increasing complexity of physical simulations, there is a need for intuitive controls over the parameters to ease the realization of desired results, especially for novice users. Further, there is a tendency to extend physical models while maintaining a plausible outcome.

Nowadays, intuitive interaction for lighting design is common and a survey can be found in [Patow and Pueyo 2003]. But other areas have been explored, including highlights and shadows [Poulin and Fournier 1992; Pellacini et al. 2002], camera placement [Gleicher and Witkin 1992], materials [Pellacini and Lawrence 2007], or indirect illumination [Schoeneman et al. 1993]. Usually, the physical simulation acts as an entry point for artists to refine the appearance. Similarly, we allow both; a simple interaction for defining physical and physically-inspired effects.

In the context of lens effects, little work exists. Kosara [2001] proposed a semantic DOF visualization. The work proves the potential of a controlled DOF, but is a 2D process, making results often unrealistic. Bousseau [Bousseau 2009] used filtering methods on lightfields to replace aperture effects. It abstracts filtering, but not the optical system, and offers no local control. Kosloff [2007] specifies blurring degrees for 3D points and uses heat diffusion, but the outcome lacks plausibility. Our approach delivers often-convincing results. We support almost-accurate physical simulations and address dynamic scenes. In particular, we enable a large variety of DOF blur crucial for artistic purposes. For instance, we support tilt-shift photography without costly rendering [Barsky and Pasztor 2004] and offer real-time feedback coupled with an intuitive control. This technique produces focal planes that are not perpendicular to the lens and has, recently, received much attention due to its strong defocus blur that can produce a miniature look (Figure 1(b)).

3 Realistic Real-Time Lens Blur

In this section, we explain our lens model and assumptions, before presenting the efficient rendering algorithm for DOF and lens effects.

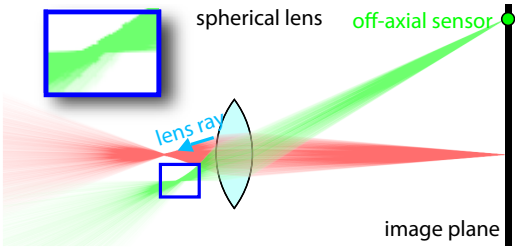

Figure 2: *Simulation of a spherical lens. For most lenses, rays do not converge exactly in a point, especially at off-axial sensors.*

Our Lens Model The purpose of a lens is to refocus ray bundles on the image plane. Depending on its application area, the lens' shape is designed accordingly [Smith 2004]. Designers often rely on path tracing to predict lens qualities by tracing rays from an image sensor through the lens system into the scene [Kolb et al. 1995].

Our real-time simulation uses geometric shape and refractive index of a lens. Consequently, many aberrations are captured from the lens' shape which influences the refraction when rays enter and exit the lens according to Snell's law. For spherical lenses, the outgoing rays can be computed accurately, involving a small constant overhead. Lens aberrations, such as spherical aberration and curvature of field (Section 4), arise naturally from this simulation, but have previously been neglected in real-time methods. Furthermore, we do not assume ray coherence in form of a perspective-projection center which many previous approaches required for efficiency. Our limitation is that we assume rays to refract exactly twice when traversing the lens and to travel along straight paths inside the lens (Figure 2). Hereby, we ignore diffraction and reflected rays.

3.1 Our Rendering Algorithm

Our algorithm works in two steps. First, we derive an image-based layered representation of the scene using a modified depth-peeling strategy. Second, for each sensor (pixel) we trace several rays. We compute the interaction with the aperture and then use a ray tracing to find scene intersections. The ray contributions are accumulated to generate the final image.

Layer Construction via Efficient Depth Peeling We avoid testing lens rays against an actual scene and, instead, derive a layered image-based representation via depth peeling [Everitt 2001] from the lens' center. Depth peeling is a multi-pass technique. Each pass *peels* off one layer of the scene; i.e., in the i^{th} pass, each pixel captures the i^{th}-nearest underlying surface by culling all geometry nearer than the z-buffer of the previous pass. The termination of the peeling is detected via occlusion queries.

A faster peeling exists [Liu et al. 2009]. However, to accelerate our ray tracing step, it is more crucial to reduce the number of layers. We use two important observations. First, layer pixels that cannot be reached by any lens rays do not need to be extracted. Second, a depth-peeled representation is point-sampled at the pixel centers, which leaves room for interpretation of the actual geometry.

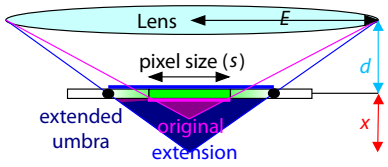

Figure 3: *We can use an offset during depth peeling to omit surfaces hidden behind already extracted pixels. Extended pixels lead to more occlusion by exploiting point-sampling ambiguities.*

Given a pixel P captured by a ray r through a pixel center, P blocks some region in space from all lens rays (Figure 3). If one thinks of the lens as a light source, this space corresponds to the shadow *umbra*. No lens ray can intersect any sample captured by r inside this umbra. Hence, during depth peeling, instead of culling against the depth of the previous pass, we offset the previous depth by the distance r travels inside the umbra.

Culling can be improved further by virtually extending a captured pixel to the neighboring pixel centers. It is important to notice that such an extension would not affect the depth peeling process (surfaces are captured exactly at the pixel centers) and, consequently, the standard depth peeling would deliver the same layers. Using these virtually increased umbra regions, the final offset x that we apply is given by: $x = \frac{d \cdot s}{E - s}$.

With our umbra method and a standard camera (FOVY=30°, dNear=1m, dFar=∞, lens radius = 9mm, resolution = 800×600), 10 layers are always enough (independently of the scene). In practice, 3-7 layers are common. Considering larger connected regions did not result in a performance gain. Our solution remains artifact-free, when assuming silhouette pixels to be extended in this way during our ray tracing.

Computing Lens Rays Our ray tracing starts on a sensor from where many rays are shot. These are blocked according to the aperture and transformed via the lens into a *lens ray* that is then tested against the scene layers.

The lens is defined by two height-field surfaces–one for each side–and we can combine the intersection test of the aperture and the first lens surface. At each intersection point, the ray is refracted according to the lens' refraction index. Alternatively, for algebraic surfaces, we can solve the intersections analytically; e.g., spherical lenses are common in reality due to their relatively cheap physical construction.

Efficient Intersection Test For the moment, let's assume a single depth layer and a lens ray to test for intersection against this layer. Naively, this involves stepping over all pixels in the layer's image

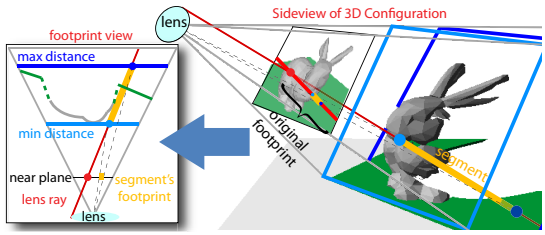

Figure 4: *Instead of searching along the entire ray, we can clamp the ray by min/max depth extents. The process can be repeated for the resulting segment.*

plane underneath the 2D projection of the ray which we call *footprint*. If the footprint is large, the intersection test is costly. We can reduce it by computing the minimum and maximum depths of the layer (e.g., via mipmapping). Intersections can only happen within this depth range, so we can clamp the original ray into these extents. The resulting 3D segment has a smaller footprint (Figure 4) and less pixels need consideration. Similarly, given the min/max depths underneath the new footprint, we can repeat the process to further narrow down the search region. After a few iterations, the remaining pixels are tested one by one to find an intersection.

Reducing the search region per ray is costly. Instead, we treat all lens rays in parallel. This implies two challenges, addressed hereafter: Deriving a bounding footprint for all lens rays in a depth interval and computing the min/max values in a footprint region.

Bounding the Footprint For a thin-lens model [Potmesil and Chakravarty 1981], the footprint of all rays for a given depth d is the *circle of confusion* (COC). To bound the footprint for a depth interval $[d_1, d_2]$, it is enough to take the maximum of the COCs at d_1 and d_2. Simple closed-form solutions [Potmesil and Chakravarty 1981] make the computation efficient.

For a geometric lens, apart for particular cases, closed form solutions are complex. Nonetheless, we deal with a finite number of lens rays and each ray can easily be clamped to a depth range. To bound the footprint of all rays for a given depth interval $[d_1, d_2]$, we intersect each ray with planes at distance $d1$ and $d2$. We collect the intersection points and compute a bounding quad in image space that we use as an approximate footprint. Given this bounding quad, we compute the underlying min/max depth values (as detailed hereafter) and repeat the process; we clamp all rays and compute a new bounding quad. Three iterations are a good trade-off between gain and cost of this step. Although it might sound expensive, our ray tracing is data-bound, leaving room for such arithmetic. The shown examples evaluate all lens rays, but 1/4 is sufficiently accurate in practice.

Computing Min/Max Values Given a footprint, we use N-buffers [Décoret 2005] to determine the minimum and maximum of the covered values. N-Buffers are a set of textures $\{T_i\}$ of identical resolution. T_0 is the original image and a pixel P in T_i contains the minimum and maximum value of T_0 inside a square of size $2^i \times 2^i$ around P. We cover the footprint rectangle using four texture lookups, corresponding to overlapping squares [Décoret 2005]. The hierarchical N-Buffer construction (T_{i+1} uses T_i) is fast, but N-Buffers are memory intensive. Our solution is to use a mipmap texture and an N-Buffer applied to a downsampled version (e.g., $1/4^{th}$ resolution) of the original layer. The memory gain and construction speedup correspond to a factor of four, while small regions can be sampled efficiently using the mipmap or the original image.

There is one catch. Some pixels, especially in latter layers, can be empty, do not capture any information, and need to be excluded during the min/max N-Buffer construction. To mark missing data,

we use the depth value zero. Before depth peeling, we clear the z-buffer to zero. During the peeling step, we exclude the output depth zero. Consequently, this value indicates that no data was output.

Multi-Layer Packing We accelerate multiple layer treatment by packing four depth values into a single RGBA texture directly after the peeling step. It allows us to scan four layers in parallel, similarly to [Policarpo and Oliveira 2006]. These layers share one N-Buffer, where T_0 is set to the per-pixel minimum and maximum of these layers. The intervals are still narrowed down quickly because lens rays are almost perpendicular to the image plane. Finally, we test all four depth values (recovered by a single lookup) simultaneously while stepping along the segment. Once the closest intersection is found, the corresponding color is retrieved.

No layer can be skipped because depth peeling does not order primitives globally, but it gives a local order in each pixel. Hence, once a pixel is empty, it remains empty for all following layers leading to large empty zones which are detected efficiently by the N-Buffer.

4 Optical Aberrations

Our geometric lens model captures many optical aberrations which are present in real cameras and particularly visible when the aperture is fully opened. Although manufacturers try to counterbalance aberrations by employing a lens set, their simulation is crucial for realistic and artistic effects (e.g., LensBaby™ exaggerates aberrations to provoke a certain appearance). We will review three cases (refer to [Smith 2004] for more examples) that we consider of interest due to their strong effects and relatively common usage in artistic shots.

Spherical Aberration In contrast to a theoretical thin lens, spherical lenses do not perfectly focus all sensor rays in a single focal point; usually, rays are more strongly bent on the lens periphery. Biconvex and aspheric lenses, like the human cornea, can reduce this effect. Visually, spherical aberration manifests in a general blur and discrepancy of sharpness and brightness of the image. This allows us to derive a softer appearance. Furthermore, halos appear around strong highlights, visible for Bokeh. It can be used to drive attention as well as to define a general mood (Figure 5(right)).

Curvature of Field (COF) COF projects a focal plane to a curved image. Rays at a large angle see the lens as if it had a smaller diameter but higher power. The image of offaxis points moves closer to the lens. This makes images clearly focused in the center but lose focus towards the boundary. Such curved image surfaces are common in many real lenses, especially telescopes. Compared to spherical aberration, the blur is more anisotropic and is often used to suggest the past, dreams, or still velocity.

Chromatic Aberration The refraction index of a lens usually depends on the wavelength of the incoming light. Often invisible, it can result in colored halos around objects (Figure 5(left)). While spherical aberration and COF are a direct consequence of our lens model, chromatic aberration needs to be simulated explicitly. We use an empirical equation proposed by Sellmeier [1871]:

$$n(\lambda) = \sqrt{1 + \frac{B_1\lambda^2}{\lambda^2 - C_1} + \frac{B_2\lambda^2}{\lambda^2 - C_2} + \frac{B_3\lambda^2}{\lambda^2 - C_3}} \quad ,$$

where B_i and C_i are material coefficients, λ the wavelength, and n the refractive index. This model, originally for a thin lens, allows us to benefit from large material databases. Physical (e.g., Borosilicate crown glass (BK7), a typical lens material) or non-physical results are possible (Figure 5). To maintain real-time performance, we do not employ a full spectral evaluation. Instead, we separately

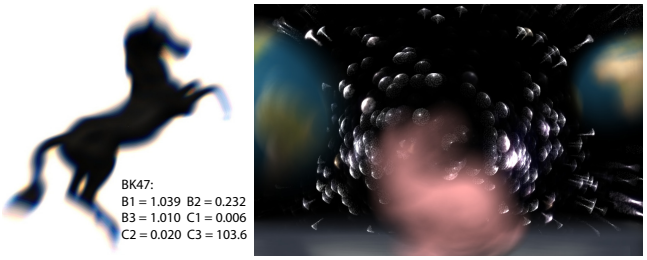

Figure 5: *Physically-based aberrations with our approach. Chromatic (left, for the BK47 glass) and spherical aberration (right).*

compute lens rays only for the three dominant wavelengths of the RGB channels, by assuming 650, 510, and 475 nm, respectively. Although not capturing the full spectrum, the approach usually leads to convincing results. Other recent work (e.g., scattering simulations in the human eye [Ritschel et al. 2009]) shows that such approximations can result in very faithful simulations. If needed, more wavelength samples could be used to improve the approximation.

5 Controlling Focus and Lens Effects

This section presents our algorithm to intuitively control lens blur. We present algorithms for physically-based lenses, but also extend DOF beyond the physical definition by allowing varying lens parameters for each image point. In this context, we also allow control over the previously presented aberrations for abstraction purposes.

User Interface As previously mentioned, controlled focus allows us to guide an observer to certain locations, emphasize objects, or create a special mood. For example, it might be of interest to always keep an object defocused in order not to reveal any details, while other elements of the scene stay constantly in focus. Changing the focal distance manually for each frame is a tedious process. In our interface, a user controls DOF at a high level by attaching attributes, like focused or defocused, directly to the scene and by keyframing them over time. A click on the screen defines a *focus point*. Internally, we store its barycentric coordinates and triangle to support animated geometry. For each focus point one can specify DOF parameters (most prominently, the amount of blur) and influence weights. Based on this input, the camera parameters are optimized to reflect the intended definitions for the current view. Per default, we exclude constraints outside the view frustum, but allow an artist to specify otherwise. We also increase the influence of nearer constraints to avoid ambiguities and use temporal interpolation to achieve temporal coherence.

5.1 Controlling Focus for Standard Lens Models

Here, we describe our intuitive focus control for common lenses.

Thin Lens Given a focal length F, the focal distance d_f is defined by the distance between image plane and lens: $u = Fd_f/(d_f - F)$ for $d_f > F$ [Potmesil and Chakravarty 1981]. We use a single focus point in the scene to let the user define the focal distance.

Spherical Thick Lens Spherical lenses are controlled similarly. The focal length can be computed via the lens maker's equation: $\frac{1}{F} \approx (n-1)\left(\frac{1}{R_1} - \frac{1}{R_2} + \frac{(n-1)t}{nR_1R_2}\right)$, where R_1 and R_2 (negative) are the lens radii, t the thickness, and n the refractive index.

Tilt-Shift Photography For *tilt-shift photography* a camera's image plane is tilted with respect to the lens, hereby tilting the focal

plane. The effects are interesting (Figure 1(b)), but the nonintuitive relation between tilt and focus can make the device difficult to operate, especially for animated scenes. On the contrary, we make the process simple to control by deriving a least-square focal plane from focus points. The focal plane is automatically transformed into an image plane tilt [Merklinger 1996]. Care is needed when the focal plane aligns with the view vector. We avoid this physical impossibility by limiting the plane normal.

5.2 Expressive Focus Control

Our system also allows for non-physically-based local parameter definitions, while standard lens models are restricted to global definitions. For artistic purposes, this is particularly interesting to enhance certain areas. For example, locality is important when different emphasis is to be put on objects residing at the same distance. The DOF itself is controlled via a focal surface that continuously interpolates the focal-depth constraints in screen space as well as the DOF extent in form of a single scalar value. Temporal smoothness is achieved again by interpolating the surface and DOF over time.

Focal Surface and DOF Interpolation The focal surface definition is based on a moving-least squares (MLS) solution. Each focal point defines kernel functions around it. In practice we use weights of α^1/d^β, where d is defined in terms of screen space distance for each focal point. For a given pixel we minimize the error function by weighting desired parameters at focus points according to these weight functions. The importance weight, α, gives finer control over the strength of the influence region of each focus point, since the best-fit surface is not necessarily identical to the designer's intention. β controls the range affected by the nearby control points; as β increases, local behavior becomes narrower. In the limit, discontinuities could be reintroduced, but $\beta = 2$–4 works well in practice. We experimented with different kernels and got comparable results. It is important to handle singularities at $d = 0$ to ensure a perfect interpolation of the control point itself. The surface is evaluated entirely on the GPU, enabling us to define per-pixel parameters.

Our system also supports focus points that indicate out-of-focus regions. A user simply specifies the offset with respect to a potential focal plane to achieve controlled defocus. In practice, this interaction is intuitive, but problems occur when such constraints overlap. In order to avoid problems, we reduce the influence of kernel functions according to their distance and slowly fade out their contribution when they become invisible. As before, this can be counteracted and an artist can keyframe the behavior differently. Our system delivers immediate feedback which eases such interaction.

5.3 Expressive Aberration Effects

We could influence aberrations via the lens shape, but, in practice, this requires much expertise and is far from an intuitive framework. Instead, for expressive rendering, we decided to use a less accurate simulation with more intuitive control parameters.

Spherical aberration is caused by varying distances that light travels inside the lens. We often observe that rays are bent more strongly on the periphery of the lens than its center. An intuitively controllable approximation is to modify the refractive power via a spline definition. Further, spherical aberration is most valuable for Bokeh effects and we provide the possibility to directly weight rays differently to influence the shape of Bokeh.

Curvature of field appears because the focal plane is associated to a curved image plane. In other words, if the sensors were on this image plane, the resulting rendering of an object on the focal plane would be sharp. To simulate this effect, we can let the user define an offset surface for the image plane. Again, we use a smooth MLS interpolation and write the resulting deformation in a buffer. As all points on this surface share the same focal distance, the lens appears to have a different focal length for each point. We adapt rays traversing this surface accordingly to take this effect into account.

Figure 6: *Examples of expressive aberration effects. Spherical aberration for Bokeh highlights (left) and curvature of field combined with a tilt-shift lens focusing on the table (right).*

Chromatic aberration can be controlled via varying refraction indices for the RGB channels which is intuitive and remains unchanged.

We found that these definitions allow us to easily control lens effects. Figure 6 and the video demonstrate how parameters can be intuitively adjusted to achieve complex appearances.

6 Results and Discussion

Performance Our system was implemented using DX10 on a Pentium Core2Quad 2.83 GHz machine. We evaluated performance in terms of geometry and GPU classes (Table 1). Despite the high scene complexity (the smallest has 98K triangles), our method resulted in high performance ranging between 100 and 30 Hz on a standard graphics card (NVIDIA GeForce GTX285). The tests with different GPU classes indicate a roughly linear scalability of our algorithm with respect to the number of stream processors (GTX285/8800). Our approach benefits from texture bandwidth, illustrated by the 9500GT which has a proportionally lower bandwidth. We compared our results to a recent algorithm (MS) [Lee et al. 2009a] and a reference method (REF) [Haeberli and Akeley 1990]. Both simulate a thin-lens model which is the only available choice for a real-time comparison. MS required 16 layers to avoid artifacts in the scenes, but we added timings for 8 layers as well. For the latter, artifacts were readily visible, while our method remained artifact-free using only 4 layers. Further, we achieve sub-linear performance with respect to the number of layers because N-Buffers perform very efficient skipping *and* intersection tests for the sparse later layers. Typically, for 8 layers, the ray tracing cost of the second four represents only 20–30%. In the same way, ray tracing generally scales slightly sub-linearly with respect to the sampling rate. For the geometric lens model a roughly constant overhead ($\approx 10\,ms$ for 100 rays) is added because we need to compute the lens rays for each pixel, which is more complex than for a thin lens.

To investigate the behavior of our method when facing a high rendering costs, we used a 0.5M and 1.5M triangle scene and applied different complex shaders to achieve the same base-rendering cost of 25 ms to produce a rendering at a resolution of 1024×768 pixels. For the 1.5M scene, computing the layered scene representation and N-Buffers took 74 ms. This process is less expensive than a naive 4-time rendering due to early-empty-pixel skipping in later layers during the depth peeling. The ray tracing is independent of the base-rendering cost and stayed cheap (4/7 ms for 32/64 rays). The timings for the 0.5M scene were similar (preprocessing: 75 ms, ray tracing:5/8 ms). This observation indicates that deferred shading (a common technique for games that extracts surface properties before applying the shaders) would reduce the overall cost drastically.

Our approach is well-suited for deferred shading because our mem-

	PRE (Depth peeling+N-Buffer) + RT (raytracing) = TOTAL			MS (8/16 layers)	REF	PSNR (db)/SSIM
	285GTX (240 SP)	8800GTX (128 SP)	9500GT (32 SP)	285GTX	285GTX	
Town (98K tri.)	4 + 6 = 10	6 + 8 = 14	16 + 79 = 95	15/26 (1.5/2.6)	125 (12.5)	34.8/0.98
Angels (407K tri.)	7 + 17 = 24	11 + 29 = 40	20 + 144 = 164	75/122 (3.1/5.1)	213 (8.9)	36.9/0.97
Dragon (935K tri.)	16 + 15 = 31	24 + 33 = 57	40 + 169 = 209	81/125 (2.6/4.0)	641 (20.7)	35.0/0.98

Table 1: *Comparison of the rendering cost (in ms) of our system (100 lens rays, 4 layers, and 800×600) with a competitor (MS) [Lee et al. 2009a] and the reference (REF) [Haeberli and Akeley 1990] with speedup factors given in parentheses. SP denotes the number of stream processors in GPUs. A quality measure is given with respect to REF by evaluating PSNR and SSIM [Wang et al. 2004].*

ory consumption is much lower compared to [Lee et al. 2009a]. In practice, our modified peeling resulted at most in seven layers for realistic scenes (less than half of depth peeling stopping at 5% pixel occupancy). For 1024×768 pixels, we use only 16–32 MB instead of 72–92 MB [Lee et al. 2009a] (details are provided in the supplementary material).

Quality The image quality of our method is similar to reference results shown by the signal-to-noise ratio (PSNR) and the structural similarity (SSIM) [Wang et al. 2004] (see Table 1 and Figure 7).

Aliasing and ghosting arise when employing an insufficient number of rays. We can improve the perceived quality using varying ray-sampling patterns (jittering) for each pixel [Lee et al. 2009a]. This solution is incompatible with accumulation buffer methods. Figure 7 illustrates the quality/sampling-rate tradeoff. For 750 samples, our solution still achieved interactive rates for a million-triangle scene and gave results indistinguishable to 10^5-rays even for a 100^2-blur kernel. Such a kernel is much larger than what most real-time approaches apply. Figures in the paper used 100 samples.

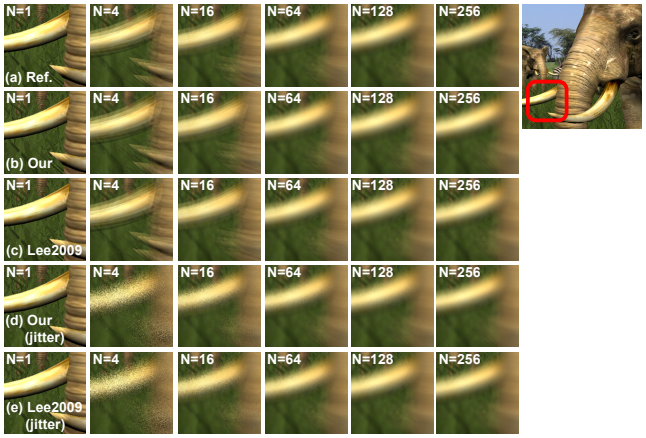

Figure 7: *Influence of the number of lens rays (N)*

To show the expressive spectrum of our system, we mimicked specialized lenses like LensBaby™ or Spiratone Portragon. The effectiveness of our interface is best demonstrated in the accompanying video. In particular, instant feedback is crucial to judge where supplementary constraints are needed. Dramatic effects can be achieved in a few clicks and the results look convincing, even under animation.

Discussion One key aspect of our method is the extended-umbra peeling which exploits the sample-based ambiguity between image and geometric representations. It is efficient for nearby and far-away pixels. For far-away pixels, the relative parallax is small and the resulting umbra is large in world space. E.g., the pixel's world-space extent can be large enough with respect to the lens, that no lens ray can access any further layer behind it. In the extreme case of a lens that degenerates to a point, our method derives a single depth layer. Independent of the scene, or the far plane, our extended-umbra peeling results always in a finite maximal amount of layers.

Our layers allow us to produce an artifact-free rendering in the sense that rays do not miss surfaces. Such property cannot be assured by other peeling methods capturing a constant amount of layers. The major difference to an accumulation buffer method (that moves the camera) is that its surface sampling changes for each view. In practice, this difference is subtle because for strong out-of-focus effect, the blur kernel is large. These advantages make our strategy also interesting for offline computation in previsualization systems.

Combining our approach with single-pass depth peeling (e.g., the AMD DirectX11 example of order-independent transparency) is an interesting avenue for future work. However, currently, such single-pass implementations report limited performance (e.g., 30 fps on an AMD HD5670 card), even inferior to our total execution time for 1M triangles. Further, such methods store all layers, and thus, the raytracing is costly. Our findings could be used as a postprocess to reduce the surface samples. Nevertheless, this does not limit the initial memory consumption. Figure 8 depicts a scene where single pixels contain up to 100 layers and our solution produces the correct result using only four layers. The high memory consumption of single-pass peeling also hinders deferred shading and incoming fragments are shaded directly.

In comparison to the single-pass decomposition of [Lee et al. 2009a], our depth peeling can be slower, but our ray tracing is more cache-efficient, treats multiple layers in parallel, and uses less and more-predictable arithmetic operations. In consequence, we achieve equivalent or better quality with a strong speedup. We also do not miss hidden fragments within layers and avoid temporal popping for geometry crossing many layers. Further, four layers are usually enough for near-accurate results. This is a very important feature and is not valid for uniform decompositions [Lee et al. 2009a].

Figure 8: *Our method handles complex cases and visibility relations*

7 Conclusion and Future Work

We presented a novel real-time lens-blur rendering system exceeding previous methods in performance and quality, and introduced the first real-time system that manages many lens aberration effects. The latter are an important component for artistic photography and, consequently, we presented a simple system to control depth-of-field blur. We further extended this control to non-physical (but plausible) results yet give large possibilities to designers.

The future work will be focused on extending our user interface to further facilitate interaction (e.g., painting metaphors). Also, in

theory, temporal interpolation for animation is already possible but there is a new possibility to integrate an event-driven control. Such a system could be useful in a game to trigger particular focus elements and to test how well users can be guided via that focus design.

We thank the reviewers, L. Baboud, and K. Subr for support and the Stanford 3D Scanning Repository, AIM@Shape, the3DStudio.com, TurboSquid, and Keenan Crane for the 3D models. This work was partly funded by the Intel Visual Computing Institute at Saarland University.

References

BARSKY, B. A., AND PASZTOR, E. 2004. Rendering skewed plane of sharp focus and associated depth of field. In *SIGGRAPH Sketches*, 92.

BARSKY, B., BARGTEIL, A., GARCIA, D., AND KLEIN, S. 2002. Introducing vision-realistic rendering. In *Proc. Eurographics Rendering Workshop*, 26–28.

BARSKY, B., TOBIAS, M., CHU, D., AND HORN, D. 2005. Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques. *Graphical Models 67*, 584–599.

BERTALMÍO, M., FORT, P., AND SÁNCHEZ-CRESPO, D. 2004. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proc. 3DPVT*, 767–773.

BOUSSEAU, A., 2009. Non-linear aperture for stylized depth of field. SIGGRAPH 2009 - Technical talk.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. *Computer Graphics 18*, 3, 137–145.

DÉCORET, X. 2005. N-buffers for efficient depth map query. In *Proc. Eurographics*, 393–400.

DEMERS, J. 2004. Depth of field: A survey of techniques. In *GPU Gems*, R. Fernando, Ed. Addison-Wesley, ch. 23, 375–390.

EVERITT, C. 2001. Interactive order-independent transparency. *White paper, nVIDIA 2*, 6, 7.

GLEICHER, M., AND WITKIN, A. P. 1992. Through-the-lens camera control. In *SIGGRAPH*, 331–340.

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.

HENSLEY, J., SCHEUERMANN, T., COOMBE, G., SINGH, M., AND LASTRA, A. 2005. Fast summed-area table generation and its applications. *Computer Graphics Forum 24*, 3, 547–556.

KASS, M., LEFOHN, A., AND OWENS, J. 2006. Interactive depth of field using simulated diffusion on a GPU. Tech. rep., Pixar.

KOLB, C., MITCHELL, D., AND HANRAHAN, P. 1995. A realistic camera model for computer graphics. In *Proc. ACM SIGGRAPH*, 317–324.

KOSARA, R., MIKSCH, S., AND HAUSER, H. 2001. Semantic depth of field. In *Proc. IEEE Information Visualization*, 97–104.

KOSLOFF, T., AND BARSKY, B. 2007. An algorithm for rendering generalized depth of field effects based on simulated heat diffusion. In *Proc. ICCSA*, 1124–1140.

KOSLOFF, T., TAO, M., AND BARSKY, B. 2009. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. In *Proc. Graphics Interface*, 39–46.

KRAUS, M., AND STRENGERT, M. 2007. Depth-of-field rendering by pyramidal image processing. In *Proc. Eurographics*, 645–654.

LEE, S., KIM, G. J., AND CHOI, S. 2008. Real-time depth-of-field rendering using 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. 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.

LIU, F., HUANG, M. C., LIU, X. H., AND WU, E. H. 2009. Single pass depth peeling via cuda rasterizer. In *SIGGRAPH Talks*.

MATHER, G. 1996. Image blur as a pictorial depth cue. *Biological Sciences 263*, 1367, 169–172.

MERKLINGER, H. M. 1996. *FOCUSING the VIEW CAMERA*. Nova Scotia.

PATOW, G., AND PUEYO, X. 2003. A survey of inverse rendering problems. *Computer Graphics Forum 22*, 4, 663–687.

PELLACINI, F., AND LAWRENCE, J. 2007. AppWand: editing measured materials using appearance-driven optimization. *ACM Trans. Graph. 26*, 3, 54.

PELLACINI, F., TOLE, P., AND GREENBERG, D. P. 2002. A user interface for interactive cinematic shadow design. In *Proc. ACM SIGGRAPH*, 563–566.

POLICARPO, F., AND OLIVEIRA, M. M. 2006. Relief mapping of non-height-field surface details. In *Proc. I3D*, 55–62.

POTMESIL, M., AND CHAKRAVARTY, I. 1981. A lens and aperture camera model for synthetic image generation. *Proc. ACM SIGGRAPH 15*, 3, 297–305.

POULIN, P., AND FOURNIER, A. 1992. Lights from highlights and shadows. In *SI3D*, 31–38.

RIGUER, G., TATARCHUK, N., AND ISIDORO, J. 2003. Real-time depth of field simulation. In *ShaderX*^{2}, W. F. Engel, Ed. Wordware, ch. 4, 529–556.

RITSCHEL, T., IHRKE, M., FRISVAD, J. R., COPPENS, J., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2009. Temporal Glare: Real-Time Dynamic Simulation of the Scattering in the Human Eye. In *Proc. Eurographics*.

ROKITA, P. 1996. Generating depth-of-field effects in virtual reality applications. *IEEE Comp. Graph. and its App. 16*, 2, 18–21.

SCHOENEMAN, C., DORSEY, J., SMITS, B., ARVO, J., AND GREENBERG, D. 1993. Painting with light. In *Proc. ACM SIGGRAPH*, 143–146.

SELLMEIER, W. 1871. Zur Erklärung der abnormen Farbenfolge im Spectrum einiger Substanzen. *Annalen der Physik und Chemie 219*, 272–282.

SMITH, W. 2004. *Modern Lens Design*. McGraw-Hill Professional.

WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Proc. 13*, 4, 600–612.

ZHOU, T., CHEN, J., AND PULLEN, M. 2007. Accurate depth of field simulation in real time. *Computer Graphics Forum 26*, 1.