Eurographics Symposium on Rendering 2013
Nicolas Holzschuch and Szymon Rusinkiewicz
(Guest Editors)

Volume 32 (2013), Number 4

Practical Real-Time Lens-Flare Rendering

Sungkil Lee
Sungkyunkwan University, South Korea

Elmar Eisemann
Delft University of Technology, Netherlands

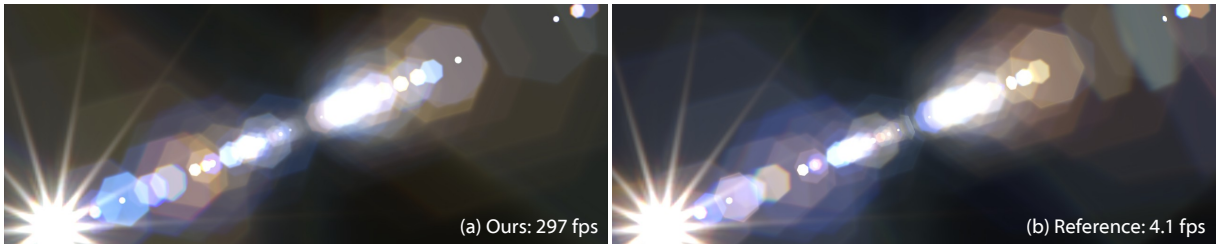

Figure 1: *Our lens-flare rendering algorithm compared to a reference state-of-the-art solution [HESL11]. Our method significantly outperforms previous approaches, while quality remains comparable and is often acceptable for real-time purposes.*

Abstract

We present a practical real-time approach for rendering lens-flare effects. While previous work employed costly ray tracing or complex polynomial expressions, we present a coarser, but also significantly faster solution. Our method is based on a first-order approximation of the ray transfer in an optical system, which allows us to derive a matrix that maps lens flare-producing light rays directly to the sensor. The resulting approach is easy to implement and produces physically-plausible images at high framerates on standard off-the-shelf graphics hardware.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms

1. Introduction

Lens flare is the result of unwanted light reflections in an optical system. While lenses are supposed to refract an incoming light ray, reflections can occur, which lead to deviations from the intended light trajectory. A single reflection would send a ray back towards the entrance plane, but an even number of reflections can redirect rays towards the sensor. If such rays carry sufficient energy to the sensor, they might produce a so-called *ghost*. As these reflected rays still cross the aperture, ghosts often share its shape. While originally an artifact, lens flare is often used for artistic purposes and is a good indicator for bright light sources [Pix08].

A typical method to compute lens flare is to rely on ray tracing to simulate light paths, assuming typically up to two or four reflections. This approach achieves the highest quality, but can become costly for complex optical systems. Recently, a more approximate but faster solution was proposed based on *polynomials* [HHH12]. While this model can handle complex optical behavior, the algorithm is still too slow for real-time applications (e.g., up to 4 secs. for complex lens models).

Real-time approaches in games and virtual reality are usually based on *texture sprites*; artists need to place and design ghosts by hand and have to rely on (and develop) heuristics to achieve a convincing behavior. Although efficient at run time, the creation process is time-consuming, cumbersome, and difficult. Further, the solution misses a physical basis and often appears unrealistic. Our work follows sprite-based methods and shares their efficiency, but we derive a more accurate physically-based solution for a given optical system.

We present a linear approximation of lens-flare effects based on a *paraxial assumption* [Smi07]. In comparison to the previous near-accurate solutions, our first-order approximation does not exhibit nonlinear deformation behavior. However, it is much more efficient, making it a perfect candidate for real-time applications. Further, our model still captures many physical characteristics of the optical system including ghost location, size, and color. Our contributions include:

- a matrix-system formulation for lens-flare effects
- an efficient rendering scheme
- an analysis to optimize rendering performance.

2. Previous Work

This section describes previous work on lens-flare rendering. Refer to Hullin et al. [HESL11] for a more detailed review.

For realistic image synthesis, camera properties are often mentioned as important factors [KMH95, LES10, SDHL11]. Physically-based real-time solutions exist for some effects, such as depth of field [LES10], but are missing for lens flare.

An early interactive approach relied on texture sprites that were animated using a hand-tweaked displacement function along a line through the screen's center [Kil00]. Real-time follow-up work mostly concentrated on introducing additional heuristically-steered features, such as size and opacity [Kin01], brightness variations [Mau01, Sek04], or light streaks [Oat04]. This trend is also reflected by the lens-flare plugins [Als09], which allow a user to tweak certain parameters, but they are not related to any underlying camera model. Our approach fills this gap and delivers a fast rendering scheme derived from actual optical systems.

Except for very recent work [HESL11, HHH12], lens systems were only considered in costly offline approaches using path tracing [Cha07], photon mapping [Kes08], or in lens-design solutions that do not aim at image generation [Toc07]. Computing lens flare via bundle tracing [HESL11] leads to high quality results, but the method quickly becomes costly for complex lens systems and involves a significant preprocessing time. The more efficient (but also coarser) solution based on a polynomial representation [HHH12] leads to higher framerates. Nonetheless, the necessary dense random sampling of the entrance pupil (the first lens surface that is exposed to the exterior light) makes it hard to reach interactive performance for realistic lens systems. The quality of our method is lower when compared to these competitors, but it makes a significant performance leap.

3. Our Approach

Lens flare is caused by light rays that pass through the optical system, but deviate from their intended path due to reflections at lens surfaces. In this section, we concentrate on this ray propagation, from which we will derive an efficient method for lens-flare rendering and add advanced features, such as anti-reflective coatings (Sec. 3.2). Building upon this model, we then derive several acceleration techniques (Sec. 3.3).

3.1. Ray Transfer Model

Optical Interfaces Lens manufacturers usually describe an optical system as a set of algebraically-defined interfaces. These interfaces are usually flat (aperture and sensor) or spherical (because spherical lenses are common and easy to produce). Each spherical interface is defined by a signed radius (convexity/concavity) and a thickness measured along the optical axis. Further, these descriptions usually contain information about materials (types of glass or air), refractive indices, and heights with respect to the optical axis. We will use such optical designs available from patents or specialized sources [Smi05] as an input (see Table 1 for an example).

Table 1: *An algebraic specification of an optical system (Heliar Tronnier; USP 2645156). Our method uses these values directly, but ignores height for the interfaces other than entrance pupil and iris aperture.*

radius	thickness	material	refractive index	sa (heights)
30.810	7.700	LAKN7	1.652	14.5
-89.350	1.850	F5	1.603	14.5
580.380	3.520	air		14.5
-80.630	1.850	BAF9	1.643	12.3
28.340	4.180	air		12.0
	3.000	air (iris aperture)		11.6
	1.850	LF5	1.581	12.3
32.190	7.270	LAK13	1.694	12.3
-52.990	81.857	air		12.3

Matrix Optics Our model is based on a first-order paraxial approximation [Smi07]. Although this approximation induces non-trivial errors for large angles above 10 degrees, it captures many important properties of lens systems. Further, it allows us to handle the ray's interaction with an optical interface via a matrix multiplication. By concatenating these matrices, one can describe even complex lens-system traversals in constant time, which has an important impact on rendering performance. Another advantage is that the analysis of an optical system is simple and can even be executed on the fly, while previous methods involved heavy preprocessing to reduce rendering times [HESL11].

The paraxial assumption refers to a small-angle approximation, which proved useful for the analysis of optical systems [Smi07]. It relies on a first-order Maclaurin expansion, which implies that $\sin\theta \approx \theta$, $\tan\theta \approx \theta$, and $\cos\theta \approx 1$. Furthermore, it is assumed that all incoming rays are meridional rays (a ray that is contained in a plane which includes the optical axis). Via the symmetry of the interfaces and the paraxial assumption, it is possible to apply a 2D analysis and all interfaces become parallel lines with corresponding interaction matrices.

An optical ray $\mathbf{r}$, whose staring point is at distance z to the sensor plane, is represented by a 2D vector $\mathbf{r} = [\, r\ \theta\,]^\mathsf{T}$, where r is a signed offset of its origin from the optical axis, and θ the angle (positive on an upward angle) between the ray and the optical axis (see Figure 2).

The interaction of a ray with a single interface i can be described via a 2×2 ray transfer matrix $\mathbf{M}_i$, often dubbed as ABCD matrix. For our purpose, the matrices for refraction, reflection, and travel in a homogeneous media between two

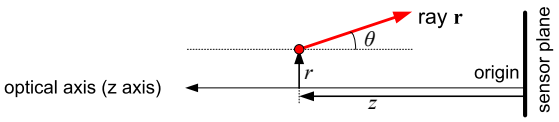

Figure 2: *2D-vector notation of a ray* $\mathbf{r} = [r\ \theta]^\mathsf{T}$.

Table 2: *Common ray transfer matrices [PPP07], where d_i is a positive displacement to the next interface at interface i, n_1 and n_2 the refractive indices at interface i and R_i its lens radius ($R > 0$/$R = \infty$ for convex/flat interfaces).*

Optical component	Ray transfer matrix
Translation ($\mathbf{T}_i$)	$\begin{bmatrix} 1 & d_i \\ 0 & 1 \end{bmatrix}$
Refraction at spherical dielectric interface ($\mathbf{R}_i$)	$\begin{bmatrix} 1 & 0 \\ \frac{n_1 - n_2}{n_2 R_i} & \frac{n_1}{n_2} \end{bmatrix}$
Reflection from a spherical mirror ($\mathbf{L}_i$)	$\begin{bmatrix} 1 & 0 \\ \frac{2}{R_i} & 1 \end{bmatrix}$

interfaces are needed. We denote $\mathbf{T}_i$ the translation matrix with displacement d_i, $\mathbf{R}_i$ the refraction, and $\mathbf{L}_i$ the reflection matrices at interface i. The matrices can be found in Table 2. They involve only values given by the lens description (refraction indices and radii) and can thus be computed for the optical system. It is important to point out that, although interfaces are represented as lines (because the model considers meridional rays), the interaction matrix is derived for curved (spherical) interfaces. This choice leads to a better approximation (including magnification) and explains the presence of radii in the formulae.

3.2. Lens-Flare Rendering

As each interface an interaction corresponds to a matrix, the entire trajectory of a ray through the optical system can be defined by a series of matrix multiplications. While standard system matrices only describe the transmission of an intended light path towards the sensor, we are interested in pursuing rays that lead to lens flares. As indicated before, these rays are characterized by an *even number* of reflections on their path; rays with an odd number of reflections eventually end up at the entrance pupil. Following previous work [HESL11] (and also because each reflection leads to a significant energy loss), we only consider paths with two reflections. Hence, there is only a finite set of matrices, which we refer to as *flare matrices*. The matrices describe possible lens-flare trajectories in the system, i.e., each matrix corresponds to one ghost. It is noteworthy that an inverse ray trajectory (from the first reflection back to the second reflection) can be described by multiplying the inverses of the corresponding ray transfer matrices (except for the translations). Figure 3 (top) shows an example of a flare path with two reflections (at I_4 and I_2).

Flare-Quad Mapping The above ray propagation model could be used directly to define realistic displacement functions for sprite-based approaches [Kil00]; due to the linearity of our system, the entrance pupil could be bound with four vertices corresponding to four rays, whose direction is given by the light direction. These vertices form a *flare quad*, which can be mapped directly to the sensor using the flare matrices. The projected quad, after applying a flare matrix, can then

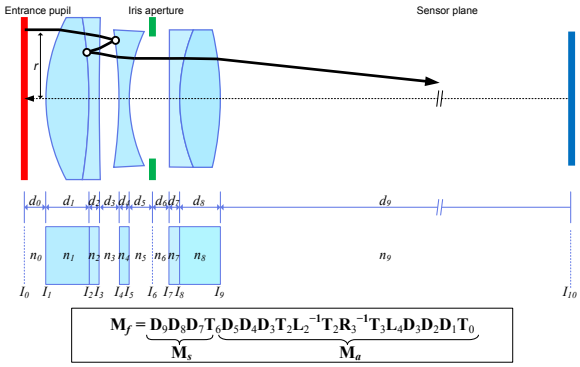

Figure 3: *Flare matrix formulation for Heliar Tronnier. I, d, and n indicate optical interfaces, distances between optical interfaces, and refractive indices, respectively.*

be used as a sprite that represents one ghost. Blending the solutions for all flare matrices leads to the final lens-flare image. In what follows, we will describe techniques to improve realism, while Section 3.3 will cover acceleration techniques.

Flare Shape Instead of using an ad hoc sprite, the shape of a ghost actually comes mostly from the optical system's aperture. In order to integrate the aperture in our model, we represent each flare matrix $\mathbf{M}_f$ as the product of two matrices; $\mathbf{M}_a$ (from the entrance pupil to the iris aperture) and $\mathbf{M}_s$ (from the iris aperture to the sensor plane) with $\mathbf{M}_f = \mathbf{M}_s\mathbf{M}_a$. Figure 3 (bottom) illustrates such a decomposition. We define $\mathbf{D}_n := \mathbf{T}_n\mathbf{R}_n$ for a concise notation.

With these two separate matrices, one can map rays from the entrance pupil to the aperture and then to the sensor. When representing the aperture as a texture (Figure 4), a simple lookup after applying $\mathbf{M}_a$ can determine if a ray is blocked. For a flare quad, it is thus enough to apply the aperture texture using texture coordinates that stem from the flare quad's mapping via $\mathbf{M}_a$ on the aperture plane. The texture-mapped quad is then projected on the sensor using $\mathbf{M}_s$.

It is important to notice that, in the decomposition, the two reflections are assumed to occur on the same side of the aperture because, otherwise, these light rays would need to pass the aperture three times. While it would be possible to consider a decomposition into four matrices, we decided to ignore these flare matrices completely, as it is unlikely to happen because the aperture is typically small. Similarly, in a full model, light rays could also hit the tube enclosing the lenses inside the optical system. We ignore these rare cases as well, with one exception; the entrance pupil's shape should not be neglected (Figure 5), as it refines the coarse initial flare quad. It can again be represented as a texture (or, if circular, by a threshold on the distance to the optical axis). We handle this situation—similarly to the aperture—by clipping those pixels of the projected flare quad on the sensor, whose initial position lies outside the entrance pupil.

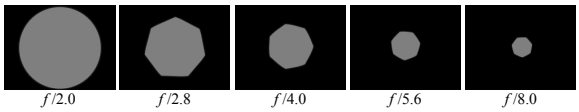

Figure 4: *Possible textures for an iris aperture.*

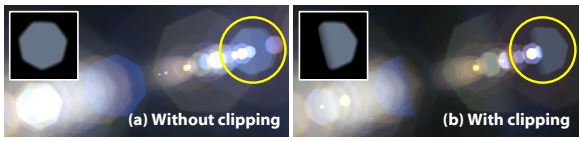

Figure 5: *Effect of clipping against the entrance pupil.*

Intensity and Color Intensities of ghosts can vary, because their size can differ drastically. Nonetheless, it is easy to scale the intensity correctly using the projected flare quad size.

A more complex subject is the handling of per-flare coloration, resulting from anti-reflective coating [HESL11]. The exact color can be computed using Fresnel anti-reflective (AR) coating; refer to the supplementary material of Hullin et al. [HESL11] for implementation details. However, our linear model will only support a single color per ghost, because we do not perform full ray tracing and per-ray reflections cannot be computed accurately. Nonetheless, even when tracing just a single ray in the center of each flare quad, it often captures the ghost's characteristic color well (Figure 1).

To improve accuracy of our simulation, we can also consider wavelength-dependent dispersion. For our purposes, we rely on a standard RGB decomposition (assuming $\lambda_R = 650$, $\lambda_G = 510$, $\lambda_B = 475$ nm) and compute wavelength-dependent refractive indices (and flare matrices) using Sellmeier's approximation [Sel71]. Our method is then applied for each color channel separately, which leads to acceptable results. In Section 3.3, we turn the three-pass into a single-pass method.

Finally, a direct light can be approximated using far-field diffraction (Fraunhofer approximation), which is equivalent to the scaled Fourier transform of the aperture. We precompute these starburst patterns with respect to all the wavelengths, following Hullin et al. [HESL11].

3.3. Accelerations

The previously-described algorithm, summarized in Figure 6, already leads to an important speedup compared to previous work, but it can still be improved. Here, we describe how to reduce rasterization cost, how to handle multi-spectral rendering efficiently, and other acceleration techniques.

Rasterization Our previous approach maps the entire flare quad to the sensor. While this projected flare quad can cover a large area on the sensor, most of the rays it represents might have been blocked by the aperture. Although these pixels do not contribute to the final result, they are still produced and

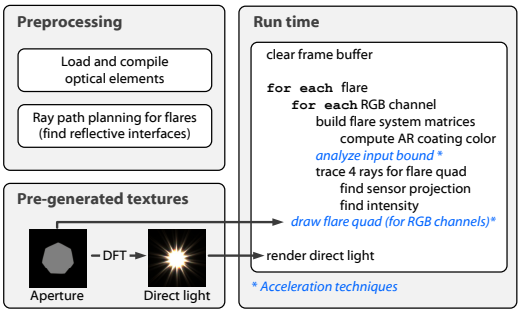

Figure 6: *Overview of the rendering pipeline of our system.*

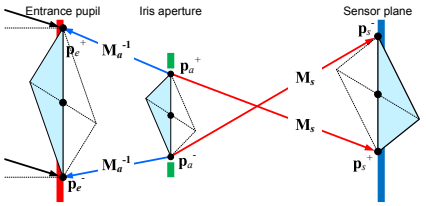

Figure 7: *Bounding of an input quad at the entrance pupil (blue arrows) using the two endpoints at the iris aperture and their mapping onto the sensor (red arrows). Note that there is an inversion between the iris aperture and the sensor plane.*

then discarded. Instead, it would be more efficient to choose a flare quad, which more closely encompasses the rays on the entrance plane that will actually pass through the aperture. Mapping this smaller flare quad will significantly reduce the cost of the subsequent rasterization.

In nonlinear optical systems, it is non-trivial to find the input rays that reach the aperture. Consequently, a costly preprocessing of several hours is needed, involving exhaustive ray tracing [HESL11]. In contrast, in our model, a matrix analysis allows us to find a tightly bounding flare quad.

Let's denote a ray on the entrance plane $\mathbf{r}_e = [r_e\ \theta_e]^\intercal$ and on the aperture plane $\mathbf{r}_a = [r_a\ \theta_a]^\intercal$. Denoting $\mathbf{M}_a$ the matrix that maps rays from the entrance to the aperture plane, we have $\mathbf{r}_e = \mathbf{M}_a^{-1}\mathbf{r}_a$. As the angle θ_e is given by the light direction, we can solve the equation for r_e for any given r_a:

$$r_e := (r_a - M_a^{12}\theta_e)/M_a^{11}, \tag{1}$$

where M_a^{11} and M_a^{12} represent the first and second element of $\mathbf{M}_a$. By choosing r_a values that tightly bound the aperture and using Equation (1), we can compute a restricted flare quad on the entrance pupil (Figure 7). This simple strategy led to important speedups ranging from 1.7 to 5.2, due to the reduced rasterization costs.

The intensity of the flare is initialized with the ratio of the flare quad with respect to the entrance pupil, which is finally modulated again by its projected size.

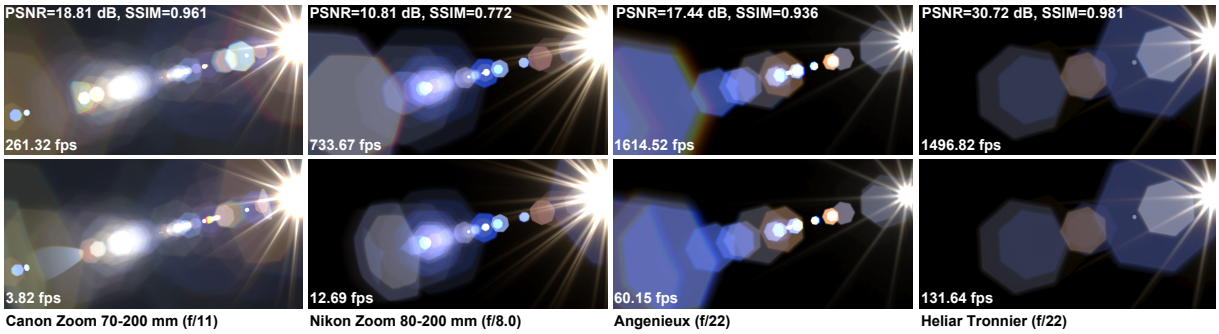

Figure 8: *Lens-flare rendering for various optical systems at a display resolution of* 1280×720*. The upper row was generated with our system, the lower row with Hullin et al. [HESL11]. The effect of a single ray evaluation of the AR coating and the paraxial approximation is particularly visible for the cyan flare in the upper right corner of the Canon example. Nonetheless, most flares are well represented and the overall look is reproduced at much higher framerates.*

Multi-Spectrum Acceleration The previously-mentioned three-pass solution, considering RGB channels independently, triples the rendering cost. For a single-pass solution, we first find a conservative flare quad that encompasses the offsets of all different channels. In most cases, such a quad is only 10–20% larger than for a single wavelength. Then, the texture coordinates for different wavelengths can be expressed with respect to a primary texture coordinate (e.g., the green channel). Then, the rasterization step of the projected flare quad can treat all three wavelengths at once. This simple strategy leads to a significant acceleration and doubles performance, by reducing rasterization and blending costs.

Intensity-Based Culling In many cases, more than half of the flare quads are rendered with low intensity and have little impact on the result. We can safely skip these flare quad entirely with negligible quality loss. Further, these cases often coincide with a large projected size and, hence, significant rasterization costs, making this choice doubly beneficial.

4. Results

Rendering Performance Our system was implemented in Direct3D 10 on an Intel Core i7 3.4 GHz with an NVIDIA GTX 680 graphics card. Performance measurements for different lens systems can be found in Figure 8. We compared our results to a recent state-of-the-art algorithm [HESL11] (with 512×512 as a grid resolution for their bundle tracing).

For all of the optical systems, our approach reached high framerates ranging from 261 Hz (for a Canon zoom lens of high complexity) to 1615 Hz (for Angenieux of medium complexity), making it a suitable solution for real-time applications (e.g., games or virtual reality). The speedup factors compared to the reference range from around 10 to 70.

As previous real-time approaches [Kin01, Mau01, Sek04, Oat04], our solution also relies on texture sprites. Hence, its cost is comparable to these solutions and the overall performance mostly depends on the number of flare quads. Further, reducing rasterization costs, is important and leads to a 2 times speedup for simpler lenses (Heliar Tronnier), and 5 times for complex systems (Nikon Zoom). We did not make use of flare culling for the comparison, which would have lead to an additional more than 30% speedup (see Figure 9).

Figure 9: *Culling low intensity flares can boost performance drastically (34% speedup here) with low visual impact.*

Image Quality Overall, our solution achieves a comparable appearance with respect to the reference renderings. For mostly linear systems such as Heliar Tronnier, our solution matches the appearance almost perfectly. However, more complex systems with nonlinear deformations can result in non-trivial differences. This is a natural consequence of our linear model with the paraxial assumption, which lacks the support for higher-order ray transfer. A comparison to the reference images and quantitative differences (for 8-bit LDR RGB images—hereby, avoiding a bias due to brightness peaks) in terms of the signal-to-noise ratio (PSNR) and structural similarity (SSIM) [WBSS04] is shown in Figure 8.

5. Discussion and Limitations

The main approximation of our method is the paraxial assumption. It well captures the basic geometric appearance (e.g., size and position of ghosts). It represents a minimal form of the polynomial approximations [HHH12], hence, leading to a constant-time ray propagation. Nonetheless, complex optical behaviors such as deformation, topological changes, and aberration are not handled correctly (Figure 10), leading to the main differences with respect to the reference.

We performed an experiment with a pre-recorded 3D look-up

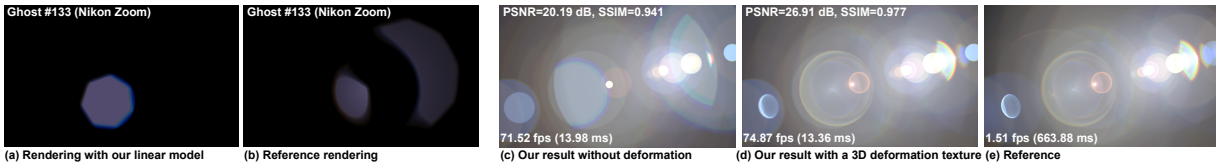

Figure 10: *Nonlinear deformations. A strongly distorted ghost (b) and our linear result (a). An extreme case; using our linear model (c), our 3D texture solution (d), and the reference (e).*

texture to determine the shape of the nonlinear flares based on the current light angle. Usually, less than 10 % of the lens-flare elements behave strongly nonlinearly—here, 24 out of 312. Even in extreme cases (Figure 10), the image quality is improved (SSIM=0.97) via the deformation texture. Its construction takes a few minutes, when relying on efficient reference renderers. However, this approach has certain drawbacks. It consumes additional memory; e.g., for a spatial resolution of 512×512 and an angular resolution of 128, 96 MB are required per ghost. Further, integrating several flare elements in the same texture makes it impossible to correctly orient the ghosts, when the light rotates around the optical axis. The disadvantages outweigh the quality gain, which is why this technique is not used elsewhere in the paper or video. An analytical deformation model could be an alternative, but, currently, it seems out of reach and remains future work.

For acceleration purposes, we only considered two reflections on the same side of the aperture. Our solution could be extended to general reflections, but we consider the restriction useful, as each reflection reduces the transported energy significantly. We also neglected energy loss from transmission and absorptance; which are usually marginal (e.g., up to a few percent of losses) and do not justify additional overheads.

6. Conclusion

We presented a practical real-time lens-flare approach that delivers physically-plausible results based on a given description of an optical system. Our approach builds upon a first-order paraxial approximation that allows us to describe light propagation as a matrix multiplication. Hereby, our solution not only accelerates online computations, but also simplifies the analysis of the lens system, which enables us to introduce further accelerations. Our solution delivers competitive results when compared to state-of-the-art solutions, and exceeds the quality of previous real-time approaches.

Acknowledgments This work was supported by the Basic Science, Mid-career, and Global Frontier (on *Human-centered Interaction for Coexistence*) R&D programs through the NRF grants funded by the Korea Government (MSIP) (No. 2011-0014015, 2012R1A2A2A01045719, and 2012M3A6A3055695) and the Intel Visual Computing Institute at Saarland University.

References

[Als09] ALSPACH T.: Vector-based representation of a lens flare. US Patent 7,526,417, 2009. 2

[Cha07] CHAUMOND J.: Realistic camera - lens flares. http://graphics.stanford.edu/wikis/cs348b-07/JulienChaumond/FinalProject, 2007. 2

[HESL11] HULLIN M., EISEMANN E., SEIDEL H.-P., LEE S.: Physically-Based Real-Time Lens Flare Rendering. *ACM Transactions on Graphics 30*, 4 (2011), 108:1–9. 1, 2, 3, 4, 5

[HHH12] HULLIN M. B., HANIKA J., HEIDRICH W.: Polynomial Optics: A construction kit for efficient ray-tracing of lens systems. *Computer Graphics Forum (Proc. EGSR'12) 31*, 4 (2012). 1, 2, 5

[Kes08] KESHMIRIAN A.: *A physically-based approach for lens flare simulation*. Master's thesis, University of California, San Diego, 2008. 2

[Kil00] KILGARD J.: Fast OpenGL-rendering of lens flares. http://www.opengl.org/resources/features/KilgardTechniques/LensFlare/, 2000. 2, 3

[Kin01] KING Y.: 2d lens flare. In *Game Programming Gems 2*, DeLoura M., (Ed.). Charles River Media, 2001, pp. 515–518. 2, 5

[KMH95] KOLB C., MITCHELL D., HANRAHAN P.: A realistic camera model for computer graphics. In *Proc. ACM SIGGRAPH'95* (1995), pp. 317–324. 2

[LES10] LEE S., EISEMANN E., SEIDEL H.-P.: Real-Time Lens Blur Effects and Focus Control. *ACM Transactions on Graphics (Proc. ACM SIGGRAPH'10) 29*, 4 (2010), 65:1–7. 2

[Mau01] MAUGHAN C.: Texture masking for faster lens flare. In *Game Programming Gems 2*, DeLoura M., (Ed.). Charles River Media, 2001, pp. 474–480. 2, 5

[Oat04] OAT C.: A steerable streak filter. In *Shader X3*, Engel W., (Ed.). Charles River Media, 2004, pp. 341–348. 2, 5

[Pix08] PIXAR: The imperfect lens: Creating the look of Wall-E. Wall-E Three-DVD Box, 2008. 1

[PPP07] PEDROTTI F. L., PEDROTTI L. M., PEDROTTI L. S.: *Introduction to Optics*. Pearson, 2007. 3

[SDHL11] STEINERT B., DAMMERTZ H., HANIKA J., LENSCH H. P. A.: General spectral camera lens simulation. *Computer Graphics Forum 30*, 6 (2011), 1643–1654. 2

[Sek04] SEKULIC D.: Efficient occlusion queries. In *GPU Gems*, Fernando R., (Ed.). Addison-Wesley, 2004, pp. 487–503. 2, 5

[Sel71] SELLMEIER W.: Zur Erklärung der abnormen Farbenfolge im Spectrum einiger Substanzen. *Annalen der Physik und Chemie 219* (1871), 272–282. 4

[Smi05] SMITH W. J.: *Modern Lens Design*. McGraw-Hill, 2005. 2

[Smi07] SMITH W. J.: *Modern optical engineering: the design of optical systems*. McGraw-Hill, 2007. 1, 2

[Toc07] TOCCI M.: Quantifying Veiling Glare. http://www.zemax.com/kb/articles/192/1, 2007. 2

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Proc. 13*, 4 (2004), 600–612. 5