Physically-Based Real-Time Lens Flare Rendering

Matthias Hullin[1] Elmar Eisemann[2,3] Hans-Peter Seidel[1,3] Sungkil Lee[4,1]

[1] MPI Informatik [2] Télécom ParisTech [3] Saarland University [4] Sungkyunkwan University

Figure 1: *Complex lens flare generated by a Canon zoom lens. Left: reference photos. Right: renderings generated using our technique at comparable settings. Even with many unknowns in the lens design and scene composition, as well as manufacturing tolerances in the real lens, the renderings closely reproduce the "personality" of the flare.*

Abstract

Lens flare is caused by light passing through a photographic lens system in an unintended way. Often considered a degrading artifact, it has become a crucial component for realistic imagery and an artistic means that can even lead to an increased perceived brightness. So far, only costly offline processes allowed for convincing simulations of the complex light interactions. In this paper, we present a novel method to interactively compute physically-plausible flare renderings for photographic lenses. The underlying model covers many components that are important for realism, such as imperfections, chromatic and geometric lens aberrations, and antireflective lens coatings. Various acceleration strategies allow for a performance/quality tradeoff, making our technique applicable both in real-time applications and in high-quality production rendering. We further outline artistic extensions to our system.

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

Keywords: Lens flare, Real-time rendering

Contact to authors:
{hullin|hpseidel}@mpi-inf.mpg.de
elmar.eisemann@telecom-paristech.fr
sungkil@skku.edu (Corresponding author)

ACM Reference Format
Hullin, M., Eisemann, E., Seidel, H., Lee, S. 2011. Physically-Based Real-Time Lens Flare Rendering. *ACM Trans. Graph. 30,* 4, Article 108 (July 2011), 9 pages. DOI = 10.1145/1964921.1965003 http://doi.acm.org/10.1145/1964921.1965003.

1 Introduction

Lens flare is an effect caused by light passing through a photographic lens in any other way than the one intended by design—most importantly through interreflection between optical elements (ghosting). Flare becomes most prominent when a small number of very bright lights is present in a scene. In traditional photography and cinematography, lens flare is considered a degrading artifact and therefore undesired. Among the measures to reduce stray light in an optical system are optimized barrel designs, anti-reflective coatings, and lens hoods.

On the other hand, flare-like effects are often used deliberately to suggest the presence of very bright light sources, hence increasing the perceived realism. In fact, nowadays the use of lens flare is every bit as popular in games as it is in image and video editing. For the production of computer-generated movies, great effort has been taken to model cinema lenses with all their physical flaws and limitations [Pixar 2008].

The problem of rendering lens flare has been approached from two ends. A very simple and efficient, but not quite accurate, technique is the use of static textures (starbursts, circles, and rings) that move according to the position of the light source, and are composited additively to the base image. Flares generated from texture billboards can look convincing in many situations, yet they fail to capture the intricate dynamics and variations of real lens flare.

At the other end of the scale, sophisticated techniques have been demonstrated that involve ray or path tracing through a virtual lens with all of its optical elements. The results are near-accurate but very costly to compute, with typical rendering times in the order of several hours per frame on a current desktop computer. Furthermore, many samples end up being blocked in the lens system, which wastes much of the computation time and leads to slow convergence. Also, the solution only holds within the limits of geometric optics. Some phenomena encountered in real lens flares, however, are caused by wave-optical effects. Integrating them into a ray-optical framework is by no means trivial and further increases the computational cost.

By combining sparse ray tracing and rasterization, our rendering technique can simulate lens flares of complex lens designs, achieving a high degree of realism at interactive frame rates. Chromatic and geometric lens aberrations are reproduced naturally and can be extended by more advanced effects.

Precisely, we make the following contributions:

- a model for realistic lens flare;
- an efficient algorithm that allows a fine-tuned tradeoff between quality and efficiency;
- plausible approximations for difficult-to-handle imperfections.

The latter also represents a means for stylization. We show that expressive lens flare is a natural extension of our work.

2 Previous Work

Computer graphics research often focuses on the simulation of light exchange and interaction inside a virtual environment. While such computations can deliver physically-plausible imagery, a certain lack of realism remains where simplified camera models fall short of their real counterparts. Many effects (e.g., depth of field or motion blur) are crucial components for realistic image synthesis and many researchers underlined the need to focus more closely on camera specificities [Kolb et al. 1995; Lee et al. 2010; Steinert et al. 2011].

More faithful camera or lens-system simulation also covers stray light which adds a significant amount of realism to the rendering. It can in part be caused by dust or imperfections in the lens system, but the most prominent features of lens flare originate from internal reflections (ghosting) [Kingslake 1992].

What makes the simulation of lens flare attractive is the observation that humans are trained to interpret the presence of flare and veiling glare as an indication of extreme brightness. This perceptual effect can be used to seemingly exceed the physical boundaries of a display device [Ritschel et al. 2009], leading to its strong use in, e.g., movies [Pixar 2008], or recent games [Wenzel 2005].

Previous interactive techniques relied on significant approximations. Kilgard [2000] suggested the use of texture sprites that are blended into the framebuffer and arranged on a line through the screen center, following an *ad hoc* displacement function. King [2001] varied sprite size and opacity depending on the angle between light source and camera. Maughan [2001] added a brightness variation that can also be controlled depending on the number of visible pixels of an area light [Sekulic 2004]. Oat [2004] concentrated on light streaks that are added using a steerable filter. [Alspach 2009] described lens flare as a set of vector shapes such as “halo”, “rays” or “rings” that are generated according to user-specified statistics. In none of these cases, an underlying camera or lens model was considered.

In other situations, more accurate simulations are needed, e.g., when compositing virtual and realistic content, when designing lens systems, or when predicting the appearance of a scene through a lens system. Previous high-quality approximations [Chaumond 2007; Keshmirian 2008] relied on path tracing or photon mapping. While such approaches deliver theoretically a high quality, several aspects; such as spectral (e.g., chromatic aberration or lens coating), diffraction effects, or aperture shape, are usually ignored. Furthermore, the visual quality for short computation times can be insufficient, making interaction (e.g., zooming) impossible.

Simulation of wave-optical effects (in particular, diffraction and interference) is usually considered out of reach for graphics applications. High-end optical design tools such as ZEMAX or Code V allow for the computation of point spread functions and even coarse predictions of stray light including diffraction effects [Tocci 2007; Perrin 2004]. Notwithstanding their physical accuracy, they offer very general solutions and hence are not optimized for efficient high-quality flare rendering. In a purely ray-based framework, [Oh et al. 2010] showed how diffraction-like effects can be emulated for simple regular structures for which the light field transforms can be expressed analytically. In this work, we approximate similar effects for general aperture shapes as a preprocessing step, using Fourier transforms.

Our rendering scheme is physically motivated, yet runs at interactive to real-time performance. Based on a ray-tracing approach, the technique does not only consider individual rays, but exploit the strong correlation of rays within a light bundle. Radiant flux is treated in a way similar to beam or pencil tracing (for a comprehensive literature overview, see e.g. [Ernst et al. 2005]), but without the need for adaptive refinement.

Further, our solution can be adapted to exaggerate or replace physical components. Its initial faithfulness ensures that the resulting imagery keeps a convincing and plausible appearance even after applying significant artistic tweaks.

3 Model of the Optical System

Description of optical system	Lens design	Materials	Aperture	AR coatings
Light transport model	Dielectric reflection/transmission	Diffraction	Scaling params	Bounce orders

Figure 2: *Building bricks of our model. Each element allows to trade physical realism against artistic expression, or detail against performance.*

In this section, we discuss various aspects of an optical system and describe how we mathematically represent them. Depending on the requirements of the application, some effects can be skipped to simplify the model and increase the performance. The following should be considered building bricks that can either be modeled as accurately as desired, exaggerated, or altered in an artistically desired way. Figure 2 illustrates the components of our system and gives an overview over our model of the optical system, as well as the light transport, and implementation (Section 4).

Geometry Light propagation is governed by light transmission through, and reflection at a set of lens surfaces and characteristic planes (entrance, aperturer, and sensor plane). In our examples, we follow the definitions of photographic lenses from [Smith 2005], as well as the patent describing a lens we have at hand [Ogawa 1996]. The geometric model is realized as a set of algebraically defined surfaces, i.e., spheres and planes. Note that our technique does not impose particular geometry or lens materials. Further, rules of good optical design are not always needed to achieve attractive flare.

Figure 3: *Single blade (left) and a polygonal iris (right)*

Iris Aperture The aperture (also called diaphragm, iris, or stop) consists of mechanical blades that control the size of a pupil by rotating into place. When the aperture is fully open, they are hidden in the lens barrel, resulting in a circular cross-section. “Stopping down” the aperture leads to a polygonal contour defined by number, shape, and position of the blades. We recreate this mechanism (Figure 3) and store the resulting mask in a texture.

Optical Media and Dispersion In terms of optical media, we constrain ourselves to perfect dielectrics with a real-valued refractive index. All optical glasses are dispersive media, i.e., the refractive index n is a function of the wavelength of light, λ. We follow Sellmeier's empirical approximation [Sellmeier 1871] to describe the dispersion of optical glasses:

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

where $B_{\{1,2,3\}}$ and $C_{\{1,2,3\}}$ are material constants that can be obtained from manufacturer databases, e.g. [Schott AG 2011], or other sources [Polyanskiy 2010].

3.1 Reflection and Transmission: Fresnel Equations

Every time a ray of light hits an interface between two media, a part of it is reflected, and the rest transmitted. It is the reflected part that gives rise to ghosting artifacts, which we seek to simulate. For smooth surfaces, the relative amounts follow Fresnel's equations, with the resulting ray directions according to the *law of reflection* and Snell's law, respectively [Hecht 2001].

The Fresnel equations provide different transmission and reflection coefficients for different states of polarization. For unpolarized light propagating from medium 1 to medium 2 (with refractive indices n_i and angles with respect to the normal θ_i), the overall reflectivity R and transmissivity T of a surface can be expressed as

$$R = \frac{1}{2}\left(\frac{n_1\cos\theta_1 - n_2\cos\theta_2}{n_1\cos\theta_1 + n_2\cos\theta_2}\right)^2 + \frac{1}{2}\left(\frac{n_1\cos\theta_2 - n_2\cos\theta_1}{n_1\cos\theta_2 + n_2\cos\theta_1}\right)^2$$

and $\quad T = 1 - R.$

The treatment of polarization is crucial for the inner workings of an anti-reflective coating (see below). Since Fresnel reflection or transmission are partially polarizing, sequences of light-surface interaction would ideally keep track of a light ray's polarization state. However, since the marginal benefit would not justify the expenses, our current implementation of the ray tracer does not propagate polarization throughout the optical system. At each optical surface, the incident light is assumed to be s- and p-polarized (see [Hecht 2001] for a definition) to equal parts.

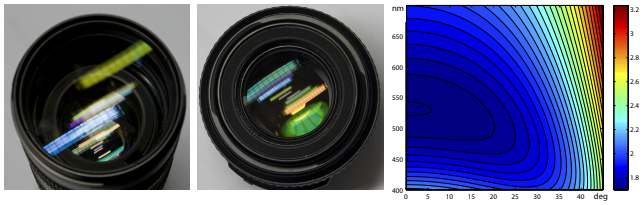

Figure 4: *Lens coatings make reflections from optical interfaces inside the lens barrel appear colored. Left: Canon EF 70-200mm f/2.8L. Middle: Canon EF 100mm f/2.8 USM macro. Right: Net reflectivity in % of a quarter-wave coating designed for 532 nm light at normal incidence ($n_{\text{glass}} = 1.5$, $n_{\text{coating}} = 1.38$, $d = 96.4\,\text{nm}$).*

Anti-Reflection Coatings In an attempt to minimize reflections, optical surfaces often feature antireflective coatings. They consist of layers of clear materials with different refractive index. Light waves that are reflected at different interfaces are superimposed and interfere with each other. In particular, if two reflections have opposite phase and identical amplitude, they cancel each other out, reducing the net reflectivity of the surface. The parameters of the multi-layer coatings used for high-end lenses are well-kept secrets of the manufacturers.

Even the best available coatings are not perfect. Their residual reflectivity is a function of wavelength and angle, $R(\lambda, \theta)$. A look into a real lens reveals that different interfaces reflect white light in different colors, suggesting that they are all coated differently (Figure 4).

Without the resources to reverse-engineer exact characteristics, we chose a so-called "quarter-wave" coating. It consists of a single thin layer. With such coating, the reflectivity of the surface can be minimized for a center wavelength λ_0 for a given angle of incidence, θ_0. This requires a solid material of very low refractive index; in practice, the best choice is often MgF_2 ($n = 1.38$). The layer thickness is chosen to result in a phase shift of $\pi/2$ (hence the name).

While an analytical expression for $R(\lambda, \theta)$ can be derived in most cases, even the simple quarter-wave coating involves multiple instances of the Fresnel equations, making the expression relatively complex. An example plot for a quarter-wave coating is shown in Figure 4. One way to approximate such a function is to store it in a precomputed 2D texture, which enables us to also record or use arbitrary available coating functions. Nonetheless, in practice, the GPU's arithmetic power is usually high enough to evaluate the function directly. The computation scheme for $R(\lambda, \theta)$ as used in our shader is provided as supplemental material.

3.2 Absorptance

All optical glasses partially absorb light that passes through them. However, this is a weak effect (with typical light loss of a few percent across the entire lens system) and of low frequency (global attenuation). We therefore chose not to include it in our model.

3.3 Diffraction

When light passes through small-scale geometry such as the iris aperture in our system, or small imperfections (fingerprints, dust, scratches), it is diffracted into geometrically shadowed regions. The physical explanation can be found in Huygens' principle which states that every point on a wavefront can be thought of as the emitter of a spherical wave. The resulting pattern is defined by the superposition of these elementary waves [Hecht 2001; Goodman 2005].

In our system, we encounter two typical occurrences of diffraction: the starburst shape centered around the image of the light source and the subtle ringing patterns around the border of each reflection ghost. In a wave-optical framework, both can be computed exactly by evaluating the so-called diffraction integral for all points on the sensor. This is very costly and by no means possible in real time. Instead, we are interested in a computationally cheap approximation. As it turns out, we can convincingly reproduce the above effects by precomputing a small set of textures using the popular Fraunhofer and Fresnel approximations to the diffraction integral, respectively.

Starburst pattern – Fraunhofer approximation Light passing through a transmissive aperture and propagating further through free space is diffracted into a characteristic far-field pattern. Assuming a uniform incident distribution of parallel (collimated) light (at wavelength λ) and a real-valued amplitude transmission function $T(x, y)$, the Fraunhofer pattern $T'(x', y')$ in an observation plane at distance z_0 from the aperture is given by the Fourier power spectrum (i.e., the squared-magnitude Fourier spectrum) of T [Hecht 2001]. The relation between image coordinates (x', y') and Fourier frequencies (u, v) is given by:

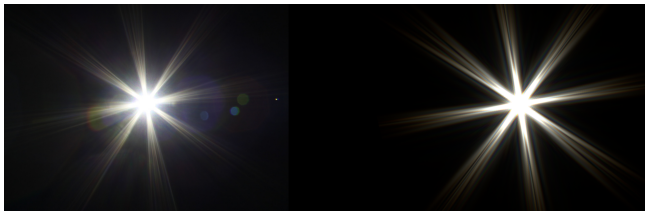

Figure 5: *Left: Photo of a bright light source. Right: Chromatic Fourier transform of an aperture transmission function.*

$$(x', y') = (u, v) \cdot \lambda \cdot z_0 \tag{2}$$

For a given aperture geometry, it is therefore sufficient to compute a single Fourier spectrum and scale it linearly depending on the wavelength. We start with the polygonal transmission function in unit size and optionally add some noise (Figure 14, right). Similar to [Ritschel2009], we then compute a 3-channel starburst texture $V_{\mathrm{RGB}}(s,t)$ by superimposing multiple scaled copies of the power spectrum of the aperture. We found wavelength steps of 5 nm to be sufficiently fine to blur out the radial ringing present in the individual spectral terms. The final texture (see Figure 5 for an example) is normalized to unit radiant flux per color channel. During runtime, we center it at the projected sensor location of the light source, and scale it in size (w, h) and intensity I as follows:

$$w = h = w_0 \cdot \# \qquad \texttt{// scale with reciprocal aperture size} \tag{3}$$

$$I = I_0 \cdot I_{\mathrm{RGB}} \cdot \#^{-4} \qquad \begin{array}{l}\texttt{// } \#^{-2} \texttt{ of light transmitted thru iris} \\ \texttt{// spread over } \#^{2} \texttt{ the starburst area}\end{array} \tag{4}$$

where $\#$ is the f-number (as in "$f/\#$"), and I_{RGB} the radiant flux entering the lens expressed as an RGB vector. We are not aware of a simple way of obtaining the accurate scaling constants w_0 ("size") and I_0 ("intensity") that are specific to the optical system. Instead, since both parameters are rather intuitive, we leave this choice to the user, allowing them to either recreate the appearance of a given optical system, or to amplify and resize the starburst as they desire.

Ringing pattern – Fresnel approximation Optical systems are usually laid out such that the aperture plane is Fourier transformed into the sensor plane by the back section of the lens system. In the presence of interreflections, arbitrary transforms are possible, ranging from an image of the aperture itself to its Fourier transform, but in particular intermediate patterns that share features with both the spatial and Fourier domains.

Mathematically, the continuum between the spatial and Fourier domains can be expressed in terms of fractional powers of the Fourier operator. Research in wave optics shows a close relation between this Fractional Fourier Transform (FrFT) and the so-called Fresnel approximation for near-field diffraction [Ozaktas et al. 2001]. Without the need for a deeper understanding of Fourier optics, we can use the FrFT code from [Ozaktas et al. 2001] to embellish given aperture functions with plausible ringing patterns (Figure 6). It does not take much effort to find a good value for the only parameter, the fractional order α, to reproduce a given flare pattern. Heuristically, we found the following to work well (longer wavelength and smaller aperture leads to stronger ringing):

$$\alpha = 0.15 \cdot (\lambda/400\,\mathrm{nm}) \cdot (\#/18) \tag{5}$$

As we will see in the next section, these precomputed textures can be used during our rasterization step, where they replace the sharp aperture image that would result from standard ray tracing.

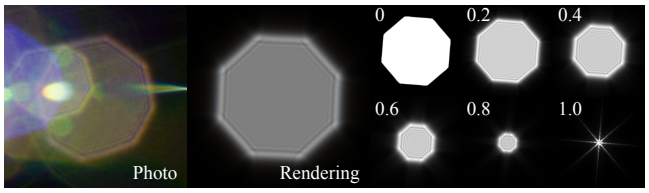

Figure 6: *Left: Close-up on a lens flare photo. Note the ringing around the contour of each ghost. Middle: Chromatic FrFT of an octagonal aperture. Right: Gradual transition from spatial to Fourier domain ($0 \le \alpha \le 1$ in steps of 0.2).*

4 Rendering System

The previous section analyzed how to model important aspects of lens-flare effects. Here, we present our rendering technique to simulate the actual light propagation. It is based on ray tracing through the optical system to the film plane (sensor). In contrast to expensive off-line approaches [Keshmirian 2008; Steinert et al. 2011], we only trace a sparse set of rays. Each ray records values about the lens-system traversal. Rays reaching the sensor implicitly define a *ray grid* across which we then interpolate the recorded values in image space. Hereby, we can approximate the outcome of rays that were never actually shot, leading to an approximate beam tracing. The overview of our rendering pipeline is illustrated in Figure 7. In the following, we will elaborate on the stages of our pipeline.

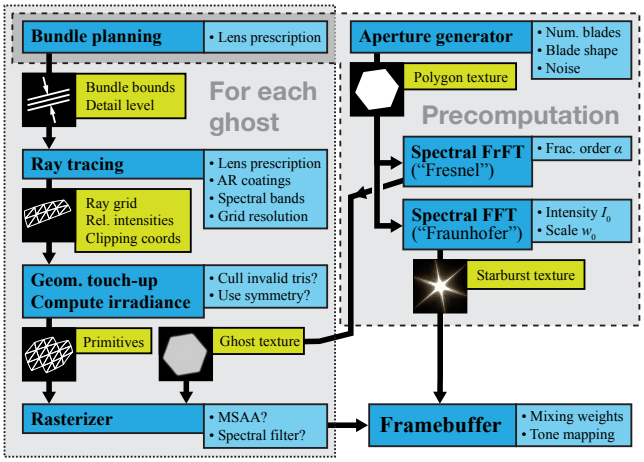

Figure 7: *Our lens flare rendering pipeline.*

Assumption We will assume a directional, or distant, light source which holds for most sources of flare (e.g., sunlight, street lights, and car headlamps). This assumption is not a necessary requirement of our algorithm, but helpful for its acceleration.

4.1 Rendering Scheme

Ghost Enumeration Rays traversing the lens system are reflected or refracted at lenses. Each flare element caused by interreflection, henceforth called "ghost", corresponds to a specific sequence of these transmissions and reflections. Only sequences involving an even number of reflections impinge on the sensor. Those with more than two reflections can usually be ignored; only a small percentage of light is reflected and they are typically by orders of magnitude weakened leading to insignificant contributions in the final image (Figure 8). We enumerate all two-reflection sequences: light enters the lens barrel, propagates towards the sensor, is reflected

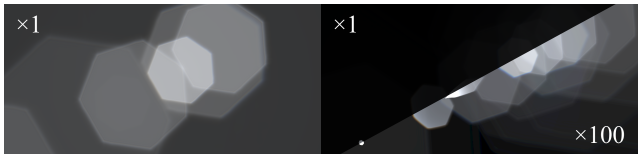

Figure 8: *Ghosts caused by 2-fold (left) and 4-fold (right) inter-reflection. The small percentage reflected at each surface significantly weakens higher-order flare, despite the higher number of such ghosts. To improve performance, we do not render them by default.*

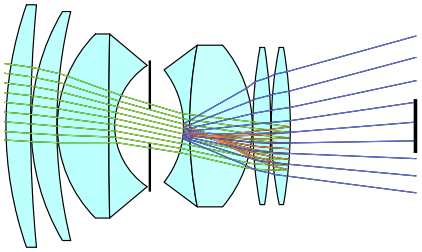

Figure 9: *One out of 91 possible double-reflection sequences for this lens design*

at an optical surface, travels back, is again reflected, and, finally, reaches the sensor (Figure 9). For n Fresnel interfaces in an optical system, there are $N = n(n-1)/2$ such sequences that are treated independently to render one ghost at a time.

Bundle Tracing For a given ghost index and incident light direction, a parallel bundle of rays is spanned by the entrance aperture of the lens barrel. Next, we select a sparse uniform set of rays to track through the lens system. Because we know the exact intersection sequence for each ghost, unlike classical ray tracing, we do not need to follow each ray with a recursive scheme, elaborate intersection tests, or spatial acceleration structures. We parse the sequence into a deterministic order of intersection tests against the algebraically-defined lens surfaces. This makes our technique particularly well suited for GPU execution.

At each intersection, we compare the hitpoint of the ray with the diameter of the respective surface and record its maximum normalized distance from the optical axis along the way through the system:

$$r_{\text{rel}}^{(\text{new})} = \max(r_{\text{rel}}^{(\text{old})}, r/r_{\text{surface}}),$$

where r is the distance of the hitpoint to the optical axis, and r_{surface} the radius of the optical element. Also, as a ray passes through the aperture plane, a pair of intersection coordinates (u_a, v_a) is stored. Note that we do not discard rays that escape from the system ($r_{\text{rel}} > 1$), since even these are valuable for interpolation in the ray grid (see below). Furthermore, we extrapolate the functionality of each optical surface virtually by relying on its defining algebraic function beyond the nominal lens diameter. To make this extension work, we replace the common nearest-surface check with a strict in-order intersection, all the while allowing the resulting ray parameter to be negative (Figure 10). This increases the numerical stability of the simulation for small ray densities, since more rays can pass through the system in a mathematically continuous way. Note that these non-physical rays will not end up in the final rendering but only serve as data points for interpolation. For interpolated rays that are actually drawn, the *next* surface is also the *nearest*.

Only when a ray can no longer be intersected with the next surface at all, or undergoes total internal reflection, it is pruned. This can create holes in the ray grid, but we did not see the need for any refinement strategies. It simply proved unproblematic because the ray's transported energy approaches zero in the vicinity of total inner reflection, making its neighbors and the area on the ray grid appear black in the final rendering anyway.

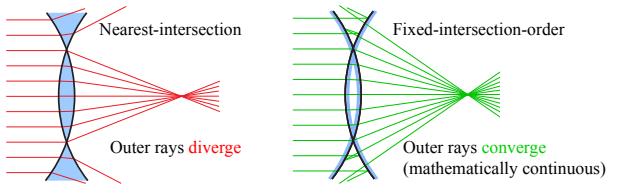

Figure 10: *Pushing the limits of a biconvex lens (illustration). Left: traditional intersection with the nearest surface along the way. Right: our intersection in fixed sequence. Note how the outermost rays still converge to the focus.*

Rasterization and Shading Once the rays have been traced through the system, they form a *ray grid* on the sensor plane (Figure 11). The set of rays is sparse and, each ray taken by itself, would deliver insufficient quality. Our goal is to interpolate information from neighboring rays to estimate the behavior of an entire ray beam. To this end, we do not use a random sparse set of rays, but initialize the ray set as a uniform grid placed at the first lens element. Each grid cell on the entrance plane can be matched to a grid cell on the sensor between the same rays. Similarly to traditional beam tracing, the total radiant power transported through each beam is now distributed evenly over the area of the corresponding quad, leading to intensity variations in the lens flare. Additional shading terms (in particular, Lambertian cosine terms) are taken into account.

Note that so far, we did not cull rays that were blocked by the lens system or aperture, but we recorded the position where they traversed the aperture (u_a,v_a), and its maximum distance to the optical axis, r_{rel}, with respect to the radius of the respective surface. When treating a beam, we can now interpolate these coordinates over the corresponding quad. Hereby, more accurate inside/outside checks for the interpolated rays become possible; we apply clipping on a fragment basis when the *interpolated* radius exceeds the limit distance. Finally, the position on the aperture is used to decide the ghost shape by a lookup in an aperture texture. This is also when the "Fresnel-like" diffraction comes in (Section 3.3), since the ringing pattern has been precomputed and stored in the aperture texture.

4.2 Accelerations

The previously described algorithm delivers convincing results for simple lens systems. Here, we present several strategies to improve upon the basic solution in terms of quality and speed.

Ray Bounding Of all the rays entering the lens from a given direction, only a small subset can actually reach the sensor. Many rays are blocked by obstacles, in particular, the iris aperture when it is set to a small diameter. To save computational resources, we therefore restrict the sparse set of rays to a rectangular region on the entrance aperture that is chosen to enclose all rays that might potentially make it all the way to the sensor.

The location and dimensions of this bounding region depend on the light direction, aperture size, and possibly other parameters (zoom, or focus) in a nontrivial way, making a run-time parameter search difficult. Instead, we propose a preprocessing step to estimate the optimal bounding region for each ghost. For a given configuration, we employ the previous basic algorithm with a low resolution grid to recover all rays that actually reach the sensor. Based on this grid, we determine a bounding rectangle on the entrance aperture. It proved

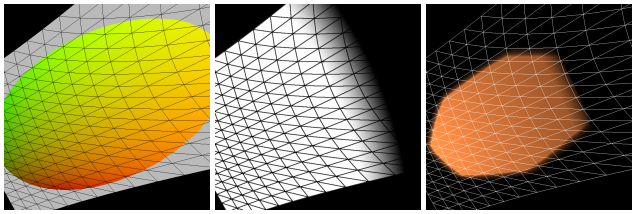

Figure 11: *From left to right: A ray bundle mapped to a grid on the sensor plane (color-coded aperture texture coordinates (u_a, v_a)), clipping radius r_{rel}, shading with aperture texture and clipping).*

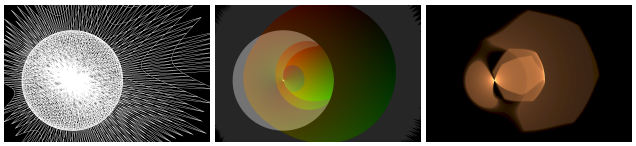

Figure 12: *A highly complex ghost. From left to right: Deformed ray grid on sensor, aperture texture coordinate, rendered caustic.*

sufficient to make the bounds wide enough to contain all valid rays for the current and all neighboring parameter settings. A subdivision scheme can help in speeding up the bounding procedure.

Adaptive Resolution Lens flare is a set of caustics of a complex optical system, which also implies that very high frequencies can occur. While ray bounding significantly improves performance and quality, subtle changes might still be missed. In our algorithm, a regular grid of incident rays is mapped to a more or less homogeneous grid on the sensor. In most cases, the grid undergoes simple scaling and translation which is captured with sufficient precision even for a coarse tessellation (Figure 11). In some configurations, though, the accumulation of nonlinear effects can cause severe deformations, fold the grid onto itself, or even change its topology (Figure 12). Such ghosts require a higher grid resolution.

We employ a heuristic approach to adapt the grid resolution for each ghost. As an indicator, we use the area of grid cells. A large variance across the grid implies that a non-uniform deformation occurred and higher precision is needed. We therefore evaluate this variance during precomputation, and assign one out of six detail levels to each ghost, with resolutions from 16×16 to 512×512 rays per bundle. Thus, through early identification of challenging ghosts the use of more sophisticated subdivision techniques can be avoided.

Intensity LOD Another piece of information obtained during the precomputation step is an approximate intensity of the resulting ghost. Given this information, during runtime, the user can control the budget by fixing the number of brightest ghosts to be evaluated.

Aperture Culling For small iris openings, rays rarely traverse the aperture multiple times without being blocked. As a result, the corresponding two-reflection sequences (with three aperture traversals) can usually be ignored without introducing strong artifacts. Hereby, the number of enumerated sequences is reduced significantly to $N = (f(f-1) + b(b-1))/2$, where f and b are the number of lens surfaces in front of or behind the aperture, respectively.

Symmetries Symmetries in the optical system can help reduce computational complexity. By design, most photographic lenses are axisymmetric, whereas anamorphic lenses (featuring two orthogonal planes of symmetry that intersect along the optical axis) are common in the film industry. The latter are currently not supported in our system, but they could be added by replacing the spherical lens surfaces with more general, ellipsoidal, shapes.

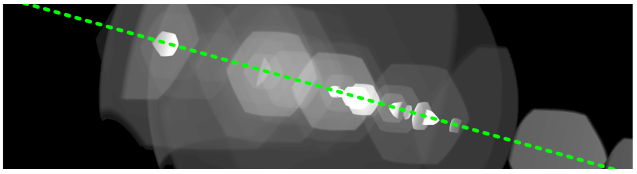

Figure 13: *Mirror symmetry - only the iris shape causes asymmetry*

For *axial* symmetry, we can reduce the amount of required precomputation drastically; all computation up to and including the ray tracing is done for a fixed azimuthal angle of incidence, and then rotated into place. Furthermore, we can reduce the sparse ray set by exploiting the *mirror* symmetry of the flare arrangement, only considering half the rays on the entrance plane. The grid on the sensor can then be mirrored along the symmetry axis. Please notice that our choice to not block rays directly, but record aperture coordinates and intersection distances, enables us to consider the whole system as symmetric—even the aperture, which in general is not, just as the resulting ghosts are not symmetric.

Spectral Rendering Treating antireflective coating and chromatic lens aberrations requires a wavelength-dependent evaluation. For a brute-force evaluation, most ghosts are well represented with only three wavelengths (RGB), but a few (typically, 3 out of 140 ghosts), can require up to 60 wavelengths for smooth results. While a level-of-detail (LOD) approach could be imagined, we render at 3 (standard quality / RGB) or a maximum of 7 (high quality) wavelengths, which comes at a moderate computational cost, but employ an interpolation strategy. We filter the result of each wavelength band in image space to create transitions. The orientation and dimension of the required 1D blur kernel have been derived during the precomputation phase, from the spatial variation between neighboring wavelength bands. The filter size is chosen to bridge the gap between the bands. The filtered representations are then blended together in the RGBA framebuffer and deliver a smooth result.

4.3 GPU Implementation

Basic Algorithm We perform the ray tracing in the vertex shader. To deal with total reflection, culled rays are flagged via a texture coordinate. The geometry shader then produces the triangle strips that form beam quads in the grid. Here, the area of each grid cell is also computed, and used to compute the irradiance. For symmetric systems where only half of the ray set is traced, the geometry shader mirrors each triangle along the symmetry axis of the flare arrangement. This doubling of triangles is more efficient than image-based mirroring. The resulting quads on the sensor are rasterized in the fragment shader that can discard fragments if they correspond to blocked rays ($r_{\text{rel}} > 1$, see Section 4.1). Per-vertex irradiance values are interpolated over the quad, and a texture lookup based on the aperture coordinate completes the rendering. Finally, all ghosts are composited additively.

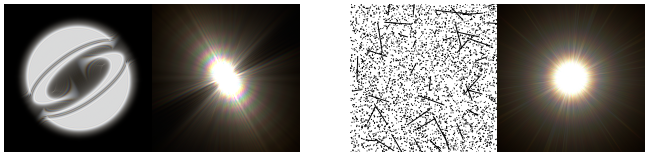

Figure 14: *SIGGRAPH logo aperture shape (left) and a procedural "dirt" pattern consisting of dots and lines (right), each along with its chromatic Fourier transform.*

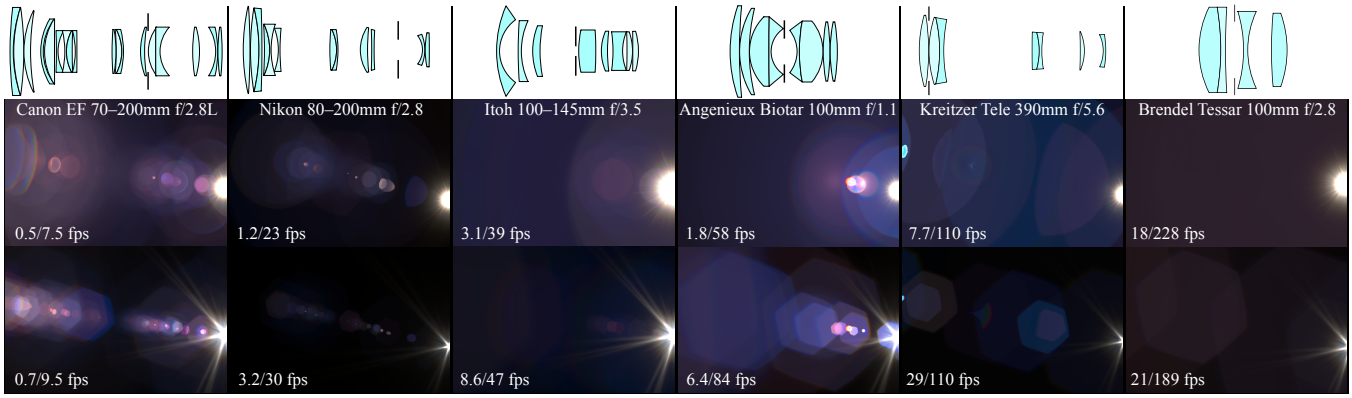

Figure 15: *Lens flare of various lens systems. Fps are given for high quality (more rays do not bring improvement) and standard settings. Top row: lens layout. Middle row: aperture fully open. Bottom row: aperture reduced by 4 f-stops.*

Smooth shading An improvement in quality can be achieved by not shading quads, but vertices and interpolating (Gouraud shading). At each vertex, we store the average value of its surrounding neighbors. The regular grid of rays, combined with the transform feedback (or the stream-out) mechanism of modern graphics hardware, makes this lookup of neighboring quad values very easy.

5 Artistic Control

Like other lens effects [Lee et al. 2010], flare can serve as a creative tool to increase the appeal of synthetic images and photographs alike. Our algorithm offers many possibilities to interact with the basic pipeline in order to exceed physical limitations while maintaining a plausible look. Due to our approach of enumerating all interreflections, only certain, maybe the most beautiful, ghosts can be selected for rendering. Furthermore, imperfections can be well represented with very approximate means.

Creative Use of Optical Elements While the most common lenses feature apertures shaped like regular polygons, any 2D shape can be used instead. As an example, we used the SIGGRAPH logo which results in an unusual starburst pattern (Figures 14) as well as transformed ghosts of the logo all over the image (Figures 16).

Adding Realism through Imperfections Lenses in the real world are often degraded by dust and imperfections on the surface that can affect the diffraction pattern. We give control over this effect by adding a texture of dust and scratches to the aperture before determining the Fourier spectrum. Drawing a dirt texture is possible, but we also offer procedural generation of scratches and dust based on user defined statistics (density, orientation, length, size). While scratches add new streaks to the lens flare, dust has a tendency to add rainbow-colored speckles. In addition, the texture could be animated to achieve dynamic effects as in [Ritschel et al. 2009].

Figure 16: *HDR video frames with added post-process lens flare. Left: "SIGGRAPH" lens equipped with a custom aperture shape. Right: A HDR frame from the short "Fiat Lux" by Paul Debevec, seen through the Nikon lens.*

Imperfect Symmetries Since real lens systems are never exactly symmetric, lens flare does not line up perfectly on the mirror axis. To model this imperfection, we add a variance that translates each ghost slightly in the image plane. This modification offers more intuitive control than a corresponding change in the lens system.

Anti-Reflective Coating The color of each individual ghost is mainly determined by the anti-reflective coatings of the lens surfaces causing it. This effect can easily be abstracted by letting the user provide color ramps or global color changes for each ghost.

6 Results

We implemented our solution on an Intel Core 2 Quad 2.83 GHz with an NVIDIA GTX 285 card. Our method reaches interactive to real-time framerates depending on the complexity of the optical system and the accuracy of the simulation. As illustrated in Figure 15, our method can be of interest for demanding real-time applications, but also for higher-quality simulations. Figure 19 shows that even at significantly reduced resolution, the ghosting computed using our technique is close to the ray-traced reference. For performance, one could even pick only those ghosts that are particularly beautiful, yielding a significant speedup while maintaining the artistic expression. In practice, culling the 20% weakest ghosts delivers 20% speedup without introducing visible artifacts. Even 40% still proved acceptable for interactive applications (speedup approx. 50%). In Figure 15, we provide performance ratings for different quality settings. A side-by-side comparison of the settings used can be found in Figure 17. The most costly effect of our solution are caustics in highly anisotropic ghosts (e.g., Figure 18) because here the ray bundles are most sensitive to spectral and spatial variation.

Our solution does need to perform a reasonably quick precomputation step to bound the sparse set of rays. For a simple lens such as a Brendel prime lens (9 ghosts), this precomputation takes less than 0.1 sec; for the Nikon zoom lens (142 ghosts), it takes 5 min.; for the Canon zoom lens (312 ghosts), it takes 20 min to iterate through all ghosts $\times$ 90 light directions $\times$ 64^2 rays $\times$ 20 zoom factors $\times$ 8 aperture stops. The latter two allow us to freely change camera settings on the fly.

Our algorithm produces physically-plausible lens flare renderings (Figure 1). Most important effects are simulated convincingly, leading to images that are hard to distinguish from real-world footage.

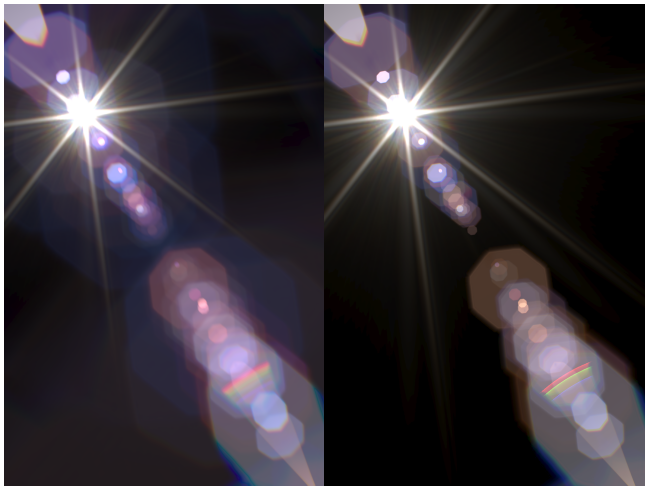

Figure 17: *Quality settings. Left: high quality (7 spectral bands, spectral filtering, supersampling). Right: normal quality (RGB, no filtering, no supersampling, remove 40% darkest ghosts). The corresponding frame rates are 6.1 fps and 20.6 fps, respectively, on an NVIDIA GTX 285.*

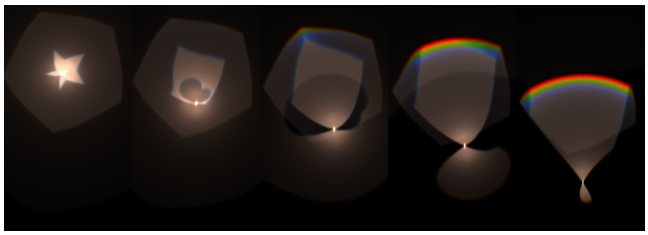

Figure 18: *Five different views on "Ghost #103" in the Nikon lens system. Note the intricate folding and chromatic variation.*

The main difference arises from imperfections of the lens system and our approximate handling of diffraction effects. Furthermore, the lens coating is unknown, forcing us to an estimate.

Our algorithm naturally handles complex deformations and caustics (Figure 18). Previous real-time methods were unable to obtain similar results because this effect can only be reproduced when light paths through the system are simulated. Our model considers many aspects that were neglected by previous approaches (e.g., the reflectivity of lens coatings as a function of wavelength and angle). Even with these improvements and at highest spatial and spectral resolutions, rendering flares for even the most complex optical designs takes no more than a few seconds. This is significantly faster than a typical path-traced solution that would take hours, if not days, to converge on today's desktop computers [Steinert et al. 2011].

The memory consumption of our algorithm is mainly defined by the textures containing the aperture and its Fourier transform (24 MB worth of 16-bit float data), as well as three render buffers (another 24 MB).

7 Discussion and Limitations

7.1 Applications

Our algorithm has low memory overhead and is computationally efficient. It could be of use for various application scenarios:

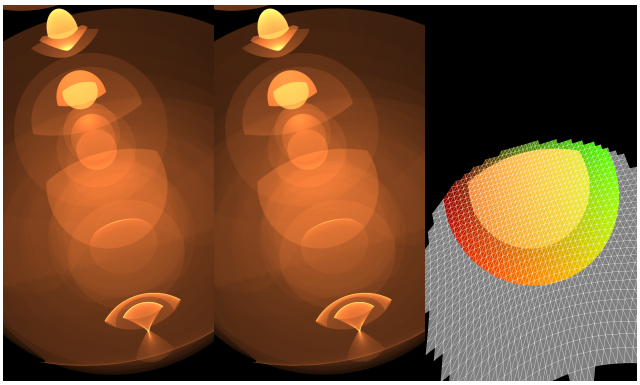

Figure 19: *Left: densely ray-traced rendering (resolution per ghost 8192^2 samples; rendering time 159 s on an NVIDIA GTX 580). Middle: rendering using sparse ray bundles at a maximum resolution of 128^2, and our interpolation technique (29.8 ms per frame on the same hardware). Right: A closer look on one of the ghosts: underlying grid structure and aperture coordinates.*

CG Movie Production The quality delivered by our rendering scheme exceeds many previous offline techniques, making it interesting as a preview, but even as a final rendering solution. Artistic control further allows a user to maintain a realistic appearance while being able to fine-tune and customize the effect.

Computer Games Deactivating costly calculations increases the overall performance, making our solution useful for games. Furthermore, our two-reflection assumption allows the user to choose particular ghosts that they consider important. For well-behaved flares, even a very small amount of rays (e.g., 4×4) delivers high quality thanks to the interpolation step.

Image and Video Processing Current lens flare filters do not appear convincing because they keep a static look, e.g., ghost deformations are ignored. Our method is temporally coherent, making it a good choice for movie footage as well. We detect and follow light sources in the image (using an intensity threshold). One could also animate the light manually to emphasize scene elements or guide the observer. Our instant feedback is of great help in this context.

Lens-System Design Even in live-action cinematography, lens flare is sometimes considered a desired effect [Woerner 2009]. Certain lens systems (such as the Lensbaby) are designed for creative use of aberrations and other optical effects. In combination with a traditional lens design tool, our algorithm could help lens designers to preview and optimize the flare characteristic of an optical design.

Deflaring Another interesting possibility would be to predict and remove flare patterns from actual photographs. However, this would require a perfect description of the optical system with its countless parameters und unknowns, which is currently out of reach.

7.2 Limitations

Light Sources Our current rendering mechanism is optimized to having a single visible point/directional light source. Area lights can be approximated by a point light and an energy emission proportional to their visibility, but such solutions remain approximate. A more accurate possibility is to sample the source at an additional cost.

Precomputing Resolution While the ray bounding precomputation traces rays through the system for various light positions, one should not conclude that these low-resolution images could be used for rendering. We initially experimented with image-warping strategies, but it proved futile because the subtle changes (such as displacement and deformation) cannot be well addressed. Also, such a solution is memory intensive, while ours only stores a small lookup table of bounding rectangles.

Aliasing A common problem when triangles become smaller than one pixel is rasterization aliasing. The situation can lead to very high intensity, but potentially error-prone rasterization. In practice, this only happens for very anisotropic ghosts (min/max ratio > 1000) and their number is very small (for the Nikon lens, 3 out of 142). If desired, we can select these flares and treat them with a higher resolution framebuffer which is, in the end, added to the standard framebuffer. Alternatively, one can replace the rasterization of these small triangles with a point rendering technique.

8 Conclusion

We presented an interactive rendering algorithm to simulate lens flare of complex lens systems. We showed superior results with respect to previous interactive solutions and even offline suggestions to a large extent. We also introduced various means to artistically modify and enhance the rendition beyond physical limitations for stylization purposes. Our algorithm is flexible in the sense that we allow a fine tradeoff between accuracy and performance by allowing the user to choose the simulated effects that are most important for the application context. Hereby, our method addresses high-quality, as well as medium-quality real-time purposes.

In the future, we imagine that our fast algorithm could serve as a tool to recover parameters of unknown optical systems. Given the fact that flare patterns are very sensitive to slight parameter changes, an analysis-by-synthesis scheme could enable nondestructive lens characterization. Such a calibrated flare synthesis might then enable us to "deflare" HDR images taken by the system. Capturing artifact-free HDR data is an unsolved problem.

We thank the anonymous reviewers for their valuable comments and suggestions. This work was partly funded by the Intel Visual Computing Institute at Saarland University. Sungkil Lee gratefully acknowledges support by the Basic Science Research Program through the National Research Foundation of Korea, funded by the Ministry of Education, Science and Technology (2011-0014015).

References

ALSPACH, T., 2009. Vector-based representation of a lens flare. US Patent 7,526,417.

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

ERNST, M., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Interactive rendering of caustics using interpolated warped volumes. In *Proc. Graphics Interface'05*, 87–96.

GOODMAN, J. W. 2005. *Introduction to Fourier Optics*, 3 ed. Roberts & Company Publishers, December.

HECHT, E. 2001. *Optics*, 4 ed. Addison Wesley, August.

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

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

KING, Y. 2001. *Game Programming Gems 2*. Charles River Media, ch. 2D Lens Flare.

KINGSLAKE, R. 1992. *Optics in Photography*. SPIE Publications.

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

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

MAUGHAN, C. 2001. *Game Programming Gems 2*. Charles River Media, ch. Texture Masking for Faster Lens Flare.

OAT, C. 2004. *Shader X3*. Charles River Media, ch. A Steerable Streak Filter.

OGAWA, H., 1996. Zoom lens. US Patent 5,537,259.

OH, S. B., KASHYAP, S., GARG, R., CHANDRAN, S., AND RASKAR, R. 2010. Rendering Wave Effects with Augmented Light Field. In *Computer Graphics Forum (Proc. Eurographics)*.

OZAKTAS, H. M., ZALEVSKY, Z., AND KUTAY, M. A. 2001. *The fractional Fourier transform with applications in optics and signal processing*. Wiley.

PERRIN, J.-C. 2004. Methods for rapid evaluation of the stray light in optical systems. SPIE, L. Mazuray, P. J. Rogers, and R. Wartmann, Eds., vol. 5249, 392–399.

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

POLYANSKIY, M., 2010. Refractive index database. http://refractiveindex.info.

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 *Computer Graphics Forum (Proc. Eurographics)*.

SCHOTT AG, 2011. Optical glass catalogue, January 2011.

SEKULIC, D. 2004. *GPU Gems*. Addison-Wesley, ch. Efficient Occlusion Queries.

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

SMITH, W. J. 2005. *Modern Lens Design*. McGraw-Hill.

STEINERT, B., DAMMERTZ, H., HANIKA, J., AND LENSCH, H. P. A. 2011. General spectral camera lens simulation. In *Computer Graphics Forum*, vol. 30, to appear.

TOCCI, M., 2007. Quantifying Veiling Glare (ZEMAX Users' Knowledge Base). http://www.zemax.com/kb/articles/192/1.

WENZEL, C., 2005. Far Cry and DirectX. http://developer.amd.com/media/gpu_assets/D3DTutorial08_FarCryAndDX9.pdf.

WOERNER, M., 2009. J.J. Abrams Admits Star Trek Lens Flares Are "Ridiculous" (interview). http://io9.com/#!5230278.