Authors manuscript. Published in The Visual Computer, first online: 14 May 2018
The final publication is available at Springer via https://doi.org/10.1007/s00371-018-1552-4.

Lens flare prediction based on measurements with real-time visualization

Andreas Walch[1] · Christian Luksch[1] · Attila Szabo[1] · Harald Steinlechner[1] · Georg Haaser[1] · Michael Schwärzler[1] · Stefan Maierhofer[1]

Abstract Lens flare is a visual phenomenon caused by interreflection of light within a lens system. This effect is often seen as an undesired artifact, but it also gives rendered images a realistic appearance and is even used for artistic purposes. In the area of computer graphics, several simulation based approaches have been presented to render lens flare for a given spherical lens system. For physically reliable results, these approaches require an accurate description of that system, which differs from camera to camera. Also, for the lens flares appearance, crucial parameters – especially the anti-reflection coatings – can often only be approximated.

In this paper we present a novel workflow for generating physically plausible renderings of lens flare phenomena by analyzing the lens flares captured with a camera. Our method allows predicting the occurrence of lens flares for a given light setup. This is an often requested feature in light planning applications in order to efficiently avoid lens flare prone light positioning. A model with a tight parameter set and a GPU-based rendering method allows our approach to be used in real-time applications.

Keywords Lens flare · Data driven workflow · Real time

1 Introduction

This work is a practical approach to gain information on lens flare occurrences for a lens system for which no specifications are known. Lens flare is a visual effect that appears due to internal reflections within a lens system or due to scattering caused by lens material imperfections. The intensity of lens flare heavily depends on the camera to light source constellation and of the light source intensity, compared to the rest of the scene [7]. Lens flares are often regarded as a disturbing artifact, and camera producers develop countermeasures, such as anti-reflection coatings and optimized lens hoods to reduce their visual impact. In other applications though, movies [16,23] or video games [21], lens flares are used intentionally to imply a higher degree of realism or as a stylistic element. Figure 1 shows a typical lens flare with highlighted lens flare elements.

Fig. 1 Outdoor lens flare with highlighted ghosts

A lens flare-aware light planning stage allows the light designer to carefully elaborate an optimal camera and light setup to avoid lens flares. Lens flare occurs whenever the light passes the lens system in an unexpected way while getting reflected multiple times within the system. If these non-image-forming paths reach the sensor, they form visible artifacts called **ghosts** [7]. Figure 2 shows how a ghost can occur.

There are two approaches to predict the occurrence of lens flare in a physically plausible way:

Andreas Walch
E-mail: walch@vrvis.at

[1] VRVis Research Center, Vienna, Austria

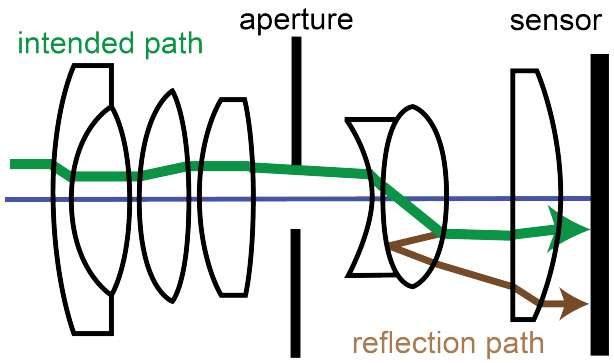

Fig. 2 Unintended reflection path, which forms a ghost

.

One approach to predict lens flare occurrences is simulation. A precise description of the lens system and all its properties are essential to carry out a physically correct simulation. The lens flare characteristics depend heavily on the lens system parameters. However, these parameters are often kept under disclosure by the manufacturers, which is why the simulation can only be approximated and lacks precision.

Another way to predict lens flare occurrence can be done by examining real-world measurements. The camera and all its hidden parameters can be interpreted as a black box. The input is a bright light source, and the black box's output is raw images. The raw images contain all lens system-dependent transformations of the input light, including physically accurate lens flares. This approach does not rely on any internal specifications of the lens system and is therefore very flexible and independent from the camera manufacturers.

Driven by these observations, we developed a data-driven workflow to predict and render lens flare, especially ghosts, for any lens system. In contrast to previous work we do not rely on any specific information of the lens system's internal structure and provide a real-time visualization.

1.1 Background - lens flares

A camera consists of a lens system, where multiple lenses are arranged along the optical axis. The combination of various lenses reduces optical aberrations and allows to design light propagation for specialized photometric demands.

Lens flare can be distinguished into three different kinds of element types: The visibly most prominent effect is a *star*-shaped element, where multiple bright streaks are radiating outward from the light source center [10]. These glare streaks are caused by diffraction at the edge of the aperture. The close area around the light source is also affected from a less distinct glare, called *halo*, which fades out gradually. The positions of these types are easy to predict, since they occur at the position of the light source.

The other prominent effect is a series of circles and rings aligned on a line, which crosses the light source center and the image center. The shapes on this array are called ***ghosts***, and their position and shape differ strongly from camera to camera. These ghosts are the most complex lens flare elements. Their occurrence is non-trivial to predict, because they depend on complex interreflection paths within the lens system.

2 Related work

The light propagation and various optical effects can be accurately simulated, if all parameters are available. In the field of optical lens design, tools like *ZEMAX* or *Code V* provide correct simulation results, but they are not intended to generate (real-time) renderings [20].

2.1 Artificial lens flare rendering

Artificially created lens flare effects are based on sprites to enrich a scene with lens flare effects, but they are not physically correct. Kilgard et. al. [11] suggest to blend lens flare textures in a post-processing effect and to arrange the sprites along a line through the image center following an ad hoc displacement function. King [12] achieves to include camera movements by changing the opacity and size of the lens flare sprites, depending on the angle between camera and light source. Maughan [15] suggests to scale the intensity of the effect by the amount of visible light pixels in the rendered image. Sekulic [17] recommends to use the occlusion queries features of common graphical processing unit (GPU) to handle performance issues. Alspach [1] presents a set of basic vector elements to approximate lens flare elements.

2.2 Physically based lens flare simulation and rendering

Physically motivated lens flare renderings are based on physically plausible simulation. Chaumond [2] uses simple path tracing, while Keshmirian [10] uses photon mapping to render lens flare. Both approaches converge slowly. Since anti-reflection coatings are not included, the color of the ghosts cannot be determined. Steinert et al. [18] present a full spectral camera model based on lens design data and well approximated lens coatings. The complex simulation approach is able to determine the color of ghosts, but the simulation time

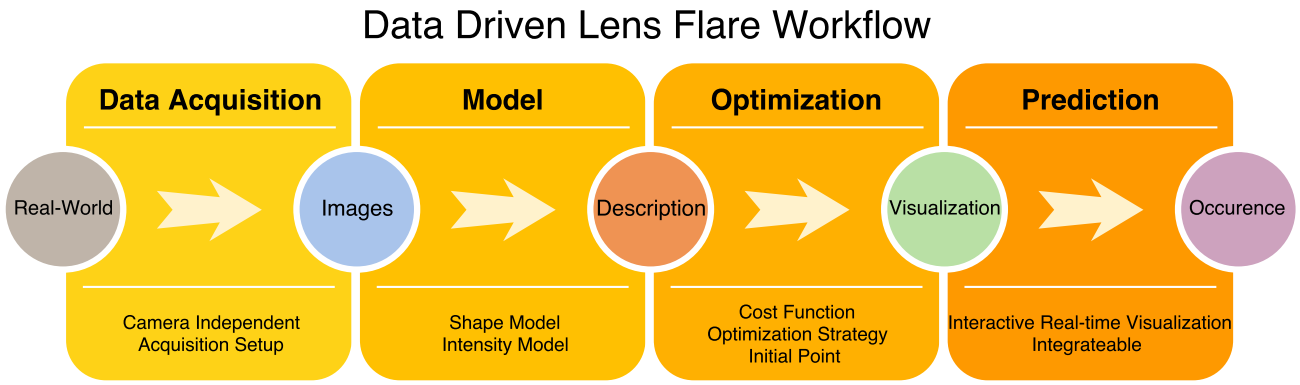

Fig. 3 Overview of our data-driven lens flare rendering and prediction workflow. First data are acquired (Sect. 3.1), upon a model is created (Sect. 3.2). Within the optimization stage the ideal parameter set is found (Sect. 3.3) and is used for interactive real-time visualization (Sect. 3.4) and occurrence prediction (Sect. 3.5).

is beyond the requirements of real-time applications. Hullin etal. [8] improve simulation efficiency by representing the lens system by polynomials, while Hanika [5] increases precision. Hullin et al.'s [7] ray tracing-based approach uses ray bundling to efficiently generate realistic lens flare textures in a pre-processing step. The colors of ghosts are approximated, while lens imperfections are artificially generated by slightly offsetting the ghosts. Lee et al. [13] improve the work of Hullin et al. [7] regarding simulation speed to fulfill requirements of real-time application, but with coarser quality. The ray transfer is efficiently approximated by ray transfer matrices, allowing to directly project a quad from the entrance pupil onto the sensor in constant time. The ghost's intensity is scaled by the projected quad, and its color is sampled by one ray in the ghost center for each color channel separately. Hennessy [6] suggests improving this approach by scaling the ghost's intensity by the effective aperture and to sample a ghost's color by a random angle for each color channel to reduce the chance of complete internal reflections. Joo et al. [9] present a method to simulate lens imperfections caused during the manufacturing process. The swinging movements of the polishing and grinding tools are simulated and used to generate imperfection textures. This approach is also capable of dealing with aspherical lenses, while the other methods are limited to spherical lenses. The texture creation is very costly due to a large number of lens samples to reduce noise, but the generated imperfection textures can be applied in a sprite based rendering approach like proposed by Lee et al. [13] to be used within real-time applications.

3 Workflow

Our approach does not assume any presumptions on the internal lens system's structure. This means, in particular, that the number of lenses, their shapes, and their materials and coatings are unknown. Hence, we have to derive the behavior of lens flare artifacts purely from captured images. As a data-driven workflow (illustrated in Fig. 3), the first stage concerns data acquisition (see Sect. 3.1). After analyzing the sampled ghosts, we sketch a basic model for ghost rendering (Sect. 3.2). The following stage optimizes the parameters of the model to best fit the captured data (Sect. 3.3), which enables a physically plausible digital representation of the captured data. Based on this sparse description, interactive real-time visualization (Sect. 3.4) and occurrence prediction is achieved (Sect. 3.5).

3.1 Data acquisition

To sample the behavior of lens flare, *the camera to light constellation* has to be sampled in discrete steps. While using a sufficient small step size, the ghost's positions do not vary too much between neighboring samples. This allows to reconstruct the behavior of each ghost over the sampled range. To reduce the amount of samples, we assume radial symmetrical lens system, where the symmetrical axis is the optical axis. Therefore it is enough to rotate the camera only around one axis to capture all essential information of a lens flare. The optical axis of the camera has to cross the light source center to depict all variations. To ensure equally small rotation steps (0.2°), we used a programmable motorized panorama rotation head from *Syrp* [19].

A fully opened aperture allows more light to reach the sensor, which reduces the ghost shape complexity and affects the exposure time. The exposure time has to be adjusted carefully to avoid over-saturated areas, but to also depict ghosts with low intensity. The brighter the light source, the less exposure time is needed to depict the same image as a less powerful light and a

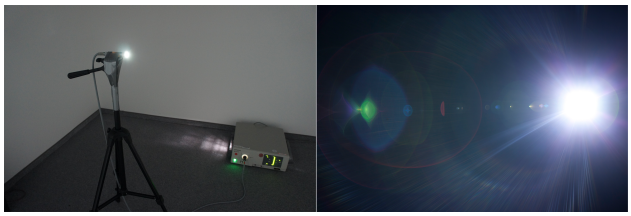

Fig. 4 Light test setup and result (1 s)

longer exposure time. To increase the possible range of intensities, multiple images with different exposure times (0.25 s/1 s/4 s) are sampled and combined to a high dynamic range (HDR) image. The HDR images are converted from the camera-specific RAW-format to the common open-source format *OpenEXR* from Industrial Light and Magic [14] to provide high quality for further processing. The light source itself is assumed to be a point light. To uphold this assumption we used a light source with a very narrow output angle. To ensure that the camera's entrance pupil is fully, but as tightly as possible, illuminated, we placed the light source carefully to simulate parallel incoming light rays (Fig. 4).

The prototype consists of a camera and a bright light source, each placed on a tripod in such a manner that the optical axis of the camera is on the same height as the light source. To reduce background noise the surroundings and tripods are covered by a black curtain. To create our final data set we used a *Canon EOS 5D Mark II* (1 s—f/1.4—24mm—ISO 100—no lens hood) and the endoscopy light system (Olympus CLV-S). The setup's complexity is intentionally kept simple and general to be reusable for any mountable camera model or lens system.

3.2 Lens flare model

The captured samples show a physically correct mapping of lens flares by a specific lens system, but the samples do not contain any description about ghosts (position, shape or appearance). Only the position of the light source is known from the sampling angle. The ghosts share a mirror axis along the captured horizon, and the shape of the ghost is only horizontally clipped by the (circular) entrance pupil. Therefore, the upper and lower parts of the ghost are mirrored, and the left and right side of the shape may differ.

We want to create a simple model to describe and create an approximate visual copy of each ghost within the capture sample. The general position of a ghost can be expressed by a position vector (M_x and M_y) within screen space. To describe the whole ghost **shape**, multiple circular segments have to be smoothly combined. Bézier curves are flexible enough to describe various forms. They can be smoothly attached to each other by placing the control points accordingly. To reduce the overall complexity of our ghost shape model, we only use cubic Bézier curves. While various shapes can be formed by cubic Bézier curves, a perfect circle can only be faithfully approximated by appending four cubic Bézier curves.[1]Four curve segments are also the lower limit to be able to represent the encountered ghosts shapes. Therefore, this restriction does not increase the complexity of our model.

Figure 5 sketches the intention of the shape model. The left L and right R side of the ghost can modify the start points of their representing cubic Bézier curve with respect to M_x, while the height h defines the assembly point and also the end points of the left and right side in respect of M_y. The curvature of each side can be separately adjusted by the control points C_l and C_r. To reduce the model complexity, the control points are restricted to be shifted only axis-aligned to support C^1 continuity.

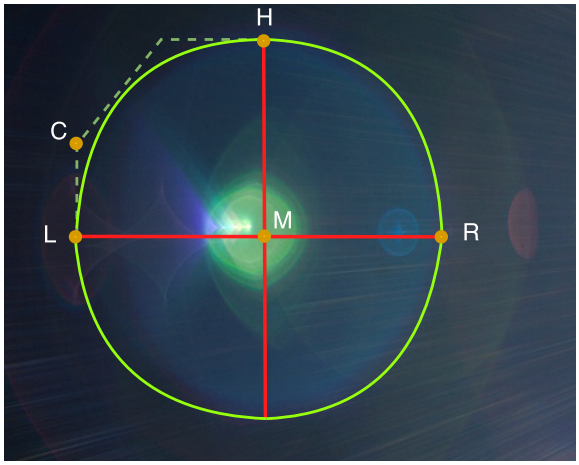

Fig. 5 Visualization of the shape model concept directly derived of a complex ghost sample. The dotted line indicates the weighting polygon for the cubic Bézier segment.

The **intensity** of a point correlates with its signed distance to the visible ghost border. We sample the curve at discrete points using binary search to quickly compute its minimal distance to a query point. After 12 iterations the difference of the binary search method is negligible to an exact, but complex polynomial root finding method. Furthermore, the lightweight iterative method is suitable to be executed directly on the graphical processing unit (GPU), which is desirable as the calculation has to be evaluated per pixel. The sign of

[1] `http://spencermortensen.com/articles/bezier-circle/`

the distance indicates whether the query point is inside or outside of the curve and can be evaluated by basic 2D-trigonometry. To determine the overall minimal signed distance for a given pixel, the minimal signed distance for each curve segment has to be evaluated and compared.

After defining the signed distance for each point its intensity value is applied by a **transfer function**. Figure 6 illustrates our intensity model. The highest intensity values usually rest upon the ghost border I_b and fade out to a constant value I_i (inside) or completely (outside). The fall-out range $F_{(i|o)}$ and slope $E_{(i|o)}$ can be adjusted, respectively, for the inside and outside. By only evaluating pixels, which can be affected by the transfer function, the pixel evaluation can be drastically reduced. As each ghost only describes light, multiple ghosts can be additively blended into a single image. Color can be interpreted as combination of multiple intensity channels; therefore, the intensity transfer function has to be applied to each channel separately. To reduce the overall complexity, the model only supports intensity or gray-scale values.

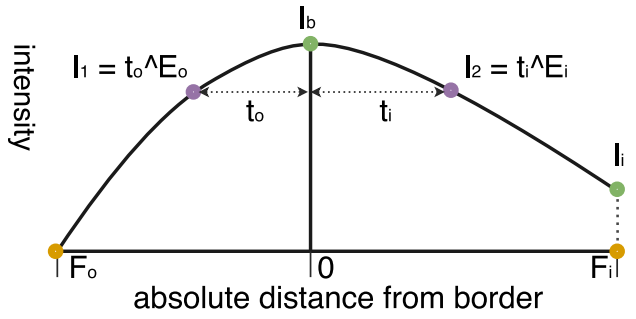

Fig. 6 Visualization of the intensity model. Intensity depends on the pixel distance to the visible ghost border. The inside and outside can be adjusted individually. I_1 and I_2 show an example how the intensity is formed for the inside (t_i) and outside (t_o).

We intentionally create a model with a minimal parameter set in regard to the upcoming stages of the workflow, but powerful enough to describe various ghosts. Furthermore, the model can be easily extended, like to represent a more complex aperture shape by using additional curve elements. The whole rendering and model evaluation is directly performed on the GPU.

3.3 Optimization

By adjusting the model's parameters the captured ghosts can be completely described or quite well approximated, because some details are maybe not depictable by the model. The difference between the captured image and the model rendering has to be as small as possible. The difference can be described by a *cost function*, while reducing the cost indicates a higher similarity between two images. This states a typical optimization problem, where the global minimum of the cost function represents the best solution.

3.3.1 Cost function

A simple cost function can be formulated by the sum of absolute pixel differences. (See Eq. 1 *mean squared error* (MSE), where X is the reference image and Y the model generated image. Further, MSE allows to weight small differences less than larger ones, which gives a better guidance for an optimization routine. Unfortunately, this basic cost function fails on real-world data, because it is not able to distinguish between background noise and actual lens flare elements. Smoothing or clipping the image with a given threshold removes noise, but it also erases information, which is not desirable. In the field of image compression, the structural similarity (SSIM) index is used to describe the perceived similarity of a compressed image to its original [22]. Unfortunately, the more complex and rather costly approach behaves quite similar to the basic MSE cost function.

$$cost_{MSE} = \frac{1}{n}\sum_{i=0}^{n}(X_i - Y_i)^2 \tag{1}$$

To avoid divergence caused by background noise, additional similarity features are necessary. The visible border edge of the ghost shape is also a valid comparison component. We use a discrete *Laplacian filter* kernel to detect horizontal, vertical and diagonal edges. To improve the edge images, each image is pre-smoothed by a *Gaussian filter* kernel to reduce noise. The Laplacian filter kernel only responds to gradient changes and therefore maps an edge as two one pixel wide lines. While calculating the difference of edge images, it is likely that two edges only cross each other but not map completely. By smoothing both edge images again, the chance rises that two edges are partly overlapping or their differences are influenced by close edges. This edge difference cost function and the basic MSE cost function are linearly combined and turn out to converge also with real-world data. To efficiently evaluate the cost function, the image differences are directly processed on the GPU. The MSE is optimized by using a *Mipmap pyramid*, where the top of the pyramid represents the averaged value for the whole image. Image filtering is optimized, by separating the Gaussian kernel into two 1D-kernels to be executed sequentially (Fig. 7).

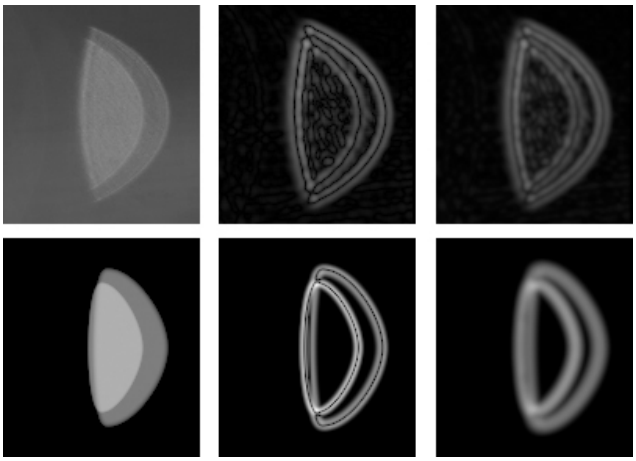

Fig. 7 Edge cost function creation. Top row shows a captured ghost of the acquisition stage, while the bottom row visualizes a model based rendering. The first column is the input file. The second column is the result of a Gaussian-Laplacian filtering. The third column presents the result of the final edge cost, where an additional Gaussian kernel is applied to the second column to increase overlapping in the difference images. For visualization purposes the edge images are brightened up.

3.3.2 Optimization Strategy

We use *multivariate gradient descent* to find the minimum of our cost function. Due to the function's complexity, the gradient cannot be derived in closed form, but it can be numerically approximated by calculating the *central differential quotient* for each parameter. The numerical gradient approximation indicates whether a parameter has to be increased or decreased. For each parameter, an individual step size is maintained, as the parameters vary in range and magnitude. A global *learning rate* ensures fast convergence by decreasing the step sizes only, if the current step size is too crude to find a better solution. In our approach the learning rate is reduced after every 100 iterations by 20%. To ensure that each parameter is equally often changed, the parameters are changed in fixed order (shape before shade). We update the parameters in an *online update* way, where the currently optimized value is directly incorporated into the model before evaluating the remaining parameters [3].

In order for gradient descent to not get stuck in local minima, it is crucial to initialize parameters close to their final values, such that the optimizer only has to fine-tune the solution. As lens flare varies smoothly over the whole image range, model parameters also change very gradually from image to image. Therefore, we can re-use the optimized parameter values from one image as seed values for neighboring images. In order to bootstrap the whole process, we choose a key frame for each lens flare element, where the element is clearly visible and define suitable model parameters. Starting from the key frame we can now track the model from image to image as described above. Additionally, as soon as we have tracked an element across multiple frames, we fit a polynomial to each parameter (in the least-squares sense) to estimate initial parameters for neighboring images more precisely to further speed up convergence. A polynomial of degree 2 proofs to be sufficient and avoids overfitting. Currently, the initial creation of elements at key frames is performed manually.

3.4 Real-time visualization

Each sample can be directly visualized by the previously fine-tuned parameter set. An interactive visualization allows the user to change the camera to light constellation while only reusing the discrete sampled constellations jitters are visible. A denser sample rate could reduce these artifacts, but increases the effort for the whole workflow extremely. To generate a smooth transition between the sampled constellations, we approximate each parameter over the sampled range by a least-squares fitted polynomial of low order. The look-up complexity for any constellation within the sampled range is reduced for each parameter by only one polynomial evaluation.

3.4.1 Visualization integrated into an application

The interactive visualization can be included into a 3D software while using the visualization as an overlay in the post processing stage. The virtual constellation angle between the camera and the light source is directly used to evaluate the previously mentioned polynomials (Sect. 3.4). Because we only support radial symmetrical lens systems, the lens flare rendering can be rotated around its image center to match the roll angle of the light source and the camera's horizon.

To allow a realistic lens flare effect, the virtual camera of the 3D application has to simulate the camera parameters of the captured camera. The focal length is fixed, and we only support a fully opened circular aperture. The intrinsic parameters (aperture stop, film speed, exposure time) adjust the brightness of an image. In case the virtual camera uses the same parameter configuration as the captured camera and the light source is identical, the brightness is equal and therefore the overlay can simply be added to the scene rendering. Otherwise the brightness of the overlay has to be scaled accordingly to match the scene's brightness. Equation 2 describes the scale factor B_{scale} between the lens flare rendering $_F$ and the rendered scene $_R$, where f is the

f-number, ISO is the film sensitivity, EXP is exposure time in seconds. In case the luminance l and the solid angle α of both light sources are known, a brightness ratio can be formed.

$$B_{scale} = \frac{f_F^2}{f_R^2} * \frac{ISO_R}{ISO_F} * \frac{EXP_R}{EXP_F} * \frac{l_R * \alpha_R}{l_F * \alpha_F} \quad (2)$$

The anti-reflection coatings are designed for specific wavelengths. Therefore, to guarantee a realistic lens flare the virtual light source has to match the sampled light's wavelength distribution. By using a normed light source with a known wavelength distribution, other light sources with a similar distribution can be roughly approximated. Otherwise the whole workflow has to be carried out again by capturing the specific light source.

3.5 Lens flare occurrence prediction

The previously presented real-time visualization (Sect. 3.4) creates physical plausible rendering for a given constellation. The generated overlay texture holds numerical intensity values, which can be further examined to predict lens flare occurrence. By analyzing the texture, statistical values can be retrieved and used as an estimator. The *minimum* and *maximum* intensity values are important to define contrast. The *sum* and *average* of all intensity values are able to approximate brightness. The ratio between *logarithmic averages* of virtual scene and overlay can be used as a glare indicator, since the logarithmic average is more stable with respect to outliers than a simple average. The presence of lens flare reduces the contrast of an image. The ratio of the *ideal contrast* and the *real contrast* describes the change of contrast, while lens flare occurs [4].

All previously described occurrence estimators represent a numerical value, but without any context or bounds, these values are not really meaningful. While comparing the current occurrence estimator to the estimators over the whole sampled range, the current constellation can be rated. By inspecting the statistic of a certain estimator, global extremes can be derived and also used to express the quality of the current state.

4 Results

Figure 8 (left) gives a rough overview how lens flare behaves, while the camera starts heading directly into the light source and is horizontally rotated away until no lens flare occurs. Figure 8 (right) shows a close up of a ghost at sample 22.8° to emphasize the complexity of ghosts.

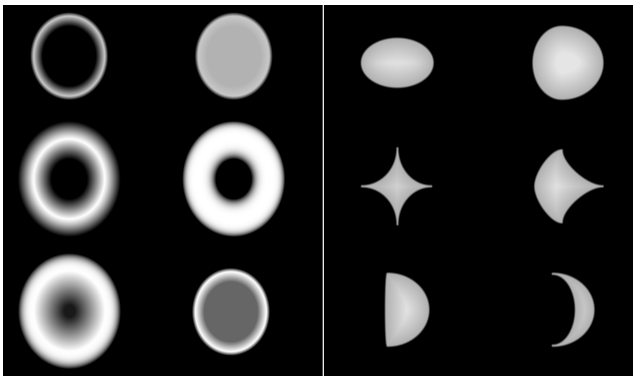

Fig. 9 Left: circular shaped ghost with different distance functions. Right: various ghost shapes generated with our model.

The lens flare model holds a dense parameter set and is very flexible at rendering various ghost shapes. Figure 9 depicts how the appearance of a ghost changes while only manipulating the distance function (left) or shape parameters (right).

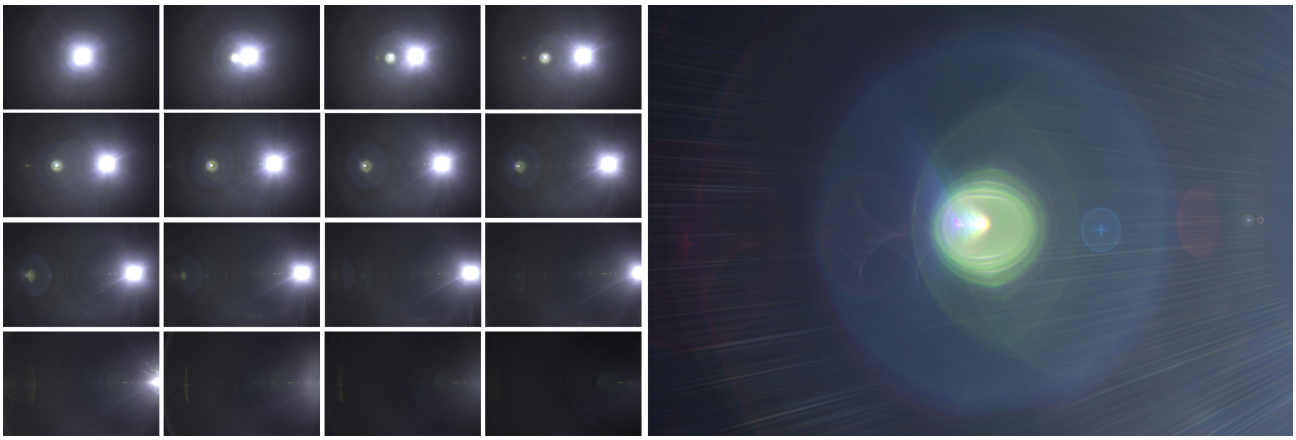

Fig. 8 Left: Rough overview of our sampled data set. Right: close-up of complex ghost at 22.8°

The optimization algorithm uses a cost function to fine-tune the model parameters to best fit a sample image. Figure 10 visualizes the optimization process between two neighboring samples and their corresponding models, while Fig. 11 displays the progress of the cost function and its components. The performance evaluation in Fig. 12 points out the rendering performance versus element resolution. Figure 13 shows a complex lens flare generated with our method.

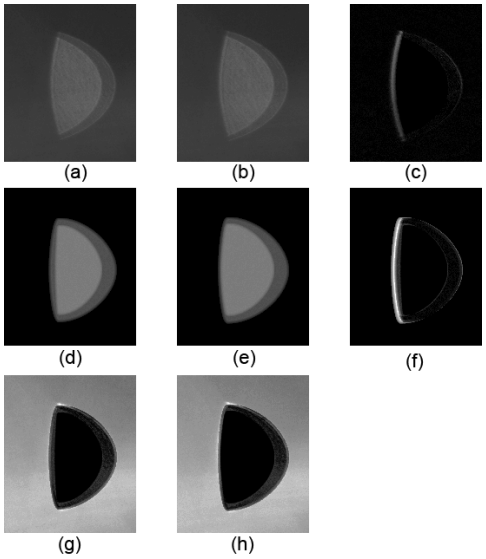

Fig. 10 Optimization process from manually placed model to the neighboring sample. Images (**a**,**b**) are succeeding samples of the acquisition stage. Image (**d**) is a rendering of a rough hand tuned model for the first sample (**a**). The parameters in (**d**) are the initial values for the optimization of (**b**). The result of the optimization is depicted in image (**e**). Images (**c**/**d**/**g**/**h**) visualize the differences of the row or column images (bright areas indicate difference).

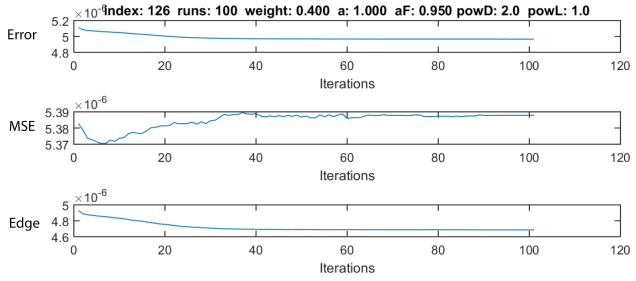

Fig. 11 Progress of the cost function and its components, while best fitting the model depicted in Fig. 10d, e. The first row shows the cost, which is a linear combination of the *MSE* cost (second row) and the *edge* cost (third row). The increase in the MSE after a few iterations is caused by the edge cost. The edge cost ensures that the MSE does not emphasize background noise too much.

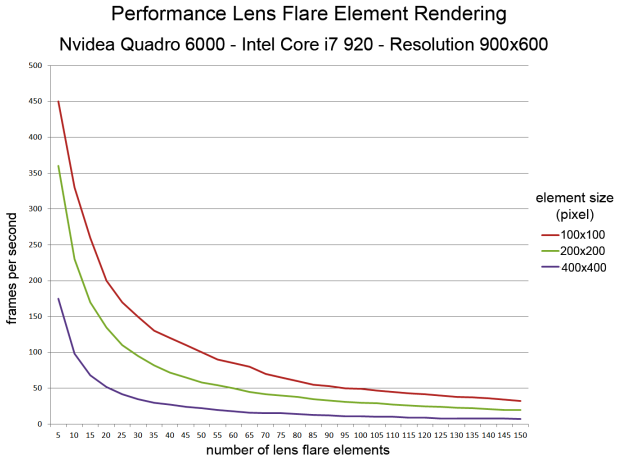

Fig. 12 Rendering performance strongly correlates with the element resolution. Real-time performance can be achieved for more than 100 elements when using smaller resolutions. In real world applications only few lens flare elements are outstandingly large.

5 Conclusion and future work

In this paper we describe a workflow to create a physically plausible lens flare rendering from measurements. The workflow is designed bottom-up in stages (*acquisition – optimization – rendering – prediction*), and allows to stop at a certain stage, if the current result already fulfills the application's goal. In contrast to simulation-based approaches we do not approximate any unknown parameters of the lens system. Our measurement setup is easy to use and camera independent. A flexible lens flare model allows to render various ghosts in real time. If details are not depictable by the current model, it can be easily extended. We present an optimization strategy including a cost function to semi-automatically adjust the parameters of the lens model to increase similarity of our rendering to the captured sample. We present an efficient way to compress the model parameters over a range of samples, while simultaneously creating a continuous description for each ghost.

This workflow describes the fundamental idea how to generate lens flare based on measurements, but there are many possible improvements to increase performance and quality at various stages. We assumed some parameters to be constant (focal length and aperture stop) mainly to reduce the amount of samples necessary in the acquisition stage. By finding the optimal capture settings for a given light source, additional samples required for HDR may become redundant, which could lead to a shorter acquisition time. The restriction regarding the aperture shape can also be loosened by ex-

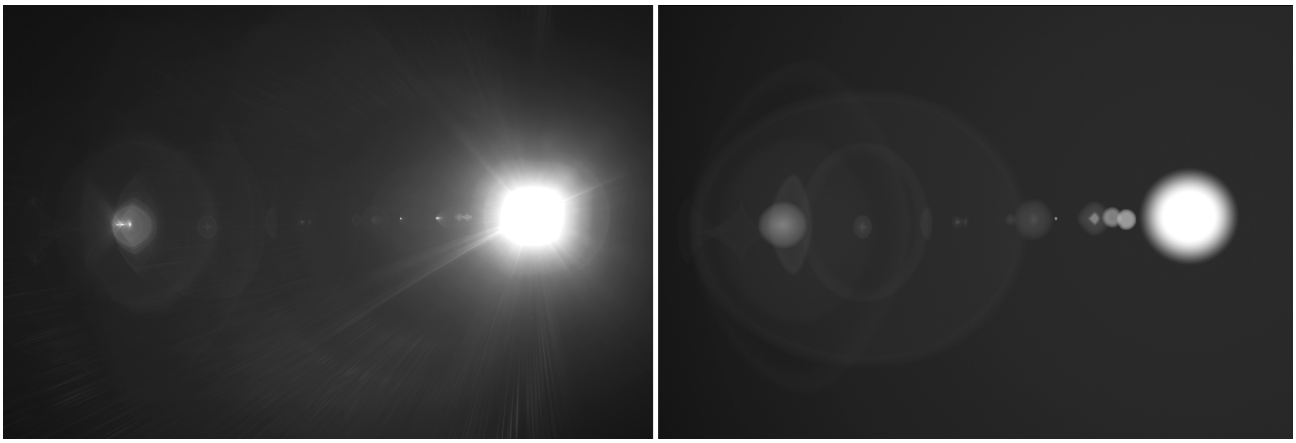

Fig. 13 Complex lens flare generated with our method. Left is the input sample. Right shows the optimized rendering.

tending the lens flare model, but this will introduce additional complexity.

The current optimization stage only supports grayscale or intensity values. Adding color would benefit our system, but would probably make the optimization stage more complex.

To fully reconstruct a captured sample of the acquisition stage, models for the *star-shape* and *halo* lens flare elements have to be developed.

The initial shape parameters of the model could be roughly set by a scan-line algorithm applied onto the sample's edge image. Furthermore, to avoid invalid parameter constellations which break the model design or do not affect the rendering, regularization can be used to "punish" certain parameter changes.

We have explored a new way to compress lens flare imagery by a model-based description.

Acknowledgements We want to dedicate this work to our late colleague Robert F. Tobler.

Compliance with Ethical Standards

Funding: VRVis is funded by BMVIT, BMWFW, Styria, SFG and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174) which is managed by FFG. Conflict of Interest: The authors declare that they have no conflict of interest.

References

1. Alspach, T.: Vector-based representation of a lens flare (2009). US Patent 7,526,417
2. Chaumond, J.: Realistic camera - lens flare (2007). https://graphics.stanford.edu/wikis/cs348b-07/JulienChaumond/FinalProject
3. Christopher M., B.: Pattern Recognition and Machine Learning. Springer (2006)
4. Franke, G.: Physical optics in photography. London: The Focal Press,— c1966 (1966)
5. Hanika, J., Dachsbacher, C.: Efficient monte carlo rendering with realistic lenses. In: Computer Graphics Forum, vol. 33, pp. 323–332. Wiley Online Library (2014)
6. Hennessy, P.: Implementation notes: Physically based lens flares (2015). https://goo.gl/OOmIkB
7. Hullin, M., Eisemann, E., Seidel, H.P., Lee, S.: Physically-based real-time lens flare rendering. ACM Trans. Graph. **30**(4), 108:1–108:10 (2011). DOI 10.1145/2010324.1965003. URL http://doi.acm.org/10.1145/2010324.1965003
8. Hullin, M.B., Hanika, J., Heidrich, W.: Polynomial optics: A construction kit for efficient ray-tracing of lens systems. In: Computer Graphics Forum, vol. 31, pp. 1375–1383. Wiley Online Library (2012)
9. Joo, H., Kwon, S., Lee, S., Eisemann, E., Lee, S.: Efficient ray tracing through aspheric lenses and imperfect bokeh synthesis. In: Computer Graphics Forum, vol. 35, pp. 99–105. Wiley Online Library (2016)
10. Keshmirian, A.: A Physically-Based Approach for Lens Flare Simulation. ProQuest (2008)
11. Kilgard, M.J.: Fast opengl-rendering of lens flares (2000). https://www.opengl.org/archives/resources/features/KilgardTechniques/LensFlare/
12. King, Y.: 2d lens flare. In: M. DeLoura (ed.) Game Programming Gems, pp. 515–518. Charles River Media, Inc., Rockland, MA, USA (2000)
13. Lee, S., Eisemann, E.: Practical real-time lens-flare rendering. In: Computer Graphics Forum, vol. 32, pp. 1–6. Wiley Online Library (2013)
14. Light, I., Magic: Openexr (22014). http://www.openexr.com
15. Mchugh, S.: Understanding camera lens flare from cambridge in colour (2005). http://www.cambridgeincolour.com/tutorials/lens-flare.htm
16. Pixar: The imperfect lens: Creating the look of wall-e. wall-e three-dvd box (2008)
17. Sekulic, D.: Efficient occlusion culling. GPU Gems pp. 487–503 (2004)
18. Steinert, B., Dammertz, H., Hanika, J., Lensch, H.P.: General spectral camera lens simulation. In: Computer Graphics Forum, vol. 30, pp. 1643–1654. Wiley Online Library (2011)
19. Syrp: Genie mini motion controller (2016). https://syrp.co.nz

20. Tocci, M.: Quantifying veiling glare (zemax users knowledge base) (2007). http://www.zemax.com/os/resources/learn/knowledgebase/quantifying-veiling-glare
21. Towell, J.: A brief history of the most over-used special effect in video games: Lens flare (2012). https://goo.gl/244iVo
22. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: From error visibility to structural similarity. IEEE Transactions On Image Processing **13**, 600–612 (2004)
23. Woerner, M.: J.j.abrams admits star trek lens flares are ridiculous (interview) (2009). https://goo.gl/ETgzXW