

On Rendering with Complex Camera Models

Bohumír Zámečník*

Supervised by: Alexander Wilkie†

Faculty of Mathematics and Physics
Charles University
Prague / Czech Republic

Abstract

One of the areas of realistic image synthesis is in modeling cameras. The goal is to provide a visual cue with depth of field and to achieve a photographic look with bokeh (out-of-focus highlights), tilt-shift and optical aberrations of real-world lenses. We provide a comparison of existing methods and fundamental approaches for depth-of-field rendering, including the recent methods, such as the image-based ray tracing. We propose a novel representation of ray transfer within complex lenses suitable for optimizing the ray generation. The open problems in this research area are presented along with sketches of possible solutions.

Keywords: Depth of Field, Lens, Photorealistic Rendering

1 Introduction

One of the areas of photo-realistic image synthesis is the image capture via camera models. Both human eyes and photographic cameras naturally depict some parts of the scene sharply while the rest is gradually blurred. It helps the viewers perceive the spatial arrangement of the scene. Simulating this depth of field as well as other effects caused by more or less complex lens systems in a physically plausible way helps in achieving more realism in rendering.

The paper is structured as follows: In chapter 2 we introduce the desired effects and the camera models used in photo-realistic rendering. Chapter 3 summarizes the most important approaches and methods for depth-of-field rendering and makes a comparison. In chapter 4 we propose a novel representation of ray transfer within complex lens systems. And finally the major open problems we found in this area are presented in chapter 5.

*bohumir.zamecnik@gmail.com

†alexander@wilkie.at

2 Camera models

In order to make useful images of a radiance field in the scene we use models conceptually based on real-world cameras. A *camera* consists of a sensor and an aperture or a lens system. The sensor is usually a rectangular grid of pixels which accumulate the incoming radiance to compute the total radiant energy. The aperture or lens system limits and/or transforms the rays of light going to the sensor in order to make an image of a part of the scene.

The most basic and widespread is model of an ideal pinhole with a point-sized aperture. It implements the perspective projection and produces all-sharp images. In practice a finite aperture is needed to pass enough light in. The solution to provide sharp images is in using refractive lenses able to focus rays emanating from a point in the scene to a point on the sensor so that a contribution of multiple light paths can be integrated.

The idealized model of a refractive lens is the thin lens model. It can be either described by a 1st order approximation of the Snell's law of refraction or by a matrix transformation in homogeneous coordinates. Generally the lenses trade off allowing more light paths for being unable to image the whole scene sharply. In the thin lens model just a single plane is sharp, the *focus plane*, the image of the sensor plane via the lens transformation.

Basically the sensor plane is perpendicular to the viewing direction, the optical axis which intersects its center. Focusing can be done by moving the sensor back and forth in the optical axis. The sensor is called *shifted* in case it is moved laterally (within the sensor plane) and *tilted* if it is not oriented perpendicularly to the optical axis. We call this the camera configuration.

In contrast to most consumer cameras the older view cameras and some special or home-made lenses offer tilt-shift configurations. This enables the photographers to focus on an arbitrary plane or change the perspective, which can be useful for artistic purposes.

Points on the focus plane are projected at points on the sensor, while the rest as sections of the cone of light through the aperture with the sensor plane. For non-tilt configurations of the thin lens model the out-of-focus points are imaged as circles (also called *circles of confusion* – CoC) which leads to blur.

The perceived sharpness depends on the spatial resolution of the sensor. For a fixed camera configuration the CoC radius as a function of position in the scene makes up a 3D scalar field. The isosurfaces of this field mark out the boundary of the *depth of field* (DoF). By limiting the maximum blur amount we get a region bounded by the corresponding isosurfaces and objects within such a region can be considered in-focus.

The intensity of images of out-of-focus points quickly decreases with the amount of defocus as the radiant power is spread over a quadratically larger area. For a very bright point light its out-of-focus image is clearly distinguishable and is called *bokeh* in photographic jargon.

The thin lens model is not capable of producing all the characteristic effects caused by physical design of real-world complex lens systems. Due to physical reasons, technological trade-offs and the usage of real-world materials such lenses might not be always capable of perfect focusing. This leads to optical aberrations and geometric distortions and it also has an impact on the imaging quality including the bokeh quality.

In optical engineering as well as in computer graphics the complex lens systems are described by a sequence of analytical surfaces, their mutual position and materials after each surface [17, 28]. Predominantly are used spherical caps for lens surfaces and circles or other shapes for diaphragms. It is the diaphragm shape which affects the bokeh quality the most.

Other lens properties such as coatings and lens effects such as lens flare (caused by internal reflections and diffraction) are out of the scope of this paper. More information can be found in [28, 15].

3 Methods of DoF rendering

In the ideal pinhole model the light goes through a singular center of projection. In contrast, other lens models allow the light paths to pass through a finite area of the aperture stop. Thus all depth-of-field rendering algorithms when solving the rendering equation [16] or its approximation must additionally integrate the light transfer over this area.

There are two main approaches how rendering algorithms solve visibility, i.e. deciding which scene primitives contribute to each pixel and vice versa. They differ in the order of nested loops over scene objects and image pixels [12]. Object-based algorithms compute the illumination for each object for each pixel and image-based ones conversely. Scan-line rasterization is an example of the first approach while ray tracing and its variants of the latter.

Another criterion to distinguish the rendering algorithms lies in the scene representation. Distributed ray tracing and rasterization belong to the group of rendering algorithms which operate on geometrically represented scenes. On the other hand post-processing methods (together with point-based rendering methods), such as filter-

ing and image-based ray tracing, operate on sample-based representation of the scene [12], eg. layered depth images [26].

Most methods make assumptions on the sensor orientation so that it might be hard to extend them to support tilt-shift configurations. The most flexible in those situations is the plain ray tracing.

For more detailed information on the various DoF rendering methods and camera models used in computer graphics the reader should also consult the existing surveys [2, 3, 10, 4, 19].

3.1 Monte Carlo ray tracing

Ray tracing methods estimate the radiant energy going to a pixel by sampling the radiance along incoming rays. In general the light transfer is recursively evaluated at the points of ray-scene intersection. Although the variants like distribution ray tracing [7], path tracing [16] and others differ in the strategy of tracing rays while evaluating the incoming radiance the ray generation is usually similar. Instead of a singular pinhole a more complex lens model is added between the sensor and the scene [17, 28]. Since the light paths have to pass the lens elements in a known order the complex lens system can be put outside the ordinary acceleration structures. See fig. 1d for an example image from our CPU implementation.

The important thing is that the lens model has a non-zero area which has to be sampled as well. In theory it is the image of the aperture stop, the exit resp. entrance pupil (when looking from the back or from the front). Those two pupils are defined only for on-axis rays. For off-axis rays in ideal thin lenses the pupil remains the same and can be sampled directly. However, in complex lens systems not only the aperture stop can block the light passage which results in the view-dependent effective pupil – projection of the visibility through the lens on a given plane. Sampling the precomputed effective pupil or at least its bounding circle leads to decreasing the amount of rays blocked inside the lens [28] and thus also the image variance. A simpler but not as efficient technique is to sample the whole surface of the outer lens element.

The circular pupils can be sampled by mapping samples from a unit square onto a unit circle with a suitable square to circle mapping [27]. For more complicated aperture shapes this can be combined with rejection sampling.

This approach can give the ground-truth results with no artifacts other than noise and it is thus considered the reference one.

3.2 Multi-view accumulation

The multi-view accumulation method [13] is based on the observation that each view through a single point on the entrance pupil of a thin lens is equivalent to some pinhole projection with an off-axis frustum [6]. Those pinhole views can be then rasterized by the GPU as usual. By

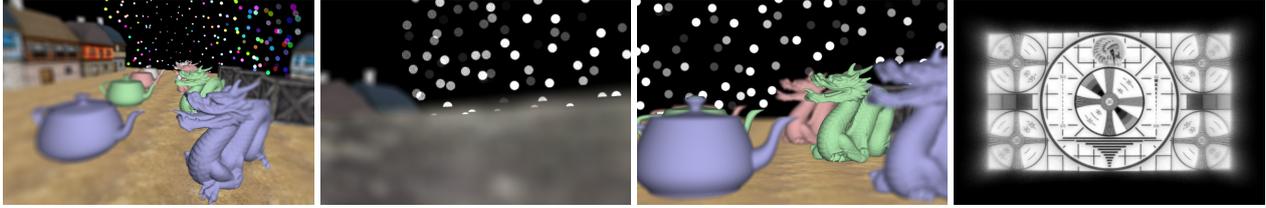


Figure 1: Example images from our implementation of various DoF rendering methods. Left to right: (a) image-based ray tracing with tilt-shift thin lens model, (b) image-based ray tracing handling partial occlusion with bokeh, (c) bokeh in multi-view accumulation, (d) sequential lens ray tracing with a biconvex lens.

sampling the entrance pupil and accumulating the rasterized views the image with depth of field is obtained. The key for the off-axis frusta construction is that they must intersect each other at the image of the sensor (on the focus plane).

The original method used the hardware accumulation buffer (with cca 12-bit integer precision). For accumulating thousands of views (to obtain correct bokeh) rendering to textures (eg. via OpenGL's Frame Buffer Objects) with at least 32-bit floating point precision and manual accumulation is needed [29]. This also allows incremental rendering where the intermediate results are displayed in real-time during the longer convergence. See fig. 1c.

The basic method is very simple but is limited to the thin lens model. Tilted configurations were shown to be supported in [5], unfortunately without any details, and are discussed in section 5. An extension to approximate complex lens systems is described in [14].

Advantages of this method compared to post-processing methods are that it displays all parts of the scene visible from the entrance pupil (not only from its center) and the visibility is already solved (by the z-buffer). A disadvantage is that the entrance pupil is sampled per-image, not per-pixel, so the convergence is slow.

3.3 Layers and their extraction

Before we can describe the DoF post-processing methods themselves we need to learn more about their input data.

In a pinhole image (perspective projection) only the parts of the scene which are directly visible from the center of projection can contribute to the output image. On the other hand for a lens with a finite aperture even parts of the scene which are occluded in the central pinhole view can become visible in other views and thus take part in the output image. Since both the directly visible and occluded parts of the scene cannot be represented in a single image, they must be stored in several layers.

Each image represents a 2D table of samples of the incident radiance function from the scene to the center of projection. Each sample might be then understood as a single light source. Except that the sampled color (or precisely radiance) from the scene is not enough for depth of field rendering since the effect of a light source on the image also depends on its depth. Thus each layer consists

of a color image and a depth image. Usually the layers store the results of frustum transformation normalized to the $[0.0; 1.0]^3$ cube.

The sampled radiance is valid only for a single direction. Assuming that the exitant radiance of scene surfaces does not vary too much when changing the viewing direction a little the sampled radiance can approximate the true radiance from another viewpoint (on the front element of the lens) quite well. This problem can be solved with deferred shading [20] where surface properties are sampled and radiance from given viewpoint is computed later.

There are two approaches in extracting layers from the scene – depth interval layers [25, 1] and depth-peeled layers [11] – each with its pros and cons.

In depth-interval layer extraction the scene is divided into disjoint intervals of depth and each layer contains the surfaces visible in that layer. This results in that the whole images are ordered by depth which could be exploited in some methods.

On the other hand depth peeling [11] produces layers where each pixel is ordered by depth independently. The first layer contains what is visible directly, the second what is hidden after the first layer and so on. This results in fewer layers, since the number of layers is limited only by the depth complexity of the scene. The difficulty is that a patch of pixels from one surface might be interspersed in many layers.

In general the depth peeled layers provide a more compact representation than depth interval layers, since there are fewer empty areas. Thus a lesser number of layers is needed, saving some memory.

The layers can be rendered by a GPU scan-line rasterizer or with a ray tracer modified with additional depth checks, resp. taking k -th intersections instead of the first ones.

For accurate rendering of strong bokeh it is necessary that the color images in the layers are HDR images, eg. represented with floating-point numbers. The resulting output image might be then tone-mapped to LDR.

3.4 Image-based ray tracing

A recent technique to accelerate depth-of-field rendering via a combination of rasterization and ray tracing is the

image-based ray tracing [20, 21]. The main idea is to rasterize a part of the scene visible from the lens and use it as a sample-based representation of the scene for the ray tracing stage. Shading of the visible scene objects is done once and then reused for the views from the many lens samples. Thus the time complexity of the ray tracing stage depends on the image resolution not on the scene complexity. From this point of view the method is most suitable for very complex scenes.

Since the rasterization hardware only supports the ideal pinhole model the view-dependent scene representation is based on a single image from the center of the entrance pupil with the field of view approximated from the original lens model. From the other sample points on the entrance pupil might be visible some parts of the scene "around the corner", so that several layered depth images have to be extracted. The layers contain both color and depth images which are afterwards treated as height-fields and can be intersected instead of the original scene objects.

The two methods differ in the kind of layered depth images used, [20] works with depth-interval layers, while [21] is based on depth-peeled layers. Properties of the layers further determine the algorithms such as height-field intersection or layer extraction and possible camera models. We will describe the second approach. Example images from our GPU implementation of this method can be seen in fig. 1a and fig. 1b.

First the perspective frustum from the entrance pupil center needs to be found, which will be used for sampling the scene into the rasterized layers. For non-tilt configurations and the thin lens model the sensor image defines the frustum shape. Similarly for a shifted sensor an off-axis frustum is constructed. For a tilted sensor a technique referenced in chapter 5 might be used. More complex lenses have to be approximated by thin or thick lens models for this purpose.

Having the frustum matrix the layers can be extracted by depth peeling [11]. Compared to the original method the depth peeling process is easier on current hardware since it is possible to utilize floating-point render textures and programmable shaders. In short, in each iteration the previous layer's depth image is utilized as a secondary z-buffer with an additional depth test to discard the nearer geometry. In practice the depth peeling code can be prefixed to ordinary material shaders. OpenGL's Array Textures can be used for storing the layers. The depth images can be stored separately, packed by four channels into one image for more efficient texture lookup later.

The ray tracing stage then consists of ray generation, intersection with the height-field layers and color accumulation. In case the GPU does not support random number generation we need to provide them in a texture. In practice we tried a 3D texture of size $64^2 \times N$ samples (where N is the number of samples of a single pixel) without excessive artifacts from tiling. The chosen area on the lens should be sampled and the ray transfer within the lens model evaluated as in the ordinary Monte Carlo ray trac-

ing. The rays need to be transformed by the frustum matrix to match the space of the height-field layers.

In the depth-peeled layers we cannot assume that the height-fields are continuous or in disjoint depth intervals. A robust technique for intersecting such height-fields is per-pixel traversal of the ray footprint, ie. the pixels under its orthographic 2D projection, along with a robust intersection test. The 2D version of the DDA algorithm for voxel traversal can be used with some modifications for more robustness in singular cases [29]. Since we treat the height-fields with nearest-neighbor interpolation the intersection tests must be extended by some epsilon tolerance. To reduce memory bandwidth the depth layers can be packed and the intersection test must be modified to work with 4-component vectors.

Several acceleration techniques have been proposed in the original method [21]. First, some geometry does not need be extracted into the layers since it can be shown that it is not visible from any point on the entrance pupil. This extended umbra depth peeling technique can lead to a high speedup, but on the other hand it assumes the thin lens model and makes the height-field treatment more complicated due to the introduction of undefined values. Second, the ray footprint traversal can be accelerated by iteratively clipping the ray extents with the knowledge of minimal and maximal height-field values under the ray footprint. Those values can be efficiently evaluated by N-buffers [9] constructed from the depth layers. Also the N-buffer queries can be done for multiple rays at once.

3.5 Filtering

The filtering approach is based on the concept of point spreading function (PSF) known from Fourier optics where the lens system is considered a linear system. The PSF is an impulse response of the system to a point light. Then the image on the sensor is given by the convolution of the exitant radiance function of the scene with the PSF. However, the model assumes no occlusion, so the visibility has to be solved by other means. Also the PSF is non-constant and depends many factors. When neglecting the wave-optics effects like diffraction the PSF is mostly affected by the aperture stop shape and directly influences the appearance of bokeh.

Most of the methods for interactive depth-of-field rendering are based on the convolution of some PSF kernel with a sample-based scene representation – a rasterized pinhole view. Many of them lead to physically incorrect results and artifacts. One reason is the lack of solving visibility, the other is in using some PSF kernels with an inappropriate convolution method.

It has been recently shown that for convolution with spatially varying kernels there is a difference and duality between gathering and spreading filters [19]. The image formation corresponds to spreading filters and using those PSF in the gathering context causes major artifacts (and vice versa).

For some restricted classes of PSFs there exist acceleration techniques (fast spreading filters) for reducing the convolution complexity from $O(n^2)$ to $O(n)$ or even $O(1)$ with respect to the rasterized kernel radius [19].

The visibility in context of filtering is usually solved by representing the visible part of the scene with multiple layers and alpha compositing them after being filtered. Since the layers are rendered by another technique such as rasterization or ray tracing the filtering methods are denoted as post-processing methods.

3.6 Comparison of ray tracing and filtering

We can compare the two approaches to DoF rendering. Ray tracing methods solve visibility correctly but have problems with optimal sampling. Filtering methods offer optimal sampling but have difficulties with solving the visibility within scene (occlusion).

Getting a large CoC sampled properly in ray tracing requires many samples to reduce noise to tolerable levels, while for in-focus areas a single sample might be sufficient. Unfortunately, for a given pixel on the sensor we are not aware of a method of efficiently generating lens rays ordered by decreasing contribution of light intensity. Eg. importance sampling successfully employed in image-based lighting [24] cannot be readily used due to possibly complex ray transformations within the lens and limited scene visibility from a pixel.

On the other hand spreading filters can rasterize the PSF according to the sensor resolution, spreading the light exactly to the affected pixels without noise present in Monte Carlo methods. The cost is that occlusion has to be solved by other means.

Another view is on supported PSFs. Ray tracing methods can support arbitrary lens models with various PSFs, but have to evaluate the propagation of rays in the lens systems which might be costly. Filtering methods are limited by the complexity of PSFs and their efficient representation, evaluation and spreading.

Ray tracing of the original scene and multi-view accumulation has also the advantage over image-based methods that they provide implicit anti-aliasing via multi-sampling. Methods working with discretized images eg. have no information of exact position of a highlight smaller than a pixel. Thus a bokeh pattern from such a highlight might be slightly translated in the image-based methods compared to more accurate results of the multi-sampling methods. Also CoCs from light sources outside the pinhole field of view might be missing there.

4 Complex lens representation

An ideal thin lens can be fully described just by its focal length and its aperture stop (a circle, polygon, raster bitmap, etc.). For real-world complex lens systems typically a simplified representation must be given. The tradi-

tional representation is followed in [17, 28] – a sequence of spherical caps or planar stops and subsequent materials. The ray transfer is evaluated by sequential ray tracing with analytical intersections and refractions. The complexity is $O(n)$ with respect to the number of elements. Given an incoming ray it is either transformed into an outgoing ray or absorbed inside the lens.

We present a conceptually novel representation of ray transfer behavior of lenses. Rays can pass through a lens either from front to back (along the optical axis) or in the opposite direction, this corresponds to forward and backward ray tracing. The ray transfer in either of the two directions can be treated as a mathematical function, a *lens ray transfer function* (LRTF), which maps incoming rays to outgoing rays and which is defined only for rays that pass. A single lens system can be described by various LRTFs depending on the particular parametrization of rays.

An LRTF can be defined either implicitly and evaluated procedurally as in sequential ray tracing or it can also be sampled into a table and evaluated approximately by interpolation. The ray transfer can be then evaluated in $O(1)$ time with respect to the number of lens elements. Precomputation of the LRTF can be utilized for optimizing the ray generation in both real-time and off-line rendering. Another benefit is the lack of need for specialized intersection routines for different types of lens elements in the evaluation stage. An interesting possibility is in measuring the ray transfer function from real world lenses without being aware of the internal design!

Given some assumptions we propose one of the LRTF parametrizations which we find useful. A ray in the 3D Euclidean space can be parametrized by a 3D position and a 3D direction. For the direction there are in fact only two degrees of freedom (eg. when using spheric coordinates). Similarly the ray position can be related to the outer surface of the lens which is assumed to be a smooth finite 2D surface (rays that do not intersect it cannot pass through the lens). The rays (both incoming and outgoing) can be thus parametrized by four parameters in total.

From the many possible ways of parametrizing the ray position we have chosen the hemispherical coordinates of the ray intersection with the bounding hemisphere over the aperture of the outer lens element surface (eg. the base circle of a hemispherical cap). It is independent on the exact shape of the outer lens surface and can be constructed even when only the aperture radius of the outer lens surface is known. Rotation around the optical axis is simple. The positions of the front and back lens apertures on the optical axis should be taken into account in the transformations for the bounding hemispheres.

In this parametrization we assume the range of ray directions is limited to a hemisphere pointing outwards from the lens. This should be no problem for the majority of lenses except for some fish-eye lenses. The obvious way to represent ray directions would be spherical coordinates, but their drawback is a singularity at the pole which results

in poor sampling of the most important region.

It is better to transform the point on the unit hemisphere (direction) to a unit circle via stereographic projection [8]. The rotation around the optical axis is still easy and the projection is without a singularity.

We define the ray direction to be represented relatively to the intersection position (the position on the unit circle is merely rotated). We can then denote the parametrized ray as (θ, ϕ, s_x, s_y) , where θ and ϕ are declination and azimuth of the position and (s_x, s_y) are stereographic coordinates of the direction; $\theta \in [0; \frac{\pi}{2}]$; $\phi \in [0; 2\pi]$; $s_x, s_y \in [-1, 1]$.

It can be shown that for lenses rotationally symmetric around the optical axis the LRTF in this parametrization is also rotationally symmetric. In particular if we denote the LRTF as L it holds $L(\theta, \phi, s_x, s_y) = L(\theta, 0, s_x, s_y) + (0, \phi, 0, 0)$. This can be exploited to reduce the sampled table dimension to three, with the values remaining a 4-vector.

For the purposes of evaluating the function values from 3D texture we can rescale the parameter ranges to $[0; 1]$. In case of ray absorption the LRTF is undefined and this can be represented with a special value; to minimize discontinuities $(1, 0, 0, 0)$ might be a good candidate.

In summary, the ray transfer through a complex lens system might be described by a sampled LRTF table, radii of the front and back lens element apertures and their distance along the optical axis. The LRTF table can be pre-computed and then utilized to speed up the ray generation phase of ray tracing. A ray has to be procedurally transformed to the parametric form before the LRTF evaluation and back to the standard form afterwards. The concept maps well to the current GPU architectures as each sampled function value can be precomputed in parallel and the evaluation is based on texture lookup and interpolation, the most optimized operations on a GPU.

5 Open problems

Depth of field How to define blur amount and depth of field for tilt-shift configurations? For thin lenses the CoC might be of an arbitrary conic section shape (including infinite-size hyperbolas).

Multi-view accumulation Although it was shown that this method can be extended to support tilt-shift configurations [5] no details were provided. We found the key is still that the frusta from the lens sample point intersect at the image of the sensor which is a general quadrilateral. In order to obtain a rectangular frustum for rasterization a proper orientation of the near plane must be found. A promising way of finding the correct transformation matrices is described in a method from the computer vision area [23].

The method approximating the image from a complex lens system with many blended pinhole views [14] should

be tested again on the modern hardware and compared with the other methods. Anyway, it would probably have similar limitations as the original multi-view accumulation.

Image-based ray tracing A single pinhole frustum might not contain all parts of the scene visible from other viewpoints on the lens. This might be a problem especially for very wide angle lenses and also for near objects. The conditions when the error is significant should be examined.

The depth-interval layer decomposition is dependent on the CoC size which is well-defined only for non-tilt configurations. Given a blur metric for tilt-shift configurations could the method with this kind of layers produce correct results?

Obtaining layered depth images via ray tracing instead of rasterization in the depth-peeling phase should be explored. The sampled scene representation would still allow cheaper intersections for large scenes and rendering the layered depth images would be simpler for tilted configurations as there would be no need for computing suitable frustum matrices.

Lens ray transfer function Better parametrizations of the LRTF should be given. The one proposed here is limited by the range of ray directions and might waste some memory with larger undefined areas. As for the range of directions on a spherical cap larger than a hemisphere the stereographic projection is general enough to produce just a larger circle which can be rescaled to the unit size.

Also the practical LRTF application to accelerate ray generation in a ray tracer should be measured for accuracy and performance.

The LRTF might be measured from real-world lenses for which a method and an apparatus need be constructed. This could allow eg. to match the lens characteristics of a real film footage with synthesized images without knowing the exact internal lens design.

By changing the aperture stop size or shape only the domain of the LRTF is affected not the values. Thus a variable-sized diaphragm even with an asymmetric shape could be represented outside the LRTF table. Also the properties of the LRTF domain might be exploited for a more compact representation (eg. a boundary of a compact region).

Does the LRTF provide enough information for lens flare rendering or what additional information is needed to produce correct results without having the original lens design?

Can the LRTF model be extended to support movable element groups (zoom lenses, focusing by group movements) without an excessive memory usage?

Currently, this LRTF parametrization assumes ray transfer at a single wavelength which would lead to a lack of chromatic aberration. Would it be possible to exploit some

coherence to support wavelength dependence without increasing the dimensionality of the precomputed LRTF table?

Rendering with wavefront aberrations was shown in the filtering approach [1]. Could they be used to represent ray transfer in complex lenses? As the list of Zernike polynomial coefficients directly represents the amount of the various optical aberrations this representation might enable modifying the behavior of existing lenses and synthesizing new ones without the need to have a geometrical design. Also the memory consumption could be quite low.

Lens sampling Sampling of the effective pupil of complex lenses should be revised for real-time rendering. In [28] the pupil is precomputed for each pixel on the sensor which is valid only for a single sensor plane. A more compact representation of an approximate effective pupil function for an arbitrary sensor plane should be explored, along with its precomputation and usage for sampling. The assumption of rotation symmetry can again be used for dimension reduction.

Filtering and PSFs The usage of PSFs of complex lens systems with filtering methods (representation, precomputation, evaluation, etc.) should be further explored.

The comparison of the fundamental approaches leads to a question for a hybrid method of ray tracing and filtering which would offer optimal sampling while computing visibility easily. One of the possibilities might be in modifying the PSF kernel on the fly – invisible parts (decided by ray tracing) would be “cut off” from the rasterized PSF kernel, then it would be differentiated and spread.

Is it possible to modify spreading filters to work with physically correct PSFs of tilt-shift configurations? Note that the rotated sensor plane may result in infinite conic section PSFs even for a simple thin lens and also the radiometric situation with natural vignetting is more complicated.

Can PSFs in polynomial spreading filters be represented by orthogonal polynomials (eg. Zernike polynomials) with strictly limited values in order to suppress numerical precision problems?

It should be verified whether fast spreading filters are really compatible with solving visibility by per-pixel layers using depth-peeled image layers. Their advantage is in requiring fewer layers compared to depth-interval layers. Per-pixel layers [22] seem to be suitable for spreading filters since in the original method a kind of spreading was done, albeit by different means. Unfortunately, [18] only mention per-pixel layers without providing details whether they in fact used per-image or per-pixel layers.

Per-pixel layers do not seem to be fully compatible with precomputed rasterized PSF differences since some per-pixel computations has to be made. Note that for deciding the output for a pixel of the rasterized CoC we need to know the source and destination pixel depths. The in-

formation on source pixels cannot be restored during the phase of spreading of PSF differences. A way to combine those two methods which seems to be possible is the following procedure:

1. take the original rasterized PSF
2. for each of its pixels make decision to which output layer it should go, which produces three clipped PSFs
3. differentiate each of them
4. spread each of them into the corresponding layer

I.e. spreading could be done after deciding the output layer and for each part of the PSF separately.

6 Conclusion

In this paper we have presented the various approaches to interactive physically-based depth-of-field rendering and compared them in several aspects. We have proposed an alternative representation of the ray transfer behavior of complex lens systems which might be useful for accelerating the ray generation in ray tracing methods. Most importantly we have shown many open problems in the area of rendering with complex camera models which could serve as a basis for further research.

7 Acknowledgments

I would like to thank to Alexander Wilkie and Josef Pelikán for their support.

References

- [1] Brian A. Barsky. Vision-realistic rendering: simulation of the scanned foveal image from wavefront data of human subjects. In *Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, APGV '04, pages 73–81, New York, NY, USA, 2004. ACM.
- [2] Brian A. Barsky, Daniel R. Horn, Stanley A. Klein, Jeffrey A. Pang, and Meng Yu. Camera models and optical systems used in computer graphics: part i, object-based techniques. In *ICCSA'03: Proceedings of the 2003 international conference on Computational science and its applications*, pages 246–255, Berlin, Heidelberg, 2003. Springer-Verlag.
- [3] Brian A. Barsky, Daniel R. Horn, Stanley A. Klein, Jeffrey A. Pang, and Meng Yu. Camera models and optical systems used in computer graphics: part ii, image-based techniques. In *ICCSA'03: Proceedings of the 2003 international conference on Computational science and its applications*, pages 256–265, Berlin, Heidelberg, 2003. Springer-Verlag.

- [4] Brian A. Barsky and Todd Kosloff. Algorithms for rendering depth of field effects in computer graphics. *Proceedings of the 12th WSEAS international conference on Computers 2008*, 2008.
- [5] Brian A. Barsky and Egon Pasztor. Rendering skewed plane of sharp focus and associated depth of field. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 92, New York, NY, USA, 2004. ACM.
- [6] Paul Bourke. Offaxis frustums - opengl. July 2007.
- [7] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18:137–145, January 1984.
- [8] H.S.M. Coxeter and H.S.M. Coxeter. *Introduction to geometry*. Wiley Classics Library. Wiley, 1989.
- [9] Xavier Décoret. N-buffers for efficient depth map query. *Computer Graphics Forum*, 24(3), 2005.
- [10] Joe Demers. *GPU Gems*, chapter Depth of Field: A Survey of Techniques. Addison-Wesley, 2004.
- [11] Cass Everitt. Interactive order-independent transparency, 2001.
- [12] Markus Gross and Hanspeter Pfister. *Point-Based Graphics (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
- [13] Paul Haeberli and Kurt Akeley. The accumulation buffer: hardware support for high-quality rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 309–318, New York, NY, USA, 1990. ACM.
- [14] Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel. An image-based model for realistic lens systems in interactive computer graphics. In *Proceedings of the conference on Graphics interface '97*, pages 68–75, Toronto, Ont., Canada, Canada, 1997. Canadian Information Processing Society.
- [15] Matthias Hullin, Elmar Eisemann, Hans-Peter Seidel, and Sungkil Lee. Physically-based real-time lens flare rendering. *SIGGRAPH 2011*, 2011.
- [16] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20:143–150, August 1986.
- [17] Craig Kolb, Don Mitchell, and Pat Hanrahan. A realistic camera model for computer graphics. 2002.
- [18] Todd Kosloff, Michael Tao, and Brian A. Barsky. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. *Graphics Interface 2009*, 2009.
- [19] Todd Jerome Kosloff. *Fast Image Filters for Depth of Field Post-Processing*. PhD thesis, EECS Department, University of California, Berkeley, May 2010.
- [20] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics (Proc. ACM SIGGRAPH ASIA)*, 28(5):1–6, 2009.
- [21] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Real-time lens blur effects and focus control. *ACM Transactions on Graphics (Proc. ACM SIGGRAPH'10)*, 29(4):65:1–7, 2010.
- [22] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. Real-time depth-of-field rendering using point splatting on per-pixel layers. *Computer Graphics Forum (Proc. Pacific Graphics'08)*, 27(7):1955–1962, 2008.
- [23] Kok-Lim Low and Adrian Ilie. View frustum optimization to maximize object's image area, 2001.
- [24] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
- [25] Cary Scofield. 2 1/2—d depth-of-field simulation for computer animation. pages 36–38, 1992.
- [26] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 231–242, New York, NY, USA, 1998. ACM.
- [27] Peter Shirley and Kenneth Chiu. A low distortion map between disk and square. *journal of graphics, gpu, and game tools*, 2(3):45–52, 1997.
- [28] Benjamin Steinert. Simulation of real photographic phenomena in computer graphics. Master's thesis, Universitat Ulm, 2009.
- [29] Bohumir Zamecnik. Interactive preview renderer for complex camera models. Master's thesis, Charles University in Prague, Faculty of Mathematics and Physics, December 2011.