Real-Time Dynamic Bokeh Rendering with Efficient Look-Up Table Sampling

Yuna Jeong, Seung Youp Baek, Yechan Seok, Gi Beom Lee, and Sungkil Lee

Abstract—This article presents a real-time bokeh rendering technique that splats pre-computed sprites but takes dynamic visibilities and intrinsic appearances into account at runtime. To attain alias-free looks without excessive sampling on a lens, the visibilities of strong highlights are densely sampled using rasterization, while regular objects are sparsely sampled using conventional defocus-blur rendering. The intrinsic appearance is dynamically transformed from a precomputed look-up table, which encodes radial aberrations against image distances in a compact 2D texture. Our solution can render complex bokeh effects without undersampling artifacts in real time, and greatly improve the photorealism of defocus-blur rendering.

Index Terms—real-time rendering, bokeh, depth of field, defocus blur, GPU

1 Introduction

Bokeh in photography refers to the way a lens system renders defocus-blurred points (in particular, strong highlights) in terms of Circle Of Confusion (COC) [1]. Aesthetic look of the bokeh is a crucial element in photorealistic defocus-blur rendering. The appearance of the bokeh is characterized by intrinsic factors of optical systems, including the shape of a diaphragm and aberrations (e.g., radial distortion, optical vignetting, and dispersion). Also, dynamic *partial visibilities* of objects are important for realistic bokeh rendering, as they matter in defocus-blur rendering [2].

Multiview sampling techniques [3], [4], [5] that distribute rays from a lens can naturally render high-quality bokeh effects. However, strong highlights in the bokeh dominate the integration of incoming rays, and force to require very dense sampling to avoid ghosting. For instance, a 256^2 bokeh pattern requires up to 64K lens samples to reveal full details, which is far from real-time efficiency.

Real-time rendering has been using the scatter-based splatting of predefined sprites to express the bokeh effects [6], [7], [8], [9], [10]. Realistic patterns can be obtained through photography or physically-based rendering [11], [12], [13], [14]. Nonetheless, the static nature of the sprites impedes the integration of visibility and aberrations, being limited only in simple non-physical patterns. Our work tackles these challenges for real-time high-quality bokeh rendering.

In this article, we present a novel real-time bokeh rendering technique that splats pre-computed sprites for performance, but integrates visibilities and aberrations as well to obtain high-quality dynamic appearances. We improve bokeh rendering in terms both of *extrinsic* (scene-related) and *intrinsic* (relating solely to optical systems) appearances. Unlike the previous distributed techniques, we decouple the visibility sampling of strong *highlights* as *bokeh sources* and the rest (Fig. 1). Regular objects go through sparse sampling, but the highlights are precisely tested for their visibilities using projective rasterization from highlights onto a lens. We thereby attain accurate screen-space visibilities, which exhibit full details of bokeh textures. For the intrinsic factors, we propose a compact 2D lookup table (LUT) and its effective parametric transformation scheme for high-dimensional aberrations. The LUT encodes only radial aberrations against image distances, which are physically ray-traced through optical systems. Our texture-parameter transformation scheme based on the thick-lens model obtains non-trivial patterns, giving more controllability beyond the physically-based ray tracing.

Our major contributions can be summarized as:

- an efficient accurate visibility sampling technique for bokeh rendering and its acceleration techniques;
- a compact bokeh LUT encoding and its effective parametric transformation scheme for spatial complexities.

- *Y. Jeong is with Korea Institute of Science and Technology Information (KISTI), Daejon, Republic of Korea. E-mail: jeongyuna@kisti.re.kr*
- *S. Baek, Y. Seok, G. Lee, and S. Lee are with Sungkyunkwan University (SKKU), Suwon, Republic of Korea. E-mail: {bsy6766, yechan, gibeom, sungkil}@skku.edu*

Manuscript received March 16, 2020; revised January 0, 2020.

2 Related Work

We briefly review previous studies on bokeh modeling and rendering with respect to extrinsic and intrinsic appearances.

Extrinsic appearances of bokeh (e.g., COC size and visibility) can be naturally handled by sampling-based defocus-blur rendering techniques such as distributed ray tracing [3], [5] or accumulation buffering [4]. The classical techniques, however, use the thin-lens model [15], and their results are not entirely physically faithful. Ray tracing through optical systems can exhibit more accurate spectrospatial aberrations [11], [12], [13]. However, they require very dense sampling to avoid aliasing, and hence, have been used in offline processes. Also, their expressive power is constrained by physical limits.

Sprite-based rendering is suitable to express aesthetic intrinsic aspects of the bokeh [6], [16]. The bokeh texture can be pre-generated in many ways such as photography [17], [18], manual drawing [19], or ray tracing [11], [12], [14], [20]. Splatting is common for real-time graphics-processing-unit (GPU) rendering, and can represent fine details well. In contrast, gather-based approaches [9], [10], [21], [22], the

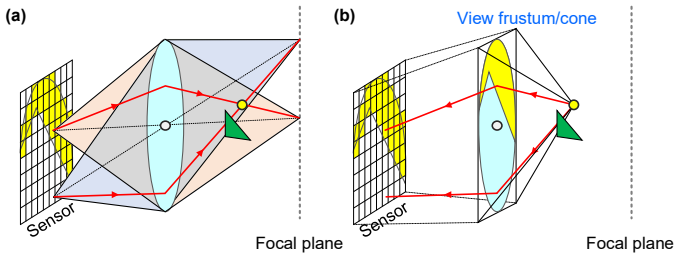


Fig. 1: *General defocus-blur rendering integrates all the rays through the lens (a), but wastes samples when needing only highlights (the red lines). In contrast, our projection from the highlight obtains precise visibility without redundancy.*

other majority, is significantly limited due to the nature of sparse/separable convolutions. The appearance is not associated with the scene in particular for visibilities, and the static pattern does not reflect dynamic aberrations. Hence, their quality has been far from realistic bokeh effects.

Our solution simultaneously tackles precise visibility, which has been available from offline ray tracing, and dynamic aesthetic aspects in the splatting. We use a compact 2D LUT for real-time performance, but involve dynamic aberrations at runtime using texture-parameter transformation. For visibility evaluation, we decouple the dense sampling of bokeh sources from regular objects. Thereby, our solution avoids redundant dense sampling for regular objects, and can be used even in defocus-blur postprocessing.

3 System Overview

The overview of the rendering pipeline of our work is shown in Fig. 2. Our system follows the basically same approach as a conventional real-time postprocessing does, where a static bokeh texture is splatted onto the regular defocus-blurred image at the locations of highlights detected from a pinhole image [23]. Our solution is largely distinguished from the conventional ones for its dynamic visibility sampling and dynamic bokeh texture generation with a pre-generated bokeh profile. We explain more details in what follows.

We generate an intensity profile texture offline (Sec. 5.2), and prepare aperture textures, depending on f-number. In the online stage, we first prepare multi-layer pinhole-image textures encoding colors, depths, and object indices every frame. We then generate regular defocus blur [5], on which the bokeh sprites are splatted.

The online bokeh rendering first extracts highlights (Sec. 4.1). We then generate visibility mask textures (Sec. 4.2). Finally, we splat bokeh textures using the LUT (Sec. 5.2) and its parameter transformation (Sec. 5.3). The center of the sprite is its position as seen in the main view, and the size is its COC diameter. Then, we apply visibility masking, aperture shapes, and other intrinsic appearances (Sec. 5.4). If the source is a clustered source where multiple point sources are merged, it is blur-filtered to achieve a smoother look. Finally, the bokeh rendering is blended onto the regular defocus-blurred image to produce the final output.

4 Visibility Sampling

Bokeh is observed differently by varying visibilities between the camera and objects. Partial occlusions with blurry objects

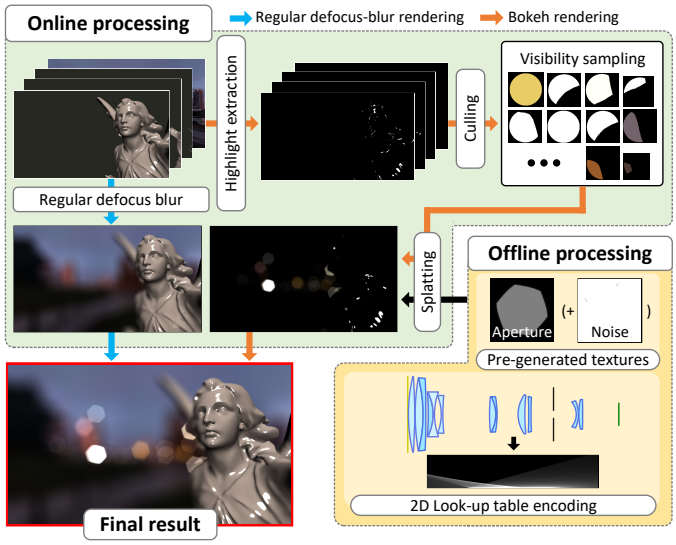


Fig. 2: *Overview of our rendering pipeline.*

require evaluating scene-dependent visibilities. Background blur against focused objects can be depth-masked [7], [8], but leads to a wrong approximation. The previous distributed techniques [3], [4], [5] can easily solve this problem, but their too dense sampling is very costly.

To this end, we introduce an accurate screen-space visibility sampling for highlights. Given highlights extracted from a scene, we first derive a point-source model and approximate the model with clustering. Then, we explain culling and warping to improve performance.

4.1 Highlight Extraction

To extract highlight pixels, we examine strong-intensity pixels of the multi-layer pinhole images that above a constant threshold (the maximum value that can be represented in a pixel; e.g., 1). High-dynamic-range (HDR) inputs need to be provided. When low-dynamic-range (LDR) inputs are only available, we convert them to HDR inputs. Learning-based great techniques exist to this end [24], [25], but are too costly for real-time rendering (e.g., 2 s/frame [24]). Instead, we simply amplify LDR intensities, which is empirical yet effective for our purpose. In our implementation, regular objects are linearly scaled (e.g., $10\times$) only for specular shading. The background color is scaled by $1+\alpha L^{10}$, where L is a luminance and α is a user-defined scale (e.g., 100–200).

When decoupling the regular defocus-blur and bokeh rendering, the total energy should be kept. Therefore, after extracting highlights, we remove their intensities from the input images. However, a simple removal leads to noticeable aliasing (dark-dot ghosts) due to the difference of sampling densities. To avoid this, we set the new (after-removal) intensity of a highlight as the average of the nearby non-highlight pixels (e.g., eight neighbors), and use only the difference from the original intensity as inputs.

4.2 Model

4.2.1 Point-Source Model

The visibility sampling in the bokeh rendering naturally follows the same principle as done in general defocus-blur

TABLE 1: *Culling methods and handling of different conditions.*

Method	Targets	Metric	Vis. sampling	Splatting
COC	Highlights	Small COC	×	×
Occlusion	Highlights	Occluded	×	×
		Fully visible	×	○
VV	Objects	All objs. culled	×	○

rendering [3]; we also follow the sampling-based approach but with rasterization. Given a point $\mathbf{q}$ on the sensor, its intensity $I(\mathbf{q})$ is the integration of the visible rays that pass through the lens system to the sensor, expressed as:

$$I(\mathbf{q}) = f(\Psi(\mathbf{q}), \Omega), \tag{1}$$

where $\Psi(\mathbf{q})$ is the set of outgoing (leaving the lens system) ray samples that originate from $\mathbf{q}$, and Ω is the set of scene objects. f is a discrete integration function defined as:

$$f(\Psi, \Omega) := \sum_{\mathbf{r}\in\Psi} \sum_{\mathbf{p}\in\Omega} L(\mathbf{r},\mathbf{p}) v(\mathbf{r},\mathbf{p}), \tag{2}$$

where $L(\mathbf{r},\mathbf{p})$ and $v(\mathbf{r},\mathbf{p})$ are the radiance and the binary visibility function (having 1 or 0) of $\mathbf{p}$ with respect to $\mathbf{r}$. Here, for every $\mathbf{r}$ (corresponding to a view in the multi-view rendering), the visibilities of all the objects require being evaluated. As already alluded to, this classical way needs very dense sampling of the rays to avoid aliasing.

Our idea to cope with the problem is decoupling the sampling density of the *highlight pixels* (potential bokeh sources) from those of regular objects so that only the visibilities for the highlights are sampled densely. Let the sole set of highlights be Θ. Also, let another (denser) set of rays be $\Phi(\mathbf{q})$. We then derive the following decomposition:

$$I(\mathbf{q}) = f(\Psi(\mathbf{q}), \Omega - \Theta) + f(\Phi(\mathbf{q}), \Theta). \tag{3}$$

Eq. (3) gives us a chance to greatly improve the bokeh rendering. Firstly, we can focus on improving the evaluation of visibilities of the highlights without involving regular objects. Secondly, the appearance of the bokeh becomes controllable for quality-performance tradeoff. For instance, we can use denser sampling only for the bokeh and employ additional acceleration techniques.

Despite this decoupled sampling, the regular objects ($\Omega - \Theta$) are still involved in the visibility evaluation for bokeh (Θ), because they can occlude highlights. This poses a challenge. Whereas only the red lines (to the highlights) are of our interest (Fig. 1(a)), we still have to test intersections against all the other objects. This is obviously too costly, and is also prone to be erroneous unless dense sampling is used.

To solve this, we introduce a *forward batch visibility sampling*. The visibility of a highlight is only related to rays in the cone formed by the highlight and the lens; see Fig. 1(b). Then, a forward tracing from the highlight only considers effective potential occluders. This forward ray casting can be equivalent to a rasterization on the frustum conservatively enclosing the cone, enabling a GPU-based projection. In this way, we can attain virtually full visibilities without aliasing, keeping sparse sampling for regular objects.

The efficiency of our visibility sampling stems from less dependency to ray samples and more dependency to the complexity of highlights. Let the COC radius of a bokeh be C.

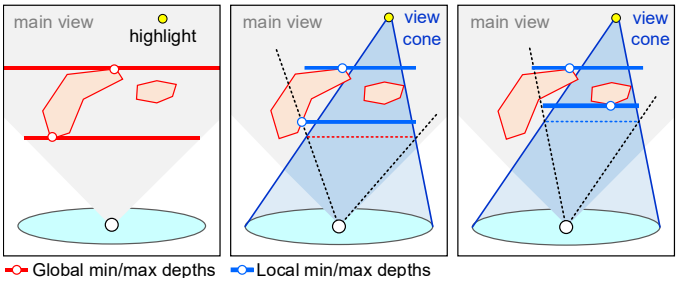


Fig. 3: *Iterative refinement of depth bounds for occlusion culling.*

For example, given $C = 128$, the classical approach requires up to 256^2 lens samples per pixel for full visibility sampling; though, a bokeh texture resolution is typically higher than 256^2. In contrast, ours requires evaluating only $|\Theta|$ (number of highlights) samples. When $|\Theta|$ is much less than C^2, ours becomes much more efficient. In addition, we can reduce the cost for visibility intersection test similarly to the occlusion and view-frustum culling (view frusta for highlights are typically very small) and aggressive approximations.

4.2.2 Clustering-Based Approximation

When there are too many highlights in the scene, brute-force repetition for visibility evaluations can be costly, losing the benefit of our approach. Merging similar point sources into groups can be a reasonable approximation for reducing the number of effective visibility evaluations.

We employ mipmapping to hierarchically merge highlight pixels in the screen space. A highlight map, having non-zero highlight pixels, initializes the finest mipmap level ($l = 0$), and the reduction is repeated (e.g., 4 times).

For each reduction level l, 2×2 input texels end up with a single output texel. To avoid handling too many combinations, we run horizontal and vertical passes successively. Thereby, each pass handles only two input texels. When more than one texels are invalid (non-highlight), we send only a valid pixel (if any) to the next pass (vertical pass or the next level). Otherwise, we merge similar texels. When two texels are still distinct, we store one (e.g., one farther from camera) to a result list, and send the other to the next pass.

Similarity for the grouping is measured in terms of object-space distance, since neighboring objects are likely to share visibilities. A similarity threshold at mipmap level l is defined as $\tau(l) = k_\tau 2^l$, where k_τ is a scene-dependent scale. The grouping averages object-space positions, keeps a maximum inter-object image-space distance, and accumulates intensities. Then, the attributes of a merged source are treated as if a single source. Further, filtering with their image-space parallaxes can be applied to achieve smooth boundaries.

When temporal coherence matters (e.g., animation), we additionally cluster the outcome of the mipmap reduction, which iteratively merge them with global pairwise comparison ($O(n^2)$). We again use object-space distance as similarity threshold. This further reduction is effective for moderate number of sources, which alleviates temporal incoherence.

4.3 Culling

Our visibility sampling involves multi-pass scene rasterization, which repeats for every highlight. This can be non-

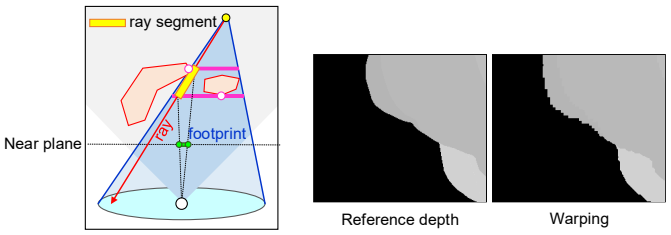


Fig. 4: *Visibility-map warping using ray tracing [5] and examples.*

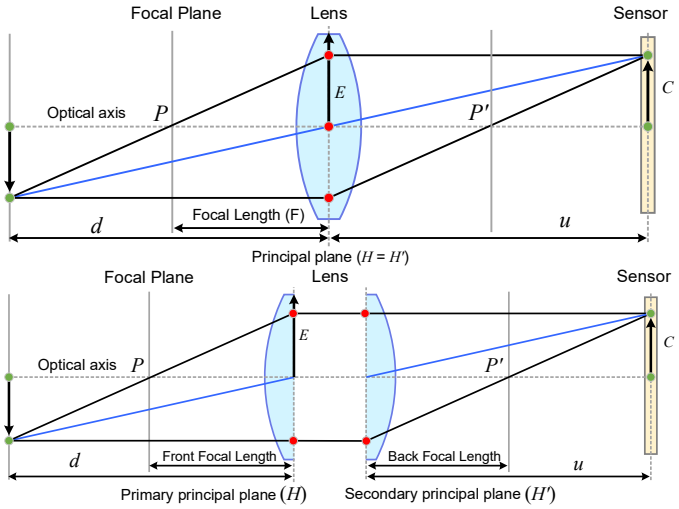


Fig. 5: *The thin-lens (above) and thick-lens (below) models [28].*

trivial for many highlights, but fortunately, there are many redundancies that can be avoided. Here, we introduce three culling techniques. The COC-size culling and occlusion culling (OC) test whether the highlights' visibility maps need to be drawn, and the view volume (VV) culling reduces the number of objects that are included in the visibility-map rendering. Table 1 summarizes the culling metrics, test targets, and how to handle each condition.

4.3.1 COC-Size Culling

Highlights nearby the focusing plane have smaller COC sizes than the rest. Fewer samples suffice for them, which can be covered by the regular defocus-blur rendering. Hence, excluding them from the decoupled visibility handling could improve performance. Precisely, given the regular lens samples N and the screen-space COC radius C of a highlight, when $\pi C^2 \leq N$, the visibility sampling can safely ignore them (e.g., $C \leq 9$ for $N = 256$). Let a set of such highlights be Π. We derive a better decomposition from Eq. (3):

$$I(\mathbf{q}) = f(\Psi(\mathbf{q}), \Omega - \Theta + \Pi) + f(\Phi(\mathbf{q}), \Theta - \Pi). \tag{4}$$

Note that involving Π hardly affects the performance of the regular defocus-blur rendering. Actually, our implementation renders even the geometries of the highlights, and just discards their colors to avoid their (tricky) removal within regular objects.

4.3.2 Occlusion Culling

Fully occluded (by objects) or *fully visible* bokehs do not need to evaluate *partial* visibilities; though, the fully visible bokehs are directly splatted for the final output. This idea leads to great reduction in the number of effective visibility maps.

For the occlusion test, we use an image-based occlusion query within the view frustum (VF) $\mathscr{F}$ of the bokeh (Fig. 3). The full occlusion is found when the highlight is farther than maximum depth within $\mathscr{F}$, and the fully visible highlight when closer than the minimum depth.

We find the min/max depth bound in $\mathscr{F}$ similarly to [26] (Fig. 3). To avoid the exhaustive rendering of all the occluders, we reuse the G-Buffer of the main-view rendering (for the final output), which is readily available in the deferred rendering. We first find a global depth bound $(d^0_{\min}, d^0_{\max})$ in the G-Buffer, where $d^0_{\min}$ is fed as a seed for the iteration below. We iterate to tighten the local depth bound:

$$(d^i_{\min}, d^i_{\max}) \leftarrow \mathscr{M}(\mathscr{F}, d = d^{i-1}_{\min}), \tag{5}$$

where $\mathscr{M}$ finds min/max values within the screen-space projection (in the G-Buffer) of the intersection of $\mathscr{F}$ and the plane $(d = d^{i-1}_{\min})$. Iterating 2–3 times suffices in practice.

To query local min/max depth values in a screen space quad, we use the N-buffers [27], which contain min/max depth values in a square area of $2^i \times 2^i$ in texture T_i, and tiling them finds min/max for any rectangular shapes [26].

4.3.3 View-Volume Culling

When a highlight passes both of COC-size and occlusion culling, it is likely to see objects in its view and its *partial* visibility map needs to be drawn. Here, we further cull the objects outside its VV to the lens. Since the lens is much smaller than the scene, the views at highlights have narrow fields of view, which potentially cull many objects in complex scenes and accelerates visibility-map rendering.

The standard VF culling (VFC) can be applied here, but we only care about *View Cone* (VC; see Fig. 1b)). Thus, objects inside VF but not in VC can be further culled. Thereby, we perform VC culling (VCC), which is tighter than the VFC.

To efficiently cull many objects, we perform the VCC in GPU, and also skip tests for objects outside the main view (for the final result). When all the scene objects turn out to be culled for a specific highlight, we bypass its visibility-map rendering; this occurs, because the OC is more conservative.

4.4 Warping

Despite the efficiency achieved by our culling techniques, the geometry rasterization passes for the visibility map can be still costly for complex scenes. We here present an alternative technique, which uses the image-based ray tracing [5] to generate visibility maps without actual geometry passes.

We start from the depth-peeled G-Buffer [5], [29]. Primary rays are defined from the highlight to lens samples (Fig. 4). The ray's projection onto the image plane of the main view defines two screen-space endpoints. Within the screen-space footprint, we trace back to front (farther to closer in depth), and find the nearest intersection; see [5] for details.

A local depth bound can define a shorter ray segment, which reduces the cost for long footprints. Finding the local depth bound is exactly the same as used in our OC, and thus, we re-use the local depth bound calculated for the OC. Fig. 4 compares the warping result and a reference depth map.

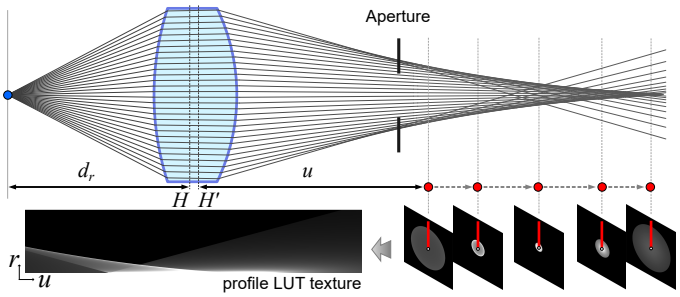


Fig. 6: *Ray tracing-based profiling of the bokeh textures along image distances and its encoding into a compact 2D texture.*

5 Bokeh Appearance Rendering

Reproduction of realistic bokeh effects also involves intrinsic characteristics of optical systems, including aberrations, dispersion, distortion, diffraction, and optical vignetting. In this section, we present how to achieve such effects, keeping bokeh splatting lightweight. We first revisit the lens models that we use. Then, we describe how to encode ray-traced patterns into a look-up table (LUT), the parameters of which are transformed for flexible runtime deformation.

5.1 Background: Lens Models

In geometric optics, the relationship of a focal length F, an object distance d ($> F$), and an image distance u ($> F$) is defined by the Gaussian lens formula:

$$d(u) = uF/(u-F) \quad \text{and} \quad u(d) = dF/(d-F). \tag{6}$$

The classical thin-lens model assumes rays refract once (Fig. 5), and d and u are defined from the single *principal plane*. Given the effective lens radius E and the focusing distance d_f (for objects to appear sharp), an image-space COC radius C for d [15] is:

$$C(d,d_f) = \left(\frac{EF}{d_f - F}\right)\left(\frac{d-d_f}{d}\right). \tag{7}$$

The thick-lens model better abstracts multi-element optical systems, where refraction occurs twice at the *primary principal plane* (H) and the *secondary principal plane* (H') [28] (Fig. 5). Parallel incoming rays from the scene refract at H' and focus at the focal point P', and parallel rays coming from the sensor refract at H and focus at the focal point P. The distance from H' to P' is referred to be a Back Focal Length (BFL), and that from H to P to be a Front Focal Length (FFL). Since we are interested in objects in the scene, we use BFL as an Effective Focal Length (EFL). H and H' can be found at the intersections from rays leaving the last element to the parallel primary rays (from the sensor and entrance planes for H and H', respectively) [30]. Since d and u are defined from where refraction occurs (i.e., the principal planes), d and u are defined to be $z-H$ and $H'-z$, respectively [31], where z is a displacement of the object from the origin. Then, we can still derive the COC size using Eq. (7).

In our case, the thick-lens model is used to encode ray-traced profiles of optical systems into an LUT. Once the profile is built, we use thin-lens model for rendering to integrate the profile to the existing thin-lens based applications.

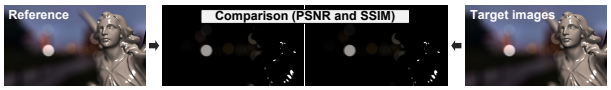


Fig. 7: *Bokeh-only quality comparison.*

5.2 Compact 2D Look-Up Table Encoding

Analytical synthesis of optical patterns is convenient [32], but often infeasible for complex patterns. An alternative is physically-based ray tracing, which can capture natural appearances [11], [12], [13], [14], [20], [30], [33]. We also trace rays similarly to the previous work, but store the intensity profile into an LUT texture. At runtime stage, we obtain more complex patterns with LUT parameter transformations.

A faithful profile requires iteration with high-dimensional parameters. What matters in our observation includes 7-D parameters: an object position $\mathbf{p} := (x,y,d)$, a lens sample $\mathbf{v} := (s,t)$, a focal depth d_f, and wavelength λ. Obviously, brute-force measurement may suffer from excessive memory and be GPU-unfriendly. Hence, dimension reduction is crucial for compact storage as well as run-time performance.

Our approach is encoding the profile in a compact 2D texture and deriving a mapping to the texture coordinate for the others. The LUT is built for the full-lens model, but for the mapping to be controllable without a ray tracer, we base our parametric mapping on the thick-lens/thin-lens model.

Precisely, we consider objects only on the optical axis (i.e., $x=0, y=0$), and batch-cast rays from $\mathbf{p}$ to all the (s,t) samples on the entrance pupil; non-zero (x,y) is approximated in the distortion (Sec. 5.4). Then, given $\mathbf{p}$, images on the sensor side appear no more points, but COCs whose intensity distributions depend on the image distance. Since the image is usually radially symmetric (except for anamorphic lenses), we can further simplify the 2D profile into 1D lines (see Fig. 6); the 1D lines are functions of radial screen-space offset r of sprite pixels from the center of the sprite.

We measure the sensor-side images with respect to a *fixed reference focal point* at $\mathbf{p}_r = (0,0,z_r)$; the object distance $d_r = z_r - H$, where $z = H$ is the the primary principal plane. Then, we end up with a 2D texture that encodes 1D profiles along the image distance u (Fig. 6). u is densely (e.g., 1024) sampled in $[-z_l, z_l]$, where z_l is the distance from the last optical element to the sensor ($z=0$). When $z_r = \infty$, the least COCs are located around the sensor, but $d_r = \infty$ is difficult to handle the parametric mapping with a finite d. So, we choose a finite d_r (e.g., 1 meter away from H). Note that alternative measurement for a fixed image distance u_r is hard due to unbounded $d \in [F,\infty)$, and is likely to be scene-dependent.

5.3 Texture Parameter Mapping

The previous formulation uses a fixed focus. We here derive mappings for other parameters to reuse the LUT at runtime.

5.3.1 Foci

As d_r (as a focal depth) moves, we may repeat profiling, but want to keep the LUT compact. To this end, we introduce a mapping for varying focal depth d_f using duality between u and d. Each pattern found at an image distance u_f in the LUT is equivalent to the image formed by the object at d_r (here, not necessarily focused), when a focus d_f is at $d(u_f)$.

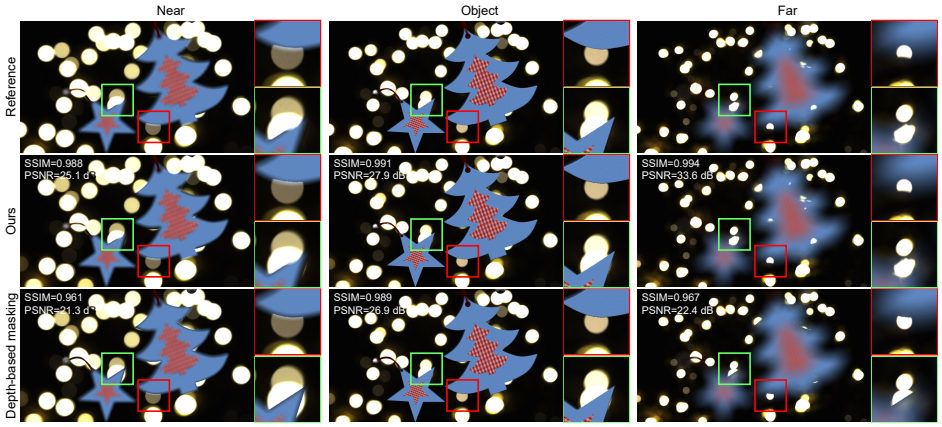


Fig. 8: *Comparison of visibility changes in the reference [4], ours, and depth-based masking [7], [8], as the focus moves front to back.*

Hence, the image of the object at d_r can be found at $u(d_f)$. As a consequence, focus-dependent images of $\mathbf{p}_r$ can be fetched using a texture coordinate $\mathbf{t} := (u(d_f), r)$, where r is a radial length of $\mathbf{v}$.

5.3.2 Object Depth

Since we can find the image of the object at d_r, what we need next is an image for a different object distance d' given the same d_f. We assume the profiles linearly scale for different object distances. Then, we can find the mapping from the configuration $\langle d', d_f \rangle$ to $\langle d_r, d'_f \rangle$ so that both have the same image-space COC radii as:

$$C(d', d_f) = C(d_r, d'_f). \tag{8}$$

Solving Eq. (8) for the unknown d'_f leads to the texture coordinate $\mathbf{t}' = (u(d'_f), r)$ for the configuration $\langle d', d_f \rangle$.

5.3.3 Spectral Dispersion

We can similarly derive the COC mapping for dispersion, but our mapping does not well capture the nonlinearity of chromatic aberrations. Fortunately, chromatic patterns can be integrated together. We integrated 25 wavelengths in $\lambda = [440, 680]$ nm, and this leads to high-quality color profiles similar to those of the reference ray tracer (see Sec. 6.2).

5.4 Advanced Appearance Models

Here, we describe further considerations beyond the intensity profile, revisiting the previous literature.

5.4.1 Aperture

The bokeh shape itself under higher f-stops (i.e., partial diaphragm opening) needs to resemble the diaphragm. To incorporate the shape, the aperture can be defined as a binary mask. We multiply it with the 2D intensity profile, and obtain the 2D sprite. More realistic aperture textures can incorporate granular noises [20] and near-field ringing [34].

5.4.2 Distortion

Additional non-trivial radial aberrations are found at periphery such as barrel and pin-cushion distortions. To reflect the distortions, we apply a simple parametric warping based on the division model [35], defined as: $\mathbf{u} = \mathbf{c} + \frac{\mathbf{d}-\mathbf{c}}{1+K|\mathbf{d}-\mathbf{c}|^2}$, where $\mathbf{u}$, $\mathbf{c}$, and $\mathbf{d}$ are an undistorted point, the center of the screen, and a distorted point, respectively. K is an empirical coefficient to control the shape (e.g., -0.1 for the barrel distortion).

5.4.3 Optical vignetting

Optical vignetting, often dubbed as "cat's eye," is an effect that is caused by the successive reduction of lens openings, which causes elliptic bokeh patterns at the periphery; reflections from housing also exist, but are negligible. We again prefer an analytic model to the ray tracing (naturally achieving this). The unblocked area can be approximated as the intersection of the projection of an earlier opening onto a rear element along the angle of the incident rays [36]. The simplest way is involving only the entrance pupil and the smallest element, leading to a sharp boundary. Successive reductions through multiple elements yield a smooth edge.

6 Results

In this section, we first report the evaluation of our visibility sampling technique. Then, we provide more examples to prove the versatility of our solution ranging from high quality to real-time rendering. Our solution is implemented on an Intel Core i9 2.8GHz machine with NVIDIA GeForce RTX 2080 Ti using OpenGL. The tests use 1920×1080 resolution.

6.1 Visibility Sampling

6.1.1 Method

Four scenes are used for the experiments; *Christmas Tree* (CT; 9 objects, 25K tri.), *BaLl* (BL; 9 objects, 36K tri.), *Statue in Street* (SS; 1 objects, 301K tri.), and *Toy Dinosaurs* (TD; 275 objects, 617K tri.). The CT scene locates objects in a shallow depth range with many background highlights (about 32K for our point-source model). The BL scene distributes balls in a wider depth range. The SS scene has a single high-complexity object. The TD scene has many complex objects.

Reference results (of the highest rendering quality) are generated with the accumulation buffering [4] with sufficient samples ($N = 64K$), which is the most accurate rasterization-based rendering technique in terms of visibilities. For comparison, we took three existing techniques, all of which are based on the thin-lens model. The accumulation buffering (yet with less samples) and image-space ray tracing [5] are chosen for object-space and image-space techniques, respectively. Simple depth-based masking is chosen for an approximate postprocessing technique [7], [8], which simply compares depths of highlight pixels and objects in pinhole images.

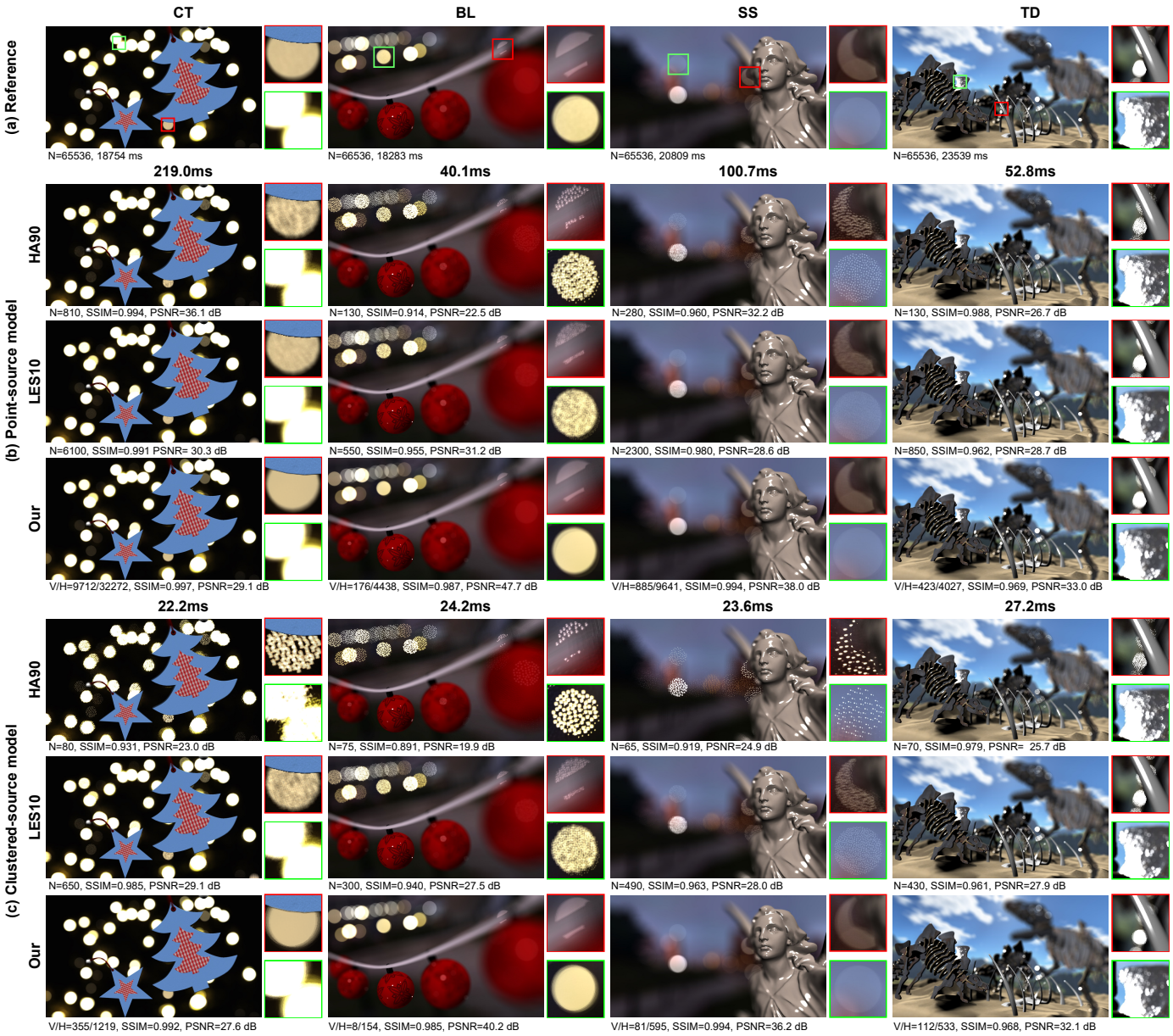


Fig. 9: *Equal-time comparison of our models with projective rasterization, accumulation buffer method [4], and image-based ray tracing [5] against reference. N, V, and H are number of samples, partially visible bokeh, and bokeh sources (highlights), respectively.*

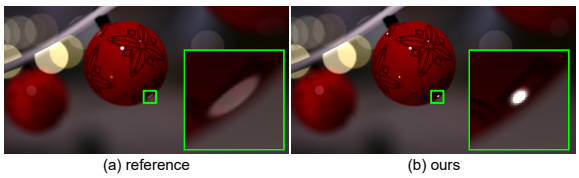


Fig. 10: *Our single-view highlight extraction does not capture view dependency (see the elongation of the highlights in the reference).*

Our bokeh rendering is designed as a postprocessing solution for defocus-blur rendering. Whereas any of methods can be used, we took the image-space ray tracing [5] to fulfill high quality and performance simultaneously; precisely, we use 4-layer depth peeling and 256 lens samples.

The quality is measured against the references in terms of peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) [37], and performance in frame time (in ms). Since ours does not generate regular defocus blur, we measured only the qualities of bokeh renderings (Fig. 7) to observe the sole effects of the bokeh rendering. To extract bokeh-only images from the references, we removed the colors of non-highlight objects, as for our highlight extraction (Sec. 4.1). Performance, however, was measured for the entire process, including the regular defocus-blur rendering.

For the bokeh-source models of our solution, we separately assessed the point-source (PS) and clustered-source (CS) models. We excluded intrinsic appearance variations for the quality comparison, since they are not available for the previous techniques. The CS model uses clustered sources that merge down to 4, 3, 6, and 13 % of the point sources for CT, BL, SS, and TD scenes, respectively. The CS model did not use the global post-merge operation, which too simplifies the resulting look (e.g., further reduction down to 1/3).

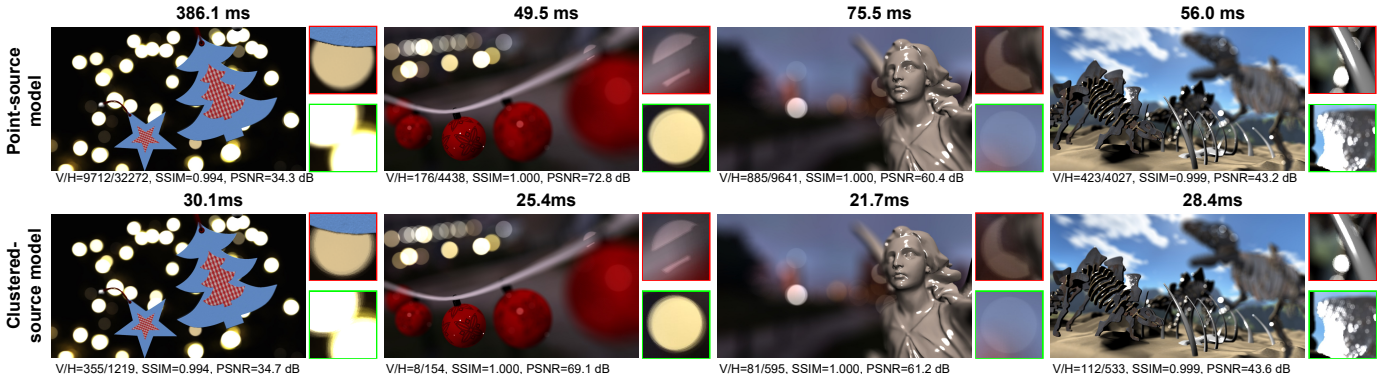


Fig. 11: *Visibility-map warping and its quality comparison. The quality comparison is measured against ours in Figs. 9(b) and 9(c).*

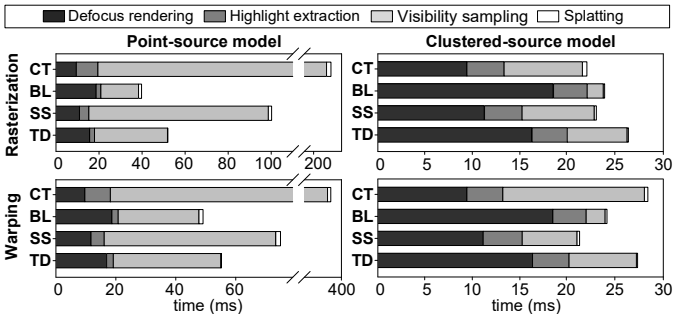


Fig. 12: *Performance breakdown of our bokeh rendering.*

6.1.2 Quality

Fig. 8 shows the overall accuracy of our visibility sampling with respect to focus changes from front to back. Ours evaluates visibility correctly, similar to those of the reference; the partial visibility of the bokeh (the red and green insets) shows overturned results, but not in the depth-based masking.

Fig. 9 shows the equal-time comparison of our method (using projective rasterization) and the two distributed techniques. Overall, our solution is inherently free from noises, and is visually comparable to the reference in both the PS and CS models, while the accumulation buffering and image-based ray tracing produce noisy undersampling artifacts. The CT, BL, and SS scenes are of high similarities (SSIM>0.98) but the TD scene is slightly lower (SSIM>0.96). The lower SSIM values for the TD scene result from one of the limitations of our solution, where highlights are extracted from a single pinhole view, and their view dependency is not captured unlike the references (see Fig. 10 for another example). The PSNR value is low in the CT scene, because our highlight extraction method cannot capture highlights outside the image periphery (left bottom of the CT scene).

Our CS approximation highly improves performance, while marginally degrading quality (< 0.05; SSIM). The approximation is still effective at 90% reduction of the point-source. A more aggressive approximation for further speed-up needs to be tuned for quality-performance trade-off.

We also evaluate the effects of the visibility-map warping against the projective rasterization in PS and CS models (Fig. 11). The warping produces almost the same results (SSIM>0.99 and PSNR>34) as the precise rasterization does.

6.1.3 Performance

The CS model significantly improves the performance against the PS model (Fig. 9). The speed-up factors of the CS model with projective rasterization against the PS model reach up to 9.9, 1.7, 4.3, and 1.9 for CT, BL, SS, and TD scenes, respectively. The speed-up is more pronounced for the scenes with many highlights (here, CT and SS).

Fig. 11 shows rendering performance of the visibility-map warping. Not all the cases benefit from the warping. A speed-up exists, when the warping is cheaper than the geometric rendering of occluders. This corresponds to SS scene, where the speed-up factors are 1.3/1.1 (PS/CS). For the other scenes (with lower geometric complexity), the projective rasterization performs better; the speed-ups of the regular rasterization are 1.8/1.4, 1.2/1.1, and 1.1/1.0 (PS/CS) for CT, BL, and TD scenes, respectively. If scene statistics is available, we can choose a better strategy.

Fig. 12 reports the per-stage breakdown of performance of our PS and CS models, including the entire rendering process from highlight extraction to regular defocus-blur rendering and splatting. The visibility sampling governs the rendering cost for the PS model, but its cost is significantly reduced with the highlight source clustering. Unlike the object-space techniques, the performance of our approach depends on the number of highlights. Therefore, CT and SS scenes with many highlights are relatively slow, unlike BL and TD scenes.

Our culling techniques greatly reduce the number of per-highlight visibility sampling. CT, BL, SS, and TD scenes cull 63, 95, 90, 88 % of highlights, respectively. In particular, the frustum culling is very effective, where the culling ratios reach up to 56, 95, 80, 43 %.

Fig. 13 reports the effects of the degree of the clustering in the CS model (i.e., the mipmap level l of the highlight map reduction). The deeper levels gain higher reduction of the highlights, trading quality for performance. In our experiments, a good balance is found at the level $l = 4$ (reduction down to 4, 3, 6, and 13 % for CT, BL, SS, and TD scenes), where the quality (in terms of SSIM) is still close to the those of the PS model, but the cost reduction is great.

6.2 Intrinsic Appearance

We here demonstrate our LUTs, and compare their intensity reconstructions along varying object depths against ray tracing. Then, we show the combinations of advanced factors.

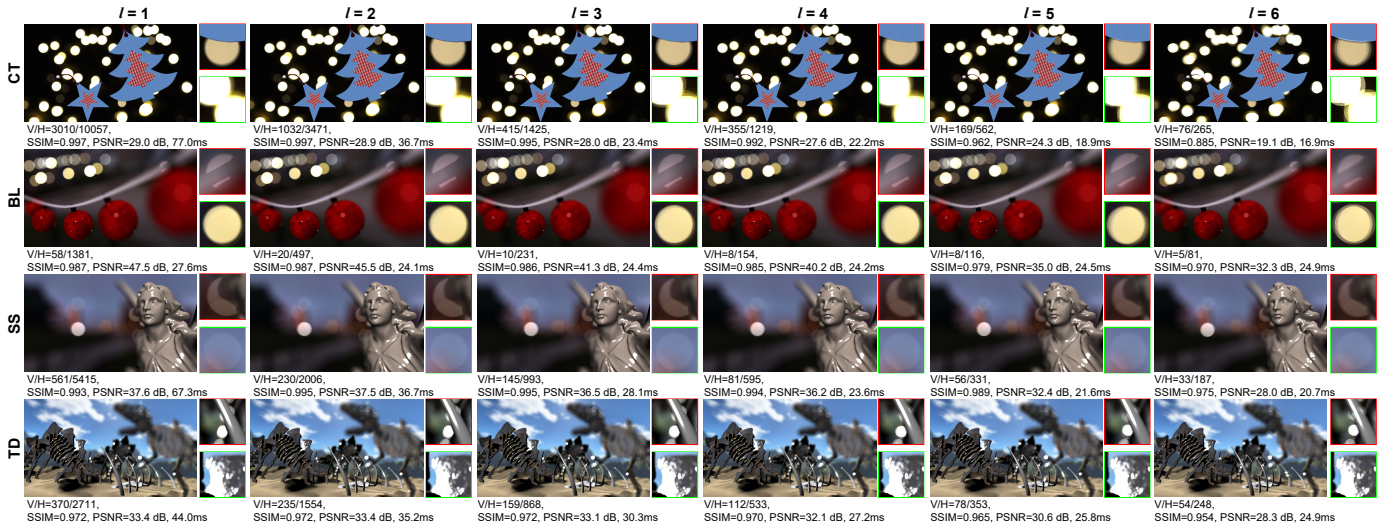
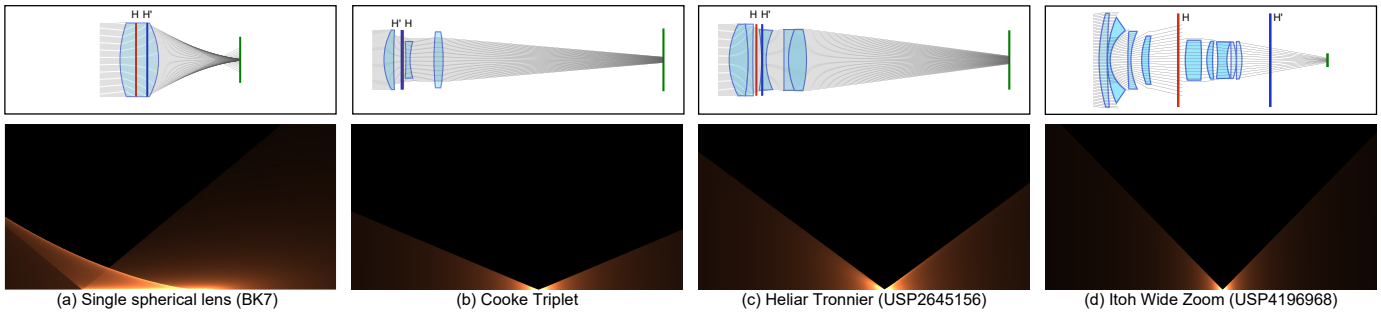


Fig. 13: *Effects of the mipmap level l in our mipmap-based highlight-source clustering.*



(a) Single spherical lens (BK7) (b) Cooke Triplet (c) Heliar Tronnier (USP2645156) (d) Itoh Wide Zoom (USP4196968)

Fig. 14: *Example achromatic bokeh-profile LUTs ray-traced for the reference wavelength ($\lambda_r = 587.6\ nm$, $d_r = 1000\ mm$).*

6.2.1 Intensity Profile

The intensity profile is generated with 1024^2 rays on the entrance pupil in an offline stage. The profile is ray-traced for $d_r = 1{,}000$ mm and $\lambda_r = 587.6$. The chromatic profile uses 25 wavelengths in [440, 680] nm. Four lens systems are used for the experiments; a simple spherical lens, triplet lens, prime lens, and zoom lens (Fig. 14). In our tests, the process takes 10–12 sec for the achromatic profile and a few minutes for the chromatic profile, but needs to be done only once.

Fig. 15 compares bokeh patterns generated with our LUTs along object distances against the ray-traced patterns; reconstructed 2D bokehs are shown below each profile. Since our mapping (Eq. 8) assumes a thick-lens model, slight errors are observed due to aberrations while evaluating principal planes. Though, the quality is high; the maximum PSNRs reach up to 60 dB, and minimum PSNRs around 28 dB.

6.2.2 Individual Effects

Further appearance improvements on optical vignetting, distortion/dispersion, and aperture shape are shown in the Fig. 16; the TD scene is excluded due to its small bokeh sizes. The CT scene used a sharp optical vignetting (using the smallest lens element), while BL and SS scenes use successive multi-element optical vignetting. The barrel distortion is exaggerated with $K = -0.4$. They greatly improve photorealism with only marginal cost (0.5–1 ms for all).

6.3 Optimization for Practical Real-Time Rendering

So far, we have shown the quality and performance of our solution, which aims to produce results similar to those of references. Here, we show more practical examples that are optimized for real-time rendering (Fig. 17). Aggressive (hand-crafted) clustering of highlights produces still good perceived qualities. We use 66 (18 points, 48 clustered sources), 42 (1 points, 41 clustered sources), and 37 (7 points, 30 clustered sources) highlights for the CT, BL, and SS scenes, respectively. For the visibility sampling, the CT and BL scenes use the projective rasterization, and SS scene uses the warping; performance difference between the two is insignificant (up to 0.2 ms). Our solution adds only 1–2 ms to to the regular defocus-blur rendering, and still achieves realistic looks.

7 Conclusion and Limitations

We presented a real-time bokeh rendering technique. Our solution can handle dynamic visibilities by decoupling bokeh sources from the regular defocus-blur rendering. Our acceleration techniques can reduce the cost for visibility map evaluation. Our intrinsic profile is stored in a compact LUT; a single 1024^2 LUT texture is enough in practice. Its lightweight parameter transformation allows us to render complex bokeh effects in real time. Finally, we conclude with limitations.

Our visibility sampling may be inefficient for excessive highlights (e.g., > 10K sources) due to repetitive rasterization.

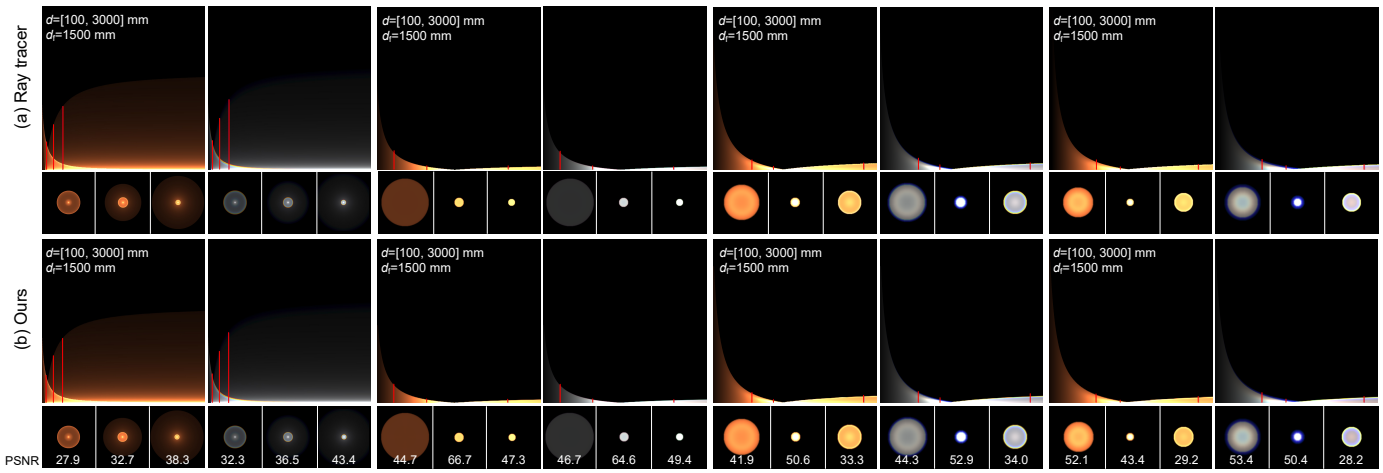


Fig. 15: *Comparison of achromatic/chromatic bokeh profiles reconstructed from the pre-generated LUTs, evaluated with respect to the object distance against the reference ray tracing. The red lines indicate the sampled depths for reconstructed example textures.*

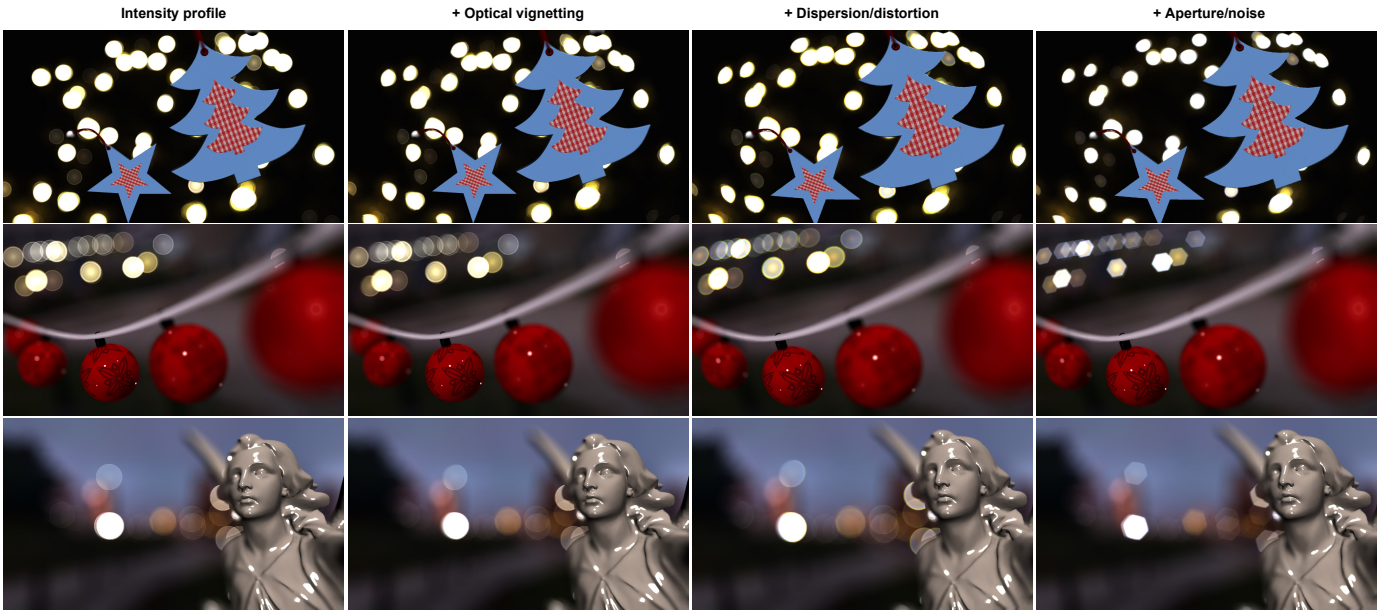


Fig. 16: *Effects of combinations of advanced intrinsic appearance models.*

A better highlight reduction strategy needs to be investigated to handle more highlights.

Our single-view highlight extraction is view-independent, whereas it should be view-dependent. Hence, the results are slightly different from the reference. Tiny specular sources may flicker, and highlights can be elongated by the displacement of views. This view dependency could be improved with sparse multi-view inputs.

Our highlight extraction and clustering-based approximation does not guarantee temporal coherence, which may cause slight instability in consecutive frames. A simple remedy can be a smooth transition of the lifetime of highlights similarly to particle systems, but further research is encouraged on temporally-coherent clustering and interpolation.

Quality of our highlight extraction can be improved with light-source estimation techniques. Recently, learning-based techniques exist [38], [39], but their estimation performance and target environment constraints limit their use in real-time rendering. This encourages further work on real-time high-quality light-source estimation.

Acknowledgments

The Lucy model is provided through the courtesy of the Stanford 3D Scanning Repository. This work was supported in part by the Samsung Research Funding & Incubation Center of Samsung Electronics (SRFC-IT1901-01) and Mid-career R&D program through the NRF grants (2019R1A2C2002449) funded by the Korea Government. Correspondences concerning this article can be addressed to Sungkil Lee.

References

[1] D. Präkel, *The visual dictionary of photography*. Ava Publishing, 2010.
[2] H. M. Merklinger, "A technical view of bokeh," *Photo Techniques*, vol. 18, no. 3, 1997.

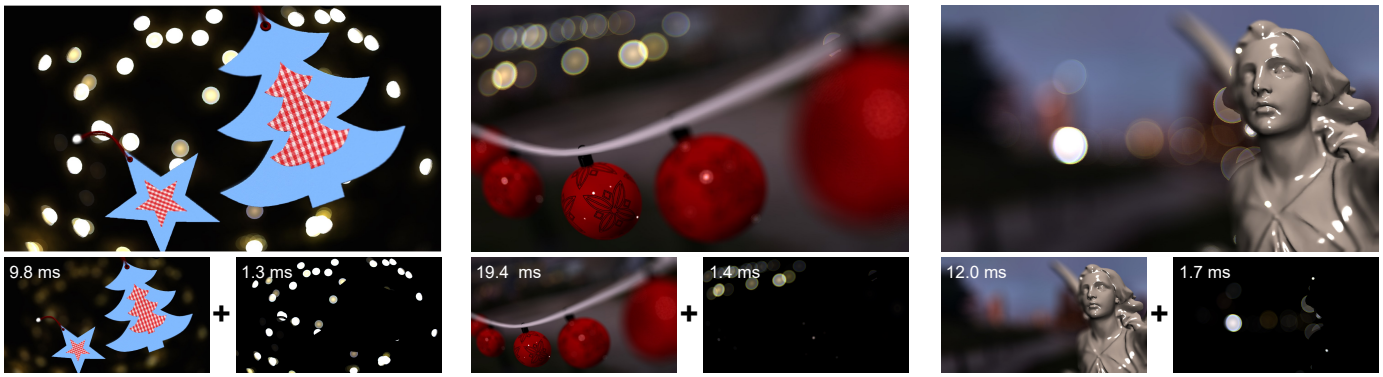


Fig. 17: *Examples of practical performance optimization with hand-crafted selection of point-source and clustered-source highlights. In the bottom row, each pair of left and right images corresponds to the base defocus-blur rendering and our bokeh-only result.*

[3] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," *Computer Graphics*, vol. 18, no. 3, pp. 137–145, 1984.

[4] P. Haeberli and K. Akeley, "The accumulation buffer: hardware support for high-quality rendering," *ACM Computer Graphics*, vol. 24, no. 4, pp. 309–318, 1990.

[5] S. Lee, E. Eisemann, and H.-P. Seidel, "Real-Time Lens Blur Effects and Focus Control," *ACM Trans. Graphics*, vol. 29, no. 4, pp. 65:1–7, 2010.

[6] S. Lee, G. J. Kim, and S. Choi, "Real-Time Depth-of-Field Rendering Using Splatting on Per-Pixel Layers," *Computer Graphics Forum*, vol. 27, no. 7, pp. 1955–1962, 2008.

[7] N. Hoobler, "High performance post-processing," in *Game Developers Conference*, 2011.

[8] M. Mittring and B. Dudash, "The Technology Behind the DirectX11 Unreal Engine "Samaritan" Demo," in *Game Developer Conference*, 2011.

[9] L. McIntosh, B. E. Riecke, and S. DiPaola, "Efficiently simulating the bokeh of polygonal apertures in a post-process depth of field shader," *Computer Graphics Forum*, vol. 31, no. 6, pp. 1810–1822, 2012.

[10] T. McGraw, "Fast bokeh effects using low-rank linear filters," *The Visual Computer*, vol. 31, no. 5, pp. 601–611, 2015.

[11] J. Wu, C. Zheng, X. Hu, Y. Wang, and L. Zhang, "Realistic rendering of bokeh effect based on optical aberrations," *The Visual Computer*, vol. 26, no. 6-8, pp. 555–563, 2010.

[12] J. Wu, C. Zheng, X. Hu, and F. Xu, "Rendering realistic spectral bokeh due to lens stops and aberrations," *The Visual Computer*, vol. 29, no. 1, pp. 41–52, 2013.

[13] J. Hanika and C. Dachsbacher, "Efficient monte carlo rendering with realistic lenses," *Computer Graphics Forum*, vol. 33, no. 2, pp. 323–332, 2014.

[14] H. Joo, S. Kwon, S. Lee, E. Eisemann, and S. Lee, "Efficient Ray Tracing Through Aspheric Lenses and Imperfect Bokeh Synthesis," *Computer Graphics Forum*, vol. 35, no. 4, pp. 99–105, 2016.

[15] M. Potmesil and I. Chakravarty, "A lens and aperture camera model for synthetic image generation," *ACM Computer Graphics*, vol. 15, no. 3, pp. 297–305, 1981.

[16] D. Liu, R. Nicolescu, and R. Klette, "Stereo-based bokeh effects for photography," *Machine Vision and Applications*, vol. 27, no. 8, pp. 1325–1337, 2016.

[17] D. Lanman, R. Raskar, and G. Taubin, "Modeling and synthesis of aperture effects in cameras." in *Computational Aesthetics*, 2008, pp. 81–88.

[18] T. Hach, J. Steurer, A. Amruth, and A. Pappenheim, "Cinematic bokeh rendering for real scenes," in *Proc. European Conf. Visual Media Production*. ACM, 2015.

[19] J. Buhler and D. Wexler, "A phenomenological model for bokeh rendering," in *ACM SIGGRAPH Abstracts and Applications*. ACM, 2002, pp. 142–142.

[20] M. Hullin, E. Eisemann, H.-P. Seidel, and S. Lee, "Physically-Based Real-Time Lens Flare Rendering," *ACM Trans. Graphics*, vol. 30, no. 4, pp. 108:1–9, 2011.

[21] M. Kraus and M. Strengert, "Depth-of-field rendering by pyramidal image processing," *Computer Graphics Forum*, vol. 26, no. 3, pp. 645–654, 2007.

[22] Y. Gotanda, M. Kawase, and M. Kakimoto, "Real-time rendering of physically based optical effects in theory and practice," in *ACM SIGGRAPH 2015 Courses*. ACM, 2015, p. 23.

[23] C. de Rousiers and M. Pettineo, "Depth of field with bokeh rendering," in *OpenGL Insights*, P. Cozzi and C. Riccio, Eds. CRC Press, 2012, ch. 15, pp. 205–218.

[24] Y. Endo, Y. Kanamori, and J. Mitani, "Deep reverse tone mapping," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 177:1–10, 2017.

[25] G. Eilertsen, J. Kronander, G. Denes, R. K. Mantiuk, and J. Unger, "HDR Image Reconstruction from a Single Exposure Using Deep CNNs," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 178:1–15, 2017.

[26] S. Lee, Y. Kim, and E. Eisemann, "Iterative Depth Warping," *ACM Trans. Graphics*, vol. 37, no. 5, pp. 177:1–13, 2018.

[27] X. Décoret, "N-buffers for efficient depth map query," *Computer Graphics Forum*, vol. 24, no. 3, pp. 393–400, 2005.

[28] W. J. Smith, *Modern Optical Engineering*. McGraw-Hill, 2000.

[29] C. Everitt, "Interactive order-independent transparency," NVIDIA Corporation, Tech. Rep., 2001.

[30] C. Kolb, D. Mitchell, P. Hanrahan *et al.*, "A realistic camera model for computer graphics," in *Proc. ACM SIGGRAPH*, vol. 95, 1995, pp. 317–324.

[31] B. A. Barsky, D. R. Horn, S. A. Klein, J. A. Pang, and M. Yu, "Camera models and optical systems used in computer graphics: part i, object-based techniques," in *Proc. Int. Conf. Computational Science and its Applications*. Springer, 2003, pp. 246–255.

[32] M. B. Hullin, J. Hanika, and W. Heidrich, "Polynomial Optics: A Construction Kit for Efficient Ray-Tracing of Lens Systems," *Computer Graphics Forum*, vol. 31, no. 4, pp. 1375–1383, 2012.

[33] E. Schrade, J. Hanika, and C. Dachsbacher, "Sparse high-degree polynomials for wide-angle lenses," *Computer Graphics Forum*, vol. 35, no. 4, pp. 89–97, 2016.

[34] P. Pellat-Finet, "Fresnel diffraction and the fractional-order fourier transform," *Optics Letters*, vol. 19, no. 18, pp. 1388–1390, 1994.

[35] A. W. Fitzgibbon, "Simultaneous linear estimation of multiple view geometry and lens distortion," in *Proc. Conf. Computer Vision and Pattern Recognition*. IEEE, 2001, pp. I125–I132.

[36] N. Asada, A. Amano, and M. Baba, "Photometric calibration of zoom lens systems," in *Proc. Int. Conf. Pattern Recognition*, 1996, pp. 186–190.

[37] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[38] Y. Hold-Geoffroy, K. Sunkavalli, S. Hadap, E. Gambaretto, and J.-F. Lalonde, "Deep outdoor illumination estimation," in *Proc. Conf. Computer Vision and Pattern Recognition*, 2017, pp. 7312–7321.

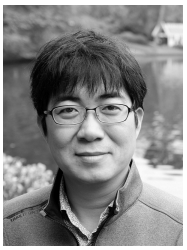
[39] S. Song and T. Funkhouser, "Neural illumination: Lighting prediction for indoor environments," in *Proc. Conf. Computer Vision and Pattern Recognition*, 2019, pp. 6918–6926.

Yuna Jeong received the B.S. degree in computer engineering at Korea Polytechnic University (2012) and Ph.D. degrees in computer engineering at Sungkyunkwan University (2019). She is senior researcher in the Intelligent Infrastructure Technology Research Center at the Korea Institute of Science and Technology Information (KISTI). Her main research interests include real-time rendering, GPU rendering, and feature engineering.

Seung Youp Baek received the B.S. degree in Computer Science at University of Texas at Austin (2015). He is a M.S. student in Computer Science and Engineering at Sungkyunkwan University. His main research interest is real-time rendering.

Yechan Seok received the B.S. degree in Information and Communication Engineering at Inha University (2019). He is a M.S. student in Computer Science and Engineering at Sungkyunkwan University. His main research interest is real-time rendering.

Gi Beom Lee received the B.S. degree in IT Convergence at Soongsil University (2019). He is M.S. student in Computer Engineering at Sungkyunkwan University. His main research interest is real-time rendering.



Sungkil Lee is an associate professor of Computer Science and Engineering at Sungkyunkwan University, Korea. He obtained his BS (2002) and PhD (2009) degrees in Materials Science and Engineering and Computer Science and Engineering at POSTECH, respectively. He was a postdoctoral researcher at the Max-Planck-Institut (MPI) fuer Informatik (2009–2011). He was a visiting professor at the Delft University of Technology (2017). His research interests include computer graphics, virtual reality, and information visualization with emphasis on GPU algorithms and optics.