Depth-of-Field Rendering with Multiview Synthesis

Sungkil Lee
Max-Planck-Institut für Informatik

Elmar Eisemann
Saarland University / MPI Informatik

Hans-Peter Seidel
Max-Planck-Institut für Informatik

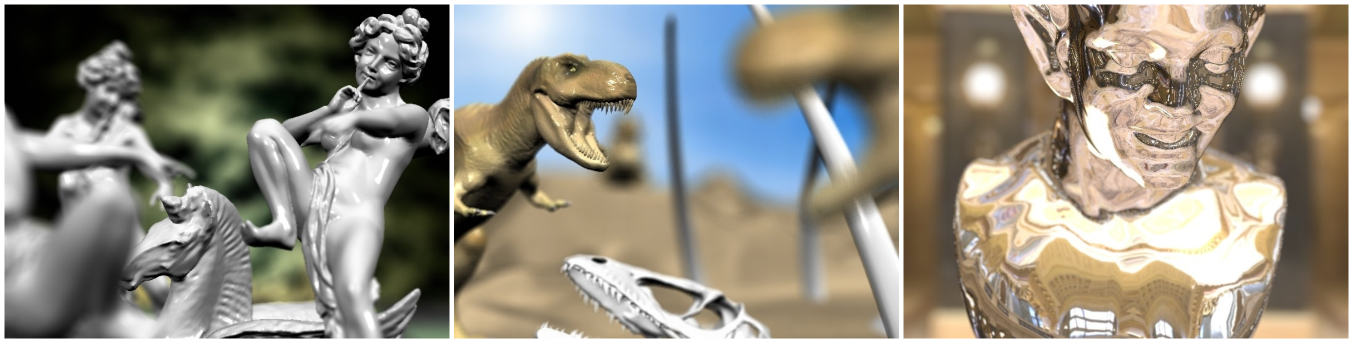

Figure 1: Our method achieves DOF blur effects comparable to accurate solutions in real time and avoids postprocessing artifacts.

Abstract

We present a GPU-based real-time rendering method that simulates high-quality depth-of-field effects, similar in quality to multiview accumulation methods. Most real-time approaches have difficulties to obtain good approximations of visibility and view-dependent shading due to the use of a single view image. Our method also avoids the multiple rendering of a scene, but can approximate different views by relying on a layered image-based scene representation. We present several performance and quality improvements, such as early culling, approximate cone tracing, and jittered sampling. Our method achieves artifact-free results for complex scenes and reasonable depth-of-field blur in real time. **This is the authors' version of the paper. The ultimate version was published at SIGGRAPH Asia 09**

1 Introduction

A finite aperture of a lens maps a 3D point to a *circle of confusion* (COC). The overlapped COCs introduce variations in blur in an image; objects in the limited *depth of field* (DOF) appear to be clear, while the rest are blurred [Potmesil and Chakravarty 1981]. Such a blur dramatically improves photorealism and further mediates pictorial depth perception [Mather 1996].

To generate focal imagery in computer graphics, a number of techniques have been attempted in the past. Despite tremendous GPU advances, the accurate DOF blur still remains challenging due to the complexity of visibility changes. Previously, accurate solutions could only be achieved by multiview sampling methods [Cook et al. 1984; Haeberli and Akeley 1990]. Since this implies rendering a scene repeatedly, their use has been limited to offline animation.

This paper presents a GPU-based real-time rendering method. Our method derives a layered image-based scene and computes an accurate DOF blur. We avoid repeated model rendering by synthesizing new views. In contrast to single-image solutions, our principle stems from multiview sampling. Hence, we support varying visibility, an arbitrary aperture, and even view-dependent shading. Our solution leads to high quality comparable to the reference [Haeberli and Akeley 1990], as shown in Figure 1. The real-time performance is achieved by using approximate cone tracing, jittered sampling (thereby reducing the number of necessary views), and early culling techniques.

The major contributions are: (1) fast view synthesis for arbitrary lens sampling; (2) a DOF-conform metric for our single-pass scene decomposition; (3) acceleration and quality improvements for higher performance; (4) support for view-dependent shading.

2 Previous Work

The majority of the previous DOF rendering methods, except multiview accumulation methods (detailed in Section 3), have relied on the postprocessing of a single-view image.

Gather methods spatially filter images by the COC size at each pixel [Rokita 1996; Riguer et al. 2003; Scheuermann 2004; Bertalmío et al. 2004; Earl Hammon 2007; Zhou et al. 2007; Lee et al. 2009]. This filtering enables high performance, but often leads to *intensity leakage* (focused areas leaking into background). Anisotropic filtering [Bertalmío et al. 2004; Lee et al. 2009] partially addresses this issue, but remains approximate.

Scatter methods [Potmesil and Chakravarty 1981] map source pixels onto COC sprites and blend them from far to near. The required depth sorting is costly, even on modern GPUs, and the technique is mostly used in offline software [Demers 2004].

Single-view methods lose hidden surfaces and lead to incorrect visibility (particularly problematic for a blurred foreground). Multilayer approaches decompose an image onto layers using pixel depths. They blur each layer using a Fourier Transform [Barsky et al. 2002], a pyramidal image processing [Kraus and Strengert 2007], anisotropic diffusion [Kass et al. 2006; Kosloff and Barsky 2007], splatting [Lee et al. 2008], or rectangle spreading [Kosloff et al. 2009], and the layers are alpha-blended. Most of them start from a single image and address partial visibility by color extrapo-

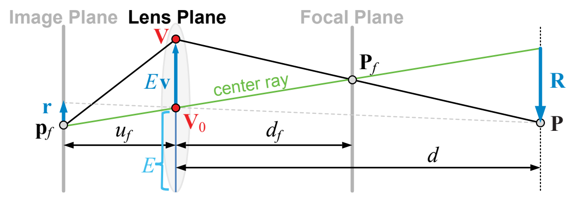

Figure 2: The thin-lens model.

lation into hidden areas. However, the extrapolation is approximate and cannot recover true scene information, which becomes apparent for a strong blur. Instead, Lee et al. [2008] used two-layer depth peeling [Everitt 2001] to address the partial visibility, but it fails if multiple layers need to be combined.

Although the alpha blending approximates visibility well for separated objects, discretization artifacts [Barsky et al. 2005] can appear when objects are spread across multiple layers. A few methods mitigated this problem using special image processing [Barsky et al. 2005], information duplication [Kraus and Strengert 2007], and depth variation [Lee et al. 2008], but the use of layers still remains difficult. Although we also rely on layers, our solution is free from discretization artifacts because we correctly simulate multiview visibility instead of using alpha blending.

3 Thin-Lens Model and Basic Method

In this section, we revisit the DOF discussion based on the thin-lens model [Potmesil and Chakravarty 1981; Cook et al. 1984], which ignores diffraction effects. While the majority of real-time work concentrated on the COC size, we focus on lens rays and the 3D and image points they intersect (see Figure 2).

For a lens with focal length F and a focal plane $\mathcal{F}$ at depth d_f, the image plane $\mathcal{I}$ is situated at:

$$u_f = \frac{F d_f}{d_f - F} \quad \text{for} \quad d_f > F. \tag{1}$$

There is a one-to-one mapping between points $\mathbf{P}_f$ on $\mathcal{F}$ and *pixel* positions $\mathbf{p}_f$ on $\mathcal{I}$. The mapping can be seen as a projection, or, equivalently, the intersection of the *center ray* through $\mathbf{P}_f$ and the center of the lens $\mathbf{V}_0$, with $\mathcal{I}$. For an infinitesimal lens, $\mathbf{V}_0$ is the camera's pinhole. For a given pixel position $\mathbf{p}_f$, all *lens rays* pass through a *lens sample* $\mathbf{V}$ on the lens plane and $\mathbf{P}_f$.

We are interested in finding all 3D points $\mathbf{P}$ on a lens ray for a given pixel $\mathbf{p}_f$. We describe these points with a 3D position on the center ray and a 2D offset vector aligned along with $\mathcal{F}$. Accordingly, we write a lens sample $\mathbf{V} := \mathbf{V}_0 + E\mathbf{v}$, where E is the radius of the lens and $\mathbf{v}$ a 2D offset vector in a unit circle. For a given distance d of $\mathbf{P}$, the 2D offset vector $\mathbf{R}$ becomes:

$$\mathbf{R} = E\left(\frac{d_f - d}{d_f}\right)\mathbf{v}. \tag{2}$$

There is a relationship with the COC when looking at the offset vector $\mathbf{r}$ of $\mathbf{P}$'s projection from $\mathbf{V}_0$ on $\mathcal{I}$:

$$\mathbf{r} = -\frac{u_f}{d}\mathbf{R} = \left(\frac{EF(d - d_f)}{(d_f - F)d}\right)\mathbf{v} =: C(d)\mathbf{v}. \tag{3}$$

Assuming $|\mathbf{v}| = 1$, the classic COC radius is $|C(d)|$.

The multiview accumulation methods [Cook et al. 1984] shoot N rays through $\mathbf{P}_f$ and lens samples $\{\mathbf{V}_1, ..., \mathbf{V}_N\}$ for each pixel $\mathbf{p}_f$. Alternatively, for a given lens sample $\mathbf{V}$, all pixels $\mathbf{p}_f$ can be treated by rendering an image of the scene from $\mathbf{V}$ [Haeberli and Akeley 1990]. The images are accumulated to yield the final result. Since the cost of repeated rendering heavily depends on the scene complexity and number of samples, these methods are generally considered inappropriate for real-time use.

4 Method

Our method starts by decomposing the scene into several layers. We discuss the layer spacing and basic idea of this decomposition (Section 4.1) before focusing on the implementation and layer creation (Section 4.2). The DOF blur is computed from this decomposition. For each image pixel, we shoot lens rays against the layered representation and accumulate their contributions (Section 4.3). Finally, we show how to improve rendering quality and/or performance by approximate cone tracing and jittering (Sections 4.4 and 4.5).

4.1 Single-Pass Scene Decomposition

The layered image-based scene representation is obtained in a single render pass. We use a pinhole camera whose center coincides with the lens center. In this configuration, all 3D points on a center ray are projected onto a single image point. Consequently, the offset vector $\mathbf{r}$ (Equation 3) defines a *relative* displacement in the view and facilitates the tracing of lens rays (Section 4.3).

The distance to the lens is virtually divided into depth intervals, each representing one layer. In principle, a drawn fragment is output on layer L if its fragment depth falls into L's depth interval. Each layer maintains an independent depth buffer. The resulting layer images are equivalent to standard renderings with clipping planes placed at the depth interval bounds of the layer and, hence, the layers capture hidden surfaces (see Figure 3 for example).

The depth intervals could be chosen uniformly or be decreased near the focal plane [Kraus and Strengert 2007], but accuracy can be improved by spacing layers uniformly with respect to the signed COC radius $C(d)$. By definition, any lens ray remains at the same relative position inside the COC. If the COC growth δ is constant from one layer to the next, so will, roughly, be the ray-travel distance in image space. Thereby, the probability of improper visibility handling is reduced. This can also be understood in terms of parallax. Near the observer/lens, small movements lead to large changes, whereas the farther scene appears to be mostly flat.

Computing the depth intervals starts from the near plane at d_{near}. The intervals are defined using δ, the aforementioned ray-travel distance in image space. The layer index for depth d is then given by:

$$l = \left\lfloor \frac{K d_f}{\delta}\left(\frac{1}{d_{\text{near}}} - \frac{1}{d}\right)\right\rfloor \quad \text{for} \quad d_{\text{near}} < d < d_{\text{far}}, \tag{4}$$

where $K := EF/(d_f - F)$. Figure 4 illustrates this spacing. Theoretically, we can define a budget of layers. The images shown in Figure 1 relied on only six layers leading to a δ of 10 pixels for a 800×600 resolution. To ensure quality for arbitrary scenes, we choose the number of layers to fulfill a spacing requirement. This was done for all other images and videos. $C(d)$ has an upper bound K and grows monotonically, but it is unbounded for d approaching zero (Figure 4). The latter is not a problem since d is bound by the near plane distance. Usually, even for a small δ (3 pixels), 16 layers are enough (see supplementary material for an analysis).

4.2 Layer Creation

On current hardware, *multiple render targets* do not allow independent z-buffers. Texture tiling is a work-around. However, it reduces performance, is difficult to handle at boundaries, and the resolution

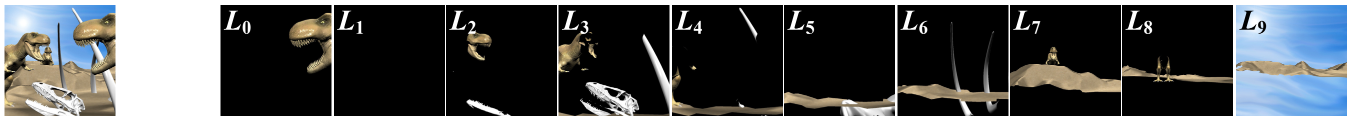

Figure 3: An image-based scene representation that decomposes the leftmost scene onto ten layers ($\{L_0, L_1, .., L_9\}$).

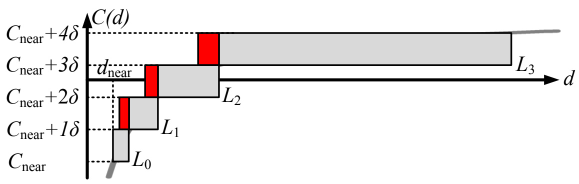

Figure 4: The layers uniformly spaced in terms of $C(d)$. Red portions represent the slight overlap at the layer boundaries.

is limited by the maximal texture size. Our solution uses so-called *layered rendering*. Triangles are sent to a slice in an array of textures, described in the DirectX^{TM}*array texture*. It supports independent depth buffers per entry. In the first pass, a GPU geometry shader redirects each triangle to the layer according to the maximum depth of its vertices. The second pass, applied to all pixels of the layer images, treats large triangles crossing multiple layers. We read a pixel depth, find its corresponding layer L, and copy the pixel to L if its depth is smaller than the depth stored in L. This is a manual depth test and ensures that we keep the nearest fragment in each layer. The intuition for using the maximum vertex depth of each triangle is based on the observation that, if a triangle overlaps a layer boundary, it is already partly in the farther layer and its probability to be hidden in the nearer layer is potentially higher. Our choice thus favors closer hidden geometry. Our solution is approximate because we do not split triangles at the layer boundaries. Consequently, boundary triangles can delete information by writing pixels in a layer they do not belong to. In practice, the differences to an accurate method are small, but the solution is more efficient.

The fragment copy step is also used to simultaneously make the layers slightly overlap by 10%. The overlap removes seam artifacts arising from pixel quantization. Further, the next section explains how our ray-tracing step also benefits from this information.

4.3 DOF Blur Using Layered Representation

Given our scene representation, we compute the DOF blur. Conceptually, each image pixel evaluates one ray per lens sample. The resulting colors are then accumulated. We rely on Equation 2 to trace these rays and Equation 3 to sample the layer textures at the resulting 2D pixel position. Each layer corresponds to a depth interval to which we can associate a segment of the lens ray. The layers are sequentially tested against their corresponding lens-ray segment and the traversal is stopped once a surface is hit. All lens rays are treated iteratively in a fragment shader. The colors of the impact points are averaged to yield the final color. There is no condition on the origin of rays nor on the way their contributions are averaged. This enables the generation of various *bokeh* aspects including large highlights, essential for realistic images [Buhler and Wexler 2002; Lee et al. 2008]. Further, using a Poisson disk sampling leads to an ideal distribution [Cook 1986].

Ray Traversal In each layer, our representation is a height field against which we test the lens rays. Precise rendering techniques exist [Baboud and Décoret 2006], but we apply a more efficient iterative approximation inspired by the *Method of False Position*. It is feasible because our layer spacing ensures that rays are almost perpendicular and the height values are bound by the layer extents.

For a given layer, we need to test the corresponding ray segment s. The two endpoints of s correspond to two texture positions with stored height values d_u and d_l. In each step, our iterative method assumes that the height field varies linearly along the ray. This means that the surface is initially defined solely by d_u and d_l. We intersect s with this linear approximation to get an intersection point I. Let $\hat{d}$ be the height of I. At the corresponding texture position, a height value d is fetched. If $d < \hat{d}$, there is a surface intersection before reaching I. With a binary search, we can rapidly find the accurate intersection. If $d > \hat{d}$, we split s at I and restart with the farther sub-segment. Figure 5(left) shows an example.

Care has to be taken when layer pixels are empty. Depending on the ray's orientation, we initialize their values either to the upper or lower bound of the layer's depth interval to produce an intersection within the layer boundaries (Figure 5(right)). In theory, the approximation does not perform well at boundaries, but the slight layer overlap and our spacing strategy remedy this to a large extent. Three iterations are usually enough for convergence. Figure 5 illustrates two examples. It is even more efficient to initialize d_u and d_l with the layer's depth bounds and only perform the two initial lookups if the pixel underneath the first intersection point is empty.

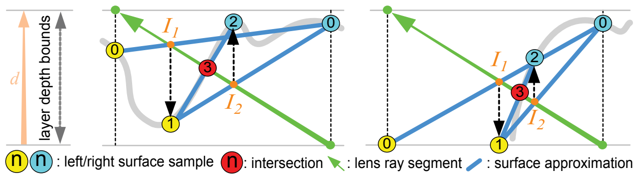

Figure 5: An example of the intersection test. The thick line is the layer's height field. We iteratively sample the surface at interval endpoints to refine a piecewise-linear approximation to find a possible intersection point.

Handling Off-Screen Pixels During the traversal, the ray might actually leave the layer image boundary. On the one hand, this problem can be solved by producing a decomposition of the scene from behind the lens with a viewport that encompasses all lens rays. On the other hand, this implies an increase of texture resolution to enable a sharp image on the focal plane. Therefore, we keep the decomposition from the lens center and discard all rays that leave the frustum. In practice, the degradation is hardly noticeable and only affects pixels in the periphery of the image, whereas the central screen part shows the highest image quality.

Early Layer Culling The layering method incurs large blank image areas. Skipping intersection tests in such regions leads to significant acceleration without a loss of quality. For a given distance, all lens rays will pass through the COC. The pixel region of interest can thus be bound by the maximum COC, which is always realized at one of the depth bounds of the layer. If a minimum depth value in this region is greater than the depth bounds of the layer, no intersection can occur and the layer is safely skipped.

To find the minimum depth, we rely on an *N-Buffer* [Décoret 2005], which is similar to a mipmap, except that each level has the same resolution as the source image. A pixel in the n^{th} N-Buffer level stores the minimum value of all pixels in a square window of size

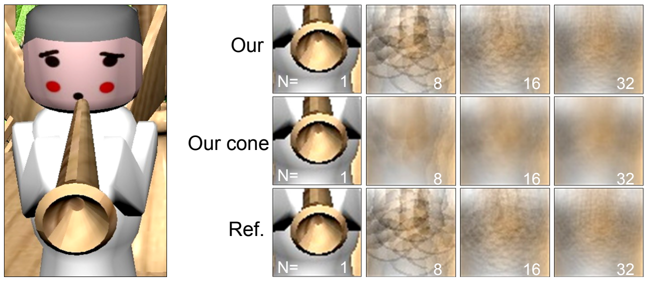

Figure 6: Results without (upper row) and with (middle row) approximate cone tracing and reference images (lower row).

2^n around the pixel. With this, it is possible to find the minimum depth in an arbitrary square area with only four texture accesses corresponding to four overlapping squares. The accuracy is much higher than conventional mipmap tests.

4.4 Approximate Cone Tracing

Similarly to the multiview accumulation methods, our solution needs sufficient lens samples for a strong blur because a single ray only results in one precise impact point. In other words, rays miss between the lens samples. We hence thicken the rays to cones whose radius is defined by half the distance to its neighboring rays. This gives non-intersecting volumes, except for the focal point where all rays meet. These cones correspond to an infinite ray set centered around the original ray and lead to higher quality.

Tracing cones can be expensive. Instead of an accurate evaluation, we approximate the computation. We define the diameter of the cone around a lens ray at distance d_i as $\lambda|C(d_i)|$, where λ is the mean distance among the lens samples. Given an impact point at distance d_i, we consider all pixels lying in a window $\mathbf{W}_c$ according to the cone's diameter. Then, we compute a coverage value (an approximation of the ratio of blocked rays to the total number rays) and an average color in $\mathbf{W}_c$. For efficiency, we use a mipmap of the layer image where we augmented the initial color by an alpha channel in which we indicate the absence or presence of data with the values zero and one, respectively. This can be achieved by clearing the layer image's color and alpha to zero and render the scene's fragments with an alpha of one during the decomposition. Consequently, the mipmapped alpha indicates the pixel coverage v_1 which is the ratio of pixels in $\mathbf{W}_c$ that received a color to those that remained empty during the decomposition. The average color c_1 in $\mathbf{W}_c$ can be obtained by dividing the mipmapped RGB by v_1. If v_1 equals one, the entire cone has been blocked and the color c_1 is reported, else we assume that only a fraction (v_1) of rays resulted in c_1, whereas the other rays continued the traversal. We estimate the ray accumulation probabilistically, using $v_1c_1 + (1-v_1)c_2$, where c_2 is the future contribution obtained by continuing the traversal along the center ray of the cone.

Artifacts usually associated to mipmaps are not introduced, because λ is much smaller than one. For instance, λ is 0.137 for 32 lens samples in our implementation which only results in a 4.4 pixel diameter (mipmap level of 2.1) for a 32-pixel COC radius. Figure 6 shows the examples of synthesized images generated with and without the cone tracing. We see that it quickly removes the fine grain and approaches the reference with many samples.

4.5 Jittered Lens Sampling

The accumulation buffer method [Haeberli and Akeley 1990] relies on multiview rendering which implies the use of the same sampling pattern for each pixel. Insufficient sampling leads to visible aliasing artifacts, especially along object boundaries. Our method allows us to change sample distributions on a per-pixel basis, trading these artifacts for less objectionable noise. The quality of 32 jittered samples approaches and even exceeds the quality of 64 samples to some extent (Figure 7).

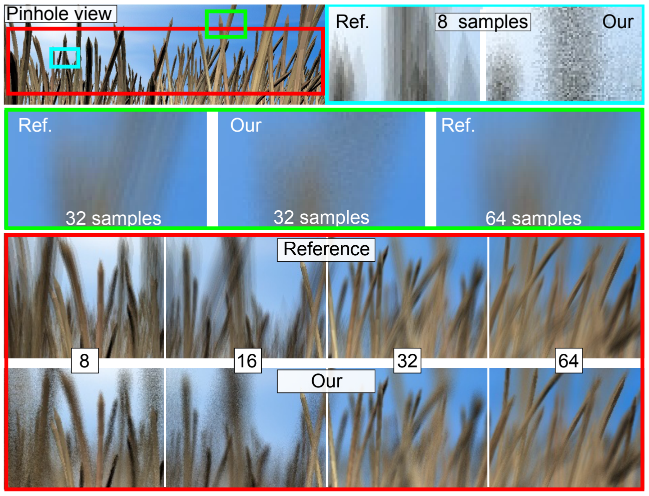

Figure 7: Jittering avoids aliasing (yellow frame), and approaches the quality of higher sampling rates (green frame). It is a good strategy, even in complex scenes (red frame).

To improve upon the aliasing artifacts, one could compute k sample sets and choose randomly amongst them for each pixel. Even though near-optimal sampling patterns could be produced, the use of shader variables to encode such sampling patterns would be expensive. The best sampling behavior resulted from Halton sequences [Keller 1996]. If N samples are used per pixel, a precomputed sequence of $2N$ samples allows us to obtain $2N$ differing, but overlapping sample sets by defining an offset in $[0, 2N-1]$ per pixel. The quality is very high, but the performance penalty is huge, because shader compilers seem unable to optimize for constant shader values when the access pattern differs per pixel.

The trade-off between performance and quality was the jittering of the lens rays by a value encoded in a noise texture. The four components of a texel allow 16 pairwise combinations. Each combination is applied to 1/16th of the lens samples. This solution performed three times faster than the previously mentioned jittering and has a marginal impact on the overall performance.

5 Results

Image Quality Our representative images shown in Figure 1 demonstrate the artifact-free results of our method. Intensity leakage, discretization, and bilinear magnification are avoided. To compare our results, we implemented three state-of-the-art approaches: Haeberli and Akeley's [1990] accumulation buffer (*reference*), the *multilayer* approach by Kraus and Strengert [2007], and the *gather* solution by Lee et al. [2009]. To measure image quality, we evaluate the peak signal-to-noise ratio (PSNR). To better compare perceptual quality, we rely on the structural similarity (SSIM) [Wang et al. 2004] (a rating of 0=worst to 1=best).

We first compare a scene that exhibits strong foreground blur (the first row of Figure 8) to illustrate approximation differences. Our method qualitatively captures the varying visibility of the reference. Note the translucent foreground that partially reveals the left angel and details of the horse. For most rays correct intersections are found, outweighing the fewer erroneous intersections. Hence, the

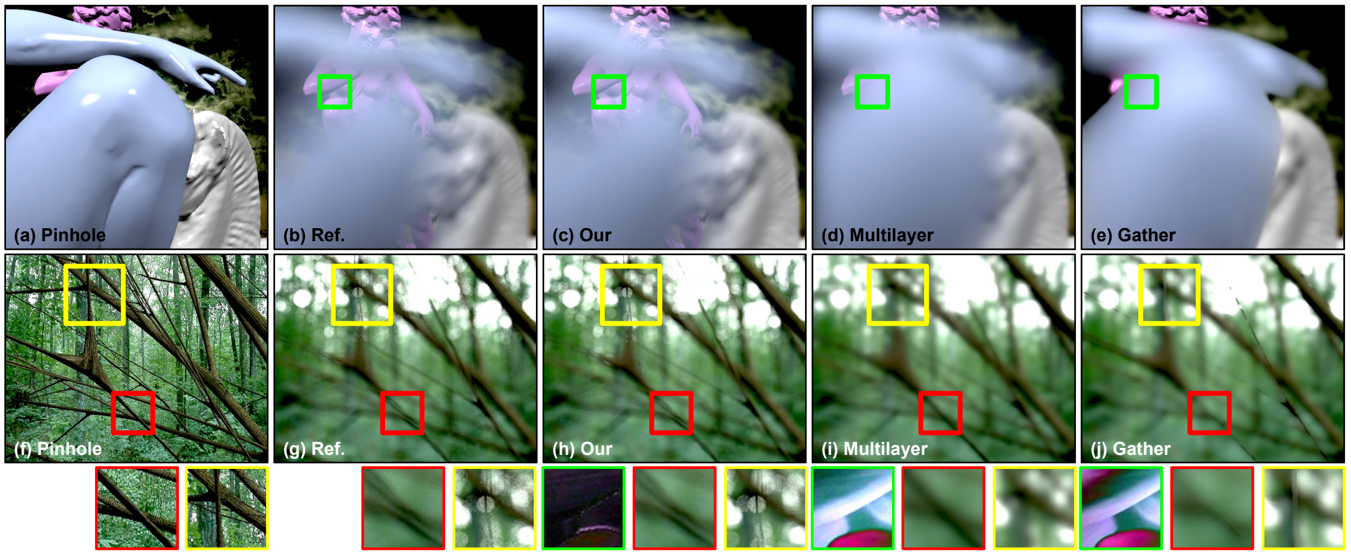

Figure 8: From upper to lower row: foreground blur, distorted geometries, bokeh, and close-ups (green = four times difference to reference).

integration of the rays leads to a good overall estimate. Wrong intersections are most problematic if several rays result in the same wrong color response. This happens mostly along silhouettes, near the focal plane. In theory, here the highest error occurs, but in practice, this results in a difficult-to-perceive small and smooth transition (green zoom: error scaled four times). The other methods suffer from the lack of hidden surface information. The *multilayer* method well approximates foreground boundaries and results in smooth variations, whereas the *gather* method suffers from artifacts. Nevertheless, the extrapolation cannot reconstruct missing details. Our method shows the highest score (SSIM 0.97, PSNR 30.84 dB) compared to the other two (0.91/22.28 and 0.88/19.53 for *multilayer* and *gather*, respectively).

We also investigated a challenging optical phenomenon. When the focal plane is situated between thin objects, the geometry appears distorted (see the red boxes in Figure 8); the two branches are crossing but appear separated. This effect can be reproduced with a real camera. The effect necessitates a close-to-physically accurate simulation that cannot be obtained with the other two methods.

The feasibility of our method for bokeh is illustrated in Figure 8 (yellow boxes). Our sample distributions and variable blending weights allow us to express complex light distributions, whereas the other two methods rely on Gaussian blur to remain compatible with downsampling.

Finally, in contrast to most real-time approaches, we can perform view-dependent evaluations. This is achieved by extracting not only color, but also surface properties during the decomposition step. Figure 9 shows an example with environment-map-based reflections. We extracted diffuse, ambient lighting, normals and shininess exponents, to apply a Phong model when the ray hits the surface. For our cone tracing, we mipmap these values. Our view-dependent evaluation resulted in a score of (0.97/41.38), whereas our view-independent evaluation reached (0.93/27.68). As a reference, the standard pinhole image only reaches a value of (0.59/18.12).

Rendering Performance We measured the rendering performance of all four methods in terms of frame rate (Table 1). The benchmark used the DirectX 10 API and a Pentium 2.83 GHz Core2Quad with an NVIDIA GTX285. Since each method depends on different parameters, we tested several configurations with respect to the number of views and layers.

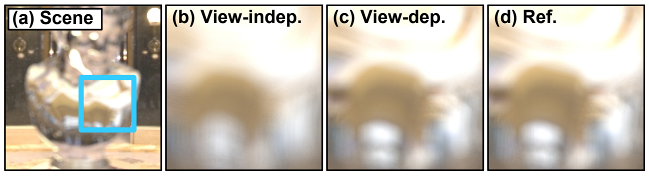

Figure 9: Reflective materials blurred with (b) view-independent and (c) view-dependent shading. (d) is the reference.

The gather method [Lee et al. 2009] was the fastest due to its simplicity. For 32 lens samples, our method showed frame rates faster than the multilayer method, but it allows more than 30 fps even for many samples and layers. Even for the extreme cases (16 layers and 256 samples), our method ran in more than 8 fps. Nonetheless, 32 and 64 lens samples proved sufficient in all our test scenes, each for dynamic and static scenes, respectively. The relatively high performance results from the early culling and ray termination (together up to 50% speed-up). In contrast, the reference method slowed down significantly with increasing scene complexity and sampling.

6 Discussion and Conclusion

Our method is a faster alternative solution to the multiview accumulation method [Haeberli and Akeley 1990]. It is highly scalable with the number of views and layers. The major limitation is the discrete layer decomposition. We addressed this by a continuous surface capture inside each layer and a quality-favoring spacing metric, which also makes our method robust to animation (see video), avoiding temporal issues. Further, even if the entire scene were flattened on a single layer, artifacts are limited (Figure 10).

Another difference with [Haeberli and Akeley 1990] is the lack of anti-aliasing because our method uses an image-based representation. Usually, the out-of-focus elements do not need anti-aliasing. A multisample shot could be produced and then blended separately for the focused elements. Alternatively, we could use multisample buffers which are available on newer graphics hardware.

Although the current implementation already produces high-quality results, it could be modified by trading performance for even more accuracy. For instance, depth peeling can capture all hidden surfaces, but implies several render passes and complex tracing. Brute-

Table 1: Performance comparison of the four methods in terms of frame rate. The benchmark was done at a resolution of 800 × 600.

	Gather	Multilayer			Reference				Our method												
N (number of views)	1	1			8	16	32	64	8			16			32			64			256
M (number of layers)	1	8	12	16	1				8	12	16	8	12	16	8	12	16	8	12	16	16
Toys (67 K triangles)	223	54	38	29	144	75	40	21	102	82	70	77	67	56	62	48	40	38	30	26	9
Dinosaurs (300 K triangles)	175	50	37	28	59	29	14	7	92	71	60	72	57	49	53	42	36	34	27	24	8
Angels (1.76 M triangles)	59	33	26	21	9	4	2	1	43	38	34	38	34	31	32	28	25	24	21	19	8

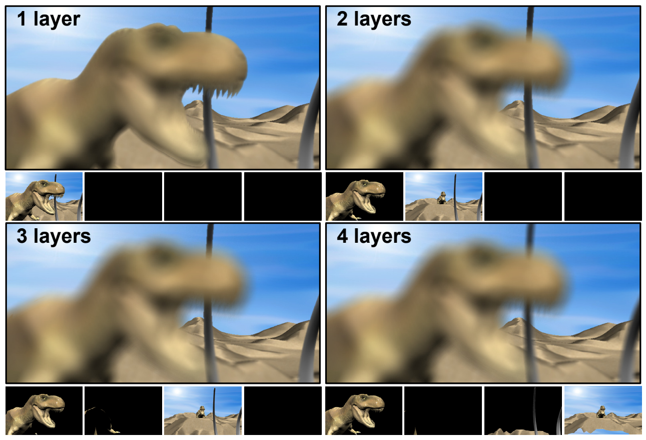

Figure 10: The effect of the number of layers on the final result.

force scanning finds accurate intersection points, but necessitates more texture lookups. Enlarging the viewport can cover off-screen areas, but requires a higher resolution rendering.

In summary, our method is a good alternative to traditional solutions and produces high-quality DOF blur in real time. It is of interest for offline productions, as well as, interactive scenarios.

Acknowledgements: We thank the anonymous reviewers, H. Bezerra, L. Baboud, and R. Herzog for their comments. The 3D models and HDR map are courtesy of the Large Geometric Models Archive at Georgia Tech, AIM@Shape, INRIA GAMMA, Taschen Books (500 3D Objects), Sven Dännart, and SolidWorks Labs. This work was supported by the Cluster of Excellence MMCI (www.m2ci.org).

References

BABOUD, L., AND DÉCORET, X. 2006. Rendering geometry with relief textures. In *Proc. Graphics Interface*, 195–201.

BARSKY, B., BARGTEIL, A., GARCIA, D., AND KLEIN, S. 2002. Introducing vision-realistic rendering. In *Proc. Eurographics Rendering Workshop*, 26–28.

BARSKY, B., TOBIAS, M., CHU, D., AND HORN, D. 2005. Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques. *Graph. Models 67*, 584–599.

BERTALMÍO, M., FORT, P., AND SÁNCHEZ-CRESPO, D. 2004. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proc. 3DPVT*, 767–773.

BUHLER, J., AND WEXLER, D. 2002. A phenomenological model for bokeh rendering. In *Sketch program ACM SIGGRAPH*.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. *Computer Graphics 18*, 3, 137–145.

COOK, R. L. 1986. Stochastic sampling in computer graphics. *ACM Trans. Graphics 5*, 1, 51–72.

DÉCORET, X. 2005. N-buffers for efficient depth map query. In *Proc. Eurographics*, 393–400.

DEMERS, J. 2004. Depth of field: A survey of techniques. In *GPU Gems*. Addison-Wesley, ch. 23, 375–390.

EARL HAMMON, J. 2007. Practical post-process depth of field. In *GPU Gems 3*. Addison-Wesley, ch. 28, 583–606.

EVERITT, C. 2001. Interactive order-independent transparency. *White paper, NVIDIA 2*, 6, 7.

HAEBERLI, P., AND AKELEY, K. 1990. The accumulation buffer: Hardware support for high-quality rendering. *Proc. ACM SIGGRAPH*, 309–318.

KASS, M., LEFOHN, A., AND OWENS, J. 2006. Interactive depth of field using simulated diffusion on a GPU. Tech. rep., Pixar.

KELLER, A. 1996. The fast calculation of form factors using low discrepancy sequences. In *Proc. Spring Conf. on CG*, 195–204.

KOSLOFF, T., AND BARSKY, B. 2007. An algorithm for rendering generalized depth of field effects based on simulated heat diffusion. In *Proc. ICCSA*, 1124–1140.

KOSLOFF, J., TAO, W., AND BARSKY, A. 2009. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. In *Proc. Graphics Interface*, 39–46.

KRAUS, M., AND STRENGERT, M. 2007. Depth-of-field rendering by pyramidal image processing. In *Proc. Eurographics*, 645–654.

LEE, S., KIM, G. J., AND CHOI, S. 2008. Real-time depth-of-field rendering using splatting on per-pixel layers. *Computer Graphics Forum 27*, 7, 1955–1962.

LEE, S., KIM, G. J., AND CHOI, S. 2009. Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation. *IEEE Trans. Vis. and CG 15*, 3, 453–464.

MATHER, G. 1996. Image blur as a pictorial depth cue. *Biological Sciences 263*, 1367, 169–172.

POTMESIL, M., AND CHAKRAVARTY, I. 1981. A lens and aperture camera model for synthetic image generation. *Proc. ACM SIGGRAPH 15*, 3, 297–305.

RIGUER, G., TATARCHUK, N., AND ISIDORO, J. 2003. Real-time depth of field simulation. In *ShaderX*^{2}. 529–556.

ROKITA, P. 1996. Generating depth-of-field effects in virtual reality applications. *IEEE CG and its Application 16*, 2, 18–21.

SCHEUERMANN, T. 2004. Advanced depth of field. In *GDC*.

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

ZHOU, T., CHEN, J., AND PULLEN, M. 2007. Accurate depth of field simulation in real time. *Computer Graphics Forum 26*, 1.