EUROGRAPHICS 2007 / D. Cohen-Or and P. Slavík
(Guest Editors)

Volume 26 (2007), Number 3

Depth-of-Field Rendering by Pyramidal Image Processing

M. Kraus[1] and M. Strengert[2]

[1]Computer Graphics and Visualization Group, Informatik 15, Technische Universität München, Germany
[2]Visualization and Interactive Systems Group, Institut VIS, Universität Stuttgart, Germany

Abstract

We present an image-based algorithm for interactive rendering depth-of-field effects in images with depth maps. While previously published methods for interactive depth-of-field rendering suffer from various rendering artifacts such as color bleeding and sharpened or darkened silhouettes, our algorithm achieves a significantly improved image quality by employing recently proposed GPU-based pyramid methods for image blurring and pixel disocclusion. Due to the same reason, our algorithm offers an interactive rendering performance on modern GPUs and is suitable for real-time rendering for small circles of confusion. We validate the image quality provided by our algorithm by side-by-side comparisons with results obtained by distributed ray tracing.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

In contrast to the infinitesimal aperture of an ideal pinhole camera, real optical systems feature a finite-size aperture. Thus, only points close to the focal plane, i.e., within the depth of field, appear to be in focus while all points that are not close to the focal plane are blurred, i.e., projected to a circle of confusion on the image plane. Depth of field does not only provide important depth cues but can also be used to draw the attention of viewers to specific regions of the image. Therefore, depth of field is an indispensable feature of photorealistic rendering and photography retouching. However, rendering with depth of field at interactive frame rates—or even in real time—without severely compromising image quality is still a challenging task as discussed in detail by Demers [Dem04].

Several of the previously published approaches to this problem are discussed in Section 2 while our algorithm is presented in Section 3. In a nutshell, our method employs image post-processing techniques to achieve interactive rendering performance for any (opaque) color image with a depth map, i.e., an RGBZ image. This image is decomposed into several sub-images according to the pixels' depth. Each of these sub-images is blurred uniformly since all pixels of one sub-image feature similar depth values, which correspond to circles of confusion of approximately the same size. The blurred sub-images are then blended in back-to-front order to compute the resulting image with depth of field.

One of the main features of our algorithm, is a new disocclusion technique for pixels that are not visible in the original pinhole image but contribute to the resulting image with depth of field because of partial occlusions, i.e., they are visible from some points on the simulated lens of finite size but not from its center. The second contribution is a compositing technique that avoids most rendering artifacts at locations where sub-images are separated. Another important contribution of our work is the implementation of all the aforementioned image processing techniques by GPU-based pyramid methods that were recently published by Strengert et al. [SKE06, KS07]. Thus, our approach is particularly well suited as a post-process for interactive GPU-based renderers.

We validate the image quality of our results, which are presented and discussed in Section 4, by a comparison with images computed by the ray tracer `pbrt` published by Pharr and Humphreys [PH04], which is based on distributed ray tracing [CPC84].

2. Background and Related Work

Instead of discussing prior work in chronological order, we will first discuss the approaches that offer the best approximation to physical depth of field and proceed with meth-

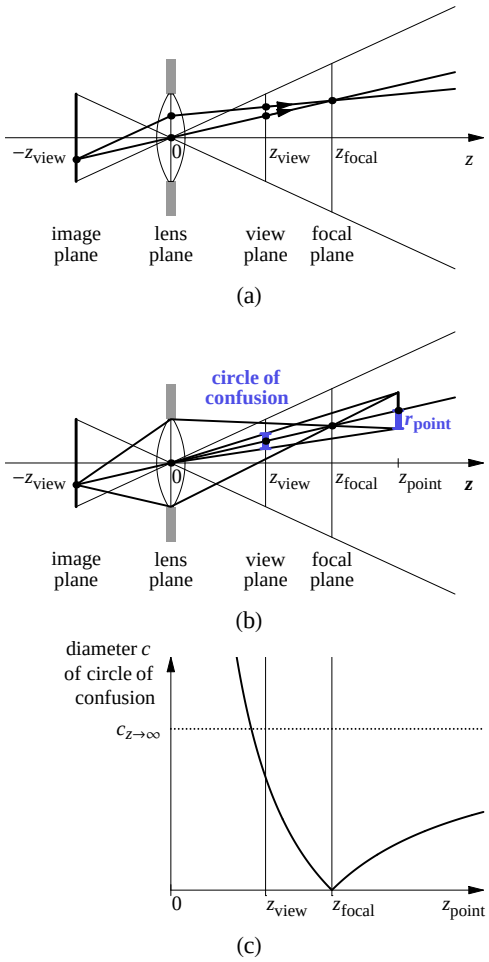

Figure 1: *A thin lens camera:* (a) *Construction of rays for distributed ray tracing.* (b) *Construction of the circle of confusion of a point.* (c) *Plot of the diameter of the circle of confusion of a point as a function of its z coordinate.*

ods that introduce additional approximations. Thus, methods suitable for high-quality off-line rendering are discussed first, followed by lower-quality interactive and real-time rendering techniques. The discussion in this section is limited to background information and approaches that are actually related to our proposed method; more comprehensive surveys of rendering techniques for depth of field are provided, for example, by Demers [Dem04] and Barsky et al. [BHK*03].

2.1. Distributed Ray Tracing

Distributed ray tracing using stochastic sampling was first suggested by Cook et al. [CPC84] (and later patented [CPC93]) for off-line rendering of effects such as gloss, soft shadows, motion blur, and depth of field. For the latter, it extends the pinhole camera model by allowing for rays that enter the three-dimensional scene through any point of a thin camera lens. In other words, a thin camera lens of finite size is stochastically sampled.

Employing the approximations for thin lenses, the lens model is reduced to the focal length f of the lens and its radius $f/(2N)$, where N denotes the f-number, also called focal ratio, relative aperture, or aperture number. (Note that Cook et al. use the symbol F for the focal length and n for the f-number.) Our camera model also includes the specification of the distance z_{view} between the lens and the view plane, which is equal to the distance between the lens and the image plane as the latter is positioned symmetrically to the view plane with respect to the center of the lens, which is also the origin of the z axis in Figure 1a. The distance of the focal plane z_{focal} (or "focal distance," not to be mistaken for the focal length f) is determined by the thin lens equation:

$$\frac{1}{z_{\text{view}}} + \frac{1}{z_{\text{focal}}} = \frac{1}{f} \quad \Rightarrow \quad z_{\text{focal}} = \frac{z_{\text{view}} f}{z_{\text{view}} - f}. \tag{1}$$

Alternatively,—and for the purpose of scene modeling often more conveniently—z_{view} can also be determined for specific values of f and z_{focal}:

$$z_{\text{view}} = \frac{z_{\text{focal}} f}{z_{\text{focal}} - f}. \tag{2}$$

To render with depth of field using distributed ray tracing, a point on the lens plane is chosen for each sample point on the image plane. The ray from this point on the image plane through the point on the lens is traced into the three-dimensional scene as illustrated in Figure 1a. While the refraction of the ray at a physically modeled lens could be computed by Snell's law, the computation can be simplified for small, thin lenses with the help of geometric optics as suggested by Cook et al. [CPC84].

For each sample point on the image plane, the algorithm starts by computing the intersection of the traditional ray through the center of the lens (corresponding to the position of the pinhole) and the focal plane at z_{focal}. Then, the ray from the chosen point on the lens plane through the previously computed point on the focal plane is traced into the scene. This is the correct direction since all rays starting at the same point on the image plane converge in the same point on the focal plane; in particular, the ray through the chosen sample point on the lens plane and the ray through the center of the lens.

Pharr and Humphreys [PH04] present this algorithm but implement it slightly differently: in the code of their ray tracer `pbrt` [PH04], rays do not start on the lens plane but on the view plane—which is identified with the near clipping plane—; therefore, the intersection point of the view plane with the pinhole ray through the center of the lens has to be shifted to the intersection of the view plane with the ray starting from the sample point on the lens plane. This shift is given by the distance of the latter point to the center of the

lens *scaled* by $(z_{\text{focal}} - z_{\text{view}})/z_{\text{focal}}$ since both rays intersect on the focal plane as illustrated in Figure 1a. Unfortunately, this scaling factor is missing in the source code of `pbrt`, version 1.02. However, the missing factor is very close to 1 if $z_{\text{view}} \ll z_{\text{focal}}$ and the factor is constant for all pixels of an image, i.e., it corresponds to a (usually very small) scaling of the lens radius.

Distributed ray tracing is based on stochastic sampling; however, the latter is not restricted to ray tracing. As mentioned by Cook [Coo86], stochastic sampling can also be applied to scan-line algorithms, e.g., "REYES" [CCC87], to generate depth of field and motion blur effects.

2.2. Splatting

While distributed ray tracing approximates physical depth of field very well, it has to trace many rays in a potentially complex scene for each pixel of the image; thus, the algorithm is most suited for off-line, non-interactive rendering. Less accurate but faster approaches to rendering with depth of field are based, for example, on splatting [KŽB03, PC82, Shi94]. In the context of GPU programming this kind of technique is sometimes characterized as "scattering." Since points that are not close to the focal plane appear blurred on the image plane, these blurred projections may be (pre-)computed and splatted into the framebuffer. In principle, this approach can also provide a good approximation to physical depth of field if all points—including occluded points—are splatted with correct depth blurring. Křivánek et al. [KŽB03] present such a system for point-based surfaces. However, the required information about (partially) occluded points is often unavailable; in particular, because many culling and deferred shading techniques are designed to skip the computation of this data in order to increase the rendering performance. Therefore, almost all approaches to depth-of-field rendering at interactive frame rates are based on post-processing color images with depth maps; e.g., RGBZ images stored in color framebuffers with depth buffers. In this case, the depth coordinate z_{point} is usually transformed before it is stored in a depth buffer. For example, the transformed depth z'_{point} could be defined as

$$z'_{\text{point}} \stackrel{\text{def}}{=} \frac{(z_{\text{point}} - z_{\text{near}})\, z_{\text{far}}}{(z_{\text{far}} - z_{\text{near}})\, z_{\text{point}}}. \tag{3}$$

The inverse relation for computing z_{point} from z'_{point} is:

$$z_{\text{point}} = \frac{z_{\text{far}} z_{\text{near}}}{z_{\text{far}} - z'_{\text{point}}(z_{\text{far}} - z_{\text{near}})}. \tag{4}$$

Splatting pixels of a previously computed image with depth map is an important post-processing method for rendering depth of field, which is characterized as a "forward-mapped z-buffer technique" by Demers [Dem04]. In fact, the first approach to depth of field in computer graphics by Potmesil and Chakravarty [PC82] employs this idea. Potmesil and Chakravarty also present a computation of the diameter c of the circle of confusion of a point at distance z_{point} that is out of focus; i.e., not on the focal plane at z_{focal}:

$$c \stackrel{\text{def}}{=} c_{z\to\infty} \frac{|z_{\text{point}} - z_{\text{focal}}|}{z_{\text{point}}} \quad \text{with} \quad c_{z\to\infty} \stackrel{\text{def}}{=} \frac{f}{N} \frac{z_{\text{view}}}{z_{\text{focal}}}. \tag{5}$$

Figure 1b illustrates a purely geometric construction of the circle of confusion while previous publications [CPC84, PC82] present a derivation employing laws of geometric optics. In Figure 1b, the point at z_{point} is projected to the image plane by intersecting this plane with a ray from the point at z_{point} to the center of the lens. All rays starting from the projected point on the image plane at $-z_{\text{view}}$ through any point on the lens plane converge in the same point on the focal plane at distance z_{focal}. Therefore, the lens of radius $f/(2N)$ is projected to a disk of radius r_{point} around the point at z_{point} with

$$r_{\text{point}} \stackrel{\text{def}}{=} \frac{f}{2N} \frac{|z_{\text{point}} - z_{\text{focal}}|}{z_{\text{focal}}}. \tag{6}$$

This disk of radius r_{point} is projected to the image plane—or in our case of Figure 1b equivalently to the view plane—by following rays through the center of the lens. Thus, the diameter of the projected circle at z_{view} is equal to

$$2 r_{\text{point}} \frac{z_{\text{view}}}{z_{\text{point}}} \tag{7}$$

which is equal to the diameter c of the circle of confusion defined in Equation (5) as can be shown with the help of Equation (6).

Note that the diameter c is specified in the same units as world coordinates. It can be converted to a length in pixels by multiplying it with the ratio of the image height h_{pix} in pixels divided by its height h in units of world coordinates.

While an improvement of the splatting algorithm by Potmesil and Chakravarty has been presented by Shinya [Shi94], it should be noted that all post-processing approaches to depth-of-field rendering have to result in a worse approximation than distributed ray tracing as their input data is insufficient since it does not include any information about partially occluded points that are not visible in the pinhole image.

2.3. Pre-Filtering

In general, most graphics hardware architectures—or more specifically, GPUs—are less suited for splatting approaches (i.e., scattering) than for techniques based on image interpolation or texture lookup operations (i.e., gathering). Therefore, the most efficient GPU implementations of depth-of-field effects employ hardware-supported filtering of the pinhole image, e.g., by mip-mapping. To this end, several blurred versions of the pinhole image with filter kernels of different sizes are generated and stored in texture images. Then, the diameter of the circle of confusion is computed for each pixel according to its depth, and the corresponding

blurred texture image is interpolated to determine the color of the resulting image with depth of field. This technique was first published by Rokita [Rok93] and is particularly well suited for GPU-based approaches as noted by Demers, who characterizes it as a "reverse-mapped (that is, texture-mapped) z-buffer technique" [Dem04].

Most variants of this method provide a rather coarse approximation of depth-of-field effects since they do not only suffer from the missing partially occluded points but also show several forms of color bleeding because depth is not taken into account by the pre-filtering. Therefore, several improvements to this technique have been published [Dem04, HSC*05, MvL00] and also patented [BLBD05]. Of particular interest in the context of our work is the "fast approximation" by Mulder and van Liere [MvL00] since it computes Gaussian image pyramids for an efficient blurring of the pinhole image. Unfortunately, none of the interactive post-processing techniques solves the problem of partially occluded points.

2.4. Blurring of Sub-Images

Barsky et al. [Bar04, BTCH05] have presented a non-interactive system, which avoids color bleeding by first decomposing the pinhole image into sub-images (corresponding to depth intervals in the depth map) before applying the blur filters. In this case, the problem of missing partially occluded points manifests itself in dark silhouettes of the sub-images. Moreover, the decomposition into discrete sub-images results in rendering artifacts at the locations where sub-images are separated as they are blurred independently. Barsky et al. address both problems by extending (i.e., extrapolating) sub-images with the help of rather complex image processing techniques. Unfortunately, this algorithm does not appear to be suitable for interactive rendering. While Barsky et al. [BTCH05] do not provide timings for their software implementation, Barsky [Bar04] reports rendering times of a few minutes per frame for a similar system.

In order to achieve an interactive rendering performance on GPUs, it is necessary to accelerate both, the blurring of sub-images and the disocclusion of partially occluded pixels. To this end, our approach employs GPU-based pyramidal image processing techniques suggested by Strengert et al. [SKE06, KS07] to efficiently extend sub-images of the pinhole image in the case of disocclusions, and to blur the resulting sub-images before compositing them in the resulting image with depth of field. While our algorithm features a similar structure as the work by Barsky et al., the algorithm and the implementation details differ drastically to allow for an interactive rendering performance on GPUs.

Another image blurring approach to depth of field is based on anisotropic diffusion of the pinhole image and was first published by Bertalmió et al. [BFS04]. Unfortunately, this approach will inevitably show artifacts that can only be alleviated by decomposing the pinhole image into sub-images according to its depth map (e.g., one foreground and one background sub-image) and processing these sub-images separately. However, to guarantee the same image quality, it is necessary to decompose the pinhole image into as many sub-images as required by the algorithm presented by Barsky et al. [BTCH05]. In this case, the uniform blurring of each sub-image is considerably more efficient.

3. Proposed Depth-of-Field Rendering

Similarly to the work by Barsky et al. [BTCH05], our proposed method decomposes an input image according to its depth map into sub-images (denoted by $I^{(i)}$ with integer indices i) and processes, i.e., blurs, these sub-images separately before they are blended from back to front (corresponding to descending indices i) to form the resulting image with depth of field. The overall data flow is illustrated in Figure 2. Each sub-image is initially set to the input pinhole image (Figure 2a) and its depth map (Figure 2b). Then, the following five steps are performed on each of the sub-images:

- culling of foreground pixels (Figure 2c),
- disocclusion of culled pixels (Figure 2d),
- matting according to the pixels' depth (Figure 2e),
- blurring (Figure 2f), and
- blending to accumulate the resulting image (Figure 2h).

In the remainder of this section, each of these steps is discussed in more detail.

3.1. Culling of Foreground Pixels

The processing of sub-images starts by computing a uniform blur radius for each sub-image as specified in Section 3.4. The particular radii are carefully chosen to achieve the optimum performance of the pyramidal blurring method. In other words, the blurring method dictates the blur radius $r_{\text{pix}}^{(i)}$ for each sub-image $I^{(i)}$. This blur radius is identified with a uniform radius of the circle of confusion that approximates the circle of confusion of all pixels of sub-image $I^{(i)}$. However, to determine which pixels of the input depth map should contribute to sub-image $I^{(i)}$, a depth coordinate $z^{(i)}$ has to be computed for each sub-image. Using Equation (5) the radius $r_{\text{pix}}^{(i)}$ can be related to $z^{(i)}$:

$$r_{\text{pix}}^{(i)} = r_{\text{pix}}^{(z\to\infty)} \frac{|z^{(i)} - z_{\text{focal}}|}{z^{(i)}} \text{ with } r_{\text{pix}}^{(z\to\infty)} \stackrel{\text{def}}{=} \frac{h_{\text{pix}}}{h} \frac{c_{z\to\infty}}{2} \tag{8}$$

where h_{pix} denotes the image height in pixels and h the same height in units of world coordinates. By solving this equation for $z^{(i)}$, the depth coordinate of sub-image $I^{(i)}$ can be computed as a function of $r_{\text{pix}}^{(i)}$:

$$z^{(i)} \stackrel{\text{def}}{=} \frac{z_{\text{focal}}}{1 + r_{\text{pix}}^{(i)} / r_{\text{pix}}^{(z\to\infty)}} \quad \text{for} \quad i < 0, \tag{9}$$

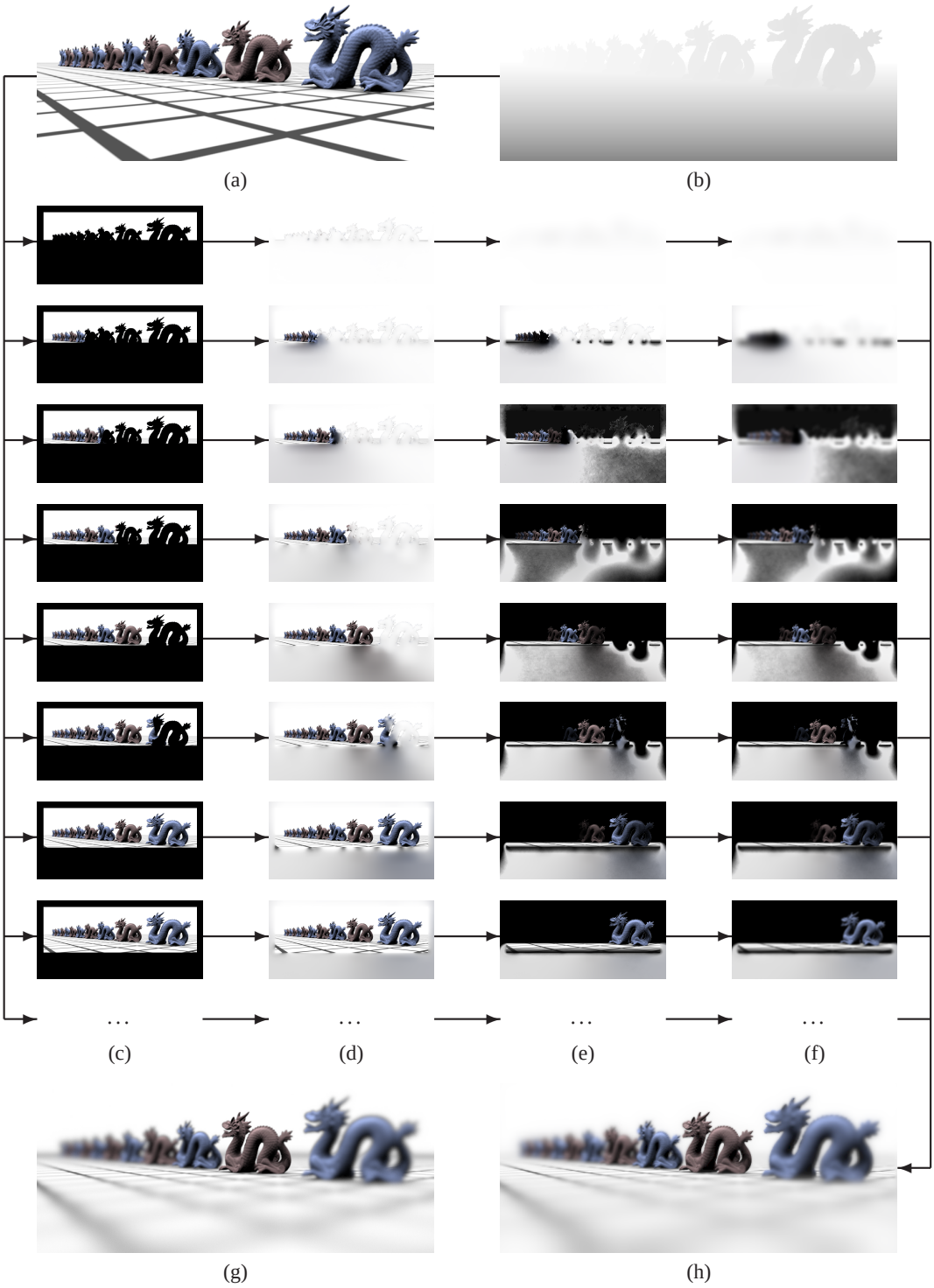

Figure 2: *Data flow in our method:* (a) *input pinhole image,* (b) *input depth map,* (c) *sub-images after culling of foreground pixels,* (d) *sub-images after disocclusion of culled pixels,* (e) *sub-images after matting,* (f) *sub-images after blurring,* (g) *ray-traced reference image,* (h) *blended result of our method. In (c)–(f) only the opacity-weighted RGB components of the sub-images* $I^{(5)}$ (top) *to* $I^{(-2)}$ (bottom) *are shown; all sub-images are embedded in an image of a larger size than the input image.*

$$z^{(0)} \stackrel{\text{def}}{=} z_{\text{focal}}, \tag{10}$$

$$z^{(i)} \stackrel{\text{def}}{=} \frac{z_{\text{focal}}}{1 - r_{\text{pix}}^{(i)} / r_{\text{pix}}^{(z\to\infty)}} \quad \text{for} \quad i > 0. \tag{11}$$

For $i < 0$, $z^{(i)}$ can be computed for any $r_{\text{pix}}^{(i)}$ while for $i > 0$—i.e., in the background of the focal plane—$z^{(i)}$ is only well defined for $r_{\text{pix}}^{(i)} < r_{\text{pix}}^{(z\to\infty)}$. This is to be expected because $c(z)$ is bound from above by $c_{z\to\infty}$ as illustrated in Figure 1c. However, since $r_{\text{pix}}^{(i)}$ is independent of $r_{\text{pix}}^{(z\to\infty)}$, there is always an index $i_{\max}$ such that $r_{\text{pix}}^{(j)} \geq r_{\text{pix}}^{(z\to\infty)}$ for all $j > i_{\max}$; thus, the depth coordinates $z^{(j)}$ are ill-defined. In most cases, the corresponding sub-images $I^{(j)}$ can be ignored since their blur radius is too large for any of the pixels of the input image; thus, these sub-images are necessarily empty and are set to transparent black. Note that $i_{\max}$ should be chosen such that $r_{\text{pix}}^{(i_{\max}-2)} < r_{\text{pix}}^{(z\to\infty)} < r_{\text{pix}}^{(i_{\max}-1)}$. Therefore, $z^{(i-1)}$, $z^{(i)}$, and $z^{(i+1)}$ might be ill-defined. This, however, does not affect the culling of foreground pixels which only depends on $z^{(i-2)}$ as explained next.

Once the depth coordinates $z^{(i)}$ of all sub-images are computed, the actual culling of foreground pixels can be performed. Due to the particular matting function discussed in Section 3.3, all pixels with depth coordinates $z < z^{(i-2)}$ are considered foreground pixels for sub-image $I^{(i)}$. These pixels are culled, i.e., set to transparent black, while all other pixels of $I^{(i)}$ are set to the opaque color of the input image as illustrated in Figure 2c.

3.2. Disocclusion of Pixels

The color of culled foreground pixels (i.e., transparent black) will bleed into regions of actual pixels of a sub-image when it is blurred as discussed and illustrated by Barsky et al. [BTCH05]. In our method, this would result in semitransparent silhouettes around culled foreground objects. In order to avoid these artifacts, the culled foreground pixels have to be "disoccluded;" i.e., color and depth of culled pixels have to be interpolated from the surrounding pixels that have not been culled.

To this end, we employ the GPU-based pyramidal interpolation method for scattered pixel data published by Strengert et al. [SKE06]. Interpolated colors are only assigned to transparent pixels, i.e., the previously culled foreground pixels. Additionally, new depth coordinates are interpolated for these pixels in order to enable the matting function, which is discussed next, to distinguish disoccluded pixels that should be part of a particular sub-image from disoccluded pixels that belong to its background and should therefore not contribute to the specific sub-image.

As a result of the disocclusion, all pixels of sub-image $I^{(i)}$ are set to an opaque color, which is either the pixel's color of the input image or an interpolated color for culled foreground pixels. Examples are presented in Figure 2d.

3.3. Matting of Sub-Images

In this work, "matting" refers to the computation of an opacity for all pixels included in a sub-image before the sub-image is blurred. The opacity of all pixels that do not belong to this sub-image is set to zero. As the decision whether a pixel is included in a sub-image only depends on its depth coordinate z, we specify the opacity by the matting function $\omega^{(i)}(z) \in [0,1]$, which is illustrated in Figure 3. The basic shape has to be adapted in some special cases as discussed below. Since opacity-weighted colors are employed, the matting function $\omega^{(i)}(z)$ is also multiplied to all color components; thus, $\omega^{(i)}(z)$ may also be considered a "weighting function." The effect of a weighted selection of pixels within a certain depth range is illustrated in Figure 2e. While the simple piecewise-linear shape of $\omega^{(i)}(z)$ was chosen to accelerate its evaluation in a GPU-based fragment program, there are some important features, which are discussed in detail next since they are crucial to avoid rendering artifacts.

First of all, the matting function is applied before blurring. This order is plausible if the pixels of the unblurred sub-image are considered sources of splats, which contribute to multiple sub-images as specified by the matting function. In fact, the matting function $\omega^{(i)}(z)$ cannot be evaluated after blurring the sub-image because splats of different depth coordinates can overlap; thus, there is no longer a unique depth coordinate z for each pixel.

Secondly, the matting function is usually a continuous function to guarantee smooth transitions of pixels—again considering them as the sources of splats—between multiple sub-images as their depth coordinates (or the focal distance or the lens radius) vary.

Thirdly, the integral of the matting function is not normalized. In fact, most pixels of the input image will contribute to three sub-images with a summed weight (or opacity) of the contributions of 2. This is necessary to address the discretization problem discussed by Barsky et al. [BTCH05], i.e., to generate blurred but completely opaque surfaces that cross multiple sub-images. In some sense, our solution extends sub-images in the direction of the negative z axis and therefore includes additional pixels of surfaces that cross multiple sub-images while surfaces are not extended if there are no further pixels within a certain depth range. On the other hand, Barsky et al. address the problem by extending sub-images in screen space and therefore require more expensive image analysis techniques. The additional opacity added by our method constitutes no problem since the blending of opacity-weighted colors discussed in Section 3.5 guarantees that a color of any opacity can be blended over an opaque pixel of the same color without changing it. An analogy from everyday life is to put an additional layer of the

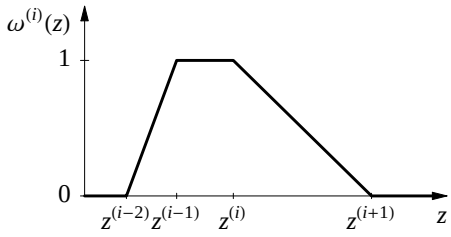

Figure 3: *The matting function $\omega^{(i)}(z)$ for sub-image $I^{(i)}$ with exemplary depth coordinates $z^{(i-2)}$ to $z^{(i+1)}$.*

same paint onto an already opaquely painted surface without changing the color of the surface.

As mentioned above, the matting function has to be adapted in some cases. For instance, the minimum index $i_{\min}$ and the maxium index $i_{\max}$ of sub-images might be fixed to guarantee an almost constant frame rate. In this case, the front ramp of $\omega^{(i_{\min})}(z)$ should be replaced by a constant plateau $\omega^{(i_{\min})}(z) = 1$ for $z < z^{(i_{\min})}$ in order to include all foreground pixels in the frontmost sub-image. The back ramp of $\omega^{(i_{\max})}(z)$ should be modified analogously. Furthermore, the same modifications of the back ramp are applied if $z^{(i)}$ or $z^{(i+1)}$ are ill-defined. For $\omega^{(i_{\max})}(z)$ we also replace the front ramp by a step function at $z^{(i_{\max}-2)}$ to avoid semitransparent silhouettes of objects in $I^{(i_{\max}-1)}$ in front of $I^{(i_{\max})}$. While the discontinuous step function in the modified matting function results in popping artifacts in animations, the semitransparent silhouettes are more objectable in static renderings. In most cases, however, both kinds of rendering artifacts are avoided by the mentioned choice of $i_{\max}$ such that $r_{\text{pix}}^{(i_{\max}-2)} < r_{\text{pix}}^{(z\to\infty)} < r_{\text{pix}}^{(i_{\max}-1)}$.

3.4. Blurring of Sub-Images

As illustrated in Figure 2f, the RGBA components of a sub-image are blurred by a pyramid method presented by Strengert et al. [SKE06] with the improvement of a 4×4 box analysis filter discussed by Kraus and Strengert [KS07]. For sub-image $I^{(i)}$ the number of reduce (and expand) operations in this pyramid method is chosen to be $|i|$. Since each level of the image pyramid corresponds to a scaling by 2, the blur radius $r_{\text{pix}}^{(i)}$ is proportional to $2^{|i|-1}$ for $i \neq 0$ while there is no blurring for $i = 0$. To estimate the constant factor, we compared the effect of the pyramidal blurring method with a convolution by a Gaussian filter of standard deviation σ_{pix}. The best fit for σ_{pix} varies between 0.8 and 0.9 depending on the screen position due to the structure of the image pyramid. Therefore, we estimate the blur radius by

$$\tilde{r}_{\text{pix}}^{(i)} \approx 0.85 \times 2^{|i|-1} \quad \text{for } i \neq 0 \quad \text{and} \quad \tilde{r}_{\text{pix}}^{(0)} = 0. \qquad (12)$$

Within our proposed method, however, this approximation underestimates the blurring. This is due to the particular matting discussed in Section 3.3 because a pixel that features a depth coordinate less than but close to the depth $z^{(i)}$ of sub-image $I^{(i)}$ also contributes to the sub-images $I^{(i-1)}$ and $I^{(i+1)}$. Either the sub-image $I^{(i-1)}$ or the sub-image $I^{(i+1)}$ features a blur radius that is twice as large as $r_{\text{pix}}^{(i)}$ (and even larger for $i = 0$). Although the weight of the contributions to these images are reduced by the matting function, the blended result will visually resemble a considerably stronger blur radius than suggested by $\tilde{r}_{\text{pix}}^{(i)}$. As a coarse approximation we estimate this effect by a factor of 2.0. Thus, we set the "perceived" blur radius $r_{\text{pix}}^{(i)}$ of sub-image $I^{(i)}$ for our method to:

$$r_{\text{pix}}^{(i)} \stackrel{\text{def}}{=} 2.0 \times 0.85 \times 2^{|i|-1} \quad \text{for } i \neq 0 \quad \text{and} \quad r_{\text{pix}}^{(0)} \stackrel{\text{def}}{=} 0. \qquad (13)$$

This definition is employed in the computation of $z^{(i)}$, which is described in Section 3.1.

3.5. Blending of Sub-Images

To composite all blurred sub-images in the output image $I_{\text{RGB}}^{\text{dof}}$, the color components of a sub-image $I_{\text{RGB}}^{(i)}$ have to be attenuated by the opacity components $I_{\text{A}}^{(j)}$ with $j < i$ of all sub-images in front of the sub-image $I^{(i)}$:

$$I_{RGB}^{\text{dof}} \stackrel{\text{def}}{=} \sum_{i=i_{\min}}^{i_{\max}} I_{\text{RGB}}^{(i)} \prod_{j=i_{\min}}^{i-1} \left(1 - I_{\text{A}}^{(j)}\right). \qquad (14)$$

This equation is implemented by blending each blurred sub-image "over" a color image buffer $\hat{I}_{\text{RGB}}$, which is initially cleared to black before it is used to accumulate the blurred sub-images:

$$\hat{I}_{\text{RGB}} \leftarrow I_{\text{RGB}}^{(i)} + \left(1 - I_{\text{A}}^{(i)}\right) \hat{I}_{\text{RGB}}. \qquad (15)$$

The RGB color components of sub-image $I^{(i)}$ must not be multiplied by its opacity components $I_{\text{A}}^{(i)}$ since the blurring discussed in the previous section implicitly computes opacity-weighted colors because color and opacity components are blurred simultaneously.

After all sub-images have been processed, the image buffer $\hat{I}_{\text{RGB}}$ stores the resulting output image $I_{\text{RGB}}^{\text{dof}}$ as illustrated in Figure 2h. In Figure 2g a ray-traced reference image is provided for a side-by-side comparison with our results, which are discussed next.

4. Discussion and Results

We validate the visual quality of images generated by our method with the help of side-by-side comparisons with renderings of the same scene by the ray tracer `pbrt` by Pharr and Humphreys [PH04]. In order to ensure a fair comparison, we employed the very same `pbrt` scene that is used to illustrate rendering with depth of field in Figures 6.8 and 6.9

Figure 4: *Comparison of renderings with depth of field generated by* `pbrt` (left column) *and images computed by our method* (right column) *for four settings of the* `pbrt` *parameter* `lensradius`*:* 0.001, 0.003, 0.01, *and* 0.03 (from top down).

of the book about `pbrt` [PH04]. The formal scene description is available on the CD-ROM accompanying this book.

The scene was first rendered by `pbrt` simulating a pinhole camera as depicted in Figure 2a. The depth map for this image was also computed as shown in Figure 2b. This data constitutes the input of our method, which allows us to enhance the pinhole image with depth-of-field effects for a large range of focal distances z_{focal} and lens radii $f/(2N)$; examples are presented in the right column of Figure 4. For comparison, we also show the resulting images computed by `pbrt` in the left column of Figure 4. As discussed in Section 3.4, our method approximates a convolution with a Gaussian filter corresponding to a Gaussian distribution of samples on the lens plane around the center of the lens.

Unfortunately, `pbrt`, version 1.02, only provides a uniform sampling of the lens disk, which results in a different visual effect, known as "Bokeh" in photography. Since these differences are of no concern in our case, we implemented a Gaussian distribution of samples on the lens plane. To this end, the uniform sampling of the lens implemented by the function call

```
ConcentricSampleDisk(sample.lensU,
   sample.lensV, &lensU, &lensV);
```

on page 270 of the `pbrt` book has to be replaced by:

```
GaussianSampleDisk(sample.lensU,
   sample.lensV, &lensU, &lensV);
```

which calls our implementation of a Box-Muller transform

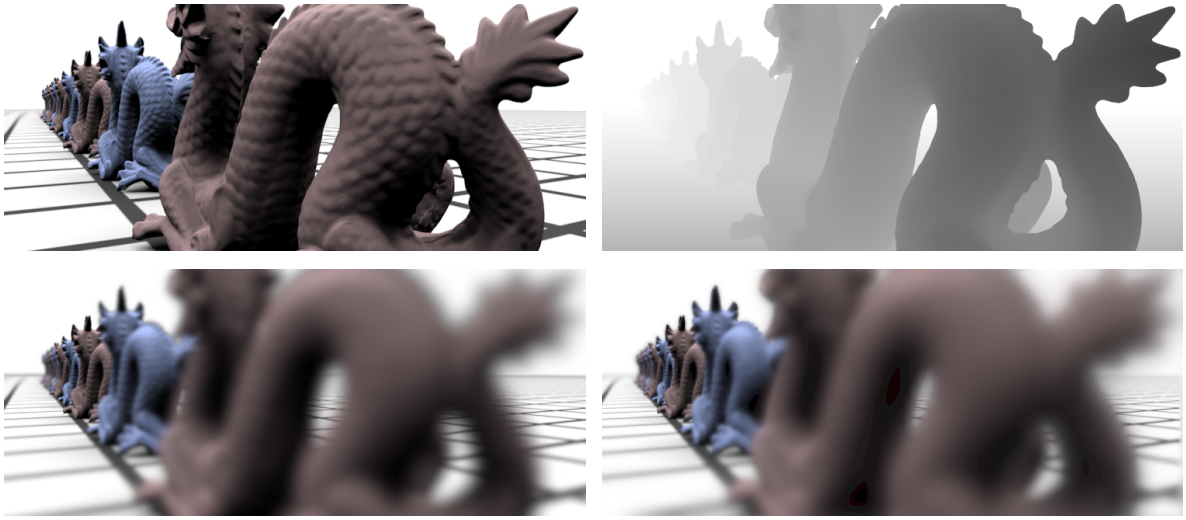

Figure 5: *Comparison of a rendering with depth of field generated by* `pbrt` (bottom, left) *and an image computed by our method* (bottom, right). *The latter is based on a pinhole image generated by* `pbrt` (top, left) *and its depth map* (top, right).

in polar form [BM58]. We added its definition to the `pbrt` file `mc.cpp`:

```
COREDLL void GaussianSampleDisk(float u1,
      float u2, float *x, float *y) {
   float r = sqrtf(fabsf(2.0f *
      (u1 > 0.0f ? logf(u1) : 0.0f)));
   float theta = 2.0f * M_PI * u2;
   *x = r * cosf(theta);
   *y = r * sinf(theta);
}
```

Our code modifications implicitly change the meaning of the `pbrt` parameter `lensradius` from the radius of a finite disk (corresponding to the lens radius $f/(2N)$) to the standard deviation σ of a Gaussian distribution.

Apart from the image dimension $w_{\text{pix}} \times h_{\text{pix}}$ and the depth coordinates z_{focal}, z_{near}, and z_{far}, our implementation requires the specification of $r_{\text{pix}}^{(z\to\infty)}$, which can be computed in terms of the parameters specified in `pbrt` scene description files:

$$r_{\text{pix}}^{(z\to\infty)} \stackrel{\text{def}}{=} \frac{h_{\text{pix}}}{h}\,\frac{c_{z\to\infty}}{2} = \frac{h_{\text{pix}}}{2z_{\text{focal}}\tan\left(\gamma_{\text{fovy}}/2\right)}\,\frac{f}{2N}. \quad (16)$$

With these parameters, our method allows us to compute similar depth-of-field effects from a pinhole image as generated by `pbrt`'s distributed ray tracing. Of course, we still cannot hope to generate identical results since our method is based on image post-processing of 2D images while distributed ray tracing has access to the actual three-dimensional scene geometry. In fact, the side-by-side comparisons in Figures 4 and 5 indicate that the blur radius $r_{\text{pix}}^{(i)}$ is still underestimated by our computation discussed in Section 3.4—resulting in too much blur compared to the reference images. Nonetheless, the visual similarity to the reference images appears to be sufficient for many—in particular interactive—applications. Moreover, hardly any of the typical rendering artifacts of interactive depth-of-field algorithms occur; in particular, there are no darkened silhouettes around objects and almost no incorrect color bleeding except for very large lens radii. However, in many cases an antialiased input image and/or depth map will result in artifacts due to the interpolation of depth coordinates at silhouettes of objects; thus, in general it is preferable to provide our method with non-antialiased input images.

We tested our OpenGL implementation on a Windows XP system with an NVIDIA GeForce 7900 GTX GPU equipped with 512 MB. All sub-images were embedded in 1024×1024 images and processed in four half-float RGBA framebuffer objects of size 1800×1024. The example depicted in Figure 2 (also depicted in the third row of Figure 4) required the processing of 12 sub-images ($I^{(5)}$ to $I^{(-6)}$) and rendered in about 70.4 ms (14.2 fps). Smaller circles of confusion result in the processing of fewer sub-images; for example, the image depicted in the first row of Figure 4 required 6 sub-images ($I^{(2)}$ to $I^{(-3)}$) and rendered in about 34.5 ms (29 fps).

5. Conclusion and Future Work

We have demonstrated that high-quality rendering of depth-of-field effects is within the reach of GPU-based implementations of the sub-image blurring approach, which was originally proposed by Barsky et al. [BTCH05]. Furthermore, we successfully implemented the required image operations—in particular, blurring and disocclusion of pixels—by pyra-

mid methods to achieve a rendering performance that is suitable for real-time rendering in the case of not too large image sizes and small circles of confusion. Many applications could benefit from this technique; for example, interactive photography retouching, virtual reality renderers, games, etc.

Future work includes performance optimizations of our prototypical implementation, research on the perceived blurring performed by our method, and on possibilities to implement approximations to non-Gaussian filters; for example, by replacing the pyramidal blurring by alternative methods based on infinite impulse response filters or fast Fourier transforms.

Acknowledgments

The authors would like to thank the anonymous reviewers for their detailed comments, Matt Pharr and Gregory Humphreys for `pbrt`, and Jared Sohn for the extended channel film plug-in for `pbrt`. The dragon model appears courtesy of the Stanford University Scanning Repository; the `pbrt` scene appears courtesy of Gregory Humphreys and Matt Pharr.

References

[Bar04] BARSKY B. A.: Vision-realistic rendering: Simulation of the scanned foveal image from wavefront data of human subjects. In *APGV '04: Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization* (2004), pp. 73–81.

[BFS04] BERTALMÍO M., FORT P., SÁNCHEZ-CRESPO D.: Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *3DPVT 2004: Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization, and Transmission* (2004), pp. 767–773.

[BHK*03] BARSKY B. A., HORN D. R., KLEIN S. A., PANG J. A., YU M.: Camera models and optical systems used in computer graphics: Parts i and ii. In *ICCSA 2003: Proceedings of the 2003 International Conference on Computational Science and its Applications* (2003), pp. 246–265.

[BLBD05] BASTOS R. M., LEW S. D., BEESON C. A., DEMERS J. E.: Depth-of-field effects using texture lookup. US patent 6,975,329, 2005.

[BM58] BOX G. E. P., MULLER M. E.: A note on the generation of random normal deviates. *The Annals of Mathematical Statistics 29*, 2 (1958), 610–611.

[BTCH05] BARSKY B. A., TOBIAS M. J., CHU D. P., HORN D. R.: Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques. *Graphical Models 67*, 6 (2005), 584–599.

[CCC87] COOK R. L., CARPENTER L., CATMULL E.: The reyes image rendering architecture. In *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (1987), pp. 95–102.

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1 (1986), 51–72.

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. In *SIGGRAPH '84: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (1984), pp. 137–145.

[CPC93] COOK R. L., PORTER T. K., CARPENTER L. C.: Pseudo-random point sampling techniques in computer graphics. US patents 4,897,806; 5,025,400; and 5,239,624, 1990-1993.

[Dem04] DEMERS J.: Depth of field: A survey of techniques. In *GPU Gems* (2004), Fernando R., (Ed.), Addison Wesley, pp. 375–390.

[HSC*05] HENSLEY J., SCHEUERMANN T., COOMBE G., SINGH M., LASTRA A.: Fast summed-area table generation and its applications. *Computer Graphics Forum 24* (2005), 547–555.

[KS07] KRAUS M., STRENGERT M.: Pyramid filters based on bilinear interpolation. In *Proceedings GRAPP 2007 (Volume GM/R)* (2007), pp. 21–28.

[KŽB03] KŘIVÁNEK J., ŽÁRA J., BOUATOUCH K.: Fast depth of field rendering with surface splatting. In *Proceedings of Computer Graphics International 2003* (2003), p. 196.

[MvL00] MULDER J. D., VAN LIERE R.: Fast perception-based depth of field rendering. In *VRST '00: Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (2000), pp. 129–133.

[PC82] POTMESIL M., CHAKRAVARTY I.: Synthetic image generation with a lens and aperture camera model. *ACM Trans. Graph. 1*, 2 (1982), 85–108.

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004.

[Rok93] ROKITA P.: Fast generation of depth of field effects in computer graphics. *Computers & Graphics 17*, 5 (1993), 593–595.

[Shi94] SHINYA M.: Post-filtering for depth of field simulation with ray distribution buffer. In *Proceedings of Graphics Interface '94* (1994), pp. 59–66.

[SKE06] STRENGERT M., KRAUS M., ERTL T.: Pyramid methods in GPU-based image processing. In *Proceedings Vision, Modeling, and Visualization 2006* (2006), pp. 169–176.