Using Opaque Image Blur for Real-Time Depth-of-Field Rendering and Image-Based Motion Blur

Martin Kraus

Department of Architecture, Design, and Media Technology
Aalborg University
Sofiendalsvej 11, 9200 Aalborg SV; Denmark
email: martin@create.aau.dk
www: http://personprofil.aau.dk/121021

Abstract

While depth of field is an important cinematographic means, its use in real-time computer graphics is still limited by the computational costs that are necessary to achieve a sufficient image quality. Specifically, color bleeding artifacts between objects at different depths are most effectively avoided by a decomposition into sub-images and the independent blurring of each sub-image. This decomposition, however, can result in rendering artifacts at silhouettes of objects. We propose a new blur filter that increases the opacity of all pixels to avoid these artifacts at the cost of physically less accurate but still plausible rendering results. The proposed filter is named "opaque image blur" and is based on a glow filter that is applied to the alpha channel. We present a highly efficient GPU-based pyramid algorithm that implements this filter for depth-of-field rendering. Moreover, we demonstrate that the opaque image blur can also be used to add motion blur effects to images in real time.

First presented at the International Conference on Computer Graphics Theory and Applications 2011 (GRAPP 2011), extended and revised for JVRB

Keywords: depth of field, blur, motion blur, glow, visual effect, real-time rendering, image processing, GPU, pyramid algorithm

1 Introduction

Depth of field in photography specifies the depth range of the region in front and behind the focal plane that appears to be in focus for a given resolution of the film. As limited depth of field is a feature of all real camera systems (including the human eye), plausible depth-of-field effects can significantly enhance the illusion of realism in computer graphics. Moreover, limited depth of field can be used to guide the attention of viewers. In fact, it is routinely used for this purpose in movies — including computer-animated movies. In recent years, it has also been used in several computer games and first applications in graphical user interfaces have been demonstrated.

There are various approaches to the computation of depth-of-field effects, which provide different trade-offs between image quality and rendering performance. Current techniques for real-time performance are based on a single pinhole image with infinite depth of field since the performance of this approach is independent of the scene complexity and graphics hardware is optimized to compute this kind of imagery. One of the most prominent rendering artifacts in this approach is color bleeding between objects at different depths.

One way to avoid these particular artifacts is the decomposition of the pinhole image into sub-images according to the depth of pixels and the independent processing of each sub-image. The main remaining arti-

urn:nbn:de:0009-6-38199, ISSN 1860-2037

fact is caused by partial occlusions. More specifically, the problem is caused by pixels of one sub-image that are occluded by the pinhole version of another sub-image in the foreground but only partially occluded by the blurred version of that sub-image. Various approaches have been suggested to address these disoccluded pixels; however, all proposed methods tend to be the most costly part of the respective algorithm in terms of rendering performance.

In this work, we solve the problem by completely avoiding disocclusions of pixels; i.e., instead of trying to render correct images with disoccluded pixels, we render plausible images without disoccluded pixels. The key element of our approach is a blurring method that does not disocclude pixels; i.e., a blur filter that does not reduce the opacity of any pixel. This filter allows us to design a considerably simplified algorithm for sub-image blurring, which is presented in Section 3.1. The details of the employed blurring method — named "opaque image blur" — are discussed in Section 3.2 and results are presented in Section 3.3.

While the application of the opaque image blur to depth-of-field rendering has been described before [Kra11], this work also demonstrates an application of the opaque image blur to image-based motion blur in Section 4. Here, we deliberately avoid pyramid algorithms in order to show that the opaque image blur is neither limited to depth-of-field rendering nor to pyramid algorithms.

Conclusions and plans for future work are discussed in Sections 5 and 6. First, however, we discuss previous work on depth-of-field rendering.

2 Previous Work

Physically correct depth-of-field effects in off-line rendering are most commonly computed by stochastic sampling of a camera lens of finite size [CPC84]. Several implementations with various improvements have been published [CCC87, HA90, KMH95, PH04].

Splatting of image points with the help of a depth-dependent point-spread function was proposed even earlier than stochastic sampling [PC82] and can also produce accurate images if all points of a scene are taken into account (including points that are occluded in a pinhole image). While the first implementations were software-based [Shi94, KŽB03], more recent systems employ features of modern graphics hardware [LKC08].

The main drawback of stochastic sampling and splatting approaches with respect to performance is the dependency on the scene complexity, i.e., the rendering of the depth-of-field effect is more costly for more complex scenes. Furthermore, the computations have to be integrated into the rendering process and, therefore, often conflict with optimizations of the rendering pipeline, in particular in the case of hardware-based pipelines. Therefore, real-time and interactive approaches to depth-of-field rendering are based on image post-processing of pinhole images with depth information for each pixel. These approaches are independent of the scene complexity and they are compatible with any rendering method that produces pinhole images with depth information.

The highest performance is achieved by computing a series of differently blurred versions (e.g., in the form of a mipmap hierarchy) and determining an appropriately blurred color for each pixel based on these filtered versions [Rok93, Dem04, Ham07, LKC09]. However, it appears to be impossible to avoid all rendering artifacts in these approaches — in particular color bleeding (also known as intensity leakage) between objects at different depths.

The most effective way to avoid these artifacts is the decomposition of the pinhole image into sub-images according to the depth of pixels [Bar04]. Each sub-image is then blurred independently and the blurred sub-images are blended onto each other to accumulate the result. However, the decomposition into sub-images can introduce new artifacts at the silhouettes of sub-images, which are addressed in different ways by the published systems [BTCH05, KS07a].

Hybrid approaches are also possible; in particular, the scene geometry can be rendered into different layers which are then blurred independently [Sco92, KB07, KTB09, LES09]. This can avoid artifacts between layers but requires non-uniform blurring techniques, which require a considerably higher performance. Another hybrid approach combines ray tracing with multi-layer rendering [LES10].

This work is based on the system presented by Kraus and Strengert [KS07a] but eliminates artifacts at silhouettes by avoiding partial disocclusions of pixels. This is achieved by employing a particular blur filter, which does not reduce the opacity of any pixel. Thus, the effect is similar to applying a glow filter [JO04] to the opacity channel. In principle, a grayscale morphological filter [Ste86] could also be used for this purpose; however, the proposed GPU-based compu-

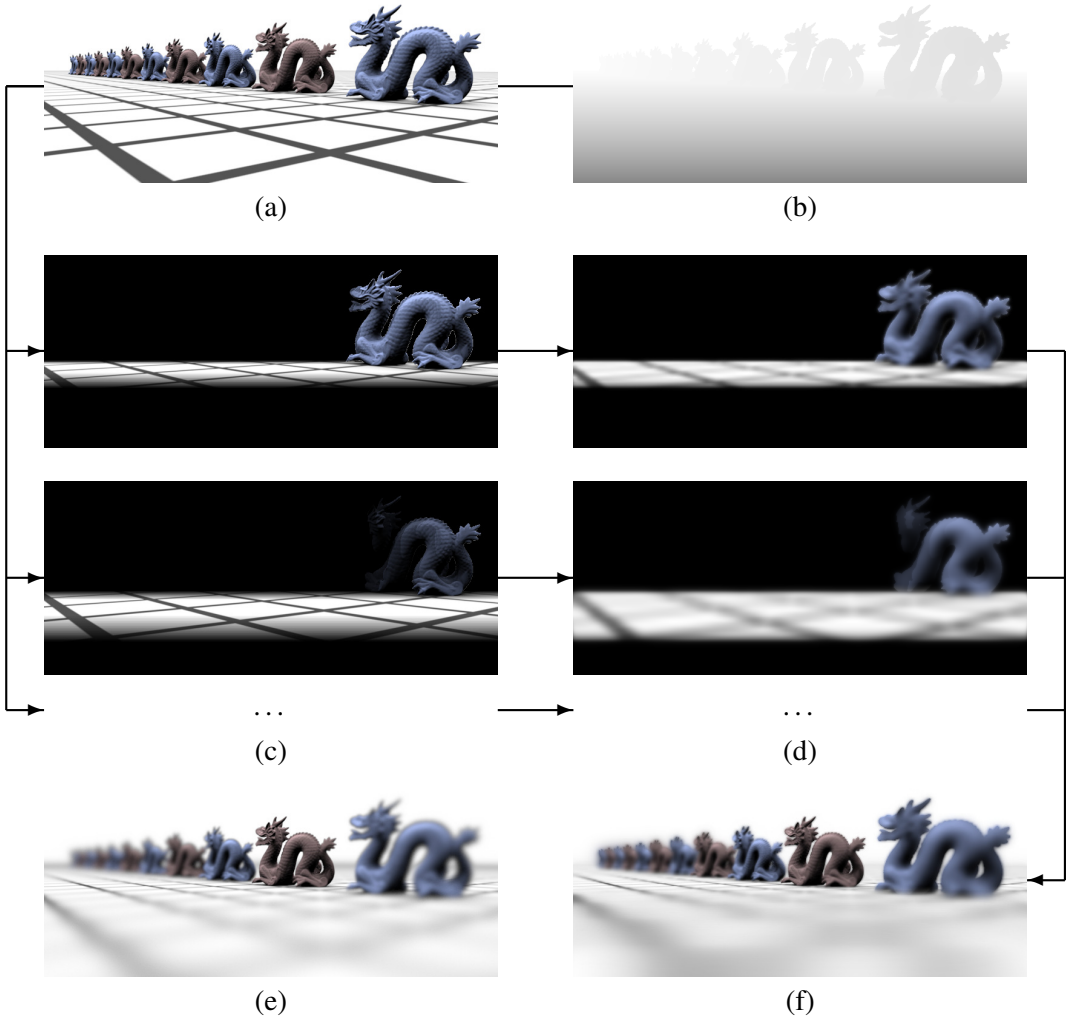

Figure 1: Data flow in our method: *(a)* input pinhole image, *(b)* input depth map, *(c)* sub-images after matting, *(d)* sub-images after opaque image blur (see Figure 3), *(e)* ray-traced reference image [PH04], *(f)* blended result of our method. In (c) and (d) only the opacity-weighted RGB components of the (-1)st sub-image *(top)* and the (-2)nd sub-image *(bottom)* are shown.

tation of glow filters offers a considerably higher performance.

3 Depth-of-Field Blur

The proposed opaque image blur filter was motivated by the problem of depth-of-field rendering. Therefore, a new depth-of-field rendering algorithm that employs the opaque image blur is presented in the next section before the actual opaque image blur is discussed in Section 3.2.

3.1 Depth-of-Field Rendering with the Opaque Image Blur

The proposed algorithm decomposes a pinhole image with depth information into sub-images that correspond to certain depth ranges as illustrated in Figure 1. Similarly to previously published methods [BTCH05, KS07a], the algorithm consists of a loop over all sub-images starting with the sub-image corresponding to the farthest depth range. For each sub-image, the following three steps are performed:

1. The pinhole image is matted according to the pixels' depth (Figure 1c).

2. The matted sub-image is blurred using the "opaque image blur" discussed in Section 3.2

(Figure 1d).

3. The blurred sub-image is blended over the content of an (initially cleared) framebuffer, in which the result is accumulated (Figure 1f).

Note that no disocclusion of pixels is necessary whereas the disocclusion step in previous published systems tends to be the most costly computation [BTCH05, KS07a].

The matting and blending in our prototype is performed as in the system by Kraus and Strengert. Specifically, we approximate the blur radius r_i of the i-th sub-image by:

$$r_i \stackrel{\text{def}}{=} 1.7 \times 2^{|i|-1} \text{ for } i \neq 0 \quad \text{and} \quad r_0 \stackrel{\text{def}}{=} 0. \quad (1)$$

The blur radius is specified in pixels and corresponds to the radius of the circle of confusion. Thus, the corresponding depth z_i of the i-th sub-image can be computed with the thin lens approximation. The result is:

$$z_i \stackrel{\text{def}}{=} \frac{z_{\text{focal}}}{1 + r_i/r_\infty} \quad \text{for} \quad i < 0, \quad (2)$$

$$z_0 \stackrel{\text{def}}{=} z_{\text{focal}}, \quad (3)$$

$$z_i \stackrel{\text{def}}{=} \frac{z_{\text{focal}}}{1 - r_i/r_\infty} \quad \text{for} \quad i > 0. \quad (4)$$

Here, z_{focal} is the depth of the focal plane and r_∞ is the blur radius of infinitely distant points. r_∞ can be expressed in terms of the focal length f, the f-number N, the field-of-view angle in y direction γ_{fovy}, and the height of the image h_{pix} in pixels:

$$r_\infty \stackrel{\text{def}}{=} \frac{h_{\text{pix}}}{2 z_{\text{focal}} \tan(\gamma_{\text{fovy}}/2)} \frac{f}{2N}. \quad (5)$$

The depths z_{i-2}, z_{i-1}, z_i, and z_{i+1} of four sub-images are used to define the matting functions $\omega_i(z)$ for pixels of the i-th sub-image as illustrated in Figure 2. The specific matting function is designed to allow for an efficient implementation in shader programs. Note that the weighting functions for the foremost and backmost sub-images are adjusted to remove the ramps at the extremes; i.e., the weight is set to 1 where there is no other sub-image that would include a pixel.

The matting of the i-th sub-image is then performed in a fragment shader by looking up the depth z of each pixel, evaluating the weighting function $\omega_i(z)$ and multiplying it to the RGBA color of the pinhole image, where the opacity A of the pinhole image is set to 1.

Figure 2: Illustration of the matting function $\omega_i(z)$ for the i-th sub-image based on the depths z_{i-2} to z_{i+1}.

After matting, each sub-image is blurred as described in Section 3.2. The resulting blurred colors RGBA_{sub} of the sub-image are then blended with the colors RGB_{buf} of a color buffer, which is initially set to black. The blending employs the "over" operator for pre-multiplied (i.e., opacity-weighted) colors [PD84] since the sub-images are processed from back to front:

$$\text{RGB}_{\text{buf}} \leftarrow \text{RGB}_{\text{sub}} + (1 - \text{A}_{\text{sub}}) \times \text{RGB}_{\text{buf}}. \quad (6)$$

After the frontmost sub-image has been processed, the colors RGB_{buf} represent the resulting image with the computed depth-of-field effect.

While this algorithm is significantly less complex than previously published algorithms for sub-image processing [BTCH05, KS07a], it strongly depends on an image blur that does not disocclude pixels, i.e., the image blur must not decrease the opacity of any pixel. The next section describes such a filter.

3.2 Opaque Image Blur

The proposed "opaque image blur" of sub-images guarantees not to disocclude pixels in order to avoid rendering artifacts that are caused by partial occlusions, which are most visible at silhouettes of objects in sub-images [BTCH05]. For an example of disoccluded pixels, consider the blurred image in Figure 3c: the blurred silhouettes become semitransparent (i.e. darker in Figure 3c), which means that previously occluded pixels of layers in the background become partially visible, i.e. they are disoccluded by the increased transparency that is caused by the blur.

To avoid these disocclusions, the opaque blur of an RGBA image only increases the opacity of pixels. This is achieved by three steps, which are illustrated in Figure 3.

1. A glow filter [JO04] is applied to the A channel of the RGBA image (Figures 3b and 3d). This glow filter must not decrease the A channel of any pixel. The result is called A_{glow}.

2. A standard blur filter is applied to all channels of the original RGBA image (Figures 3a and 3c). The result is called $\text{RGBA}_{\text{blur}}$.

3. The opacity of the blurred image is replaced by the opacity computed by the glow filter

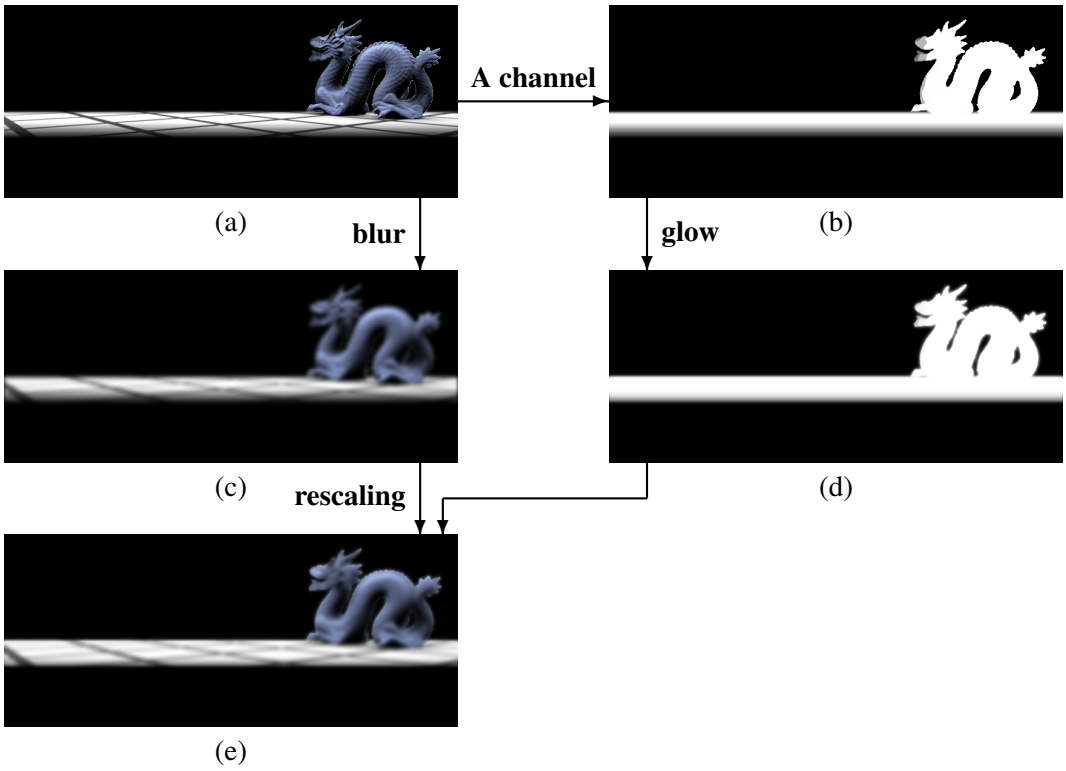

Figure 3: Data flow in the opaque image blur: *(a)* input RGBA image (only RGB is shown), *(b)* opacity (i.e., A channel) of the input image visualized as gray-scale image, *(c)* standard blur filter applied to the input RGBA image (A is not shown), *(d)* glow filter applied to the opacity of the input image, *(e)* resulting opaque image blur.

(Figure 3e). To this end, the blurred colors are rescaled since they are considered opacity-weighted colors. The result is called $\mathrm{RGBA}_{\mathrm{sub}}$:

$$\mathrm{RGBA}_{\mathrm{sub}} \stackrel{\mathrm{def}}{=} \mathrm{RGBA}_{\mathrm{blur}} \times \frac{\mathrm{A}_{\mathrm{glow}}}{\mathrm{A}_{\mathrm{blur}}} \qquad (7)$$

For each sub-image of the algorithm described in Section 3.1, the result $\mathrm{RGBA}_{\mathrm{sub}}$ is then used in Equation 6.

Without the color rescaling, the increased opacity $\mathrm{A}_{\mathrm{glow}}$ would result in dark silhouettes around objects of full opacity. To avoid artifacts, the range of the glow filter should not be larger than the range of the blur filter. Otherwise, the color rescaling is likely to increase colors that are unrelated to the objects that caused the increased opacity.

While any non-decreasing glow filter and any standard blur filter can be used to implement an opaque image blur, we propose to employ pyramid algorithms for both filters because of their favorable performance on GPUs. Moreover, pyramid versions of the glow filter and the blur filter can share a common analysis phase, which reduces the total computational costs by about one quarter.

For the standard blur we employ a pyramidal blur [SKE06] with a 4×4 box analysis filter [KS07b]. The analysis phase of this pyramidal blur corresponds to a mipmap generation; however, the number of required levels is limited by the strength of the blur. For the algorithm discussed in Section 3.1, $|i|$ levels of the image pyramid have to be computed for the i-th sub-image. The synthesis phase of the pyramidal blur iteratively expands the i-th pyramid level to the original size with a synthesis filter that corresponds to a biquadratic B-spline interpolation [SKE06].

The glow filter makes use of the opacity $\mathrm{A}_{\mathrm{ana}}$ of the exact same analysis pyramid as the pyramidal blur. However, the synthesis is modified in order to guarantee that the opacity of no pixel is decreased. This is achieved by multiplying the transparency of each expanded level, i.e., $1 - \mathrm{A}_{\mathrm{exp}}$, with the transparency of the corresponding analysis level of the same size, i.e., $1 - \mathrm{A}_{\mathrm{ana}}$. The resulting transparency determines the opacity $\mathrm{A}_{\mathrm{syn}}$ of the new synthesis level for the pyra-

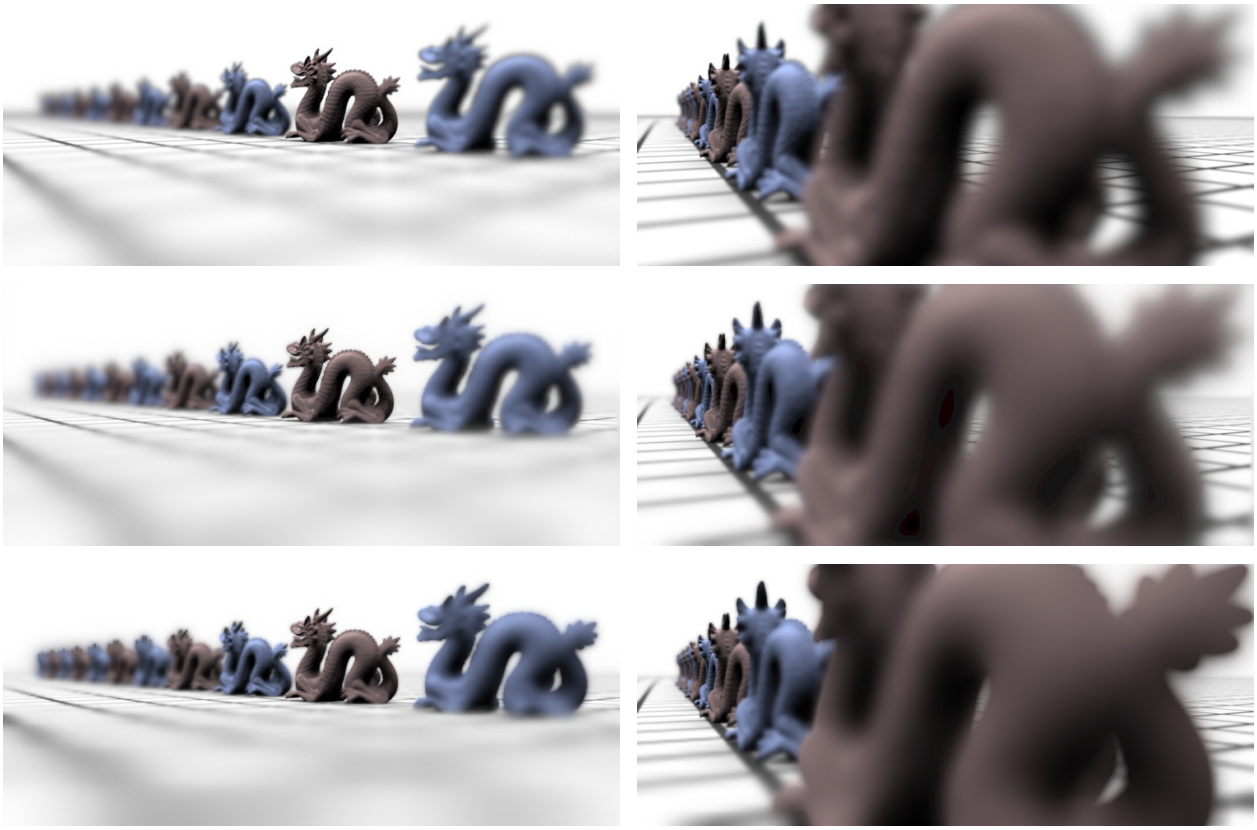

Figure 4: Comparison of renderings with depth of field generated by `pbrt` [PH04] *(top row)*, the method published by Kraus and Strengert *(middle row)*, and the proposed method *(bottom row)*.

midal glow:

$$A_{syn} \stackrel{\text{def}}{=} 1 - (1 - A_{exp})(1 - A_{ana}) \quad (8)$$

$$= A_{exp} + A_{ana} - A_{exp}A_{ana}. \quad (9)$$

It is straightforward to implement this blending in a fragment shader.

After both pyramid algorithms have been performed, the blurred colors have to be rescaled to the opacity computed by the glow filter as discussed above. For efficiency, this step should be combined with the final synthesis step of the pyramidal blur and the final synthesis step of the pyramidal glow. Since the final synthesis steps expand the image to the full size of the input image, it is particularly beneficial to implement these steps as efficiently as possible.

3.3 Results

Figure 4 compares two images generated by our method with ray-traced images computed with pbrt [PH04] and images produced by the method proposed by Kraus and Strengert. Obviously, our method avoids any disocclusion which results in too opaque objects. On the other hand, color bleeding between objects at different depths is still avoided. Due to the nonlinear glow, the silhouettes of objects are too sharp in our method. This is, however, a consequence of the particular glow filter employed in this work. We assume that there are alternative glow filters that produce better visual results.

Our method performed at 110 ms per frame on a 13" MacBook Pro with an NVIDIA GeForce 320M, while the method by Kraus and Strengert required 208 ms, i.e., almost twice as much. While our implementation of both methods was not optimized and includes some copy operations that could be avoided, we don't see many possibilities to optimize the disocclusion part of the method by Kraus and Strengert; therefore, it is unlikely that we overestimate the improvement gained by avoiding the disocclusion. Furthermore, it should be noted that our algorithm is easier to implement since algorithms for the disocclusion of pixels tend to be rather complex. On the other hand, the performance of our method is worse than methods based on computing differently blurred versions of a pinhole image [Ham07, LKC09].

Figure 5: Image of a Piaggio Vespa PX 125 E [Dan07] *(left)* and a manually generated alpha mask of the foreground *(right)*.

The image quality achieved by the proposed algorithm is also between these two approaches: it avoids artifacts such as color bleeding between objects of different depths, which often occur in methods that do not use sub-images [Ham07, LKC09]. On the other hand, the image quality is reduced in comparison to other approaches based on sub-images [BTCH05, KS07a] because of the missing disocclusions and the particular opaque image blur.

Note that the implementation of our method revealed an interesting side effect: artifacts at the boundaries of the view port are effectively avoided by the opaque image blur if there is a border of transparent black pixels. In particular, a border width of one pixel is sufficient. In contrast, the system by Kraus and Strengert requires an embedding of the view port in a larger framebuffer and relies on extrapolation to generate pixel data that provides continuous blurring at the boundaries of the original view port.

4 Motion Blur

In order to illustrate another application of the opaque image blur proposed in Section 3.2, this section employs the opaque image blur to generate motion blur. Previous work on image-based motion blur includes work by Rosado [Ros08], which does not discuss disocclusion of pixels and work by Brostow et al. [BE01], who used multiple images to generate motion blur and therefore were not plagued by the disocclusion of pixels.

4.1 Image-Based Opaque Motion Blur

In this work, we consider only the addition of motion blur to user-specified areas of an image without blur. The proposed algorithm is illustrated with the help of the image in Figure 5a. In addition to the RGB colors of the input image, opacities (i.e. an A channel) has to be supplied to specify the area of the image that should be blurred such that the illusion of motion blur is achieved. In our case, opacities are specified manually to mask the red Vespa. In computer-generate images, this masking could be achieved automatically by tagging pixels that belong to the moving object; e.g., by rasterizing object IDs into a separate image buffer.

An alternative approach to the proposed mask-based method is to render the moving object into a different image buffer than the rest of the scene. However, this would mean that areas that are occluded by the moving object become visible and have to be shaded, which would usually cost rendering performance. Therefore, this alternative approach is not considered here.

The goal of the proposed method is to blur the masked area, which represents the foreground, without blurring any of the background nor with disoccluding any of the background areas that are occluded by the (masked) foreground. As illustrated in Figure 6, the opaque image blur presented in Section 3.2 can be employed as follows:

1. A glow filter is applied to the opacities, i.e. the A channel of the RGBA image (Figures 6b and 6d). The result is called $\mathrm{A}_{\mathrm{glow}}$. The specific glow filter is discussed in Section 4.2.

2. The RGB channels of the input RGBA image are multiplied with the A channel (Figure 6c). Then a 1D-box filter in the direction of motion is applied to the opacity-weighted RGB channels and the original A channel (Figures 6a and 6c). The result is called $\mathrm{RGBA}_{\mathrm{blur}}$. (Of course, other filters than box filters could be used just as well.)

3. The opacity of the blurred image is replaced by the opacity computed by the glow filter (Figure 6e). The opacity-weighted colors are rescaled correspondingly; see Equation 7.

4. In an additional step, the result of the previous step is blended over the input RGBA image (Figures 6f and 6g).

Apart from the multiplication with opacities in step 1 and the additional blending step, this process is identical to the opaque image blur proposed in Section 3.2. However, some alternatives and extensions should be mentioned.

Firstly, it would be possible to employ a pyramidal blur filter instead of the box filter. However, this would require to rotate the image to align the direction

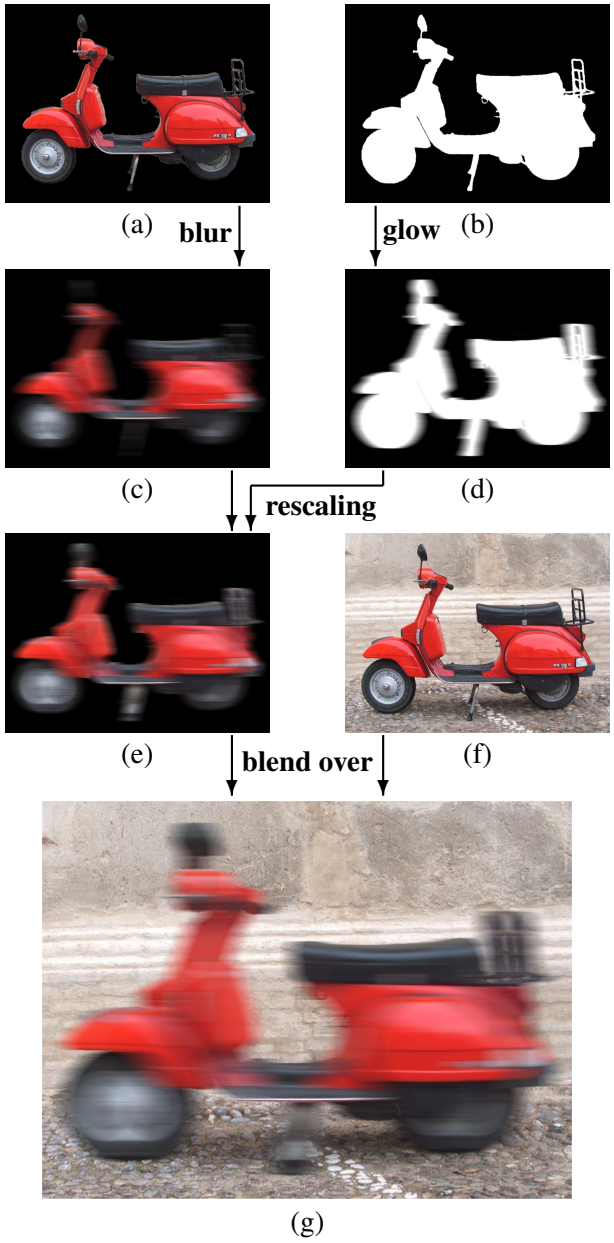

Figure 6: Data flow in the opaque motion blur: *(a)* opacity-weighted colors of the input image (i.e. multiplied with the A channel), *(b)* opacity (i.e., A channel) of the input image visualized as gray-scale image, *(c)* 1D box filter applied to (a) (A channel is not shown), *(d)* glow filter applied to the opacity of the input image, *(e)* colors of (c) rescaled according to A in (d), *(f)* input image, *(g)* result of blending (e) over (f).

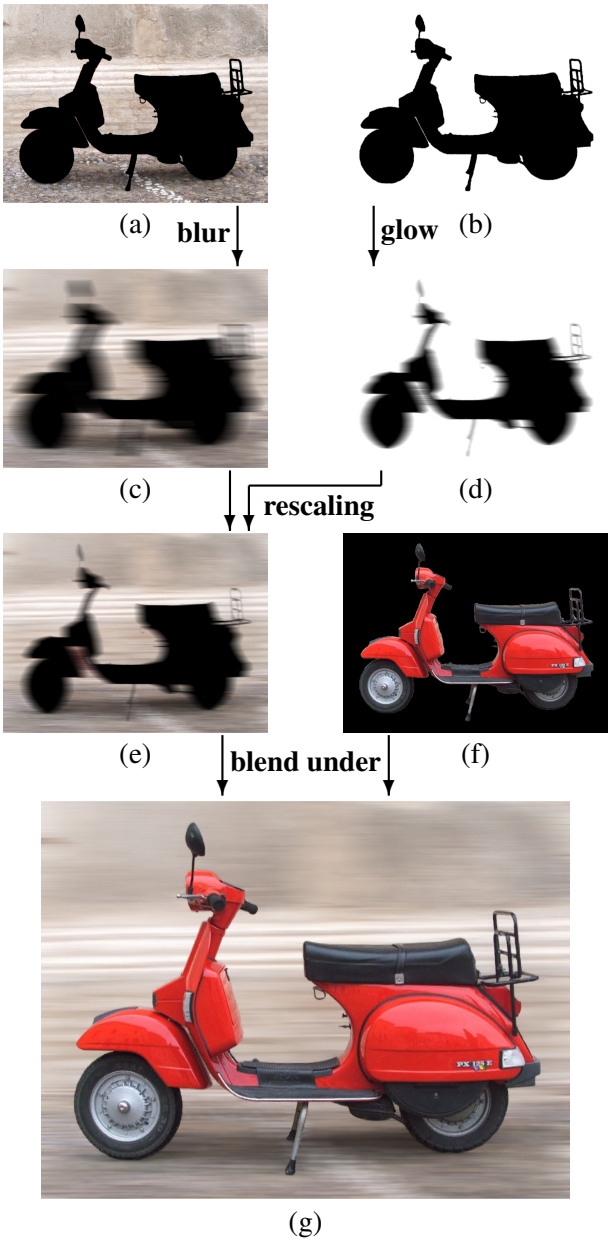

Figure 7: Data flow in the opaque motion blur of backgrounds: *(a)* opacity-weighted colors of the input image, *(b)* opacity of the input image (here the inverted A channel of Figure 6b), *(c)* 1D box filter applied to (a), *(d)* glow filter applied to the opacity of the input image, *(e)* colors of (c) rescaled according to A in (d), *(f)* colors of the input image weighted with the inverted opacity (here the same as Figure 6a), *(g)* result of blending (e) under (f)

of motion with one of the screen axes. Furthermore, a 1D pyramidal blur filter is less efficient than a separable 2D filter since the number of pixels is only halved from one pyramid level to the next. On the other hand, a straightforward implementation of a 1D filter of fixed size in a fragment shader with the help of a for-loop avoids the costs of any render target switches and does not require additional image memory other than for the

input image and for the result. In fact, it is straightforward to implement the proposed algorithm in one pass of a fragment shader over all pixels of the resulting image. The performance of this fragment shader is mainly determined by the number of texture lookups, which is directly determined by the size of the motion blur filter. (As discussed in the next section, the glow filter does not require additional texture lookups if RGBA lookups are employed.) Thus, for motion blur of limited filter size, the usage of a pyramid blur filter is not justified. However, pyramid blur filters or Fast Fourier Transform (FFT) filters would provide better performance for strong motion blur effects with large filter sizes.

Secondly, the proposed method for image-based motion blur can be generalized to more than one moving object by blurring the image of each masked object separately and blending all images together — provided than an unambiguous depth order is available.

Thirdly, the described process only applies to moving objects in the foreground. In the case of motion blur of the background (for example, caused by camera panning) and also for moving objects that are occluded by static objects in the foreground, some changes are required. Figure 7 illustrates the process to add motion blur to the background of the image in Figure 5. Here, the only change is to blend the result "under" (instead of "over") the appropriately masked (i.e. opacity-weighted) foreground [PD84]. Note in particular that the result in Figure 7g does not show any of the black band artifacts around the sharp foreground, which can be observed if colors are not appropriately rescaled as discussed by Barsky et al. [BTCH05]. In fact, in this particular case, the effect of the opaque image blur is limited to rescaling the colors of the blurred RGBA image to full opacities.

4.2 Glow Filter

In order to avoid additional render passes, we designed a particularly efficient glow filter for the implementation of the opaque motion blur filter described in the previous section. As stated in Section 3.2, the most important restriction is that the glow filter does not decrease the A channel of any pixel. To this end, we apply a triangle filter to each pixel, which is of the same size as the box filter used to blur the opacity-weighted colors. However, instead of adding the results of all filtered pixels (which would result in oversaturated opacities), the maximum filtered opacity is used as the output of the glow filter. In terms of an equation:

$$\mathrm{A}_{i,\mathrm{glow}} = \max_{k=i-w,\ldots,i+w} \left\{ \left(1 - \frac{|k-i|}{w}\right) \mathrm{A}_k \right\} \quad (10)$$

where $\mathrm{A}_{i,\mathrm{glow}}$ is the resulting opacity at sample position i and A_k denotes the input opacity at position k. w denotes the width of the triangle filter such that its size is $2w + 1$.

With the help of the built-in "max" function of GLSL, this glow filter can be efficiently computed in a fragment shader within the same for-loop as the mentioned box filter of the opacity-weighted colors.

4.3 Results

The images in Figures 6g and 7g were computed with an implementation of the proposed algorithm in a GLSL shader within the game engine Unity 3.4, which allowed for rapid prototyping. The render time for images of 1024×768 pixels and a filter that is 101 pixels wide (i.e. 101 RGBA texture lookups per fragment) was 14 ms on a Windows XP PC with an NVIDIA Quadro FX 770M GPU.

It should be noted that we made no attempt to accurately simulate actual motion blur; thus, the image quality is certainly worse than previously published techniques to simulate motion blur, e.g. by Cook et al. [CPC84].

5 Conclusion

We demonstrated a new technique — the "opaque image blur" — to handle disocclusions in depth-of-field rendering and image-based motion blur. While the proposed depth-of-field rendering algorithm offers a unique trade-off between performance and image quality, the image-based opaque motion blur is a real-time rendering technique that can be implemented in a single rendering pass and provides high performance for small filter sizes at the cost of physical accuracy.

6 Future Work

Future work includes research on alternative glow filters for the opaque image blur described in Sections 3.2 and 4.2. Of particular interest are glow filters that result in physically correct visual effects.

As mentioned in Section 4.1, the application to motion blur of multiple moving objects requires an unambiguous depth order of these objects. Additional

research is necessary for the case that such an order is not available since this is likely to result in unacceptable rendering artifacts.

Further potential applications of the opaque image blur include the application of generalized depth-of-field effects [KB07] to arbitrary parts of bitmap images. In these cases, some parts of an image have to be blurred without any information about the disoccluded pixels, which is very similar to the image-based motion blur discussed in Section 4. In both cases, the opaque image blur avoids the disocclusion of pixels, and therefore offers a very efficient alternative to more costly disocclusion techniques.

References

[Bar04] Brian A. Barsky, *Vision-Realistic Rendering: Simulation of the Scanned Foveal Image from Wavefront Data of Human Subjects*, APGV '04: Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization, 2004, pp. 73–81, ISBN 1-58113-914-4.

[BE01] Gabriel J. Brostow and Irfan Essa, *Image-based motion blur for stop motion animation*, Proceedings of the 28th annual conference on computer graphics and interactive techniques, SIGGRAPH '01, ACM, 2001, pp. 561–566, ISBN 1-58113-374-X.

[BTCH05] Brian A. Barsky, Michael J. Tobias, Derrick P. Chu, and Daniel R. Horn, *Elimination of Artifacts Due to Occlusion and Discretization Problems in Image Space Blurring Techniques*, Graphical Models **67** (2005), no. 6, 584–599, ISSN 1524-0703.

[CCC87] Robert L. Cook, Loren Carpenter, and Edwin Catmull, *The Reyes Image Rendering Architecture*, SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, 1987, pp. 95–102, ISBN 0-89791-227-6.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter, *Distributed Ray Tracing*, SIGGRAPH '84: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, 1984, pp. 137–145, ISBN 0-89791-138-5.

[Dan07] Danzeb, *Piaggio vespa px 125 e arcobaleno del 1984 fotografata al raduno moto storiche di medole nel 2007*, Wikimedia Commons under Creative Commons Attribution-Share Alike 3.0 Unported license: http://commons.wikimedia.org/wiki/File:PiaggioVespaPX125E_Arc1984R01-.JPG, 2007.

[Dem04] Joe Demers, *Depth of Field: A Survey of Techniques*, pp. 375–390, Addison Wesley, 2004.

[HA90] Paul Haeberli and Kurt Akeley, *The accumulation buffer: hardware support for high-quality rendering*, SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques, ACM, 1990, pp. 309–318, ISBN 0-89791-344-2.

[Ham07] Earl Hammon, *Practical Post-Process Depth of Field*, GPU Gems 3 (Hubert Nguyen, ed.), Addison Wesley, 2007, pp. 583–606, ISBN 978-0-321-51526-1.

[JO04] Greg James and John O'Rorke, *Real-Time Glow*, GPU Gems (Randima Fernando, ed.), Addison Wesley, 2004, pp. 343–362, ISBN 0-321-22832-4.

[KB07] Todd J. Kosloff and Brian A. Barsky, *An algorithm for rendering generalized depth of field effects based on simulated heat diffusion*, ICCSA'07: Proceedings of the 2007 international conference on Computational science and its applications, Lecture Notes in Computer Science, vol. 4707, 2007, pp. 1124–1140, ISBN 978-3-540-74482-5.

[KMH95] Craig Kolb, Don Mitchell, and Pat Hanrahan, *A realistic camera model for computer graphics*, SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, ACM, 1995, pp. 317–324, ISBN 0-89791-701-4.

[Kra11] Martin Kraus, *Using Opaque Image Blur for Real-Time Depth-of-Field Rendering*, Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP 2011), 2011, pp. 153–159, ISBN 978-989-8425-45-4.

[KS07a] Martin Kraus and Magnus Strengert, *Depth-of-Field Rendering by Pyramidal Image Processing*, Computer Graphics forum (Proceedings Eurographics 2007) **26** (2007), no. 3, 645–654, ISSN 1467-8659.

[KS07b] Martin Kraus and Magnus Strengert, *Pyramid Filters Based on Bilinear Interpolation*, Proceedings GRAPP 2007 (Volume GM/R), 2007, pp. 21–28, ISBN 978-972-8865-71-9.

[KTB09] Todd J. Kosloff, Michael W. Tao, and Brian A. Barsky, *Depth of field postprocessing for layered scenes using constant-time rectangle spreading*, Proceedings of Graphics Interface 2009, GI '09, 2009, pp. 39–46, ISBN 978-1-56881-470-4.

[KŽB03] Jaroslav Křivánek, Jiří Žára, and Kadi Bouatouch, *Fast Depth of Field Rendering with Surface Splatting*, Proceedings of Computer Graphics International 2003, 2003, pp. 196–201.

[LES09] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel, *Depth-of-Field Rendering with Multiview Synthesis*, ACM Transactions on Graphics (Proc. ACM SIGGRAPH ASIA) **28** (2009), no. 5, 1–6, ISSN 0730-0301.

[LES10] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel, *Real-Time Lens Blur Effects and Focus Control*, ACM Transactions on Graphics (Proc. ACM SIGGRAPH'10) **29** (2010), no. 4, 65:1–7, ISSN 0730-0301.

[LKC08] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi, *Real-Time Depth-of-Field Rendering Using Splatting on Per-Pixel Layers*, Computer Graphics Forum (Proc. Pacific Graphics'08) **27** (2008), no. 7, 1955–1962, ISSN 1467-8659.

[LKC09] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi, *Real-Time Depth-of-Field Rendering Using Anisotropically Filtered Mipmap Interpolation*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), no. 3, 453–464, ISSN 1077-2626.

[PC82] Michael Potmesil and Indranil Chakravarty, *Synthetic Image Generation with a Lens and Aperture Camera Model*, ACM Trans. Graph. **1** (1982), no. 2, 85–108, ISSN 0730-0301.

[PD84] Thomas Porter and Tom Duff, *Compositing digital images*, SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques, ACM, 1984, pp. 253–259, ISBN 0-89791-138-5.

[PH04] Matt Pharr and Greg Humphreys, *Physically Based Rendering: From Theory to Implementation*, Morgan Kaufmann Publishers Inc., 2004, ISBN 9780123750792.

[Rok93] Przemyslaw Rokita, *Fast Generation of Depth of Field Effects in Computer Graphics*, Computers & Graphics **17** (1993), no. 5, 593–595, ISSN 0097-8493.

[Ros08] Gilberto Rosado, *Motion Blur as a Post-Processing Effect*, pp. 575–581, Addison-Wesley, 2008.

[Sco92] Cary Scofield, *2 1/2-D depth-of-field simulation for computer animation*, Graphics Gems III, Academic Press Professional, 1992, pp. 36–38, ISBN 0-12-409671-9.

[Shi94] Mikio Shinya, *Post-filtering for Depth of Field Simulation with Ray Distribution Buffer*, Proceedings of Graphics Interface '94, 1994, pp. 59–66, ISBN 0-9695338-3-7.

[SKE06] Magnus Strengert, Martin Kraus, and Thomas Ertl, *Pyramid Methods in GPU-Based Image Processing*, Proceedings Vision, Modeling, and Visualization 2006, 2006, pp. 169–176, ISBN 978-1-58603-688-1.

[Ste86] Stanley R. Sternberg, *Grayscale Morphology*, Computer Vision, Graphics, and Image Processing **35** (1986), no. 3, 333–355, ISSN 0734-189X.

Citation

Martin Kraus, *Using Opaque Image Blur for Real-Time Depth-of-Field Rendering and Image-Based Motion Blur Detection: HDR for Dynamic Scenes*, Journal of Virtual Reality and Broadcasting, 10(2013), no. 5, December 2013, urn:nbn:de:0009-6-38199, ISSN 1860-2037.