Variable-Sized, Circular Bokeh Depth of Field Effects

Johannes Moersch and Howard J. Hamilton
Department of Computer Science, University of Regina, Regina, SK, Canada

Abstract

We propose the Flexible Linear-time Area Gather (FLAG) blur algorithm with a variable-sized, circular bokeh for producing depth of field effects on rasterized images. The algorithm is separable (and thus linear). Bokehs can be of any convex shape including circles. The goal is to create a high quality bokeh blur effect by post processing images rendered in real-time by a 3D graphics system. Given only depth and colour information as input, the method performs three passes. The circle of confusion pass calculates the radius of the blur at each pixel and packs the input buffer for the next pass. The horizontal pass samples pixels across each row and outputs a 3D texture packed with blur information. The vertical pass performs a vertical gather on this 3D texture to produce the final blurred image. The time complexity of the algorithm is linear with respect to the maximum radius of the circle of confusion, which compares favorably with the naive algorithm, which is quadratic. The space complexity is linear with respect to the maximum radius of the circle of confusion. The results of our experiments show that the algorithm generates high quality blurred images with variable-sized circular bokehs. The implemented version of the proposed algorithm is consistently faster in practice than the implemented naive algorithm. Although some previous algorithms have provided linear performance scaling and variable sized bokehs, the proposed algorithm provides these while also permitting more flexibility in the allowed blur shapes, including any convex shape.

Keywords: Depth of field, bokeh, blur, separable algorithm.

1 Introduction

Despite advances in hardware, today's real-time graphics are still far from photorealistic because geometric data is being drawn in an inaccurate and unconvincing way. Until unbiased path tracing becomes feasible, we will continue to employ shortcuts to maintain real-time speed while increasing realism. One such shortcut is an approximation of the *depth of field* effect, which is caused by the use of a lens with a non-zero aperture. This effect causes blurring of objects that are not on the focal plane. Since an aperture of zero is impossible, even our eyes generate this effect. The larger the aperture of the lens, the greater the effect. Film making and photography use lenses with large apertures, so in these media, the effect can be pronounced.

As shown in Figure 1, light originating from a point lying on the focal plan (solid lines in Figure 1) focuses on a single point in the image plane. In contrast, light originating from a point that does not lie on the focal plane (dashed lines in Figure 1) will not focus on the image plane. Instead, it will focus either in front of or behind the image plane. This results in the light spreading over an area on the image plane, which is called the *circle of confusion*. The radius of this circle (called the *blur radius*) is directly related to the lens size; the larger the aperture, the greater the blur radius.

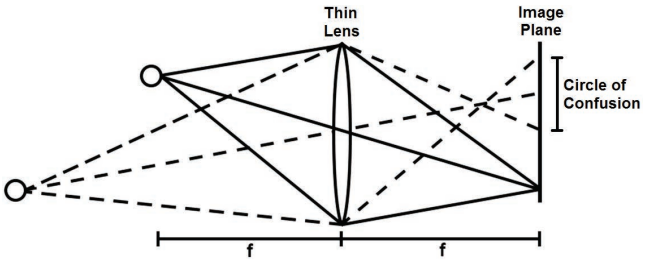

Figure 1: In-focus light (solid lines) and out-of-focus light (dashed).

The depth of field effect has a dramatic impact on the appearance of an image, as can be seen in Figure 2 [18]. This effect is indispensable in non-interactive visual media and its absence has been a barrier to creating compelling cinematic experiences in real-time interactive media. In recent years, a variety of techniques to emulate this effect in screen space have been proposed [5, 6, 8, 10], but most have been lacking either in visual quality or in performance. To improve performance, a standard approach exploits separability. A filtering algorithm is considered *separable* if it breaks the quadratic two dimensional processing into two distinct one dimensional passes, each with linear complexity. Although a rectangular box filter and a circular Gaussian filter are separable, a circular box filter is not.

Figure 2: An image taken with an aperture of f/32 (left) and f/5.6 (right) [18].

Here, we propose the Flexible Linear-time Area Gather (FLAG) blur algorithm with a variable-sized, circular bokeh for producing depth of field effects on rasterized images. The algorithm is separable (and thus linear). Bokehs can be of any convex shape including circles. In Section 2, we review previous work on creating depth of field and bokeh effects. In Section 3, we describe a quadratic-time naive algorithm and our linear-time approach. In Section 4, we show experimental results for these two techniques. Finally, in Section 5, we present conclusions.

2 Previous Work

The *blur shape* of a depth of field effect describes the shape and intensity distribution of the region over which each pixel is blurred. As discussed below, many proposed techniques and their early applications in computer games used a Gaussian blur effect, whereby the blur is most pronounced at a central point and then gradually reduces in every direction outward. In reality, the

* {moerschj, hamiltoh}@uregina.ca

Graphics Interface Conference 2014
7-9 May, Montreal, Quebec, Canada

optical effect causes light to spread at a constant intensity. More physically accurate depth of field effects are commonly referred to as *bokeh* depth of field effects. The use of the term "bokeh" implies the blur shape being used is physically plausible. In particular, circular blur shapes, like those in the right side of Figure 3, are physically plausible because they are produced by any optical system with a circular aperture.

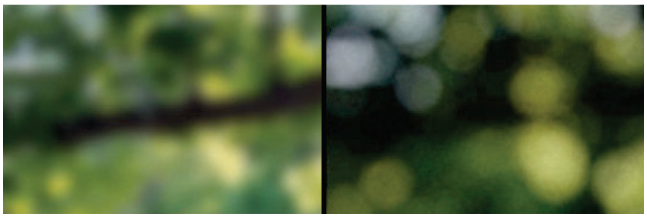

Figure 3: The same image blurred with Gaussian blur (left) and a circular bokeh filter (right) [17].

Demers surveys techniques developed up to 2004 for creating depth of field effects [2]. He states that Z-Buffer Depth-of-Field algorithms are best for real-time use. Wu et al. provide an off-line method for rendering realistic spectral bokeh effects due to chromatic aberration using circles and other shapes [19].

Mulder and van Liere increase the speed of processing by simulating depth of field effects accurately in the centre of attention and less accurately in peripheral areas [12]. The intensity distributions over the circles of confusion at the centre of attention are uniform, but those elsewhere are Gaussian.

Kass et al. give a good description of the problem of real-time computation of blur effects [7]. They devised a linear-time GPU-based algorithm for computing Gaussian blur with tridiagonal matrices [7]. Zhou et al. provide an efficient two-pass algorithm for the task assuming that blurring should fall off with the inverse square of distance [21]. Kraus and Stengert use a GPU-based pyramidal interpolation technique to perform Gaussian blurring [9]. They devised a novel technique that adds the effect of pixels that are invisible in the original image but contribute to the final image because they are visible from part of the simulated lens.

Lee et al. devised a method to produce layered image-based scenes with accurate computation of depth of field blur. Instead of using a single source image, they use multiview sampling, so their technique is not directly comparable to the proposed technique.

Real-time Gaussian depth of field effects first appeared in commercial products in 2007 and can be found in games such as *Crysis* [3] and *Call of Duty 4: Modern Warfare* [6] (see Figure 4).

Figure 4: Example of depth of field in Activision's *Call of Duty 4: Modern Warfare* [6].

Although numerous approaches to creating a depth of field effect have been devised, only two have been shown to be achievable in real time on common consumer-level graphics hardware: scatter-based blur and gather-based blur.

Scatter-based blur: *Scatter-based* blur methods spread a fraction of a pixel's intensity to every pixel within its circle of confusion [13]. This can be done by drawing overlapping quads with additive blending. This approach is straight forward, but the complexity is quadratic. Epic Games (in their *Samaritan Demo*) [4] and *Capcom* (in *Lost Planet* on PCs) [15] used this approach. The massive overdraw caused by rendering thousands of overlapping polygons makes the fill cost high with this technique.

Gather-based blur: *Gather-based* blur methods produce an output value for each pixel by accumulating samples from a set of nearby pixels [6, 14, 19]. Such methods take advantage of current graphics architectures, which have high memory bandwidth and a fast texture cache optimized for clustered texture sample operations. However, the magnitude of texture sample operations required by naive gather-based methods is too large for current hardware, so they perform poorly.

In 2009, Kawase proposed a *separable*, gather-based blur algorithm to solve the problem of obtaining 2D bokeh blur effects in linear time [8]. A key insight behind the algorithm is to decompose a polygonal blur shape into a set of rhombi; this idea has also been used by McIntosh et al. [11] and mentioned by Zanuttini [20]. This decomposition allows each rhombus of the blur shape to be calculated as a separable filter (much like a typical separable box filter). For example, a hexagon can be decomposed into three rhombi. The algorithm provides an efficient means of creating a large bokeh blur, but it is limited to polygonal blur shapes. Variable-sized bokeh shapes can be accomplished through depth checks during the gather. As mentioned, the technique is restricted to polygonal blur shapes that can be decomposed into rhombi, and each additional rhombus (without a shared edge) decreases performance. This technique is used in Electronic Arts' *Frostbite 2* game engine [1].

Crytek has since developed an alternative high-performance gather-based technique [16]. Instead of decomposing the gather into multiple blur passes on different axes, they decompose it into a coarse-grained 2D pass to create the bokeh shape and a fine-grained 2D pass to fill in the gaps. Additionally, they employ a technique named *n-gon mapping*, which maps all samples into the maximum sized blur shape, to optimize the sample distribution. This approach allows the gather area to be easily changed based on depth. They also employ a low resolution maximum circle of confusion radius buffer to determine the sample area required. The approach performs relatively well and produces results of reasonably high quality. However, the small number of pixels considered in the initial gather can cause fine details to be missed, and this technique suffers from quadratic growth of complexity as either blur radius or sampling resolution are increased. This technique is used in the Xbox One game *Ryse: Son of Rome*.

3 Approach

The proposed approach can best be understood in contrast to a naive algorithm. We describe the naive algorithm in Section 3.1 and the proposed FLAG algorithm in Section 3.2.

3.1 Naive Approach

The naive approach to producing a high quality bokeh blur with a variable radius for the circle of confusion is to use a gather-based blur algorithm. For any pixel (called the *working pixel*), a 2D group of nearby pixels is defined as the *gather pattern* (blur shape). For example, a square of (say) 7x7 pixels, centered on the working pixel, may be used as the gather pattern. For each working pixel, the algorithm performs a 2D gather of the pixels in the pattern, and accumulates the colour value of each pixel whose

circle of confusion includes the working pixel divided by the area of its circle of confusion. Instead of performing the gather at every pixel, it is performed at a spacing of every *k* pixels on both the horizontal and vertical axes. By multiplying the spacing by the dimensions of the gather pattern, the *gather area* can be calculated. The largest circle that can fit inside the gather area is the largest circle of confusion that can be handled by the algorithm. Therefore, half of one dimension of the gather area in pixels equals to the maximum radius of the circle of confusion.

As an example, in *Ryse: Son of Rome*, Crytek applies a spatially optimized variant of the naive approach with a 7x7 gather pattern with a spacing of perhaps 3 to 8 pixels between the samples, and then supplements the naive algorithm with an additional blur pass, which covers the artifacts.

If the naive algorithm were performed with a large gather pattern that has low spacing between the samples, the result would be of high quality. However, due to the algorithm's quadratic time complexity, the performance cost quickly becomes high. For example, a 7x7 gather pattern only requires each pixel to sample 49 neighbouring pixels, but if the size of the gather pattern is doubled or tripled on each axis, the number of samples per pixel rises to 196 or 441, respectively.

3.2 The FLAG Algorithm

The FLAG blur algorithm produces variable-sized, circular blur shapes in linear time with respect to the radius of the blur shape. Due to its linear performance scaling, it can achieve high performance while performing fine sampling. The technique depends on random access buffers, which expose the ability to perform scattered writes. (In DirectX 11, this functionality is provided by Unordered Access Views.) Like Kawase's technique, the algorithm works much like a conventional separable box filter. Many of the same advantages and limitations that apply to the polygonal decomposition technique also apply to this technique [8]. The key difference is in the horizontal pass, which writes out a 3D texture (instead of a 2D one). The vertical pass then reads from this 3D texture. The use of the 3D texture allows for bokeh shapes that cannot be decomposed into rhombi, such as circles. In total, the algorithm consists of three passes: the circle of confusion pass, the horizontal pass, and the vertical pass. Each of these passes is now described and then the complexity of the algorithm is analyzed.

3.2.1 Circle of Confusion Pass

In its first pass, the algorithm calculates the radius of the circle of confusion for each pixel in the source image from its depth value. This calculation requires several pieces of information about the optical system, namely the diameter of the aperture (*A*), the focal length (*F*), the focal plane distance (*P*), and the pixel depth (*D*). The radius of the circle of confusion (denoted *r*) is calculated using Equation 1 [16]:

$$r = \left| A\left(\frac{F(P-D)}{D(P-F)}\right)\right| \quad (1)$$

The output of the first pass is stored in a 2D texture. Since the next pass only needs the *r* value and the RGB color value for each pixel, they should be stored together for efficiency of access. (In our implementation, we pack them together in a 64 bit RGBA value with the color in the RGB fields and the circle of confusion in the A field.)The resulting 2D texture is the only input required by the next pass.

3.2.2 Horizontal Pass

The only output of the horizontal pass is a 3D texture bound as a random access buffer for scattered writes. The output 3D texture contains all the information required by the vertical pass. The set of Z values at each {X, Y} location in the 3D texture contains the cumulative contribution of all pixels in row Y to any pixel in column X. The value at {X, Y, 0} is the contribution of row Y to the working pixel at {X, Y}, the value at {X, Y, 1} is the contribution of row Y to the first pixel directly above and below the working pixel at {X, Y ± 1), the value at {X, Y, 2} is for the pixel 2 places above and below the working pixel, etc.

This information can be calculated efficiently because of two properties of the blur shape employed. First, the blur shape is convex, and second, the entire shape has constant intensity. If the blur shape has vertical symmetry, the memory requirements can be halved. All of these properties are physically plausible. These properties also make the calculation of the 3D texture quite simple. The example in Figure 5 shows a horizontal gather operation on a single pixel, i.e. on the working pixel located in column X, row Y. Here, the size of the gather pattern is 13x13. Three example pixels (shown for visibility as red, green, and blue) are blurring onto the working pixel in the middle. Suppose their radii happen to be as shown by the red, green, and blue circles, respectively. The amount that a circle overlaps column X depends on its radius and the nearness of the pixel to the working pixel. For example, the blue circle affects pixels in column X up to 5 positions away from the working pixel. By accumulating the contributions of the pixels in row Y to column X, the set of values shown on the far left can be calculated. These values are then written into the output 3D texture.

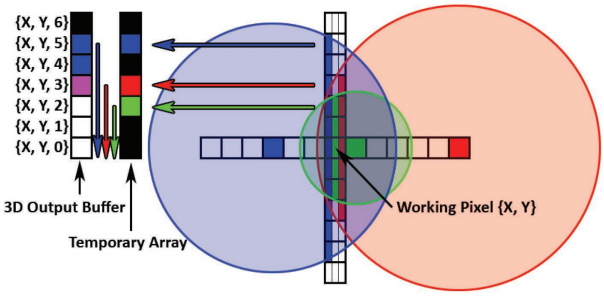

Figure 5: An example of how to calculate the 3D texture.

To maintain linear time complexity, the gather and accumulation are performed separately. While gathering for pixel {X, Y} along row Y, a temporary array is used. The cell in this array that corresponds to the point where the perimeter of the circle of confusion for each relevant pixel in row Y intersects column X is set to that pixel's intensity divided by the area of its circle of confusion. In Figure 5, the perimeter of the blue circle intersects column X in pixel {X, Y – 5}, which is 5 away from the working pixel, so position 5 in the temporary array is set to the colour value from pixel {X, Y – 5} divided by the area of the blue circle. For convenience, the affected pixel in the temporary array is shown as blue in Figure 5. After the gather operation for the working pixel is complete, an accumulation process is performed from the end of the temporary array to its start. At each consecutive pixel of the temporary array, the total accumulated so far is stored in the corresponding position of the 3D output array. In the example, the accumulated total from the last position of the temporary array is stored in position {X, Y, 6} of the output array; the total from the last two positions is stored in position {X, Y, 5}, and so forth. The overall effect is to build a rough cumulative distribution in the set of Z values for {X, Y}.

3.2.3 Vertical Pass

The vertical pass performs gathering similarly to a basic separable box filter, as shown in Figure 6. The only difference is that the gather is against a 3D texture instead of a 2D one. In the example separable box filter, shown in Figure 6(a), the values in the working pixel {X, Y} and three values on either side of it in column X are accumulated to form the sample for pixel {X, Y}. With our method, at each step, the texture is sampled using the Z value that corresponds to the difference in Y between the positions of the working pixel and the sample pixel. In Figure 6(b), the relevant samples are outlined in white. For example, for pixel {X, Y}, we accumulated values from {X, Y, 0}, {X, Y ± 1, 1}, {X, Y ± 2, 2}, etc. The effect is to find the cumulative contribution of all neighbouring pixels onto the working pixel. The final result is the total of each sample added together.

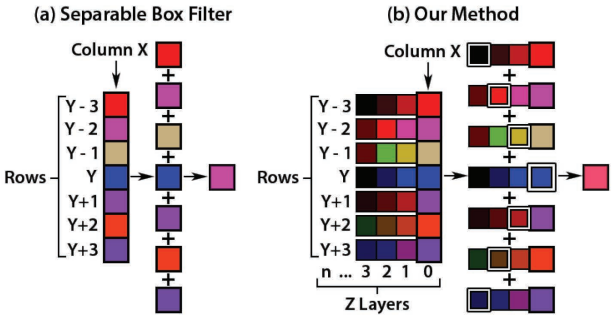

Figure 6: An example showing how the 3D texture is sampled.

Figure 7: Results for hoops image: (a) original, (b) blurred.

3.2.4 Complexity Analysis

The circle of confusion pass has constant time complexity per pixel and both the horizontal and vertical passes have linear time complexity per pixel, so the overall time complexity is linear. Most of the cost is due to the horizontal pass, because it writes out large amounts of data. The memory cost of the 3D output buffer of the horizontal pass is linear in the gather pattern's Y dimension. For example, for a 13x13 gather pattern, the Z dimension in the 3D array is ceiling(13/2). This factor, when combined with high screen resolutions, can amount to hundreds of megabytes. A 1080p, full resolution implementation with a 41x41 equivalent gather area requires 332 MB for the 3D buffer alone. However, the memory costs of the circle of confusion pass and vertical pass are both constant per pixel, and are comparatively negligible.

4 Results

The images of hoops and pillars in Figure 7 and 8, respectively, show the equivalent of a 41x41 gather pattern (1681 samples per pixel) sampling every second pixel ($k = 2$). The gather area spans an 81x81 pixel region, which allows for a maximum circle of confusion radius of 40 pixels.

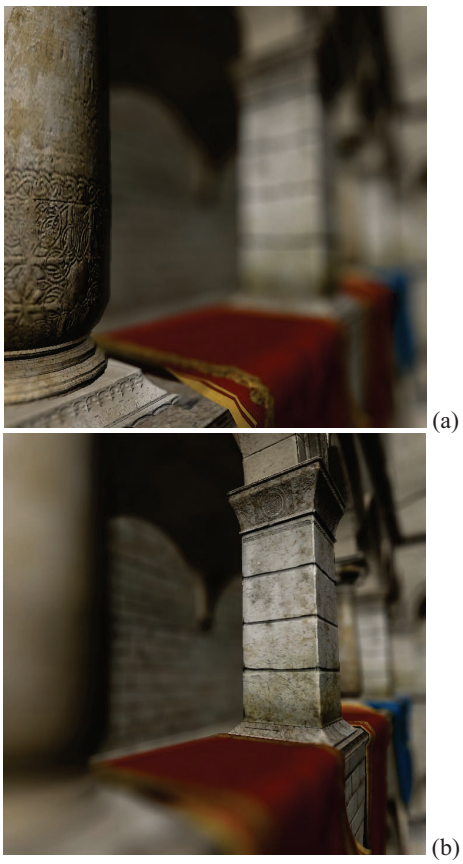

Figure 8: Results for pillars image: (a) near focus, (b) mid focus.

At high screen resolutions with large gather pattern sizes, the naive approach performed inconsistently, so the following tests were run at 800x600 (on an ATI HD5870). We determined the *frame time* as the average (mean) time per frame across 500 frames. Every value reported in Figure 9 is the average of 5 runs with identical parameters.

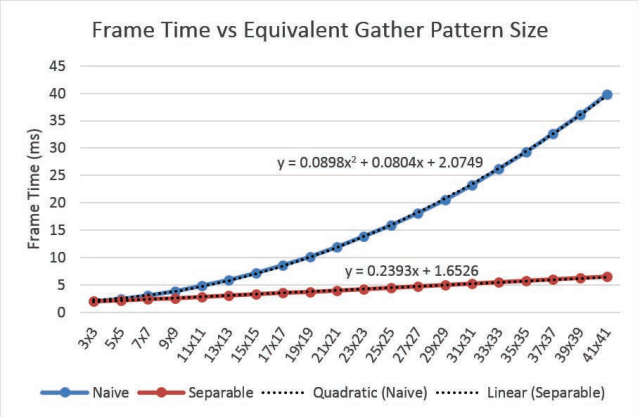

Figure 9: Frame Time versus Filter Sizes.

As the size of the gather pattern (and thus the maximum radius of the circle of confusion) increases, the total time to render each frame increases. The results show that the cost of the FLAG algorithm grows linearly with respect to the gather pattern size, while the naive approach grows at a quadratic rate. For almost all gather pattern sizes, our technique outperforms the naive implementation, and at large sizes the difference is dramatic.

The implemented naive approach samples all pixels in a complete square with a side length that is two times the maximum circle of confusion radius instead of sampling only the relevant pixels in the inscribed circle. An improved implementation could achieve the same result with approximately 20% fewer samples. Nonetheless, even with this improvement, the naive approach would still be significantly slower than the FLAG algorithm.

5 Conclusions and Future Work

The FLAG blur algorithm produces high quality results relative to its performance costs. While similar to Kawase's approach, its use of a 3D buffer permits the FLAG algorithm to process circular blur shapes. Any convex constant intensity blur shape can be rendered. It also supports variable blur radii, which are important for creating convincing images. The theoretical time complexity of the algorithm is linear in the size of the blur radius. Experimental results show that a GPU implementation of FLAG scales up linearly with the blur radius. As expected, FLAG significantly outperforms a quadratic naive implementation. While the output is not artifact free and the memory cost is high, the blur itself is of high quality. Through further improvements to the algorithm and its implementation, it could be feasible for use in providing depth of field for real-time interactive media.

The main limitation of FLAG is its memory cost. The 332 MB buffer size for a 1080p implementation with a 41x41 gather area is too large for most applications. Also, since PC graphics drivers overlap the processing of several frames, multiple buffers of this size will be allocated simultaneously. Future research could reduce the buffer size. Half resolution or quarter resolution 3D buffers could be used, which should reduce the buffer size to 42 MB and 5.2 MB, respectively. Further memory could be saved by decreasing the precision of the 3D buffer.

Performance could potentially be improved by limiting the size of the gather area. By down-sampling the circle of confusion map to a lower resolution using a maximum operation, one could obtain a map that contains the maximum radius of the circle of confusion for each large chunk of pixels. This map could be used to reduce the size of the gather pattern with no impact on quality.

The proposed algorithm can create artifacts that are inherent to gather based blur techniques such as ours. For instance, foreground objects have sharp edges where they meet focused objects, and halos occur where background objects are occluded. Many of the techniques used to solve these problems for other blur technique could also be applied to our algorithm.

References

[1] Colin Barré-Brisebois and John White. "More Performance!", *Advances in Real-time Rendering Course*, ACM SIGGRAPH, 2011.

[2] Joe Demers, "Depth of Field: A Survey of Techniques." In *GPU Gems*. Addison-Wesley, 375–390, 2004.

[3] Crytek, "Crysis," 2007.

[4] Epic Games, "The Technology Behind the DirectX 11 Unreal Engine 'Samaritan' Demo," *Game Developers Conference* (*GDC*), 2011.

[5] Yoshiharu Gotanda, "Star Ocean 4: Flexible Shader Management and Post Processing," *Game Developers Conference* (*GDC*), 2009.

[6] Earl Hammon, "Practical Post-Process Depth of Field" in H. Nguyen (ed), *GPU Gems 3*, Addison-Wesley, 583–606, 2007.

[7] Michael Kass, Aaron Lefohn, John Owens, *Interactive Depth of Field using Simulated Diffusion on a GPU*, Technical Report, UC Davis, 2006.

[8] Masaki Kawase, "Anti-Downsized Buffer Artifacts," *CEDEC 2009*. Silicon Studio Inc, 2009. http://www.daionet.gr.jp/~masa/archives/-CEDEC2009_Anti-DownsizedBufferArtifacts.ppt

[9] M. Kraus and M. Strengert, "Depth of Field Rendering by Pyramidal Image Processing," *Computer Graphics Forum*, 26(3):645-654, 2007.

[10] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel, "Depth of Field Rendering with Multiview Synthesis," *ACM Transactions on Graphics*. 28(5), Article 134, 2009.

[11] L. McIntosh, B.E. Riecke, and S. DiPaola, "Efficiently Simulating the Bokeh of Polygonal Apertures in a Post-Process Depth of Field Shader," *Computer Graphics Forum*, 31(6):1810–1822, 2012.

[12] Jurriaan Mulder and Robert van Liere, "Fast Perception-Based Depth of Field Rendering," *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 129–133, 2000.

[13] Michael Potmesil and Indranil Chakravarty, "A Lens and Aperture Camera Model for Synthetic Image Generation," *ACM SIGGRAPH*, 15, 3, 297–305, 1981.

[14] Przemyslaw Rokita, "Generating Depth-of-Field Effects in Virtual Reality Applications," *IEEE Computer Graphics and its Application* 16, 2, 18–21, 1996.

[15] Stefan Salzl, "CEDEC 2007: Capcom on Lost Planet, Part II," 2007. http://www.beyond3d.com/content/news/499

[16] Tiago Sousa, "CryEngine 3 Graphics Gems," *Advances in Real-Time Rendering Course*, ACM SIGGRAPH, 2013.

[17] Wikipedia, Bokeh. http://en.wikipedia.org/wiki/Bokeh

[18] Wikipedia, Depth of Field. http://en.wikipedia.org/wiki/Depth_of_field

[19] Jiaze Wu, Changwen Zheng, Xiaohui Hu, and Fanjiang Xu, "Rendering Realistic Spectral Bokeh due to Lens Stops and Aberrations," *The Visual Computer*. 29(1):41-52, 2013.

[20] Antoine Zanuttini, *Du photoréalisme au rendu expressif en image 3D temps réel dans le jeu vidéo*, PhD thesis, Université Paris 8.

[21] Tianshu Zhou, Jim X. Chen, and Mark Pullen, "Accurate Depth of Field Simulation in Real Time," *Computer Graphics Forum*, 26(1):15-23, 2007.