EG UK Theory and Practice of Computer Graphics (2010)
John Collomosse, Ian Grimstead (Editors)

Efficient Image Blur in Web-Based Applications

M. Kraus

Department of Architecture, Design, and Media Technology, Aalborg University, Denmark

Abstract

Scripting languages require the use of high-level library functions to implement efficient image processing; thus, real-time image blur in web-based applications is a challenging task unless specific library functions are available for this purpose. We present a pyramid blur algorithm, which can be implemented using a subimage copy function, and evaluate its performance with various web browsers in comparison to an infinite impulse response filter. While this pyramid algorithm was first proposed for GPU-based image processing, its applicability to web-based applications indicates that some GPU techniques are of more general interest than previously assumed.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation: Bitmap and framebuffer operations

1. Introduction

Efficient image blur is of increasing interest for many applications. For example, graphical user interfaces employ it to direct the user's attention to modal dialogs while depth-of-field effects enhance the immersion in games and virtual reality environments. Efficient image blur is also one of the best researched visual effects. In fact, the theory of infinite impulse response (IIR) filters provides an algorithm of optimal time complexity [YvV95].

However, IIR filters are often not the most efficient method for image blurring. In web-based applications, for example, the limited performance of interpreted scripting languages requires the use of high-level library functions for efficient image processing. Another example are GPUs (graphical processing units) because the massive parallelism of their architecture is not well exploited by IIR filters. An interesting consequence is that algorithms specifically designed for GPUs may perform better than IIR filters in web-based applications.

In this work, we adapt a GPU-based pyramid algorithm for efficient image blur to the specific requirements of a popular framework for web-based applications, namely HTML5 [HH10] and JavaScript/ECMAScript [ECM09]. On GPUs, this pyramid algorithm was made possible by the render-to-texture functionality. Similarly, the possibility to copy subimages between HTML5 canvas elements [HHSG09] allows us to implement the algorithm efficiently in JavaScript.

The rest of this paper is organized as follows. Section 2 discusses previous work and introduces the basic pyramid algorithm for image blurring. Section 3 presents three different variants of the analysis phase of the pyramid blur algorithm while Section 4 discusses three variants of the synthesis phase. Section 5 presents results and Section 6 shows some potential applications of image blur in computer graphics. Conclusions are discussed in Section 7.

2. Previous Work

Blur filters have many applications in computer graphics; some examples—such as depth-of-field rendering and soft shadows—are illustrated in Section 6. In some cases, however, it is still very difficult to obtain visually satisfactory results in real time—even with GPU-based algorithms [Dem04]. The problem of efficient blurring is even more severe for web-based applications in JavaScript unless blur filters are offered by the web browser [Mic10].

JavaScript implementations of blur filters show limited performance and are therefore not very common. One publicly available implementation [Nic10] implements a separable two-dimensional Gaussian blur filter by two successive one-dimensional convolutions with a truncated Gaussian kernel. The time complexity of this approach is linear

© The Eurographics Association 2010.

DOI: 10.2312/LocalChapterEvents/TPCG/TPCG10/009-016

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

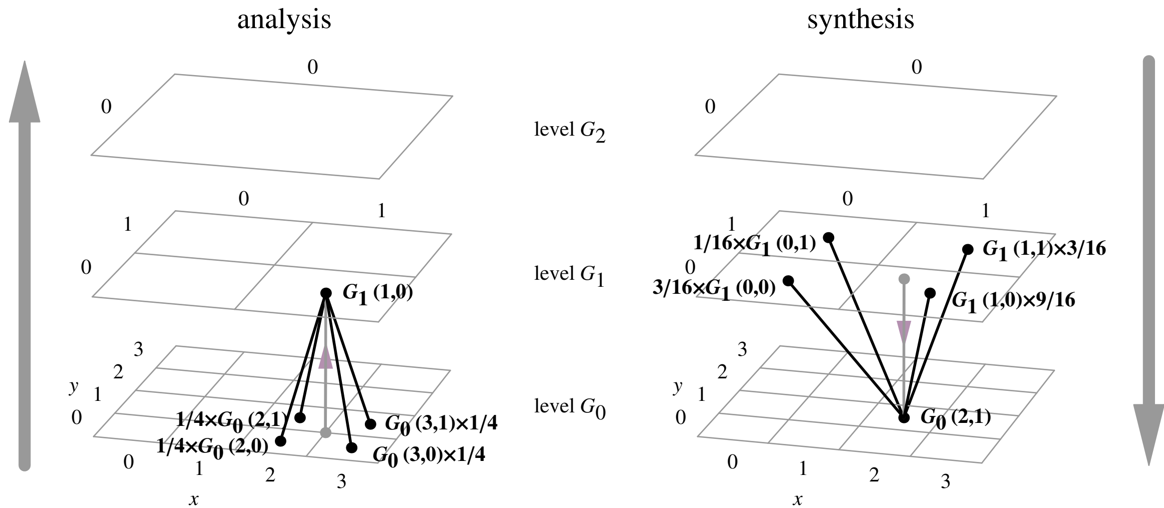


Figure 1: *Basic structure of the pyramid blur methods: bottom-up analysis* (left) *followed by a top-down synthesis* (right) *of image data. The analysis operation for $G_1(1,0)$ averages only four pixels and is therefore often too narrow to avoid artifacts. The synthesis filter applied to compute $G_0(2,1)$ results in a biquadratic B-spline filter. The gray dots represent the coordinates of the corresponding single bilinear texture lookups.*

in the width of the kernel; thus, this approach is not suitable for strong blur effects. A more efficient algorithm published by Young et al. [YvV95, vVYV98] approximates Gaussian blur filters by infinite impulse response (IIR) filters. This algorithm is linear in the number of output pixels; thus, it is of optimal time complexity.

However, IIR filters are less relevant in GPU-based approaches since the required sequential processing does not match well to the parallel architecture of GPUs [Gre05]. The most common GPU-based approach to blurring is a combination of downsampling and upsampling of images (see for example the work of Hammon [EH07]). In particular, the interpolation of images of coarse mipmap levels as described by Demers [Dem04] is a common technique, which resembles a pyramid algorithm [Bur81]. Pyramid algorithms are of optimal linear time complexity in the number of processed pixels and are well suited for GPUs that support rendering to textures as shown by Strengert et al. [SKE06].

Figure 1 (reproduced from [SKE06]) illustrates a basic GPU-based pyramid blur algorithm with a 2×2 analysis filter (which can be implemented by an appropriate bilinear texture lookup) and a biquadratic B-spline synthesis filter (which can also be implemented by a blinear texture lookup with appropriate texture coordinates). As discussed by Kraus [Kra09], the staircase artifacts of this approach can be significantly reduced by 4×4 analysis filters. The blur strength of the original approach is varied in discrete steps by performing the synthesis and analysis only on a limited number of levels (see Figure 7). An extension to continuous blur strength was presented by Kraus [Kra09].

In this work, the pyramid blur algorithm published by Strengert et al. [SKE06] is adapted to web-based applications using HTML5 [HH10] and JavaScript/ECMAScript [ECM09], in particular the `drawImage` method of the `CanvasRenderingContext2D` [HHSG09]. Since some of the details of its specification are relevant for the implementation of our proposed algorithm, we summarize it here: The `drawImage` method may be invoked as "`drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)`" where the "source rectangle is the rectangle whose corners are the four points `(sx, sy)`, `(sx+sw, sy)`, `(sx+sw, sy+sh)`, `(sx, sy+sh)`." And the "destination rectangle is the rectangle whose corners are the four points `(dx, dy)`, `(dx+dw, dy)`, `(dx+dw, dy+dh)`, `(dx, dy+dh)`." Furthermore, the specification states: " When `drawImage()` is invoked, the region of the image specified by the source rectangle must be painted on the region of the canvas specified by the destination rectangle, after applying the current transformation matrix to the points of the destination rectangle."

HTML5 [HH10] and JavaScript/ECMAScript [ECM09] have been implemented in popular web browsers such as Safari (we refer to version 4.0.5 on Windows XP), Chrome (version 4.1.249.1064 on Windows XP), and Firefox (version 3.6.3 on Windows XP). Furthermore, Internet Explorer 9 will support HTML5 to a wider extent than previous versions [Hac10]. The availability of HTML5 and JavaScript

© The Eurographics Association 2010.

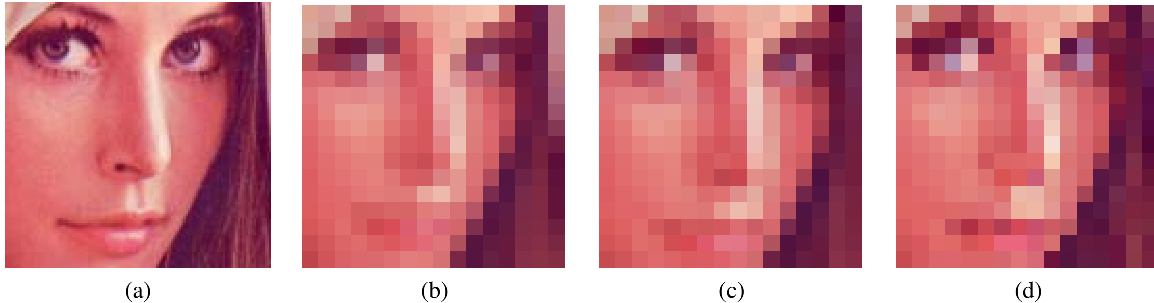


Figure 2: *Minification:* (a) *detail* (128×128 *pixels) of the Lena image and minifications to* 16×16 *pixels using* (b) *Safari,* (c) *Chrome, and* (d) *Firefox.*

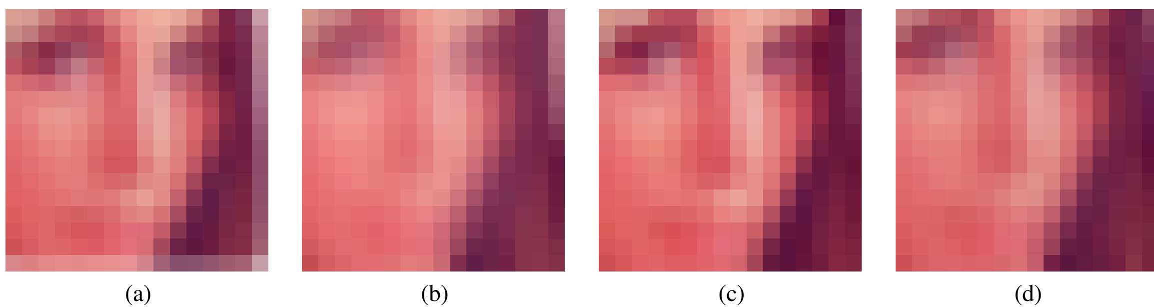


Figure 3: *Analysis:* (a) *basic analysis (see Section 3.2) using Safari, and a wider analysis filter (see Section 3.3) using* (b) *Safari,* (c) *Chrome, and* (d) *Firefox.*

across browsers and operating systems (including operating systems for mobile devices such as iPhone OS and Android) makes it an important programming framework for interactive graphical applications of many kinds. Thus, it is worthwhile to research the applicability of published computer graphics algorithms within this framework—not only in order to find the most suitable algorithm but also to learn about the applicability of algorithms that have been proposed for different APIs, in particular hardware-accelerated OpenGL.

3. Adapted Analysis

The analysis of the pyramid blur algorithm repeatedly downsamples an input image by a factor of 2 in each dimension. The number of downsampling steps determines the strength of the resulting blur. This section discusses two implementations of the analysis phase of the pyramid blur algorithm using the `drawImage` method of HTML5. First, however, we discuss how to replace the analysis by an image minification using `drawImage`.

3.1. Minification instead of Analysis

If the `drawImage` method is used to downsample an image, i.e., if the destination rectangle covers less pixels than the source rectangle (see Section 2), a so-called minification filter is employed to determine the colors (and opacities) of the resulting output [SA03]. The HTML5 standard does not specify this minification filter and in fact different filters are employed by different implementations. For example, Firefox employs subsampling, i.e. each pixel of the output image is determined by just one pixel of the input image (without any actual filtering), see Figure 2d. The minification filter of Chrome is slightly wider but still rather narrow; see Figure 2c for an example. Safari employs a substantially wider minification filter, which is presumably based on a mipmap hierarchy [SA03]; see Figure 2b.

In the case of a sufficiently wide minification filter, a downsampling step using a single call to `drawImage` may replace the whole analysis phase of the pyramid blur algorithm (see Figure 5 for examples). While this approach results in the most efficient blur algorithms, the minification filter of actual implementations of `drawImage` will in most cases be too narrow to avoid staircase artifacts. Therefore, this approach should only be used to blur static images for which the artifacts are less objectionable.

3.2. Basic Analysis

A basic analysis for the pyramid blur algorithm can be efficiently implemented by repeatedly downsampling an image

© The Eurographics Association 2010.

Figure 4: *Maxification:* (a) *The* 16×16 *pixels minification computed by Safari (see also Figure 2b) is* (b) *maxified by Safari to* 128×128 *pixels. Analogously, the minifications in Figures 2c and 2d are maxified by* (c) *Chrome and* (d) *Firefox.*

using `drawImage` where each call to `drawImage` downsamples the image by a factor of 2 in both dimensions.

The smoothest results are obtained with the widest analysis filters. In the case of the `drawImage` method, all pixel sampling positions should therefore be translated by $(0.5, 0.5)$ in order to ensure that the minification filter is centered at the corners between pixels. In the mentioned implementations of HTML5, this can be achieved either by specifying an appropriate translation transformation using "`translate(0.5, 0.5);`" or by offsetting both coordinates `dx` and `dy` of the "destination rectangle" (see Section 2) by 0.5. However, due to the small minification filters employed by Chrome and Firefox, this approach is ineffective for these browsers. On the other hand, the approach works well in Safari, see Figure 3a.

3.3. Wider Analysis Filters

As discussed by Kraus [Kra09], wider box analysis filters are advantageous for the pyramid blur algorithm. In HTML5, a wider anlysis filter can be implemented by additional calls to `drawImage` with translated sampling positions. For example, the translations could be $(1,0)$, $(0,1)$, and $(1,1)$ pixels relative to the original coordinates. Again, these translations can be specified either by the `translate` method or by offsetting the coordinates of the destination rectangle. The resulting colors of the `drawImage` calls have to be scaled appropriately. The most efficient way to achieve this is by specifying the `globalAlpha` member of the `CanvasRenderingContext2D` [HHSG09]. Furthermore, the resulting images have to be added. This can be efficiently implemented by specifying `"lighter"` as the `globalCompositeOperation` of the `CanvasRenderingContext2D`.

An implementation of a wider analysis filter by 4 calls to `drawImage` is rather expensive in terms of performance but it results in the best visual quality on all tested browsers, see Figures 3b, 3c, and 3d. It should be noted that the quality of the analysis filters can only be evaluated in an actual pyramid blur algorithm; see Section 5.

4. Adapted Synthesis

The synthesis phase of the pyramid algorithm repeatedly upsamples the coarsest image level computed by the analysis until the dimensions of the original image are reached. This section discusses two methods for this synthesis using `drawImage`. First, however, we discuss the use of the `drawImage` method to replace the synthesis.

4.1. Maxification instead of Synthesis

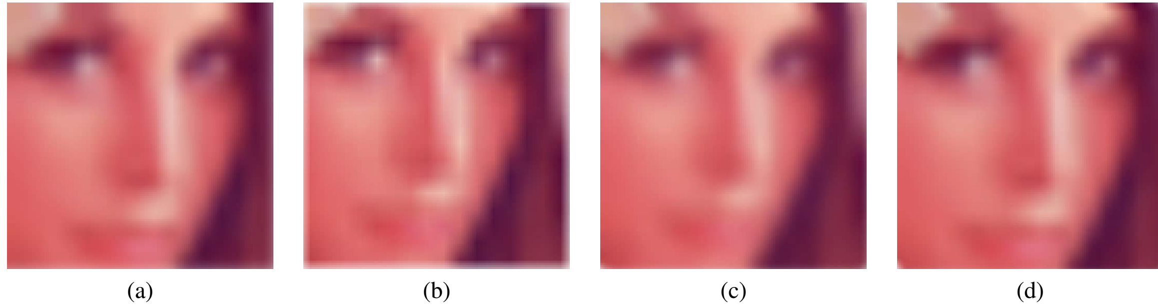
If the `drawImage` method is used to upsample images, a so-called maxification filter (usually an interpolation filter) is used to compute the pixels of the result [SA03]. Similarly to the minification filter, HTML5 does not specify the maxification filter. Therefore, implementations of HTML5 employ various filters; for example, Chrome and Firefox employ bilinear interpolation which results in typical diamond-shaped artifacts (see Figures 4c and 4d). Safari employs a smoother interpolation filter, which unfortunately results in stronger staircase artifacts (see Figure 4b in comparison to a depiction of the coarse image level in Figure 4a). Employing a maxification instead of the synthesis is extremely efficient; however, it is usually not suitable for a blur algorithm due to the strong visual artifacts.

4.2. Biquadratic B-Spline Filtering

Strengert et al. [SKE06] suggest to use a bilinear texture interpolation to achieve a biquadratic B-spline filtering in the synthesis as illustrated in Figure 5a. If the maxification filter is a bilinear interpolation (e.g. with Chrome and Firefox), this approach can be easily implemented with the `drawImage` method by translating the coordinates of the destination rectangle by $(0.25, 0.25)$ (again either using the `translate` method or by offsetting the destination coordinates in the `drawImage` call). As noted by Kraus [Kra09] this biquadratic B-spline filtering can produce very good visual results in a pyramid blur algorithm if a suitable analysis filter is employed.

Unfortunately, this approach is not applicable to Safari

© The Eurographics Association 2010.



(a) (b) (c) (d)

Figure 5: *Synthesis:* (a) *synthesis resulting in a biquadratic B-spline filter using Chrome;* (b) *synthesis for bilinear interpolation using Safari; two-steps-forward-one-step-back method using* (c) *Safari and* (d) *Chrome.*

Table 1: *Time in milliseconds for different blur strengths in a* 128×128 *pixels image using Safari; see Figure 7.*

steps in pyramid	1	2	3	4
time	2.4	3.5	4.6	5.7

due to its different maxification filter. However, it is interesting to note that a translation by $(0.5, 0.5)$ will result in a bilinear interpolation on all tested browsers, including Safari, see Figure 5b.

4.3. Two Steps Forward, One Step Back

While the biquadratic B-spline filtering is the best alternative for the synthesis if the maxification filter of `drawImage` is a bilinear interpolation (e.g. in the case of Chrome and Firefox), higher-order interpolation filters (such as employed by Safari) require a different approach unless they can be used to replace the whole synthesis (which is not the case for Safari due to staircase artifacts).

Our suggestion is to apply an upsampling scheme that resembles a "two steps forward, one step back" approach. Specifically, each upsampling step first upsamples the image by a factor of 4 and then downsamples it by a factor of 2 resulting in a total magnification by a factor of 2. The only exception is the very last synthesis step, which should only employ a single upsampling by factor 2 to avoid excessive performance costs. In order to widen the maxification and minification filters, all destination coordinates of the employed `drawImage` calls are translated by $(0.5, 0.5)$. This approach resulted in the visually best results with Safari (see Figure 5c) and it also works well with implementations that employ a bilinear maxification (see Figure 5d).

5. Results

The described blurring techniques were implemented and tested on a Microsoft Windows XP Professional PC with an Intel Core2 Duo CPU T9600 (2.8 GHz). Table 1 and Figure 7 show results depending on the number of processed pyramid levels. Since the input image of 128×128 pixels is already rather small, the higher pyramid levels are even smaller; thus, the number of pixels per pyramid level is not very relevant for the performance, but each additional level requires about 1.1 milliseconds.

Table 2: *Time in milliseconds for blurring images of different dimensions with Safari (the image of* 128×128 *pixels is depicted in Figure 7c).*

size	512^2	256^2	128^2	64^2	32^2	16^2
pyramid	81	20	4.6	2.0	0.8	0.6
IIR	270	66	17	4.2	1.1	0.4

Table 3: *Time in milliseconds for different blurring methods (pyramid blur with basic analysis, a wider analysis filter, and IIR filter) using different browsers; see Figure 8.*

	pyramid blur			IIR filter		
	Safari	Chrome	Firefox	Safari	Chrome	Firefox
basic	4.6	2.5	5.7	17	37	15
wide	10.2	6.0	14.3	17	37	15

Table 2 presents the performance of a pyramid blur method with Safari (using the basic analysis described in Section 3.2 and the two-steps-forward-one-step-back synthesis described in Section 4.3) and the performance of a JavaScript implementation of an infinite impulse response (IIR) filter to approximate Gaussian blur [YvV95]. While the timings of both methods are about linear in the number of pixels, the timings of the pyramid blur method show a considerably smaller slope. Thus, these timings suggest that the processing of pixels by the `drawImage` method is more efficient than the processing of pixels in JavaScript. Therefore, the proposed pyramid blur method is particularly well suited for large images.

© The Eurographics Association 2010.

Figure 6: *Applications of image blur:* (a) *glossy reflection,* (b) *depth of field,* (c) *glow,* (d) *soft shadow, and* (e) *inpainting.*

Table 3 and Figure 8 present the recommended variants of the discussed pyramid blur method for the three tested browsers using 3 pyramid steps on an image of 128×128 pixels. We have also included results using an IIR filter for comparison. For Safari the synthesis should employ the two-steps-forward-one-step-back approach (see Section 4.3) while Chrome and Firefox should use the synthesis resulting in a biquadratic B-spline filter (see Section 4.2). Depending on the required visual quality, either the basic analysis (see Section 3.2) or the wider analysis filter (see Section 3.3) should be employed. The pyramid blur methods are in general faster than the implementation of the IIR filter; however, the actual speedup factor depends strongly on the particular web browser: it varies between about 1 and 15. Moreover, it also depends on the dimensions of the image as shown by Table 2.

6. Applications of Image Blur

Figure 6 illustrates some of the possible applications of image blur in computer graphics. Further applications include motion blur and various transition effects. All examples are based on a version of the Lena image that is depicted in the upper half of Figure 6a. This image was augmented with a manually specified opacity channel in order to better convey the nature of the effects.

The glossy reflection in the lower half of Figure 6a was produced by a combination of blur, geometric mirroring, and reduced opacity. The illusion of depth of field in Figure 6b was achieved by blending several blurred versions of the image over each other while the glow effect in Figure 6c simply blends the original image over one blurred version. Blending the original image and several blurred versions over more strongly blurred versions results in a basic inpainting technique as suggested by Lefebvre et al. [LHN05] and shown in Figure 6e.

The soft shadow in Figure 6d was generated by rendering a rectangle filled with the shadow color and then drawing a blurred version of the image on top of it with the composite operation `"destination-in"` [HHSG09]. Rendering soft shadows might appear to be an academic exercise since the `CanvasRenderingContext2D` provides the possibility to render soft shadows. However, Chrome does not implement this feature, Firefox limits the supported blur, and the implementation of Safari is inefficient for strong blur. These are three good reasons why a web application might require its own implementation of soft shadows.

7. Conclusions

This paper describes efficient blur methods for web-based applications using HTML5 and JavaScript; in particular, the presented blur methods are in general more efficient than implementations of IIR (infinite impulse resonse) filters if a reduced visual quality is acceptable. Furthermore, an efficient method for biquadratic B-spline upsampling has been presented. Unfortunately, the latter method is only applicable to implementations of the `drawImage` method that employ bilinear interpolation. For implementations using higher-order interpolation, we suggest a “two steps forward, one step back” approach, which provides a similar image quality.

Our experiments showed that the proposed methods are not beneficial for very small images. While we assume that most web images are sufficiently large to benefit from our methods, it should also be noted that the presented approach is not limited to bitmap images but may be applied to any graphics that is dynamically rendered in a canvas element. This is likely to include large and even full-screen graphics, which would benefit considerably from our approach.

The presented methods are based on previously published GPU-based techniques [SKE06, Kra09]; thus, this work also

© The Eurographics Association 2010.

proves that some of these techniques are relevant even without GPUs. This insight should motivate further research on the applicability of GPU-based methods in other contexts, in particular HTML5 with JavaScript.

Additionally, there are at least two conclusions for the evolving HTML5 standard. First, the specification of the interpolation method and the minification filter of the `drawImage` method would allow programmers to write more efficient platform-independent code since these filtering methods are crucial for the proposed blur algorithm and presumably also for other algorithms. Second, many more techniques proposed for GPUs and/or the OpenGL fixed-function pipeline could be exploited in HTML5 if the `CanvasRenderingContext2D` would provide more methods for efficient image processing. In particular, a more flexible specification of the blending operation similar to the specification of the OpenGL blend equation [SA03] could be very useful, for example in order to process color channels independently.

References

[Bur81] BURT P. J.: Fast Filter Transforms for Image Processing. *Computer Graphics and Image Processing 16* (1981), 20–51. 2

[Dem04] DEMERS J.: Depth of Field: A Survey of Techniques. In *GPU Gems* (2004), Fernando R., (Ed.), Addison Wesley, pp. 375–390. 1, 2

[ECM09] ECMA INTERNATIONAL: *Standard ECMA-262: ECMAScript Language Specification, 5th edition*. Tech. rep., ECMA International, December 2009. 1, 2

[EH07] EARL HAMMON J.: Practical Post-Process Depth of Field. In *GPU Gems 3* (2007), Nguyen H., (Ed.), Addison Wesley, pp. 583–606. 2

[Gre05] GREEN S.: Image Processing Tricks in OpenGL. Presentation at GDC 2005, 2005. 2

[Hac10] HACHAMOVITCH D.: IEBlog: HTML5 and Same Markup: Second IE9 Platform Preview Available for Developers. IEBlog (The Windows Internet Explorer Weblog): `http://blogs.msdn.com/ie/archive/2010/05/05/html5-and-same-markup-second-ie9-platform-preview-available-for-developers.aspx`, last accessed: 5/11/2010, 2010. 2

[HH10] HICKSON I., HYATT D.: *HTML5 — A vocabulary and associated APIs for HTML and XHTML*. W3C working draft, World Wide Web Consortium, March 2010. 1, 2

[HHSG09] HICKSON I., HYATT D., SCHEPERS D., GRAFF E.: *Canvas 2D API Specification 1.0*. W3C editor's draft, World Wide Web Consortium, October 2009. 1, 2, 4, 6

[Kra09] KRAUS M.: Quasi-Convolution Pyramidal Blurring. *Journal of Virtual Reality and Broadcasting 6*, 6 (August 2009). `urn:nbn:de:0009-6-18214,`, ISSN 1860-2037. 2, 4, 6

[LHN05] LEFEBVRE S., HORNUS S., NEYRET F.: Octree Textures on the GPU. In *GPU Gems 2* (2005), Pharr M., (Ed.), Addison Wesley, pp. 595–613. 6

[Mic10] MICROSOFT: MSDN Library: Blur Filter. MSDN library: `http://msdn.microsoft.com/en-us/library/ms532979%28VS.85%29.aspx`, last accessed: 5/11/2010, 2010. 1

[Nic10] NICKERSON P.: Pretty Fast Gaussian Blur in Javascript. Author's blog entry: `http://pvnick.blogspot.com/2010/01/im-currently-porting-image-segmentation.html`, last accessed 5/11/2010, 2010. 1

[SA03] SEGAL M., AKELEY K.: *The OpenGL Graphics System: A Specification (Version 1.5)*. Silicon Graphics, Inc., 2003. 3, 4, 7

[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. 2, 4, 6

[vVYV98] VAN VLIET L. J., YOUNG I. T., VERBEEK P. W.: Recursive Gaussian Derivative Filters. In *Proceedings of the 14th International Conference on Pattern Recognition* (1998), vol. 1, pp. 509–514. 2

[YvV95] YOUNG I. T., VAN VLIET L. J.: Recursive Implementation of the Gaussian Filter. *Signal Processing 44* (1995), 139–151. 1, 2, 5, 8

© The Eurographics Association 2010.

(a) (b) (c) (d)

Figure 7: *Blurring an image of* 128×128 *pixels with different strengths by performing* (a) *1,* (b) *2,* (c) *3, or* (d) *4 steps of the analysis and the synthesis (using the basic analysis filter and the two-steps-forward-one-step-back synthesis with Safari).*

(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 8: *Blur techniques with the highest performance:* (a) *basic analysis and two-steps-forward-one-step-back synthesis using Safari; basic analysis and biquadratic B-spline synthesis using* (b) *Chrome and* (c) *Firefox;* (d) *for comparison an approximation to a Gaussian filter using an IIR filter [YvV95]. Blur techniques producing the best image quality:* (e) *wide analysis filter and two-steps-forward-one-step-back synthesis using Safari; wide analysis filter and biquadratic B-spline synthesis using* (f) *Chrome and* (g) *Firefox;* (h) *for comparison a wider IIR filter.*

© The Eurographics Association 2010.