Adding Motion Blur to Still Images

Xuejiao Luo*
Delft University of Technology
Télécom ParisTech

Nestor Z. Salamon†
Delft University of Technology

Elmar Eisemann‡
Delft University of Technology

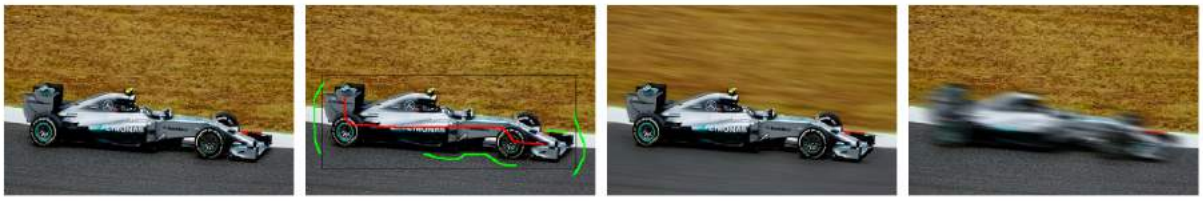

Figure 1: Given a single input image (left), users can segment content by scribble annotations (center-left). Next, a motion-blur effect is created to give the illusion of object motion during capture. (Original image source: pixabay.com)

ABSTRACT

Motion blur appears in images as a visible trail along the motion path of the recorded object. It plays an important role in photography to convey a sense of motion but can be difficult to acquire as intended by the photographer. One solution is to add motion blur in a post process but current solutions involve much manual intervention and can lead to artifacts that mix moving and static objects incorrectly. In this paper, we propose a novel method to add motion blur to a single image that generates the illusion of a photographed motion. Relying on a minimal user input, a filtering process is employed to produce a virtual motion effect. It carefully treats object boundaries to avoid artifacts produced by standard filtering methods. We illustrate the effectiveness of our solution with various complex examples, including multiple objects, reflections and high intensity light sources. Our post-processing solution can achieve a convincing outcome, which makes it an alternative to attempting to capture the intended real-world motion blur.

Index Terms: Image Processing and Computer Vision [I.4.0]: General—Image processing software

1 INTRODUCTION

Motion blur can be used as an artistic effect to convey a sense of motion in still images. In photography, this effect can be acquired by opening the camera shutter for an extended period. During the exposure, moving objects with respect to the camera will result in a different projection location on the camera sensor, which results in a visible streak in the final image [12]. While being an important technique [13], it is very challenging to control or acquire an intended result. Parameters such as the shutter speed, camera motion, illumination, lens and filter configurations strongly influence the result but their effect is difficult or even impossible (e.g., if the object is moving irregularly) to estimate. Further, capturing slowly moving objects, such as stars or clouds, requires a very long exposure to convey even a small sense of motion. Finally, in some cases, a photographer might want to keep a fast moving object in focus, which results in only the background being affected by the motion blur. To achieve this, the photographer needs to precisely follow the object with the camera to keep the position perfectly stable, which is very challenging.

Easier than capturing the result is to produce it in a postprocess. However, with a single image, current image editing software provides blur tools for general purposes. This requires users to manually extract the regions of interest and experiment with different effects. Additionally, many filters result in artifacts at object boundaries, in form of undesired color leakage.

Our method enables a user to add motion blur to a single still image. To this extent, we propose a segmentation tool to select objects of interest to then define the intended motion blur. Multiple objects can be extracted and motion paths freely chosen. Our method relies on an edge-aware filtering that delivers convincing results, while keeping the user interaction simple and avoiding additional scene information.

This paper is organized as follows. In the next section, we revisit previous work. We then describe our approach (Sec. 3), before presenting results of our method (Sec. 4) and concluding (Sec. 5).

2 RELATED WORK

Blur in photography is often used for artistic purpose, to guide the observer, emphasize important elements, or achieve a desired look and composition. The two most common sources of camera blur are motion blur and depth of field.

Depth of field has received much attention and is also a perceptually well explored effect [5]. Several hardware and algorithmic solutions have been proposed. Light-field cameras [8], special sensors [11], coded apertures [2], stereo setups [1], or synthetic reconstruction [20] can be used to enable post-processing of the depth-of-field effect.

Motion blur conveys a sense of motion, however, it is often considered an undesirable artifact, as it can result from camera shake. In consequence, modern cameras often involve stabilization systems [9] to avoid the effect. Our goal is to allow a user to control motion blur for artistic purpose.

For an image sequence, a computational solution to reconstruct motion-blur has been proposed in form of the virtual exposure [21]. Here, short exposure shots are combined to simulate a long-exposure result. Originally conceived to simulate high-dynamic range photography, the work addresses also moving objects. The latter would result in ghosting artifacts due to an exposure gap between the individual shots when accumulating the images. They rely on an optical flow algorithm to fill in the missing transitions to obtain a motion-blurred output. Commercial systems, such as Reel Smart Motion Blur [15] rely on optical flow to estimate the movement between images to then apply a directed blur kernel. A virtual exposure was

*e-mail: X.Luo-3@tudelft.nl, cheryl.xuejiao.luo@gmail.com
†e-mail: N.ZiliottoSalamon@tudelft.nl
‡e-mail: E.Eisemann@tudelft.nl

Graphics Interface Conference 2018
8-11 May, Toronto, Ontario, Canada

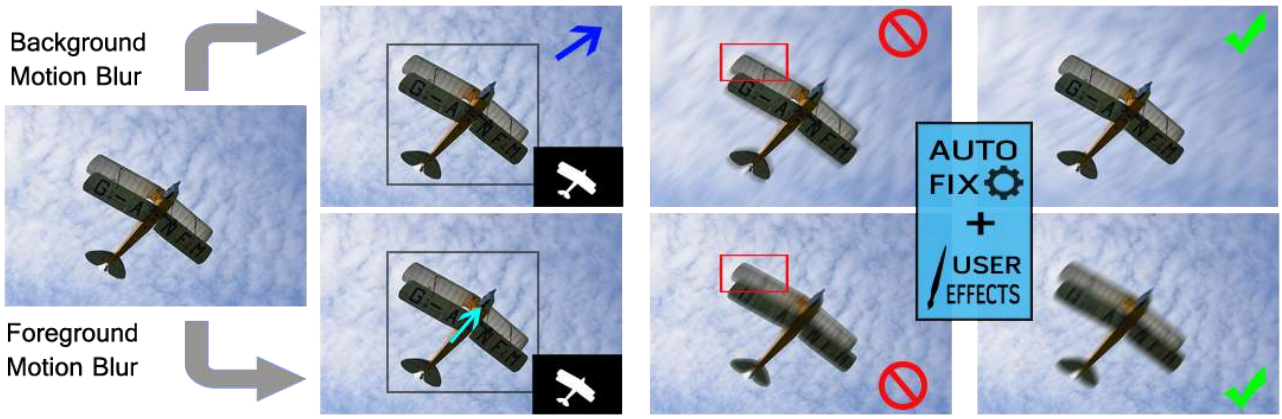

Figure 2: Overview. From left to right: the user can select target regions of an input image (left) using simple annotations (bounding box, scribbles). A motion can be defined for the desired target region, which launches a filtering process to add plausible motion blur. (Input image source: pixabay.com)

also used for light painting, which is able to describe the motion blur caused by a moving light source [19].

Commercial solutions for computational motion blur on a single image also exist. One example is the GIMP [3] motion blur tool. Unfortunately, here, the user has no efficient control over the blur propagation, leading to unrealistic artifacts when mixing foreground and background content. Similarly, Adobe Photoshop [14] can achieve a motion-blur effect but requires significant manual intervention; object segmentation, inpainting and manual organization of layers need to be performed beforehand. Our integrated solution works directly on a single image and facilitates control and definition of motion blur.

Similar in spirit are motion-blur solutions for real-time 3D applications, such as [4, 6, 10, 16]. Typically, they involve deferred shading [18] to derive information such as per-pixel motion, depth, or object id. Our method shares ideas regarding post-processing but relies on a single photograph.

3 Our Approach

Our algorithm adds motion-blur effects to a single image based on a few simple user annotations. Fig. 2 shows an overview of our solution.

The user can select objects via an image-segmentation solution (Sec. 3.1) and can then define the motion of the selected object or area. The algorithm then produces a motion-blurred result while avoiding artifacts around object boundaries (Sec. 3.2). Finally, our solution is extended to the simulation of high dynamic range effects (Sec. 3.3). In the following, we will describe the details of our approach.

3.1 Object Selection

A moving object on the foreground blends with the background, while a moving background does not blend into the static foreground. The differing visibility relationships lead to very different outcomes; a static foreground object will cover the background during the entire exposure and will maintain crisp boundaries, while a moving foreground object will result in a fuzzy boundary. This difference makes it necessary to distinguish the order of the objects present in the image. Therefore, we will first discuss how to select objects in the image.

One of the most user-friendly segmentation methods is GrabCut [17]. The user defines a rectangle containing the potential foreground object. Additional scribbles can be provided to refine the mask segmentation. GrabCut then partitions the image into foreground and background pixels. We then separate the foreground pixels into connected components [7] to define the different objects.

Figure 3: GrabCut foreground extraction. Left: the user provided bounding box and scribbles to guide segmentation. Right: the matte mask for segmented object.

Fig. 3 shows one example of the GrabCut extraction. The user defined an object bounding box and, if necessary, improvement scribbles to assign potential foreground and background regions (left). The extracted mask defines the foreground object (right).

In case that foreground objects overlap, the algorithm can be recursively applied by running the GrabCut on the previously extracted foreground regions. In each step, the output results in a fore- and a background label, which induces a natural ordering of the objects. These can then be processed individually with their own motion path and intensity.

During our experiments, we found that we rarely need to distinguish more than four objects. Hence, we allow the user to determine directly four levels of ordering in the interface by drawing corresponding scribbles. Fig. 4 illustrates a multi-object labeling.

In order to ease explanations, we will drop the foreground and background labels and refer to all extracted regions as objects, which are ordered from back to front.

3.2 Motion Blur

We simulate uniform motion blur by convolving an object with a motion-blur kernel (referred as *psf*) defined by the motion trajectory. E.g., a horizontal translation by n pixels results in a horizontal kernel of size n pixels. The kernel is normalized such that its integral (sum of all pixels) is equal to one. In the example, each kernel pixel will contain a value of $1/n$. Hereby, no energy is created when convolving the input. To compute the kernel for a general linear motion we use the formulation proposed in Matlab. First the bounding box of the provided segment indicating the motion is

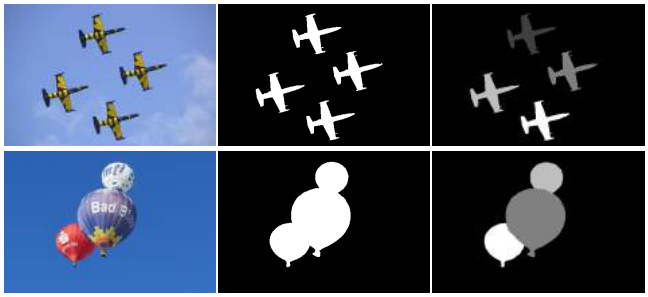

Figure 4: Multiple objects segmentation. Distinct target regions can be segmented, allowing localized and distinct effect control. (Images source: pexels.com (top) and pixabay.com (bottom))

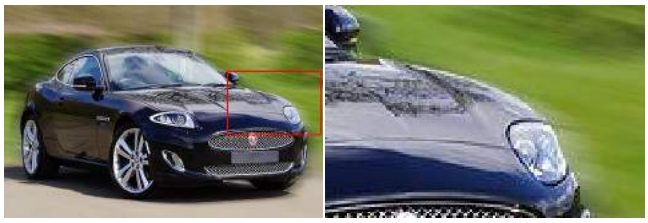

Figure 5: Color leakage when not respecting object boundaries.

determined and extended by two pixels. Then, for each bounding box pixel, we compute one minus the distance of its center to the segment. Next, all values are clamped between zero and one to eliminate negative values. Finally, the resulting kernel is normalized by the total sum of all pixels.

Having derived a blur kernel per object, it seems tempting to visit every pixel of the labeled input image and simply apply the corresponding *psf* kernel. Unfortunately, this results in color bleeding artifacts, as illustrated in Fig. 5.

Similarly, when applying edge-aware filtering, which avoids blurring across object boundaries, the result is unrealistic. Sharp boundaries are maintained for moving foreground objects (Fig. 6), while one would have expected a fuzzy boundary.

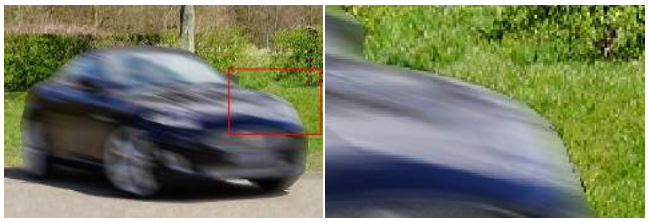

Figure 6: Unrealistic sharp edges from edge-aware filtering.

In order to produce a more plausible result, we will derive blending masks to composite the objects from back to front, one by one. In other words, a given object is motion blurred, and then composited with the current background, which will address the issues of Fig. 6. After explaining the corresponding details, we will show how to address the issues of Fig. 5.

Compositing Fore- and Background

To describe the composition algorithm, we will focus on the steps for a single object. Let B_k be the current background image (B_0 is initialized with zero). For an object O_k, we produce an image I_k, which is a black image into which we copy all pixels labeled with O_k from the input. Further, we produce a mask M_k, which is black except for the pixels that correspond to pixels labeled with O_k. We then convolve both images with the *psf* of O_k. Intuitively, the image $psf * M_k$, where $*$ denotes the convolution, corresponds to a mask that indicates how much the foreground will occlude the background. For example, if the object is not moving, *psf* is by construction a Dirac (a single pixel is equal to one), which implies that the the mask describes exactly the pixels of the original object. Given the convolved results and the background B_k, we compute the new background B_{k+1} as

$$B_{k+1} = psf * (M_k I_k) + (1 - psf * M_k) B_k.$$

In practice, it is possible to avoid the actual derivation of the masks by performing an integration along the kernel directly on the input image. Further, we do not need intermediate background images and it is enough to incrementally composite the motion-blurred objects in a single resulting image.

While conceptually simple, the above approach is still imperfect. It relies on the assumption that each object is entirely visible in the original input image. Unfortunately, this is rarely the case. As soon as an object is moving, visibility relationships change and parts previously-occluded by the object will be revealed. A challenge is to estimate the content that is disoccluded. Assuming that the disoccluded regions look like the original image results in the artifacts shown in Fig. 5. Similarly, assuming the disoccluded pixels are simply black results in a dark halo.

Handling Disocclusions

To correct for the disocclusion artifacts, we propose to use an inpainting procedure. While more advanced solutions could be employed (i.e. [22,23]), we found that a simpler strategy proved sufficient. The reason is that the part will either be in motion itself or overlapped by a moving element, which naturally hides many of the details in the inpainted area.

Adding inpainting to our solution, the main algorithm remains the same; objects are treated front to back, but before filtering with their *psf*, an inpainting procedure is applied. For an object O_k, we will examine its boundary to find pixels adjacent to an object O_j that is nearer (i.e., $j > k$, as objects are ordered). If there is none, O_k does not require any inpainting. If there is an overlap, we want to extend O_k beneath the potentially uncovered region of O_j. Inspired by recent real-time methods [6], we mirror the content of O_k into the area that is covered by O_j. Specifically, we only mirror along the horizontal, vertical and diagonal axes. We choose the mirror direction d closest to the motion direction of O_k (or of O_j in case that O_k is static). The value of a pixel p in O_j is then defined by finding a pixel q from which we copy the value. We determine the position of q by walking from p towards O_k along d, one pixel at a time. On the way, we maintain a counter, initialized at one, that is incremented whenever an encountered pixel is inside O_j and decremented when outside O_j. When the counter reaches zero, we have found q. Using this simple inpainting leads to a significant improvement (Fig. 9).

3.3 High Dynamic Range Motion Blur

Our method can also be extended to increase the expressiveness and quality of the resulting images by hallucinating high-dynamic range (HDR) content. In our original approach, light sources that undergo motion blur will appear dull, as the maximum value in an image is one and will be spread over a large area by the kernel.

In real-world environments luminance can span a wide range. While our human eyes can adapt to large intensity variations, with standard photography, values in the sensor might saturate or be clipped. High-dynamic-range imagery is produced by recording several images with different exposure times, which are fused to

capture a larger range of intensities. Working with a high-dynamic range representation has a significant effect on the result. A clipped value when blurred will lead to a dimmed result in comparison with its original version. Having values that exceed the limits of the display will still be dimmed by a blur, but will maintain a higher intensity and potentially even still saturate after the blur is applied.

Bright and glowing elements are often clipped, e.g., a bright car headlight in a night scene. In our solution, a user can indicate regions in which values were clipped by placing a bounding rectangle around them. Then our solution expands the values in this region from the range of $[t,1]$ to a range of $[t,2^T]$, where t and T are user-defined thresholds (per default, $t = 0.98$, $T = 2$) using the function $f(x) = t * pow(1 + (x-t)/(1-t), T)$. Fig. 16 shows a result.

4 Results

We have implemented our framework using OpenCV/C++. All results were created on a laptop with an Intel i5 2.2GHz and 8GB RAM. We did not optimize the performance of our solution; on average, it takes 3 seconds to segment the content and apply the motion blur vectors on an 960×540 image. Since the input is simple, novice users can create convincing results in less than 10 seconds.

A large variety of examples are shown in Fig. 18. The top row shows a boy with a ball, where the motion blur on the ball adds activity to the scene and guides the observers focus. A similar result is obtained in the second row - here the background was blurred to underline the stormy sea and sky, which leads to an increased focus on the surfer. The third-row example adds a clear sense of speed to the horse movement that was missing from the original shot. The fourth row illustrates the smoothness of the motion-blurred results, even in the presence of a complex path, which adds to the calm atmosphere of the photo. Similarly, the fifth example additionally shows that the gradient in the sky remains almost perfectly unaltered. Row six and seven illustrate how motion blur can indicate actions of people, resulting in a more dynamic photo. Finally, the eighth row shows how HDR can add to the apparent brightness of the back lights. The strong motion blur is used to add a sense of danger with regard to the slippery road to the image. All these examples show the large variety of options for the controlled use of motion blur. In the following, we demonstrate key features of our algorithm.

The user defines objects with very little effort, as evidenced by the simple input shown in Fig. 7. The black rectangular bounding box contains potential foreground regions; additional scribbles were added for refinement (red and green scribbles).

Figure 7: User input for image segmentation and resulting masks.

To apply motion blur to the background, a user can draw a motion vector over the desired area. Corresponding results are shown in Fig. 8 (top) applying a convolution kernel to every pixel. The color leakage artifacts seen in Fig. 8 (top) are minimized by our approach, creating a natural transition along object and background edges, as shown in Fig. 8 (middle). The user can decide to change the motion path at any time to expore the resulting effect, such as in Fig. 8 (bottom).

Figure 8: Background motion blur results. Top: motion blur with artifacts. Middle: our artifact minimization approach. Bottom: a different motion vector is defined. (Images source: pixabay.com)

Fig. 9 illustrates cases where objects are set in motion, like the car (left) and eagle (right). Fig. 9 (top) illustrates the corrected result from Fig. 6. For a matter of comparison, Fig. 9 (bottom) shows a different motion direction applied to the target objects.

Figure 9: Foreground motion blur results. Our artifact minimization blending is applied to all results. Top and bottom rows differ on the motion vector chosen by the user for the same objects.

For scenes with more than one object, each object can be motion blurred with different motion paths and intensities. Fig. 10 (left) shows one motion-blurred target (one dog, one balloon), while Fig. 10 (right) shows the result when simulating different motion directions and speeds. Analogously, Fig. 11 illustrates how the impression of a scene can be influenced when switching the motion targets; here, either to the player (left) or to the ball (right). Similarly, motion blur can be used to change the semantics of a scene, such as in Fig. 12, where the hand motion is used to indicate an agreement.

Figure 10: Multiple objects with distinct motion directions. (Images source: pixabay.com)

Figure 11: Motion blur on different targets, changing scene impression. (Image source: pixabay.com)

Figure 12: Motion blur indicating the semantics of the scene. (Image source: pixabay.com)

Fig. 13 shows the importance of the composition order. Fig. 13 (right) follows our back-to-front solution, avoiding the artifacts on Fig. 13 (left).

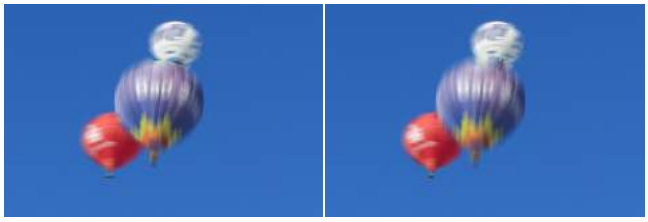

Figure 13: Motion blur with overlapping objects. (Image source: pixabay.com)

Fig. 14 compares our results to a manual manipulation in Photoshop. Here, the background received a linear motion blur. Just applying Photoshop's Motion Blur and Path Blur tools (left, center-left) leads to similar artifacts as in Figs. 5 and 6. In consequence, an artist needs to first derive layers, which can be non-trivial. Further, each layer requires manual inpainting, which necessitates experience and can be a difficult task, especially for complex content. After processing the layers, a manual compositing is needed to derive an acceptable result (center-right). Our approach leads to similar results (right), while avoiding manual layering and compositing. Further, our occlusion handling performs well without any user interaction. In consequence, our approach is easy to use and allows even beginners to quickly produce convincing results.

Figure 14: Comparison with Photoshop tools. From left to right, Photoshop results using Motion Blur, Path Blur, Content Aware Fill with layer compositing, and our approach. (Image source: pixabay.com)

Our framework also supports general motion curves. To this extent, the motion curve could be decomposed into linear segments, which are used by our solution and the results are accumulated. In practice, we found that it is sufficient to directly apply the motion curve and choose for a vertical and horizontal occlusion handling, depending on the local orientation, which is computationally more efficient. Such general curves are well suited to simulate a long exposure with non-linear motion, e.g., due to a hand-held acquisition. Fig. 15 demonstrates a curved motion paths on the ball to simulate a non-linear bounce (top) and a time-lapse sequence (bottom).

Figure 15: Motion blur with a non-linear path. (Images source: pixabay.com (top) and maxpixel.freegreatpicture.com (bottom))

The hallucinated HDR content created with our solution is shown in Fig. 16. The light blue rectangles illustrate the selected the region for the HDR expansion. The change affects the look of the motion blur, which results in a more realistic effect (right) compared to the standard approach (middle).

Figure 16: Hallucinated HDR expanding high intensity values. The bright lights of the cars create visible trails (top, middle), as does the sun when applying a strong background blur (bottom). (Images source: pixabay.com)

While our approach can produce convincing results, it also has its limitations. The selection of objects is quite simple, but when moving objects overlap, a decision is needed to determine which element is to be considered in front. While our interface imposes an ordering, it does not support objects motion that would lead to several encounters of two objects during which their respective ordering changes. Nevertheless, this restriction is not very problematic because a long motion path results in extreme motion blur, which is rarely attractive and usually not employed by a user. For this reason, we also did not include control over the velocity of objects along a motion trajectory, which would be easy to add to our interface.

It is still true that strong motion, which uncovers large areas, is also problematic for our inpainting procedure. The estimated initially-hidden content will become more visible in these cases and might then reflect the observer's expectations. Especially unveiled objects whose shape is easy to estimate for an observer might result in a discrepancy. Nevertheless, for most practical cases, our inpainting is sufficient, as evidenced by the many natural-looking examples in this paper.

Other challenges are transparent and reflective surfaces, which are difficult to handle. Fortunately, a human observer is typically not very strong in interpreting physical effects correctly and with ease. Regarding reflections, if the blur of the original object and its reflected counterpart do not perfectly match, the illusion might be sufficient. To facilitate adding plausible reflections, we use a simple extension to our interface that allows a user to scribble a mirror axis, which is used to copy the annotations from one side of the reflection to the other. Fig. 17 shows an example.

Figure 17: Motion applied to target object and its reflection. (Image source: pixabay.com)

Figure 18: The diversity of scenes exploit by our framework. Blue lines indicate the motion path. (Images from: unsplash.com, pexels.com and pixabay.com)

5 Conclusion

We presented a solution to add motion-blur effects to a single image in a post-process. Our solution allows for very simple user interaction and requires only little effort. Despite the method's simplicity, convincing results can be obtained in seconds and the outcome is easier to control than with a real-world capture.

Acknowledgments

This work was partially funded by the Brazilian agency CNPq. We also would like to thank unsplash.com, pexels.com, maxpixel.freegreatpicture.com and pixabay.com for the copyright-free (CC0) images used in this paper.

References

[1] J. T. Barron, A. Adams, Y. Shih, and C. Hernández. Fast bilateral-space stereo for synthetic defocus. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[2] M. J. Cieślak, K. A. Gamage, and R. Glover. Coded-aperture imaging systems: Past, present and future development–a review. *Radiation Measurements*, 92:59–71, 2016.

[3] GNU Image manipulation program (GIMP). https://www.gimp.org/. Accessed: 2017-12-15.

[4] J.-P. Guertin, M. McGuire, and D. Nowrouzezahrai. A fast and stable feature-aware motion blur filter. In *Proceedings of High Performance Graphics*, HPG '14, pp. 51–60, 2014.

[5] R. T. Held, E. A. Cooper, J. F. Obrien, and M. S. Banks. Using blur to affect perceived distance and size. *ACM Trans. Graph. (TOG)*, 29(2), 2010.

[6] J. Jimenez. ACM Siggraph courses: Advances in real-time rendering - next generation post processing in call of duty. ACM, 2014.

[7] R. Laganiere. *OpenCV 3 Computer Vision Application Programming Cookbook*. Packt Publishing Ltd, 2017.

[8] Lytro camera. https://www.lytro.com/. Accessed: 2017-10-20.

[9] C. MacManus. The technology behind sony alpha dslr's steadyshot inside. http://bit.ly/2CWbMDw. Accessed: 2017-12-20.

[10] M. McGuire, P. Hennessy, M. Bukowski, and B. Osman. A reconstruction filter for plausible motion blur. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2012.

[11] H. Nagahara, S. Kuthirummal, C. Zhou, and S. K. Nayar. Flexible depth of field photography. In *European Conference on Computer Vision*, pp. 60–73. Springer, 2008.

[12] F. Navarro, F. J. Serón, and D. Gutierrez. Motion blur rendering: State of the art. *Computer Graphics Forum*, 30(1):3–26, 2011.

[13] B. Peterson. *Understanding Shutter Speed: Creative Action and Low-Light Photography Beyond 1/125 Second*. Amphoto Books, 2008.

[14] Adobe photoshop. https://adobe.ly/1g8lSDp. Accessed: 2017-12-15.

[15] Realsmart motion blur. http://revisionfx.com/products/rsmb/.

[16] G. Rosado. Motion blur as a post-processing effect. In H. Nguyen, ed., *GPU Gems 3*, pp. 575–581. Addison-Wesley, 2008.

[17] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph. (TOG)*, 23(3):309–314, 2004.

[18] T. Saito and T. Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, pp. 197–206, 1990.

[19] N. Z. Salamon, M. Lancelle, and E. Eisemann. Computational light painting using a virtual exposure. In *Computer Graphics Forum*, vol. 36, pp. 1–8. Wiley Online Library, 2017.

[20] Synthcam. http://bit.ly/2FQy4LZ, 2011. Accessed: 2017-12-09.

[21] J. Telleen, A. Sullivan, J. Yee, O. Wang, P. Gunawardane, I. Collins, and J. Davis. Synthetic shutter speed imaging. *Computer Graphics Forum*, 26(3):591–598, 2007.

[22] Y. Wexler, E. Shechtman, and M. Irani. Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3), 2007.

[23] R. A. Yeh, C. Chen, T. Y. Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5485–5493, 2017.