Controllable Motion-Blur Effects in Still Images

Xuejiao Luo, Nestor Z. Salamon, and Elmar Eisemann

Abstract—Motion blur in a photo is the consequence of object motion during the image acquisition. It results in a visible trail along the motion of a recorded object and can be used by photographers to convey a sense of motion. Nevertheless, it is very challenging to acquire this effect as intended and requires much experience from the photographer. To achieve actual control over the motion blur, one could be added in a post process but current solutions require complex 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 handles object boundaries to avoid artifacts produced by standard filtering methods. We illustrate the effectiveness of our solution with various complex examples, including multi-directional blur, reflections, multiple objects, and illustrate how several motion-related artistic effects can be achieved. Our post-processing solution is an alternative to capturing the intended real-world motion blur directly and enables fine-grained control of the motion-blur effect.

Index Terms—motion blur, long exposure, image processing, post-production.

1 Introduction

Motion blur, as an artistic effect, is able to convey a sense of motion in still images. A photographer can create such effects by opening the camera shutter for an extended period of time. During the exposure, moving objects with respect to the camera will result in a different projection location on the camera sensor, which produce visible trails in the final image [1]. While being an important technique [2], 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. In some other 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. Similarly difficult is the production of a precise camera/object trajectories on the image plane.

Easier than capturing the result directly, is to produce it in a post-process. However, current image editing software provides mostly blur tools for general purposes, which requires users to manually extract regions of interest and experiment with different effects. Additionally, standard filters typically result in artifacts at object boundaries, such as undesired color leakage.

Our method enables a user to easily add motion blur to a single still image with only little user intervention. 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 to deliver convincing results, while keeping the user interaction simple and avoiding additional scene information. The algorithm in this paper builds upon our original method presented in [3]. This extended version provides a user evaluation and several new contributions, namely, a multi-directional motion blur to support more complex object and camera motions, as well as several approaches to produce artistic effects based on commonly applied shutter techniques.

This article is organized as follows. In the next section, we revisit previous work. We then describe our approach (Sec. 3), including methods to produce various artistic effects (Sec. 4), before presenting the results of our method (Sec. 5) and concluding (Sec. 6).

• *X. Luo, N. Z. Salamon and E. Eisemann are with the Computer Graphics and Visualization Lab (CGV), TU Delft, The Netherlands.*

Manuscript received xxxx, 2018; revised xxxx, 201x.

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 [4]. Several hardware and algorithmic solutions have been proposed. Light-field cameras [5], special sensors [6], coded apertures [7], stereo setups [8], or synthetic reconstruction [9], enable post-processing of the depth-of-field effect.

Motion blur conveys a sense of motion but is often considered an undesirable artifact, as it can result from camera shake. In consequence, modern cameras usually involve stabilization systems [10] 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 [11]. 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. Direct accumulation of the images would result in ghosting artifacts due to an exposure gap between the individual shots. 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 [12] rely on optical flow to estimate the movement between images to then apply a directed blur kernel. A virtual

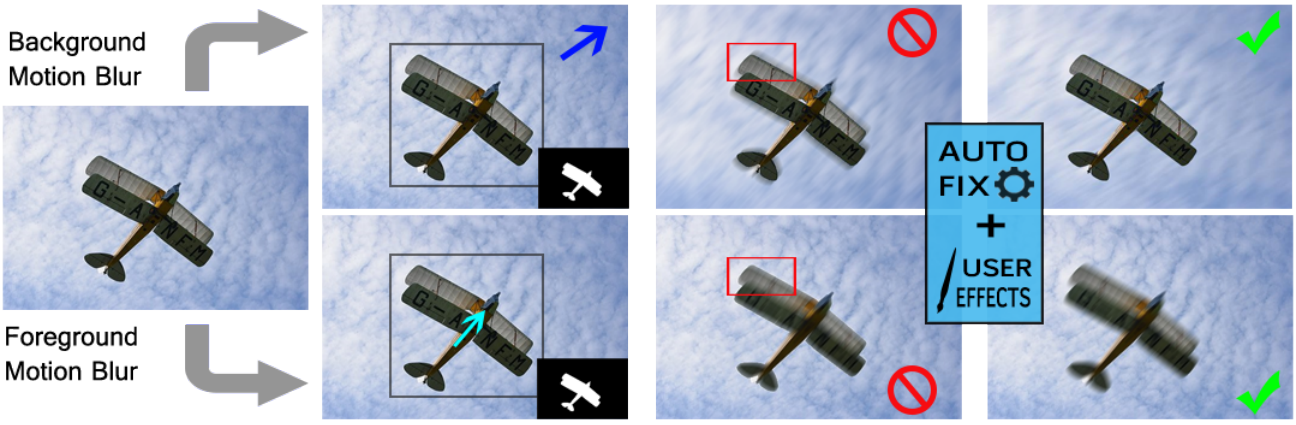

Fig. 1: Overview. From left to right: the user can select target regions of an input image using simple annotations (bounding box, scribbles). A motion can be defined for the desired target region, which launches a filtering process to add plausible motion blur. (Input image source: pixabay.com)

exposure was also used for light painting, which is able to describe the motion blur caused by a moving light source [13].

When relying on video input, a pixel-wise blur can be generated from an estimated motion trajectory of the object [14]. The authors create a blur kernel from the disparity along motion vectors in the stabilized video. However, background and camera motion cannot be decoupled and the blur is limited to moving areas. Moreover, the video input can grow significantly when dealing with slow moving objects, such as clouds or stars. Our method can be applied to any object/region in a single image.

Commercial solutions for computational motion blur on a single image exist. One example is the GIMP [15] motion blur tool and the Adobe Photoshop [16] motion-blur effect. These require 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.

Motion depiction has also been investigated for virtual scenes, even with a programmable interface for expressive results [17]. For efficient motion blur computation, perceptual factors can be integrated in the computations [18]. Most real-time 3D applications, such as [19], [20], [21], [22], make use of deferred shading [23] to derive information such as per-pixel motion, depth, or object id, which are used in a filtering process. 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. 1 shows an overview of our solution.

The user can select objects via an image-segmentation method (Sec. 3.1) and can then define the motion of the selected object or area. The algorithm produces a motion-blurred result while avoiding artifacts around object boundaries (Sec. 3.2). First, we will describe our solution for linear motion per object before addressing general motion paths per pixel (Sec. 3.3). Additionally, we present several extensions of our approach to add high dynamic range (Sec. 4.1), and artistic effects inspired by photographic techniques (Sec. 4.2 and Sec. 4.3).

3.1 Object Selection

A moving object in the foreground blends with the background, while a moving background does not blend into a 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. Consequently, we will first focus on how to extract objects from the image.

For the segmentation of objects, a variety of methods could be used. Recently, machine learning techniques (e.g. [24], [25], [26]) are increasingly successfully employed for segmentation tasks. Still, for creative content creation, users might desire segmentation masks that differ from the typical automatic segmentation inferred from a learning process. For this reason, we opted for a constrained segmentation based on user annotations. One of the most user-friendly segmentation methods is GrabCut [27]. 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 [28] to define the different objects.

Fig. 2 shows two examples of the GrabCut extraction. The user defined an object's bounding box and, if necessary, scribbles, which assign potential foreground and background regions. The extracted mask defines the foreground object (the thumbnails in the lower right corner).

Fig. 2: GrabCut foreground extraction. (Images source: pixabay.com)

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 an ordering of the objects, which can be adapted by the user. These objects can then be processed individually with their own motion path.

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. 3 illustrates a multi-object labeling.

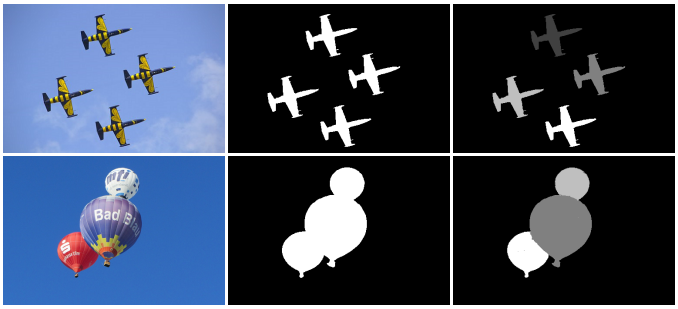

Fig. 3: Multiple objects segmentation. Distinct target regions can be segmented, allowing localized and distinct effect control. (Images source: pexels.com (top) and pixabay.com (bottom))

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 Uniform Motion Blur

A linear motion can be defined by a motion direction and length. We simulate the motion blur by convolving an object with a motion-blur kernel (*psf*) defined by the motion trajectory. For example, a horizontal translation by n pixels results in a horizontal kernel of n pixels, which is normalized such that its integral (sum of all pixels) is equal to one (here, each kernel pixel contains $1/n$). The latter avoids creating energy 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 determined and extended by two pixels. Then, for each bounding-box pixel, we compute one minus the distance of the pixel 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. For general motion paths, the provided path is decomposed into linear segments, which are individually handled as before.

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. 4.

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. 5), while one would have expected a fuzzy boundary.

For 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 composed with the current background, which handles the problems in Fig. 5. After explaining the corresponding details, we will show how to address the issues of Fig. 4.

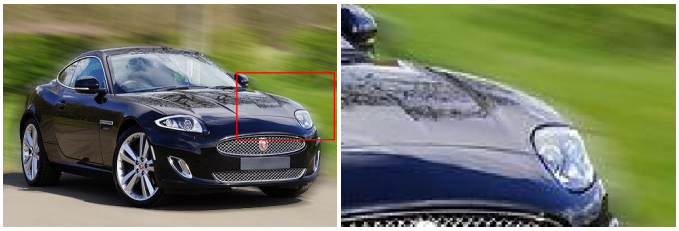

Fig. 4: Color leakage when not respecting object boundaries.

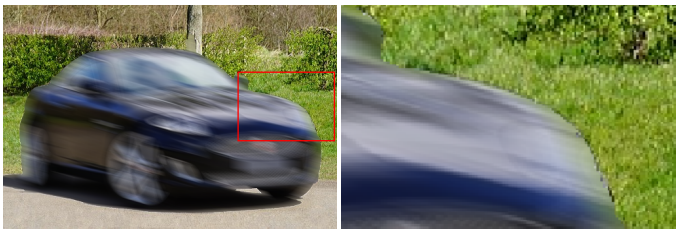

Fig. 5: Unrealistic sharp edges from edge-aware filtering.

Composing Fore- and Background

To describe the algorithm to compose fore- and background, 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 having ones in pixels that correspond to those labeled with O_k (one can interpret this step as adding an alpha channel). 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 equal to one), which implies that 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 compose 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. Neglecting disoccluded regions and assuming that they look like the original image will result in the artifacts shown in Fig. 4. Similarly, assuming the disoccluded pixels are simply black results in a dark halo.

Handling Disocclusions

To correct for the disocclusion artifacts, we propose an inpainting procedure. While more advanced solutions could be employed (i.e. [29], [30], [31]), we found the simpler strategy usually

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 [22], we mirror the content of O_k into the area that is covered by O_j. We choose the mirror direction d based on 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. The zero counter indicates that we have traveled the same distance inside O_j as outside, it is then located at a reflected position with respect to the object boundary. If we encounter an object O_l ($l > j$) while following d, we reverse d, which performs a *ping-pong* inpainting. Using this simple inpainting leads to a significant improvement (Fig. 11).

3.3 Multi-Directional Motion Blur

Now that we have discussed the case of per object motion, more complex motion paths for each pixel are addressed with our multi-directional motion blur. This extension increases the expressiveness of the algorithm significantly. For example, when photographers move a camera forward or sideways, the perspective foreshortening leads to blur effects that can enhance the impression of depth, yet they cannot be described with a single linear path per object. In other cases, such as a spinning wheel, a linear motion trajectory fails in reproducing a plausible motion-blur effect. In this section, we propose an extension to the previous principle. Here, unlike convolving the original image region with a single motion-blur kernel, each pixel will receive its own motion direction. In order to facilitate the annotation, the user defines a few motion paths, whose properties are propagated throughout the image using a diffusion process [32], [33]. This diffusion process results in an image with minimal gradient variation, under the constraint that the original annotations are maintained.

To describe this process formally, we will derive an image, in which each pixel will contain a motion vector defined by a length l and a direction $d := (\cos(\theta), \sin(\theta))$ for a given angle θ. Initially, the user will only sparsely annotate a few pixels. Let $\mathscr{I}$ be the set of pixels for which the user provided an input. For an index $(i, j) \in \mathscr{I}$, we denote the user-defined motion annotation as $m_{i,j}$. The image M containing the diffused three-component motion vectors is defined by:

$$\Delta M = 0 \quad with \;\; M(i,j) = m_{i,j} \;\; \forall (i,j) \in \mathscr{I}$$

The resulting image M is smooth, as the equation $\Delta M = 0$ implies that the gradient is minimized, while the user annotations are maintained. This equation system can be solved with an iterative process. To this extent, one can iteratively average neighboring pixels via $M(i,j) = (M_{i-1,j} + M_{i+1,j} + M_{i,j-1} + M_{i,j+1})/4$, while maintaining the values of the pixels in $\mathscr{I}$, until convergence. More efficient solutions involve using multi-grid [32], sparse-system solvers [33], or even optimized methods for diffusion curves [34].

The initial user annotations of the pixels in $\mathscr{I}$ are done using a special annotation tool. The user defines the motion direction by drawing a line segment whose orientation and length define the desired values (d, l). By default, the pixels below the segment will be added to $\mathscr{I}$ but it is also possible to mark an area, or single pixel, with a brush to then associate the drawn segment to this area.

Given the diffused motion vectors M, the next step is to derive the corresponding motion-blurred image I_m from the input image I. The idea is to start in each pixel and walk along the defined motion trajectory. This path line integration is similar to the process of visualizing flow [35]. For each pixel p, we compute the motion-blurred result $I_m(p)$ by averaging the values along a path of the motion length $\lfloor M(p).l \rfloor$ over the input image I, guided by the direction $M(p).d$:

$$I_m(p) = \frac{1}{\lfloor M(p).l \rfloor} \sum_{i=0}^{\lfloor M(p).l \rfloor} I(p_i),$$

where

$$\begin{aligned} p_0 &= p \\ p_{i+1} &= p_i + M(p).d \end{aligned}$$

Disocclusions during this multi-directional motion blur are handled similarly to before, only now the mirror-padding is applied according to the motion path. Fig. 6 shows a result (bottom row, right). The user-provided motion paths and the diffused result are illustrated as well.

Fig. 6: Multi-directional motion blur. Top row: the input image with user scribbles and multiple motion paths (left) and the segmentation mask (right). Bottom row: the diffusion map M with the colors encoding the direction and length of each motion path (left), and the final result (right). (Image source: pixabay.com)

4 Extensions

Our method is able to add the illusion of a standard motion blur to a still image. In this section, we will describe extensions of our solution. First, we show how to increase the realism of the motion blur for very bright sources by hallucinating high-dynamic range content, then we propose two artistic additions, which are often used in practice, the Harris shutter and addition of motion trails.

4.1 High Dynamic Range Motion Blur

For strong light sources exceeding the range of the sensor sensitivity and thus the pixel values, the impact on the appearance of the motion blur can be easily underestimated. 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. Bright and glowing elements are often clipped, e.g., a bright car headlight in a night scene. High-dynamic-range (HDR) imagery is produced by recording several images with different exposure times, which are fused to capture a larger range of intensities. Typically, these images are then 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. To achieve this effect, we propose to virtually produce HDR content.

In our solution, a user can indicate regions in which values were potentially 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 $f(x) = t * pow(1+(x-t)/(1-t),T)$. While very coarse, the accuracy of such an expansion is of lower perceptual significance, still advanced conversions would be possible [36].

Fig. 7 shows an example of hallucinated HDR content created with our solution. 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).

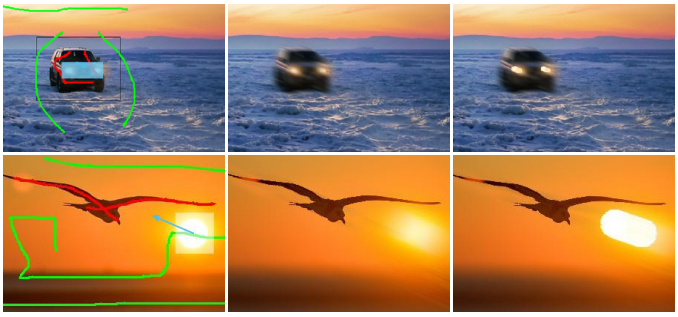

Fig. 7: Hallucinated HDR expanding high intensity values. The bright lights of the car create visible trails (top), as does the sun when applying a strong background blur (bottom). (Images source: pixabay.com)

4.2 Harris Shutter using Motion Blur

To show the flexibility of our solution and ability to also reproduce artistic effects used in photography, we added support for the simulation of a Harris shutter effect. The effect conveys an appealing and colorful outcome by masking certain channels over time. Typically, the shutter effect consist of capturing the scene in different time intervals, using only a single channel for each exposure [37]. Alternatively, it can also be achieved by recording a video and using the complementary channels from different frames to composite the final image. In other words, motion in a scene will result in several differently-colored projections.

In our solution, the motion blur is used to create the time-shifted frames. The target region is defined by the user and we manipulate each channel separately. The green channel is used as reference (middle) point. The red and blue channels contain the result after following half the length of the provided motion blur in opposite directions. Results of the Harris shutter simulation are shown in Fig. 8. With this artistic effect, one can steer attention to scene composition.

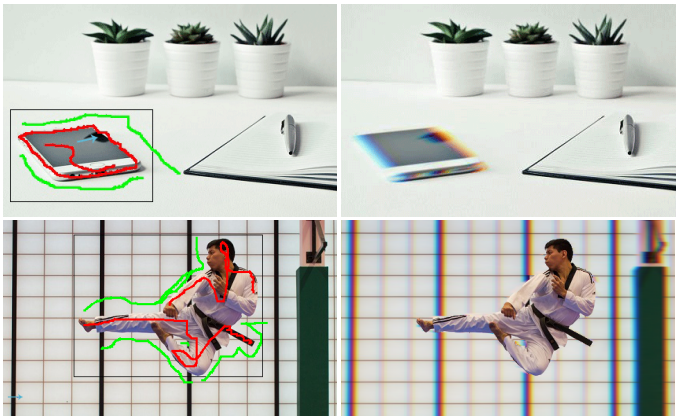

Fig. 8: Harris shutter motion blur results in a colorful outcome to steer attention. (Images source: pixabay.com)

4.3 Motion Trails

Another artistic means to illustrate movement are motion trails. These are used in photography but usually shot in front of a dark background. Here, a flash or strong light is used at the end of the capturing process. Hereby, the object in its final position will be more visible than during the previous time frame. As a consequence, it seems as if the object leaves a trail behind that is easily understood by the observer as a displacement. We can simulate such an effect by compositing the foreground object on top of the motion-blurred result.

An example is shown on Fig. 9, using the multi-directional blur to create motion trails simulating the punch impact. Please notice that this effect can also be selectively applied on different parts of an object by using the diffused multi-directional motion blur.

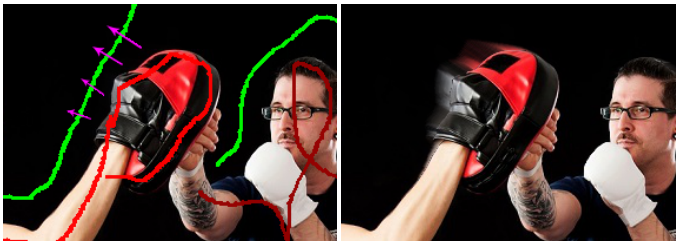

Fig. 9: Motion trails indicating the action while keeping the final position on focus. (Image source: pixabay.com)

5 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. The uniform motion blur is linear in terms of complexity with

respect to the image dimensions. It takes 3 seconds to segment the content and apply a motion path on a 960×540 image. Since the input is simple, novice users can create convincing results in less than 10 seconds. For the multi-directional motion blur, the computation time is directly linked to the diffusion process. The image size, number of motion paths, and number of iterations until convergence do govern the cost. While it would be possible to use hierarchical [38] or advanced solvers [33], [34] on the GPU, which can run at interactive rates, we use a standard CPU solver to increase compatibility. In practice, users took around 2 minutes to obtain the results shown in this paper.

A large variety of examples are illustrated in Fig. 25. The top row (a-b) shows a boy with a ball, where the motion blur on the ball adds activity to the scene and guides the observer's focus. The same row (c-d) adds a clear sense of speed to the horse movement that was missing from the original shot. On the second row, a similar result is obtained: in (a-b) the background was blurred to underline the stormy sea and sky, which leads to an increased focus on the surfer; (c-d) show that the gradient in the sky remains almost perfectly unaltered. Row three illustrates how motion blur can emphasize actions to underline the semantics of a photo. The fourth row (a-b) 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. The same row (c-d) shows how our HDR effect can add to the apparent brightness of the back lights. The motion blur is used to add a sense of danger with regard to the slippery road in the image. The fifth row, illustrates the effects of the Harris Shutter, where the originally simple scenes are enriched by the vivid color additions. Finally, the sixth and the seventh rows show applications of the multi-directional motion blur involving diffused motion vectors. In the sixth row, we show how the motion blur can support the emphasis on the main character (a-b), or create the impression of vertigo (c-d). The seventh row shows a subject emphasized using motion lines (a-b) and also a spinning motion to illustrate the generality of the solution (c-d). The images exemplify 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 previously shown in Figs. 2 and 3. To apply motion blur to the background, a user can draw a motion vector over the desired area. The color leakage artifacts seen in Fig. 4 are minimized by our approach, creating a natural transition along object and background edges, as shown in Fig. 10(top). The user can decide to change the motion path at any time to explore the resulting effect and also can apply to more cases, such as in Fig. 10 (bottom). Fig. 11 illustrates cases where objects are set in motion, like the car (left) and eagle (right). Fig. 11 (top) illustrates the corrected result from Fig. 5. For a matter of comparison, Fig. 11 (bottom) shows a different motion direction applied to the target objects.

For scenes with more than one object, each object can be motion blurred with different motion paths and intensities. Fig. 12 (left) shows one motion-blurred target (one balloon, one dice), while Fig. 12 (right) shows the result when simulating different motion directions and speeds. Analogously, Fig. 13 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 guide an observer, such as in Fig. 14, where the hand motion influences the way an observer analyzes the scene.

Fig. 10: Background motion blur results. Rows differ on the motion vector chosen by the user for the same objects.

Fig. 11: Foreground motion blur results. Our artifact minimization blending is applied to all results. Rows differ on the motion vector chosen by the user for the same objects.

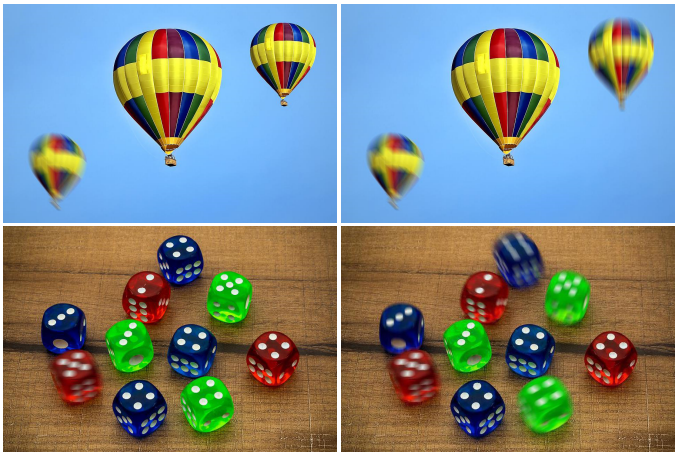

Fig. 12: Multiple objects with distinct motion directions. (Images source: pixabay.com)

Fig. 13: Motion blur on different targets, changing scene impression. (Image source: pixabay.com)

Fig. 14: Motion blur indicating the semantics of the scene. (Image source: pixabay.com)

When using general motion paths, they can be treated as several linear segments. In practice, it is typically sufficient to directly apply a convolution with the whole path and only perform a vertical and horizontal occlusion handling, depending on the local orientation, which is more efficient to evaluate. General motion 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).

Fig. 15: Uniform motion blur with a non-linear path. (Images source: pixabay.com (top) and freegreatpicture.com (bottom))

Fig. 16 shows multi-directional motion blur results with a complex motion per pixel. The purple arrows indicate the different directions of the motion paths. The results artistically emphasize the character (top), increase the perception of speed (middle), and simulate the spinning motion on a timelapse (bottom).

Fig. 17 uses hallucinated HDR content to maintain the brightness of the sun. Fig. 18 adds the Harris shutter effect, while Fig. 19 shows expressive motion trails.

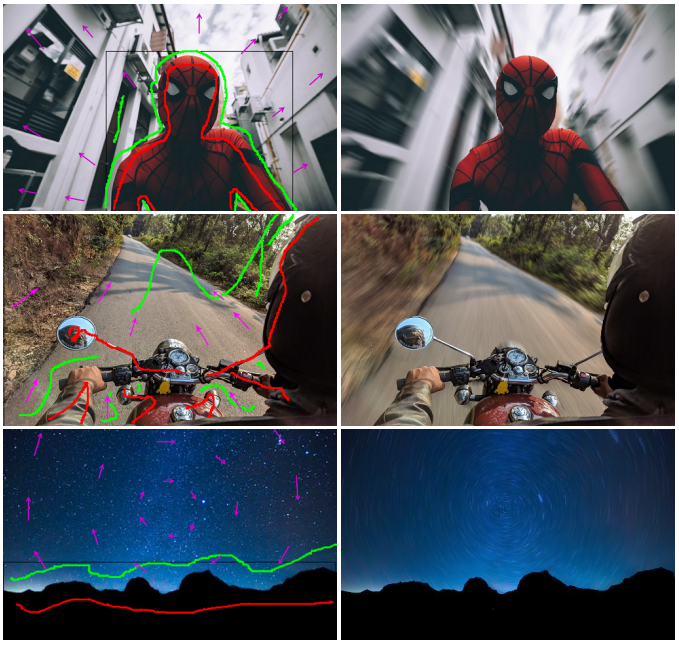

Fig. 16: Multi-directional motion blur results. (Images source: unsplash.com (top), pexels.com (middle), and pixabay.com (bottom))

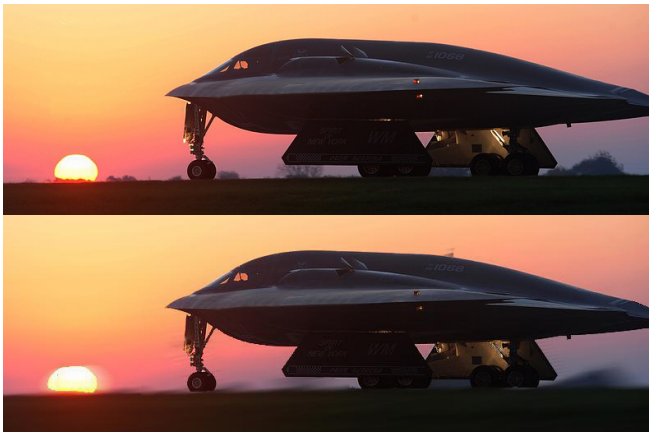

Fig. 17: High Dynamic Range motion blur keeping high intensity values. (Image source: pixabay.com)

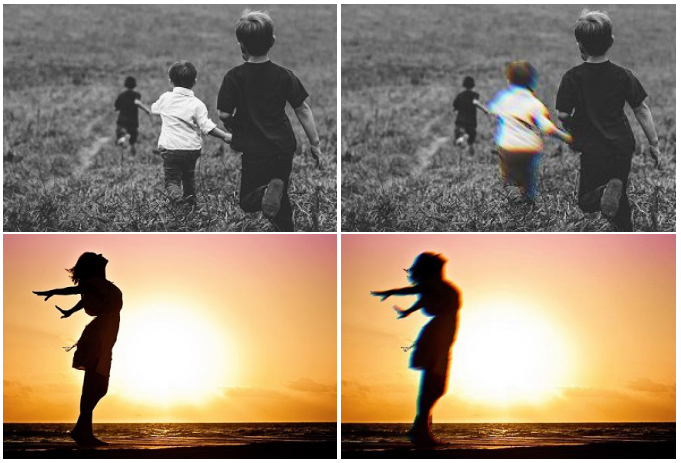

Fig. 18: Harris Shutter motion blur results creating subtly colored motion and vintage effects. (Images source: pixabay.com)

Fig. 19: Motion trails simulating distinct movements while keeping the subject on focus. (Image source: pixabay.com)

Discussion

Our approach introduces novel effects that are not easily reproducible in standard software. Motion blur can also be applied to an object using Photoshop, but it requires an extended workflow explained in the following. Fig. 20 shows the outcome of our method and 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. 4 and 5. In consequence, an artist needs to first derive layers, which can be non-trivial. Further, each layer requires manual inpainting, which can be a difficult task and require experience, 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 automatically.

Fig. 20: Comparison with Photoshop tools for the background motion blur. From left to right, Photoshop results using Motion Blur, Path Blur, Content Aware Fill with layer compositing, and our approach. (Image source: pixabay.com)

Despite its simplicity, our method's inpainting typically enables natural looking motion-blurred result for moderately strong motion blur and requires no user interaction for occlusion handling. The inpainting mechanism provided by Photoshop [29], [39] is more general and often provides a very detailed infilling. Nevertheless, it does not take the motion direction into account and might create unwanted structures that cannot be found in close proximity of the inpainted area. A comparison is shown in Fig. 21.

Fig. 21: Inpainting results from Photoshop (center) and our solution (right) when a large object is removed. (Image source: pixabay.com)

A direct comparison of our method to manually recreating motion blur effects in professional software tools is difficult, as the expertise of the users plays a significant role. To still provide some insights on practical usage, we chose to perform a simple evaluation with 10 users. These users had varying degrees of expertise in Photoshop, but used our solution for the first time. We gave them three photos that were relatively easy to segment and decompose in Photoshop to not require much expertise. They were asked to mimic a motion-blurred result. Overall, our system was still considered easier to use (4.3 (our) vs. 2.9 (PS) on a Likert scale of 5 - higher being better). Less time was spent using our method to create the desired results (average: 6 minutes 13 seconds (our) vs. 15 minutes 25 seconds (PS)) and the users were more satisfied with their results (4.4 (our) vs. 3 (PS), on a Likert scale of 5 - higher being better). While giving an indication, this evaluation did not even cover all aspects of our solution. A more extensive study remains future work. The user-evaluation details are presented in the supplementary material.

While our approach can produce convincing results, it also has its limitations. When moving objects overlap, a decision is needed to determine which element is to be considered in front, as illustrated in Fig. 22. However, our solution does not support object motion that would lead to several encounters of two objects changing their respective order.

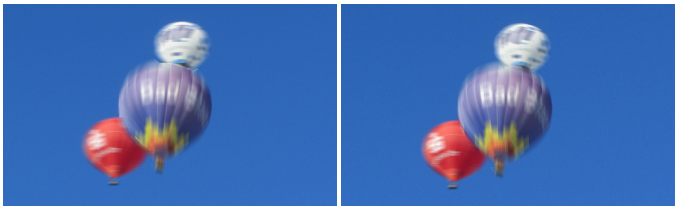

Fig. 22: Motion blur with overlapping objects. Arbitrary depth assignment with artifacts (left) and our back-to-front ordering (right). (Image source: pixabay.com)

Very strong motion is problematic because the area behind the moving object can become entirely visible (in the limit, the moving object would be completely transparent). A second problem can also occur when a moving object is initially partially covered. If the object has a shape that is easy to estimate for an observer, a discrepancy might arise. Fig. 23 illustrates such case. The head of the soccer player (left), for example, was enlarged in the inpainting, which darkens the motion trail. However, extreme motion blur is rarely attractive and usually not employed by a user. For most other cases, our inpainting is sufficient, as evidenced by the examples in this paper.

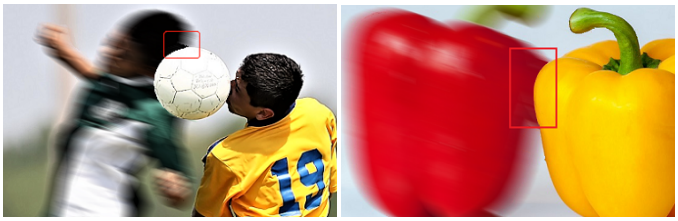

Fig. 23: Inpainting artifacts with strong motion and small occluded areas. (Images source: pixabay.com)

Finally, a challenge when applying motion blur is to deal with transparent and reflective surfaces. 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 still be sufficient. To facilitate adding plausible reflections, we use also created 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 when using uniform motion blur. An example is shown in Fig. 24.

Fig. 24: Motion applied to target object and its reflection. (Image source: pixabay.com)

6 Conclusion

We presented a solution to add motion-blur effects to a single image in a post-process. It allows for a simple user interaction and requires only little user 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. We illustrate that our solution provides support for complex motion paths and is able to reproduce several motion-blur-related photographic effects. Our algorithm can be applied to any image and does not require a specialized acquisition routine, which eases its use and increases its applicability.

Acknowledgments

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

References

[1] F. Navarro, F. J. Serón, and D. Gutierrez, "Motion blur rendering: State of the art," *Computer Graphics Forum*, vol. 30, no. 1, pp. 3–26, 2011.
[2] B. Peterson, *Understanding Shutter Speed: Creative Action and Low-Light Photography Beyond 1/125 Second*. Amphoto Books, 2008.
[3] X. Luo, N. Z. Salamon, and E. Eisemann, "Adding motion blur to still images," in *Graphics Interface 2018*, ser. GI 2018. Canadian Human-Computer Communications Society, 2018, pp. 99–105.
[4] R. T. Held, E. A. Cooper, J. F. O'brien, and M. S. Banks, "Using blur to affect perceived distance and size," *ACM Trans. Graph. (TOG)*, vol. 29, no. 2, 2010.
[5] "Lytro camera," https://www.lytro.com/, accessed: 2017-10-20.
[6] H. Nagahara, S. Kuthirummal, C. Zhou, and S. K. Nayar, "Flexible depth of field photography," in *European Conference on Computer Vision*. Springer, 2008, pp. 60–73.
[7] M. J. Cieślak, K. A. Gamage, and R. Glover, "Coded-aperture imaging systems: Past, present and future development–a review," *Radiation Measurements*, vol. 92, pp. 59–71, 2016.
[8] J. T. Barron, A. Adams, Y. Shih, and C. Hernández, "Fast bilateral-space stereo for synthetic defocus," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
[9] "Synthcam," http://bit.ly/2FQy4LZ, 2011, accessed: 2017-12-09.
[10] C. MacManus, "The technology behind sony alpha dslr's steadyshot inside," http://bit.ly/2CWbMDw, accessed: 2017-12-20.
[11] J. Telleen, A. Sullivan, J. Yee, O. Wang, P. Gunawardane, I. Collins, and J. Davis, "Synthetic shutter speed imaging," *Computer Graphics Forum*, vol. 26, no. 3, pp. 591–598, 2007.
[12] "Realsmart motion blur," http://revisionfx.com/products/rsmb/.
[13] N. Z. Salamon, M. Lancelle, and E. Eisemann, "Computational light painting using a virtual exposure," in *Computer Graphics Forum*, vol. 36, no. 2. Wiley Online Library, 2017, pp. 1–8.
[14] S. Liu, J. Wang, S. Cho, and P. Tan, "Trackcam: 3d-aware tracking shots from consumer video." *ACM Trans. Graph. (TOG)*, vol. 33, no. 6, 2014.
[15] "GNU Image manipulation program (GIMP)," https://www.gimp.org/, accessed: 2017-12-15.
[16] "Adobe photoshop," https://adobe.ly/1g8lSDp, accessed: 2017-12-15.
[17] J. Schmid, R. W. Sumner, H. Bowles, and M. H. Gross, "Programmable motion effects," *ACM Trans. Graph. (TOG)*, vol. 29, no. 4, 2010.
[18] F. Navarro, S. Castillo, F. J. Serón, and D. Gutierrez, "Perceptual considerations for motion blur rendering," *ACM Trans. on Applied Perception (TAP)*, vol. 8, no. 3, p. 20, 2011.
[19] G. Rosado, "Motion blur as a post-processing effect," in *GPU Gems 3*, H. Nguyen, Ed. Addison-Wesley, 2008, pp. 575–581.
[20] M. McGuire, P. Hennessy, M. Bukowski, and B. Osman, "A reconstruction filter for plausible motion blur," *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2012.
[21] J.-P. Guertin, M. McGuire, and D. Nowrouzezahrai, "A fast and stable feature-aware motion blur filter," in *Proc. of High Performance Graphics*, ser. HPG '14, 2014, pp. 51–60.
[22] J. Jimenez, "ACM Siggraph courses: Advances in real-time rendering - next generation post processing in call of duty," 2014.
[23] T. Saito and T. Takahashi, "Comprehensible rendering of 3-d shapes," in *Proc. of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '90, 1990, pp. 197–206.
[24] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, 2018.
[25] A. Romero, M. Drozdzal, A. Erraqabi, S. Jégou, and Y. Bengio, "Image segmentation by iterative inference from conditional score estimation," *arXiv preprint arXiv:1705.07450*, 2017.
[26] G. Lin, A. Milan, C. Shen, and I. D. Reid, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," in *Computer Vision and Pattern Recognition (CVPR)*, vol. 1, no. 2, 2017.
[27] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," *ACM Trans. Graph. (TOG)*, vol. 23, no. 3, pp. 309–314, 2004.
[28] R. Laganiere, *OpenCV 3 Computer Vision Application Programming Cookbook*. Packt Publishing Ltd, 2017.
[29] Y. Wexler, E. Shechtman, and M. Irani, "Space-time completion of video," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, 2007.
[30] 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 *Computer Vision and Pattern Recognition (CVPR)*, 2017.
[31] C. Barnes and F.-L. Zhang, "A survey of the state-of-the-art in patch-based synthesis," *Computational Visual Media*, vol. 3, no. 1, pp. 3–20, 2017.
[32] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin, "Diffusion curves: A vector representation for smooth-shaded images," in *ACM Trans. Graph. (SIGGRAPH)*, vol. 27, 2008.
[33] H. Bezerra, E. Eisemann, D. DeCarlo, and J. Thollot, "Diffusion constraints for vector graphics," in *Proc. of the 8th International Symposium on Non-photorealistic Animation and Rendering*, 2010.
[34] S. Jeschke, D. Cline, and P. Wonka, "A gpu laplacian solver for diffusion curves and poisson image editing," *Transaction on Graphics (Siggraph Asia 2009)*, vol. 28, no. 5, pp. 1–8, 2009.
[35] J. J. van Wijk, "Image based flow visualization," *ACM Trans. Graph. (TOG)*, vol. 21, no. 3, pp. 745–754, Jul. 2002.
[36] B. Masia, S. Agustin, R. W. Fleming, O. Sorkine, and D. Gutierrez, "Evaluation of reverse tone mapping through varying exposure conditions," in *ACM Trans. Graph. (TOG)*, vol. 28, no. 5, 2009.
[37] D. D. Busch, *Nikon D200 Digital Field Guide*. John Wiley & Sons, 2006.
[38] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin, "Diffusion curves: A vector representation for smooth-shaded images," *ACM Trans. Graph. (TOG)*, vol. 27, no. 3, 2008.
[39] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *ACM Trans. Graph. (TOG)*, vol. 28, no. 3, 2009.

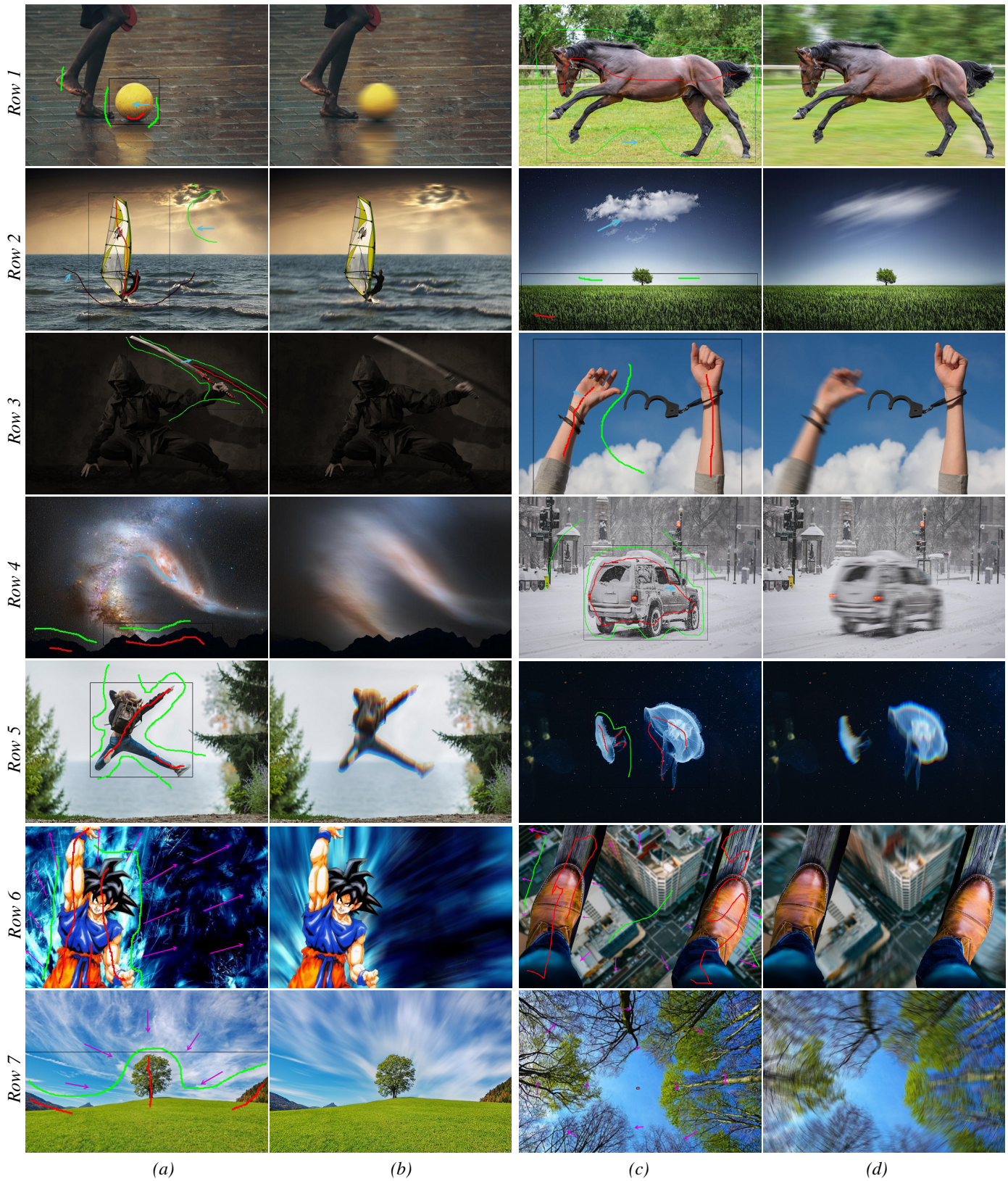

Fig. 25: The diversity of scenes exploit by our framework. Blue and purple arrows indicate, respectively, the motion path of uniform and multi-directional motion blur. (Images from: unsplash.com, pexels.com, pixabay.com and easylife-online.com)

Xuejiao Luo received her diploma (MSc) from Télécom ParisTech, France, in 2017. She is a PhD student at Delft University of Technology in the Computer Graphics and Visualization group. Her research interests include computer graphics and computer vision.

Nestor Z. Salamon received his Bachelor (2012) and Master (2015) degrees in Computer Science at Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil. From 2010 to 2015, he worked as Software Designer at Hewlett-Packard Brazil R&D. He is currently pursuing his PhD in Computer Graphics at Delft University of Technology (TU Delft), The Netherlands. His research interests include computational photography, image processing and multimedia content creation.

Elmar Eisemann is a professor at TU Delft, heading the Computer Graphics and Visualization Group. Before he was an associated professor at Telecom ParisTech (until 2012) and a senior scientist heading a research group in the Cluster of Excellence (Saarland University / MPI Informatik) (until 2009). He studied at the École normale supérieure in Paris (2001-2005) and received his PhD from the University of Grenoble at INRIA Rhône-Alpes (2005-2008). He spent several research visits abroad; at the Massachusetts Institute of Technology (2003), University of Illinois Urbana-Champaign (2006), Adobe Systems Inc. (2007,2008). His interests include real-time and perceptual rendering, alternative representations, shadow algorithms, global illumination, and GPU acceleration techniques. He coauthored the book ”Real-time shadows” and participated in various committees and editorial boards. He was local organizer of EGSR 2010, 2012 and HPG 2012. His work received several distinction awards and he was honored with the Eurographics Young Researcher Award 2011.