Pacific Graphics 2019
C. Theobalt, J. Lee, and G. Wetzstein
(Guest Editors)

*Volume 38 (2019), Number 7*

# ShutterApp: Spatio-temporal Exposure Control for Videos

Nestor Z. Salamon , Markus Billeter and Elmar Eisemann

Delft University of Technology, The Netherlands

**Figure 1:** *Left: single frame from 240Hz short-exposure video and a simulated long exposure at 30Hz by averaging 8 frames. Middle: Using the 240Hz input, our method enables mixing a long exposure in the periphery with a short exposure for the details on the pendulum. Via user annotations in the video, different shutter functions can be defined (top right). Annotations and shutter functions can be keyframed over time. Based on the annotations, our method defines an interpolated shutter function for each pixel (bottom right).*

**Abstract**

*A camera's shutter controls the incoming light that is reaching the camera sensor. Different shutters lead to wildly different results, and are often used as a tool in movies for artistic purpose, e.g., they can indirectly control the effect of motion blur. However, a physical camera is limited to a single shutter setting at any given moment. ShutterApp enables users to define spatio-temporally-varying virtual shutters that go beyond the options available in real-world camera systems. A user provides a sparse set of annotations that define shutter functions at selected locations in key frames. From this input, our solution defines shutter functions for each pixel of the video sequence using a suitable interpolation technique, which are then employed to derive the output video. Our solution performs in real-time on commodity hardware. Hereby, users can explore different options interactively, leading to a new level of expressiveness without having to rely on specialized hardware or laborious editing.*

**CCS Concepts**

• ***Computing methodologies*** → ***Computational photography;*** • ***Human-centered computing*** → *Interaction techniques;*

## 1. Introduction

In photography, the shutter controls when incoming light reaches the image sensor. Together with sensor sensitivity (ISO) and lens opening (aperture), the shutter speed defines the image exposure. In the first cameras, the shutter was a simple mechanic device manually moved in front of the lenses. Later, shutter devices with different shapes were created, from rotary-discs to blinds and diaphragms. Nowadays, most digital cameras implement an electronic shutter, which simply blocks or lets photons pass to an active sensor element.

In cinematography, many directors still use rotary-disc shutter devices for creative choices, typically turning synchronously to the 1/24th of a second frame time. The exposure of a frame is then controlled by an angular cut on the rotary-disc, while its spinning speed controls the video framerate. The ratio between the open and closed angles influences how sharp or smooth the scene motion is registered. A common practice is to use a 180° angle cutout, which is typically perceived as a natural motion blur by the audience [Hos18]. A faster shutter (smaller open angle) leads to crisp content and sharp motion. A good example is the movie *Saving Private Ryan*, which uses a 45° shutter to convey a frightening ambiance [Lei19]. Smaller angles mimic the effect of newsreels. Slower shutters (large open angle) result in motion blur, often used to give a sense of fast motion. Longer exposure can also smooth perceived motion, as used by David O. Russell's in *The Fighter* to subtly correct jittery movements [Des11]. Finally, ramping shutter speeds contributed to the energetic atmosphere of *Mad Max: Fury Road* [Pau16].

While controlling the shutter has been established as a useful measure to influence the perception of a video sequence and to enable different visual styles, the available options are rather limited. Even

mixing different motion patterns on the same take requires multiple cameras and compositing techniques. Our solution addresses this topic and we provide a novel technique to virtually simulate and combine different shutters in space and time. Unlike previous work, we let artists define spatio-temporally varying virtual shutters in a postprocess using a simple user interface. All results are presented in real time, which supports the shutter design process. A key advantage of our solution is that only sparse spatial and temporal annotations are needed, which then define varying per-pixel shutter functions for the entire video sequence. We demonstrate the reproduction of common real-world shutters and illustrate various options for artistic choices. In this context, we made the following technical contributions:

- an efficient shutter interpolation procedure;
- a technique to extend sparse shutter definitions; and
- a real-time interface for shutter design and compositing.

## 2. Related Work

The impact of shutter functions depends greatly on the motion in the captured scene, which can be spatially varying. When combining sharp and blurred objects in the scene, it is natural to consider matting to define target areas and to composite shots from different cameras. While recent approaches (e.g. [OLXK18,CGHD19,LL18]) can segment the masks, the process is still costly and challenging, as motion blur trails will jut out of the masks. Because of this, a single-frame motion blur requires inpainting to fill in the partially visible background underneath the motion trails [LSE19]. Consequently, such spatial mixing is difficult to obtain for real-world footage.

Glassner [Gla99] investigated the behavior of different shutter shapes on synthetic scenes, including a virtually-simulated "Slit Scan" shutter used in the iconic stargate scene from Stanley Kubrick's *2001: A Space Odyssey* [Beh15]. The slit scan effect is analogous to nowadays electronic camera sensor: the image is composed by partial sensor scans. The scan occurs line by line and in fractions of a second, i.e., while the sensor is exposed to light at a given shutter speed. If the speed is not as fast as the moving objects in the scene, the *rolling-shutter* effect is observed. While such effect is occasionally applied for artistic reasons, for many applications it is not desirable and spatial and angular warpings have been proposed to correct for image distortions [NFM07].

When the scene has little motion, the effects of varying shutters can be subtle, and their perception varies with different viewers' preferences and expertise [AWA*16]. Nevertheless, it has been shown that spatially varying exposure times can influence gaze motion [SBE*15]. Further, extreme examples, such as a stop-motion look, as seen in Cooper and Schoedsack's *King Kong*, is perceived as very unnatural. The effect is due to the complete lack of motion blur, as the ape model was recorded via still imagery. Interestingly, Brostow and Essa [BE01] proposed a solution to simulate fast shutter speeds and create a stop-motion look from blurred videos.

Postprocessed motion blur simulates a long exposure shutter by accumulating video frames. Such frame aggregation has been explored by tracking image features to backproject the motion onto a single background image [LWCT14] or by stabilizing the video on the focus object and averaging the moving pixels [LDG19]. In both cases, the blur effect is created from camera motion. Other approaches use optical flow and/or 3D motion vectors to globally track and create a per-pixel motion blur [RSM]. Spatially-varying blur requires masking and layer compositing via external software [RSM] or soft brushes [LDG19]. Nonetheless, the temporal motion blur strength is constant after each keyframe. Our method not only supports frame aggregation for motion-blur, but uses a general shutter function description that allows us to achieve many more effects (including rolling shutters, stuttered motion, or ghosting). In addition, we allow shutter functions to change both over space and time, and ensure seamless interpolation of these.

When processing high frame rate videos, another prominent context of shutter design is temporal filtering. Downsampling can be used for display on conventional screens [FCW*10] or to adapt framerate according to image content [TDMS16]. Nonetheless, the process has to be carefully applied, as framerate was shown to be more appreciated than resolution under budget constraints [DBS*18]. Disney's short movie Lucid Dreams of Gabriel [GS14] experiments with different spatio-temporal expressive effects to enhance storytelling. Our solution enables a wealth of temporal manipulations via its shutter-design interface.

## 3. Method

For our algorithm, we assume an input video that is *fully exposed*, meaning that the shutter is open during the whole frame time and no delay is induced from one frame to the next. While such input is not standard, modern devices and computational approaches enable the construction of such a sequence relatively easily. For standard videos, we prepare in-between frames using optical flow [WRHS13]. Unlike the remaining steps of our method, this preparation is an offline preprocessing step and we discuss the details in Appendix B.

Given the input video, users can interact in real-time to add simple shutter annotations in form of scribbles, defining regions that will be using the according shutter. Shutter definitions are interpolated over time and space using a diffusion mechanism, creating a shutter function for each pixel of the video. See Figure 2 for an illustration of the full pipeline.

In the following, we first explain the mathematical model to simulate and interpolate shutter functions (Sections 3.1 and 3.2). We then discuss the implemented user interface (Section 3.3) and how to provide sparse input to create shutter definitions for the entire video sequence. The latter is achieved via a diffusion process and an interpolation of the diffused shutter information. The efficiency of these two important steps is crucial for achieving real-time feedback and the details are described in Section 3.4.

### 3.1. Image formation with a Virtual Shutter

The traditional camera image-formation model defines a frame as the integration of the incoming light over time while the shutter is open. For a virtual shutter defined on a video with a fixed frame rate, time is discretized. The shutter function becomes a set of weights, one for each discrete quantum of time.

Formally, the input is a fully-exposed video $V$ consisting of $T \in \mathbb{N}$ frames, such that $V(t)$, with $t < T \in \mathbb{N}_0$, is the $t^{th}$ frame. For a

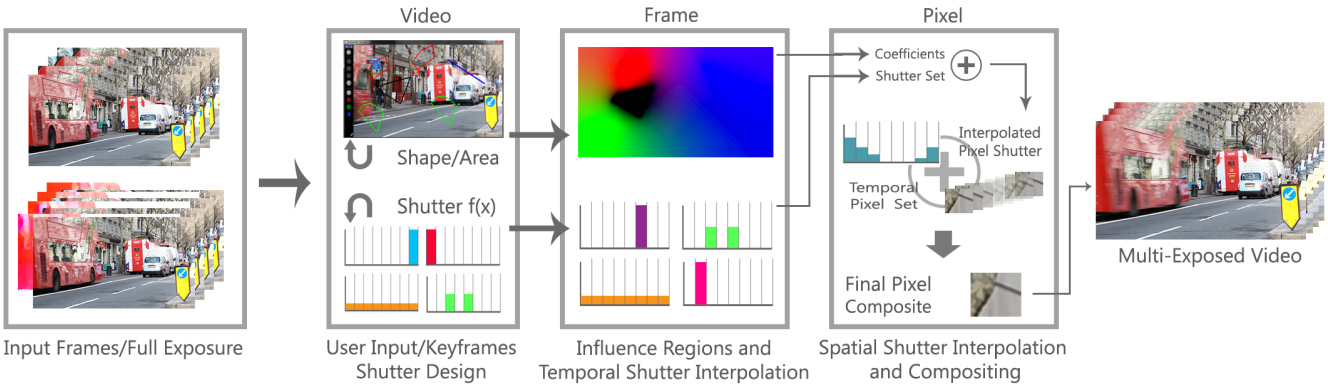


**Figure 2:** *Our system accepts an input video. If this video is not already fully exposed, the full exposure is created in a preprocessing step. The user can then design custom shutter functions, create shutter areas and define keyframes for temporal changes. The resulting multi-exposed video is immediately visible, as the back-end pipeline runs in real-time.*

frame $t$, a *shutter function* $s_t$ acts as a filter and attributes weights to $\mathcal{T}+1$ frames of the input video from frame $t$ onwards, leading to a *filtered frame* $F(t)$:

$$F(t) = \sum_{\tau=0}^{\mathcal{T}} s_t(\tau)\, V(t+\tau).$$

The simple case of reproducing the input video $O(t) = V(t)$ would define $s_t(\tau) = \delta_{0\tau}$, where $\delta_{ij}$ is the Kronecker delta (one if both indices match, otherwise zero). Figure 3 illustrates the result of different shutter functions for the same video input. As the frame rate of an output video $O$ does not have to match the frame rate of the input video $V$, a monotonic *frame-mapping function* $M : \mathbb{N}_0 \to \mathbb{N}_0$ can be used to define the $i^{th}$ output frame $O(i) = F(M(i))$. The video $O$ can then be played using the suitable frame time.

### 3.2. Shutter-function Interpolation

In our approach, shutter definitions will be specified on a per-pixel basis, thus enabling different shutters in different frame locations and at different times. To transition between the shutter definitions, we need to provide an interpolation among them. A trivial choice would be a linear value interpolation. Unfortunately, this solution does not result in a meaningful outcome.

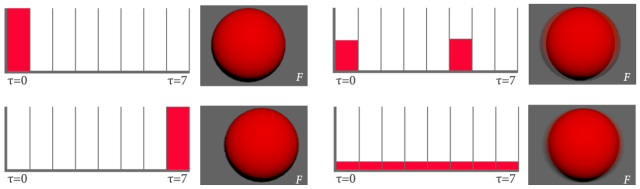


**Figure 3:** *Different shutter functions yield distinct results on the same input video of a horizontally moving sphere. Left: A shutter function equal to $\delta_{0\tau}$ reproduces the original video (top). By changing the location of the delta function, the video is time shifted (bottom). Right: a simulated shutter device with two cuts (top) results in a compositing of two frames, while a constant exposure (bottom) creates a motion-blurred result.*

Consider the shutter functions $s_1(\tau) = \delta_{0\tau}$ and $s_2(\tau) = \delta_{(15)\tau}$ (see Figure 4, top), which means that $s_2$ results in a frame that occurs 15 frames after the frame produced by $s_1$. The linearly interpolated shutter function (Figure 4, bottom right) would just blend both frames. For a natural transition, one would expect that intermediate frames between both time steps are obtained (Figure 4, bottom left). This can be achieved via displacement interpolation [BvdPPH11].

Displacement interpolation is typically applied to probability distribution functions and can be done by value interpolation of their inverse cumulative distribution functions [Rea99]. We will present our efficient implementation in Section 3.4.1.

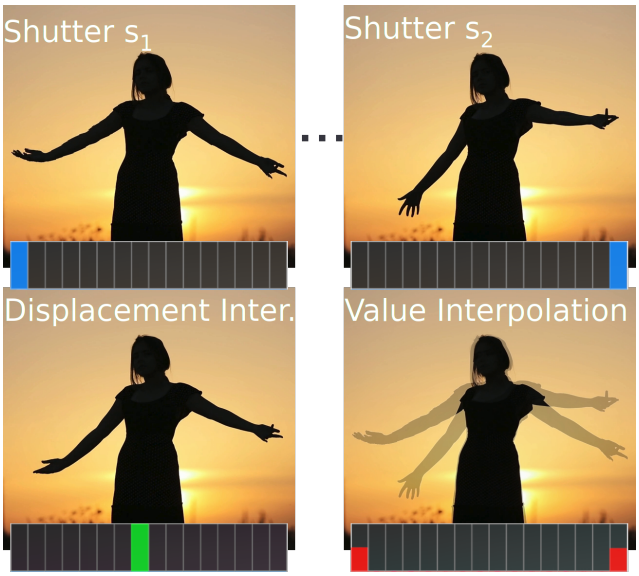


**Figure 4:** *Comparison of displacement and value interpolation. Interpolating the input shutter functions (displayed in the top row) at roughly 50-50 results in the images in the bottom row, depending on the interpolation method. Displacement interpolation (bottom left) smoothly shifts time across the transition, resulting in a sharp output image constructed from the in-between frames. Value interpolation of the shutters results in a shutter function that just mixes the input images (bottom right). Video source: pixabay.com.*

### 3.3. Interface

In order to define and apply different shutter functions to an input video, we need to indicate their regions of influence, both spatially and temporally. Our application provides a graphical interface (Figure 5) to support users in this task.

To apply shutter effects to the video, the first step is to define a default shutter function that will be applied to all frames, unless additional shutters are defined. To this extent, the user first chooses the number of shutter weight entries $\mathcal{T}$, which triggers a graph-bar editor. Here, the individual shutter-function values are set by dragging the corresponding bars up and down, representing values between zero or one. Alternatively, the user can choose among shutter functions from a predefined library.

To create an additional shutter, the user draws a scribble on a wanted frame. Again, the user defines a corresponding shutter function. The new shutter is then active for this frame and applied to the area covered by the scribble. To use the shutter over several frames, the user advances in the video and can place key frames. A keyframe enables the user to redefine the scribble (position, size, orientation) or the shutter function. For all intermediate frames between keyframes, the shutter is interpolated (meaning its shutter function, as well as the defining scribbles attributes). Once the shutter definition is complete, the procedure can be repeated.

For now, the shutter definitions would only be applied directly to the pixels underneath the scribble annotation. Instead, we actually want to extend the shutter definitions to the entire frame. To this extent, we first collect all active shutters at time $\tau$ and derive their interpolated representation from the user defined keyframes. Using their scribble annotations, a diffusion process is executed to find shutter interpolation weights for all pixels in the frame. We express this diffusion process as the solution to a heat transport problem, where the shutter annotations are used as local constraints, similar to the work of Orzan et al. [OBW*08]. In each pixel, the resulting interpolation weights define a shutter function that is used to process the input video. We rely on an efficient implementation, which we detail in Section 3.4.2, such that all steps can be executed in real-time. Hereby, the user has immediate feedback and can then adjust, delete or add new shutters until the desired result is obtained.

An illustration of the interface is shown in Figure 5, which contains the graph-bar edit of a shutter function, as well as scribble annotations for several shutters. Here, a shutter with a longer exposure time is applied to the wood block that is moving towards the characters, where a short exposure is used.

### 3.4. Implementation

To achieve real-time performance with high accuracy, our solution relies on suitable algorithmic choices and an efficient GPU implementation. The two main performance bottlenecks are the interpolation of shutter functions and the diffusion of the shutter annotations. While diffusion accelerations exist [GWL*05, JCW09], we opted for an alternative solver that is easy to implement, does not require geometric or curve primitives, and extends to more complex diffusion annotations [BEDT10] without sacrificing quality.


**Figure 5:** *Interface for interactively defining shutter functions and their influence regions. Users design shutter functions using the bar graph editor, whilst additional details, such as the number of frames in the function, can be changed using one of the collapsed fold-outs. Influence regions are defined by drawing scribbles using the corresponding color. The results are immediately visible. The sequence is from the Big Buck Bunny movie by the Blender Institute [Goe08].*

#### 3.4.1. Efficient Shutter-Function Interpolation

To describe the interpolation procedure, given $N$ shutter functions $s_i$, we define the total exposure $e(s_i) = \sum_{k=0}^{\mathcal{T}} s_i(k)$ and the normalized accumulation, denoted by a capital letter: $S_i(\tau) = \sum_{k=0}^{\tau} s_i(k)/e(s_i)$. Given the interpolation coefficients $c_i$ for shutter $s_i$ ($\sum c_i = 1$ and $c_i \geq 0$), we wish to find the interpolated shutter $q$. We make use of the observation that $q(\tau) = (Q(\tau+1) - Q(\tau))e(q)$, where $e(q) = \sum_{i=1}^{N} c_i e(s_i)$. Hence, having $Q$ allows us to find $q$. $Q$ is indirectly defined via its inverse, which is, in turn, given by a linear combination of the inverses of the accumulated shutter functions [Rea99]: $Q^{-1} = \sum_{i=1}^{N} c_i S_i^{-1}$. This relationship provides a solution to determine $q$ (Figure 6): first compute all $S_i$, invert them to derive $Q^{-1}$, then invert this function to find $Q$ and use it to determine $q$.

The functions $Q$ and $Q^{-1}$ depend on the per-pixel and per-frame coefficients $c_i$, requiring an efficient method for evaluating these functions in real-time. In the following, we describe our approach. To simplify, but without loss of generality, we will assume that for all shutter functions $e(s_i) = 1$.

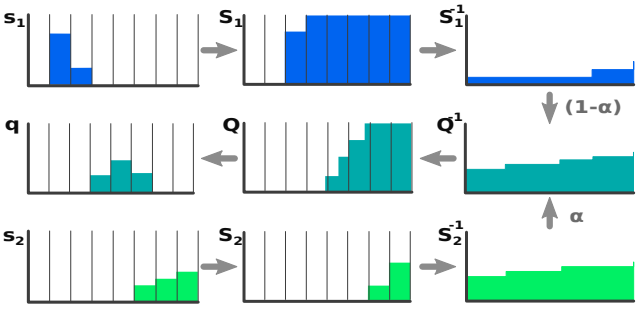


**Figure 6:** *The process of interpolating shutter functions $s_1$ and $s_2$ to produce a new shutter function $q$ requires several steps. First, the respective accumulation functions are derived ($S_1, S_2$). Next, these functions are inverted and then interpolated. The resulting function represents $Q^{-1}$, which is inverted so $q$ can be obtained by deriving and discretizing the representation.*

We consecutively determine the value for $q(\tau)$, with $\tau = 0 \dots \mathcal{T}$, retrieve each time the corresponding frame pixel, apply the weight and accumulate the result. In this way, the full function $q$ does not need to be stored in memory. As $q(\tau) = Q(\tau+1) - Q(\tau)$, we can instead solve for $Q(\tau)$, with $\tau = 0 \dots \mathcal{T}$ and only need the previous and current value of $Q$ in memory. For now, we will assume that we have access to $Q^{-1}$. If we localize $z$ such that $\tau = Q^{-1}(z)$, we have $z = Q(\tau)$. As $Q < 1$, we can deduce that $z \in [0,1]$ and as $Q^{-1}$ is monotonically increasing, we can employ a bisection method to solve for $z$. By default, we use a fixed number of nine search iterations on this interval, which, due to the interval search, yields an error of at most $2^{-10}$. This error is sufficiently small, as the precision loss of an 8 bit video is magnitudes larger. However, the iterations can be adapted for higher dynamic range content.

To complete the calculation of the values of $q$, we still need to be able to compute $Q^{-1}(z)$ during the bisection method. We recall $Q^{-1} = \sum_{i=1}^{N} c_i S_i^{-1}$, which thus means that we need to invert $S_i$. We can again solve an equation of the form $z = S_i(\tau)$. Instead of a bisection, a binary search over discrete elements is more suitable, since $S_i$ is efficiently represented by an array of $\mathcal{T}$ elements.

For numerical robustness, we consider the function $S_i$ to be piecewise linear. In fact, this reflects that the values of $S_i$ stem from an integration of constant exposure over the frame time. Based on this, we first search for the first element $w+1$, such that $S(w+1) > z$. The value $S(w)$ is guaranteed to be smaller or equal to $z$. We then compute the refined result $\tau = w + (z - S(w))/(S(w+1) - S(w))$, linearly interpolating $w$ and $w+1$ based on $z$.

To reduce the average cost of the searches, we successively shrink the search space. In each iteration $k$ of the bisection method, a $z_k$ will be updated to a new location $z_{k+1}$. Assume in iteration $k$, we found the set of $w_{k,i}$ in the binary searches. Due to monotonicity, if $z_{k+1} \geq z_k$, then each $w_{k+1,i} \geq w_{k,i}$ and if $z_{k+1} < z_k$, then $w_{k+1,i} \leq w_{k,i}$. Hereby, the search space for the next $w_{k+1,i}$ shrinks.

A similar optimization can be applied when evaluating consecutive values of $Q$. We can restrict the lower bound of $w_{0,i}$ based on the previous values after convergence because $Q(\tau) \leq Q(\tau+1)$. In principle, a last option exists to shrink the interval during the bisection but this case turned out to be inefficient because we already apply the method with only a few steps. A pseudo-code of our efficient shutter interpolation implementation is presented in Appendix A.

### 3.4.2. Efficient Shutter Diffusion

To extend the shutter annotation scribbles to the entire frame, we rely on a diffusion process. Similar to Orzan et al. [OBW*08], we express the diffusion as a heat transport problem, where user annotations are local constraints. We follow their approach and rely on a multi-grid solver but perform a customized downsampling to achieve a high quality diffusion only from the pixel image, without resorting to geometric primitives.

Specifically, we use two images, a mask image identifying the constraint locations, and an image defining the values of the constraints at those locations. We reduce the problem size by consecutively halving the resolution until we reach a size of $2 \times 2$. For the smallest image, the solution can be solved immediately and it is then repeatedly upsampled and recombined with the constraints at the next resolution level, while applying a small number of diffusion steps (Jacobi iterations). This process is repeated until we reach the original image size.

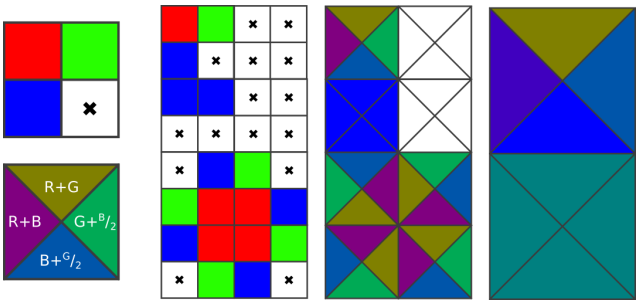


**Figure 7:** *On the left, we show the diffusion influences arising from the $2 \times 2$ pixel block containing a red, a green and a blue constraint as well as one constraint-free location. On the right, we illustrate the downsampling of influence regions. In particular, the four fully-enclosed red pixels do not affect the outside at all, and their contribution disappears after downsampling once.*

To faithfully maintain the constraints during downsampling, we analyze each $2 \times 2$-pixel block before collapsing and actually store four values representing an approximation of the accumulated values that are emitted into the four axis-directions. We refer to the four values along the axis as an *influence block*. The first influence blocks are formed by $2 \times 2$-pixel groups (Figure 7, left). Starting from an edge of this block, if both adjacent pixels are filled, they both contribute evenly and we average their values. If one pixel is empty but the next behind is filled, the closer one contributes with a weight of one, the farther one with a weight of 0.5. If there are no constraints in the nearby pixels but both farther pixels contain constraints, then they contribute evenly. If there is only one pixel, it contributes alone. If the block contains no constraints, it does not emit anything.

In the following steps, $2 \times 2$ groups of influence blocks are combined safely by maintaining their outward influences and discarding interior ones (Figure 7, right). We employ the same weighting scheme as in the initial step. However, instead of relying on the single color value at each location in the block, we fetch the location's influence in the currently-considered direction.

During the diffusion process, we naively upsample without any filtering, replicating pixel values across whole blocks. Upsampling with linear filtering would introduce additional complexity due to having to consider surrounding constraints. We are able to limit the number of Jacobi iterations performed at each level significantly: we perform six iterations for sizes up to $64 \times 64$, two iterations for higher resolutions. The Jacobi iterations must consider both constraint influences (if present) and the diffused values. Intermediate images storing diffused values use 16 bits of precision per channel, as an 8 bit color depth results in small gradients vanishing very early. Figure 8 illustrates a set of user constraints along with diffused results. We compute the results for up to four color channels (RGBA) simultaneously; constraint locations are stored in a separate binary map. Note that constraints in close proximity remain properly separated without bleeding into each other.

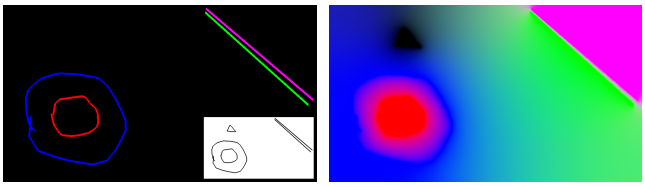

**Figure 8:** *The input image (left) shows the user-drawn scribbles that create the constraints. The mask (white inset at the bottom) defines the locations of the constraints. Note the zero-valued (=black) constraint in the top-left of the input. The result of the diffusion process (right) form the per-pixel coefficients for the shutter interpolation. Importantly, the red influence is fully contained by the outer blue constraint, and the green and pink regions stay well separated.*

## 4. Results

We implemented the described system as a desktop software that enables users real-time editing and exploration of different shutter functions in various video clips. All our experiments were performed on a standard desktop system running Windows 7, with a Intel i7 3820 CPU, 16GB of RAM, and a NVIDIA GTX 1080 GPU with 8GB of VRAM. Our input videos were acquired in-house (using a Samsung Galaxy S8, recording at 120-240Hz), unless a different source is noted in the image captions (typically recorded at 24-30Hz). While the usage of high frame rate videos is a desirable choice to improve the physical accuracy of the time integration, this is not a requirement. Hence, the input frame rates vary depending on the video source. The output frame rate is determined from the input video and user selections, such as the size of the shutter function.

We can achieve a number of different effects with our system. For example, an object or region can be highlighted by applying a short shutter, producing a sharp output, while the rest of the image can use a long shutter that results in motion blur. Figure 1 employs this effect to keep the text on the moving pendulum readable and attract attention to it. Figure 9 shows an example where the movement of one hand is made less visible, to focus on the precise movements of the right hand. For these effects, the user draws annotations on the image around the areas of interest; the results are shown in real-time.

The effect can be extended to vary over time. In the video sequences show in Figure 10, keyframes are used to change the shutter functions and areas over time. At different moments, a single moving individual is made sharp to produce a contrast against the background, which is abstracted using a long exposure. The sharp areas are further keyframed to follow the subjects. Before switching to a new person, the shutter function fades out to smoothly merge the previously highlighted person into the crowd, illustrating the advantage of our interpolation method. The effect can be reversed, and attention can be removed from a person instead.

In Figure 11 we demonstrate a time-varying rolling shutter. As the car drives past, the direction of the rolling shutter switches, causing the skewing arising from the rolling shutter to reverse. We realize the rolling shutter in our framework by defining two shutter functions at the top and bottom of the image, selecting the first and last frames inside the shutter function's window, respectively. The spatial interpolation ensures that the shutter function shifts smoothly across the window size. Another artistic effect is illustrated

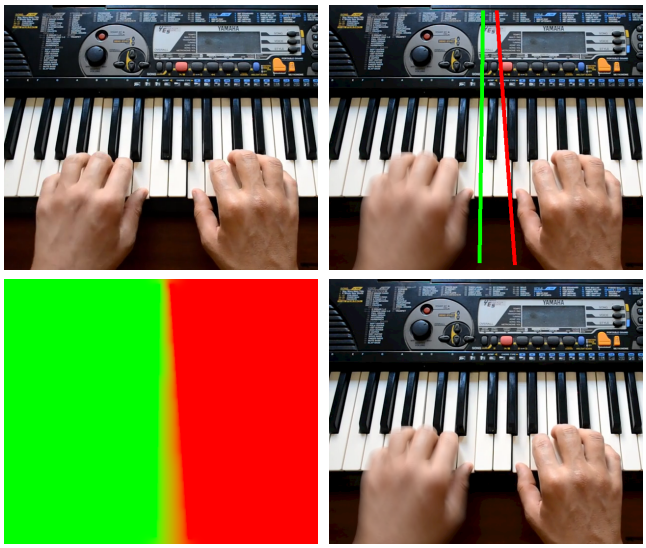

**Figure 9:** *Top: Original frame (left) and user-drawn annotations (right). Bottom: Diffused per-pixel coefficients and the resulting frame after applying the shutter functions. Note the blur on the left hand contrasting with the sharp motion on the right side. Video source: pixabay.com at 25Hz. Rendering time (avg): 0.59ms in 960×540.*

in Figure 12, where the light trails were composited to resemble a light painting while keeping the person sharp.

We list the per-frame total rendering time for all results in their respective figure captions. Total time measures wall-time, and thus includes overheads from other processing. For the shown results, we defined two shutter functions of varying size. The number of shapes and keyframes vary across examples.

To isolate performance on the two core steps of our approach (Diffusion and Interpolation), we specified a set of shapes (two lines and two lassos) and up to four shutters. The results are evaluated in $1920 \times 1080$ (full HD) and $3840 \times 2160$ (4K) resolution and presented in Table 1, under *Ours (full resolution)*. For completeness, times for intermediate steps (shape and shutter interpolation for key frames (CPU) and shape rasterization) are aggregated and listed as *Other*. Since the diffusion process is image-based, performance only depends on resolution. In addition to different resolutions, we explore the interpolation of different numbers of shutter functions ($N$) and different lengths ($\mathcal{T}$). We compare our method to a simplified interpolation implementation (see column *Naive Histogram Interpolation*), i.e. one without our search-space optimizations.

The shutter interpolation step is further sped-up by aggregating coefficients. Here, we rasterize shapes and compute diffusion in half resolution, giving each $2 \times 2$ block of pixels the same coefficients for interpolation. We perform the interpolation only once for such a block, which significantly reduces the computation time for the interpolation step (see column *Ours w/ aggregation (2x2)* in Table 1). We thereby achieve real-time performance even at 4K resolution, while minimally impacting image quality: we obtained SSIM [WBS*04] differences of 0.9982 (average) and 0.9777 (worst) compared to the full-resolution results (this difference is considered high quality for video compression [LB15]).

**Figure 10:** *A motion blur shutter can be applied selectively to different regions of the image. Here, the keyframed regions track a person in the images. In the left images, we keep one moving person sharp over a short time span, while blurring the rest. This highlights and draws attention to the person. On the right, we do the opposite, and remove attention from the person crossing the hall. Left video source: pixabay.com at 25Hz. Rendering time (avg): 1.93ms in 960×540. Right image sequence captured in house at 240Hz. Rendering time (avg): 1.58ms in 1280×720.*

**Table 1:** *Run-time performance at full HD and 4K.* Diffusion *refers to our shutter diffusion GPU process, and* Other *encompasses various CPU processing steps such as the shape and shutter interpolation for keyframing. GPU-based spatial shutter interpolation and image composition times are listed under* Shutter Interp. & Comp.*, and are measured for different numbers of shutter functions (N) and sizes (𝒯).*

| | | | **Naive Histogram Interpolation** | | | | **Ours (full resolution)** | | | | **Ours w/ aggregation** ($2 \times 2$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **full HD** | Diffusion | | – | | | | *0.88 ms* | | | | *0.35 ms* | | | |
| | Other | | – | | | | *0.42 ms* | | | | *0.39 ms* | | | |
| | Shutter Interp. & Comp. | $N$ \ $\mathcal{T}$ | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 |
| | | 2 | *5.69 ms* | *12.85 ms* | *28.69 ms* | *64.80 ms* | *3.50 ms* | *6.88 ms* | *13.85 ms* | *28.10 ms* | *0.92 ms* | *1.79 ms* | *3.57 ms* | *7.31 ms* |
| | | 3 | *8.05 ms* | *17.94 ms* | *40.07 ms* | *88.84 ms* | *4.91 ms* | *9.87 ms* | *20.33 ms* | *42.05 ms* | *1.35 ms* | *2.71 ms* | *5.54 ms* | *11.61 ms* |
| | | 4 | *10.40 ms* | *23.20 ms* | *52.26 ms* | *117.35 ms* | *6.48 ms* | *12.95 ms* | *27.49 ms* | *57.13 ms* | *1.74 ms* | *3.46 ms* | *7.27 ms* | *15.01 ms* |
| **4K** | Diffusion | | – | | | | *3.04 ms* | | | | *0.90 ms* | | | |
| | Other | | – | | | | *0.57 ms* | | | | *0.44 ms* | | | |
| | Shutter Interp. & Comp. | $N$ \ $\mathcal{T}$ | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 |
| | | 2 | *22.86 ms* | *51.01 ms* | *107.82 ms* | *244.04 ms* | *14.04 ms* | *27.22 ms* | *54.47 ms* | *111.13 ms* | *3.59 ms* | *6.96 ms* | *13.77 ms* | *28.37 ms* |
| | | 3 | *31.76 ms* | *72.04 ms* | *154.93 ms* | *365.18 ms* | *19.78 ms* | *39.07 ms* | *80.15 ms* | *167.51 ms* | *5.36 ms* | *10.59 ms* | *21.50 ms* | *44.83 ms* |
| | | 4 | *39.39 ms* | *88.53 ms* | *203.04 ms* | *456.94 ms* | *25.32 ms* | *51.37 ms* | *108.50 ms* | *226.09 ms* | *6.83 ms* | *13.68 ms* | *28.36 ms* | *58.46 ms* |

**Figure 11:** *Rolling shutters are realized by defining two time-shifted shutter functions and interpolating between these. Shutter spans 64 frames. By keyframing the shutter functions, we can cause the effect to flip midway, reversing the skewing as the car drives past. Black squares added for anonymization. Video captured in house at 240Hz. Rendering time (avg): 2.24ms in 1280×720.*

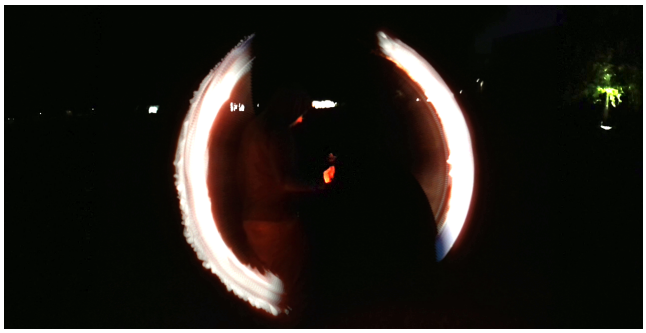

**Figure 12:** *Light trails created with a long exposure shutter (spanning 0.5 seconds = 120 frames). The remainder of the background and the person holding the fire pole are kept sharp. Video captured in house at 240Hz. Rendering time (avg): 8.71ms in 1280×720.*

## 5. Conclusion

We presented a novel method to influence the shutter functions of a video in a postprocess as well temporally, as also spatially. We propose a real-time interface, which allows artists to preview their modifications. It is possible to produce a large variety of results with little user interaction, making it possible to explore many alternatives before settling on the desired effect. The quick feedback being key, we propose optimized algorithms to achieve spatial interpolation of the shutter definitions. Our solution enables even novice users

to explore many different opportunities to enhance, stylize and also impact the gaze of observers. ShutterApp is a step forward in shifting typical settings that need to be defined during recording to a postprocess and even offers many more possibilities beyond standard shutter systems.


### Acknowledgments

The authors would like to thank pixabay.com for the CC0 videos used in this paper, and Michael Stengel for the discussion on designing shutter functions. This work was partially funded by the Brazilian agency CNPq (Process No 202551/2015-6), the Swiss National Science Foundation (Advanced Postdoc Mobility Project 174321), and DyViTo/European Union's Horizon 2020 programme (Grant Agreement No 765121).


### References


[AWA*16] ALLISON R. S., WILCOX L. M., ANTHONY R. C., HELLIKER J., DUNK B.: Expert viewers' preferences for higher frame rate 3d film. *Journal of Imaging Science and Technology 60*, 6 (2016). 2

[BE01] BROSTOW G. J., ESSA I.: Image-based motion blur for stop motion animation. *Proc. ACM SIGGRAPH* (2001). 2

[BEDT10] BEZERRA H., EISEMANN E., DECARLO D., THOLLOT J.: Controllable diffusion curves. In *INRIA Technical Report* (2010). 4

[Beh15] BEHIND THE NEWS: How did Douglas Trumbull make the Stargate sequence in 2001 A Space Odyssey? https://youtu.be/P1gn06np-7g, 2015. Accessed: 2019-07-30. 2

[BvdPPH11] BONNEEL N., VAN DE PANNE M., PARIS S., HEIDRICH W.: Displacement interpolation using lagrangian mass transport. *ACM Transactions on Graphics 30*, 6 (2011). 3

[CGHD19] CHEN X., GIRSHICK R., HE K., DOLLÁR P.: Tensormask: A foundation for dense object segmentation. *arXiv preprint arXiv:1903.12174* (2019). 2

[DBS*18] DEBATTISTA K., BUGEJA K., SPINA S., BASHFORD-ROGERS T., HULUSIC V.: Frame rate vs resolution: A subjective evaluation of spatiotemporal perceived quality under varying computational budgets. In *Computer Graphics Forum* (2018), vol. 37. 2

[Des11] DESOWITZ B.: Keeping 'The Fighter' Authentic. https://bit.ly/2JyQrGo, 2011. Accessed: 2019-07-30. 1

[FCW*10] FUCHS M., CHEN T., WANG O., RASKAR R., SEIDEL H. P., LENSCH H. P. A.: Real-time temporal shaping of high-speed video streams. *Computers and Graphics 34*, 5 (2010). 2

[Gla99] GLASSNER A.: An open and shut case. *IEEE Computer Graphics and Applications 19*, 3 (1999). 2

[Goe08] GOEDEGEBURE S.: Big Buck Bunny. Short Movie, 2008. 4

[GS14] GROSS M., SCHRIBER S. A.: Lucid dreams of gabriel. Short Movie, 2014. 2

[GWL*05] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D., HUMPHREYS G.: A multigrid solver for boundary value problems using programmable graphics hardware. In *SIGGRAPH Courses* (2005). 4

[HKML15] HYUN KIM T., MU LEE K.: Generalized video deblurring for dynamic scenes. In *IEEE Conference on Computer Vision and Pattern Recognition* (2015). 9

[Hos18] HOSER T.: *Introduction to Cinematography: Learning Through Practice*. Taylor & Francis, 2018. 1

[IMS*17] ILG E., MAYER N., SAIKIA T., KEUPER M., DOSOVITSKIY A., BROX T.: Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition* (2017). 9

[JCW09] JESCHKE S., CLINE D., WONKA P.: A GPU laplacian solver for diffusion curves and poisson image editing. *ACM Transactions on Graphics 28*, 5 (2009). 4

[JSJ*18] JIANG H., SUN D., JAMPANI V., YANG M.-H., LEARNED-MILLER E., KAUTZ J.: Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *IEEE Conference on Computer Vision and Pattern Recognition* (2018). 9

[KTDvG16] KROEGER T., TIMOFTE R., DAI D., VAN GOOL L.: Fast optical flow using dense inverse search. In *European Conference on Computer Vision* (2016). 9

[LB15] LOH W., BONG D. B. L.: Video quality assessment method: Md-ssim. In *IEEE Intl. Conference on Consumer Electronics* (2015). 6

[LDG19] LANCELLE M., DOGAN P., GROSS M.: Controlling motion blur in synthetic long time exposures. In *Computer Graphics Forum (Eurographics)* (2019), vol. 38. 2

[Lei19] LEIRPOLL J.: Debunking the 180-degree shutter rule. https://bit.ly/2Gr2Jif, 2019. Accessed: 2019-07-30. 1

[LL18] LI X., LOY C. C.: Video object segmentation with joint re-identification and attention-aware mask propagation. *DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (2018). 2

[LSE19] LUO X., SALAMON N. Z., EISEMANN E.: Controllable motion-blur effects in still images. *Transactions on Visualization and Computer Graphics* (2019). 2

[LWCT14] LIU S., WANG J., CHO S., TAN P.: Trackcam: 3d-aware tracking shots from consumer video. *ACM Transactions on Graphics 33*, 6 (2014). 2

[NFM07] NICKLIN S. P., FISHER R. D., MIDDLETON R. H.: Rolling Shutter Image Compensation. In *RoboCup 2006: Robot Soccer World Cup X*, vol. 4434 LNAI. Springer, 2007. 2

[OBW*08] ORZAN A., BOUSSEAU A., WINNEMÖLLER H., BARLA P., THOLLOT J., SALESIN D.: Diffusion curves: a vector representation for smooth-shaded images. In *ACM Transactions on Graphics* (2008). 4, 5

[OLXK18] OH S. W., LEE J., XU N., KIM S. J.: Fast user-guided video object segmentation by deep networks. *DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (2018). 2

[Pau16] PAUL J.: Capture Intense Cinematic Action With High Shutter Speed. https://bit.ly/2JBh3qd, 2016. Accessed: 2019-07-30. 1

[Rea99] READ A. L.: Linear interpolation of histograms. *Nuclear Instruments and Methods in Physics Research A425* (1999). 3, 4

[RPCY17] REN W., PAN J., CAO X., YANG M.-H.: Video deblurring via semantic segmentation and pixel-wise non-linear kernel. In *IEEE International Conference on Computer Vision* (2017). 9

[RSM] ReelSmart Motion Blur. http://revisionfx.com/products/rsmb/. Accessed: 2019-05-04. 2

[SBE*15] STENGEL M., BAUSZAT P., EISEMANN M., EISEMANN E., MAGNOR M.: Temporal video filtering and exposure control for perceptual motion blur. *IEEE Transactions on Visualization and Computer Graphics 21*, 5 (2015). 2

[TDMS16] TEMPLIN K., DIDYK P., MYSZKOWSKI K., SEIDEL H.-P.: Emulating displays with continuously varying frame rates. *ACM Transactions on Graphics 35*, 4 (2016). 2

[TSY*07] TELLEEN J., SULLIVAN A., YEE J., WANG O., GUNAWARDANE P., COLLINS I., DAVIS J.: Synthetic shutter speed imaging. *Computer Graphics Forum 26*, 3 (2007). 9

[WBS*04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P., ET AL.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004). 6

[WRHS13] WEINZAEPFEL P., REVAUD J., HARCHAOUI Z., SCHMID C.: Deepflow: Large displacement optical flow with deep matching. In *IEEE International Conference on Computer Vision* (2013). 2, 9

## Appendix A: Shutter Interpolation Method Implementation

Pseudo code for evaluation of the functions $Q$, $Q^{-1}$ and $S_i^{-1}$, introduced in Section 3.4.1. We use the same notation and names for variables and functions here and in the text. Arrays of size $N$ (the number of shutter functions) are denoted with a `[]`-postfix. Values relating to the search space optimization are highlighted in color. Our implementation placed the values of $S_i$ in shared memory for better random-access performance.

```
Q( τ, c[], searchBoundsLo[] )
    -> (float, uint[])
{
    // Define the search bounds. Lower bounds are based on
      previous evaluations of Q. Upper bounds always start at
      the maximum value (T+1). The bounds are updated each
      iteration of the bisection (below).
    los = searchBoundsLo[];
    his = [T+1, ..., T+1];

    // Bisection. The bisection performs a fixed number of
      steps, halving the remaining search interval each iteration.
      After 9 steps ⇒ dz = 2^-11, and the "true" |z* − z| ≤ 2dz.
    bounds = [];
    z = 0.5f, dz = 0.25f;

    for(j = 0; j < SEARCH_ITERATIONS; ++j)
    |   (val, bounds) = Q^-1( z, c, los, his );
    |
    |   if( val ≤ τ )
    |   |   z += dz;
    |   |   los = bounds; // Adj. lower search bounds
    |   else
    |   |   z -= dz;
    |   |   his = bounds; // Adj. upper search bounds
    |
    |   dz *= .5f;

    // Return result and indices (bounds) at which the value
      was found. The next call to Q (with τ ≥ the current τ) will
      receive the returned bounds as searchBoundsLo.
    return (z, bounds);
}


Q^-1( z, c[], searchBoundsLo[], searchBoundsHi[] )
    -> (float, uint[])
{
    res = 0.f;
    bounds = [];

    // Q^-1 = Σ_{i=0}^{N} c_i S_i^-1. Evaluate each S_i^-1 in turn, using the
      restricted search space. In addition, propagate the indices
      from the search in order to update the search space.
    for( i = 0; i < N; ++i )
    |   (t,k) = S_i^-1( z, searchBoundsLo[i],
    |       searchBoundsHi[i] );
    |
    |   res += c[i]*t;
    |   bounds[i] = k;

    return (res, bounds);
}
```

```
S_i^-1( z, searchBoundsLo, searchBoundsHi )
    -> (float,uint)
{
    // Binary search. Find the index of the first element
      greater than or equal to z, The search is restricted to the
      range [searchBoundLo, searchBoundHi]. Compare to, e.g.,
      the C++ standard function std::upper_bound.
    lo = searchBoundsLo;
    hi = searchBoundsHi;

    while( lo < hi )
    |   mid = (lo+hi)/2;
    |   if( !(z < S_i[mid]) )   lo = mid+1;
    |   else                    hi = mid;

    // Linear interpolation. Interpolate between the found value
      (≥ z) and the previous value (< z). If the search converges
      to 0, return 0.f (avoid out-of-bounds accesses).
    if( lo == 0 )  return (0.f, 0);

    vhi = S_i[lo];
    vlo = S_i[lo-1];
    vinterp = (lo-1) + (z-vlo)/(vhi-vlo);

    // Return the linearly interpolated value and the index at
      which it was found. The latter is used to restrict the search
      space in future searches.
    return (vinterp, lo);
}
```

Usage (e.g., in the body of a fragment or compute shader):

```
// ... (initialization, etc.) ...
boundsLo = [0, ..., 0];
Q_τ = Q_τ-1 = 0.f;
for( τ = 1; τ ≤ T; ++τ )
|   Q_τ-1 = Q_τ;
|   (Q_τ,boundsLo) = Q( τ/T, c, boundsLo );
|   contrib_τ = (Q_τ-Q_τ-1) * texelFetch( ... ).rgb;
|   // ... (use/accumulate contrib_τ) ...
// ... (finalize and output) ...
```

## Appendix B: Full Temporal Exposure

We assume as input a video with full temporal exposure, i.e., where the shutter of the recording camera does not close. Standard videos will not always fulfill this requirement, e.g., a 180° shutter only records half of the frame time. Nevertheless, digitally recorded high framerate videos often come close.

Furthermore, if the sequence consists of short exposed frames, one can also rely on optical flow [WRHS13, IMS*17, KTDvG16] from neighboring frames to form virtual intermediate images that fill the gaps. Telleen et al. [TSY*07] compute the in-between frames by aligning images and calculating pixel movement to simulate different photo exposures. Jiang et al. [JSJ*18] train and employ an encoder-decoder network that can create a virtually unlimited number of plausible intermediate frames.

For longer exposures, motion blur can occur. Here, a direct interpolation is insufficient to produce frames with a full temporal exposure, as the time intervals of the frames might overlap after adding the intermediate frames. A remedy can be deblurring technique [RPCY17, HKML15] that can transform the long exposure frame into an approximate short exposure.