The Visual Computer
https://doi.org/10.1007/s00371-022-02545-8

ORIGINAL ARTICLE

Preemptive text warping to prevent appearance of motion blur

Zixun Yu[1] · **Manuel M. Oliveira**[2] · **Daniel G. Aliaga**[1]

Accepted: 18 May 2022

Abstract
We present a preemptive image-based method to reduce motion blurring. Motion blur appears when there is a relative motion between the scene and the viewer/camera. A preemptive method pre-filters the content before being displayed in order to mitigate the occurrence of motion blur. Our experiments and user study have shown that such preemptive methods are fundamentally subject to producing visual artifacts as a consequence of unavoidable ringing or intensity oscillations. Frequency-domain analysis shows that the energy weakened at certain frequencies leads to those artifacts. We present a method to process alphanumeric content so that it has lower energy on frequencies eliminated by a given motion blur kernel. Our processed image, when motion blurred, will have a sharper appearance and less artifacts as compared to various alternative approaches. We demonstrate the effectiveness of the proposed technique with simulated and real-world experiments as well as user feedback. Our results show that our approach yields content robust to motion blur while still being perceptually similar to the original text.

Keywords Image processing · Motion deblurring · Visual system

1 Introduction

Motion blur is a fundamental vision phenomena that occurs when an object is moving with respect to a sensor during some finite exposure time. While the amount of motion blur can be reduced by decreasing exposure time or reducing the relative speed between the target object and the sensor, it causes unavoidable difficulties in recognizing text and objects and hinders computer vision as well. The occurrence and hindrance of motion blur is well studied [14,28]. Motion blur can be modeled as:

$$I_p = O(M(D(I))), \tag{1}$$

where I is the displayed image, D can be either a display device or some printed version of I, M models the emitted light moving over a period of time, and O models the observer (either a human or a camera), resulting in the final perceived/captured image I_p, where ideally $I_p = I$.

Various prior methods attempt to mitigate the perception of motion blur. The most common scenario are post-processing solutions, $R(I_p) = R(O(M(D(I))))$, where R is a digital deblurring algorithm (e.g., coded exposure photography [28], Lucy–Richardson deconvolution [21], Wiener filtering [30], DeblurGAN [17]). However, these approaches are fundamentally not possible for human observers, since the post-processing cannot be added to the human visual system, and its not always possible to obtain an $R(I_p) = I$. Therefore, while reducing motion speed, reducing shutter time (only in the case of cameras) or using high refresh rate displays ameliorates motion blur none of these methods prevent (or eliminate) motion blur.

Our main observation is that one can pre-filter a target image to compensate for motion blur so that upon the occurrence of motion blur such content appears significantly blur-free and sharp in the ideal case (Fig. 1); in other words, our approach is preemptive:

$$I \approx W(I) = O(M(D(P(W(I))))). \tag{2}$$

Our preemptive method $P(W(\cdot))$ alters the content I so that upon going through the display, motion integration,

✉ Zixun Yu
yu645@purdue.edu

Manuel M. Oliveira
oliveira@inf.ufrgs.br

Daniel G. Aliaga
aliaga@cs.purdue.edu

[1] Department of Computer Science, Purdue University, West Lafayette, IN, USA

[2] Instituto de Informática, UFRGS, Porto Alegre, RS, Brazil

Published online: 28 June 2022

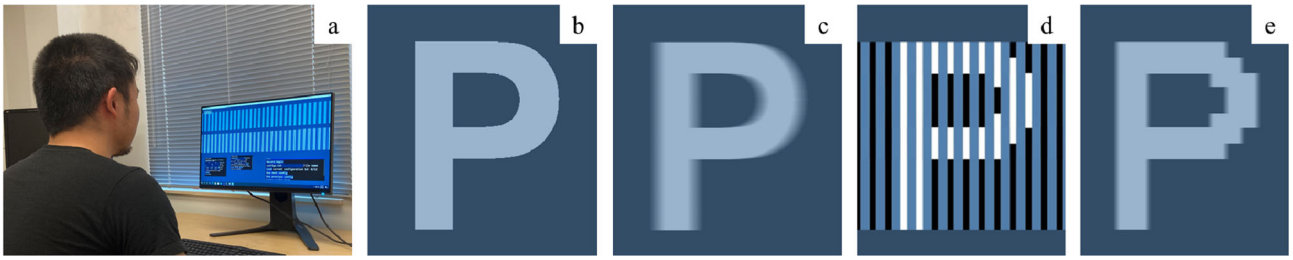

Fig. 1 *Preemptive text warping on displays for HVS*; **a** scenario where a user looks at the display (FHD 25" and 60cm distance) and the content on the screen moves horizontally. **b** Original content displayed, a letter "P" taking about 1/4 of the vertical space on the screen. **c** Simulated blurred result when image **b** moves 24px per frame. **d** Alternate image using our method. **e** Simulated result when image **d** moves at the same speed, but appearing sharper than **c**

and observer steps, it appears sharper and blur-free to the observer. Since the method is preemptive and done on the image content itself, our solution is applicable to both human and camera-based observers. This is unlike prior methods which attempt to remove/minimize blur after the fact and/or only improve the display D to reduce motion blur [1].

Our solution can be used in a variety of applications. For example, our user study shows that motion blur, perceived on high-quality and high-frame rate digital displays, can be significantly reduced by our method (see Sect. 7). Therefore, our method can make use of on-board inertial sensors to improve personalized displays (e.g., phones, tablets, laptops) undergoing displacements (e.g., moving or shaking). Further, digital signs and displays can adjust the content with our technique to a measured relative motion (e.g., by using the speed of the vehicle). The amount of blur can be determined in advance, or in real time, so our method can be applied during viewing.

Our preemptive approach to motion blur consists of three main steps. First, we model the motion blur to be induced by scene or camera motion parallel to the image plane. As in previous works [6,14,19,28], we model such motion blur as a convolution with a box kernel whose point spread function (PSF) is proportional to the speed of motion. Second, we alter the shape of a provided alphanumeric content so that it does not contain frequencies discarded by the convolution with the given box filter. While this can be accomplished in the frequency or spatial domain, it is possible to interpolate between the original alphanumeric shapes and the preemptive alphanumeric shapes in the spatial domain. Thus, we pursue a spatial domain-based modification enabling a trade-off between motion blur tolerance and content originality. Third, the altered alphanumeric content is given to a pre-filtering engine [23] that generates a pattern designed to improve contrast under a given PSF (in our case, motion blur). Normally, such pre-filtering produces notorious visual artifacts as a consequence of unavoidable ringing or intensity oscillations. However, since our altered alphanumeric content does not contain the frequencies discarded by the prescribed amount of motion blur, the end result is significantly free of these visual artifacts. Thus, the appearance of the alphanumeric content is much cleaner.

We demonstrate the effectiveness of our approach in simulated and real-world experiments as well as in a user study. Moreover, we compare the performance of our method to various alternatives, and show it better preserves contrast and sharpness of the modified alphanumeric content.

The main **contributions** of our work include:

- A preemptive text warping algorithm that yields content void of the frequencies discarded by a specified motion blur (Sect. 4). Our algorithm can handle most text/symbols provided in raster or vector format;
- A geometry-based method to interpolate between ideal blur-tolerant appearance and initial appearance (Sect. 4.3), thus enabling a quality/blur trade-off.

2 Related work

Deconvolution and Deblurring Methods The use of deconvolution to reduce/remove motion blur is a well studied problem [3], with techniques exploring both blind [5] and non-blind deconvolution [10,12]. However, image deconvolution is fundamentally a post-processing method, and moreover, it is an ill-posed problem sensitive to small perturbations in the input data [34]. In the case of motion blur, the associated PSFs discard spatial high-frequency details, making it generally impossible to recover the original signal. More recently, researchers have used deep learning techniques to perform deblurring [17] and resolve details [36]. In other work, researchers have also tried to find blur in an image and then determine whether the blur can and should be deblurred [38].

Computational Cameras Several approaches have been developed that alter the camera capture process or the camera. Raskar et al. [28] proposed the use of a coded exposure technique to preserve high-frequency content in the blurred

images, thus improving deconvolution results. Fortunato and Oliveira [11] presented a coded aperture solution based on factorable masks that preserves high-frequency content, allowing the recovery of depth and color from a single photograph using inverse filtering. Levin et al. [19] and Cho et al. [6] describe how to facilitate blur deconvolution by instantaneously moving the camera during capture. All these techniques, however, are also post-processing. In contrast, our technique preemptively modifies the input so that its high-frequency spatial content is not (significantly) affected by motion blur.

Computational Displays Prior work has proposed custom programmable and multilayer displays to compensate for the viewer's optical limitations (e.g., refractive errors) [15,16,22,26]. Such solutions are intended to allow users to see the displayed content without the need of spectacles or contact lenses and are tailored for the needs of each individual. Instead of optical aberrations of the visual system, our method takes into account the motion blur PSF to generate content that looks sharp under the prescribed amount of motion blur and does not require customized hardware.

Pre-filtering A few methods focus on filtering the image before viewing (a.k.a pre-filtering) for resolution enhancement (e.g., Lee et al.. [18]) and others for avoiding optical aberrations. In this latter category, methods use Wiener filtering [2,4,15,25] or constrained total variation [23] to alter an image but suffer from severe contrast reduction (e.g., 5-10× loss) and/or visual ringing artifacts (i.e., unwanted intensity oscillations). Some methods perform spectral remapping when down-sampling so as to preserve the appearance of high-frequency structured patterns [13], but such a solution is not effective to prevent motion blur.

3 Motion blur

3.1 Frequency response

Linear motion blur occurs due to relative translation at constant speed between an object and a sensor, where the object can be images shown on computer screens or real-life objects, and the sensor can be a camera or a human. Motion blur can be modeled in the spatial domain as a convolution with a box kernel [6,14,19,28]. This means that the motion stage of the image formation from Equation 1 is represented using a box kernel K:

$$I_{\text{blur}} = M(D(I)) = K \otimes D(I). \tag{3}$$

3.2 Snapping

Regardless of the pre-filtering methodology, the zeros in the frequency response of the motion blur kernel will remove certain frequencies present in the original content. A box kernel of size k in the spatial domain is a rectangular function:

$$f(x) = \begin{cases} 1/k & \text{if } x \in [0, k] \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Its modulation transfer function (MTF) is a sinc function:

$$\text{MTF}(f) = |F(X)| = \text{sinc}(kX) = \frac{\sin k\pi X}{k\pi X} \tag{5}$$

with zeros points located at $Z_k = \{\pm 1/k, \pm 2/k, \pm 3/k, \dots\}$ (see blue curve in Fig. 2). This means that in any pre-filtered image, frequencies in Z_k are killed by the kernel. For example, Fig. 3 (right) illustrates a *target image* of a single vertical gray bar of width 42 pixels on a solid red background (top row). The *blurred image* represents the appearance when undergoing motion blur induced by a kernel of size 17 pixels (second row). Pre-filtering the target image for a motion blur kernel of size 17 creates the *pre-filtered image* [23] (third row) which upon motion blur results in a blurred pre-filtered image exhibiting visual artifacts (Fig. 3 fourth row). The artifacts inside and outside the gray bar are caused by the frequencies removed by the zeros in the kernel's frequency response (see Fig. 2). The strength of the artifacts increases with number of bars and higher contrast demands.

The target image can also be seen as a rectangular function but with a different width T, and its MTF has zero amplitude at $Z_T \in \{\pm 1/T, \pm 2/T, \pm 3/T, \dots\}$. If $T = nk$ where $n \in \mathbb{N}^+$, then we have $Z_T \supseteq Z_k$, which means that the frequencies killed by the kernel does not exist in the target. This implies that the image content can avoid having energy on Z_k. Figure 3 (left) shows an experiment where we alter the width of the vertical gray bar to be an integer multiple of 17 pixels (the blur kernel size). Thus, the bar width in 34 pixels (top row) and the zeros of the kernel's frequency response align well with the zeros of the vertical bar's frequency response. Thus, there is no energy in the content at the frequencies canceled by the motion blur kernel. Therefore, pre-filtering does not introduce ringing artifacts (bottom row, left). The images in the right column are not a multiple of 17 hence show ringing artifacts (bottom row, right). Thus, the image formation process *snaps* together whenever bar width is an integer multiple of the blur kernel size. A similar solution applies to more complex content as well. For example, for a case of multiple bars we must consider the background spacing which should also be an integer multiple of the blur kernel size. Further, the distance to the left and right boundaries of the image must also be an integer multiple of the blur kernel size (or very large as compared to the size of the text).

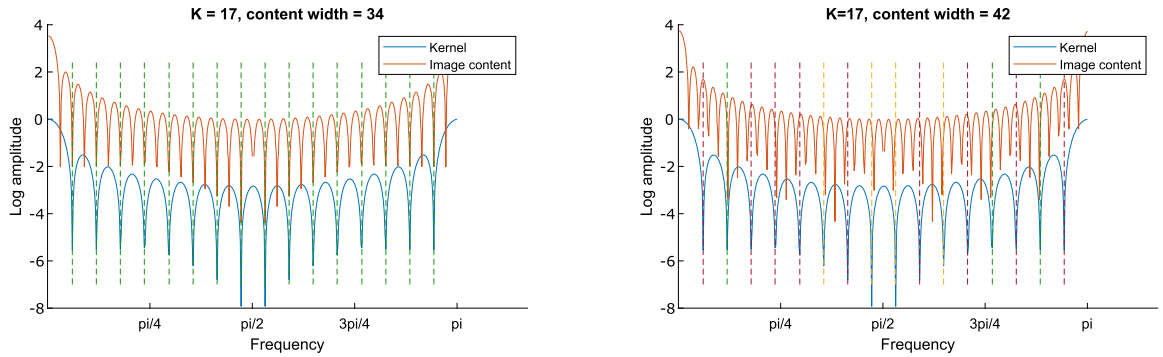

Fig. 2 *Frequency response* Left: Fourier spectra of a motion blur kernel (box filter) of size $K = 17$ (blue) and of one row of an image with a single bar (Fig. 3) of width 34 in the middle (orange). Bar width is a multiple of K, resulting in the zeros lining up and thus reduced artifacts in the blurred pre-filtered image. Right: same blur kernel but image with bar width of 42 (not a multiple of K). The kernel removes frequencies indicated by red dashed lines, which leads to visual artifacts

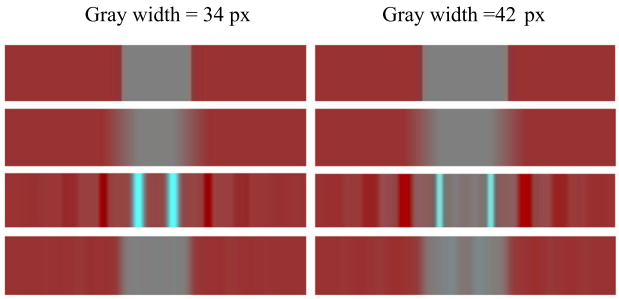

Fig. 3 *Snapping* A comparison of target images (top row), motion blurred target (2nd row), pre-filtered target (3rd row), and blurred pre-filtered target (4th row). See text for details

4 Content design

Our algorithm for designing alphanumeric content suitable for pre-filtering consists of three stages: *vectorization*, *snapping*, and *interpolation*. It supports solid-colored symbols by applying our methodology separately to each color channel. The ultimate goal is to produce an image $I_{\text{warp}} = W(I)$ to replace the image I so that its Fourier spectra does not have energy in the frequencies killed by the kernel K. The resulting image I_{warp} goes into the pre-filtering step described in Sect. 5.

4.1 Vectorization

In a first step, we convert the alphanumeric symbols into a vectorized format and divide each of them into multiple horizontal slabs. In our method, the symbols to be displayed can be represented using one of the following options:

- image vectorization (Xia et al. [35], Nehab et al. [24]);
- dot-matrix versions of the symbols [33]; or,
- splines of TrueType font glyphs, preferably sans-serif.

Note that the image vectorization approach supports any symbol besides alphanumeric characters, such as graphical symbols, road sign symbols, and so forth.

To divide a symbol into horizontal slabs, we use a vertical sweep line algorithm [31]. We define an "event" on the boundary of a symbol to be whenever its defining polyline/polycurve has a corner/kink. We place two horizontal slab lines tangent to the top and bottom of the symbol, respectively. Then, we sweep a horizontal line from top to bottom and whenever an event of the vector format is hit, a new horizontal slab line is added. The result is a partitioning of the letter into horizontal slabs, where each slab may have a different height (Fig. 4a). Depending on how the symbol was rasterized some special treatment might be necessary. For example, with a dot-matrix pattern we can either (i) surround the pattern with a polyline and partition as described in the previous paragraph, or (ii) use each row of the dot-matrix pattern as a horizontal slab (Fig. 4d). Afterward, we replace each foreground segment of a horizontal slab with blocks of size K (Fig. 4b,e). Further, for the image vectorization case,

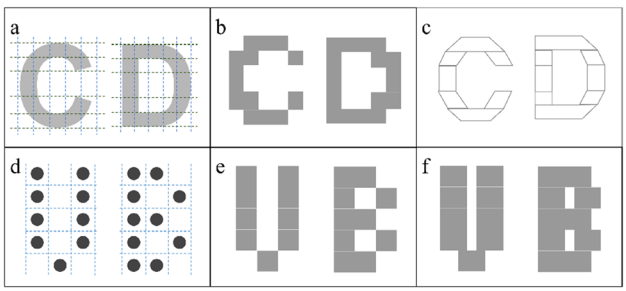

Fig. 4 *Illustration of our method* **a** Letters C and D rasterized from a sans-serif font. They are divided into horizontal slabs (horizontal dashed lines). **b** Letters C and D built using blocks of width K. **c** The quadrilateral representation for the foreground segments of the letters. **d** Dot-matrix representation of letters V and B. **e** The result when $K = N$. **f** The result when $K = 1.3N$

we further tighten the quadrilateral around each foreground segment to the original content but ensure C0 continuity with the foreground segment above and beneath it (Fig. 4c)—this fitment is not necessary with dot-matrix versions because in this case the original content is formed by rectilinear blocks.

We point out that letters require at least five horizontal slabs since such is sufficient to distinguish the letters with 20/20 vision—this stems from the Snellen optotypes that use at least 5 lines of one arc-minute each per letter.

4.2 Snapping

4.2.1 Definition: native kernel size

We define the *native kernel size* N to be the width of the "stroke" used to draw a symbol. For dot-matrix symbols, this is the size in pixels of one dot-matrix square. For the vectorized or spline-based font, it is the horizontal width of each foreground segment.

It is useful to express motion blur as a multiple of the native kernel size N of a symbol. For example, given a motion blur kernel size K that is larger than N, two options are to enlarge the symbol or to increase the symbol weight (e.g., make the symbol *bold* face) so as to reduce the relative effect of motion blur. This entails enlarging N and/or changing the number of segments per horizontal slab of the letter. However, the most extreme case is when the symbol size is to be kept constant (or is already at its maximum desired size) and the symbol is already at its maximum weight. At this point however, a minimum symbol topology must be maintained. At least for English letters, while more segments might be desired, a letter needs at least 3 segments per horizontal slab (e.g., the middle horizontal row of an "O" must at least be a foreground–background–foreground segment sequence). Dot-matrix patterns and Snellen optotypes as small as 3x5 pixels (but more often 5x5 or 5x7) can produce distinct representations for 26 letters and some alphanumerical symbols as well. In this extreme case, the native kernel size of the letter can be no more than 1/3 of the letter width. Altogether, using N enables us to understand the effect of the motion blur independent of the absolute values but rather in relation to the ratio of the sizes of the native kernel and the motion blur kernel.

4.2.2 Snapping algorithm

The vectorization and native kernel size are used to alter the symbol so that the resulting image $W(I)$ does not have frequencies that would be removed by motion blur. The warping method is divided into two strategies:

– *Fully Snappy:* visual artifacts are eliminated because the spacing between and among foreground and background elements are all integer multiples of K; and

– *Foreground Snappy:* some visual artifacts remain because only the width of the foreground elements are an integer multiple of K.

In the fully snappy configuration, the native kernel size N is changed to match K. This approach is straightforward for dot-matrix symbols but additional treatment is potentially needed for vectorized fonts. For vectorized fonts, while making $N = K$ will causes the foreground to "snap" into place, the space in between foreground segments of the symbol (i.e., the white-space or background) might not be a multiple of K. Thus, the location of the surrounding foreground pixels will need to be altered so that the background is also snappy. This warping must be done while keeping at least C0 continuity (as well as foreground segments also being a multiple of K). Initially, we performed this alteration using a mesh-warping scheme applied to the vertices of the vector-based representation of each symbol. But, this led to several problematic mesh distortion and symbol recognizability issues. For image-vectorized symbols, we instead rasterize all horizontal slabs onto a grid of cells, each cell of width N. Then, the grid cells mostly occupied by foreground segments are considered foreground grid cells and the rest are the background grid cells (Fig. 4b).

For the foreground-snappy configuration, we only adjust the width of the foreground segments. The Snellen optotypes, as mentioned previously, inform us that at least 3x5 segments are needed in order to distinguish between 26 English letters and a set of numeric and punctuation symbols. In this case, without making the letter any wider, we can scale each foreground segment to be close, but not equal, to $1.5N$ (i.e., the two foreground segments sum to almost $3N$, leaving at least a small sliver of background—see Fig. 4f). For vectorized letters, this configuration also works but the maximum multiple of N that can be achieved depends on the ratio of the native kernel size to the entire letter width. A foreground-snappy configuration is useful because it does not alter the width of the letter. However, it is limited to blur kernel sizes K less than $1.5N$, in general.

In summary, the content is typically designed based on the relative size between the kernel and the targeted image:

1. If the kernel size is large, then we adopt a dot-matrix design (Fig. 4d). Fully snappy design (Fig. 4e) is used if $K = N$, otherwise ($K > N$) one has to use a foreground-snappy design (Fig. 4f).
2. If the kernel size is small enough, then we use a vectorized representation of the letter (Fig. 4a) with some $N \geq K$. Fully snappy design is feasible (Fig. 4b), and we also produce a trapezoidal representation of the original shape (Fig. 4c) for the next optional step.

4.3 Optional interpolation

We support an optional geometry-based interpolation between the content designed for pre-filtering $W(I)$ and the original symbol I. The aforementioned partitioning of the symbol into horizontal slabs, and each slab into quads, can be used to define a correspondence between the pre-filtered and the original content. The parameterized version of symbol warping $W(I, t)$ interpolates each quad from its trapezoidal form (at interpolation parameter value $t = 0$, Fig. 4c) to its fully snappy form (at $t = 1$, Fig. 4b). Presumably, the 2 quadrilaterals have their parallel edges collinear. The interpolation is then straightforward, interpolating 4 corners of the quadrilaterals respectively.

There is a perceptual dissimilarity in appearance of every letter after being warped, and we notice that such dissimilarity varies by letter. For example, the appearance of letter E does not change much after warping, but letters G and K change drastically (see Fig. 8). Such dissimilarity is subject to human perception. Rather than making subjective evaluations, we use the perceptual distance measure of Zhang et al. [37]. Their deep learning-based method is trained to provide a perceptual distance measure between two images. Their system can be used to guide a perceptually linear interpolation between the original and pre-filtered form of *each* symbol. This means we can choose a threshold level of perceptual distance and obtain a suitable per symbol interpolation. For example, the letter E is straightforward to modify for good pre-filtering, while K is harder. For various interpolant t values, we compute the perceptual distance between non-warped letter $W(I, 0)$ and $W(I, t)$. Shown in Fig. 5, distance is low for all variants of E but increasingly higher for K as t grows. Thus we can pick a single global maximum distance threshold to select a set of letters that perceptually deviate about the same amount from the original.

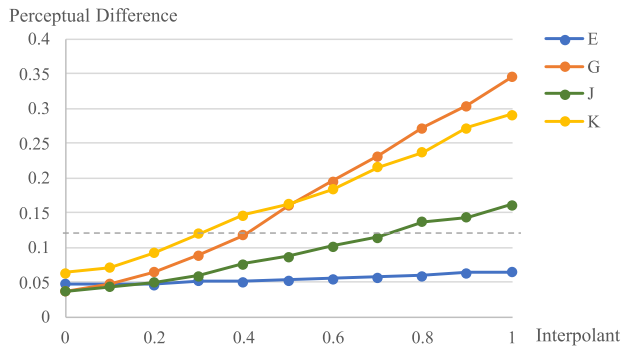

Fig. 5 Perceptual dissimilarity for different letters at various interpolants. Letters with mostly horizontal/vertical straight strokes do not differ much after warped, but others (curvy/diagonal strokes) have a higher difference. With a chosen level of dissimilarity (e.g., the gray dashed line at perceptual difference of 0.12), one can find the interpolant for individual letters so that they are at the same level of perceptual similarity to the original design (0.3 for K, 0.4 for G, 0.7 for J, and 1.0 for E, since the gray line is above it for all interpolants). See Fig. 12 for an example

5 Pre-filtering and display

5.1 Algorithm

We pre-filter the snappy content so that when it undergoes motion blur the result has a relatively sharp appearance with few or no visual artifacts. The pre-filter minimization finds an F that solves

$$I_{\text{pre}} = P(I_{\text{warp}}) = \arg\min_{F} \|K \otimes F - I_{\text{warp}}\|_2,$$

where $I_{\text{warp}} = W(I, t)$ is the result from Sect. 4 and I_{pre} is the resulting pre-filtered image. While there are multiple pre-filtering options, we choose the method of Montalto et al. [23] because it centers on producing a relatively high-contrast result. However, rather than using a Zernike polynomial-based PSF, we use a box filter one. Nonetheless, their total variation-based solution can still produce pre-filtered patterns with a controlled maximum relative total variation. The resulting pixel values are constrained to the [0, 1] range and resulting image exhibits higher contrast as compared to Wiener filtering or inverse filtering. Examples of this pre-filtering content are shown in Fig. 10.

Montalto et al.'s pre-filtering method, as well as others, requires reserving intensity value ranges above and beneath the maximum and minimum pixel values to be used by the method. Based on recommendations by Montalto et al. [23], we use 0.2 as the minimum background pixel value and 0.8 as the maximum foreground pixel value. For 8-bit pixel values, this range corresponds to [51, 204].

5.2 Ringing vs. contrast tuning

As mentioned above, Montalto et al.'s method allows us to trade off contrast for the addition of total variation (which appears as ringing artifacts) using a single scalar parameter. A small parameter value means less ringing but also less contrast; a large value results in more contrast at the cost of additional ringing. Since our method helps with reducing the unwanted effects of ringing artifacts, we can essentially achieve an improved result with less visual artifacts. In Sect. 6, we explore the impact of this parameter (θ) on the results of our preemptive technique.

5.3 Display calibration

For digital display usage, we must ensure the snappy content is displayed with a linear intensity ramp. We computed I_{pre}

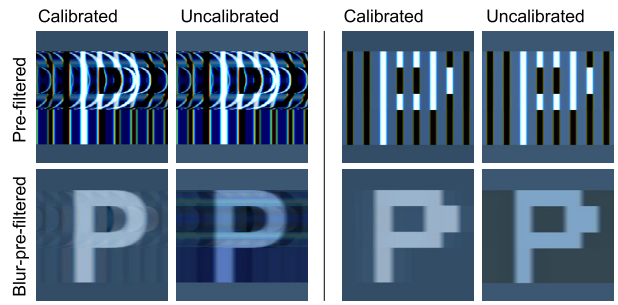

Fig. 6 *Left* Letter P pre-filtered without warping, with calibrated color (top left) and uncalibrated color (top right). The simulated blur of those images is placed below them. For the uncalibrated image, color intensity is not linearly mapped to the emitted irradiance, hence color inaccuracy in the blurred result. *Right* The same layout but using snappy images. The uncalibrated image, due to its usage of less color tones, does not suffer much from color nonlinearity

in the intention that it should be used as $K \otimes I_{\text{pre}}$. But in reality, convolution happens after images are displayed, after pixel values I are mapped to irradiance values $D(I)$. Usually for consumer displays, a power function can describe the monitor's irradiance mapping:

$$I_{\text{disp}} = D(I_{\text{pre}}) = A I_{\text{pre}}^{\gamma}$$

where popular γ values vary from 1.8 to 2.2 and A normalizes the dynamic range. To keep the linearity to go from I_{pre} to $I_{\text{disp}} = D(I_{\text{pre}})$, we apply the inverse irradiance mapping immediately after obtaining I_{pre}:

$$I'_{\text{pre}} = D^{-1}(I_{\text{pre}}) = (I_{\text{pre}}/A)^{1/\gamma},$$

and display I'_{pre} instead. Figure 6 shows the result of pre-filtered images observed with blurring, using calibrated and uncalibrated colors, and the uncalibrated images worsens the perceived artifact.

6 Experiments

We have used our system to create multiple simulated and real-world experiments shown in this section and in supplementary material. The system is implemented in C++/Python and generates all results at near interactive speeds on a 2.9 GHz Intel i7 PC with 16 GB of RAM. For real-world captures, we use a 25-inch 60Hz HD display and a Canon EOS Rebel T6i camera. For our user study, we use a 240 Hz Full HD (1920 × 1080) 25-inch LCD display (the higher FPS enables displaying faster motions). The compute time, using a prototype CPU-only implementation, to automatically generate a solution for a typical symbol is within 200 milliseconds—this includes vectorizing the font and snapping (20ms), and pre-filtering (180 ms). The display/camera radiometric calibration was done only once for all images in this paper.

Figure 7 shows simulated results for two letters (E and K), for various motion blur sizes ($K = 20, 25, 35$), including fully snappy (left) and foreground-snappy (right) designs. These letters were chosen because they represent two categories of symbols: *easily snappable* (E) and *triangular shapes* (K). The latter requires one to trade off shape similarity for robustness to motion blur. For each letter, we compute a vectorized version (i.e., dot-matrix design in this case), a pre-filtered version, and the blurred pre-filtered result (i.e., what "a person would see"). The letter E (as well as other letters like T, L, I and H) is straightforward to make snappy and thus its warped version has high similarity to the original one. The letter K (as well as R, N, M, and others) is more challenging because of the background that must appear within the general confines of the letter. The foreground-snappy solutions do not need to be as wide, which is particularly useful for the smaller kernel sizes, although at the cost of some ringing artifacts.

Our approach handles colored symbols by performing the content design of Sect. 4 and computing a pre-filtered solution separately for each color channels. Figure 8 illustrates this by showing adjacent letters against a background, all with different colors.

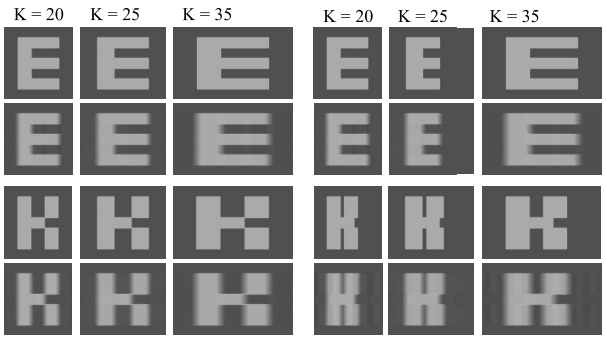

Fig. 7 *Blur size variation* Letters E and K generated using a 3 × 5 dot-matrix design for $K = 20, 25$, and 35 pixels (1st and 3rd rows) for fully snapped (left) and foreground-snapped (right) designs. Letters blurred after pre-filtering (2nd and 4th rows)

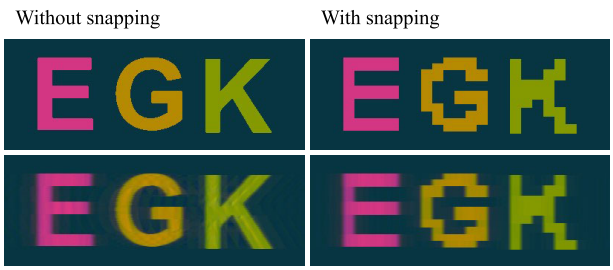

Fig. 8 *Colored letters* The original letters (left), when blurred after pre-filtering (second row), exhibit some ringing artifacts. The snapped ones (right) reduce the occurrence of such artifacts

We can also optionally interpolate between our snappy content design and the initial vectorized representation of each letter. Figure 12b shows the letters "TVC" at different interpolation states. At close to original (top row), the content exhibits ringing artifacts. Near the fully snappy solution (bottom row) the artifacts are less noticeable. By choosing a perceptual distance limit of 0.3, we can find the interpolation value for each symbol that produces an output close to but not more than 0.3 perceptual distance from the original. The Δ_P values, shown on left side of Fig. 12, are the perceptual distance of the letters with respect to the original sans-serif fonts according to Zhang et al.'s metric [37].

As described in Sect. 5.2, we can alter the trade-off between ringing artifacts and contrast. A close inspection of Fig. 9 reveals that the pre-filtered images exhibit progressively more ringing artifacts as the value of the parameter θ increases, while their corresponding blurred pre-filtered counterparts become progressively sharper. The example shown in Figs. 1b and 1d were obtained using $\theta = 1000$.

Figure 10 compares our method with various alternatives. In particular, we show the naive appearance of the motion blurred word "FAST" considering $K = 15$ (Fig. 10a). Figure 10b shows the result obtained when enhancing the content using the Cornsweet illusion [7], where contrast on the edges are increased. The letters still look blurry after being convolved with the kernel. Figure 10c shows the result of using a Wiener pre-filtering, which achieves better sharpness after blurred, but exhibit artifacts. Our approach (Fig. 10d) avoids ringing artifacts and blurry edges, at the expense of the appearance of curvy and triangular shaped fonts.

Although post-processing methods work in a different stage, we show the results of deconvolution methods to remove motion blur in Fig. 11. The top image is a part of a camera-captured image which suffers from motion blur. Two methods, Lucy–Richardson (LR) deconvolution [21] and Fortunato et al. [12] are used. The deconvolution does improve readability but leaves visual ringing artifacts and is done digitally *after* capture—thus, it is not preemptive like our method (see Fig. 9).

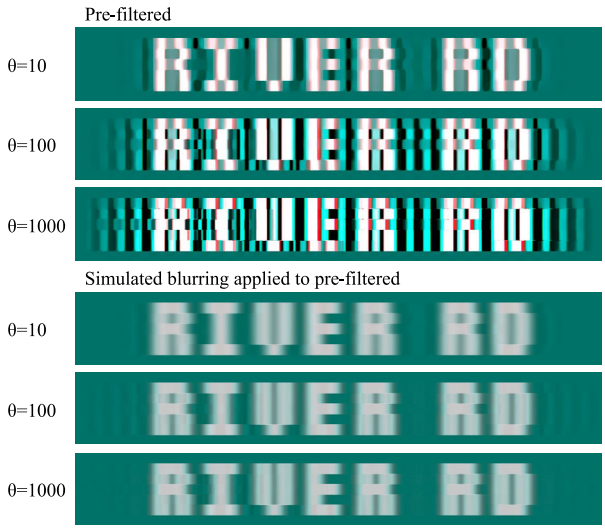

Fig. 9 *Road sign variants* Road sign shown in Fig. 1d with different θ values used for pre-filtering [23]

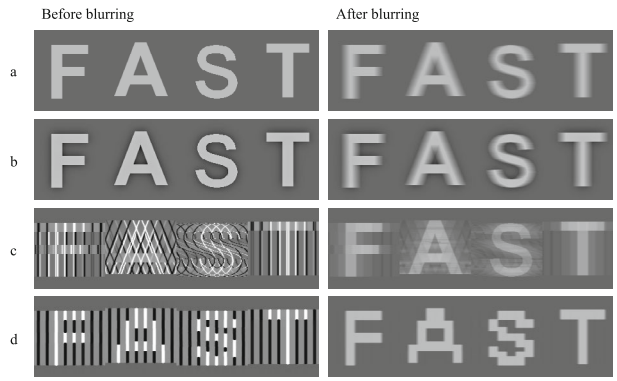

Fig. 10 *Comparison* Word "FAST" in Arial font-face, preprocessed with different approaches (left) and blurred by the same kernel with $K = 15$ pixels (right). **a** Original letters. **b** Preprocessed with Cornsweet illusion. **c** Pre-filtered by Wiener deconvolution. **d** Our method

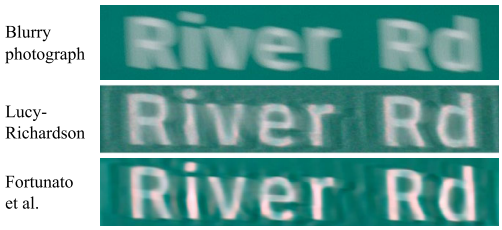

Fig. 11 *Deconvolution* Top: Part of a blurry photograph. Middle: top image deconvolved by Lucy–Richardson using post-processing. Bottom: The same image deconvolved using Fortunato's method [12]

Figures 12, 13 and 14 show several examples captured in the real world, using a video camera. Figure 12 shows a word moving at 1px per frame ($\approx$ 1.6cm / second) in front a video camera capturing at 4 fps. Figure 12a shows the pre-filtered content moving on a display. Figure 12b is what the camera captures if the display–camera system is not radiometrically calibrated. In contrast, Fig. 12c is the improved result after radiometric calibration. The first four rows correspond to perceptual distances of 0.1, 0.2, 0.3, and 0.4, respectively. The last row corresponds to a fully snappy design. We also included the original typeface at the top row as a comparison against Montalto's method [23], i.e., pre-filtering without altering the shape first. Both the uncalibrated (b) and calibrated (c) colors show that the original design exhibits visual artifacts, while the snappy letters minimized the artifacts. Our user study in Sect. 7 shows that the users perceive more

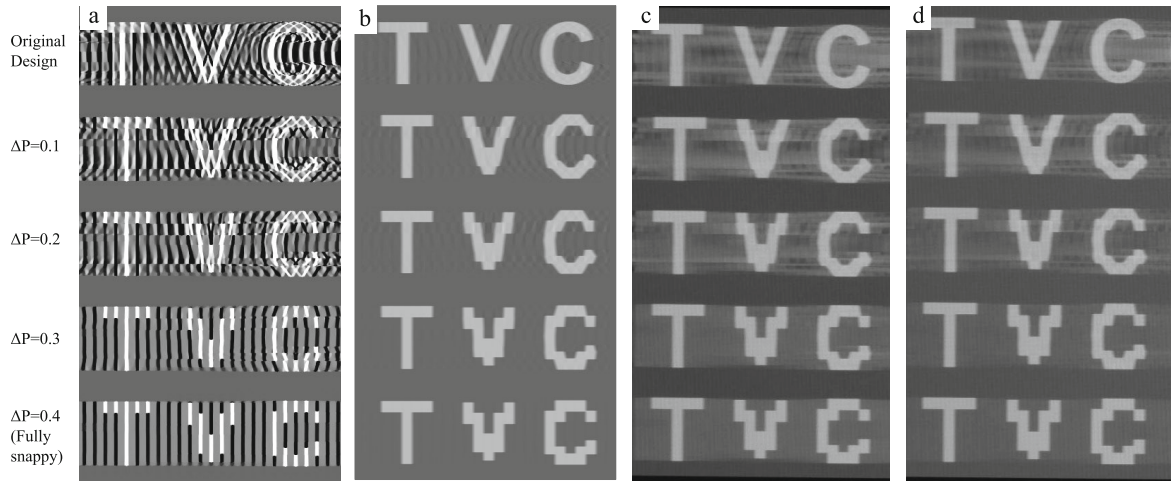

Fig. 12 *Geometric interpolation guided by perceptual distance and real-world calibration:* Letters "TVC" designed from Arial fonts, pre-filtered and blurred using the same kernel. The original font-face is at the top row, and the following rows at different perceptual distances to the original design (ΔP values). **a** Pre-filtered letters for $K = 15$ pixels, displayed on a monitor, and moving 1 pixel/frame at increasing perceptual distances from original. **b** Simulated blurred result of (**a**). **c** Photograph of **a** without radiometric calibration, with exposure time of 0.25 seconds, during which the camera integrates 15 video frames. **d** Same with **c** but with radiometric calibration

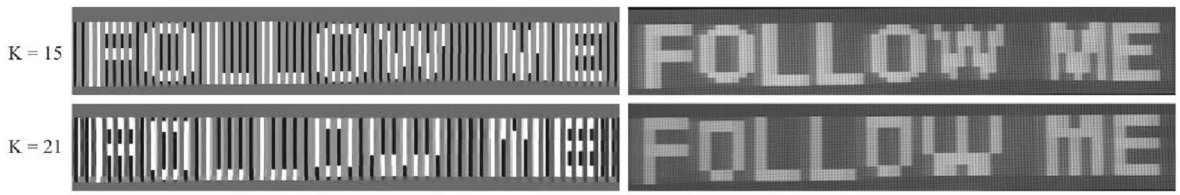

Fig. 13 *Real-world experiment for varying motion speeds*. (Left) Pre-filtered letters rendered to around 100 pixels wide and made snappy for $K = 15$ pixels (top) and 21 pixels (bottom). (Right) Captured by a camera with corresponding shutter time

Fig. 14 *Real-world experiment with printed content.* (Left) static shot of the scene. Letters "TOYO TIRES" and "20" are made snappy and pre-filtered, printed and fixed on a toy car, facing toward the camera. (Middle) a frame from the captured video (around 0:50 in the supplementary video) while the car is in motion. (Right) close-up views. Top: snappy text under motion blur; middle: pre-filtered text under the same motion blur; bottom: number "20" of original design and blurred pre-filtered version

artifacts on the non-snappy letters, to the point that it affects reading.

Figure 13 has frames from a recorded video of a phrase moving about the screen at two different speeds, corresponding to blur kernel sizes of $K = 15$ and $K = 21$ pixels, respectively. As the text moves and changes speed, a new pre-filtered design is computed and shown (Fig. 13(left)). Regardless, the captured blurred pre-filtered appearances are nearly sharp and roughly similar (Fig. 13(right)).

Figure 14 shows how our method works on printed content (see supplementary video). We recorded a video of a toy car moving down a slope due to gravity. The camera is facing the car and is tilted such that it is horizontally aligned with the slope. The text on the car suffers from motion blur due to the motion of the car. Then we computed snapped and pre-filtered

text imagery, printed and fixed onto the car: "20" on the side of the car, close to the existing text "20" that comes with the toy car; "TOYO TIRES" on top, as an attempt to reproduce the text above the rear tire. All the content on the car suffers from the same motion blur, but the pre-filtered images appear sharp. The car's motion is at constant acceleration instead of constant speed, which we do not account for. Please see the video in supplementary materials.

We recommend the readers to see the other part of the supplementary video, which is similar to what the participants saw in our user study. The clip has the word "PACE" shown in 3 different variants: original design, non-snappy pre-filtered, and snappy pre-filtered. The reader is encouraged to pause at any frame of the video and verify that it has sharp content, but the original content (without pre-filtering) appears blurry once the video starts to play and eye-tracking is established. The video playback rate should be smooth and constant.

7 User study

7.1 Human perception of electronic displays

Upon perceiving an object in motion, human eyes can establish motion tracking to the object within 0.1 seconds [9]. While the eyes move at the same speed as the object, human eye motion is continuous while the monitor refreshes the screen every certain period of time. For example, for a monitor of 60Hz refresh rate, once the human eye motion tracking is established, then within any 16ms ($\approx 1s/60$) of time interval between 2 monitor refreshes, the human eyes move but the content on the monitor stays still. Assuming that the content moves K pixels every frame (as many rolling banners do), then this would result in a perceived motion blur of size K in the human eye. This is a perfect scenario where our method can be applied.

7.2 Setup

In our user study, 22 volunteers observed 36 randomly ordered pairs of moving images on a 25-inch 240 Hz display located about 25 inches in front of them (Fig. 1a). The application updates the content at 60 Hz, in order to replicate the behavior of regular 60Hz displays. The task was to report the perceived level of sharpness/blurriness and of artifacts in the bottom moving image compared to the top moving image. The top moving image was always the baseline (i.e., naively rendered content, snappy or non-snappy) and the bottom image was either:

– *control images:* non-pre-filtered image, snappy or non-snappy, or

– *experimental images:* pre-filtered image, same snappiness as the top image.

Table 1 Choices we gave to users to respond on sharpness/blurriness and artifact strength

Sharpness increase		Artifact strength	
Level	Description	Level	Description
-2	Strongly blurrier	0	None
-1	Slightly blurrier	1	Mild
0	Similar	2	Medium
1	Slightly sharper	3	Strong
2	Strongly sharper	4	Severe

The image content was vertical bars or one letter (from the set E, C, N, G, P) and moved at one of a randomly selected speed level: low (i.e., slow enough that motion blur was barely present), high (i.e., as fast as possible but not disturbing) and medium (i.e., the midway speed).

The users report their perceived sharpness/blurriness increase/decrease and artifact strength with one of the 5 choices shown in Table 1. We designed 5 levels of increased sharpness that the users can choose from, and map them to a scale from – 2 to + 2. The level of artifacts is described from "None" (not perceivable) to "Severe" (interferes the viewing experience), and quantified from 0 to 4. We collected users' feedback and group them by the image content (letters or vertical bars) and motion speed.

7.3 Analysis

Our study shows that for *all the test letters and vertical bars* at medium and high speeds, our approach provided increased sharpness as compared to the corresponding control images (see Table 2 and Fig. 15 top). Our approach also exhibits reduced artifacts at medium and high speeds as compared to the non-snappy pre-filtered content, at a statistical significance of p-value less than 0.05 (see Table 3 and Fig. 15 bottom). In contrast, the non-snappy pre-filtered content only sometimes produced increased sharpness (see less occurrence of $p < 0.05$ in Table 2) and always displayed noticeable visual artifacts at medium and high speeds.

A few exceptions to the superior performance of our method occurred at low speed for some content. In particular, the snappy pre-filtered letters E and N were reported (with statistical significance) to be slightly more blurry than the original versions. Upon close inspection, each had 1-2 users that provided un-intuitive responses (e.g., perhaps due to fatigue or confusion). Removing those user responses would result in our method also demonstrate increased sharpness in those cases.

Table 2 Statistics of increased sharpness in the experimental images and *p*-values compared to the corresponding control images

Content	Snappy	Low speed		Medium speed		High speed	
		Mean ± SD	*p*-value	Mean ± SD	*p*-value	Mean ± SD	*p*-value
Bars	Y	1.36 ± 1.00	**0.001**	1.50 ± 0.70	**0.000**	1.40 ± 0.91	**0.001**
	N	0.60 ± 0.91	**0.023**	0.30 ± 1.29	0.123	0.30 ± 1.35	0.234
Letter C	Y	1.14 ± 0.69	**0.008**	1.57 ± 0.53	**0.000**	1.86 ± 0.38	**0.000**
	N	0.71 ± 1.11	0.086	0.71 ± 1.25	0.055	1.28 ± 0.49	**0.001**
Letter E	Y	0.11 ± 0.60	0.500	1.11 ± 0.11	**0.001**	1.44 ± 0.53	**0.000**
	N	0.44 ± 0.73	0.097	0.67 ± 0.87	**0.022**	1.11 ± 0.60	**0.000**
Letter G	Y	1.00 ± 0.50	**0.007**	1.44 ± 0.73	**0.002**	1.78 ± 0.44	**0.000**
	N	0.89 ± 1.05	0.069	1.22 ± 0.83	**0.001**	1.00 ± 1.50	0.085
Letter N	Y	1.00 ± 1.00	0.140	1.86 ± 0.38	**0.000**	1.86 ± 0.38	**0.000**
	N	1.14 ± 0.38	**0.000**	1.57 ± 0.53	**0.000**	1.29 ± 0.49	**0.000**
Letter P	Y	0.75 ± 0.89	**0.021**	1.63 ± 0.52	**0.001**	1.63 ± 0.52	**0.000**
	N	0.25 ± 0.89	**0.226**	0.88 ± 0.99	**0.025**	1.13 ± 0.99	**0.009**

The bold values are p-values less than 0.05. It shows that pre-filtering does improve perceived sharpness in both snappy (Y) and non-snappy (N) content, and snappy ones have a few more scenarios with significance ($p < 0.05$) than non-snappy ones. See top row of Fig. 15 for plotted data of snappy content

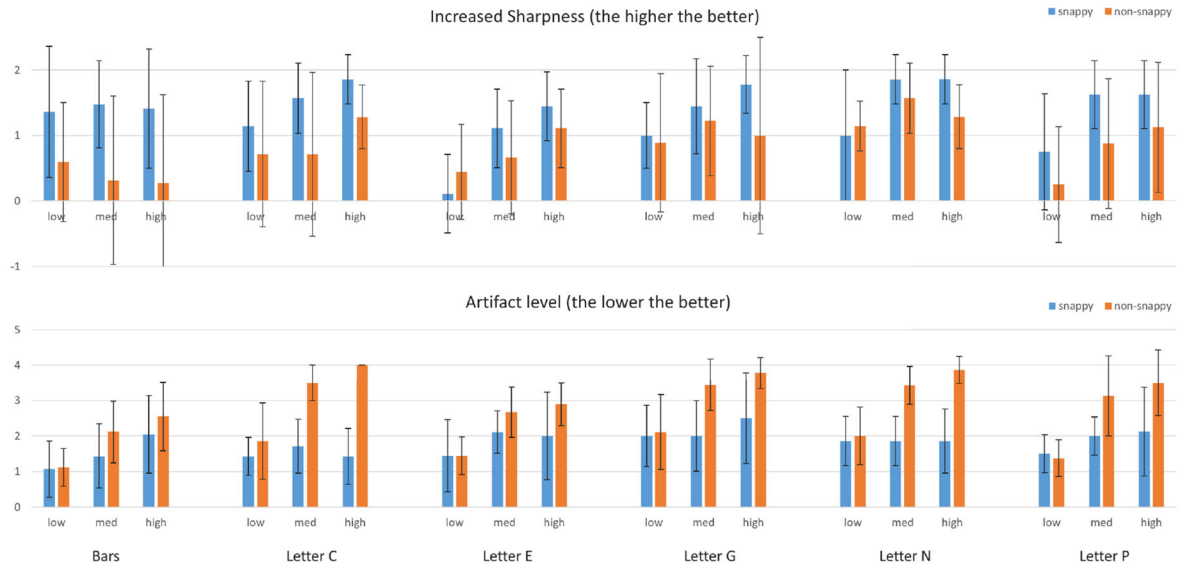

Fig. 15 Data plot of Tables 2 (top) and 3 (bottom)

Table 3 Artifact level

Content	Snappy	Low speed		Medium speed		High speed	
		Mean ± SD	*p*-value	Mean ± SD	*p*-value	Mean ± SD	*p*-value
Bars	Y	1.07 ± 0.79	0.406	1.43 ± 0.90	**0.012**	2.05 ± 1.09	**0.035**
	N	1.11 ± 0.53		2.11 ± 0.87		2.54 ± 0.96	
Letter C	Y	1.43 ± 0.53	0.145	1.71 ± 0.76	**0.002**	1.43 ± 0.79	**0.000**
	N	1.86 ± 1.07		3.50 ± 0.50		4.00 ± 0.00	
Letter E	Y	1.44 ± 1.01	0.500	2.11 ± 0.60	**0.001**	2.00 ± 1.22	**0.000**
	N	1.44 ± 0.52		2.67 ± 0.71		2.89 ± 0.60	
Letter G	Y	2.00 ± 0.87	0.297	2.00 ± 1.00	**0.002**	2.50 ± 1.27	**0.008**

Table 3 continued

Content	Snappy	Low speed		Medium speed		High speed	
		Mean ± SD	*p*-value	Mean ± SD	*p*-value	Mean ± SD	*p*-value
	N	2.11 ± 1.05		3.44 ± 0.73		3.78 ± 0.44	
Letter N	Y	1.86 ± 0.69	0.178	1.86 ± 0.69	**0.003**	1.86 ± 0.90	**0.002**
	N	2.00 ± 0.82		3.43 ± 0.53		3.85 ± 0.38	
Letter P	Y	1.50 ± 0.53	0.299	2.00 ± 0.53	**0.003**	2.13 ± 1.25	**0.018**
	N	1.38 ± 0.52		3.13 ± 1.13		3.50 ± 0.93	

The bold values are p-values less than 0.05. Both snappy (Y) and non-snappy images (N) are pre-filtered and their appearance under motion exhibits artifacts perceived by human. Snappy text produces significant less artifacts ($p < 0.05$) than non-snappy ones on medium and high speeds. See bottom row of Fig. 15 for plotted data

8 Conclusions, limitations, and future work

We have presented a preemptive method to preserve contrast and sharpness of text subjected to motion blur. Our technique prevents information loss by pre-filtering the target text in such a way that the resulting frequency content has no relevant information at the frequencies removed by the blur. We demonstrated the effectiveness of our technique with simulated and real-world experiments and a user study.

As limitations, our current approach cannot handle smooth gradients or highly textured patterns. Moreover, for very large kernel sizes, our mechanism resorts to creating very wide symbols which might not be desirable.

As future work, first we would like to dynamically adjust the parameters of our method in response to changes in motion. Second, we plan to support general image content by potentially also using rasterization and dithering techniques. Finally, we wish to create content robust to a continuous range of motion blur kernel size and thus alleviate the need for accurate motion blur prediction.

Funding This work is funded by CNPq-Brazil (312975/2018-0), CAPES Finance Code 001, NSF #1816514, NSF #2107096, and NSF #1835739.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Akşit, K.: Patch scanning displays: spatiotemporal enhancement for displays In. Opt. Exp. **28**(2), 2107–2121 (2020)
2. Alonso, M., Barreto, A., Adjouadi, M.: Digital image inverse filtering for improving visual acuity for computer users with visual aberrations. Inv. Probl. Sci. Eng. **16**(8), 957–966 (2008)
3. Banham, M., Katsaggelos, A.: Digital image restoration. IEEE Sign. Process. Mag. **14**(2), 4–41 (1997)
4. Brown, M.S., Song, P., Cham, T.-J.: Image pre-conditioning for out-of-focus projector blur. In: IEEE Conference on Computer Vision Pattern Recognition (CVPR 2006), vol. 2, pp. 1956–1963. IEEE (2006)
5. Campisi, P., Egiazarian, K. (eds.): Blind Image Deconvolution. Theory and Applications. CRC Press, Taylor & Francis Group (2007)
6. Cho, T.S., Levin, A., Durand, F., Freeman, W.T.: Motion blur removal with orthogonal parabolic exposures. In: 2010 IEEE International Conference on Computational Photography (ICCP), pp. 1–8 (2010)
7. Cornsweet, T.: Visual Perception. Academic Press (2012)
8. Debevec, P., Malik, J.: Recovering high dynamic range radiance maps from photographs. In: ACM SIGGRAPH 2008 Classes, pp. 1–10 (2008)
9. Didyk, P., Eisemann, E., Ritschel, T., Myszkowski, K., Seidel, H.P.: Apparent display resolution enhancement for moving images. In: ACM SIGGRAPH 2010 Papers, pp. 1–8 (2010)
10. Eboli, T., Sun, J., Ponce, J.: End-to-end interpretable learning of non-blind image deblurring (2020). arXiv preprint arXiv:2007.01769
11. Fortunato, H.E., Oliveira, M.M.: Coding depth through mask structure. Comput. Graph. Forum **31**(2), 459–468 (2012) (Proceedings of Eurographics)
12. Fortunato, H.E., Oliveira, M.M.: Fast high-quality non-blind deconvolution using sparse adaptive priors. Vis. Comput. **30**(6–8), 661–671 (2014)
13. Gastal, E.S.L., Oliveira, M.M.: Spectral remapping for image downscaling. ACM Trans. Graph. **36**(4) (2017)
14. Har-Noy, S., Nguyen, T.Q.: LCD motion blur reduction: a signal processing approach In. IEEE Trans. Image Process. **17**(2), 117–125 (2008)
15. Huang, F.-C., Lanman, D., Barsky, B.A., Raskar, R.: Correcting for optical aberrations using multilayer displays. ACM Trans. Graph. (TOG) **31**(6), 1–12 (2012)
16. Huang, F.-C., Lanman, D., Barsky, B.A., Raskar, R.: Eyeglasses-free display: towards correcting visual aberrations with computational light field displays. ACM Trans. Graph. (TOG) **33**(4), 1–12 (2014)
17. Kupyn, O., Martyniuk, T., Wu, J., Wang, Z.: Deblurgan-v2: deblurring (orders-of-magnitude) faster and better. In: IEEE Conference Computer Vision (ICCV 2019) (2019)
18. Lee, H., Didyk, P.: Real-time apparent resolution enhancement for head-mounted displays. Proc. ACM Comput. Graph. Interact. Tech. **1**(1) (2018)
19. Levin, A., Sand, P., Cho, T.S., Durand, F., Freeman, W.T.: Motion-invariant photography. ACM Trans. Graph. (TOG) **27**(3) (2008)

20. Lin, S., Gu, J., Yamazaki, S., Shum, H.Y.: Radiometric calibration from a single image. In: IEEE Conference Computer Vision Pattern Recognition (CVPR 2004), vol. 2, pp. II–II (2004)
21. Lucy, L.B.: An iterative technique for the rectification of observed distributions. Astron. J. **79**, 745–754 (1974)
22. Masia, B., Wetzstein, G., Didyk, P., Gutierrez, D.: A survey on computational displays: Pushing the boundaries of optics, computation, and perception. Comput. Graph. **37**(8), 1012–1038 (2013)
23. Montalto, C., Garcia-Dorado, I., Aliaga, D., Oliveira, M.M., Meng, F.: A total variation approach for customizing imagery to improve visual acuity. ACM Trans. Graph. (TOG) **34**(3), 1–16 (2015)
24. Nehab, D.: Converting stroked primitives to filled primitives. ACM Trans. Graph. (TOG) **39**(4), 137 (2020)
25. Oyamada, Y., Saito, H.: Focal pre-correction of projected image for deblurring screen image. In: IEEE CVPR, 1–8 (2007)
26. Pamplona, V.F., Oliveira, M.M., Aliaga, D.G., Raskar, R.: Tailored displays to compensate for visual aberrations. ACM Trans. Graph. (TOG) **31**(4), 1–12 (2012)
27. Peli, E., Woods, R.L.: Image enhancement for impaired vision: the challenge of evaluation. Int. J. Artif. Intell. Tools **18**(03), 415–438 (2009)
28. Raskar, R., Agrawal, A., Tumblin, J.: Coded exposure photography: motion deblurring using fluttered shutter. In: ACM SIGGRAPH 2006 Papers, pp. 795–804. Association for Computing Machinery (2006)
29. Ritschel, T., Smith, K., Ihrke, M., Grosch, T., Myszkowski, K., Seidel, H.P.: 3d unsharp masking for scene coherent enhancement. In: ACM SIGGRAPH 2008 Papers, pp. 1–8. Association for Computing Machinery (2008)
30. Robinson, E.A., Treitel, S.: Principles of digital wiener filtering. Geophys. Prospect. **15**(3), 311–332 (1967)
31. Shamos, M.I., Hoey, D.: Geometric intersection problems. In: 17th Annual Symposium on Foundations of Computer Science (sfcs 1976), pp. 208–215. IEEE (1976)
32. Shin, K.H., Ah, J.Y., Kim, K.D., Shin, H.H., Chung, I.J.: P-25: Acceptable Motion Blur Levels of a LCD TV Based on Human Visual System In SID Symposium Digest of Technical Papers, vol. 37, no. 1. Blackwell Publishing Ltd, Oxford, UK (2006)
33. Shutterstock.: Dot matrix letters images, (Accessed 14, 2020)
34. Tikhonov, A.N., Arsenin, V.Y.: Solutions of Ill-Posed Problems. W.H. Winston (1977)
35. Xia, T., Liao, B., Yu, Y.: Patch-based image vectorization with automatic curvilinear feature alignment. ACM Trans. Graph. (TOG) **28**(5), 1–10 (2009)
36. Xu, X., Sun, D., Pan, J., Zhang, Y., Pfister, H., Yang, M.H.: Learning to super-resolve blurry face and text images. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2017)
37. Zhang, R., Isola, P., Efros, A. A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: IEEE Conference on Computer Vision Pattern Recognition (CVPR 2018), pp. 586–595 (2018)
38. Zhang, S., Shen, X., Lin, Z., Měch, R., Costeira, J.P., Moura, J.M.F.: Learning to understand image blur. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)

Zixun Yu is a PhD student at Purdue University. He received his bachelor's degree from Nanjing University in 2016. His research interest include computer graphics, image processing, and computational photography.

Manuel M. Oliveira is a Full Professor at the Federal University of Rio Grande do Sul (UFRGS) in Brazil. He received his PhD from the University of North Carolina at Chapel Hill in 2000. Before joining UFRGS in 2002, he was an assistant professor at SUNY Stony Brook (2000-2002). In the 2009–2010 academic year, he was a visiting associate professor at the MIT Media Lab. His research interests include computer graphics, image processing, pattern recognition, computational photography, machine learning, and vision (both human and machine). He is an associate editor of ACM TOG and Computer Graphics Forum, and a former associate editor of IEEE TVCG and IEEE CG&A.

Daniel G. Aliaga is an Associate Professor at Purdue University. He received his PhD from the University of North Carolina at Chapel Hill in 1999. Before Joining Purdue University in 2003, he worked as a researcher at AT&T Bell Labs and Princeton University. His research interest includes computer graphics, computer vision and visualization. He is an Associate Editor for IEEE TVCG and Visual Computing Journal, and formerly Computer Graphics Forum and Graphical Models.