Video Folding: Increased Framerate for Semi-Repetitive Sequences

Chris May, Manuel M. Oliveira, and Daniel Aliaga

Abstract—We introduce a technique to synthetically increase the framerate of semi-repetitive videos (i.e., videos of motion that repeats but not in an identical fashion) to aid in visualization. By reordering and combining frames from all repetitions, we produce a single non-repetitive sequence with much higher temporal resolution. Then, we use a novel frame warping technique based on a dense corrective flow to counteract differences between repetitions. The resulting video maintains smoothness of motion and additionally allows for seamless, infinite looping. We demonstrate the effectiveness of the proposed solution both quantitatively, by measuring the improvement over existing methods, and qualitatively, by performing a user evaluation and providing several examples in the paper and accompanying video.

Index Terms—Computer graphics, picture/image generation, image-based rendering, image processing and computer vision, image processing software, time-varying imagery

1 Introduction

Generating slow-motion video from existing footage is a powerful tool for content analysis, entertainment, simulation, and games. For example, slowing down footage of sporting events can help professionals analyze the motion and make adjustments to athletes' training. However, videos are recorded at fixed framerates, so simply reducing the playback speed results in choppy motion due to individual frames being displayed for longer periods of time. Additional work must be done to smoothly fill in the temporal gaps between the original frames. In addition, for video retiming (e.g., framerate conversion), it might be necessary to re-sample the motion at a different speed.

Many previous works use frame interpolation to generate new in-between frames for an existing video. These methods may use optical flow [1], deep learning [2], or phase-based interpolation [3] to create the novel frames. However, in general these approaches do not work well with large motion in-between adjacent captured frames, and in some cases the adjacent frames may not even capture the underlying motion that occurred. Recording using faster framerate cameras is also a possibility, but may not always be feasible (e.g., cost or storage limitations). It also does not address processing existing videos, and even with faster framerates a further slow-down might be desired. Hence, prior works are not well suited to handle large motion in-between frames and to maintaining the motion dynamics with such large in-between frame motion.

Our key observation is that if the motion is at least semi-repetitive (i.e., the motion generally repeats but not in an identical fashion), information that is lost between consecutive frames may be recovered in later repetitions of the same motion. We exploit this fact by reordering the input frames so that similar frames from different repetitions are placed near each other. In this way, many repetitions from the input video are interleaved, or *folded*, into a non-repeating coherent sequence in the output video. As opposed to frame interpolation, this methodology does not depend on creating many novel in-between frames. Instead, we use existing re-ordered frames to create an apparently slowed-down version of the original video.

Our *video folding* (VF) approach consists of three main stages. First, during *frame reordering*, a video sequence is produced having one "repetition" of the motion but using all recorded frames. Since the fundamental motion is not exactly repetitive, our one repetition corresponds to a slowed-down playback of an "averaged" motion cycle, constructed based on all the provided repetitions. In Figure 1, frames from an input video are reordered into a "single repetition", in which similar frames are placed near each other (left). Second, because each repetition is slightly different, *corrective flows* are computed to bring all frames into a smooth and coherent sequence. The insets in Figure 1 show how the hand and foot are at slightly different positions in each frame, and the corrective flow images (middle) show the (color-coded) directions these features must move for them to stay in the same place. Third, each frame is *warped* along its corrective flow to produce the final video. Afterwards, a temporal smoothing operation is performed as well. Figure 1 shows the frames after warping (right), where the hand and foot (as well as other features) are now in roughly the same position in each frame. Altogether, we have applied our approach to real-world and synthetic videos of moving humans, animals, and devices. Our method is not limited to a frame rate range but instead depends on the motion speed per frame and number of repetitions. Nevertheless, we show empirical examples using frame rates of 2 to 120 frames-per-second. We show results, user evaluations, and comparisons in the paper and accompanying video.

Our main **contributions** include:

- A method to create slowed-down versions of semi-

- *C. May and D. Aliaga are with Purdue University. E-mails: may5, aliaga@purdue.edu.*
- *M. Oliveira is with UFRGS. E-mail: oliveira@inf.ufrgs.br.*

Manuscript received ??.

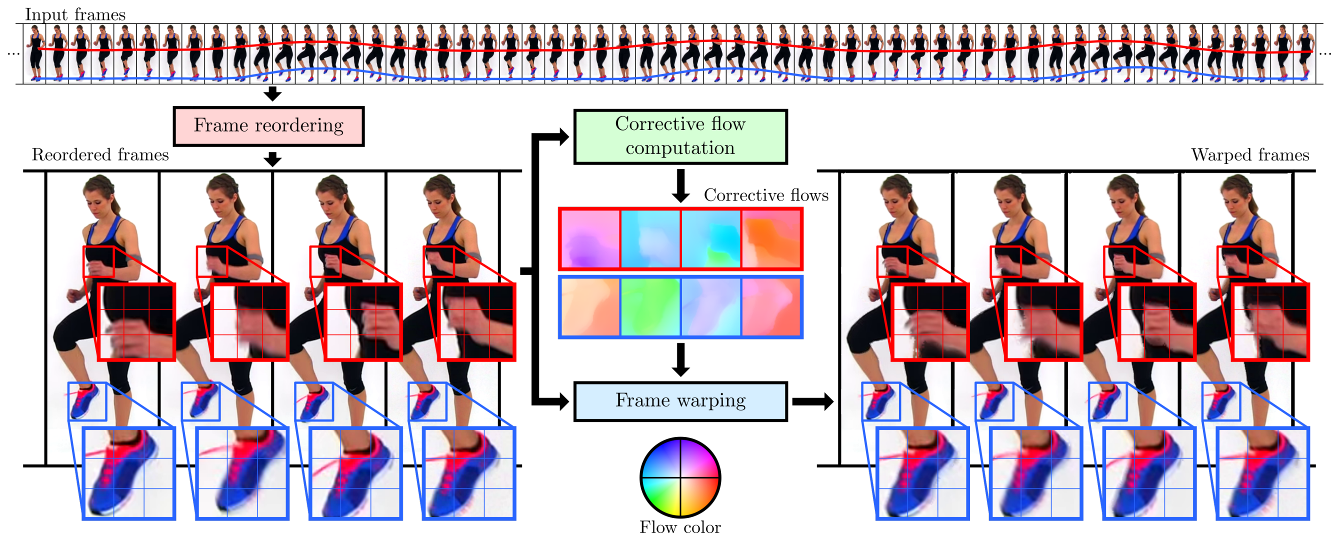

Fig. 1. Video folding for increasing the framerate of semi-repetitive videos. Top: frames from an input video sequence. The position of the hand and foot are shown throughout three repetitions in the video. Left: frames after reordering into a single repetition. Despite the generally-repetitive motion, in the insets it can be seen how the positions of the hand and foot still vary between consecutive reordered frames. This creates a distracting jitter effect during playback. Middle: corrective flow images computed for each of the reordered frames. The colors indicate the directions in which features must be warped to produce smooth motion, according to the legend at the bottom. Right: the same reordered frames after warping along the corrective flows. The insets show that the hand and foot are now in roughly the same position, indicating that the motion is now smooth.

repetitive motion videos (Section 3), which fills in missing details between frames via frame reordering, and for which we show up to 120x slowdown;

- A corrective flow formulation (Section 3.2), based on optical flow, which allows us to smoothly combine frames from non-adjacent repetitions into a single coherent sequence; and
- An image warping algorithm (Section 3.3), which allows us to produce the final video sequence using our corrective flow.

2 Related Work

2.1 Frame interpolation

Slowing down videos typically involves some form of frame interpolation, wherein new frames are created in-between consecutive existing frames in the original video. Most classical frame interpolation methods rely on dense pixel correspondences between neighboring frames such as optical flow [4], [5]. Some early work in optical flow includes a global variational model introduced by Horn and Schunck [6], as well as a local formulation by Lucas and Kanade [7]. Many other methods have been based on these, some of which use feature matching to detect large motion [8], [9]. However, most optical flow algorithms cannot correctly handle large displacements or changes in lighting conditions, which in turn produce artifacts in the interpolated frames. Our method uses optical flow as part of its pipeline, but most large displacements between frames are removed by reordering (Section 3.1), provided the semi-repetitive motion in the input video is well sampled. There are other frame interpolation methods that do not rely on optical flow. Meyer *et al.* [3] use a phase-based method that is computationally efficient, but is also limited in the range of motion that can be handled. Recently, deep learning has been successfully applied to both frame interpolation [2], [10], [11], [12], [13], [14] and optical flow estimation [2], [12], [15], [16], [17], [18]. However, these methods still rely on creating novel intermediate frames using estimated motion, and suffer from artifacts when the amount of motion between frames is large. Also, given that these methods employ deep neural networks, they require large datasets for training and validation, and may not generalize well to unseen data.

2.2 Image warping

In the third phase of our pipeline (Section 3.3), frames must be forward-warped in order to smooth the output motion. Shade *et al.* [19] describe Layered Depth Images, for which splat-based warping is effective due to the back-to-front ordering guaranteed by epipolar geometry [20]. Mark *et al.* [21] perform view interpolation by triangulating several pre-rendered images, and warping the resulting meshes to a new viewpoint, resolving order and occlusions with per-pixel depth information. However, neither of these methods are appropriate for our pipeline because the frames we process do not have any camera information, nor do they have per-pixel depth. Baker *et al.* [4] describe a warping algorithm that involves splatting flow vectors onto an intermediate image, which Herbst *et al.* [1] extend to use bidirectional flows to better handle occlusions and ordering. As both of these warping methods are specific to frame interpolation, they require two temporally neighboring frames in order to resolve occlusions. In contrast, our pipeline does not produce any in-between images; warping is used to alter individual frames, not create novel frames at intermediate timestamps. We describe a triangulation-based forward warping algorithm. Our method addresses the aforementioned problems using video segmentation as a proxy for per-pixel depth, and inpainting to fill in gaps in

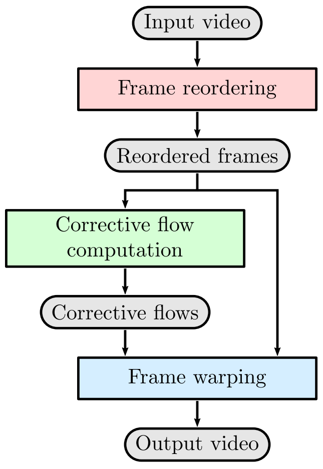

Fig. 2. Our video folding pipeline. The input video is first reordered into a single coherent non-repeating sequence. Corrective flows are computed for each frame, using neighboring frames in the reordered sequence. All frames are then warped along their corrective flows to produce the output video.

the output, without relying on consecutive frames from the source video.

3 Video Folding

The key idea behind our method is that each repetition within the input video is a different sampling of the same general motion. We would like to produce a novel repetition of this same motion, but with a much denser sampling, resulting in a video with significantly higher framerate than the original. This is done by interleaving all frames from each repetition, reordering them into a sequence consisting of a single repetition. We capitalize on the fact that during capture of semi-repetitive motion, we obtain different phase offsets of the same approximate motion, and by combining them we obtain the inter-frame motion more precisely.

Ideally, the motion being recorded is exactly periodic. In this case, provided that the frames are reordered correctly, no frame warping is needed as there are no inconsistencies between repetitions. However, it is crucial that the period of motion and camera framerate are relatively prime (i.e., out of phase). This ensures that each succeeding repetition is captured at some temporal offset which results in recording the motion in between frames of previous repetitions. Otherwise, if the period of motion is an exact integer multiple of the framerate, each repetition contains the same set of frames, and there is nothing to be gained by combining multiple repetitions.

In real life, such repetitive motion is rarely ever exactly periodic. For organic subjects such as people and animals, small perturbations in motion make each repetition unique. Even mechanical subjects may not exactly repeat. These unique repetitions cannot be simply combined by frame reordering alone. Warping must be applied to each frame to make it coherent with that of frames from other repetitions at near the same instant in time relative to the period of motion. To this end, we introduce a *corrective flow formulation* based on inter-frame optical flow, and describe a frame warping algorithm to produce the final output video.

Our approach consists of a pipeline (Figure 2) with three main stages: frame reordering, corrective flow calculation, and frame warping.

3.1 Frame Reordering

The first stage of our pipeline involves reordering frames based on the period of motion. Because our work focuses on organic semi-repetitive motion, repetitions will in general have slightly different lengths and content, and it is non-trivial to automatically determine a correct ordering. However, these video sequences can contain a very large number of frames, which makes manually sorting intractable. We describe a semi-automatic sorting method that requires a small amount of user input and produces a reasonable ordering of the video frames.

Before frame reordering can happen, it may be necessary to stabilize the input sequence. Video stabilization is performed in the presence of camera motion and other large global transformations throughout the sequence. In our current system, we detect features between frames and compute affine transformations to best align them. We leave to future work the integration of automatic video stabilization (e.g., [22]). The end goal is to ensure that global inconsistencies between repetitions are reduced. All further operations are performed on the post-stabilized frames.

Let I be the sequence of N frames in the original video (after stabilization), where I_i is the i^{th} frame in the sequence, and let t_i be the timestamp of I_i relative to the beginning of a repetition, with $1 \leq i \leq N$ and $0 \leq t_i < 1$. Then we want to produce a permutation J of I such that t_j is monotonically increasing with j, and with j as an index into J. In other words, frames in I are sorted according to their relative timestamp to produce J. The values of t_i are determined semi-automatically. Some user input is required to mark the first frame of each repetition (e.g., 5-30 simple frame selections in most cases). For every such marked frame I_k, we set $t_k = 0$. The number of frames n_r in each repetition r can be determined by counting the number of unmarked frames in between each marked frame (+1, for the initial marked frame). For simplicity, we assume that I_1 is marked; i.e., no unmarked frames occur before the first marked frame. Then each unmarked frame I_i is assigned the relative timestamp

$$t_i = \frac{i - k}{n_{r_i}}, \tag{1}$$

where $k < i$ is the index in I of the previous marked frame, r_i is the repetition in which frame I_i occurs, and n_{r_i} is the number of frames in repetition r_i. The frames of I are then sorted into J according to t_i. Figure 3 provides a visual explanation of this step.

We refer to t_i as a relative timestamp, but note that this is independent from a frame's physical timestamp given by the camera. While physical timestamps are separated by a fixed interval, relative timestamps depend on the duration of the repetition in which a frame occurs. It is not necessary for all repetitions to have the same duration; some repetitions may take longer to complete than others. The assignment of these relative timestamps effectively scales the interval of time between frames so that all repetitions have the same duration.

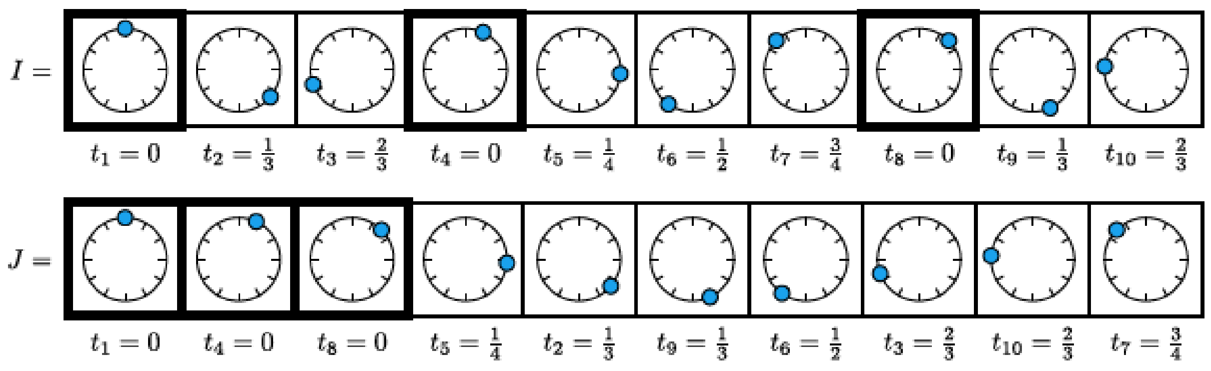
$I =$
$t_1 = 0$ $t_2 = \frac{1}{3}$ $t_3 = \frac{2}{3}$ $t_4 = 0$ $t_5 = \frac{1}{4}$ $t_6 = \frac{1}{2}$ $t_7 = \frac{3}{4}$ $t_8 = 0$ $t_9 = \frac{1}{3}$ $t_{10} = \frac{2}{3}$
$J =$
$t_1 = 0$ $t_4 = 0$ $t_8 = 0$ $t_5 = \frac{1}{4}$ $t_2 = \frac{1}{3}$ $t_9 = \frac{1}{3}$ $t_6 = \frac{1}{2}$ $t_3 = \frac{2}{3}$ $t_{10} = \frac{2}{3}$ $t_7 = \frac{3}{4}$

Fig. 3. Semi-automatic frame reordering. Marked frames (highlighted) represent the first frame of each repetition. The first and third repetitions have 3 frames each, while the second repetition has 4 frames. Top: the original video sequence. Bottom: the video sequence after semi-automatic sorting. Note that the subscript to t on the top and bottom sections refers to the index into I, not J, so as to illustrate the sorted order.

Since repetition lengths are measured in number of frames, there might be repetitions with the same length, which causes some frames to have the same relative timestamp. The relative order of these frames is undefined by the above semi-automatic sort. In order to break these ties, the user can choose to manually sort only the marked frames, which determines a sorted order for all repetitions. Any frames that have equal relative timestamps will then be sorted by this repetition order instead. This logic assumes the speed of motion is the same across all repetitions. If this is not the case, the resulting sequence will be only partially ordered.

While the above algorithm produces reasonable orderings for semi-repetitive motion videos, in practice the results are not perfect. After applying this sort to various input videos and comparing to a fully manual sorted ordering, the number of inversions is typically $\mathcal{O}(N)$, which can be remedied by manually swapping neighboring frames until a desired order is achieved. Note however that it may not be strictly necessary to fine-tune the order of frames in this way. Even if the frames are only partially ordered, the corrective flow is computed over a local neighborhood, so most inversions will have a negligible effect on the output frames. In the results section we show the robustness of our method to some ordering inaccuracy.

We highlight that depending on the content of the video there may not be a single correct ordering of frames. This may be caused by different areas of the subject moving at varying speeds over many repetitions. Take for example a video of a running horse (e.g., Figure 6). In each repetition, the front legs are moving at a slightly different speed than the back legs, causing the legs to be not perfectly synchronized. If the sorting order is defined on the front legs, then the back legs will appear to move out of order. While this behavior is a limitation of our method, we have found that the warping phase does a reasonable job at reducing such visual artifacts.

3.2 Corrective Flow Computation

The nature of organic, semi-repetitive motion is such that each repetition is slightly different from the others. Small

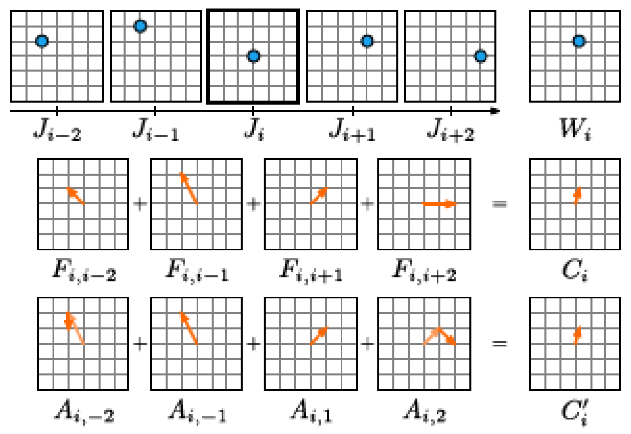

Fig. 4. Corrective flows. Top: video frames in the reordered sequence, with a tracked feature in blue. W_i is the result of warping frame J_i along either corrective flow. Middle: optical flow fields from frame J_i to each other frame in the neighborhood, with window size $s = 2$; C_i is the direct corrective flow, obtained with Equation (5). Bottom: accumulated flow fields from frame J_i to each other frame; C'_i is the accumulative corrective flow, obtained with Equation (4).

perturbations between repetitions become very noticeable when played as a single sequence. Thus even if a perfect ordering is achieved, the resulting sequence will not contain smooth motion. Instead, each frame must be warped to a configuration that changes gradually and naturally over the duration of the video. To this end, we define a per-pixel *corrective flow*, which we later use to warp individual frames to produce the final motion. This corrective flow is calculated using the dense optical flow between frames. Since computing optical flow is typically computationally expensive, we define two variations of corrective flow: *accumulative* and *direct* corrective flow. The former uses fewer optical flow fields than the latter, but is also less accurate.

Accumulative corrective flow sums optical flow fields throughout a local neighborhood. This formulation only requires optical flows to be computed between direct neighbors in the reordered sequence. In more detail, let $F_{i,j}$ be the per-pixel optical flow from frame J_i to frame J_j in the reordered sequence. That is,

$$J_j(\mathbf{x} + F_{i,j}(\mathbf{x})) = J_i(\mathbf{x}). \tag{2}$$

We define a recursive flow accumulation $A_{i,k}$ as the accumulated optical flow from frame J_i to frame J_{i+k}, as an approximation to the direct optical flow between the frames:

$$A_{i,k}(\mathbf{x}) = \begin{cases} A_{i,k+1}(\mathbf{x}) + F_{i+k+1,i+k}(\mathbf{x} + A_{i,k+1}(\mathbf{x})) & k < 0 \\ \mathbf{0} & k = 0 \\ A_{i,k-1}(\mathbf{x}) + F_{i+k-1,i+k}(\mathbf{x} + A_{i,k-1}(\mathbf{x})) & k > 0 \end{cases} \tag{3}$$

This formulation concatenates flow vectors along successive optical flow fields, sampled at the previous accumulated flow's destination, until the destination frame is reached. Optical flow fields are sampled using bilinear interpolation, clamped to the image boundaries. The accumulated flow is then used to calculate the accumulative corrective flow:

$$C_i(\mathbf{x}) = \frac{\sum_{k=-s}^{s} A_{i,k}(\mathbf{x})\, w(k)}{\sum_{k=-s}^{s} w(k)}, \tag{4}$$

where s is a scalar window size, and $w(k)$ is a weight function. Indices outside the interval $[1, N]$ wrap around, as the sequence is expected to be cyclic. The window size determines the range of neighboring frames in the reordered sequence that affect the corrective flow. This parameter should be proportional to the number of repetitions in the sequence, since we expect the surrounding frames to capture a similar instant in time relative to the period of motion. The weight function can be chosen to be uniform (i.e., $w(k) = 1$) or varying (e.g., a Gaussian curve so as to reduce the influence of frames that are further away from the current frame). The corrective flow C_i, when applied to frame J_i, produces an output frame whose content is warped to the average configuration over a local neighborhood. Because adjacent frames' corrective flows have overlapping contributions, this average configuration changes gradually over time, resulting in smooth output motion. However, errors in the individual flow fields quickly add up, leading to noticeable artifacts in the resulting corrective flows and warped results.

Alternatively, we also define a *direct corrective flow*, which does not accumulate optical flows between frames:

$$C_i(\mathbf{x}) = \frac{\sum_{k=-s}^{s} F_{i,i+k}(\mathbf{x})\, w(k)}{\sum_{k=-s}^{s} w(k)}. \tag{5}$$

This formula requires optical flow computation between each frame and a number of its surrounding frames. If the window size is large, this can mean that a great many flows must be computed, which may be impractical due to the computationally expensive nature of most optical flow algorithms. However, since there is no accumulation of errors in individual flow fields, this variation is much more accurate than the accumulative corrective flow. When possible, it is recommended to use direct corrective flows instead of accumulative corrective flows and such is what we use in all our reported examples. Figure 4 shows the difference between direct and accumulative corrective flows.

Whether using direct or accumulative corrective flow, the resulting flow field will contain errors. These errors stem from the underlying optical flow computations, which in general have trouble handling motion discontinuities [4].

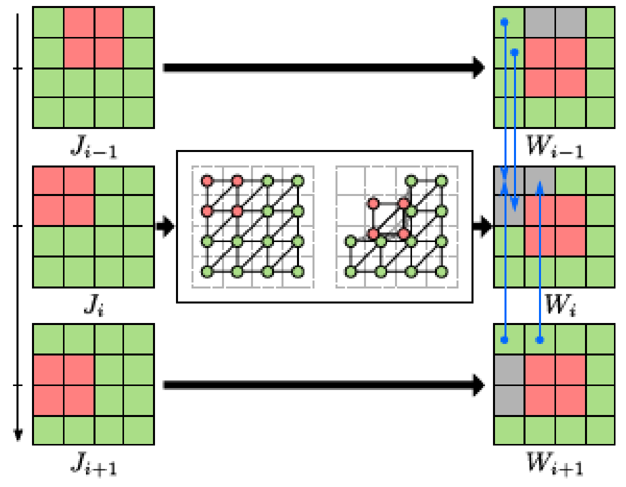

Fig. 5. Frame warping. Left: a selection of reordered frames. Foreground pixels are red, background pixels are green. Middle: warping steps for the center frame J_i. First the image is triangulated, followed by vertex translation along the corrective flow. Stretched and flipped triangles, shown with dashed edges, are culled before rasterization. Right: the frames after warping, during infilling. Holes are shown as gray pixels. The blue lines indicate which non-hole pixels from neighbor frames are used to fill the holes in W_i.

As both formulations of corrective flow combine multiple optical flow fields, these errors are expected to accumulate. We attempt to combat these errors by applying a bilateral filter [23] to the corrective flow fields prior to using them for warping. Qualitatively, the results appear to have fewer artifacts with the bilateral filter than without it.

3.3 Frame Warping

To produce the output video, each frame J_i in the reordered sequence must be warped along its corrective flow field C_i to produce a sequence W of warped frames W_i. This stage of the pipeline is challenging for the following reasons.

- The corrective flows are forward mappings, requiring a forward warping technique.
- The frames to be warped are not from contiguous time segments in the source video.
- Thus the usual strategy of using adjacent frames to help ameliorate occlusions, dis-occlusions, and holes is not possible – an alternative approach is needed.
- We do not assume per-pixel depth information, so it must be approximated.

3.3.1 *Forward warping*

The corrective flow defines a mapping from each pixel in the input image to a location in the output image. This mapping is not invertible – we cannot map each pixel in the output image to a location in the input image due to foldovers and disocclusions. If this were the case, warping would be a simple matter of assigning to each pixel in the output image the sampled color at the mapped location in the input image. Since we only have a forward mapping, we must employ a forward warping method to produce each warped frame.

Unlike inverse warping, forward warping is challenging for two reasons. First, not all pixels in the output image will be mapped to. Divergent flows create disocclusions, which appear as holes in the output image. These holes must be intelligently filled to maintain image quality. Second, multiple pixels in the input image might map to the same output image pixel. This is primarily caused by occlusions, when objects move to overlap one another in the output image. In these cases, special consideration must be given to which sample to display at the mapped location; i.e., which pixel from the input image is visible in the output image. Forward warping techniques can be found in frame interpolation [1], [4] and view interpolation [19], [21]. However, these methods use information from neighboring frames or per-pixel depth in order to resolve the aforementioned challenges. In our case, the warped image W_t is *not* an intermediate frame, nor do we have per-pixel depth information, so we cannot rely on the same occlusion reasoning logic.

We instead approximate per-pixel depth by using video segmentation prior to warping. Our implementation employs the semi-supervised technique of Marki *et al.* [24] to efficiently label pixels in all frames I_t of the original sequence (i.e., before reordering) as either foreground or background. This labeling L_t is a real-valued image, with pixels in the range of $[0, 1]$ (foreground to background), representing the label assignment as well as the confidence of such assignment. It is later used to resolve occlusions. Video segmentation is performed on the original sequence instead of the reordered sequence because the segmentation process relies on temporally cohesive regions within video frames. The reordered sequence is less cohesive over adjacent frames than the original sequence, since neighboring frames are from different repetitions.

To generate the warped image W_t, we first convert the input frame J_t into a regular triangle mesh. Each pixel is represented by a vertex located at the pixel coordinates, with a vertex color equal to the input pixel color. Additionally, each vertex is given a depth value according to the segmentation labeling L_t. A regular triangulation is constructed from these vertices, with each triangle defined by a counterclockwise vertex ordering. Then, each vertex is individually translated along its corrective flow vector to its destination location, without altering connectivity. The resulting mesh represents the warped frame, and is rasterized onto the output image. Figure 5 illustrates this step.

3.3.2 *Resolving occlusions and disocclusions*

Disocclusions in the warped frame appear as stretched triangles, which must be removed from the mesh before rasterization. A triangle is determined to be stretched if its area is greater than a multiple α of the original area, or if the ratio of its longest edge to its shortest edge is greater than a threshold β. In practice, we have found that setting $\alpha = 2$ and $\beta = 3$ produces good results. These stretched triangles are removed from the mesh prior to rasterization, creating holes that are later filled.

Occlusions appear as folds in the mesh, with occluding objects represented by overlapping groups of triangles. Some of these triangles are flipped, particularly those that span occluding object boundaries. These flipped triangles are detected by their clockwise vertex order, and are automatically culled with back-face culling enabled. The remaining front-facing overlapping triangles are also automatically resolved by rendering with depth testing enabled. Background triangles have a greater depth value assigned by the labeling, and thus will be occluded by foreground triangles. In some cases, overlapping triangles will share depth values, caused by a lack of true depth from the video segmentation labels. The resulting pixel color in these cases is ambiguous, determined by the rasterizer used. This is a limitation of our method, and is likely grounds for future work.

After rasterizing the warped mesh onto the output image, holes caused by disocclusions must be filled. This process uses color information from neighboring frames in the warped sequence (and *not* using neighboring frames in the original sequence as is typically the case), thus this phase must occur after all frames have undergone warping. As part of the warping process, we produce a binary mask M_t to represent holes in the output image:

$$M_t(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \text{ is not in a hole,} \\ 1 & \text{if } \mathbf{x} \text{ is in a hole.} \end{cases} \tag{6}$$

Holes are then filled as follows:

$$W_t = \neg M_t \times W_t + \frac{\sum_{k=-s}^{s} (M_t \wedge \neg M_{t+k}) \times W_{t+k}\, w(k)}{\sum_{k=-s}^{s} (M_t \wedge \neg M_{t+k})\, w(k)}. \tag{7}$$

In other words, each pixel that is in a hole in W_t is filled with the average color of pixels in neighboring warped frames at the same location, that are also not in holes in the neighboring warped frames, within some neighborhood window size s. In Figure 5, holes are shown as gray pixels, which are filled by neighboring warped frames. $w(k)$ is a weight function that can be uniform (i.e., $w(k) = 1$), or can be chosen to reduce the influence of far away neighboring frames (e.g., a Gaussian curve). In the above equation, $\times$, $+$, $\neg$, and $\wedge$ are all per-pixel operators.

As the corrective flow represents the flow to the average frame configuration within a neighborhood, most disocclusions caused by the warp are likely to be unoccluded in some neighboring frames. Thus, we expect the above algorithm to fill most holes caused by such disocclusions. Still, it is possible that some holes remain after this filling process. Any such remaining holes are filled by a simple outside-in approach, where pixels in holes are filled by the average color of neighboring non-hole pixels in the same image, until all holes have been filled.

The resulting warped sequence exhibits the same general motion as the input video, but at a much higher framerate while also maintaining smoothness of motion. However, if there are slight illumination changes between repetitions, the output sequence appears to flicker. To reduce this effect, we apply a smoothing operation on the output sequence, whereby each warped frame is averaged with its two neighboring frames. This substantially reduces the flickering effect, at the cost of a small degree of additional blur.

4 RESULTS

In this section, we show several results of our method. While we include pictures in this section, we encourage the reader to view our accompanying video for further results analysis.

The time to process a single video is dominated by the optical flow calculation. We experimented with several optical flow algorithms (i.e., [8], [15], [25], [26]), and found that MDP-Flow [8] overall produces output with the fewest visual artifacts, and hence is our preferred flow estimation algorithm when time permits. MDP-Flow is, however, computationally expensive, requiring on average 3.5 minutes to compute the optical flow between a pair of frames for the videos we have processed, typically around 512x512 pixels in size. Using direct corrective flows, this can quickly add up to days of computation time. We alternatively use PWC-Net [15], which takes 4 seconds per flow on average using a Nvidia GeForce RTX 2080. Depending on the number of repetitions used and on the number of frames per repetition, the total compute time for a video typically varies from around 10 to 30 hours using PWC-Net.

4.1 Framerate increase

Figure 1 demonstrates the application of our method. The top row displays a subset of frames from an input video in their original order. Three repetitions of the overall motion are visible, and in this example the positions of the left hand and right foot are shown throughout the sequence. After frame reordering (left), we obtain a non-repeating coherent sequence. These frames are from separate repetitions, but all occur at roughly the same point in time relative to the period of motion. The global positions of features within these frames are similar, but by zooming in on the hand and foot, you can see that there are notable positional differences between them. When played in a video, these small differences become very evident and distracting. The middle of the figure shows color-coded corrective flows for the zoomed-in areas of each frame. The hand/foot features are assigned different directions in each corrective flow field, corresponding to the displacement towards the average position of these features over a local neighborhood. The reordered frames are then warped along their corrective flows (right). In the closeups, the hand and foot are closer to the same areas in each frame. When played in a video, the resulting motion is smoother than before warping.

4.2 Frame reordering

During frame reordering, frames within the video sequence are sorted according to their approximate position within a repetition. However, since the motion is organic, it may be difficult or even impossible to achieve an exact ordering of frames. Despite this fact, the warping phase corrects inconsistencies between repetitions, including those caused by out-of-order frames. Thus, it is only necessary to obtain a partial ordering of the video sequence in order to obtain smooth results. Figure 6 demonstrates that even if the exact sorted order is unknown, the warping phase can produce reasonable results. Close-ups of a warped frame from three different orderings of a sequence are shown. The left-most frame shows the result of warping a completely manually sorted sequence. It is considered as close to the perfect order as possible. The other two frames show the result when the sequence is only partially ordered. As the sequence becomes less ordered, the quality degrades. These partial orderings are generated by starting from the manually sorted sequence, and introducing random swaps between neighboring frames. The algorithm for generating a partial

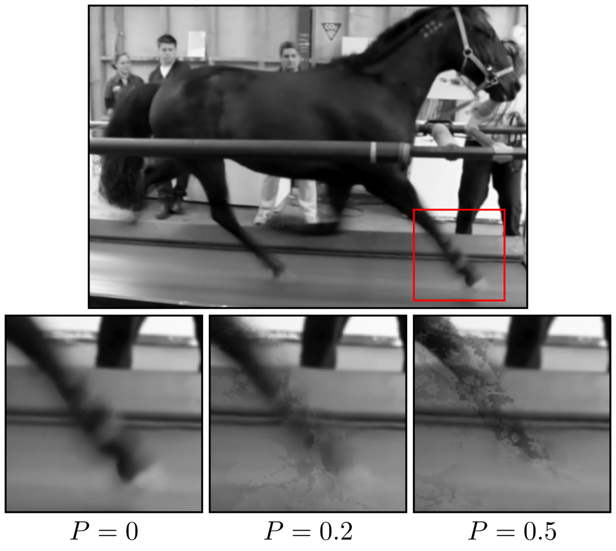

Fig. 6. The effects of a partial ordering on the final results. Top: a representative frame from a sequence of a horse on a treadmill. Bottom: closeups of a front leg after warping under different partial orderings. Left: the result of warping when the sequence is completely manually sorted. Middle: warping with 20% random adjacent swaps. Right: warping with 50% random adjacent swaps.

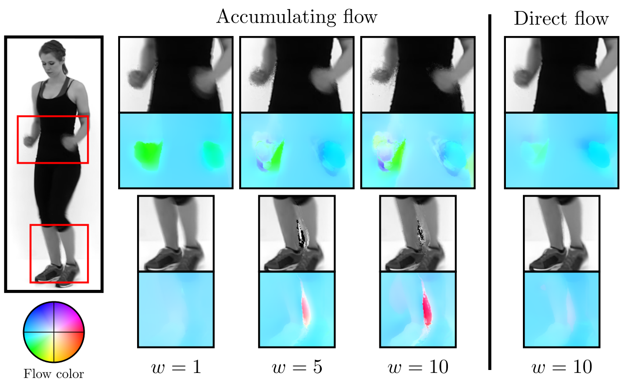

Fig. 7. Accumulative flow vs. direct flow. Accumulative flow is calculated with increasing window sizes to demonstrate the accumulation of errors. The closeups show the warped results paired with color-coded corrective flow images. Direct flow is calculated with the largest window size shown for accumulating flow, but there are far fewer artifacts.

ordering is as follows. N^2 Bernoulli trials are performed with a probability P. If a trial succeeds, then an index i between 1 and $N-1$ is uniformly randomly chosen and frame J_i is swapped with frame J_{i+1}.

For the middle frame in Figure 6, P is set to 0.2. The resulting frames contain more blur than those in the manually sorted sequence, but the motion in the video sequence is still smooth. In the right-most frame, P is set to 0.5. The images for this sequence contain even more blur, and the video quality degrades significantly.

4.3 Corrective Flow

Corrective flow is calculated to counteract small differences between repetitions. As previously mentioned, there are two variations of corrective flow: accumulative corrective flow and direct corrective flow. The former requires fewer optical

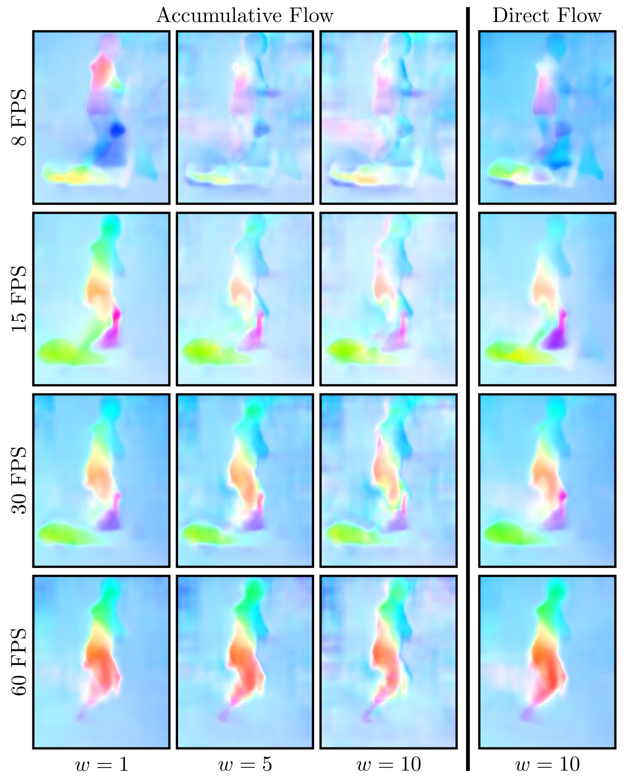

Fig. 8. Accumulative flow vs. direct flow at various framerates. For all framerates, accumulative flows exhibit bigger noise/irregularities due to chaining of multiple optical flow fields. Both methods present less irregularities at higher framerates but overall direct flow always presents less error. See main text for further explanations.

flow fields to be calculated, but is much less accurate than the latter. Figure 7 demonstrates the difference between the two flows. For accumulative flow, the errors accumulate with the window size w. This is because many optical flow fields are chained together during calculation of the corrective flow. At the largest window size shown, the flow fields have a lot of noise around the object boundaries. On the other hand, the direct flow is calculated at the largest window size shown, but does not exhibit such artifacts.

Figure 8 further shows accumulative and direct corrective flows but at various framerates, using the video shown in Figures 9 and 15. Here it can also be seen that, for all framerates, as the window size w increases, the accumulative flow field gains more noise and irregularities (e.g., in the background and the border of the moving foreground). For example, at $w = 10$ and 8 FPS there is considerably more noise/irregularities as compared to $w = 1$ and 8 FPS. At that same framerate, the direct flow appears less noisy. Since we know the rigid background is not moving (or at most is uniformly moving to the left as indicated by the blueish tone), any color variation in the background is incorrect. Similarly, the foreground objects (e.g., legs, arms) are smoothly moving so irregularities on their borders is an indication of incorrect flow. Further, we can observe in general at higher framerates the overall noise/irregularities is reduced (e.g., $w = 10$ at 60 FPS has less noise than at 8 FPS) but still direct flow presents itself better.

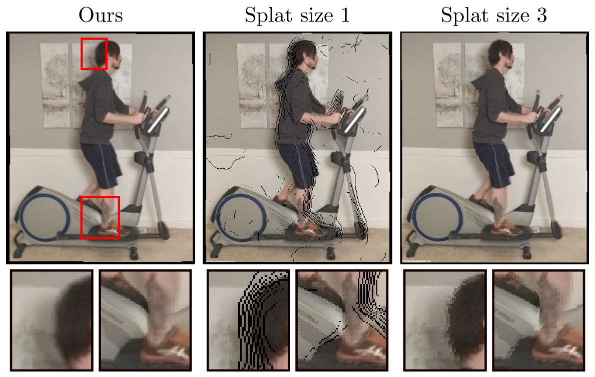

Fig. 9. Mesh-based warping vs. splatting. At smaller splat sizes, holes in stretched regions appear. Larger splat sizes fill in the holes, but produce more pronounced staircase artifacts around the motion boundaries.

4.4 Frame Warping

Many existing methods (e.g., [1], [4], [19]) employ a splatting-based approach for forward warping, while others (e.g., [21]) use a mesh-based approach. While both approaches are capable of forward-warping images, splatting a stretched region can produce holes in between mapped samples that must be infilled. Further, if such a region occludes another region, samples from the occluded one may be visible though these holes, creating errors for infilling. A mesh-based approach, on the other hand, treats the image as a continuous surface, thus holes in general do not appear unless triangles are removed. The surface representation also automatically resolves occluded regions by simply rasterizing with backface culling and depth testing enabled.

Figure 9 compares our warping technique to a simple splat-based approach, which draws colored pixels at each input pixel's destination according to its corrective flow vector, as in Equation 2. With 1x1 pixel splats, stretched regions create holes in the output image. Increasing the splat size to 3x3 pixels can fill in these holes, but the motion boundaries suffer from the larger splats, creating ”chunky” regions. Our mesh-based approach has neither of these limitations.

4.5 Comparison to frame interpolation

To evaluate the results of our method, we compare to two state of the art frame interpolation methods, DAIN [14] and SepConv [10]. Starting with a video of a bicyclist recorded at 240 FPS, we reduce the framerate to 8 FPS by removing 29 of every 30 frames from the sequence. Since normally there will be significant motion blur for low framerate videos, the remaining frames (keyframes) are blended with the adjacent 29 removed frames to simulate a longer exposure time. We also generate a non-blended version to study the effects of our method on sharp image content.

We apply our method and both frame interpolation methods to these reduced framerate videos. For frame interpolation, we generate 29 intermediate frames between each pair of keyframes to bring the output framerate back to 240 FPS. Since both of these methods generate a single interpolated frame halfway between both input frames, we apply each method recursively in a binary search pattern, using intermediate interpolated frames as input, to obtain the desired number of interpolated frames between each pair of keyframes. For our method, we limit ourselves to using 30 repetitions of the reduced framerate video, to bring the output framerate to approximately 240 FPS. Our actual output framerate is not exactly 240 FPS due to differences in speed between repetitions.

The result of each method is compared to the ground truth, which is the original 240 FPS video without discarding any frames. A simulated motion blur version is also generated by blending together every 30 frames, but without removing any from the sequence. This resembles a high framerate video with overlapping exposure times.

The comparison is done using the CW SSIM metric [27], which measures structural similarity between images while reducing the impact of localized spatial transformations. This is important because our method warps image content to its average position over many repetitions, thus its shape is slightly different from the original video and frame interpolation. Additionally, each repetition in the original sequence has a slightly different duration, while our method outputs a single repetition with the average duration. To ensure our output is synchronized with the ground truth, we compare to a single repetition with a longer duration than our output, and drop frames from the ground truth and both interpolation outputs such that the keyframes occur at the same frame number in all outputs.

Figure 10 shows the results of these comparisons. Without motion blur, neither frame interpolation method is able to accurately reconstruct the ground truth for intermediate frames that are far from keyframes (i.e., halfway between two keyframes). The interpolated frames have significant artifacts because the motion between the two keyframes is too large. Our results do not depend on distance between keyframes, and the visual quality is vastly improved because the reordered frames are warped to a far lesser extent. Near keyframes, the distortion in the frame interpolation outputs is greatly reduced. For the simulated motion blur results, there is still distortion in the frame interpolation outputs, but the image content is much more forgiving.

The CW SSIM values are plotted in Figure 11. The metric is calculated on a subregion of each frame, highlighted in Figure 10, that excludes largely-static regions. Each peak corresponds to a keyframe, at which the frame interpolation error is zero. Interpolation error is evident in the valleys between keyframes. While both frame interpolation methods show a regular oscillation in error, our method's performance is irregular. This is because each consecutive frame of our output is from a different point in the original sequence. While our results are never as high as frame interpolation near keyframes, there is overall less error in the areas between keyframes without simulated motion blur. With the simulated blur, all methods have lower error and the relative differences are less obvious. Nonetheless our method tends to show better values in between keyframes.

In both cases, with and without simulated motion blur, frame interpolation fails to accurately capture the motion that occurs between keyframes. Whereas the ground truth motion of the feet is smooth and circular, the frame interpolation outputs exhibit a piece-wise linear transition between feet positions in the keyframes. Still images are insufficient to observe this behavior; we refer the reader to our supplementary video for examples of this motion.

It is worth noting that, because the frame contents in our results are slightly different from the original video, a quantitative comparison will be biased towards frame interpolation results. The use of CW SSIM is intended to reduce this bias by allowing for localized spatial transformations, but it is very difficult to eliminate the bias completely. This metric also does not take into account temporal motion smoothness, and only looks at individual frame similarity.

4.6 Varying framerates and repetitions

We study the performance of our method given different framerates and number of repetitions as input. Implicitly, this analysis depends on the speed of the recorded motion, thus the absolute numbers reported are not universal – it does show, however, how repetitions can be used to effectively slow-down a video to rates beyond that possible using a single repetition. In Section 4.5 we reduce the framerate of a 240 FPS video to 8 FPS. In a similar manner, we also generate versions of this video at 2, 4, 15, 30, 60, and 120 FPS, and apply simulated motion blur to each. These videos are processed with our method, using the appropriate number of repetitions to restore the output framerate back to 240 FPS (e.g., for 15 FPS input, we use 16 repetitions).

For each reduced-framerate input video, we also alter the number of repetitions that are used, producing different output framerates. For example, with a 15 FPS input video, we use 2, 4, 8, and 16 repetitions to produce output framerates of 30, 60, 120, and 240 FPS, respectively. Outputs from all combinations of input framerate and number of repetitions are evaluated against their respective ground truth videos, which are generated as in Section 4.5, along with dropping frames to match the output framerates.

We compare our results for each of these combinations to a simple optical flow-based frame interpolation method, which behaves as follows. Adjacent keyframes are warped along their optical flows towards an intermediate frame, proportionally according to the interpolated frame's timestamp. These warped keyframes are blended proportionally to produce the intermediate frame. This frame interpolation method is applied to all combinations of input framerate and number of repetitions, producing videos with the same output framerate as our method and the ground truth.

Both our method and frame interpolation are evaluated quantitatively against ground truth using the CW SSIM metric [27]. Figure 12 shows the results of these comparisons. Each combination of framerate and number of repetitions is represented by a cell in the graph, colored according to the difference between the CW SSIM index for our results and that for frame interpolation, both using the ground truth as the baseline. A green cell indicates our result is closer to ground truth than frame interpolation. The CW SSIM index is calculated per frame, then averaged over the duration of the video to produce the values in the graph.

These values indicate that our method performs better at lower input framerates and with more repetitions. At higher framerates, with less motion in between frames, frame interpolation performs better than our method. At very low framerates, a single frame occupies a very large portion of the period, so most of the motion is lost, and

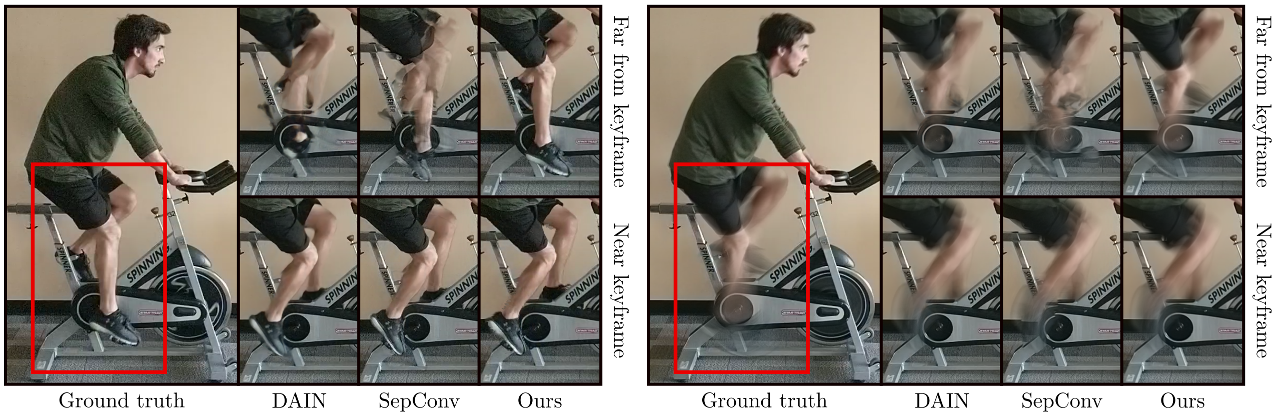

Fig. 10. Comparison to frame interpolation methods, with and without blur, for output frames near a keyframe and near the halfway point between keyframes. Left: results without simulated motion blur. The large motion between keyframes introduces errors to frame interpolation methods, whereas our results are mostly without artifacts. Right: results with simulated motion blur. All methods perform better, but the blur obscures most details. Note: the shown ground truth is for the frames corresponding to far from keyframe (top row, which represents frame 46 in Figure 11).

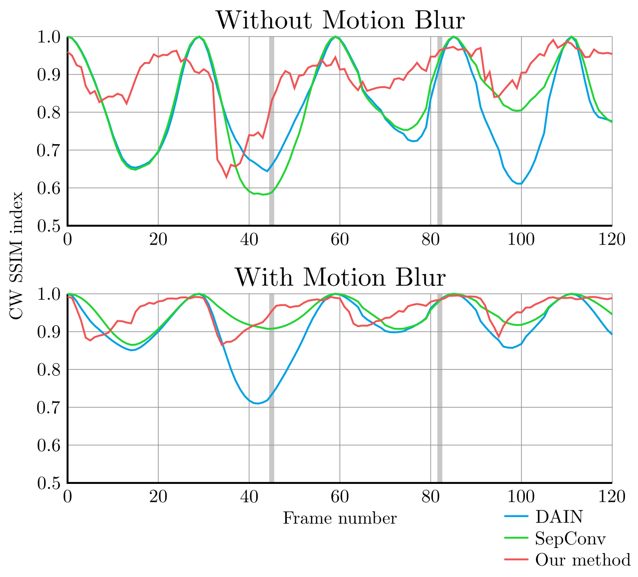

Fig. 11. Plot of CW SSIM against the ground truth, for a single output repetition, evaluated within the highlighted region in Figure 10. The peaks correspond to keyframes in the input, and the highlighted columns correspond to the compared frames in Figure 10. Our method performs comparably to frame interpolation in the presence of motion blur, and performs slightly better without motion blur.

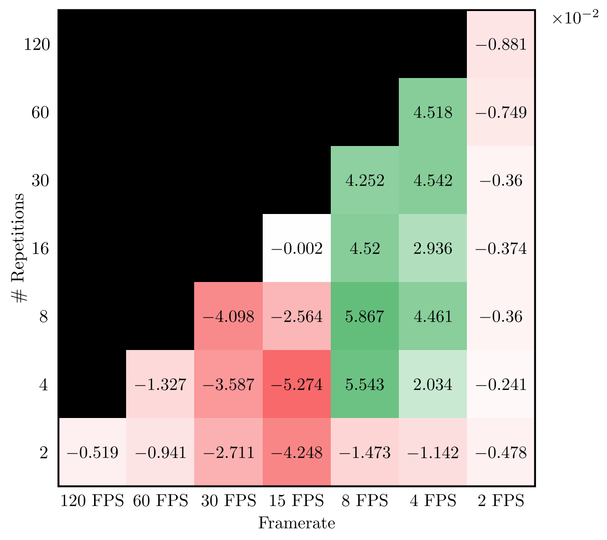

Fig. 12. Quantitative comparison of our method vs. frame interpolation, measured by CW SSIM. Each cell is a combination of framerate and number of repetitions, with the value as difference between our method and frame interpolation, scaled by 10^2 (i.e., green means our method is more similar to the ground truth, and red means it is less similar).

neither method performs well. Note that while these results seem to indicate that our method only works best at low framerates, we are actually able to produce high quality videos at much higher speeds. The black cells in Figure 12 would have higher output framerates than the original video, thus we have no ground truth datasets to compare to so we leave the cells empty.

We also provide a qualitative comparison to frame interpolation in the form of a user evaluation. 20 participants, aged 24 to 36, were presented 42 pairs of videos in random order, and asked to choose which video in each pair appeared more natural. Each pair contained our results alongside frame interpolation results, with two pairs for each combination of framerate and number of repetitions (one with our results on the left, and the other with our results on the right). Participants had no knowledge of which video corresponded to which result set.

Figure 13 shows the results of the user study. Each cell represents a combination of framerate and number of repetitions, with each cell listing the proportion of responses that favored our results over frame interpolation. Overall, users found that our results looked more natural than frame interpolation for lower framerates with many repetitions, but were less decisive about combinations of high framerates and fewer repetitions. We highlight that the quality of an output video is inherently subjective, and quantitative metrics are limited in their ability to capture such aspects. Thus, the results of the subjective experiment perhaps are more useful than the quantitative results.

We show results at different framerates and repetitions in Figure 14. On the left of each image-pair is an image from frame interpolation, and on the right is the same frame

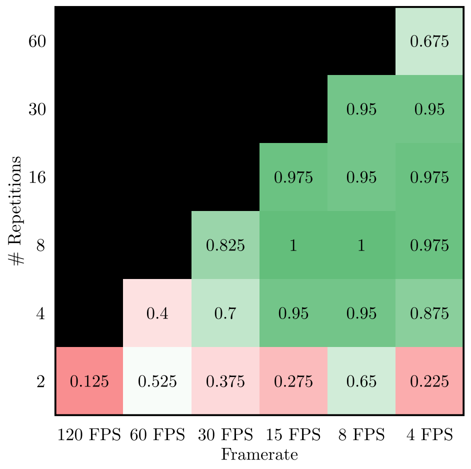

Fig. 13. Quality comparison of our method vs. frame interpolation, as evaluated by user study participants. Each cell represents a combination of framerate and number of repetitions, with the value being the proportion of user selections that favor our results over those of frame interpolation.

from our results. As framerate decreases, the amount of blur predictably increases. In the middle set, frame interpolation appears to have more noticeable artifacts than our method, despite scoring closer to the ground truth. In the right-most set, one can clearly see strong artifacts with frame interpolation due to optical flow failing to correspond between neighboring frames with a lot of motion. Our results, on the other hand, look much cleaner.

Additionally, we show a comparison to frame interpolation methods DAIN [14] and SepConv [10] for various framerates using another video (Figure 15). The original version of this video was filmed at 60 fps, and reduced-framerate versions were created by dropping frames. For each reduced-framerate version, we apply our method using 20 repetitions (i.e., 20x slowdown), and also apply frame interpolation on a single randomly-chosen repetition, producing 19 intermediate frames between each pair of keyframes. Figure 15 shows closeups of interpolation artifacts taken from a center intermediate frame (i.e., equidistant between two keyframes), along with the corresponding warped frame produced by our method. The frame interpolation results display visual artifacts, especially at lower framerates, due to the large motion in between keyframes, whereas our results appear to have fewer artifacts at all framerates.

4.7 Background stabilization

While this work focuses on ensuring the subject undergoes smooth motion throughout the sequence, a side effect of warping is that the background becomes stabilized. Especially in the cases of hand-held cameras, the background can move relative to the subject. Since frames from many different repetitions are reordered, it is likely that consecutive frames in the reordered sequence will have very different camera positions, and the background will appear to shift significantly. Because optical flow is a dense mapping, the corrective flow will also account for the movement of the background. Thus, after warping, the background will stabilize, even if it moves independently from the object of focus. Figure 16 demonstrates this property by comparing blended frames from before and after warping. Before warping, two consecutive frames display the shift in the background, while after warping, the same two frames are aligned.

4.8 Applications

The proposed technique lends itself to some interesting applications. The most obvious would be playback of semi-repetitive videos in super slow motion. This could be to achieve a particular impact on the viewer, or to assist in scientific analysis of fast, repetitive processes (e.g., movement of humans, wing motion of a hummingbird).

Another side-effect of applying our method is that the resulting video exhibits seamless, infinite looping. While the original video may have many repetitions, there is often a sharp discontinuity between the last and first repetitions when played repeatedly. The output of our pipeline, however, has no such discontinuity because all repetitions are merged into one. This can be useful if the intended application requires repeating the video for an undetermined amount of time (e.g., in video games).

There is also potential for our method to assist in video motion deblurring. Deblurring is considered to be an ill-posed problem because frequency information is lost in the blurred frames. However, Agrawal *et al.* [28] have shown that an invertible point-spread function (PSF) can be constructed from multiple exposures of the same motion by varying the exposure time. In a similar fashion, a sharp frame can be recovered from multiple blurry exposures that contain that same instant in time. If we consider each repetition of an input video to cover the same interval in time, then frames from different repetitions overlap in time, allowing us to recover sharp frames corresponding to the overlapping time intervals. Figure 17 shows preliminary results of such a video deblurring method. However, this only works if the motion is exactly repetitive. If the motion is only semi-repetitive, such as the videos used in this work, then the overlapping exposures contain different motions, making deblurring more challenging. Our technique may enable deblurring of such semi-repetitive videos, since all repetitions are combined and warped, effectively creating exactly repetitive motion. This is potential future work.

5 Limitations and Failure Cases

Our approach is mainly limited by the amount and type of repetitions in the input video as well as the ability to compute an optical flow. If the motion is not repetitive enough, or there are significant changes in lighting or background, our method will not produce a high-quality output. There is also an issue if the motion is *exactly* repeating, without any phase offset between repetitions (i.e., each repetition consists of the exact same set of frames repeated). In this case, there is nothing to be gained from reordering and warping, since there is no extra information to take advantage of. Our method also depends on the quality of the optical flow between adjacent frames in the reordered sequence.

Additionally, we are currently constrained to a single period of motion within a video. Multiple objects simultaneously undergoing repetitions of different lengths are problematic, because the duration of repetition determines the order of frames in the output video.

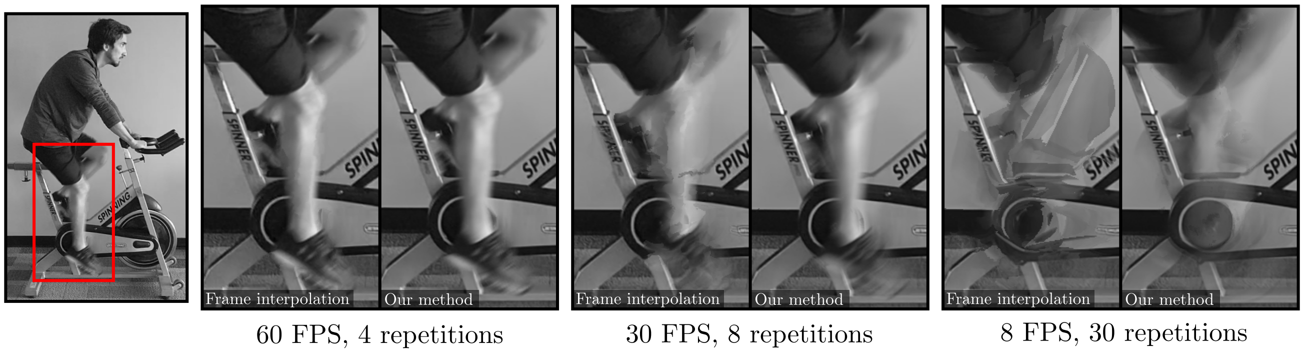

Fig. 14. Frames comparing our results to frame interpolation using 60 FPS and 4 repetitions (left), 30 FPS and 8 repetitions (middle), and 8 FPS and 30 repetitions (right).

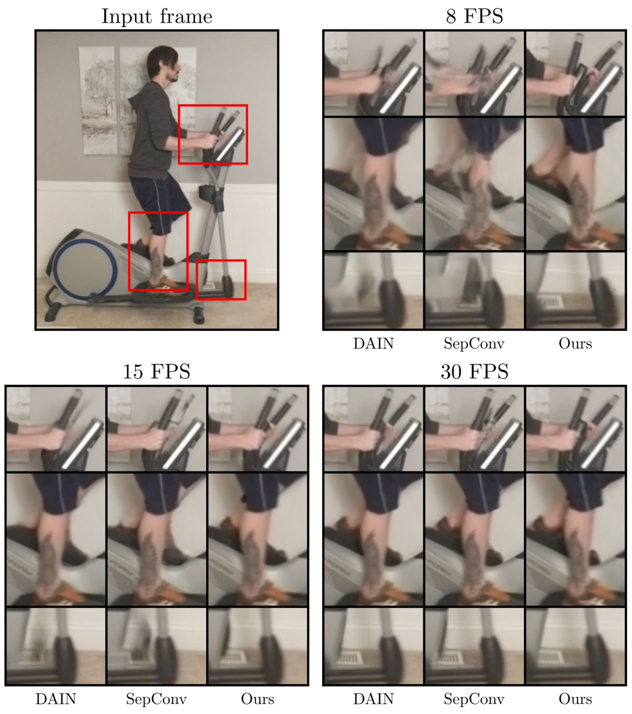

Fig. 15. Comparison to frame interpolation methods at various framerates. Visual artifacts in the frame interpolation results can be clearly seen at lower framerates (8 FPS), while at higher framerates (15 and 30 FPS) they start to disappear. Artifacts in our results are less noticeable at all framerates.

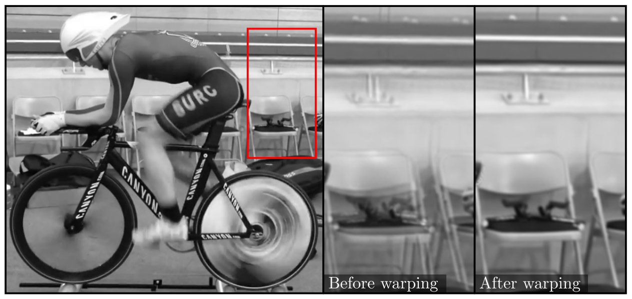

Fig. 16. Background stability. Closeups show blends of two frames, focused on a portion of the background. Before warping (left closeup), the background is shifted due to different camera positions between repetitions. After warping (right closeup), the background is aligned.

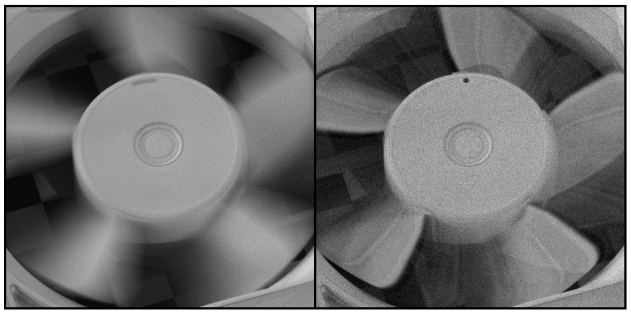

Fig. 17. Deblurring. A blurry video of a spinning fan (left) is deblurred using overlapping time segments (right).

6 Conclusions and Future Work

We have presented a method to transform semi-repetitive videos into high-framerate versions that can be played back in slow motion. As opposed to existing techniques that rely on frame interpolation, our method only uses existing frames, capitalizing on the extra motion information gained by repeated movements. We achieve extremely slow motion by reordering all repetitions from the original video into a non-repeating coherent sequence, and then warping individual frames such that the resulting motion remains smooth. Our pipeline requires minimal user input, and is shown to recover motion details that are lost with existing methods. We have demonstrated the effectiveness of our technique through quantitative and qualitative experiments.

As future work, we foresee several items. For instance, a deep learning based warping method might assist in providing smoother motion output. Further, although we use current video stabilization methods, further stabilization between the different repetitions can be improved. Finally, we may investigate the possibility of multiple repetition periods, involving motion segmentation, as well as the previously mentioned video deblurring extension.

Acknowledgments

This work is funded by NSF 1816514, 10001387, and NSF 1835739, and Manuel M. Oliveira acknowledges CNPq-Brazil fellowships and Grants 312975/2018-0 and 423673/2016-5.

References

[1] E. Herbst, S. Seitz, and S. Baker, "Occlusion reasoning for temporal interpolation using optical flow," Department of Computer Science and Engineering, University of Washington, Seattle, WA, Tech. Rep. UW-CSE-09-08-01, 2009.

[2] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, "Super slomo: High quality estimation of multiple intermediate frames for video interpolation," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[3] S. Meyer, O. Wang, H. Zimmer, M. Grosse, and A. Sorkine-Hornung, "Phase-based frame interpolation for video," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1410–1418.

[4] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, no. 1, pp. 1–31, 2011.

[5] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43–77, 1994.

[6] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[7] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, vol. 2, 1981, pp. 674–679.

[8] L. Xu, J. Jia, and Y. Matsushita, "Motion detail preserving optical flow estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1744–1757, 2012.

[9] T. Brox and J. Malik, "Large displacement optical flow: descriptor matching in variational motion estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 3, pp. 500–513, 2011.

[10] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive separable convolution," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 261–270.

[11] S. Niklaus and F. Liu, "Context-aware synthesis for video frame interpolation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[12] Z. Liu, R. Yeh, X. Tang, Y. Liu, and A. Agarwala, "Video frame synthesis using deep voxel flow," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1742–1750.

[13] S. Meyer, A. Djelouah, B. McWilliams, A. Sorkine-Hornung, M. Gross, and C. Schroers, "Phasenet for video frame interpolation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[14] W. Bao, W.-S. Lai, C. Ma, X. Zhang, Z. Gao, and M.-H. Yang, "Depth-aware video frame interpolation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[15] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[16] G. Long, L. Kneip, J. M. Alvarez, H. Li, X. Zhang, and Q. Yu, "Learning image matching by simply watching video," in *Proc. European Conference on Computer Vision (ECCV)*, 2016, pp. 434–450.

[17] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2758–2766.

[18] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[19] J. Shade, S. Gortler, L.-w. He, and R. Szeliski, "Layered depth images," in *Proc. SIGGRAPH '98*. ACM, 1998, pp. 231–242.

[20] L. McMillan and G. Bishop, "Head-tracked stereoscopic display using image warping," in *Stereoscopic Displays and Virtual Reality Systems II*, vol. 2409. International Society for Optics and Photonics, 1995, pp. 21–31.

[21] W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3d warping," in *Proceedings of the 1997 Symposium on Interactive 3D graphics*. ACM, 1997, pp. 7–ff.

[22] J. Yu and R. Ramamoorthi, "Selfie video stabilization," in *Proc. European Conference on Computer Vision (ECCV)*, 2018.

[23] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 1998, pp. 839–846.

[24] N. Märki, F. Perazzi, O. Wang, and A. Sorkine-Hornung, "Bilateral space video segmentation," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 743–751.

[25] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Scandinavian conference on Image analysis*. Springer, 2003, pp. 363–370.

[26] M. Drulea and S. Nedevschi, "Total variation regularization of local-global optical flow," in *Intelligent Transportation Systems Conference (ITSC)*, 2011, pp. 318–323.

[27] Z. Wang and E. P. Simoncelli, "Translation insensitive image similarity in complex wavelet domain," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2, 2005, pp. ii–573.

[28] A. Agrawal, Y. Xu, and R. Raskar, "Invertible motion blur in video," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, 2009, p. 95.

Chris May is currently a PhD student at Purdue University as well as a graduate research assistant. He received a BS degree in computer science from Purdue University West Lafayette in 2013. His research interests include computer graphics, image processing, 3D reconstruction, and urban modeling.

Manuel M. Oliveira is a Full Professor at the Federal University of Rio Grande do Sul (UFRGS) in Brazil. He received his Ph.D. 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 cover most aspects of computer graphics, especially in the frontiers among graphics, image processing, pattern recognition, and vision (both human and machine). He is an associate editor of ACM TOG and a former associate editor of IEEE TVCG and IEEE CG&A.

Daniel Aliaga is an Associate Professor of Computer Science at Purdue University. He obtained his Ph.D. and M.S. degree from UNC Chapel Hill and his B.S. degree from Brown University. Dr. Aliagas research is primarily in the area of 3D computer graphics but overlaps with computer vision. His research is in the multi-disciplinary area of urban inverse modeling and design, codifying information into images and surfaces, and visual computing frameworks including high-quality 3D acquisition methods. To date Prof. Aliaga has over 110 peer reviewed publications and has chaired and served on numerous ACM and IEEE conference and workshop committees.