Animating Still Natural Images Using Warping

THI-NGOC-HANH LE, National Cheng-Kung University, Taiwan, Republic of China
CHIH-KUO YEH, School of Computer Science and Software, Zhaoqing University, China
YING-CHI LIN and TONG-YEE LEE, National Cheng-Kung University, Taiwan, Republic of China

From a single still image, a looping video could be generated by imparting subtle motion to objects in the image. The results are a hybrid of photography and video. They contain gentle motion in some objects, while the rest of the image remains still. Existing techniques are successful in animating such images. However, there are still some drawbacks that need to be investigated, such as too-large computation time necessary to retrieve the matched videos or the challenges of controlling the desired motion not only in terms of a single region but also in terms of consistency in regions. In this work, we address these issues by proposing an interactive system with a novel warping method. The key idea of our approach is to utilize user's annotations to impart motion to certain objects. With two proposed phases in terms of preserve-curve-warping and cycle warping, a looping video is generated. We demonstrate the effectiveness of our method via various experimental challenging results and evaluations. We show that with a simple and lightweight method, our system is able to deal with animating a still image's problems and results in realistic motion and appealing videos. In addition, using our proposed system, it is easy to create plausible animation using simple user annotations without referencing the video database or machine learning models and allows ordinary users with minimal expertise to produce compelling results.

CCS Concepts: • **Computing methodologies → Image manipulation;**

Additional Key Words and Phrases: Animation, still images, cycle warping, preserve-curve-warping

ACM Reference format:
Thi-Ngoc-Hanh Le, Chih-Kuo Yeh, Ying-Chi Lin, and Tong-Yee Lee. 2023. Animating Still Natural Images Using Warping. *ACM Trans. Multimedia Comput. Commun. Appl.* 19, 1, Article 4 (January 2023), 24 pages.
https://doi.org/10.1145/3511894

1 INTRODUCTION

A scenery picture composed of animation objects is pervasive in nature. From a single still image, a looping video could be generated by imparting subtle motion to objects in the image. The results are a marvelous hybrid of photography and video. They contain gentle motion in some objects,

The authors would like to thank the reviewers for the many constructive comments that help improve the paper. This work was supported in part by the Ministry of Science and Technology (under nos. **111-2221-E-006-112-MY3** and 110-2221-E-006-135-MY3), Taiwan, Republic of China and the Key Area Research Program of Universities in Guangdong Province (Nature science), China (under no. 2020KQNCX095).
Authors' addresses: T.-N.-H. Le, Y.-C. Lin, and T.-Y. Lee, National Cheng-Kung University, Tainan, Taiwan, Republic of China; emails: {ngochanh.le1987, yclin0017}@gmail.com, tonylee@mail.ncku.edu.tw; C.-K. Yeh, School of Computer Science and Software, Zhaoqing University, China; email: simpson.ycg@gmail.com.

1551-6857/2023/1-ART4 $15.00
https://doi.org/10.1145/3511894

Fig. 1. (a) User's annotation (green curves) is needed in our system to create appealing result and (b) user's annotation (arrows) is needed in Plotagraph [26].

while the rest of the image remains still. Producing such appealing images is not only an interesting research field but also a potential industry application.

This attractive research topic has been early introduced by various approaches, such as synthesizing video texture from a source video [4, 7, 29] and example-based approaches [11, 22–24, 27]. However, these approaches have two main drawbacks. First, these systems need a huge computation to retrieve appropriate video candidates from the database. And if the retrieved results are not adequate, then they further run a refinement step. Second, such sample-based systems are restricted to the domain, and thus the animated objects and kinds of motions generated by their systems are limited.

In recent years, some researchers [5, 12, 18] have used deep learning–based approaches to automatically animate still images. More appealing animating results are generated by these systems. However, the motions in the animating results and the animated regions are under the control of the network in such systems. Their results suffer from some shortcomings. Unnatural distortions occur in Endo et al. [5]'s results. The animated regions and static regions are not fully recognized in Holynski et al. [12]. This phenomenon is represented in the following: (1) some animating regions are frozen instead of animating, and (2) the motion in a certain region affects the surrounding regions. Moreover, in these works, the lack of any reliance on training dataset prevents certain methods from animating arbitrary scenes.

Besides the aforementioned research, some commercial applications have been released for photo animation in recent years, such as Pixaloop [25], and Plotagraph [26]. With these applications, we can easily animate our images. Nevertheless, there exist some limitations in these applications. In Plotagraph, a large number of user annotations are needed to control the motion in the results to be realistic (as shown in Figure 1(b)). This might be tailored to the artists rather than ordinary users with minimal expertise to turn still content into state-of-the-art looping videos. In Pixaloop, the animating regions are specified by users' brushing. Hence, the resultant boundaries between the animating regions and static regions are usually abstract. This yields boundary artifacts in the results. Besides, the curvature of the strokes painted by users is mostly damaged in the animating results.

In this article, we present an interactive system that animates still images with a novel warping method. Our proposed technique warps pixels directly instead of neural features. This enables our system to have more predictable results and to be faster to run. Our system takes as input an arbitrary still image. The output is a realistic and looping animated video. We demonstrate that, by properly providing the scenes that adhere to fluid motion(some adequate strokes), we can efficiently generate appealing animated video while preserving the curvature of the given strokes. With these functionalities, our system allows ordinary users with minimal expertise to easily explore compelling results.

To achieve that, we propose a novel method, namely **Preserve-Curve-Warping (PCW)**, which is especially useful for preserving the curvature of the input strokes in the results. Besides, to reduce boundary artifacts of animated and static regions, we incorporate an explicit matting step to extract the animated regions and animate them independently. Moreover, we provide further functionality that discards the static pixels that are mixed up in the animated region. This certainly eliminates the ghost artifact caused by such pixels and enables our system to succeed with challenging images. To validate the effectiveness of our animating scheme, we test it with a rich variety of challenging cases. Realistic and appealing results are obtained. We also compare our results to those of the prior methods and the existing commercial applications to demonstrate our capability in animating still images. In addition, we further present the objective evaluation of the quality of our generated results. In summary, our main contributions are listed as follows:

- We develop an interactive system that efficiently animate a single still image without manual refinement or reference videos.
- We propose a novel warping method, *Preserve-Curve-Warping*, that is especially useful for preserving the curvature of the input strokes in the results. This benefit enables our system to animate various objects with flexible animating style.
- Our proposed animating scheme could generate a looping video and ease the discontinuity artifacts occurring between loops compared to previous works [3, 12].
- Various experiments illustrate that our method is more accessible and can have more predictable results. This allows ordinary users with minimal expertise to explore compelling results.

2 RELATED WORK

"Motion without movement" is a technique introduced in Freeman et al. [7] to move displaying patterns continuously without changing their positions. With a similar goal to this technique, Schödl et al. [29] propose a new type of medium, called a *video texture*. In this method, the video texture is synthesized from a finite set of images by randomly rearranging (and possibly blending) original frames from a source video. Unlike this method, a video texture is created from a video source; Chuang et al. [4] attempt to animate a natural scene in a single image by creating video textures from a static image rather than from a video source.

Example-based approaches can reproduce realistic motion without complex parameters by directly transferring reference videos [11, 22–24, 27]. In these works, the abundance is given to the sample retrieval process. Thus, they focus on proposing a good algorithm for the retrieval sector. Their results may be unnatural if they fail in retrieving an appropriate video that is similar to the region image specified by the user. There are some drawbacks in these works, such as they lack of various types of fluid motion. Thus, the motions in their results are not dynamic. Besides, infinite loop video is also a limitation.

Related to the infinitive loop videos, Bhat et al. [3], Liao et al. [16], and Schödl et al. [29] use different ways to generate infinitive loop output. In Schödl et al. [29], the video texture is an infinitely varying stream of video images. The video texture is synthesized from a finite set of images by randomly re-arranging original frames from a source video. Bhat et al. [3] synthesize new video by manipulating flow lines. In this approach, particle texture moves along the input flow line. They define a matrix to represent the particles and their positions on the flow line in frames. To reduce the discontinuity between output frames, they modify the matrix by changing the entries in the original matrix. A progressive video loop is a major issue in Liao et al. [16]. Through an optimization framework, they define a spectrum of loops with varying levels of activity, from completely static to highly animated [16].

Most recently, the work that first attempts to create a waterfall animation on a still waterfall image is introduced in Lin et al. [17]. However, this approach solely focuses on animating the waterfall. Besides, their system also needs a video sequence to extract a flow animation to form the animation.

Inspired by the evolution of deep learning–based approaches in recent years, Endo et al. [5], Logacheva et al. [18], and Holynski et al. [12] propose a novel network to animate a still image to a looping video. Endo et al. [5] introduce a self-supervised learning to animate a landscape. To generate a video from a still image, a training dataset of time-lapse videos is used in this study to learn the motion and the appearance in an outdoor image. This method can generate a rational animation at higher resolution and longer-term sequences than previous works. Yet, they mainly focus on landscape animations, especially of skies and water; the variety of types of motion is also a limitation in their approaches. DeepLandscape [18] extends the structure of StyleGAN to animate landscape images. Yet, their model is built for landscape videos and does not attempt to learn full temporal dynamics of videos. The method proposed in Holynski et al. [12] takes as input a single static image and produces a looping video texture. More realistic video are generated from their proposed animating system. However, there are still drawbacks in their results such as ghosts in the boundary of the animation region and still regions or that some animation regions are frozen in the final video.

Our approach does not rely on any video dataset to generate motions in the animation output. Instead, we let the users specify both the regions they want to be animated and the animation styles they desire to be applied to the specified regions. Treating the animation styles specified by the users as the sample, our system directly generates such a plausible animation output. While Endo et al. [5] focus on the animating landscape, Okabe et al. [22, 23, 24], Prashnani et al. [27], and Gui et al. [11] are limited in the various types of motion and need to combine their method with another method to adopt an infinite output, our system makes the impression by producing infinite videos with various types of single input.

3 METHODOLOGY

3.1 System Overview

The pipeline of our animating scheme is illustrated in Figure 2, which consists of four main steps: *Animating region extraction, Flow generation, Preserve-Curve-Warping*, and *Cycle warping*. The system gets as input an arbitrary still image. We first base on users' selections to segment objects from the input image. The segmented objects indicate the regions are to be animated. This process is called *Animating region extraction*. Each of them is then animated singly. To animate a certain region, our system allows the users to draw some strokes on the region to describe their expected animating style. Thereafter, the *flow generation* is proposed to calculate a dense motion map, which represents the motion weight at each pixel in the input image followed by the strokes. Eventually, two phases are proposed: *Preserve-Curve-Warping* plays a role to preserve the curvature of the motion in animating results as those of the users' strokes; meanwhile, *Cycle-warping* is used to ease the discontinuity at the gap of two loops in rendering the final looping video.

3.2 Animating Region Extraction

The primary goal in this step is to define the **Animating Regions (AR)** in the given still image. Our approach determines the ARs based on users' interests. That is, we let the users interactively select the regions that they are to make animating by brushing along the boundary of such regions (Figure 3(a-1)). Then, we adopt the image matting method Gastal and Oliveira [9] to extract the ARs from other image compositions. We choose this method to extract the ARs, because brush-painting is just a roughly demarcation and Gastal and Oliveira [9]'s method can accurately extract sufficient

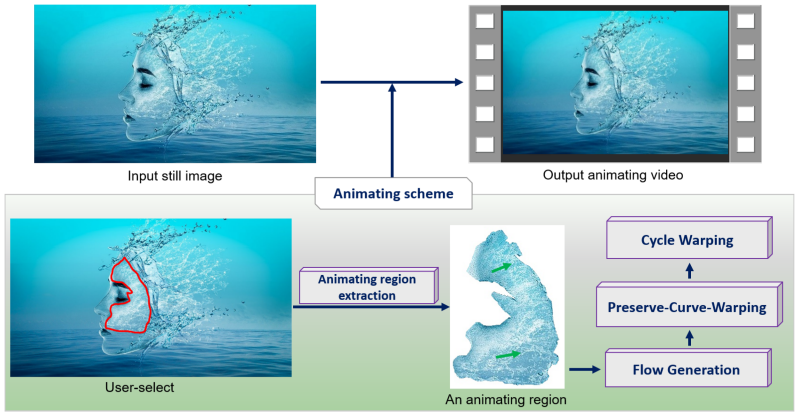


Fig. 2. Our system overview of generating animating video from a still image.

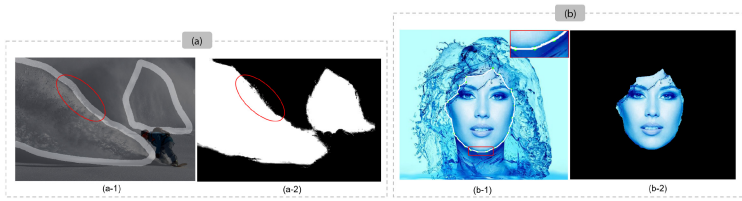


Fig. 3. (a-1) Brushes are painted to demarcate the animation region. (a-2) Result after employing image matting method Gastal and Oliveira [9]. (b-1) The set of points (green circles) is specified the contour of the object. (b-2) The object, which is still in the final result, is extracted from image content.

information of the ARs. The output of this step is an alpha matte (as in Figure 3(a-2)). Such regions are later animated independently by our proposed animating algorithm. At first glance, this scheme might be similar to the pipeline of previous works [4, 24] and the existing commercial applications [25, 26]. We argue this by pointing out at least two drawbacks of the technique that is used in this step of the related work pipeline. First, the motion of a certain AR affects neighbor regions, since the ARs and the still regions are not isolated. Second, since the user-select regions are used to animate directly, in the cases that the ARs compose of complicated textures (such as the example shown in Figure 4(b)), obvious ghost artifacts occur. To address these issues, our AR extraction process is designed differently from prior techniques as follows.

To avoid the first phenomenon, we do both animate ARs independently and isolate still objects/regions from the image content. The reason is that due to the content structure of the input images as well as the users' interest, the animation regions and the still objects might be overlapped, as in Figure 3(b). To isolate these objects from the image content, the users specify in advance a set of contour points of the object. These points are used to construct the boundary of the still object. Thereafter, an interactive tool, namely "Intelligent Scissors" [20], is adopted to find the minimum cumulative cost path between the starting point and a set of target points and finally extract the object from the other animation regions. With this technique, we not only overcome the mentioned drawback but also achieve good performance at the boundary of neighbor regions in the animating video result.

For the second phenomenon, we propose a function that rather than directly using the ARs from users' selection, we filter them before animating (as the pipeline shown in Figure 5). In a certain

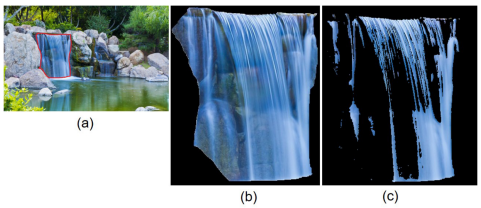


Fig. 4. Visual comparison of AR and filtered AR. (a) Input image, the AR and filtered AR of the region masked in red in the input image are presented in (b) and (c), respectively.

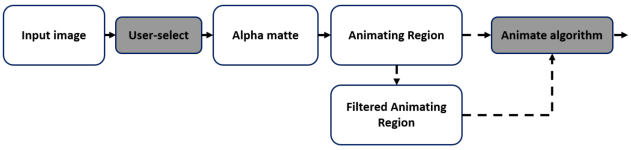


Fig. 5. The pipeline of animating region extraction. Once the animating region is defined, the user can either use it directly to animate or filter it before animating.

alpha matte $\mathcal{A}$, the foreground (pixel value of 1) indicates the animating region, and meanwhile the background (pixel value of 0) indicates the still objects. An animating region (denoted by $\mathcal{R}$) is a set of pixels that is obtained by mapping the foreground of the alpha matte back to the input image $\mathcal{I}$ as follows:

$$\mathcal{R}(x,y) = \begin{cases} \mathcal{I}(x,y), & \text{if } \mathcal{A}(x,y) = 1 \\ 0, & \text{if } \mathcal{A}(x,y) = 0 \end{cases}, \tag{1}$$

Accordingly, the filtered AR (denoted by $\mathcal{R}^F$) is formulated as

$$\mathcal{R}^F(x,y) = \begin{cases} \mathcal{R}(x,y), & \text{if } \mathcal{R}(x,y) \in \Theta \\ 0, & \text{otherwise} \end{cases}, \tag{2}$$

where $\Theta = (\theta_r, \theta_g, \theta_b)$ is the pixel color threshold whose three component thresholds are defined by users' adjustment. To do this, for an extracted animating region $\mathcal{R}$, our system allows the user to pick an arbitrary pixel color $\theta_{pick} = \{p_r, p_g, p_b\}$. We then use θ_{pick} as initial threshold to suggest the user's adjustment. Despite the picked pixel is still or animating one, the user can drag the bar to adjust the threshold Θ until he or she is satisfied with $\mathcal{R}^F$. In Equation (2), if a pixel in $\mathcal{R}$ does not meet the threshold, then it is assigned a pixel value of 0 in $\mathcal{R}^F$ (i.e., to be discarded). That is, such pixels are not used when we feed $\mathcal{R}^F$ to the animating process. Hence, the filtered AR differs the AR from the number of animating pixels. We show these results in Figure 4. In this example, we hope to discard the pixels of stone, because these pixels animated with the waterfall pixels produce an unrealistic result. We can see that applying Equation (2) on Figure 4(b) yields a filtered AR without stone pixels (Figure 4(c)). Therefore, in the cases that the selected animating regions composing of impure textures, the users can filter before animating. This function facilitates the user to obtain good results and removes the barrier of adopting various input images and animating objects. In our supplementary video, we present ablations to demonstrate the effectiveness of this functionality. Please visit our project website http://graphics.csie.ncku.edu.tw/AnimatingImages to access the videos for a better visualization.

3.3 Flow Generation

Similarly to most of the animating systems, we use the user's annotations to specify the flow direction. Two main differences are that (1) our system directly uses them to extract per-pixel flow direction without an expensive video retrieval process and (2) while the prior system from Okabe et al. [24] computes a dense flow field by interpolating the sparse information to deal with the sparse user strokes, we can impart the flow to the entire animating object by solving an optimization despite how sparse the strokes are.

On a certain animating region, our system lets users draw some strokes in advance. In the following, we call such strokes the **User-Specified Path (USP)**. To control the consistency of the flow in an animating region, we first generate a 2D grid to cover the input image. Each grid vertex v has coordinates (v_x, v_y) and a flow vector $\kappa(\kappa_x, \kappa_y) \in \mathbb{R}^2$. The value of κ represents the flow direction of the vertex. Our goal is to estimate the flow direction at the pixel level. We first rely on the USPs to solve the flows at grid vertices to achieve this. After that, we interpolate them to the pixel level. The reasons we use both grid level and pixel level are probably stated in three advantages: (1) easier to control flow at pixels to be consistent and smooth if there are multiple USPs painted on a region, (2) computational inexpensive, and (3) better control the curvature of the generated animation as the given USPs.

To find the flow vector at each vertex, we first divide the USPs into segments in the length as the size of the grid. This results in there exists only one point $p^u(p^u_x, p^u_y) \in USP$ in a cell. We reversely interpolate p^u to define κ at four vertices $(v_i, i = 1 \ldots 4)$ of the cell covering p^u. To do this, we calculate the positional correlation between p_u and the cell as

$$\begin{cases} w_x = \frac{p^u_x - v_{1x}}{v_{2x} - v_{1x}} \\ w_y = \frac{p^u_y - v_{1y}}{v_{4y} - v_{1y}} \end{cases}, \tag{3}$$

Accordingly, the bilinear interpolation weights [28] of four vertices (w_v) are respectively calculated as follows:

$$\begin{cases} w_1 = w_x \times w_y \\ w_2 = (1 - w_x) \times w_y \\ w_3 = (1 - w_x) \times (1 - w_y) \\ w_4 = w_x \times (1 - w_y) \end{cases}. \tag{4}$$

We then feed these weights to an optimization to estimate the flow vectors at all vertices. Initially, each vertex is assigned an equivalent flow vector (e.g., we set $\kappa_i = (1, 1)$ in our experiment). That is, all vertices have the same flow vector in both direction and magnitude. After optimization, the flow vectors of vertices are updated to make them consistent with the direction of the user-specified paths. Accordingly, the optimization is formulated as follows:

$$\min \sum_{i=0}^{\mathbf{N}} \sum_{j=0}^{\mathbf{N}_{nb}} \| \kappa_i - \kappa_j \|^2 + \lambda \left\| \sum_{i=1}^{4} (w_i \kappa_i) \right\|^2, \tag{5}$$

where $\mathbf{N}$ is the number of vertices in the input image, $\mathbf{N}_{nb}$ is the number of neighboring vertices that is defined by Moore neighborhood [30] (i.e., $\mathbf{N}_{nb} = 8$), and w_i is the interpolation weights of the vertices that cover the USP, and the constants λ we use as the smoothing terms. Equation (5) consists of two terms. The second term is to estimate the flow vectors at the vertices covering the USPs. Meanwhile, the first term imparts the estimated flow vectors in the first term to all the vertices. This is adopted by the flows of its neighboring vertices, i.e., a vertex flows to one of the

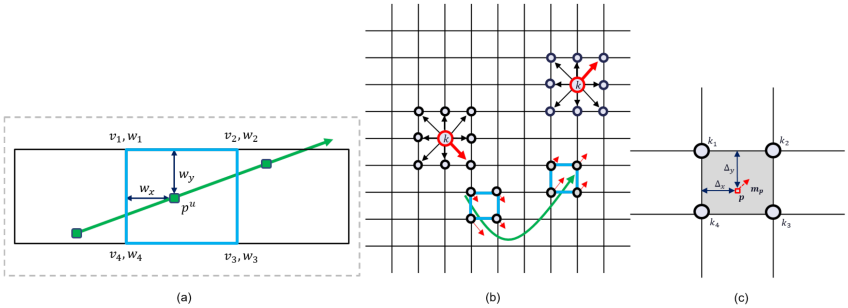


Fig. 6. (a) Reverse interpolation in Equation (3) and (4) with the user-specified path visualized in green stroke. Visualization of flow generation at grid vertices level (b) and pixel level (c). In (b), the user-specified path (USP) is shown with the green curve. Four vertices of the cell grid (in bright blue) that covers the composition points in the USP are assigned the weights (in red thin arrow). The vertices that are not close to the USP (v) are imparted by the optimization to the goal direction (red thick arrow). (c) Zoom-in of the interpolation in a cell to define the motion energy (m_p) at a certain pixel (p) once the flow direction of the vertices are found.

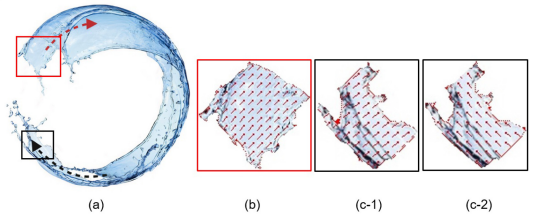


Fig. 7. Visualization of the flow generation. (a) An animating region with two user-specified paths. The first path is in red, and the second path is in black. (b) and (c-1) are the zoom-in of the patches (red square and black square, respectively). The flow in (c-1) is the same as in (b) that is imparted by the first user-specified path. The flow in (c-1) is changed as in (c-2) after drawing the second user-specified path.

eight neighboring directions. Without the second term, the flow directions are not imparted to all the vertices. And if there is no first term, then only the vertices covering the USPs impart the flow. In term of the grid size, our early experiments show that the grid size ranging from 8 to 12 yields sufficient smooth animation. In our experimental test, to obtain a good result, we set $\lambda = 10$ and $\mathrm{N}_{nb} = 8$.

As shown in Figure 7, the flow directions at vertices are optimized by Equation (5). In this example, the user draws two paths on the animating region (Figure 7(a)), which are visualized with red and black arrows, respectively. We can see that the flow in Figure 7(b) and Figure 7(c-1) are consistent with two paths that are specified on the corresponding region.

Once the direction values at each vertex are computed, we interpolate [31] them densely to all pixels. See Figure 6(c) for a visualization. We assume that pixel p belongs to a grid cell whose flow vectors at four vertices are denoted as $\kappa_1, \kappa_2, \kappa_3, \kappa_4$. These values are formulated with the fractional parts of the column and row location Δ_x, Δ_y to obtain the flow direction at p as the follows:

$$m_p = \Big\|(1-\Delta_y)\Big(\kappa_1(1-\Delta_x) + \kappa_2\Delta_x\Big) + \Delta_y\Big(\kappa_4(1-\Delta_x) + \kappa_3\Delta_x\Big)\Big\|. \tag{6}$$

We call flow value at pixel level in the term *motion energy* (m_p). Tracing this interpolation at pixels yields a dense motion map. This map represents the motion energy at every single pixel followed by the user-specified paths.

3.4 Animation Generation

With the obtained dense motion map, we design an animating algorithm to generate a final video. In designing such a novel animating scheme, we have two criteria. First, the animation in the output video should preserve the curvature of user-specified paths. Second, animation artifacts should be minimized in the generated looping videos. To achieve these, motivated by Cinemagraph,[1] we consider each animation motion composes of partitions of seamless motions. That is, each pixel in an animating region moves in a defined cycle and finally forms a looping video. We propose two phases in the terms of *"Preserve-curve-warping"* and *"Cycle warping."* The first phase plays a role in estimating the positions of pixels in the in-between frames. When the sequence is looped, there is a discontinuity between the two frames where the pixel abruptly moves backward from the terminal point to the starting point in the corresponding cycle. This discontinuity appears simultaneously for every pixel in the image making it a very noticeable artifact. The Cycle warping's job is to reduce such discontinuity and produce the final looping video.

3.4.1 Preserve-curve-warping. For a pixel p in an AR, the PCW algorithm is proposed to estimate the position of p at each timestep in a certain cycle. The term *cycle* in our study is defined with three properties: (1) *starting point* is the current position of p; (2) *duration* is assigned by the flow speed of animating, which is input by the user; and (3) *terminal point* is estimated such that the distance from it to the starting point is close to the duration of the cycle.

Future prediction has been studied in some methods [8, 13, 14, 19, 21, 32], but there are clear differences from the proposed algorithm. Mottaghi et al. [21] predict a pedestrian's future trajectory in a first-person video but require past frames to obtain an accumulative trajectory of the instance. However, some algorithms [14, 19] require both starting and end points to estimate long-term trajectories of instances. Mottaghi et al. [21] define motion scenarios and classify object instances in a single image into one of the pre-defined motion scenarios. Kim et al. [13] attempt to this challenge by requiring only a present frame. Therefore, these studies focus on scene classification rather than on future motion prediction on flow. Our PCW algorithm is a kind of warping that changes objects into another through a seamless transition. However, our primary goal is to preserve the curvature of the estimated trajectory as smooth as possible as in the user-specified paths, our PCW algorithm is thus differently designed to control the movement of pixels to meet this objective.

Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a sequence of timesteps, where n is the total number of frames in a cycle, and $\tau_i \in [0 \ldots 1]$. Now, we use PCW algorithm to estimate positions of pixels in the the frames. We call the estimated points in the term *control point* and the path that connects these control points is called *trajectory*. The procedure of PCW algorithm is presented in Algorithm 1. We explain the algorithm in detail as follows.

First, all the USP in an AR are converted to vector form as shown in Figure 8. We do not do this job naively to avoid the gap artifacts occur at two adjacent cycles. See Figure 9 for the visualization. For instance, pixel p_1 is animated with cycle 1 and pixel p_2 is animated with cycle 2. If the vectors are formed naively, then the tail of the next vector should be the head of the previous vector. Pixels always move from their current position to the estimated position. Hence, in the cases the estimated terminal position is not ideal, there is a significant gap between two cycles. For a certain USP, which composes of n composition points $\{\pi_j\}, j = 0, \ldots, n-1$, the tail ($t$) and head ($h$) coordinate of each vector a_i is respectively defined as

$$t_i = \begin{cases} \pi_0, & \text{if } i = 0 \\ h_{i-1} - \eta, & \text{if } i > 0 \end{cases}, \tag{7}$$

[1] http://cinemagraphs.com.

ALGORITHM 1: Preserve-Curve-Warping algorithm

Input: Set of USP $\mathbf{P} = \{P_i\}$
Output: Trajectory of a pixel p in a cycle.

for each path $\mathbf{P}_i, i = 1, \ldots, N_u$ **do** /* N_u *is the total USP in a certain region* */
  Initialize set $\mathbf{V}$; /* *this set stores all the vectors*/
  while $\pi_j < length(P_i)$ **do** /* π_j *is a composition point in* P_i*/
    Calculate tail (t) of vector by Equation (7);
    Calculate head (h) of vector by Equation (8);
    $\vec{a} \leftarrow (t, h)$;
    Add $\vec{a}$ to $\mathbf{V}$;
  end while
end for
Initialize the estimated trajectory $\mathbf{T}$; /* $\mathbf{T}$ *stores set of control points* $\mathbf{c}$ */
for each timestep $\tau_i, i = 0, \ldots, 1$ **do**
  for each $\vec{a_i}$ in $\mathbf{V}$ **do**
    Find a paired point q on $\vec{a_i} \in \mathbf{V}$ by Equation (9);
    Calculate weight of q by Equation (10);
    Calculate the resultant preserve-curve-translation of $h_i \in \vec{a_i}$ by Equation (12);
  end for
  Calculate a control point $\mathbf{c}$ by Equation (11);
  Add $\mathbf{c}$ to $\mathbf{T}$;
  Update τ_i;
end for
Return the estimated trajectory $\mathbf{T}$.

and

$$h_i = t_i + s_r, \tag{8}$$

where s_r represents the sample rate that is specified by the flow speed adjusted by the user; η is an overlapping term that is practically used to control the smooth motion in final result. In our implementation, we empirically set $\eta = 4$. By simply adding the overlapping term in this manner, we can further preserve the curvature of the USPs in the resultant animating video. See our supplementary video for a better visualization for the ablation study of this technique.

Once the USPs are converted to vector form, we use them to estimate the control points. As a result, each USP has a corresponding set of vectors $\mathbf{V} = \{a_i\}$. To control the curvature of the trajectory of pixel p as close as possible to a USP $\mathbf{P}$, we find on every vector a_i a point that is paired with p at a timestep τ, denoted as q_i. The coordinate of q_i is defined as

$$q_i = \mathbf{P}[x] \; s.t. \; x = f(\vec{a_i}) * \tau_i, \tag{9}$$

where $f(.)$ is the function that returns the magnitude of $\vec{a_i}$. Chasing the index x on the set $\mathbf{V}$ yields a set of paired points $\mathbf{Q} = \{q_i\}$. Here, $i \in [0 \ldots n_k]$, and n_k is the total number of vectors in $\mathbf{V}$.

With the respect of obtaining the best trajectory, each q_i is assigned a weight. This value is treated to guarantee the final motion to be smooth under the restriction of the motion map obtained from the previous process. Accordingly, the weight of q_i is formulated as follows:

$$\omega_i = (m_p)^{\mu}/(\varepsilon + E_i), i \in [0 \ldots N_v], \tag{10}$$

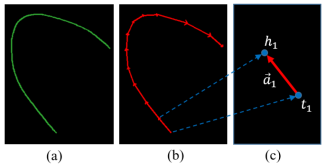


Fig. 8. User-specified path in stroke form (a), vector form (b), and zoom-in of the first vector ($\vec{a}_1$) with the tail t_1 and the head h_1.

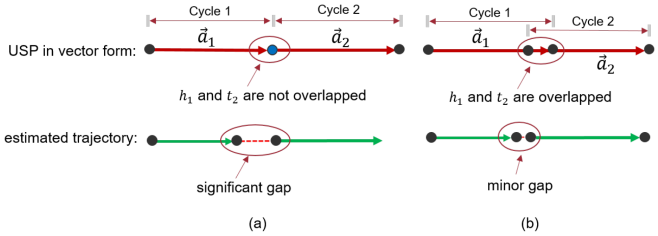


Fig. 9. Visualization of the effect of overlapping term in animating results. (a) Without the overlapping term, the tail of $\vec{a_2}$ (t_2) is the head of $\vec{a_1}$ (h_1), and (b) with the overlapping term, h_1 and t_2 are overlapped. There is a significant gap between the estimated trajectories of two adjacent cycles in (a), and it is reduced in (b) in the cases that the estimated terminal point of cycle 1 is not tightly asymptotic to the head of the corresponding vector.

where m_p is the motion energy of pixel p obtained by Equation (6), E_i is the Euclidean distance [33] between q_i and p, μ denotes a smoothness constant, and ε is a small constant used to avoid denominator zero.

By formulating the head h_i of $\vec{a}_i \in \mathbf{V}$ and the weights ω_i of $q_i \in \mathbf{Q}$, the control point c of the estimated trajectory is found as

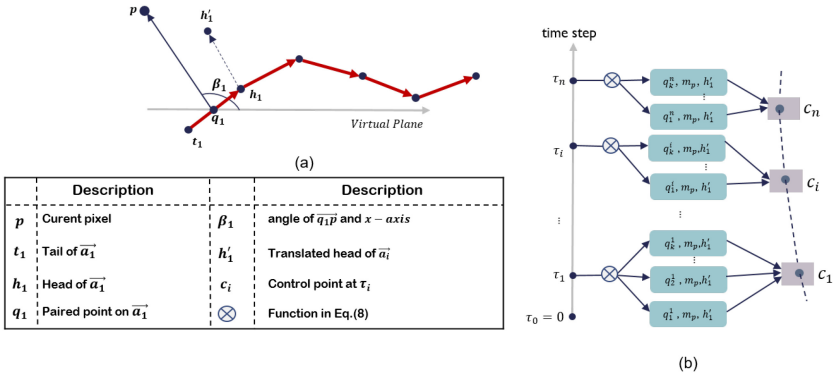
$$c = \frac{\sum_{i=0}^{N_v} \left(\psi(h_i) * \omega_i\right)}{\sum_{i=0}^{N_v} \omega_i}, \tag{11}$$

where N_v is the total number of vectors in $\mathbf{V}$ and $\psi(.)$ is the preserve-curve-translation function of the head of the vectors. Basically, in Euclidean geometry, a translation [2] is a geometric transformation that moves every point of a figure or a space by the same distance in a given direction. If v is the translation vector and p is the initial position of some object, then the translation function T_v will work as $T_v(p) = p + v$. In the condition where we naively apply this function in our translation method, the smoothness of movement might not be guaranteed in the cases where the USPs are not straight lines. We argue this by using the angle between $\overrightarrow{q_i p}$ and x-axis (Ox) to preserve the curvature of the estimated trajectory. The radian of this angle, denoted by β_i, is calculated by a virtual plane, which is parallel with Ox, and $\overrightarrow{q_i p}$. Accordingly, we formulate our preserve-curve-translation of a given point $u = (x, y)$ as

$$\psi(u) = (x + E_i * \cos\beta_i, y - E_i * \sin\beta_i), \tag{12}$$

where E_i is the Euclidean distance between q_i and p, $\beta_i = \angle(q_i p, Ox)$.

We demonstrate the above steps of our PCW algorithm in Figure 10(a). Repeating this procedure at each timestep yields a set of control points that forms the trajectory movement of pixels in a cycle (as shown in Figure 10(b)). We use the terms *h-PCW* to stand for the above procedure, since the head (h) of the vectors is used in PCW algorithm.



Description		Description	
p	Curent pixel	β_1	angle of $\overrightarrow{q_1 p}$ and $x-axis$
t_1	Tail of $\overrightarrow{a_1}$	h'_1	Translated head of $\overrightarrow{a_i}$
h_1	Head of $\overrightarrow{a_1}$	c_i	Control point at τ_i
q_1	Paired point on $\overrightarrow{a_1}$	$\otimes$	Function in Eq.(8)

Fig. 10. (a) Interpreting the proposed translation used in our PCW procedure. (b) Visualization of the *h-PCW* procedure.

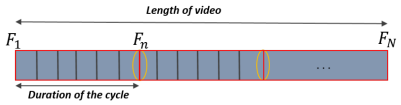


Fig. 11. The looping video of length N, which is composed of frames F_i, where $i = 1 \ldots N$. This video consists of a number of cycle. Each cycle is a partial video that consists of n frames. There is a discontinuity at the junction between two adjacent loops (highlighted in yellow circle).

3.4.2 Cycle Warping. This study addresses the animation of a specified region as a loop motion. Therefore, when the sequence is looped, there is a discontinuity between the two adjacent frames where the pixel abruptly moves backward from the terminal point to the starting point in the corresponding cycle (visualized in Figure 11). In this subsection, we present how we ease such discontinuity by our proposed cycle warping.

Recall the aforementioned assumption that each pixel moves in a cycle. By applying the h-PCW procedure, the pixel moves from the starting position to the terminal position of the cycle to form a trajectory that satisfies the aforesaid criteria. Nonetheless, to make a never-end loop, the pixel is necessitated to move back to the starting position of the loop whenever it terminates the ending position of the cycle. This phenomenon results in significant discontinuity. In other words, the h-PCW method by itself is not sufficient to construct an infinite-loop animation. We resolve this issue by relying on the benefit of the proposed PCW method. That is, we create two sets of animated frames by operating the PCW method with two distinct parameters. One is to animate the current pixel to move from its position to the terminal of the corresponding cycle. The other one is to estimate the trajectory of pixel from the terminal position to the starting point of the same cycle. Two layers are accordingly produced. The frames in the final video are generated by blending these two layers.

We denote the set of control points of pixel p obtained from h-PCW procedure as $\mathbf{T} = \{c_i\}$. The mentioned artifact is resolved by strategically calculating the copy of $\mathbf{T}$, denoted by $\mathbf{T}' = \{c'_i\}$. This set is required to satisfy two criteria: (1) preserving the curvature of the closest user-specified path, and (2) the distance between c_i and c'_i is asymptotic as tight as possible to the duration of the corresponding cycle. To achieve this, the PCW procedure is used. However, the parameter

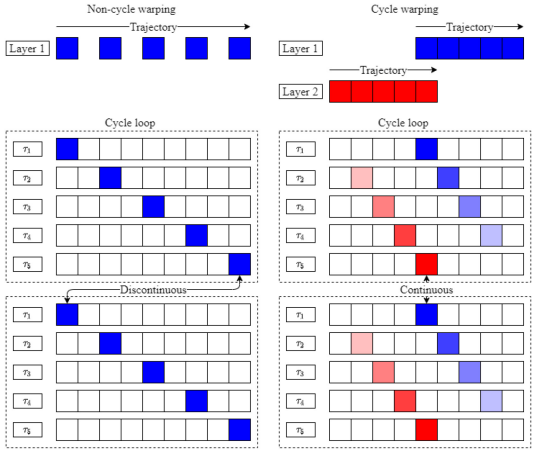


Fig. 12. Illustration of cycle warping.

employed in this stage is the tail of the entire vectors instead of using the head in **T**, that is,

$$c' = \frac{\sum_{i=0}^{N_v} \left(\psi(t_i) * \omega_i\right)}{\sum_{i=0}^{N_v} \omega_i}, \tag{13}$$

where $\psi(.)$ is function in Equation (12) whose parameter is the tails (t_i) of a_i. Accordingly, we call this the terms of t-PCW. By utilizing the Preserve-Curve-Warping procedure on the head and the tail of the control vectors, two layers are simultaneously produced. The first layer (denoted as F^h) is generated by h-PCW. The other one (denoted as F^t) is gained by t-PCW.

Once the two layers are generated, our system blends their textures to produce the rendered frames (**F**) as follows:

$$\mathbf{F} = \mathbf{F}^h * \tau_i + \mathbf{F}^t * (1 - \tau_i), \tag{14}$$

where $\mathbf{F}^h$ and $\mathbf{F}^t$ is the layer obtained from h-PCW and t-PCW, respectively; $0 \leq \tau_i \leq 1$. The illustration of cycle warping is outlined in Figure 12. Frames in two layers are generated independently. In this figure, we suppose there are five timesteps in a cycle. After blending two layers, the texture and movement of pixel at each timestep τ_i in a loop are visualized. When the pixel terminates at the end of the loop, a copy of itself is formed at the starting position of the cycle. Therefore, when the pixel starts a new loop, the discontinuities do not occur.

The infinite loop videos have been solved in References [3, 12]. Bhat et al. [3] define a matrix of positions in a particle trajectory along the flow line Bhat et al. [3]. The discontinuity artifact is reduced by a change to the entries of the matrix. However, there is still a discontinuity along the diagonal stepped line. To create a never-end loop video, in training, Holynski et al. [12] generate two frames at opposite ends of the video and ask it to interpolate an intermediate frame. Yet, by observing their published results, the artifacts still occur in some cases (such as in Figure 16; we will explain the reason for these artifacts in Section 4). Fortunately, by blending two layers, which are generated using PCW in the same cycle, we make the changes between two adjacent loops smooth not only in terms of texture but also in terms of physical movement.

Once each region is animated independently by use of the above procedures, the animated regions are composited back together to produce the final animating video. By isolating still objects and individually animating with distinct motion styles and flow speeds, we not only obtain the realistic motion and consistency between animation regions but also eliminate the artifact occurring

Fig. 13. Some of the still images we use to evaluate our method in our experiments. The user-specified paths of exemplar images are presented in Figure 24 in the appendix.

at the boundary of still and animation region. Taking the image in Figure 16 (the third column) as an example, the motion style and flow speed of the waterfall and water smoke are quite different. By our aforementioned strategy, we can easily control such animating properties of these two regions. Please see our demo video for a visualized demonstration of our system capability compared to related work.

4 EXPERIMENTAL RESULTS

Experimental parameters. We implement and evaluate our interactive system on a PC equipped with Intel Core i7-770 CPU (3.6 GHz) and 16 GB RAM. The **User Interface (UI)** is developed by Qt toolkit [6]. For the optimization issue, we use Eigen Optimizer [10] to solve the optimum problems in our method. We render the final videos in the length of 1 minute and up to 30 frames/second.

4.1 Our Results and Discussion

To evaluate our method, we test it on several still images of different animating objects, including *natural waterfall scenes, water splash, abstract painting, surfing, bodies of water, beaches, galaxies, smoke, cycle motions,* and *cloud in the sky*. Figure 13 presents some still images we use to generate animating results in our experiments. Some of these objects have been solved in prior works. Yet, in our study, we exhibit more challenging cases of such objects that prior works have not addressed. For instance, the waterfalls over the stones are shown in Figure 13(a) and Figure 13(d). Since our system gives a functionality to discard the still pixels in the selected animating region, realistic results can be generated by our system without any artifact caused by the still pixels. Animating the waterfall scenes in Figure 13(b) and Figure 13(c) is also a challenge due to the number of animating objects is considerable and they are overlapped (such as the rainbow on the flowing river). Nevertheless, we can generate appealing results, i.e., we animate all the animating objects with different animating styles, different speeds without damaging the rainbow on the river or the flow of the regions that are affected by their neighboring regions. Being unrestricted in domain enables our system to animate arbitrary scenes unrealistically, such as Figure 13(e),

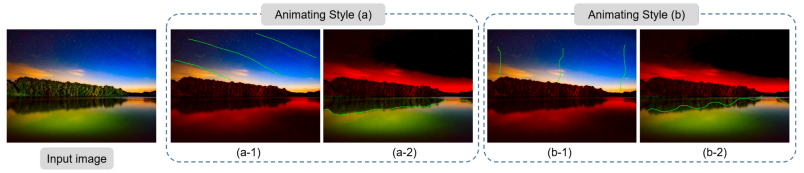


Fig. 14. Animating variations from the same input image. Panels (a-1,) and (b-1) and panels (a-2) and (b-2) derive different animating styles in the result.

Figure 13(f), Figure 13(k), Figure 13(l), Figure 13(m), and Figure 13(n). Isolating and animating objects singly greatly facilitates our system to succeed in these challenging cases. Finally, thanks to our PCW method, our system exhibits appealing results when challenging such significant curve motions as in Figure 13(g), Figure 13(i), Figure 13(j), and Figure 13(k). Readers are referred to our project website to access the better visual videos and more results. Moreover, we prove the ability of our system with the competition on the results of the prior works in the following subsection.

Controlling the Animating styles. One of the interesting aspects of our method is the ability to be flexible in animation styles. We can generate different animation styles from the same input image without restriction (e.g., the video example). This aspect is adopted by the flow generation method and the PCW algorithm. Not being restricted by video examples or any source data enables our system to generate animating styles following the guidance of the input strokes. That is, what the users draw is what the users get. In particular, let us take a region in Figure 13(m) as an example. We show the zoom-in detail in Figure 23. Here, Figure 23(a) and Figure 23(b) are the cutouts from the input image. In Figure 23(a), the user can draw only one simple stroke (in green) to demonstrate the seaway. However, the resultant animating style guided by this stroke might be tedious. The style motion of the seaway might be different in a particular region. For example, the waves lap the side of the boat, or the waves move back when they crash on the shore (highlighted in green rectangles). To create such interesting animating motions, the users just draw more strokes as shown in Figure 23(b) to control the sub-regions in the current animating region to animate them with different styles. Moreover, isolating selected regions and animating them independently allows us to obtain more creative and appealing results. Figure 14 shows that on the same animating region, with some strokes, our system derives different animation styles. Please visit our website to see the visual video on these examples.

Motion speed. In terms of motion speed, this is a weighty plus of our system. Our demo video demonstrates that our system can generate appealing results with different motion speeds. To control this issue, our system gets the input "*speed*" from the users via our UI. As mentioned in Section 3.4.1, when working on PCW algorithm, we must convert the USPs to vector form in advance. The vectors are in the same magnitude assigned by the input speed. The magnitude of vectors indicates the duration of the cycle. Therefore, the amplitude of pixel movement in the cycle demonstrates the level of motion speed as the pixels in an animating region moves simultaneously. In other words, the motion speed increases linearly with the input speed. Hence, the users just simply input the speed to obtain the desired motion speed in the animating video. While this issue is a concern in prior systems, since the motion speed is normally dependent on the video example or training data, this property enables our system to advance these systems to produce more interesting results. For example, in Figure 14, two selected regions (e.g., sky and the river) can be animated not only with different styles but also with separate motion speeds. As a result, we can obtain more creative and realistic results.

Fig. 15. Comparison between the results of Reference [5] (first row) and our results (second row). The input images are shown in Figure 26.

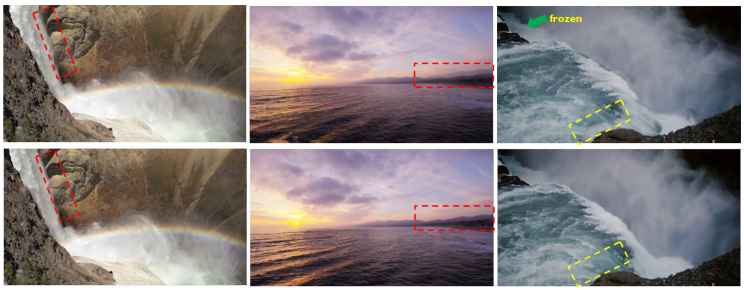


Fig. 16. The comparison between our method (second row) and Holynski et al. [12] (first row). In the first two columns, in the compared method [12], the still regions are effected by the surrounding motions (masked in red rectangle). Meanwhile, such artifacts do not occur in our results. In the last column, a discontinuity artifact occurs at the gap of two loops (highlighted in yellow rectangle). Compared with their results, the discontinuity is almost unrecognizable in ours. Besides, some regions in their final results are frozen instead of being animated. Please see the supplementary material for visual demonstrations. The input images are shown in Figure 26 in the appendix.

4.2 Qualitative Evaluation

Figure 15 shows the examples of the limitations in Endo et al. [5]'s results, i.e., they are the unnatural distortions and the challenge in controlling the estimated motion to be realistic. The first limitation is found due to the predicted frames are reconstructed only from an input image [5]. The reason for the second issue is that the motion estimation network in this work uses a latent code as input to the network, and different codes produce different predicted motions. Therefore, defining a suitable random latent code for each input image to produce a plausible animation result is an issue needed to investigate. To specify an appropriate latent code, the authors use arrow annotations from the user for motion control. Yet, the consistent motion in the relative regions, such as the motion of cloud reflects on the surface of the river, is still a concern in their study.

Figure 16 demonstrates the comparison between our study and Holynski et al. [12]. The system proposed in Holynski et al. [12] sometimes fails to isolate the animation regions and the still regions, thus the still regions will be animated by their surrounding motions. As shown in this figure, the patches of the rock, which located nearby the waterfall, are affected by the motion of the waterfall. Thus, pixels in these patches move along with the intended motion, or the ghost occurs at the mountain, since it is affected by the motion of the ocean. Also, some regions are frozen instead of being animated, since they are not correctly recognized.

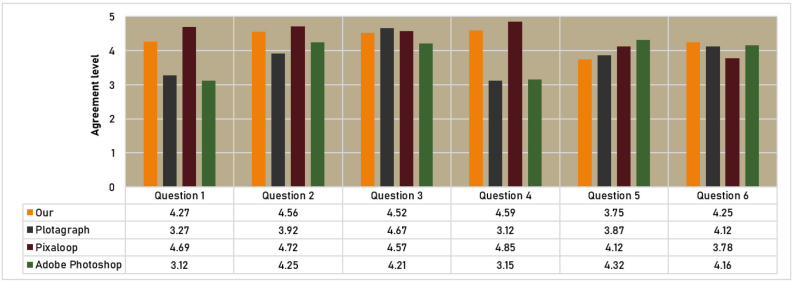
Fig. 17. Visualization the results generated by the commercial applications Plotagraph (left) and Pixaloop (right). The artifacts in their results masked in red rectangles.

In contrast to Endo et al. [5] and Holynski et al. [12], our system creates better results that are mentioned as the limitations in the compared works. Relying on the user's annotations, the animation regions are isolated and animated independently. Thanks to our PCW method and cycle warping method, each pixel in a certain animation region is animated along the given curve without distortion. Since the related animation parameters are input via the system UI and each region is animated independently, our system is easily controls the final animation design and successfully obtains the consistent motion in regions; moreover, the motions of a region are not affected by the others. Thence, we can overcome the mentioned drawbacks from the compared systems and generate more plausible animating results.

Apart from above, in the supplementary material, we show the comparisons between our system performance and those in References [3, 4, 15]. The input images are outlined in Figure 25. Lai et al. [15] propose a method to analyze the curvilinear structures of the strokes in Chinese painting. However, their proposed technique cannot animate dynamic effects. We tested on their published data (Figure 25(b-1,2). The video results show that our system can create realistic animation results with various motion styles and different speeds of flow. Figure 25(a-1,2,3) are the input images used in Bhat et al. [3]. They propose an algorithm to analyze the motion of textured particles in the input video along user-specified flow lines. Although we generate animation from single still image, our generated videos are as appealing as the real videos in theirs. The scheme in Chuang et al. [4] is related to ours. However, Chuang's system solely provides the motion of trees, water, boats, and clouds. In the comparisons (on Figure 25(c-1,2,3)), the results show that our results are more appealing with the diversity either motion styles or animating scenes.

Besides, we also compare our results with those that are generated by the existing commercial applications Pixaloop [25] and Plotagraph [26]. Generally, these animating applications can generate appealing results. However, we found that they still suffer from some limitations. For instance, the artifact occurs in both two applications as shown in Figure 17. And, the speed of animating regions are set to be the same. This may due to the animating regions and the still regions are not isolated in their scheme. Specifically, when challenging to animate such an image in Figure 17(right), the ghost artifact occurs in their results in the waterfall regions. The reason is that the textures of stone are immersed in the waterfall. Their systems use all pixels in such regions to animate. This results in the stones are animated instead of being still. In contrast, our method outperforms in this competition in terms of the abilities in removing the still pixels that get mixed up with animating pixels in a selected region and controlling the animating regions with distinct motion speeds. Please see the videos for better visual comparisons on our project website.

In terms of infinite loop videos, we evaluate this issue by comparing our proposed system with those in Holynski et al. [12]. The comparisons are shown in Figure 16. From the figure, we can observe the obvious artifact occurs at the gap of two loops in Holynski et al. [12]'s results. Fortunately, such discontinuity artifacts are significantly reduced and almost unrecognizable in our results. In



	Question 1	Question 2	Question 3	Question 4	Question 5	Question 6
Our	4.27	4.56	4.52	4.59	3.75	4.25
Plotagraph	3.27	3.92	4.67	3.12	3.87	4.12
Pixaloop	4.69	4.72	4.57	4.85	4.12	3.78
Adobe Photoshop	3.12	4.25	4.21	3.15	4.32	4.16

Fig. 18. Results of the user study about intuitiveness of the animating systems.

the supplementary video, the visual comparisons on this point are shown between Holynski et al. [12], our approach without cycle warping, and our approach with cycle warping.

4.3 Objective Evaluation

We further conduct two user studies to evaluate the effectiveness of our system and the visual quality of our animating results comparing to those of the aforementioned prior works. A total of 30 participants are invited to join in our user studies. They are of different ages (age range of 20–35) and background (9 of them have image processing or graphics-related backgrounds).

The first user study is designed to evaluate the intuitiveness of our interactive system. We initially show the participants the tutorial video of the interactive tools of Plotagraph [26], Pixaloop [25], Adobe Photoshop [1], and our system. We asked the participants six questions:

Question 1: Do you think the system is easy to operate?
Question 2: Do you think the system needs to draw fewer lines and the operation is closer to the user?
Question 3: Do you think the animation produced by the system meets your expected results?
Question 4: Do you think the operations of the system do not require the user have expert background?
Question 5: Do you think that the system can produce diversified results?
Question 6: Do you think the result produced by the system is appealing?

For each video, the participants answer six questions by voting in one of the following five levels: *strongly disagree, disagree, neutral, agree, strongly agree*, which correspond to scores of 1, 2, 3, 4, and 5, respectively. Thereafter, we compute the average score from 30 participants as the effectiveness of our proposed system. A higher score indicates better agreement. Figure 18 shows the statistics results. The statistics results of this user study in Figure 18 show that our method is selected at a high-rate level. Among six criteria, the rating of our method in questions 1, 2, 4, and 5 is lower than Pixaloop but higher than the others. However, the rating of our method is the highest in question 6 and tightly comparable with Pixaloop [25] and Plotagraph [26] in question 3. These indicate that our system is less intuitive than Pixaloop and Plotagraph but more judged to be able to produce appealing results. In the aspect of question 5, our method has the lowest rating in the contrast systems. This might be due to the participants' judge this aspect by watching the demo video instead of practical experience. However, the rating of this criteria is still at an acceptable level. In conclusion, our method is judged to be easy to use and can generally yield appealing results in this user study even that the compared commercial systems are not research papers.

The second user study is designed to evaluate the visual quality of our animating results. Beside our method, two animating systems are chosen in this user study. They are Endo et al. [5], and

Table 1. The Statistics of Rating Score (%) on the Animating Results

Testing cases	Our method	Pixaloop	[5]
1- Ban Gioc waterfall	62	27	11
2- Tahquamenon Falls	57	24	19
3- Iguazu Falls	76	15	9
4- Women of splash	48	20	32
5- Cycle of splash	42	30	28
6- Art eye	32	43	25
7- Surfing	23	40	37
8- Galaxy girl	64	15	21
9- Galaxy	29	38	33
10- Smoke of mountain	42	32	26
11- Abstract painting	28	38	34
12- River starry night	47	26	27
Average:	**40.27**	30.6	29.13

Pixaloop [25]. The source code of Endo's method is published by the authors and the application Pixaloop is free for personal use. Thus they are trustful enough to use and fair for comparisons. We select 12 still images from Figure 13, which have diversity in the scene, to generate 12 corresponding animating video results by each method. We first show 12 sets to the participants. Each set consists of an input still image, and three animating videos. The resultant videos are displayed in random order to prevent participants from inferring the animating method. The participants are not provided any method information. We simply ask them to choose the one that they have a perceptual feeling that is the best among the three results. We received 30 answers on 12 sets. The number of votes on an animating result is performed in percentage. That is, the method that has a higher percentage means animating result generated from that method is better. The results are presented in Table 1. The statistics shows our results obtain the majority votes, since the average score is quite higher than the contrast methods' results. From this result, we can conclude that most people are more impressionable to our animating results.

4.4 Ablation Study

The influence of AR extraction. Our system utilizes the user-select input to define the animating region, as shown in Figure 5. The filter is proposed in this manner as an optional function to eliminate the still pixels in the cases that the animating region is not a one-type-object. To clarify the influence of this function, we compare the generated results with and without the filter on the input image shown in Figure 19. Note that other commercial software such as Pixaloop [25] and Plotagraph [26] do not use this filter. So, their results might be similar to Figure 19(a). From the ablation, we can see that without the filter the still pixels are animated. This phenomenon yields the inconsistency in the texture of the still subject in the AR and the corresponding still object surrounding the AR. And thus, ghost artifacts occur obviously (Figure 19(a)). See the better visualization in the supplementary video. As we mentioned in Section 3.2, our system allows the users to adjust the threshold to obtain the filtered animating region $\mathcal{R}^F$. That is, the pixel color threshold Θ in Equation (2) varies for each input image. Hence, the "cleanliness" of $\mathcal{R}^F$ depends on the users' perceptual feeling. We visualize this ablation in Figure 19. We can see that, applying the filter can discard the still pixels (e.g., pixels of the stone mixed in the waterfall) from the animating region compared to the contrary case. However, the level of "cleanliness" of $\mathcal{R}^F$

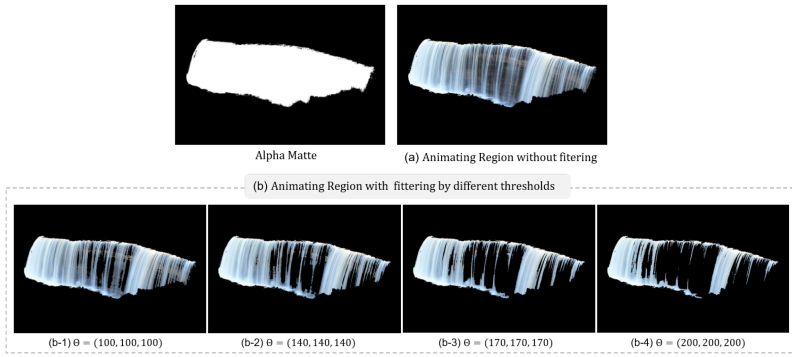


Fig. 19. Visualization of the ablation of filter function on the exemplar image "Tahquamenon Falls".

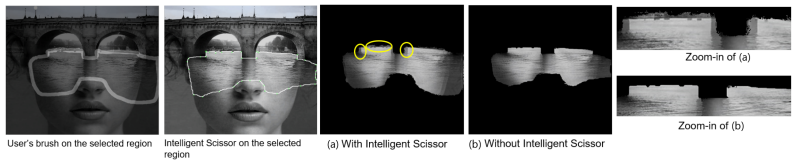


Fig. 20. The influence of matting result on the quality of animating result. The differences of with/without fixed boundary are highlighted in yellow circles.

depends on the threshold. For instance, RF in Figure 19(b-1) and Figure 19(b-2) might be coarse but it is too sharp in Figure 19(b-4) and it is fine in Figure 19(b-3).

In terms of the influence of the performance of matting method [9] on the AR extraction, we visualize this ablation in Figure 20. The matting technique by itself does well in terms of generating matting results based on user's selection but it is not sufficient to avoid the boundary artifacts. In particular, a certain AR, which is selected by the user, might have neighboring regions including animating region or still region. Based on user's selection, the extracted AR certainly contains "incorrect pixels" (i.e., the pixels that are not expected to select) at the boundary. As shown in Figure 20(a), animating these pixels yields obvious ghost artifacts. In Figure 20(b), by employing the Intelligent Scissor method [20], the "clean-boundary ARs" are obtained. Consequently, the boundary artifacts are almost resolved. In practically, the users are recommended to include this step after brushing along the AR to avoid such artifacts. In summary, after selecting an animating region, our system gives two functionalities: (1) a filter for eliminating the artifacts caused by the still pixels and (2) an Intelligent Scissor step for avoiding the boundary artifacts.

Overlapping term. We use the overlapping term in converting the user-specified path to control path to make the final animating smooth. We remove this term in Equation (7) to generate results and compare with ours. The comparison shows that without the overlapping term, there is a considerable artifact in animating result if the user-specified paths are not the straight line. By using this term, our method can be flexibly applied to generate smooth animating results with complex user annotations. See the visual ablation in our supplementary materials.

Verify the effect of PCW algorithm. Wecompare the generated results with and without PCW algorithm to verify the effect of PCW algorithm. As shown in Figure 21, using PCW method can generate the animating results without being damaged by the curvature of the user-input strokes (Figure 21(b-2)). Without PCW method, it totally fails to control the motion of the result to

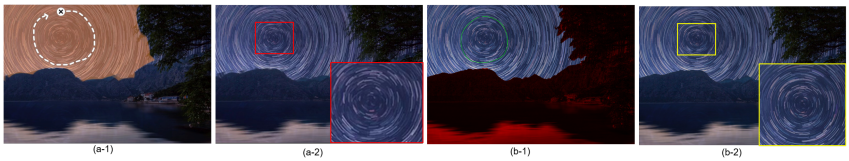


Fig. 21. (a-1) user-input with Pixaloop application, (b-1) user-input with our system, (a-2) the animated frame is damaged by the curvature of the input stroke without PCW method, and (b-2) the animated frame generated by our PCW method can preserve the curvature of the input stroke well.

Table 2. The Average Running Time of Our Method for Different Input Images

Testing images	# regions	one-loop video	looping video
1- Ban Gioc waterfall	5	9.24 s	31.02 s
2- Tahquamenon Falls	3	7.92 s	29.36 s
5- Cycle of splash	1	4.32 s	15.27 s
7- Surfing	2	6.17 s	27.03 s

follow such user-input stroke (Figure 21(a-2)). See the supplementary video for a better ablation visualization.

Cycle warping. The cycle warping method is proposed to generate a looping video with the minimized discontinuity artifacts. Without this phase, to start a new loop, pixels in a certain animated region abruptly move backward from the terminal point to the starting point in the cycle. Discontinuity artifacts certainly occur. With cycle warping, such artifacts are eliminated effectively. Please see the supplementary video for a better ablation visualization.

4.5 Timing

Table 2 shows the average running time of our method on the input images in the same resolution of 1,024 × 1,024. We select the images in Figure 13 that have different number of animating regions. We estimate the execution time of our system when rendering a one-loop video (length of 1 second) and a looping video (length of one minute). All the videos are rendered at the rate of 30 frames/second. The results in the table show that the total computation time increases linearly with the number of animating regions. On average, our system takes less than 10 seconds to render a one-loop video and about 30 seconds to render a looping video. In the flow generation, it takes 1.23 seconds on average to solve Equation (5). Meanwhile, the system introduced by Okabe et al. [24] uses 15 computers, which consist of 100 cores in total, and finishes the video retrieval up to one minute. Yet, they sometimes need to refine the alpha masks if the retrieved candidates are not matched. Compared to this system, our scheme is simpler in terms of intuitiveness and faster in terms of rendering such an animating video from a still image.

4.6 Limitations

We have presented a scheme that animate various types of objects. However, in the cases of the selected regions have an obvious shape change, our method may not perform well (as shown in Figure 22). Currently, our method does not change the boundaries of animating regions. Besides, since our system generates video from only a single still image, an animation video with

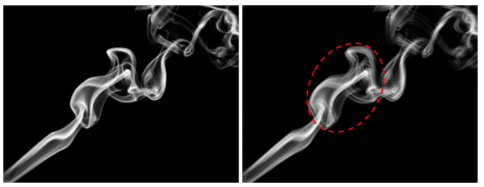


Fig. 22. The bad case is generated from our system. The left one is the input image. The right one is a frame in the output video. The artifacts occur in this example, such shown in the red circle.

high-resolution is an issue we take into account in our future work. Nevertheless, even with these limitations, plausible results are still obtained from our proposed system.

5 CONCLUSIONS AND FUTURE WORK

In this article, we introduce an interactive system that animates still images with a novel warping method. We demonstrate that the novel warping method, *Preserve-Curve-Warping*, proposed in this study is especially useful for preserving the curvature of the input strokes in the results. This gives our system the capability of animating various objects with flexible animating styles. Our results and evaluations show that the proposed animating scheme substantially outperforms prior works and overcomes the drawbacks in existing commercial applications. In terms of high-resolution results, we would like to address this issue in our future work by taking more images in this part to enhance the quality of output videos. Besides, we will investigate such techniques to handle the change on the region boundary in our future work. This could be a possibility to extend our system on more objects, such as birds flying or swing motion.

APPENDIX

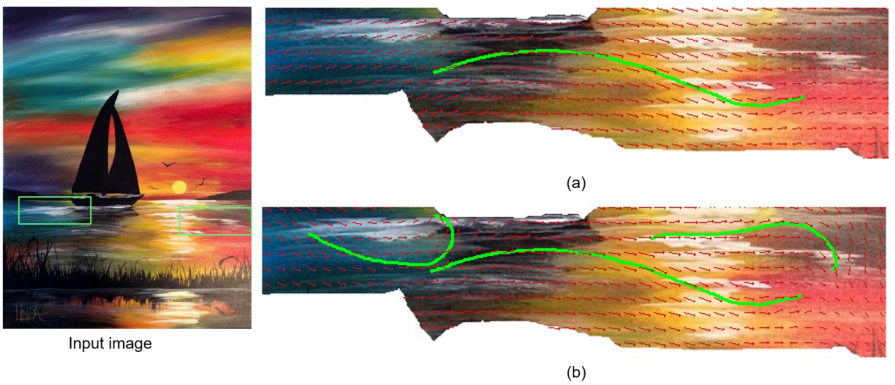


Fig. 23. Demonstration of different animating styles in the same region. (a) and (b) are cutouts from the input image.

Fig. 24. The user-specified paths of some exemplar images in Figure 13 (the green strokes). The resultant animations followed these strokes are visualized in our demo video. Please see on our project website.

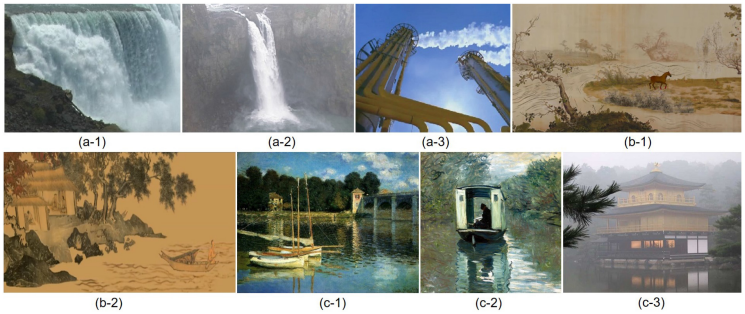


Fig. 25. The input images used in the experiments to compare with Bhat et al. [3] ((a-1)–(a-3)), Lai et al. [15] ((b-1) and (b-2)), and Chuang et al. [4] ((c-1)–(c-3)).

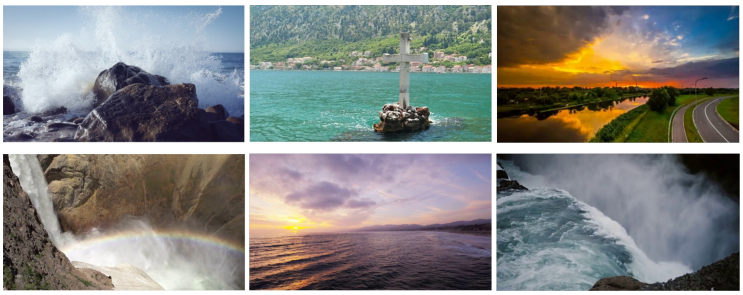


Fig. 26. The input images used on the comparisons with Endo et al. [5] and Holynski et al. [12].

ACKNOWLEDGMENTS

The authors thank the reviewers for the many constructive comments that help improve the article.

REFERENCES

[1] Adobe Photoshop. 2004. Retrieved from https://www.adobe.com/products/photoshop.html.
[2] George B. Arfken and Hans J. Weber. 1999. Mathematical Methods for Physicists: A Comprehensive Guide. Academic Press.
[3] Kiran S. Bhat, Steven M. Seitz, Jessica K. Hodgins, and Pradeep K. Khosla. 2004. Flow-based video synthesis and editing. In *ACM SIGGRAPH 2004 Papers*. 360–363.
[4] Yung-Yu Chuang, Dan B. Goldman, Ke Colin Zheng, Brian Curless, David H. Salesin, and Richard Szeliski. 2005. Animating pictures with stochastic motion textures. In *ACM SIGGRAPH 2005 Papers*. 853–860.
[5] Yuki Endo, Yoshihiro Kanamori, and Shigeru Kuriyama. 2019. Animating landscape: Self-supervised learning of decoupled motion and appearance for single-image video synthesis. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–19.

[6] Eirik Eng. 1996. Qt GUI Toolkit: Porting graphics to multiple platforms using a GUI toolkit. *Linux J.* 1996, 31es (1996), 2–es.
[7] William T. Freeman, Edward H. Adelson, and David J. Heeger. 1991. Motion without movement. *ACM Siggraph Comput. Graph.* 25, 4 (1991), 27–30.
[8] Ruohan Gao, Bo Xiong, and Kristen Grauman. 2018. Im2flow: Motion hallucination from static images for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5937–5947.
[9] Eduardo S. L. Gastal and Manuel M. Oliveira. 2010. Shared sampling for real-time alpha matting. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 575–584.
[10] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. Retrieved from http://eigen.tuxfamily.org.
[11] Yan Gui, Li-zhuang Ma, Chao Yin, and Zhi-hua Chen. 2012. Preserving global features of fluid animation from a single image using video examples. *J. Zhejiang Univ. Sci. C* 13, 7 (2012), 510–519.
[12] Aleksander Holynski, Brian Curless, Steven M. Seitz, and Richard Szeliski. 2020. Animating pictures with eulerian motion fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5810–5819.
[13] Kyung-Rae Kim, Whan Choi, Yeong Jun Koh, Seong-Gyun Jeong, and Chang-Su Kim. 2019. Instance-level future motion estimation in a single image based on ordinal regression. In *Proceedings of the IEEE International Conference on Computer Vision*. 273–282.
[14] Kris M. Kitani, Brian D. Ziebart, James Andrew Bagnell, and Martial Hebert. 2012. Activity forecasting. In *Proceedings of the European Conference on Computer Vision*. Springer, 201–214.
[15] Yu-Chi Lai, Bo-An Chen, Kuo-Wei Chen, Wei-Lin Si, Chih-Yuan Yao, and Eugene Zhang. 2016. Data-driven npr illustrations of natural flows in chinese painting. *IEEE Trans. Vis. Comput. Graph.* 23, 12 (2016), 2535–2549.
[16] Zicheng Liao, Neel Joshi, and Hugues Hoppe. 2013. Automated video looping with progressive dynamism. *ACM Trans Graph.* 32, 4 (2013), 1–10.
[17] Chih-Yang Lin, Yun-Wen Huang, and Timothy K. Shih. 2019. Creating waterfall animation on a single image. *Multimedia Tools Appl.* 78, 6 (2019), 6637–6653.
[18] Elizaveta Logacheva, Roman Suvorov, Oleg Khomenko, Anton Mashikhin, and Victor Lempitsky. 2020. Deeplandscape: Adversarial modeling of landscape videos. In *Proceedings of the European Conference on Computer Vision*. Springer, 256–272.
[19] Wei-Chiu Ma, De-An Huang, Namhoon Lee, and Kris M. Kitani. 2017. Forecasting interactive dynamics of pedestrians with fictitious play. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 774–782.
[20] Eric N. Mortensen and William A. Barrett. 1995. Intelligent scissors for image composition. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. 191–198.
[21] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. 2016. Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3521–3529.
[22] Makoto Okabe, Ken Anjyo, Takeo Igarashi, and Hans-Peter Seidel. 2009. Animating pictures of fluid using video examples. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 677–686.
[23] Makoto Okabe, Ken Anjyor, and Rikio Onai. 2011. Creating fluid animation from a single image using video database. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 1973–1982.
[24] Makoto Okabe, Yoshinori Dobashi, and Ken Anjyo. 2018. Animating pictures of water scenes using video retrieval. *Vis. Comput.* 34, 3 (2018), 347–358.
[25] Pixaloop 2021. Retrieved from https://www.pixaloopapp.com/.
[26] Plotagraph 2017. Retrieved from https://plotaverseapps.com/.
[27] Ekta Prashnani, Maneli Noorkami, Daniel Vaquero, and Pradeep Sen. 2017. A phase-based approach for animating images using video examples. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 303–311.
[28] William H. Press, William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, and William T. Vetterling. 1989. *Numerical Recipes in Pascal: The Art of Scientific Computing*. Vol. 1. Cambridge University Press.
[29] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. 2000. Video textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. 489–498.
[30] Eric W. Weisstein. 2005. Moore neighborhood. In *From MathWorld—A Wolfram Web Resource*.
[31] Wikipedia contributors. 2020. Linear interpolation. Retrieved January 8, 2021 from https://en.wikipedia.org/w/index.php?title=Linear_interpolation&oldid=986522475.
[32] Takuma Yagi, Karttikeya Mangalam, Ryo Yonetani, and Yoichi Sato. 2018. Future person localization in first-person videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7593–7602.
[33] Jin Zhang. 2007. *Visualization for Information Retrieval*. Vol. 23. Springer Science & Business Media.

Received 9 September 2021; revised 12 December 2021; accepted 17 January 2022