Selectively De-Animating Video

Jiamin Bai [1] Aseem Agarwala [2] Maneesh Agrawala [1] Ravi Ramamoorthi[1]
University of California, Berkeley [1] Adobe[2]

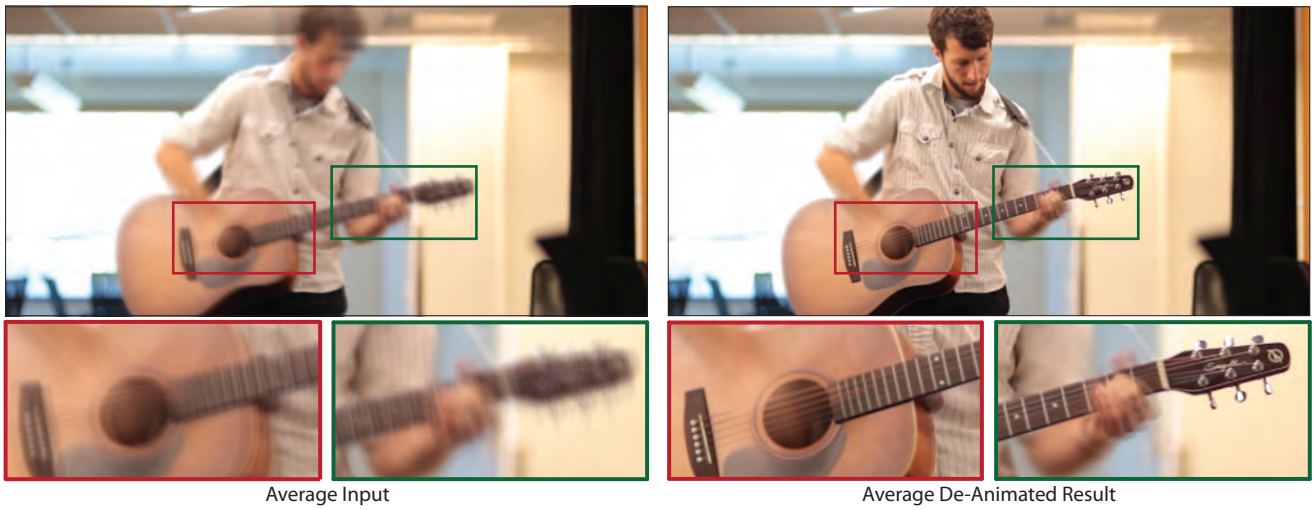


Figure 1: *Large-scale motions of the guitar body can make it difficult to follow the finer-scale motions of the strings and fingers. We visualize the amount of movement by averaging the frames of the input video (left) and find that the body and fretboard of the guitar, as well as the strings and fingers are blurred because they move a lot. With our selective de-animation technique, we remove the large-scale motions of the guitar to make it easier to see the finer scale motions. Averaging the frames of our de-animated result (right) shows that the body and fretboard are sharp and therefore immobilized. Note that while the strings and fingers are sharper than in the visualization of the input video, they remain blurry because their fine-scale motions are retained in our de-animated result. We encourage the reader to view the paper video, to see this comparison in video form.*

Abstract

We present a semi-automated technique for selectively *de-animating* video to remove the large-scale motions of one or more objects so that other motions are easier to see. The user draws strokes to indicate the regions of the video that should be immobilized, and our algorithm warps the video to remove the large-scale motion of these regions while leaving finer-scale, relative motions intact. However, such warps may introduce unnatural motions in previously motionless areas, such as background regions. We therefore use a graph-cut-based optimization to composite the warped video regions with still frames from the input video; we also optionally loop the output in a seamless manner. Our technique enables a number of applications such as clearer motion visualization, simpler creation of artistic *cinemagraphs* (photos that include looping motions in some regions), and new ways to edit appearance and complicated motion paths in video by manipulating a de-animated representation. We demonstrate the success of our technique with a number of motion visualizations, cinemagraphs and video editing examples created from a variety of short input videos, as well as visual and numerical comparison to previous techniques.

Links: DL PDF WEB VIDEO

1 Introduction

The large-scale motion of an object can sometimes make it difficult to see its finer-scale, internal motions, or those of nearby objects. Consider a video of a guitar teacher demonstrating the finger motions required to play a song. The fine motions of the fingers on the strings and frets are obscured or visually masked by the larger-scale, gross motions of the guitar body. The finger positions required to play notes and chords would be much easier to follow if the guitar were immobilized (Figure 1 and accompanying video).

In this paper, we present a semi-automated technique for selectively *de-animating* or removing the large-scale motions of one or more objects. The user draws a small set of strokes indicating the regions of the objects that should be immobilized and our algorithm warps the video to remove the gross motion of these regions while leaving finer-scale, relative motions intact. The warp may, however, propagate unwanted motions to other regions that should remain still, such as the background. To eliminate such undesirable motion, we develop a method for compositing the warped video with still regions taken from frames of the input. The resulting videos highlight fine-scale, internal motions of one or more objects, as well as relative motions of nearby objects. In the guitar example (Figure 1) we have immobilized the guitar and composited still frames of the head and body to make it easier for viewers to focus on the finger positions.

Our algorithm facilitates the creation of *cinemagraphs* (`cinemagraphs.com`) which have recently become a popular form of motion visualization. They are designed to draw the viewer's attention to specific objects and motions in a scene. Cinemagraphs lie between the traditional media of video and

photographs; the most important objects remain animated while the surrounding objects are held still. While some cinemagraphs simply composite a moving object onto a still frame background, the most effective also eliminate distracting, large-scale motions of the primary objects so that viewers can focus on the subtle, fine-scale motions. Cinemagraphs are also designed to loop endlessly like a video texture [Schödl et al. 2000]. Therefore, users can optionally use our compositing algorithm to also compute a seamless video loop. Well-designed cinemagraphs can be surreal and are often unexpectedly engaging. They are now widely seen in major online newspapers, magazines, advertising and retail websites. Our cinemagraph examples can be seen in Figure 6.

Beyond motion visualization and cinemagraphs, we also demonstrate applications of our de-animation technique to video editing. First, we show how de-animation facilitates appearance editing of moving objects in video. The user edits an object after it is immobilized in a single frame of the de-animated video, and our system propagates these edits back to the original video (Figure 7). Second, we show that it is easier to animate a virtual object that moves within the frame-of-reference of a larger object. After the animator plans the motion of the virtual object in the de-animated video, our system adds in the original motions of the larger, frame-of-reference object (Figure 8).

Our main technical contribution is the first semi-automated method for selectively de-animating video to eliminate the gross motions of a specific object or objects. The two main steps of our algorithm follow the general approach of previous work, namely content-preserving warps for video stabilization [Liu et al. 2009], and graph-cut video compositing [Kwatra et al. 2003]. Novel to our problem domain is the challenge of only stabilizing locally-defined regions which have been identified by the user *in a single frame*. Therefore, our energy function for warping is more complicated, and requires a multi-stage solution method. We also modify the energy functions used for compositing from those found in previous work to achieve good results for our problem domain.

2 Related Work

Visualizing motion is a common focus of many previous systems. One approach is to depict the motion using a static image [Goldman et al. 2006; Caspi et al. 2006; Barnes et al. 2010; Correa and Ma 2010; Assa et al. 2005; Agarwala et al. 2004; Kim and Essa 2005]. These methods use repetition, blur, spatial offsetting, montage and diagrammatic elements to represent motion, but because the result is a still, viewers must mentally reconstruct timing information. Another approach is to create a short video abstract that retains only the most salient motions. Truong and Venkatesh [2007] survey a number of recent techniques for identifying such salient video segments. Pritch et al. [2008] summarize surveillance video by non-chronologically rearranging spatio-temporal video segments representing salient activity. While these methods compress the length of a video, they do not emphasize important motions within the frame.

Motion magnification [Liu et al. 2005] highlights small spatial motions (e.g. breathing, deformations) by exaggerating them. Thus, its goal is the opposite of ours, as we focus on emphasizing the fine-scale motions of an object by removing larger-scale motions. In fact, motion magnification could be used in conjunction with our approach to further emphasize the most important motions. Bennett and McMillan [2007] provide tools for creating time-lapse videos with stroboscopic motion trails to emphasize translational motions. Rubinstein et al. [2011] remove transient, distracting motions from time-lapse sequences to increase the comprehensibility of longer-term movements. Our work is directly inspired by the vision of the Moment Camera [Cohen and Szeliski 2006] as we provide tools for isolating short motions that represent key moments in a scene.

Web-based instructions for creating simple cinemagraphs ask users to manually create a single alpha matte around the moving object and composite it with a still background frame.[1] Recently, Tompkin et al. [2011] presented a more automated technique for creating such simple cinemagraphs. They focus on stabilizing camera motion in handheld video, masking moving objects and generating seamless loops. These approaches cannot de-animate large-scale motions of specific objects while leaving the finer-scale motions intact. Video textures [Schödl et al. 2000; Kwatra et al. 2003; Agarwala et al. 2005] are a well-known approach for seamlessly looping stochastic motions. Like cinemagraphs, a video texture lies between photography and video, but a video texture does not suppress distracting motions within the frame. Another approach to creating cinemagraphs is to start with a still image, manually segment it into objects, and then add motion using procedural motion templates [Chuang et al. 2005]. In contrast, we focus on creating cinemagraphs from video by removing the motion of objects, instead of adding motion to an image.

There are now several publicly available tools for creating cinemagraphs that were developed concurrently with our work, such as Cliplets from Microsoft Research[2], and several iphone apps[3]. However, these tools do not attempt to remove the large-scale motion of specific objects, and thus are not successful on most of our examples (we provide a comparison on the project web page).

Our technical approach is based on two methods introduced in previous work. First, we remove motion of an object using a content-preserving warp, similar to Liu et al. [2009]. However, since we are immobilizing user-specified objects, rather than smoothing camera motion, the overall energy function we minimize is different. We must incorporate the influence of user-drawn strokes on a single frame across the entire video, and therefore require a multi-stage, global solution over all video frames rather than a frame-by-frame approach. Second, we composite video regions spatially and temporally using graph-cuts [Boykov et al. 2001; Kwatra et al. 2003]. Like Agarwala et al. [2005], we composite both static (single-frame) and video regions; however, our energy function is significantly different, since we incorporate constraints from user-drawn strokes, and we can utilize additional information about the location of occluding boundaries.

An alternative to graph-cut-based seamless compositing would be to extract a temporally-varying alpha matte for the user-selected foreground with video matting techniques [Chuang et al. 2002; Bai et al. 2009]. We could then immobilize the foreground, construct a background "clean plate," and re-composite. We chose the seamless compositing approach because current video matting techniques are still effort-intensive, and in our case the colors and textures of the warped and un-warped videos are already similar.

While it is currently possible to de-animate video manually in video effects tools, like Adobe After Effects, the typical pipeline would involve tracking several manually identified points, attaching affine transforms to those points, and then performing a frame-by-frame mesh-based warp to manually remove any motion not modeled well by an affine transform. Overall this process would be extremely laborious and prone to human error. As we show in Section 7, automatic stabilization methods like After Effects' Warp Stabilizer [Liu et al. 2011] are primarily designed to smooth overall camera motion and often fail when used to immobilize specific objects.

[1] http://fernandojbaez.com/cinemagraph-tutorial

[2] http://research.microsoft.com/en-us/um/redmond/projects/cliplets/

[3] http://kinotopic.com, http://cinemagr.am, http://www.icinegraph.com

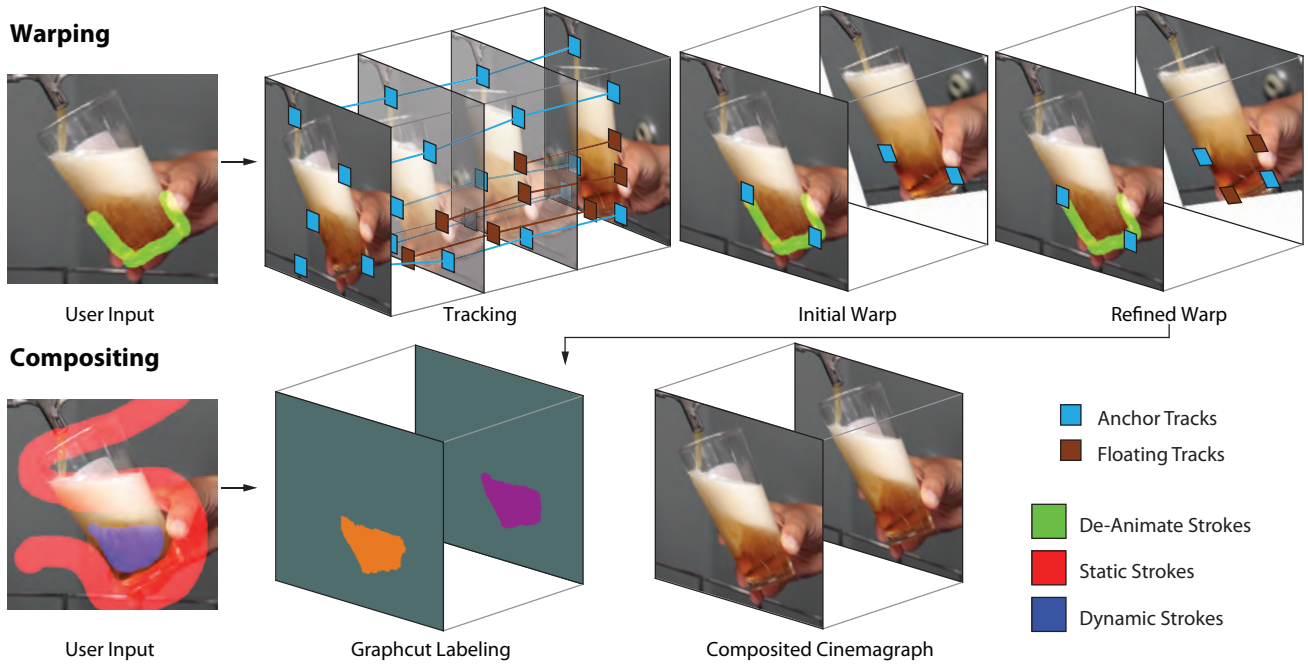


Figure 2: *A diagram of our overall system. The algorithm proceeds in two stages, Warping (Section 5) and Compositing (Section 6), and utilizes three types of user-drawn strokes on a single reference frame (left).*

3 User Input

We designed our de-animation method to give users creative control over the output while minimizing repetitive effort. Users must specify three types of information using a coarse set of strokes on a single frame of the input video; (1) which regions of the video should be de-animated so that their large-scale motions are eliminated (Figure 2, green strokes), (2) which regions of the video should be held static (red strokes) and (3) which should remain dynamic (blue strokes). Note that we use the terms *de-animated* and *static* to refer to two different type of output regions in the output video. De-animated regions have been warped to remove large-scale motions, and may be either static (e.g., completely still) or dynamic (e.g., moving) in the final output. For example, in Figure 1, the large-scale motions of the guitar are de-animated, *but* its fine-scale internal motions, such as the motions of its strings, remain dynamic. The teacher's arms should also remain fully dynamic and the large-scale motions are *not* removed. Other regions, like the head, are completely static in the final video.

To make the task easier for our algorithm, we ask the user to draw green de-animate strokes on only the most static regions of the object, avoiding any internal motions that should be preserved (Figures 2, 5 and 6). Blue strokes indicate output regions that should remain dynamic, while red strokes indicate those that should be completely static. Finally, the user can specify whether the final output should loop seamlessly, and a minimum duration for the loop. Such looping is commonly used to create cinemagraphs.

4 Overview

As shown in Figure 2, our algorithm has two main stages; warping (Section 5) and compositing (Section 6). The warping stage first tracks feature points through the input video. It uses a subset of the tracks to compute an initial warp and then refines the warp with additional tracks. The compositing stage combines the warped video with static frames of the unwarped video to eliminate undesirable motions in regions that the user specifies should remain still (red strokes) while preserving motions in regions that should remain dynamic (blue strokes). The result is a de-animated video, in which the large-scale motion of one or more objects have been removed, and parts of the video are completely static.

Our approach relies on a few assumptions about the input video. First, we avoid the issue of camera stabilization by assuming the input was captured with a tripod, or previously stabilized. Second,we assume the large-scale motions can be de-animated with 2D warps of the video frames and that complex 3D analysis to reverse self-occlusions is not necessary. Third, we assume the objects to be de-animated are shot in front of a defocused, uniform, or uniformly-textured background. As we will show, this assumption allows our algorithms to de-animate objects without building accurate alpha mattes for them. Despite these assumptions we have found that we can apply our technique to a large class of videos.

5 Warping

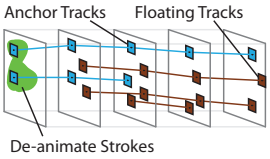
Our algorithm begins by tracking a set of points throughout the sequence. We use Kanade-Lucas-Tomasi (KLT) tracking [Lucas and Kanade 1981; Shi and Tomasi 1994] to follow distinctive points in the input video $I(x, y, t)$. Tracks may begin or end at any frame, and most tracks do not last throughout the entire video. We define the set of tracks as a table of 2D coordinates $K(s, t)$, where s is a unique index for each track, and t indicates time (frame number); $K(s, t) = \emptyset$ if track s is not present on frame t. Since very short tracks are often unreliable, we remove tracks whose duration is less than 15% of the input video duration.

The user draws green strokes on reference frame t_a of the input video. The goal of our warping algorithm is to warp the video so that the content indicated by the green de-animate strokes remains spatially fixed to that location for all frames of the output video. We define $K'(s, t)$ as the location of the tracks after warping, and $K_G(s, t)$ as the subset of the tracks that lie on the indicated content.

Similarly, $K'_G(s,t)$ are the locations of these tracks after warping. The goal of our warp can be expressed mathematically as,

$$K'_G(s,t) = K_G(s,t_a). \quad (1)$$

How do we identify the tracks K_G? For tracks that exist on the reference frame t_a, ($K(s,t_a) \neq \emptyset$), we simply check if they lie within the green strokes. We call these *anchor tracks*, and collect them into the subset $K_A(s,t)$ (see inset). The task is more complicated for tracks that do not exist on frame t_a, but still lie on the content marked in green. We call these *floating tracks*, or $K_F(s,t)$, so that $K_G = K_A \cup K_F$. If we knew how to warp the video to immobilize this content then we could simply check for tracks that fall in the same green strokes across the entire video to identify floating tracks.



Since we don't know the warp a priori, our approach is to compute two consecutive warps. The initial, approximate warp uses only the anchor tracks, i.e., the constraint $K'_A(s,t) = K_A(s,t_a)$. If enough anchor tracks exist throughout the entire input video, this initial warp can successfully immobilize the content indicated by green strokes. It is more common, however, for the anchor tracks to last a shorter duration. We therefore use the initial warp to identify additional floating tracks (e.g. the set of tracks that lie in the green strokes after the initial warp) and then compute a more accurate, refined warp using both sets of tracks.

5.1 Initial Warp

Given just the anchor tracks $K_A(s,t)$, our goal is to warp the video so that $K'_A(s,t) = K_A(s,t_a)$. One approach is to simply fit a full-frame warp like a homography to each frame that minimizes the squared sum of errors of this constraint. This approach can work well for some scenes, but others, which are less rigid and planar, require a more flexible representation (Figure 11). We therefore use a spatially-varying warp defined on a rectilinear mesh placed on each video frame, similar to the approach of Liu et al. [2009]. However, in their case the main warping constraint is given by the output of a 3D reconstruction, while in our case, the goal is to warp each output frame so that the anchors align with the reference frame.

More specifically, we create a 64×32 rectilinear grid mesh on each input frame, with vertices $V(i,j,t)$ and use it to compute an output mesh $V'(i,j,t)$. We can express each track $K(s,t)$ as a bilinear combination of the four vertices of the quad enclosing the track on frame t. We define $\mathbf{V}(s,t)$ as the vector of these four vertices; $\mathbf{V}'(s,t)$ represents the same four vertices on the output grid. The vector $\mathbf{w}(s,t)$ contains the four bilinear interpolation weights that sum to 1, so that $K(s,t) = \mathbf{w}(s,t) \cdot \mathbf{V}(s,t)$ on the input mesh, and $K'(s,t) = \mathbf{w}(s,t) \cdot \mathbf{V}'(s,t)$ on the output mesh we are solving for. The main constraint (Eqn. 1) of the initial warp then becomes

$$E_a = \sum_{s \in K_A, t} l(s,t) |K_A(s,t_a) - \mathbf{w}(s,t) \cdot \mathbf{V}'(s,t)|^2, \quad (2)$$

where the unknowns are the output mesh vertices contained in $\mathbf{V}'$. We use the weighting function $l(s,t)$ of Liu et al. [2009], to temporally fade-in and fade-out the strength of the constraints from each track to preserve temporal coherence. (The weights vary from 0 to 1 with a linear ramp window of 15 frames for fading-in and fading-out when the track points appear and disappear).

We include an additional term E_s on the warp to reduce visible distortion by encouraging each quad to undergo a similarity transform. We use the formulation given by Liu et al. [2009], but describe it in the Appendix for completeness. The final energy function is

$$E = E_a + \omega E_s. \quad (3)$$

We set the weight $\omega = 4$ through trial-and-error. Since the energy function is a sum of quadratic terms, we minimize it with respect to V' using least squares to solve for the output grid mesh. Finally, we render the output sequence by texture mapping the input frames onto the output grid mesh.

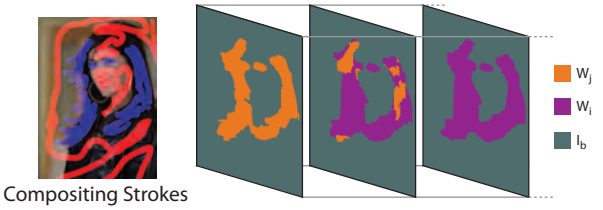


Figure 3: *The computed labeling for three frames of a cinemagraph. In this example, graph cut chooses a single still frame label (gray) for the background and static regions of the model, and two dynamic candidates (purple and orange) for the hair and eyes.*

5.2 Refined Warp

To further improve the result we use floating tracks. We first eliminate floating tracks that fall outside the green strokes. We use the output of the initial warp, which approximately aligns each video frame with the reference frame t_a, and include a warped floating track $K'_F(s,t)$ only if it falls within the green strokes (since the track may vary with time, we require it to lie within the strokes for at least 75% of its duration).

We cannot use the constraint in Eqn. 2 for floating tracks, since we do not know their locations $K_F(s,t_a)$ in the reference frame. Instead, we constrain the floating tracks to remain fixed in space over time

$$E_f = \sum_{s \in K_F, t} l(s,t) |\mathbf{w}(s,t) \cdot \mathbf{V}'(s,t) - \mathbf{w}(s,t+1) \cdot \mathbf{V}'(s,t+1)|^2. \quad (4)$$

Since this constraint includes unknown mesh vertices at different moments of time, we cannot solve for each frame's output mesh separately, as in Eqn. 3 and in Liu et al. [2009]. We therefore solve for the entire output video mesh simultaneously using least squares. The final energy function we minimize to compute the refined warp is

$$E = E_a + E_f + \omega E_s. \quad (5)$$

6 Compositing

The final step is to composite the warped video with static regions taken from a few frames of the input video, in order to remove any extraneous motions outside the immobilized video regions.

We use graph cuts to perform Markov Random Field (MRF) optimization, to seamlessly composite dynamic regions from the warped video with static regions from input frames into a single output video. This stage of the algorithm takes as input the original video $I(x,y,t)$ and the warped video $W(x,y,t)$, and produces a final output video $O(x,y,t)$.

Each pixel p of the output video volume $O(x,y,t)$ is assigned a discrete label $\lambda(p)$ that respects a set of labeling constraints and minimizes an energy function. Each candidate label λ corresponds to a pixel source, which we refer to as a *candidate video volume*.

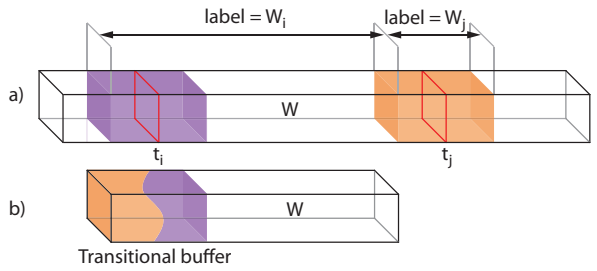


Figure 4: *We select two candidate video volumes, W_i and W_j **(a)**, and overlap them in the output to form a seamless loop **(b)**. Transition regions (purple and orange) surround frames t_i and t_j (red), and W_j is time-shifted to overlap with W_i in the output, so the first and last frames of the loop come from consecutive input frames.*

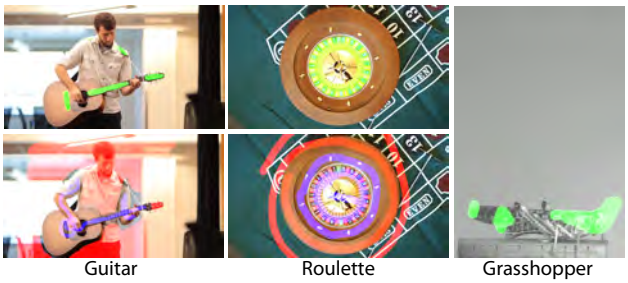


Figure 5: *User provided strokes for our motion visualization examples. Green strokes are for de-animation, while red and blue strokes are for compositing. Note that minimal user input and coarse strokes suffice for our method.*

An example of a candidate video volume is the warped video. The labeling constraints apply to pixels that are indicated by user-drawn red and blue strokes. The energy function measures the degree to which a transition between candidate videos is visually noticeable. The resulting labeling therefore produces an output video that respects the user-drawn strokes, and consists of regions of the candidate videos with seamless, spatio-temporal transitions between them (Figure 3).

If the user specifies that the output should loop seamlessly, we first follow the approach of Schodl et al. [2000] and find a pair of starting and ending frames that are visually similar in the warped video. Specifically, we search for a pair of frames (t_i, t_j) in $W(x, y, t)$ that are at least as far apart as a minimum duration specified by the user, and that minimize the sum of RGB distances between their pixels. We only compare pixels within the blue strokes, since these regions will remain dynamic in the final composite. We then reduce the input video volume to this duration, and set the output duration to the same number of frames.

6.1 Candidate Video Volumes

We define a set of labels $\mathbf{L}$, that represents two types of candidate video volumes: *dynamic candidates* that are copies of the warped video $W(x, y, t)$, and *static candidates* that are still-frames from the input video repeated to fill the duration of the output. The first type allows us to place dynamic, warped video into the parts of the output constrained by blue strokes, while the second type allows us to select static regions for parts of the output constrained by red strokes. We denote the set of dynamic candidates as $\mathbf{W}$ and the set of static candidates as $\mathbf{S}$, so that $\mathbf{L} = \mathbf{W} \cup \mathbf{S}$.

We first add the warped video to the dynamic label set as W_i. If the output should loop seamlessly, we add another copy of the warped video, W_j, to the labels $\mathbf{W}$ as shown in Figure 4. We create a transition region of time (21 frames) surrounding frames t_i and t_j, and the two labels indicate videos beginning at the start of those transition regions. We time-shift W_j to overlap it with W_i in the output, so that the first and last frames of the output loop come from consecutive input frames. The spatio-temporal transition between W_i and W_j within the transitional buffer will be the best seam for looping from frame t_j to frame t_i.

We define the static candidate label and video volume $I_k(x, y, t)$ as a video where the k'th frame of the input video is repeated for the duration of the output. Using all the frames of the input video as candidates is cost-prohibitive, so we evenly sample to select five frames. If b is the time interval that evenly samples the input five times, we define the set of static candidates $\mathbf{S} = \{I_b, I_{2b}, \ldots, I_{5b}\}$.

Finally, the user can optionally choose to include a "clean plate" still frame candidate containing just the background of the scene.

6.2 Labeling Constraints

We create a final composited result by choosing a label $\lambda(x, y, t)$ from the set $\mathbf{L}$ for each pixel. We place four constraints on this labeling. The first two are based on the user-drawn compositing strokes. We represent the strokes as $\upsilon(x, y) \in \{\text{red}, \text{blue}, \emptyset\}$. Blue-stroked pixels must come from dynamic candidates, while red-stroked pixels must come from static candidates.

1. If $\upsilon(x, y) = \text{blue}$, $\lambda(x, y, t) \in \mathbf{W}$
2. If $\upsilon(x, y) = \text{red}$, $\lambda(x, y, t) \in \mathbf{S}$

The second pair of constraints limit the durations of the videos W_i,W_j, and are only used if the output must seamlessly loop. In order to achieve the temporal transition shown in Figure 4b, the first and last frames of the overlapped transition regions must come from W_j and W_i, respectively, except for pixels that are static.

3. $\lambda(x, y, 0) \neq W_i$.
4. $\lambda(x, y, 20) \neq W_j$.

6.3 Energy Function

In order to generate a visually seamless composite video we minimize an energy function that measures the visual noticeability of transitions between candidate video volumes. Like Kwatra et al. [2003], we prefer two kinds of seams; those for which the RGB values are similar across the seam and those that lie along strong edges. However, unique to our application is the knowledge that the dynamic candidates typically contain foreground objects, while the static candidates typically contain background. Transitions between foreground and background along foreground boundary edges are desirable, since they correspond to occlusion boundaries. Therefore, we only use the edge-strength in the dynamic video for seams between dynamic and static candidates.

We define (p_1, p_2) as two neighboring pixels in the output (we use a 6-neighbor system across space and time), and $C(p, \lambda(p))$ as the color of the pixel p in candidate video volume $\lambda(p)$. We use λ_1 and λ_2 as shorthand for $\lambda(p_1)$ and $\lambda(p_2)$, respectively. If the labels of the neighboring pixels are equal ($\lambda_1 = \lambda_2$), the seam cost is zero since no seam exists. Otherwise, the seam cost Φ is

$$\Phi(p_1, p_2, \lambda_1, \lambda_2) = \frac{\gamma(p_1, p_2, \lambda_1, \lambda_2)}{Z(p_1, p_2, \lambda_1, \lambda_2)} \quad (6)$$

$$\gamma(p_1, p_2, \lambda_1, \lambda_2) = |C(p_1, \lambda_1) - C(p_1, \lambda_2)|^2 + |C(p_2, \lambda_1) - C(p_2, \lambda_2)|^2 \quad (7)$$

Figure 6: *User provided strokes for our cinemagraph examples. Green strokes are for de-animation, while red and blue strokes are for compositing. Note that minimal user input and coarse strokes suffice for our method.*

As in Kwatra et al., the seam cost measures RGB differences across the seam ($\gamma(\cdot)$) divided by edge strength ($Z(\cdot)$) across a seam. For our application, we choose to average the edge strength between two static candidates and between two dynamic candidates, and only use edge strength in dynamic candidates for seams between dynamic and static candidates

$$Z(p_1, p_2, \lambda_1, \lambda_2) = \tag{8}$$
$$\begin{cases} \sigma(p_1, p_2, \lambda_1) & \lambda_1 \in \mathbf{W} \wedge \lambda_2 \in \mathbf{S} \\ \sigma(p_1, p_2, \lambda_2) & \lambda_1 \in \mathbf{S} \quad \wedge \lambda_2 \in \mathbf{W} \\ \frac{1}{2}[\sigma(p_1, p_2, \lambda_1) + \sigma(p_1, p_2, \lambda_2)] & \text{Otherwise} \end{cases}$$

where edge strength within a specific video candidate λ is represented by $\sigma(p_1, p_2, \lambda)$, and computed with a 3×3 Sobel filter averaged across RGB.

In total, we seek a labeling that minimizes

$$\sum_{p1,p2} \Phi(p_1, p_2, \lambda_1, \lambda_2) \tag{9}$$

for each pair of neighboring pixels (p_1, p_2), subject to our stated constraints. We minimize this energy function using the alpha expansion algorithm [Boykov et al. 2001]. Once a labeling is computed, we create a final video simply by copying pixels from the candidate videos. An example labeling is shown in Figure 3.

When the length and resolution of the output is too large to compute within reasonable time and memory, we downsample the input so that there are less than 5 million variables in the MRF optimization. If the downsampling factor is less than 8, the full-resolution labeling is created by bicubic interpolation of the low-resolution labeling, which creates feathered transitions between the regions. If the factor is larger, we use hierarchical graph-cuts [Agarwala et al. 2005] at a 4x downsampled resolution before performing a final upsampling with interpolation.

7 Results

We captured a variety of scenes to evaluate and demonstrate our algorithm. Input sequences range from 3 seconds to 12 seconds. We down-sample the frames if necessary such that the height of the video is 720 pixels. We show the input strokes used to create some of our examples in Figures 5 and 6; these strokes are the only user effort required by our system. All examples are shown in the accompanying video. For each, we also provide a video that compares our approach to alternative de-animation methods on our project webpage.

Motion Visualization

Our first four results reveal motion that is otherwise difficult to see (Figure 5).

Figure 7: *The image on the left shows the location of the logo the user places in a single frame. The image on the right shows the logo edit successfully affixed onto the glass 50 frames later. Note that the water has significantly changed.*

Figure 8: *The image on the left shows the animated path the user drew to generate a winning number for roulette. The image on the right shows the same path of the ball in the unwarped video. Notice the complex path the user would have to plan if the inner wheel was not the frame of reference.*

Guitar: We de-animate both the guitar and the shoulder of the musician to clarify the finer-scale motions of his fingers. Our spatially varying warp is able to handle multiple opposing constraints for immobilizing and preserving the shape of both the guitar and shoulder (Figure 1). For this example, we include a clean plate image of the scene as a static candidate during compositing.

Roulette: In roulette, a ball is rolled clockwise along the outer rim while an inner wheel is rotating counter-clockwise. The complicated motions of the ball as it hits the inner wheel makes it difficult to see its path. We de-animate the rotating inner wheel to better reveal the path and physics of the ball and possibly allow the viewer to predict the outcome of the games.

			Comparison Methods			Steps of our Algorithm			Timings for SVWF	
	# of frames	Input	H	AE	AEM	HA	SVWA	SVWF	Warping	Compositing
Beer	75	0.025	0.011	0.022	0.026	0.009	0.009	**0.009**	22 s	404 s
Glass	83	0.020	0.012	0.015	0.017	0.010	0.010	**0.010**	73 s	576 s
Grasshopper	44	0.031	0.013	0.042	0.033	0.016	0.007	**0.006**	14 s	-
Guitar	200	0.040	0.019	0.037	0.055	0.019	0.016	**0.017**	144 s	1287 s
Model K	270	0.026	0.008	0.024	0.031	0.008	0.009	**0.009**	42 s	300 s
Model S	117	0.010	0.002	0.004	0.006	0.002	0.001	**0.001**	260 s	650 s
Roulette	250	0.036	0.061	0.066	0.088	0.028	0.028	**0.028**	204 s	354 s
Snorri	380	0.127	0.063	0.273	0.275	0.063	0.026	**0.028**	959 s	-

Figure 9: *Variances of de-animated results, with respect to the target frame. Values are computed in RGB space, varying from 0 to 1. Leftmost columns present the time required for the warping and compositing steps of the SVWF variant of our algorithm.*

H	Homographies with all tracks
AE	After Effects Warp stabilizer
AEM	After Effects Warp stabilizer with object mask
HA	Homographies with Anchor tracks
SVWA	Spatially Varying Warps with Anchor tracks
SVWF	Spatially Varying Warps with Floating tracks

Figure 10: *Legend of method acronyms used in comparisons.*

Grasshopper: The finer-scale leg motions of a jumping grasshopper can be difficult to see because of the larger-scale motions of its body [Sutton and Burrows 2011]. While entomologists sometimes glue insects to sticks in order to eliminate the large-scale motions and observe fine-scale motions of legs and wings, the glue and sticks can also impact the range of motions an insect can perform. For example, a full jumping motion is impossible while glued to a stick. Instead, we de-animate a video of an unencumbered grasshopper so that the motions of its legs are easier to see at lift-off. It is also easier to compare the length of the legs at full extension to the length of the body when the grasshopper is de-animated. We do not use our compositing technique for this example, and instead just crop the warped video.

Snorricam: A Snorricam[4] is a specially constructed camera device rigged to an actor's body who then moves through a scene. The result is that the actor is mostly static while the scene moves relative to her. We can simulate this effect without special hardware by capturing hand-held video, and then de-animating the actor's body. We show video of such a result on the project web site, as well as average image visualizations of the input and de-animated output; the actor's body is much sharper in our output.

Cinemagraphs

Our next four examples are seamlessly looping cinemagraphs (Figure 6).

Beer and Glass: We have generated two cinemagraphs that immobilize glassware while retaining the dynamic motions of the liquid within. The first example shows beer pouring into a glass while the second shows water swirling in a glass. Because the glassware, hands and all surrounding objects are still, the motions of the liquid are unexpected and surreal. De-animating glassware is challenging because the glass is thin and transparent, and the dynamic liquid within can confuse trackers. Nevertheless our approach is able to properly immobilize the rims of the glassware. We also show a second result for the beer example where we animate the beer flowing from the spout. Since the flow is already immobilized in the input, we use a special, yellow stroke to constrain those pixels to come from the input video during compositing.

[4] http://en.wikipedia.org/wiki/SnorriCam

Model K and Model S: It is particularly challenging to de-animate human faces and bodies because their natural motions involve non-rigid deformations. Our Model K and Model S cinemagraphs eliminate the motion of the faces and bodies while the motions of the eyes and hair remain intact. As in the glassware examples, the juxtaposition of static and dynamic regions is surreal because the resulting cinemagraphs are neither photographs nor videos.

Video Editing

Editing the appearance or motions of objects in video is a challenging problem. With our de-animation techniques users can modify a single reference frame in which the object is immobilized and our system propagates the edits back into the original video. While our approach is inspired by earlier work on editing video by modifying a reference frame [Rav-Acha et al. 2008], we offer a simpler method, but one that cannot handle as significant deformations.

Appearance Editing: In Figure 7, we add a logo onto the moving glass. For the edit to look realistic, the logo must appear to be attached to a fixed location on the surface of the glass. We de-animate the glass, then superimpose the logo at the desired location. When the edited video is un-warped, the logo appears properly attached to the glass. Even the liquid seen through the logo appears natural.

Motion Editing: In Figure 8, we edit the motion of a roulette ball. In the original video the motion of the ball becomes very complicated as it hits the inner rotating wheel. Designing a new path for the ball is extremely challenging because we must account for the rotational motion of the inner wheel. Instead, we first de-animate the inner wheel to provide a still frame of reference and then create a new motion path for the ball with respect to the immobilized wheel. With de-animation we no longer have to account for the relative motion of the inner wheel. Once we are satisfied with the new path of the ball we un-warp the video to obtain the final result. The accompanying video shows the full edit and animation.

8 Evaluation

Warping: Figure 9 quantitatively evaluates the success of the warp in immobilizing user-specified regions. (A table of method acronyms is in Figure 10.) The numbers in Figure 9 are mean RGB pixel variances, computed within the green strokes across all frames of the output video. While lower variance usually implies better de-animation, correlation with our perceptual assessment of the de-animated video is not exact; in particular, brief temporal artifacts can be very noticeable without significantly changing variance. We therefore strongly encourage the reader to view the video on our project page which show a detailed comparison for each sequence.

No existing method is designed specifically for de-animating objects. The closest related problem is video stabilization, which is designed to remove camera motion rather than object motion. We

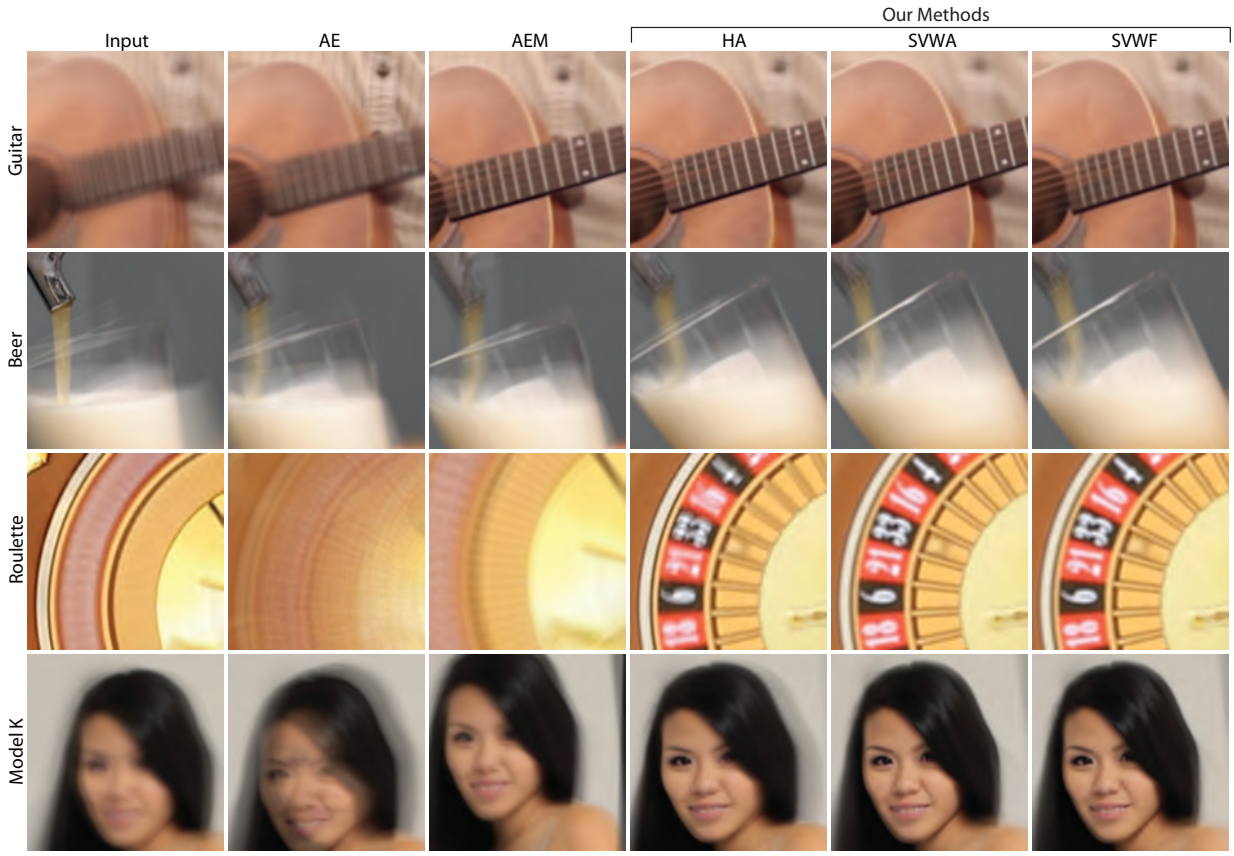


Figure 11: *Average image of de-animated video. Notice the blurring of the guitar, rim artifacts on the beer, blurring of the roulette wheel, and ghosting in model K for the input and several comparison methods. Our result SVWF shown rightmost performs significantly better.*

compare our approach to the Warp Stabilizer [Liu et al. 2011] in Adobe After Effects to show that existing stabilization techniques are not sufficient for our task, with the caveat that we are using the existing Warp Stabilizer for a purpose it was not designed for. We include two types of comparisons; we use Warp Stabilizer directly on either (1) the input video directly (AE) of (2) a version of the input video in which we manually masked out the background so that the stabilizer would eliminate the motion of the desired object (AEM). We compute AE and AEM with the stabilization set to 'no-motion' using 'Subspace warp' with 'detailed analysis'. We also compare with simply computing the best homography, using all tracks (H). In all cases, our algorithm achieves lower numerical variance. From the videos, it is clear that our approach is also qualitatively better at immobilizing user-selected regions. Finally, to visualize our results, we compare the average image after de-animation for some examples in Figure 11. In the ideal case, the average image would be perfectly sharp, and our final method on the right should be compared to the input on the left. However, ghosting and blurring is clearly visible in results for H, AE and AEM.

Figure 9 also evaluates the steps of our warping algorithm. First, we can use just the anchor tracks and apply a simple homography to warp each frame to the reference frame (HA). Our initial warp in Section 5.1 goes beyond a homography to use a spatially-varying warp (SVWA). Our final refined warp in Section 5.2 also uses the floating tracks (SVWF). In some cases, HA is adequate, but for more complex examples like Roulette or Grasshopper (see video), a spatially-varying warp is essential. Even for the Beer example, Figure 11 shows some jerking in the rim. A similar issue arises for Model K, where HA achieves the lowest numerical variance, but contains temporal jerks due to non-rigid deformations, resulting in the ghosting effects for HA in the bottom row of Figure 11. Using only the anchor tracks and SVWA is adequate in many cases, and close to the final numerical variance. However, there are difficult examples like Roulette and Guitar, where we obtain a more accurate result by also using floating tracks.

Compositing: Figure 12 shows the average image for the input, after warping the beer glass (Section 5), and compositing with the still background (Section 6). In the input, large-scale motions of the glass (blurred in the visualization) can mask the internal motion of the liquid. After de-animation, the glass is sharp, but distracting motions (again, blurred in this visualization) are transferred to the background. The final composite appears sharp for both the glass and background. The final video preserves internal motions of the liquid, while removing gross motions of the glass.

User Interaction and Timings: Tracking takes approximately 30 seconds for the longest video. The time for warping and compositing is listed in Figure 9, and currently takes a few minutes. User interaction to draw the coarse strokes can often be done in 1-2 minutes per video. Hence, the complete pipeline to generate a cinemagraph typically takes less than 15 minutes, with only a fraction of that time due to user input. Note that our algorithm is currently unoptimized, and we expect to be able to speed it up considerably in the future. Nevertheless, this workflow is significantly faster than the largely manual process artists must use today. Indeed, it is reported that skilled artists can take several hours, if not a day or two to create complex cinemagraphs[5].

[5] http://www.filmindustrynetwork.biz/nyc-photographer-jamie-beck-cinemagraph/12173

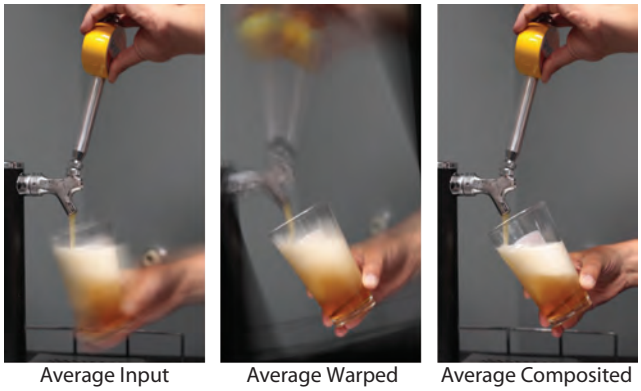


Figure 12: *The average image of the input sequence, warped sequence and the composited sequence shows the effectiveness of combining the still background with the de-animated beer glass while retaining the internal motions of the liquid.*

9 Limitations

There are limits on the operating range of our method. It cannot handle or remove large 3D motions, such as a face rotating from a frontal to a profile view, or the full body of a walking person. An example is shown in Figure 13, where we try to de-animate the subject's face, which undergoes large 3D rotations. The artifacts are easier to understand by watching the video on our project webpage. A good de-animation result should have the large-scale motions of the eyes, nose and mouth removed. However, due to the extreme motions of the face, our system fails to find a spatially varying warp that immobilizes the face. The nose and mouth still exhibit large scale motions as they do not remain in the same spatial location throughout the sequence.

Finally, since our method does not include video matting, we limit the types of background we use for our examples. For example, if our algorithm moves foreground objects relative to their background during warping, and the background is non-uniform, successful compositing might require alpha mattes to composite the de-animated foreground onto still images. Failing to composite only the foreground object from the warped video might result in animated background regions in the output sequence, which is distracting as shown in the Ketchup example in the bottom of Figure 14. In the video on our webpage, the out of focus grid 'swims' as the background region from the warped video is included in the output.

An obvious solution is to include matting in our algorithm, and we made a first attempt to model the foreground and background appearances with Gaussian Mixture Models (GMMs), using an energy term during compositing that prefers cuts along object boundaries. This simple approach is adequate in some cases; in Figure 14, the compositing seam does not cut through the background for Ketchup when we use GMMs. However, GMMs fail for some cases when the foreground and background appearances are very similar. For example, the compositing seam cuts through the hand of the musician in the guitar example instead of its boundary as shown in the top row of Figure 14. This artifact appears because the color of the hand is similar to the color of the wall on the right and the ceiling. In this case, the energy minimization finds a composition that is seamless and obeys the color distribution of the foreground and background but fails to achieve the goal of matting. In summary, including matting in our algorithm remains a difficult challenge and topic of future work; while an approach based on GMMs or an alternative technique might succeed, we do not currently have a robust solution.

Figure 13: *Our method fails to de-animate a face undergoing a large 3D rotation; the nose and mouth are not aligned throughout the output sequence. The image sequence shows the 1^{st},with de-animate strokes overlaid on it, 58^{th} and 83^{rd} warped frame.*

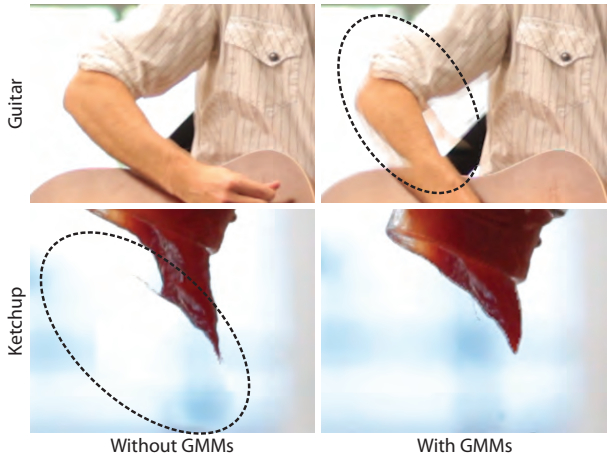


Figure 14: *Our method might include background regions from the warped video in the final composite, causing distracting motions in the output for Ketchup. GMMs can be used to model the foreground and background appearance to achieve a better composition in this case. However, GMMs does not work for all cases as shown in Guitar, where it fails to find the boundary for compositing.*

10 Conclusion and Future Work

There are a number of avenues for future work. Our algorithm minimizes several energy functions in succession; it computes an initial and then a refined warp, and then a composite. We could model the problem as a single energy function, and/or iterate over the various steps. For example, we could reduce the strength of warping constraints for background video regions that are not used in the final composite. However, this approach would increase running time. We could also automatically create specific types of cinemagraphs captured from hand-held devices; for example face portraits could use face-specific techniques to eliminate user effort.

We have presented a simple and largely automated method for selectively de-animating objects in a video, removing gross motions to focus on finer-scale movements. The user need only specify coarse strokes on a single frame of the input. With the increasing use of video as a medium to enhance still imagery, we believe our technique will lead to much future work on analyzing motions to create more compelling and illustrative videos.

Acknowledgements: We thank the reviewers for their detailed comments, and Sean Arietta for playing the guitar for Figure 1. We would also like to thank the models, Kathy Ngo, Sheryl Marie and Christina Russo. This work was supported in part by ONR PECASE grant N00014-09-1-0741, NSF grants CCF-0643552, IIS-1016920, an A*STAR NSS PhD fellowship, gifts and software from Adobe, and the Intel Science and Technology Center for Visual Computing.

References

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM Transactions on Graphics 23*, 3 (Aug.), 294–302.

AGARWALA, A., ZHENG, K. C., PAL, C., AGRAWALA, M., COHEN, M., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2005. Panoramic video textures. *ACM Transactions on Graphics 24*, 3 (Aug.), 821–827.

ASSA, J., CASPI, Y., AND COHEN-OR, D. 2005. Action synopsis: pose selection and illustration. *ACM Transactions on Graphics 24*, 3 (Aug.), 667–676.

BAI, X., WANG, J., SIMONS, D., AND SAPIRO, G. 2009. Video snapcut: Robust video object cutout using localized classifiers. *ACM Transactions on Graphics 28*, 3 (July), 70:1–70:11.

BARNES, C., GOLDMAN, D. B., SHECHTMAN, E., AND FINKELSTEIN, A. 2010. Video tapestries with continuous temporal zoom. *ACM Transactions on Graphics 29*, 4 (July), 89:1–89:9.

BENNETT, E. P., AND MCMILLAN, L. 2007. Computational time-lapse video. *ACM Transactions on Graphics 26*, 3 (July), 102:1–102:6.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 11 (Nov.), 1222–1239.

CASPI, Y., AXELROD, A., MATSUSHITA, Y., AND GAMLIEL, A. 2006. Dynamic stills and clip trailers. *The Visual Computer 22*, 9-10, 642–652.

CHUANG, Y.-Y., AGARWALA, A., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2002. Video matting of complex scenes. *ACM Transactions on Graphics 21*, 3 (July), 243–248.

CHUANG, Y.-Y., GOLDMAN, D. B., ZHENG, K. C., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2005. Animating pictures with stochastic motion textures. *ACM Transactions on Graphics 24*, 3 (Aug.), 853–860.

COHEN, M. F., AND SZELISKI, R. 2006. The moment camera. *Computer 39*, 8, 40–45.

CORREA, C. D., AND MA, K.-L. 2010. Dynamic video narratives. *ACM Transactions on Graphics 29*, 4 (July), 88:1–88:9.

GOLDMAN, D. B., CURLESS, B., SALESIN, D., AND SEITZ, S. M. 2006. Schematic storyboarding for video visualization and editing. *ACM Transactions on Graphics 25*, 3 (July), 862–871.

KIM, B., AND ESSA, I. 2005. Video-based nonphotorealistic and expressive illustration of motion. In *Computer Graphics International 2005*.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3 (July), 277–286.

LIU, C., TORRALBA, A., FREEMAN, W. T., DURAND, F., AND ADELSON, E. H. 2005. Motion magnification. *ACM Transactions on Graphics 24*, 3 (Aug.), 519–526.

LIU, F., GLEICHER, M., JIN, H., AND AGARWALA, A. 2009. Content-preserving warps for 3d video stabilization. *ACM Transactions on Graphics 28*, 3 (July), 44:1–44:9.

LIU, F., GLEICHER, M., WANG, J., JIN, H., AND AGARWALA, A. 2011. Subspace video stabilization. *ACM Transactions on Graphics 30*, 1 (Jan.), 4:1–4:10.

LUCAS, B. D., AND KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. *International Joint Conference on Artificial Intelligence*.

PRITCH, Y., RAV-ACHA, A., AND PELEG, S. 2008. Nonchronological video synopsis and indexing. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 11 (Nov.), 1971–1984.

RAV-ACHA, A., KOHLI, P., ROTHER, C., AND FITZGIBBON, A. 2008. Unwrap mosaics: A new representation for video editing. *ACM Transactions on Graphics 27*, 3 (Aug.), 17:1–17:11.

RUBINSTEIN, M., LIU, C., SAND, P., DURAND, F., AND FREEMAN, W. T. 2011. Motion denoising with application to time-lapse photography. *IEEE Computer Vision and Pattern Recognition (CVPR)* (June), 313–320.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 489–498.

SHI, J., AND TOMASI, C. 1994. Good features to track. In *Computer Vision and Pattern Recognition*, 593 –600.

SUTTON, G. P., AND BURROWS, M. 2011. Biomechanics of jumping in the flea. *J Exp Biol 214*, 5 (Mar.), 836–847.

TOMPKIN, J., PECE, F., SUBR, K., AND KAUTZ, J. 2011. Towards moment imagery: Automatic cinemagraphs. *Visual Media Production, Conference for 0*, 87–93.

TRUONG, B. T., AND VENKATESH, S. 2007. Video abstraction: A systematic review and classification. *ACM Trans. Multimedia Comput. Commun. Appl. 3*, 1 (Feb.).

Appendix

We compute the shape term used in Eqn. 3 by first splitting the input quad mesh into a set of triangles. Since each quad in the input mesh is a square, each triangle is an isosceles right triangle. Considering a single triangle with vertices (V_1, V_2, V_3), if V_2 is opposite the hypotenuse, we can construct vertex V_1 from the other two vertices,

$$V_1 = V_2 + R_{90}(V_3 - V_2), \quad R_{90} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{10}$$

If this triangle undergoes a similarity transform, its vertices after the transform will still follow this equation, and we can measure the deviation of a transformed triangle from a similarity transform by the error in this equation. We can therefore write

$$E_s = \frac{1}{8} \sum_{triangles} ||V_1' - (V_2' + R_{90}(V_3' - V_2')||^2, \tag{11}$$

where the sum ranges over each of the eight triangles containing each vertex. Using all eight triangles is redundant, but avoids special handling at the mesh boundaries.