Responsive Action-based Video Synthesis

Corneliu Ilisescu
University College London
C.Ilisescu@cs.ucl.ac.uk

Halil Aytac Kanaci
University College London
A.Kanaci@cs.ucl.ac.uk

Matteo Romagnoli
Testaluna srl
romagnoli@testaluna.it

Neill D. F. Campbell
University of Bath
N.Campbell@bath.ac.uk

Gabriel J. Brostow
University College London
G.Brostow@cs.ucl.ac.uk

ABSTRACT

We propose technology to enable a new medium of expression, where video elements can be looped, merged, and triggered, interactively. Like audio, video is easy to sample from the real world but hard to segment into clean reusable elements. Reusing a video clip means non-linear editing and compositing with novel footage. The new context dictates how carefully a clip must be prepared, so our end-to-end approach enables previewing and easy iteration.

We convert static-camera videos into loopable sequences, synthesizing them in response to simple end-user requests. This is hard because a) users want essentially semantic-level control over the synthesized video content, and b) automatic loop-finding is brittle and leaves users limited opportunity to work through problems. We propose a human-in-the-loop system where adding effort gives the user progressively more creative control. Artists help us evaluate how our trigger interfaces can be used for authoring of videos and video-performances.

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User interfaces - Prototyping

Author Keywords

Video Editing; Video Textures;
Sprites; Cinemagraphs; Interactive Machine Learning.

INTRODUCTION

In a SIGCHI acceptance speech [21], Dan Olsen outlined the three properties that characterize a great medium of expression: Range, Empowerment, and a Balanced Structure. Good *range* indicates a wide variety of possible expressions, *empowering* mediums lower the required skills and cost to reach excellent results while a *balanced structure* constrains the user to make new outputs possible. By these measures, we find Live Looping [23], where music is recorded and played back in real-time, to be an inspirational medium for authoring music. Present-day musicians like Reggie Watts[1] and Kimbra[2] can

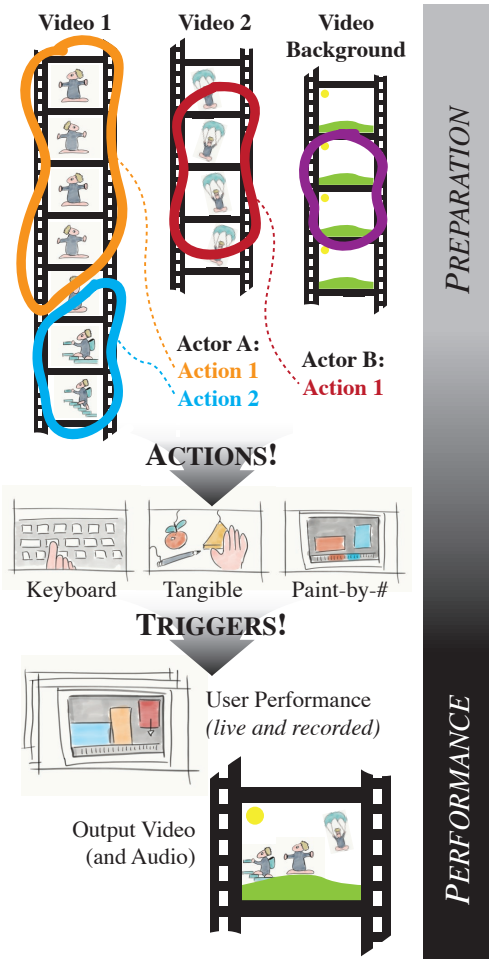


Figure 1: Illustration of how users prepare raw videos to make an interactive live performance. Our end-to-end technology assists with finding and segmenting loopable actions in video inputs (orange, blue, red). Then, discrete but compatible actions can easily be triggered during a show.

[1] https://youtu.be/0gKWfvd-chA?t=123s
[2] https://youtu.be/DgmoHtnoi7k?t=27

This is an author-prepared print that includes the errata mentioned on the project page at visual.cs.ucl.ac.uk/pubs/actionVideo/. The original published version appears in the ACM Digital Library (dx.doi.org/10.1145/3025453.3025880).

easily accumulate simple sounds, faithful to the original audio-clips, yet they have the flexibility and precise control to overlay and repeat clips to compose complex music that transcends their solo-musician appearance. We want to make a "cousin" of Live Looping for the video domain[3], as illustrated in Fig. 1.

Presently, technologies for video-authoring have good Range [21], meaning that they are flexible and accurate in depicting many subjects. But they lack a Balanced Structure and Empowerment, which require confining flexibility to ensure even novices succeed, without curtailing what experts can create. Our goal is to develop a tool that enables a medium of expression characterized by all of these three properties. In particular, we aim to a) "lower the floor" so that novices can participate, b) "raise the ceiling" so that a single artist can compose expressive pieces and performances while c) catering for the widest range of inputs possible.

We achieve this by adapting looping concepts to video in a prepare- and perform-structure (see Fig. 1). In the *preparation* stage, we treat all moving elements as video *sprites*, *i.e.* a bendy tube of pixels in a stack of sequenced images, like Lu *et al.* [19]. We call these *actors*. If a video features only one actor, this is simply a whole-frame sprite. Each tube is then manipulated in time, while maintaining the original spatial properties, to create the output. This is done by splitting a sprite's frames into clusters of *actions*, and allowing artists to choose which subset of frames to show during the second part of our approach: the live *performance*. A user requests actions through a wide range of trigger interfaces, such as tangible widgets, keyboard, or paint-by-numbers, while our system ensures smooth loops within clusters and transitions between them. Artists can also edit synthesis constraints to ensure sensible actor behavior.

In this paper, we present the following contributions:

1. a *constrained optimization algorithm* wrapped inside an expressive new interface, that allows users, for the first time, to control actors in video by high level interactions;

2. an *end-to-end system* to create, iteratively repair, and control video sprites: artists can quickly improve clips that are hard to segment or loop, and check their quality live without jumping between disjoint tool-chains;

3. a *responsive new medium of expression*, that enables artists to merge video assets they made and rehearsed earlier in a live performance.

We assume the input videos to our system adhere to the following criteria: a) the camera is stationary, b) there are no large differences in lighting over the sequence, c) the background is mostly stationary, d) the filmed *actors* are mostly well separated from each other and e) the *actions* they perform are visually distinct. We show how, despite these assumptions, our system can cater for widely different videos featuring various types of *actors* and *actions* and produce rich and diverse results. We also present interviews with several video-artists and evaluate the triggering interfaces to understand the pros and cons of this new medium of expression and adjust the underlying technology.

[3] YouTube's MysteryGuitarMan uses labor-intensive methods and has almost 3M followers: https://youtu.be/EQXA7ErL708

RELATED WORK

In this work, we are inspired by methods for direct video manipulation [8] and navigation [5, 12]. Eventually, we aim to make interactive synthesis of new videos as fun as the manipulation and non-sequential playback of existing videos is in the above methods. We simultaneously tackle the problems of looping arbitrary videos and influencing synthesis of novel content through user annotation. We now highlight works related to those problems and interesting interfaces for video editing.

Video looping and animation

With their pioneering work on *Video Textures*, Schödl *et al.* [28] tackled the problem of indefinitely playing back a finite length video without visible transitions. It worked by finding interchangeable pairs of frames, for single subject videos depicting repetitive or stochastic motion, but struggled with videos containing multiple independent subjects or complex motions.

Kwatra *et al.* [15] treat video looping as a texture synthesis problem, solved using an energy minimization approach. Extensions for panoramic videos and stereo panoramic videos were proposed in [1] and [4] respectively. Liao *et al.* [17, 16] model motion on a per-pixel basis and segment the input video into spatio-temporal regions of similar motion automatically. Finally, Sevilla-Lara *et al.* [29] focus on videos exhibiting camera motion which significantly increases the looping complexity and thus limit themselves to single dominant subjects.

In contrast to the above, our technique enables more meaningful synthesis by introducing additional knowledge rather than simply focusing on loop finding and disguising transitions.

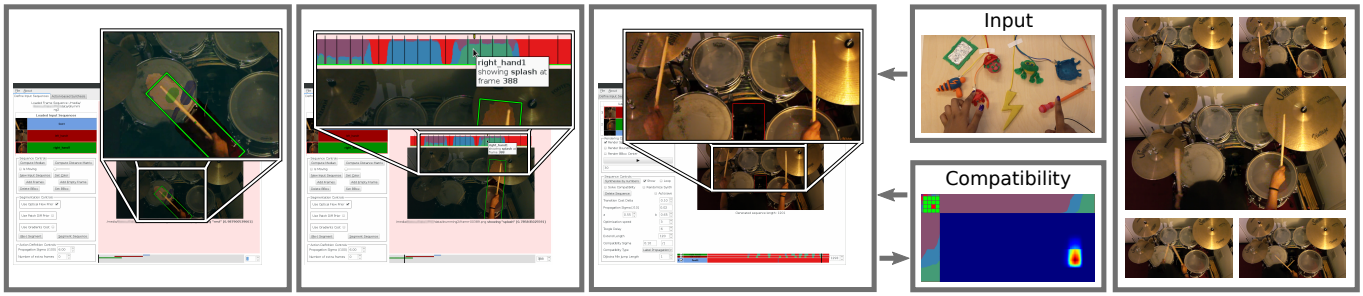
Video-based animation

There are many examples of methods to create novel animations from filmed footage found in the literature. The original *Video Textures* paper by Schödl *et al.* [28] and follow-up work [27], allows segmented sprites of animals, annotated with velocity vectors, to be controlled interactively using the mouse pointer. In contrast, Bhat *et al.* [3] create animations of stochastic elements, such as smoke and water, by leveraging user-defined flow patterns to loop and re-position them. Flagg *et al.* [7] introduce a specialized technique for human video textures that can create animations of human motions from a database.

The systems described above do not have any knowledge about what is taking place in the input video. Therefore, synthesized video elements look plausible, but random. Some indirect user control is possible by making some assumptions and devising custom energy functions (such as looping through more or less heterogeneous frames [17]). In contrast, we generalize by giving users the tools to interactively define their subjects and how to control them "by example".

Video editing

A cinemagraph is a traditionally hand-made medium of expression that combines still and moving imagery. In recent years, much effort [2, 10, 17, 31] has gone into automating this time-consuming process. Users can decide what areas to animate in [31], combine small looped clips called *cliplets* in [10] or scribble over patches to automatically animate or de-animate them [2, 17]. Similar to these works, we reduce the degrees of freedom of captured footage by dividing it into disjoint patches.



(a) Track and Segment (b) Action Definition (c) Responsive Synthesis (d) Rendered Output

Figure 2: Overview of our interactive video synthesis pipeline: (a) The first, *optional*, step is to track and segment the actors we wish to control, such as the two sticks and foot of the drummer in this example. (b) The user defines a set of actions for each actor by tagging example frames. Here, actions are hitting a specific drum or cymbal and resting. (c) A new video is synthesized given input commands mapped to actions and, optionally, frame compatibility information. The compatibility knowledge is learned over time, as the user tags pairs of frames, and the output is changed accordingly. (d) The synthesized sequence is composited and rendered seamlessly (using Poisson Blending [22] and our custom compositing algorithm).

Unlike them, however, we do not restrict users to a binary decision of "(not) animate" and allow them to easily decide *how* to animate through our object-action mapping.

Related to our method, video re-timing allows one to re-order filmed events for new and interesting effects. Shah *et al*. [24] focus on condensing large amounts of video into short animations but often result in disturbing artifacts such as ghosting. Users are given tools to cut and re-arrange trajectories in a spatio-temporal 3D volume in [19, 30]. In *DuctTake* [26], events filmed in the same scene at different times are composed together using a graph-cut energy optimization, while Liao *et al*. [18] build on this by allowing music to drive event re-ordering. Finally, Rav-Acha *et al*. [25] bend time for image patches by projecting their pixels onto evolving time fronts.

In contrast to the above techniques, which do not natively support looping, our method can re-arrange frames arbitrarily. Additionally, unlike in our approach, users must synchronize events manually to avoid incompatibilities, *e.g*. colliding cars [30]; outputs are limited to ones where filmed interactions are preserved [19], or they are not supported altogether [25].

SYSTEM OVERVIEW

We design our end-to-end interface to allow content-creators to quickly prototype their ideas. The more effort they are willing to invest, the higher the quality and complexity their results can achieve. Through discussions with six different technical artists, interactivity (as opposed to automation) and responsiveness were identified as stand-out characteristics of this medium of expression; we emphasize these aspects in our prototype system.

Broadly, videos are prepared before being used in one or more performances. With this in mind, we start by providing the necessary tools to define elements of interest which we call *actors*. These can be full-frame video sequences, such as our Toy and Candle datasets (see Fig. 10 and Tab. 1), or localized objects, such as the cars in Havana or hands in Drumming.

For the objects, we provide semi-automatic tracking and segmentation capabilities (Fig. 2a). We enable the user to correct any mistakes in the bounding box tracks interactively. Similarly, for separating the tracked object from the background, our tool provides previews of generated action video clips, together or in isolation. Users can then correct and influence the quality of the final segmentation by scribbling over the resulting masks.

The next step is the most critical and represents the core of our new medium of expression. Using our simple UI (Fig. 2b), users associate a set of *actions* to each actor, specifying the moment in the video timeline. For instance, each musical note in Toy, or drum hit in Drumming, represents semantically and visually distinct actions. Users define these by tagging a few example frames while the remaining ones are labeled automatically, based on visual similarity, using a machine learning approach. This reduces the required user input and provides almost instant feedback, allowing users to validate the automatic action association and, if necessary, refine it by tagging more examples.

A new video performance synthesizes a number of output layers, each of which corresponds to an actor. Without further guidance, our algorithm can seamlessly loop through the actor input frames by finding visually smooth transitions (similar to [28]). Users can, however, guide the live video performance by pressing keys mapped to actors' actions (Fig. 2c), requesting what to see and when. As we show later, this simple but powerful interaction mechanism enables more creative input methods such as *MakeyMakey* [11], *synthesis-by-numbers* [9] or custom game logic.

Our novel and fast synthesis algorithm balances the importance of meeting users' requests with maintaining the visual quality of loop transitions, to create a new video interactively. Users can further refine the output by tagging incompatible frames or actions, so that actors interact only in desirable ways; for example, diggers should only load parked trucks (see Digger in Fig. 7). Our synthesis algorithm uses this information to improve the resulting output, completing the human-machine feedback loop that makes results possible, in response to high level triggers.

Finally, we can perform an optional post-processing step (Fig. 2d) to improve the quality of the output sequence recorded

during the interactive phase described above, producing the final results shown in the supplemental video. We use seamless blending to remove artifacts due to illumination changes and then merge the actor patches together with the background ensuring that the overlapping regions are handled correctly.

The following sections provide the technical and implementation details required to reproduce our system; these are followed by the results and evaluation.

ACTOR PREPARATION

We now describe the steps and tools used to *prepare* a raw video for use during a live performance. The result of this stage is a set of *actor sequences*: video sprites associated to actions that can be interactively triggered during synthesis. Optionally, actors can be tracked and segmented to improve looping and increase output variability.

Tracking and segmentation

Critical to looping algorithms is the ability to find similar frames or patches, at different points in the timeline, that can be used interchangeably to "jump" between different parts of the video. This is impossible for complex videos, such as ones with multiple, independently moving objects (see HAVANA). Methods such as [17] partially address this problem by adapting their patch shape to best suit looping, but are prone to cutting objects, introducing visible seams. We choose to let users decide interactively which elements they may want at showtime.

First, users track bounding boxes around objects. In our system, we chose to use the CMT tracker [20] because a) it is easy and quick to correct in an interactive setting (see our UI in Fig. 5) and b) it estimates both scale and orientation along with the position of the bounding box.

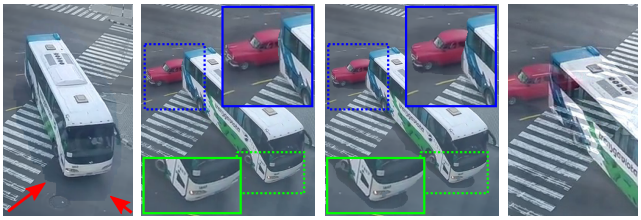
We then use the bounding box to constrain our custom, graphcut-based foreground (FG) segmentation algorithm. Unlike traditional approaches, we aim to composite the patches on their original background (BG). We therefore allow BG pixels to belong to the FG patch as long as *all* FG pixels are correctly classified (see Fig. 3a). To ensure this, users can correct any errors in the labeling by interactively scribbling over patches (the colored strokes in (2) in Fig. 5).

Segmentation algorithm

After estimating the static background as the per-pixel median of all input frames, we use the seam-finding algorithm in *Graphcut textures* [15] to separate FG from BG pixels. We use their pairwise term to conceal seams, and a novel unary term that enforces seam consistency over time and FG pixels to be within the bounding box. Formally, the unary term U for pixel s at position $\mathbf{X}(s)$ belonging to the FG in frame t is defined as

$$U(s,\mathbf{X}) = (1-\alpha)\left[-\frac{1}{2\sigma^2}\left\|\mathbf{X}(s)-\mathbf{X}_c\right\|^2\right] + \alpha\left[1-M_{t-1}\left(\mathcal{F}_{\leftarrow t}(\mathbf{X}(s))\right)\right], \quad (1)$$

where $\mathbf{X}_c = (x_c, y_c)$ are the coordinates of the center of the bounding box in image space, $\mathcal{F}_{\leftarrow t}(\cdot)$ is the optical flow function that maps a pixel to its location in the previous frame [6] and $M_{t-1} \in \{0,1\}$ is the pixel mask (FG/BG) of the previous frame $t-1$. We use $\alpha = 0.35$ against a fixed cost to the BG. User-defined scribbles fix pixels' unary cost depending on their association; see Fig. 3 for an example output.



(a) On BG (b) Ours final (c) Thresholded (d) Mix clone

Figure 3: Result of our user-in-the-loop segmentation procedure and post-process compositing: the raw image patch is placed on the original background (a) and composited using seamless cloning [22] to remove lighting changes w.r.t. BG (red arrows in (a)) and our custom algorithm to resolve occlusions (b). Thresholding the BG difference introduces artifacts (c), while the "mixed seamless cloning" in [22] does not resolve occlusions (d). Input © Brooks Sherman.

Action definition

The main innovation of our paper is the direct mapping between arbitrary, user-defined, semantic actions and video synthesis commands. Users quickly and intuitively guide our synthesis algorithm towards their goal by issuing these commands; for instance, requesting a candle flame to flicker to the right. In contrast, traditional approaches expect users to manipulate the timelines of several clips by cutting, re-arranging and synchronizing them [10, 19]; we believe this makes for far less intuitive and powerful video synthesis.

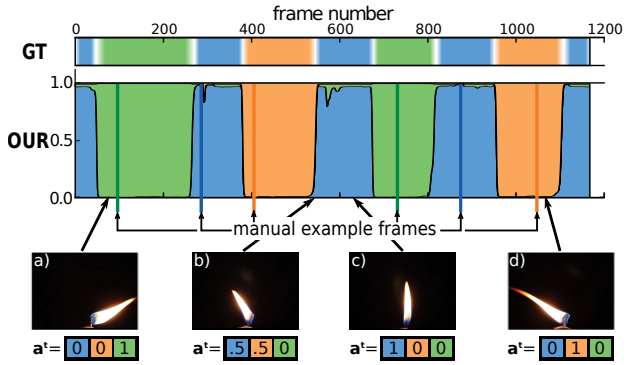


Figure 4: We show the automatically propagated action assignments (OUR) as opposed to the ground truth (GT). Two examples are given manually (vertical lines) for each one of the three actions (denoted with different colors). The values of the action vector $\mathbf{a}^t$ are shown for 4 example frames. Note how frame b) is correctly "softly" assigned to an action between "left" and "rest" (not present in GT).

Action recognition is a well studied problem in the literature. However, existing methods focus on specific use cases, *e.g.* human actions [32]. Not wanting to impose restrictions, we allow users to indicate actions of interest "by example", using

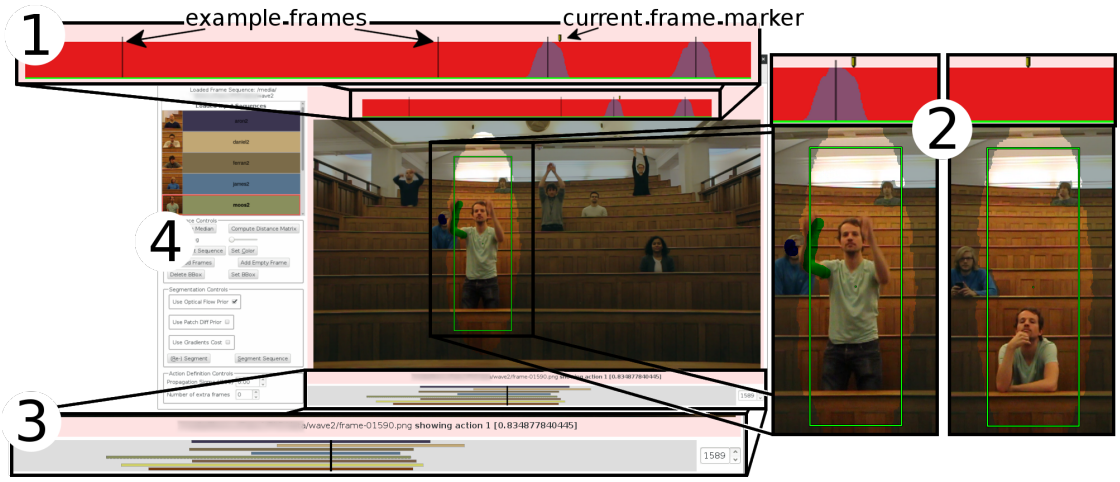


Figure 5: Our actor definition user interface: (1) *actions* associated to each frame of a tracked object (*e.g.* the closest person is shown "sitting" in each red frame and "standing" in each purple frame, with in-between frames shown as a combination of the two colors); (2) two example frames with associated actions (above each and denoted by marker), bounding box and segmentation with corrective strokes (blue = BG, green = FG); (3) *input video timeline*: the black vertical line indicates the current frame and the colored horizontal lines indicate frames where each actor (in this case people) has been tracked; (4) list of tracked actors (identified by their unique color also used in (3)).

our *responsive* interactive tool (Fig. 5). To define actions, users inspect actor sequences (potentially tracked and segmented) and indicate example frames for each action with the press of a button. Users receive *immediate* feedback on the quality of the frame-to-action association of the remaining input frames, as they are automatically compared to the user-given examples.

We can view this as a fuzzy clustering problem, where each action (*e.g.* "sit" and "stand" for WAVE in Fig. 5) is a cluster. In practice, we represent the action visible in frame t of actor sequence $\mathcal{S}$ for which l distinct actions have been defined as an l-dimensional vector $\mathbf{a}^t$. It represents a probability distribution over the action space, so $||\mathbf{a}^t|| = 1$. Intuitively, the higher the value of the l^{th} element of $\mathbf{a}^t$, the more representative is frame t of the l^{th} action. For frames indicated as examples of a given action, $\mathbf{a}^t$ takes the form of a binary vector with a 1 for the specified action and 0's elsewhere. For instance, given the $l = 3$ actions defined for CANDLE (*i.e.* "**rest**", "**left**" and "**right**" in Fig. 4), a confident example frame showing the flame flickering to the left would be associated the action vector $\mathbf{a}^t = [0,1,0]$ (Fig. 4d).

We then quickly propagate the user-given information to the remaining frames using [33]. Action vectors $\mathbf{a}^t$, with $||\mathbf{a}^t|| = 1$, are assigned to all frames, softly clustering them into different actions based on similarity to example frames. The distance between each frame pair (t,t') is defined as

$$D(t,t') = \frac{1}{N_{\text{O}}} \sum_{n=1}^{N} \Big[\mathbf{I}\big(t,\mathbf{X}(n)\big) - \mathbf{I}\big(t',\mathbf{X}(n)\big) \Big]^2, \qquad (2)$$

where we take the $\mathbf{L}_2$ distance between color intensities $\mathbf{I}\big(t,\mathbf{X}(n)\big)$ and $\mathbf{I}\big(t',\mathbf{X}(n)\big)$ of every pixel n. If the actor sequence has been tracked, we first place the frame's segmented patch onto the static background as shown in Fig. 3a. This ensures Eq. 2 can be used for both tracked and full frame sequences, and spatial relationships are preserved. To avoid bias due to camera-related effects, such as foreshortening, we normalize the distance measure by the number of overlapping pixels N_{O} between each frame's bounding box. We set N_{O} to the whole frame area if no bounding box is defined. For space reasons we do not discuss the propagation further. Please see [33] and specifically their Eq.(5) for more details.

Each input frame is associated to an action-cluster "softly" as shown in Fig. 4. This is critical, as frames for which no clear association exists (*e.g.* (Fig. 4b)), are used as in-between transitions by our synthesis algorithm. In contrast, traditional video annotation tools, such as ANVIL [13], enable a similar partitioning of video sequences but with hard boundaries between *manually* defined intervals (see Fig. 4 GT), losing the expressiveness of fuzzy assignments in the process. Moreover,

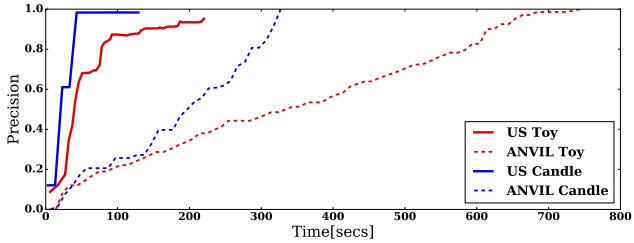


Figure 6: Time we spent labeling actions for datasets TOY (700 frames) with 9 different actions and CANDLE (4000 frames) with 3 different actions using our system or ANVIL [13]. Precision computed w.r.t ANVIL labels.

as shown in Fig. 6, we experienced a 2× to 3× speed-up in reaching the accuracy permitted by ANVIL (and ignoring in-between frames) thanks to the automatic label propagation from [33].

VIDEO PERFORMANCE

In this section, we show how we synthesize a live video performance given a set of input actor sequences. First, users interactively define the frame compatibility measure, which is later used to avoid implausible outputs. Second, we present our optimization strategy that balances user commands, frame compatibility and transition quality information to synthesize new videos.

Frame compatibility

As we will see later, users guide the video synthesis by requesting when actors should perform actions. When multiple actors are present in the same frame however, outputs can exhibit implausible situations depending on when users issue their commands (see HAVANA cars or CANDLE flames). For instance, CANDLE flames could flicker in different directions at the same time (Fig. 10a), a digger could start loading a moving truck (Fig. 7a) or cars could collide in HAVANA (Fig. 8). In our system, these *incompatibilities* take the form of two actors' frames being composited together onto the output background.

In essence, we again want to assign frames to a set of clusters fuzzily, as we did for our action definition. These clusters further decompose the actions into sub-sequences. Users mark them as (in)compatible w.r.t. the sub-sequences of other actors, indicating which sets of frames should be allowed to co-exist in the output video. Given actor sequences $\mathcal{S}_i$ and $\mathcal{S}_j$, we define the compatibility between their respective clusters m and n as

$$B(i,j,m,n) = \begin{cases} 1 & \text{if compatible} \\ 100 & \text{if incompatible} \end{cases}. \qquad (3)$$

Initially, there is one sub-sequence cluster for each user-defined action, so m and n are in the range $[0,l)$. Later, we discuss how users marking (in)compatibilities changes the number of clusters. We initialize $B(i,j,m,n) = 1$ for all combinations of m and n. We use $\mathbf{c}_{i\to j}^{t_i}$ to denote the vector containing the probability that frame t_i of actor $\mathcal{S}_i$ belongs to clusters compatible with actor $\mathcal{S}_j$. Similarly, we define $\mathbf{c}_{j\to i}^{t_j}$ for frames of actor $\mathcal{S}_j$. We initialize $\mathbf{c}_{i\to j}^{t_i} = \mathbf{a}^{t_i}$ as it provides an initial division of the input frames, the combination of which could be incompatible. The compatibility between frame t_i of $\mathcal{S}_i$ and frame t_j of $\mathcal{S}_j$ is then defined as

$$\chi(t_i,t_j) = \sum_m \sum_n \mathbf{c}_{i\to j}^{t_i}[m]\, \mathbf{c}_{j\to i}^{t_j}[n]\, B(i,j,m,n), \qquad (4)$$

where $\mathbf{c}_{i\to j}^{t_i}[m]$ denotes the m^{th} element of $\mathbf{c}_{i\to j}^{t_i}$. Intuitively, the higher the probability that two frames belong to two compatible clusters, the lower the value of $\chi(t_i,t_j)$, denoting a low compatibility cost.

Using our GUI (Fig. 8) at synthesis time, users can tag pairs of frames as compatible or incompatible. Given the pair (t_i,t_j) we allow two options, which we illustrate for t_i only, as they are analogous for t_j:

1. *specialize* the compatibility by using t_i as an example for a new cluster $\tilde{m}$, re-running label propagation using the extended set of examples to compute (the now 1D larger) $\mathbf{c}_{i\to j}^{t_i}$ for all frames of $\mathcal{S}_i$, extending B by one row, setting $B(i,j,\{m \mid m \neq \tilde{m}\},n) = 1$ and $B(i,j,\tilde{m},n)$ according to the user input;

(a) Without **(b) With**

Figure 7: Compatibility illustration. Here, the digger is requested to "load" a truck while the truck is asked to "drive" away in both cases. (a) Without frame compatibility, the two actors are free to perform these incompatible actions, with obvious artifacts. (b) With it, the digger is forced by our algorithm to "load" only when the truck actor is "parked". Input © Perfect Lazybones/Shutterstock.com

2. *refine* the compatibility measure by leaving B unchanged, setting t_i as a new example for the cluster $m = \operatorname{argmax}_m \left[\mathbf{c}_{i\to j}^{t_i}\right]$ it most likely belongs to and, again, re-running label propagation to re-compute $\mathbf{c}_{i\to j}^{t_i}$.

If the compatibility of (t_i,t_j) is changed, we assume option 1, otherwise, the user is asked to decide.

Intuitively, using the example of the two cars at the crossing in Fig. 8, if one chooses to *specialize*, each cluster will contain frames showing the car in different parts of the intersection. All frames showing the cars in the middle of the crossing can then be marked as incompatible and avoided in the output, making our synthesis continuously smarter.

Action-based video synthesis

An output video is composed of the frames of one or more actor sequences, re-arranged to *infinitely loop* showing specific *actions* and the *transitions* between them. We phrase our synthesis process as a labeling problem over a two-dimensional graph with D rows and K columns. Each d row is an output layer that contains the frames $\{t_i\}$ of a specific actor sequence $\mathcal{S}_i$, re-arranged to adapt to the user's commands. Each k column represents a final output frame as the union of the frames chosen for each layer. For instance, the TOY result has one row with output frames straight from the input, while the HAVANA output has as many rows as controllable cars. The label assigned to each (d,k) node is the index of an actor's frame.

Users control the output video by selecting one output layer at a time (see *output timeline* in Fig. 8) and pressing the key associated to the action they want the actor to perform. This in turn defines for each output frame k a D-dimensional requested action vector $\{\mathbf{r}_d^k\}$, $d \in [1,D]$. We use a smooth-step function to automatically switch from the currently shown action to the one associated to the user-pressed key stroke.

Formally, our optimization strategy minimizes the energy function

$$E = \sum_{k=1}^{K} \sum_{d=1}^{D} \alpha E_{\text{A}} + (1-\alpha)\left(\beta E_{\text{C}} + (1-\beta) E_{\text{T}}\right), \qquad (5)$$

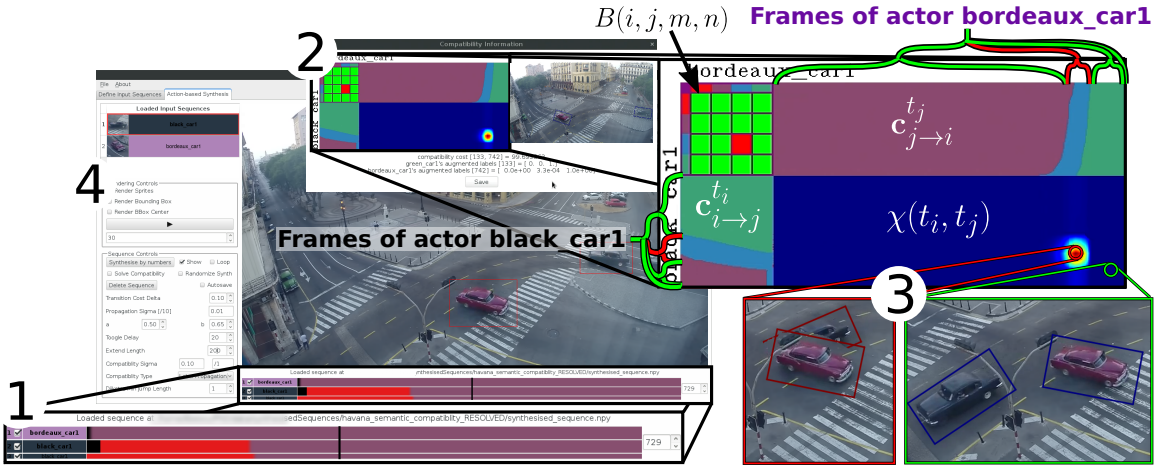


Figure 8: Our video synthesis user interface: (1) *output timeline*: lists the used actor sequences (*e.g.* 1 bordeaux car and 2 black cars) and the color coded user-given commands (*e.g.* red for the car to stay hidden and purple for it to drive through the crossing); (2) *frame compatibility tagging* interface: pairs of frames (previewed to the right of the compatibility info) can be tagged as compatible or incompatible; (3) *compatibility info* for the two selected actors (i and j): frame association to compabtibility clusters per actor ($\mathbf{c}^{t_i}_{i\to j}$ and $\mathbf{c}^{t_j}_{j\to i}$), the cluster-pair compatibility ($B(i,j,m,n)$, see Eq.3), for instance, here, the 3rd cluster of each actor (dark cyan above) are incompatible as denoted by red in the cell $(2,2)$ and the frame compatibility measure $\chi(t_i,t_j)$ from Eq 4 (blue denotes a low cost and red denotes a high cost) (4) list of available actor sequences (added to the output timeline shown in (1)) and synthesis parameters. Input © Brooks Sherman

where E_{A},E_{C} and E_{T} are *action*, *compatibility* and *transition costs* respectively. Intuitively, the higher the value of α, the more responsive the synthesis is to the requested actions (as E_{A} counts more towards total energy) at the expense of looping quality. This is in turn controlled by the other two terms, balanced by β. The higher its value, the more important it is to show compatible frames (E_{C}) at the expense of smooth transitions (E_{T}). Both parameters are user-tuned.

We now define the three components of our energy function. For output layer d, E_{A} is the cost of showing a frame t_i^k of actor sequence $\mathcal{S}_i$, in output frame k, based on whether the action it shows $\mathbf{a}^{t_i^k}$ matches the requested action $\mathbf{r}_d^k$ and is defined as

$$E_{\mathrm{A}}\left(d,t_i^k\right)=\frac{1}{2\sigma_{\mathrm{A}}^2}\left\|\mathbf{a}^{t_i^k}-\mathbf{r}_d^k\right\|^2. \tag{6}$$

E_{C} is the cost of showing a pair of frames t_i^k and t_j^k, from output layers d and d' respectively, in the same output frame k based on the compatibility cost $\chi(\cdot,\cdot)$ from Eq. 4 and is defined as

$$E_{\mathrm{C}}\left(d,t_i^k\right)=\sum_{j\in[1,D]\setminus d}\chi\left(t_i^k,t_j^k\right). \tag{7}$$

Finally, E_{T} is the cost of showing actor $\mathcal{S}_i$'s frame t_i^k in output frame k after showing a frame $t_i^{(k-1)}$ in the previous output frame $(k-1)$ and is defined as

$$E_{\mathrm{T}}\left(d,t_i^k\right)=\exp\left[\frac{1}{\sigma_{\mathrm{T}}^2}D\left(t_i^{(k-1)},t_i^k\right)\right], \tag{8}$$

where $D(\cdot,\cdot)$ is defined in Equation 2.

Real-time performance

The objective in Eq. 5 can be solved using any discrete energy minimization solver that supports multi-label nodes and arbitrary cost functions, *e.g.* TRW-S [14]. Unfortunately, TRW-S fails to converge in our experiments and our system *requires* immediate feedback to enable live performances.

Instead, we propose an iterative approach that minimizes our objective function locally. We define the new energy function

$$E'(d)=\sum_{k=1}^{K}E_{\mathrm{U}}+E_{\mathrm{P}}, \tag{9}$$

and solve it using dynamic programming, one row d at a time, to synthesize K output frames showing the actor associated to the given row. The only inter-row energy is E_{C} (Eq. 7), that ensures compatibility between frames of different actors, which we "bake" into the unary term E_{U}. We define it and pairwise term E_{P} as

$$\begin{aligned}E_{\mathrm{U}}&=\alpha E_{\mathrm{A}}+(1-\alpha)\beta E_{\mathrm{C}},\\ E_{\mathrm{P}}&=(1-\alpha)(1-\beta)E_{\mathrm{T}},\end{aligned} \tag{10}$$

respectively. At each iteration we minimize Eq. 9 for each row in the order the output layers are defined and update E_{U} to consider the already computed rows. We have found that, one iteration is enough to satisfy our three constraints and enables real-time synthesis. Note that, the result seen during a live performance might differ from the one synthesized using the full set of action requests recorded during it. This is due to the fact that, we can better optimize frame compatibilities and transitions between actions, once we exactly know what each actor will be requested to do and when. In our experiments, even long (see Table 1) actor sequences such as the people in

WAVE or the flame in CANDLE never require more than 500ms to synthesize 400 frames. We further drastically reduce these timings (10× to 30×) by using our compression strategy described below. Typically, we synthesize less than 100 frames in < 20ms every 2-3 seconds in a live performance scenario, making our system very reactive.

Optimization Compression

Jumps are rarely perfect, as pointed out by [28], so higher quality outputs involve fewer jumps (*i.e.* the original timeline is followed for as long as possible). With this insight, we speed up our optimization further by synthesizing a subset of the output frames using a subset of the input frames, and "filling in the blanks" using subsequent frames. For instance, we can optimize for half the output frames using only every other input frame and gain a 10× speed-up. We experimented with up to 4× compression, meaning we optimize for every 4^{th} frame, with no visible quality penalty.

Post-Processing Rendering

Our optional post-process first uses seamless cloning by Pérez *et al*. [22], to merge each patch to the static background and remove artifacts (red arrows in Fig. 3a) due to illumination changes. Then, our custom compositing algorithm resolves occlusions between overlapping segmented actor patches.

As shown in Fig. 3a, our segmented patches often contain background pixels. This was deliberate as it allows us to retain small details such as soft shadows, and works well when patches are placed on their original background. When patches of different actors overlap in the synthesized frame (Fig. 3b), BG pixels may obscure FG pixels. For each pixel where this happens, we dynamically decide which patch is most likely to be FG based on its color intensity difference to the BG. This approach is more flexible and gives better results than setting a global threshold on the background difference, as shown in Fig. 3c. It is also more suited to our problem than the "mixed seamless cloning" in [22] which does not perform well with complex backgrounds or occlusions, and introduces ghosting (Fig. 3d).

CREATIVE SYNTHESIS

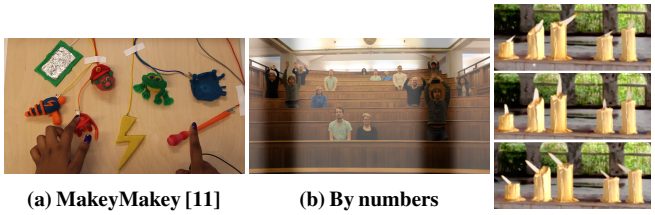
In contrast to existing tools, our system accepts high-level, user-defined commands that guide our video synthesis algorithm. Users need simply request when they want an actor to perform an action, enabling many alternative and fun ways of creating videos, which we now present.

Keyboard and MakeyMakey

The simplest way to create a video with our system is by using a keyboard. An action for each actor can be mapped to a specific key stroke which, when pressed, signals our synthesis algorithm to show frames from that category. Given the simplicity of our mapping process, we can use more creative input methods too. For instance, in Fig. 9a and in our supplemental video, an artist uses MakeyMakey [11] and some *Play-doh* figurines to create a video using our DRUMMING dataset, where specific drums or cymbals are hit when the associated figurine is touched.

Synthesis by numbers

Our system enables creation of video analogous to image synthesis [9]. We associate actions to solid colors, and create



(a) MakeyMakey [11] **(b) By numbers** **(c) Game logic**

Figure 9: Using actions as an abstraction for synthesis commands enables new and creative ways of creating videos. Trigger commands can be given through touching a keyboard or *Play-doh* figurines (a), animated color bars (b) or context specific game logic (c).

an animated control sequence showing those colors using any paint tool. Actions are then triggered according to the colors shown by the control sequence. For instance, Fig. 9b, shows an animated black bar crossing the screen from left to right. At each output frame, people in WAVE are asked to "stand" if they are under the black bar and "sit" otherwise. This allows us to quickly create a Mexican Wave. In our supplemental video we show that we can easily change the control sequence to quickly synthesize completely different waves.

Game Logic

Our system also allows external factors to drive video synthesis. In particular, custom video-game logic can be programmed to issue commands to our synthesis algorithm based on dynamic game-related events. For instance, we have embedded a precomputed set of outputs of our controllable CANDLE into a game level (see Fig. 9c and supplemental video). Then, the game logic decides how the candle should react to its own wind simulation, by for instance, making it flicker to the left or to the right.

RESULTS

The new medium of expression described in this paper enables the creation of a wide variety of video performances. To stimulate the reader's creativity, we and our users have produced a number of output videos using the system. They can be seen in our supplemental video[4], as stills in Fig. 10, and are briefly described here. In Table 1, we provide information about the actors defined for each dataset and the needed user effort.

We use CANDLE as a didactic example. After segmenting the flame using pixel intensity, we define three actions ("left", "right", "rest") and thus are able to have it react according to a hypothetical breeze. Given multiple copies of a candle, as shown in our supplemental video, we also tag pairs of frames showing distinct actions as incompatible, such that our synthesis algorithm can ensure they all react to the breeze randomly, but in the same manner, without having to manually ensure it for each flame. Similar results are achieved from within a videogame level, as shown in Fig. 9c.

Range

HAVANA and THEME PARK show the flexibility of our method and its ability to avoid incompatibilities. These and the subsequent

[4] Visit our website http://visual.cs.ucl.ac.uk/pubs/actionVideo/

(a) CANDLE (b) TOY (c) WAVE (d) HAVANA (e) DRUMMING (f) DIGGER (g) WINDOWS (h) PLANES

Figure 10: Sample output frames. Inputs to (d) © Brooks Sherman, (f) © Perfect Lazybones/Shutterstock.com, (g) © Pavel L/Shutterstock.com, (h) © Cysfilm. Results sequences shown in supplemental video.

Dataset	*Actors*	*Actions*	*Average #frames*	*Avg Prep [s/actor]*	*Output layers*
CANDLE	1	{3}	1168	60	8
TOY	1	{9}	702	210	1
WAVE	18	{2}	1124	1320	15
HAVANA	13	{2}	587	1257	
THEME PARK	13	{1, 2}	630	820	21
DIGGER	2	{2}	160	405	2
WINDOWS	23	{2}	115	60	54
PLANES	7	{2}	105	917	10
DRUMMING	3	{2, 4, 5}	506	2440	3

Table 1: Example input videos. Note how some datasets contain actors with different numbers of actions. For each dataset, we show the average number of frames per actor, average number of seconds necessary to prepare them and the number of layers in the corresponding output video.

examples differ from CANDLE because they cannot or would be very hard to make using existing tools. Multiple moving elements were tracked and segmented, and associated to the actions "visible" and "invisible". Thanks to our frame compatibility measure, we are able to avoid collisions between cars and people when they are visible at the same time in the output video. Similarly, in DIGGER the user ensured a digger only loads a truck when it is parked, by tagging a few incompatible pairs of actor-frames where the truck is moving while the digger tries to load it.

With the remaining datasets, we showcase further creative interactions with our system. Using WAVE, we create a Mexican Wave simply by creating a control animation as seen in Fig. 9b. We can then quickly alter the result by simply changing the control sequence, to add a second subsequent wave, one in the opposite direction, or even an interlaced one. In DRUMMING, we can control a drummer playing his instrument by simply touching *play doh* figurines representing funny sounds, while with TOY we can create a video showing specific songs being played onto a colorful xylophone after filming random notes being hit. WINDOWS allows us to map windows on a building facade to pixels in a grid, and render a compelling game of Tetris by manipulating the light switches. With PLANES, airplanes take off in sync with a user hitting the spacebar to the rhythm of a well known videogame theme-song. Finally, a game-developer used HAVANA to create the CounterLoop videogame (see project webpage). All these outputs and use-cases are prepared with the same workflow, qualitatively demonstrating its range.

EMPOWERMENT EVALUATION

As they are our closest competitors, we aim to replicate our Mexican Wave output (Fig. 10c) using either of [17, 19, 10]. Liao *et al*. [17] can deal with complicated scenes but it merely finds the best possible looping patches. It does not allow the user to choose where to loop or when people should sit and when to stand. The system of Lu *et al*. [19] can allow us to splice together different sub-clips and re-arrange them. However, they create unnatural speed-ups or slow-downs when people need to sit for longer or less than the input video, because of their time scaling algorithm, and do not account for transitions between the clips as our looping does automatically.

We informally compare our system to *Cliplets* [10] by recreating an 8-actor Mexican Wave (see supplemental video) and one candle flame using CANDLE. Similar to [19], *Cliplets* works by defining, manipulating, and arranging layers of video clips to create the output. Each layer shows one looping animation (*e.g*. an actor performing one action) or input frames as captured (which we used to transition between actions of the same actor). For instance, a video of a flame flickering left and then right, requires three layers: a) "loop flame left", b) "playback flame going from left to right", c) "loop flame right". Users must manually define when to show each loop and its length and find transition frames in b) such that there is no visible jump when hiding layer a) to show b) and when hiding b) to show c). This very time consuming process is slowed further if the result needs changes, as changing looping time or animation order requires carefully re-arranging layers and redefining transitions, effectively starting over. In contrast, our system enables live performances after a one-off preparation stage. The output is created as an endless stream and the user is free to play-act and improvise in real time, an invaluable ability unique to our system. Using *Cliplets*, it took us 4× and 9× longer to recreate WAVE and CANDLE respectively. This is mainly due to manually inspecting the video to find the right subset of frames to loop through or use as transitions between animations.

For reasons discussed above, several methods [10, 17, 19] were not able to successfully recreate our outputs. Separately from range, we want to assess whether our action-based synthesis *empowers* users' creativity [21] and helps express it better and faster than baselines. We therefore compare against Adobe After Effects (AE) because, with proper training, it gives users commercially-accepted tools that should reproduce our results. We gathered 6 novice users, that had never used either system, and asked them to recreate the Mexican Wave sequence. In particular, they were instructed to create a left-to-right wave,

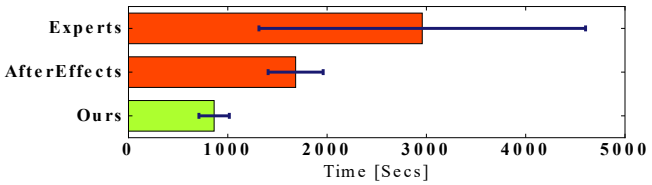


Figure 11: Average timings necessary to replicate a variation of our Mexican Wave result. From top to bottom: *expert* users of NukeStudio, Blender, and AfterEffects; novice users of *AfterEffects*; novice users of *our* system.

Figure 12: An artist using our live performance system.

some idle animation in the middle (*i.e.* sitting people) and a final right-to-left wave. After relevant training, half of the users used our system while the remaining half performed the same task using AE. Both sets of users were given the same 7 sprites as input that had already been tracked and segmented.

Fig. 11 shows that users of our system were roughly twice as fast, indicating that our system is indeed easy to use. In their words, they really enjoyed the simplicity with which actions are defined, the responsive visualization (Fig. 5), and the immediate video feedback that comes with action requests during synthesis. In the supplemental video, we qualitatively show that the AE results are inferior to ours. This is because our system automatically finds the best transitions between actions, while the AE users need to manually align the clips showing the sitting and standing actions and decide when to transition between them. Finally, we also asked three expert Nuke Studio, Blender, and AE users to recreate the same sequence. Their timings (also in Fig. 11) show a larger variability depending on their willingness to find optimal-looking jumps. In fact, they were given the same inputs and task as the novice AE users, but no further instructions, and their results are of varying quality.

DISCUSSIONS WITH ARTISTS

To assess the *balanced* structure [21] of our system, we informally interviewed three digital artists, mostly involved with game development or live performance design, and introduced them to our system (see Fig. 12 using DRUMMING). They agreed that our system takes a large step toward making "video composition more like playing a musical instrument", enabling live performances with immediate video feedback, such as seen in DRUMMING. We were surprised that one asked to sacrifice video quality for better responsiveness, especially if sound feedback is present. As shown in our supplemental video, we were able to cater to this request by favoring E_A(Eq. 5) at the expense of good looking transitions. The result can then be improved, immediately after recording the user commands, by re-synthesizing the sequence with the default video settings.

Interestingly, artists saw our system as a "sketching tool" for quick prototyping, such as seen using *synthesis by numbers* to create the variations of the Mexican WAVE. In fact, experimenting with choreographies was a suggested use case, such as filming dancers improvising and re-arranging their moves using our system after the fact. They also expressed the desire to have the synthesis algorithm as part of game engines, as they feel it gives them important control over sprite synthesis. When shown our CANDLE video, they immediately recognized its value for game development and suggested further content, such as water drops into puddles. Finally, they suggested a number of "shared experiences" for teaching and training that our system would make possible. For example, a trainer could decide which exercises people should perform and give them live video tutorials, or could trigger traffic scenarios.

CONCLUSION

We presented a system that facilitates a new medium of expression, where videos are created much like live looping is composed. Users define actors, optionally tracking and segmenting them, and associate actions to triggers, which can take the form of multiple interfaces. Our workflow helps both novices and advanced users to prepare their footage and, for the first time, turn it into interactive live performances.

Limitations

The quality of our results, ultimately depends on the input videos and the users' willingness to invest the necessary effort to process them. The longer the actor sequences, the greater the variability and coverage of situations. For instance, there are no clean transitions between hitting some notes in TOY and the rest position, because the mallet hand rarely leaves the view in the input video, resulting in occasional jumpy animation. In general, this holds for short videos where there is too much variability but not enough coverage. This problem could be tackled with interpolation and morphing techniques similar in spirit to [29]. Additionally, there is always a trade-off between how quickly the synthesis shows the desired action, and how smooth the transition looks. Being able to successfully camouflage bad jumps would reduce the time necessary to transition between actions. It would also remove the input lag between the button press and the on-screen response, critical for live performances such as DRUMMING.

ACKNOWLEDGMENTS

We would like to thank Mike Terry for his invaluable feedback. Thanks to all the volunteers and artists for their availability during our validation phase, especially Tobias Noller, Evan Raskob and Ralph Wiedemeier. The authors are grateful for the support of EU project CR-PLAY (no 611089) www.cr-play.eu and EPSRC grants EP/K023578/1, EP/K015664/1 and EP/M023281/1.

REFERENCES

1. Aseem Agarwala, Ke Colin Zheng, Chris Pal, Maneesh Agrawala, Michael Cohen, Brian Curless, David Salesin,

and Richard Szeliski. 2005. Panoramic Video Textures. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 821–827. DOI: http://dx.doi.org/10.1145/1186822.1073268

2. Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. 2012. Selectively De-Animating Video. *ACM Transactions on Graphics* (2012). http://graphics.berkeley.edu/papers/Bai-SDV-2012-08/

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 (SIGGRAPH '04)*. ACM, New York, NY, USA, 360–363. DOI:http://dx.doi.org/10.1145/1186562.1015729

4. V. Couture, M.S. Langer, and S. Roy. 2011. Panoramic stereo video textures. In *Computer Vision (ICCV), 2011 IEEE International Conference on*. 1251–1258. DOI: http://dx.doi.org/10.1109/ICCV.2011.6126376

5. Pierre Dragicevic, Gonzalo Ramos, Jacobo Bibliowitcz, Derek Nowrouzezahrai, Ravin Balakrishnan, and Karan Singh. 2008. Video browsing by direct manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 237–246.

6. Gunnar Farnebäck. 2003. Two-frame motion estimation based on polynomial expansion. In *Image Analysis*. Springer, 363–370.

7. Matthew Flagg, Atsushi Nakazawa, Qiushuang Zhang, Sing Bing Kang, Young Kee Ryu, Irfan Essa, and James M. Rehg. 2009. Human Video Textures. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*. ACM, New York, NY, USA, 199–206. DOI: http://dx.doi.org/10.1145/1507149.1507182

8. Dan B Goldman, Chris Gonterman, Brian Curless, David Salesin, and Steven M Seitz. 2008. Video object annotation, navigation, and composition. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. 3–12.

9. Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. 2001. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 327–340.

10. Neel Joshi, Sisil Mehta, Steven Drucker, Eric Stollnitz, Hugues Hoppe, Matt Uyttendaele, and Michael Cohen. 2012. Cliplets: Juxtaposing Still and Dynamic Imagery. UIST. http://research.microsoft.com/apps/pubs/default.aspx?id=183754

11. JoyLabz. 2012. http://makeymakey.com. (2012). http://makeymakey.com

12. Thorsten Karrer, Malte Weiss, Eric Lee, and Jan Borchers. 2008. DRAGON: a direct manipulation interface for frame-accurate in-scene video navigation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 247–250.

13. Michael Kipp. 2014. ANVIL: A Universal Video Research Tool. In *Handbook of Corpus Phonology. Oxford University Press*. Chapter 21, 420–436.

14. Vladimir Kolmogorov. 2006. Convergent tree-reweighted message passing for energy minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 28, 10 (2006), 1568–1583.

15. Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. 2003. Graphcut Textures: Image and Video Synthesis Using Graph Cuts. *ACM Transactions on Graphics, SIGGRAPH 2003* 22, 3 (July 2003), 277–286.

16. Jing Liao, Mark Finch, and Hugues Hoppe. 2015. Fast Computation of Seamless Video Loops. *ACM Trans. Graph.* 34, 6, Article 197 (Oct. 2015), 10 pages. DOI: http://dx.doi.org/10.1145/2816795.2818061

17. Zicheng Liao, Neel Joshi, and Hugues Hoppe. 2013. Automated Video Looping with Progressive Dynamism. *ACM Trans. Graph.* 32, 4, Article 77 (July 2013), 10 pages. DOI:http://dx.doi.org/10.1145/2461912.2461950

18. Zicheng Liao, Yizhou Yu, Bingchen Gong, and Lechao Cheng. 2015. Audeosynth: Music-driven Video Montage. *ACM Trans. Graph.* 34, 4, Article 68 (July 2015), 10 pages. DOI:http://dx.doi.org/10.1145/2766966

19. Shao-Ping Lu, Song-Hai Zhang, Jin Wei, Shi-Min Hu, and Ralph R. Martin. 2013. Timeline Editing of Objects in Video. *IEEE Transactions on Visualization and Computer Graphics* 19, 7 (2013), 1218–1227.

20. Georg Nebehay and Roman Pflugfelder. 2015. Clustering of Static-Adaptive Correspondences for Deformable Object Tracking. In *Computer Vision and Pattern Recognition*. IEEE.

21. Dan Olsen. 2012. CHI 2012 Lifetime Achievement in Research Award. (2012). https://www.youtube.com/watch?v=XDLCaBe_UWo

22. Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson image editing. In *ACM Transactions on Graphics (TOG)*, Vol. 22. ACM, 313–318.

23. Michael Peters. accessed 2016. The Birth of Loop. (accessed 2016). http://www.loopers-delight.com/history/Loophist.html

24. Yael Pritch, Alex Rav-Acha, and Shmuel Peleg. 2008. Nonchronological Video Synopsis and Indexing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 11 (2008), 1971–1984. DOI: http://dx.doi.org/10.1109/TPAMI.2008.29

25. Alex Rav-Acha, Yael Pritch, Dani Lischinski, and Shmuel Peleg. 2005. Evolving time fronts: Spatio-temporal video warping. In *SIGGRAPH'05*.

26. Jan Rüegg, Oliver Wang, Aljoscha Smolic, and Markus Gross. 2013. DuctTake: Spatiotemporal Video Compositing. *Computer Graphics Forum* 32, 2pt1 (2013), 51–61. DOI:http://dx.doi.org/10.1111/cgf.12025

27. Arno Schödl and Irfan A. Essa. 2002. Controlled Animation of Video Sprites. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*. ACM, New York, NY, USA, 121–127. DOI:http://dx.doi.org/10.1145/545261.545281

28. 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 (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 489–498. DOI: http://dx.doi.org/10.1145/344779.345012

29. L. Sevilla-Lara, J. Wulff, K. Sunkavalli, and E. Shechtman. 2015. Smooth Loops from Unconstrained Video. In *Proceedings of the 26th Eurographics Symposium on Rendering (EGSR '15)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 99–107. DOI: http://dx.doi.org/10.1111/cgf.12682

30. R. Shah and P.J. Narayanan. 2013. Interactive Video Manipulation Using Object Trajectories and Scene Backgrounds. *Circuits and Systems for Video Technology, IEEE Transactions on* 23, 9 (Sept 2013), 1565–1576. DOI: http://dx.doi.org/10.1109/TCSVT.2013.2248972

31. James Tompkin, Fabrizio Pece, Kartic Subr, and Jan Kautz. 2011. Towards Moment Images: Automatic Cinemagraphs. In *Visual Media Production (CVMP), 2011 Conference for*. 87–93. DOI: http://dx.doi.org/10.1109/CVMP.2011.16

32. Daniel Weinland, Remi Ronfard, and Edmond Boyer. 2011. A survey of vision-based methods for action representation, segmentation and recognition. *Computer vision and image understanding* 115, 2 (2011), 224–241.

33. Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, and others. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, Vol. 3. 912–919.