Responsive Action-based Video Synthesis

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

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

---

\(^1\)https://youtu.be/0gKWfvd-chA?t=123s
\(^2\)https://youtu.be/DgmoHtnoi7k?t=27

---

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.
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, 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. [20]. 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.

---

### RELATED WORK

In this work, we are inspired by methods for direct video manipulation [8] and navigation [5, 13]. 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. [16] 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. [18, 17] 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 [18]). 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, 11, 18, 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 [11] or scribble over patches to automatically animate or de-animate them [2, 18]. Similar to these works, we reduce the degrees of freedom of captured footage by dividing it into disjoint patches.

---

\[^3\]YouTube’s MysteryGuitarMan uses labor-intensive methods and has almost 3M followers: [https://youtu.be/EQXA7ErL788](https://youtu.be/EQXA7ErL788)
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 [12], synthesis-by-numbers [10] 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 [18] 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 KCF tracker [9] 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 [16] 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 $X(s)$ belonging to the FG in frame $t$ is defined as

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

where $X_c = (x_c, y_c)$ are the coordinates of the center of the bounding box in image space, $\mathcal{F}_{t-1}(\cdot)$ 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 $a=0.35$ 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.

*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 [11, 20]; we believe this makes for far less intuitive and powerful video synthesis.

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...
We then quickly propagate the user-given information to 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)).

We can view this as a fuzzy clustering problem, where each 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 frame-to-action association of the remaining input frames, as they are automatically compared to the user-given examples.

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 $s$ 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

$$D(t,t') = \frac{1}{N_0} \sum_{n=1}^{N} \left| I(t, X(n)) - I(t', X(n)) \right|^2,$$

where we take the $L_2$ distance between color intensities $I(t, X(n))$ and $I(t', X(n))$ 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 $N_0$ between each frame’s bounding box. We set $N_0$ 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 [14], 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 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)).](image)

![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 [14]. Precision computed w.r.t ANVIL labels.](image)
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 $S_i$ and $S_j$, we define the compatibility between their respective clusters $m$ and $n$ as

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

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 $c_{i \rightarrow j}^m$ to denote the vector containing the probability that frame $t_i$ of actor $S_i$ belongs to clusters compatible with actor $S_j$. Similarly, we define $c_{j \rightarrow i}^n$ for frames of actor $S_j$. We initialize $c_{i \rightarrow j}^m = \alpha^m$ as it provides an initial division of the input frames, the combination of which could be incompatible. The compatibility between frame $t_i$ of $S_i$ and frame $t_j$ of $S_j$ is then defined as

$$\chi(t_i,t_j) = \sum_m \sum_n c_{i \rightarrow j}^m[m]c_{j \rightarrow i}^n[n]B(i,j,m,n), \quad (4)$$

where $c_{i \rightarrow j}^m[m]$ denotes the $m^{th}$ element of $c_{i \rightarrow j}^m$. 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) $c_{i \rightarrow j}^{\tilde{m}}$ for all frames of $S_j$, extending $B$ by one row, setting $B(i,j,\{m \neq \tilde{m}\},n) = 1$ and $B(i,j,\tilde{m},n)$ according to the user input;

2. **refine** the compatibility measure by leaving $B$ unchanged, setting $t_i$ as a new example for the cluster $m = \text{argmax} \{c_{i \rightarrow j}^m\}$; it most likely belongs to, and again, re-running label propagation to re-compute $c_{i \rightarrow j}^{t_i}$.

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 $S_i$, 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 $\{r_k^d\}$, $d \in [1,D]$. 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

$$E = \sum_{k=1}^{K} \sum_{d=1}^{D} \alpha E_A + (1-\alpha) \left( \beta E_C + (1-\beta) E_T \right),$$

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
We now define the three components of our energy function. 

\[ E_A(\delta t_i) = \frac{1}{2\sigma_A^2} \left\| \alpha_i^{\delta t_i} - r_{\delta t_i} \right\|^2. \]  (6)

\[ E_C(d, t^k_i) = \sum_{j \in \{1, D\}|d} \chi(t^k_i, t^k_j). \]  (7)

\[ E_T(d, t^k_i) = \exp \left( \frac{1}{\sigma_T^2} D(t^{(k-1)}_i, t^k_i) \right). \]  (8)

where \( E_A, E_C \) and \( E_T \) are action, compatibility and transition costs respectively. Intuitively, the higher the value of \( \alpha \), 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 \( \beta \). 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 new energy function. For output layer \( d \), \( E_A \) is the cost of showing a frame \( t^k_i \) of actor sequence \( S_i \), in output frame \( k \), based on whether the action it shows \( a^{\delta t_i} \) matches the requested action \( r_{\delta t_i} \) and is defined as

\[ E_A(\delta t_i) = \frac{1}{2\sigma_A^2} \left\| \alpha_i^{\delta t_i} - r_{\delta t_i} \right\|^2. \]  (6)

\[ E_C(d, t^k_i) = \sum_{j \in \{1, D\}|d} \chi(t^k_i, t^k_j). \]  (7)

\[ E_T(d, t^k_i) = \exp \left( \frac{1}{\sigma_T^2} D(t^{(k-1)}_i, t^k_i) \right). \]  (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 [15]. 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

\[ E'(d) = \sum_{k=1}^{K} E_U + E_P, \]  (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

\[ E_U = \alpha E_A + (1-\alpha)\beta E_C, \]

\[ E_P = (1-\alpha)(1-\beta) E_T, \]  (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 4th 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 [12] 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 [10]. We associate actions to solid colors, and create 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 pre-computed 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 video4, 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. 9e.

Range
Havana and Theme Park show the flexibility of our method and its ability to avoid incompatibilities. These and the subsequent

4Visit our website http://visual.cs.ucl.ac.uk/pubs/actionVideo/
We can then quickly alter the result by simply changing the Wave simply by creating a control animation as seen in Fig. 9b. With the remaining datasets, we showcase further creative applications and asked them to recreate the Mexican Wave sequence. 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 [11, 18, 20] 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,
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 had 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.cplay.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,


