Hyper-Lapse From Multiple Spatially-Overlapping Videos

Miao Wang, *Member, IEEE*, Jun-Bang Liang, *Member, IEEE*, Song-Hai Zhang, *Member, IEEE*,
Shao-Ping Lu, *Member, IEEE*, Ariel Shamir, *Member, IEEE*,
and Shi-Min Hu, *Senior Member, IEEE*

***Abstract*— Hyper-lapse video with high speed-up rate is an efficient way to overview long videos, such as a human activity in first-person view. Existing hyper-lapse video creation methods produce a fast-forward video effect using only one video source. In this paper, we present a novel hyper-lapse video creation approach based on multiple spatially-overlapping videos. We assume the videos share a common view or location, and find transition points where jumps from one video to another may occur. We represent the collection of videos using a *hyper-lapse transition graph*; the edges between nodes represent possible hyper-lapse frame transitions. To create a hyper-lapse video, a shortest path search is performed on this digraph to optimize frame sampling and assembly simultaneously. Finally, we render the hyper-lapse results using video stabilization and appearance smoothing techniques on the selected frames. Our technique can synthesize novel virtual hyper-lapse routes, which may not exist originally. We show various application results on both indoor and outdoor video collections with static scenes, moving objects, and crowds.**

***Index Terms*— Hyper-lapse video, time-lapse, video editing, video synthesis.**

I. Introduction

WITH the proliferation of shared videos, and as more devices are used to capture egocentric video, it has become much easier to find video collections containing multiple videos sharing a common view or location, e.g. the videos informally captured by tourists at a zoo, a museum or a scenic spot. Our work uses such collections to create continuous hyper-lapse from multiple spatially-overlapping video sources, generating stable navigation results with speed-ups (Figure 1).

Hyper-lapse videos are an efficient way to overview long continuous video sequences. Existing hyper-lapse video creation methods take a single video source and a target speed-up factor s (typically $5\times$ to $30\times$) and produce fast-forward video output. The naive approach to achieve a hyper-lapse video is to evenly sample $\upsilon = s$ video frames. However, such an approach can produce unstable results as any camera shake is amplified. Recent methods have presented non-uniform sampling of frames and measure tempo-spatial coherency to create smooth hyper-lapse output. Our work uses a non-uniform sampling strategy with multiple videos, with different routes and camera velocities, to create the hyper-lapse output.

The challenge when using multiple videos is that frame coherency does not hold any more, and the hyper-lapse camera route is not simple to defi e. When using multiple videos, content may differ, speed may differ, lighting may differ, and the underlying camera routes could differ. Even if the cameras' routes overlap, the input videos are not guaranteed to be synchronized or aligned. Even if they are synchronized, plausible transitions between videos are still needed. Even if such transitions are found, coherence of motion and appearance of the selected frames in multiple video sources may differ. Lastly, the amount of data to be processed in multiple videos presents an additional challenge, requiring efficient sampling and optimization.

To address these challenges, we present a two-stage approach: a pre-processing stage finds commonalities in videos in the collection, then a hyper-lapse creation stage defines a (virtual) camera route and renders the result. In pre-processing, we select candidate videos that potentially have similar views. Each such pair is aligned to find candidate transition points where the two video views are similar enough to permit a transition from one to the other. This information is used to build a graph of transitions for all videos in the collection; it encodes the computational costs related to video stability, velocity and speed-up.

In the hyper-lapse creation stage, there are multiple possible camera routes allowing different hyper-lapse video outputs, corresponding to different paths in the graph. We present several ways to define routes by supporting user-defined constraints through a simple user interface. Given the route definition we formulate multi-source hyper-lapse creation as a shortest path search problem on the transition graph. Optimization is used to find the best path in the graph that meets the user-defined constraints, and is continuous in terms of content,

Manuscript received August 10, 2016; revised February 13, 2017 and August 17, 2017; accepted August 30, 2017. Date of publication September 4, 2017; date of current version January 12, 2018. This work was supported in part by the National Natural Science Foundation of China under Project 61561146393 and Project 61521002, in part by the China Postdoctoral Science Foundation under Project 2016M601032. The work of A. Shamir was supported by the Israel Science Foundation under Project 2216/15. The work of S.-P. Lu was supported in part by VUB (SRP), in part by Innoviris (3DLicornea), and in part by FWO under Grant G025615. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Catarina Brites. *(Corresponding author: Shi-Min Hu.)*

M. Wang, J.-B. Liang, S.-H. Zhang, and S.-M. Hu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: shimin@tsinghua.edu.cn).

S.-P. Lu is with the Department of Electronics and Informatics–ETRO, Vrije Universiteit Brussel, 1050 Brussel, Belgium.

A. Shamir is with the Computer Science Department, The Interdisciplinary Center, Herzliya 4610101, Israel.

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author. The supplemental videos include an algorithm description and experimental results. The PDF includes pseudocode for shortest path optimization. The total size of the video is 0.341 MB. Contact wm.thu@qq.com for further questions about this work.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifie 10.1109/TIP.2017.2749143

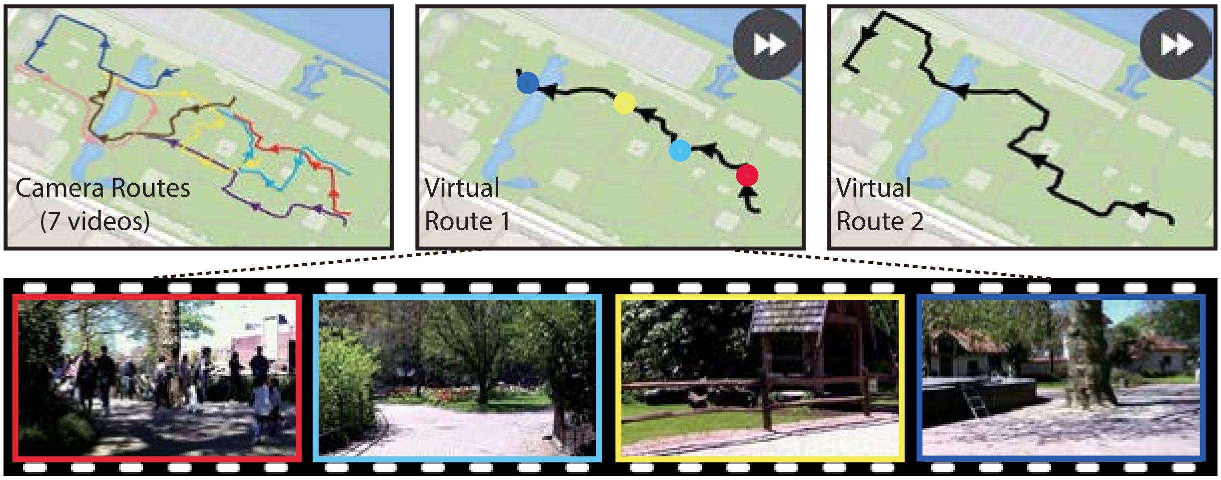


Fig. 1. Given a collection of first-person videos with partially overlapping routes (indicated at the top-left in different colors), our algorithm can generate long hyper-lapse videos by seamlessly combining several camera routes to create a virtual route. Two such routes created from the ‘Zoo’ video collection are shown at top-center and top-right. Representative frames of virtual route 1 are shown in the bottom row; frame borders and positions on the route map are highlighted using the corresponding original route colors.

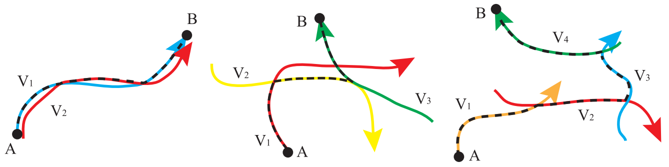


Fig. 2. Different possible scenarios for multi-video hyper-lapse going from point A to point B: multiple overlapping camera routes (left), a complex route combining several videos (middle), and a long virtual camera route (right).

speed, movement and appearance. Our basic scheme extends Joshi et al.’s frame sampling and assembly method [1], but our graph path can use several videos; the hyper-lapse video output has smooth transitions between them. Finally, we render the hyper-lapse video with motion and appearance smoothing to generate a visually pleasing result.

Our method is able to create hyper-lapse videos in various scenarios (Figure 2). First, if several videos have similar camera routes, stable hyper-lapse output is more likely to be synthesized by combining the best part of each video while smoothly transitioning between them. Furthermore, we can also combine several partially-overlapping camera routes to create longer routes, and produce virtual camera routes that no single video portrays. The output hyper-lapse can be customized based on user preferences. For example, the user may prefer hyper-lapse videos having fewer distractions in frames, or which include specified sub-routes; a tool is provided for interactive hyper-lapse navigation.

Our main contribution is a method to create hyper-lapse videos from multiple spatially-overlapping video inputs. Beyond this, our technical contributions are:

- A novel hyper-lapse transition graph data structure that encodes possible *hyper-lapse frame transitions*. This graph structure supports *simultaneous* optimization of a user-specified target virtual route as well as selecting the most visually stable frames in the hyper-lapse result.
- A multiple video hyper-lapse construction technique including a pre-processing strategy for efficient frame alignment in a collection of videos, a temporal distance estimate for cross-video frames, the construction of the transition graph and the optimization of the hyper-lapse objective function using the shortest path on the graph.
- An interactive tool to determine virtual hyper-lapse camera routes by including constraints on videos and frames to include or exclude in the result. A simple visual interface permits the user to add constraints, which are automatically converted to transition graph constraints.

II. Related Work

A. Multi-Video Processing

Several recent works in graphics have utilized multiple media sources to create novel media output or interfaces. Large (internet) photo collections have been used to reconstruct 3D scenes and explore them [2], to create time-lapse videos [3], or simply for summarization [4]. All these works assume that the photos in the collection share a common view or location. In contrast to such dense image collections, video collections are usually sparser. However, they are becoming denser, allowing the use of the same assumption for the creation of novel output or interfaces.

Various appealing editing effects have been realized given multiple related videos. The pioneering Aspen Moviemap [5] interactively generated a dynamic map using videodisc technology and collected images. Pongnumkul *et al.* [6] designed a system which creates storyboards for browsing tour videos with thumbnail shots of sampled locations on a map. Ma *et al.* [7] explored and visualized large-scale video content via VideoMap system. High quality video-based rendering from structured or unstructured videos was explored to interpolate new views [8], [9]. Rüegg *et al.* [10] synthesized novel content by seamlessly blending frames from videos given user interaction concerning alignment and appearance constraints. Wang *et al.* [11] proposed to stabilize video tones by smoothing color states. Chen *et al.* [12] blended objects from multiple sources in the gradient domain. Our method combines several videos, but only uses original footage frames and does not create new viewpoints or blend frames. Arev *et al.* [13] automatically edited footage of multiple social videos. They chose cuts from one video to another according to cinematography guidelines, to provide a better narrative flow and viewing experience. They used a similar graph to ours, but solved a somewhat complementary problem. They searched for cuts to edit a movie, while we search for smooth transition points between videos to create one continuous hyper-lapse.

Tompkin *et al.* [14], [15] explored interactive video navigation within large video collections by seeking scene co-occurrence and smooth transitions between videos. They too used a graph to represent the video collection. In their Videoscape graph, nodes represent potential portals between videos and edges represent video segments. The graph structure provides topological information for navigating between videos. In contrast, our transition graph represents frames as nodes and directly encodes possible *hyper-lapse* transitions between frames as transition edges. The novelty of our graph structure provides the ability to optimize the hyper-lapse route as well as the hyper-lapse frames simultaneously in a single graph optimization procedure.

VideoSnapping [16] temporally aligned multiple asynchronous videos in a global view, and allowed the combination of unsynchronized videos. This work was later used in continuous control of facial performance in videos [17]. Visual Odometry [18] estimated motion from visual input alone. Chen *et al.* proposed matching frames from multiple cameras into a consistent frame [19]. Our approach adapts the VideoSnapping approach to find *candidate hyper-lapse transitions* between videos.

None of these previous approaches provides hyper-lapse output. We present a novel type of multiple-source video processing that uses sampling and assembly of frames from multiple video sources.

B. Time-Lapse and Hyper-Lapse Videos

Time-lapse is an approach to video summarization which displays the main events of a long video or a collection of images in fast-forward. For instance, in [3], internet photos captured at popular sites were mined and combined to generate time-lapse videos. The basic idea of time-lapse video creation is to uniformly skip frames to achieve the required playback speed. More advanced is to use non-uniform sampling of frames [20] to generate results matching visual objectives. Recently, applications based on time-lapse videos have been developed. Shih *et al.* [21] proposed to transfer style of a scene to a different time utilizing a time-lapse video dataset. Lu *et al.* [22] proposed an interactive tool for image composition from time-lapse sequences. Stroboscopic image can be produced from hand-held video [23], where foreground objects flicker several times while background is static.

Hyper-lapse video is a specific form of time-lapse video, in which the original video source usually captures some human activity in first-person view. Because such video is usually captured by a hand-held or helmet camera, the results can be affected by severe shaking. Directly applying time-lapse video creation techniques to such video would amplify the shaking and fail to produce stable results. To address this problem, Kopf et al. synthesized hyper-lapse videos by scene reconstruction and image-based rendering [24]. Using the recovered 3D geometry, their approach first plans a novel smooth camera trajectory then synthesizes corresponding frames by blending content from adjacent original frames. Their approach provides appealing hyper-lapse results, but requires very long processing time. Recently, hyper-lapse creation based on frame sampling was proposed in [1], [25]. The key idea of these works is to find frame matches in a video and then use these to jump to different parts, forming the basis of video textures [26]. Works on hyper-lapse [1], [25] share the same strategy of non-uniformly sampling coherent frames, while using different optimization methods: Joshi et al. use dynamic programming to optimize frame selection [1], while Poleg et al. formulate frame sampling as a shortest path problem [25]. Although generating less stable results than [24], frame-sampling based methods are more efficient and handle dynamic objects better. Our approach is inspired by frame sampling-based hyper-lapse creation methods. However, it cannot be straightforwardly adapted to multiple videos. First, valid candidate hyper-lapse transitions between multiple videos have to be determined carefully for visual smoothness; second, optimization on a large collection of videos is more complicated than on a single video.

Concurrent work closely related to ours concentrates on synthesizing a wider-view hyper-lapse from single and multiple egocentric videos [27]. It uses mostly similar input camera routes to generate a wide view (panorama) of the same route, but may also work with partial overlapping camera routes, like our method. We concentrate on producing alternative routes from diverse input camera routes, allowing different possible input scenarios (Figure 2). To create a multiple video panorama, they use a similar graph structure to ours, but rely on a simpler video correspondence strategy, which only considers the number of matched features without further camera pose validation. In addition, our work allows user-specified hyper-lapse creation using a friendly user interface. Interactive hyper-lapse navigation and route synthesis are controlled by defining special energy terms and a penalty encoding on the transition graph.

C. Video Stabilization

Stabilization techniques estimate the camera trajectory from 2D or 3D perspective and then synthesize a new smooth camera trajectory to remove the undesirable high-frequency motion. 2D video stabilization methods estimate homography or affine transformations between consecutive frames and smooth these transformations temporally. In early work, low-pass filters were applied to individual model parameters [28]. Grundman et al. applied L^1-norm optimization to synthesize a path consisting of simple cinematography motions [29]. Recently, Liu et al. [30] modeled the camera motion on multiple local camera paths. 3D-based stabilization methods reconstruct a 3D scene [2] and estimate the 3D camera trajectory. Liu et al. [31] proposed the first 3D stabilization method using content-preserve warping. The later subspace video stabilization method [32] smooths long tracked features using subspace constraints. It can deal with cases of reconstruction failure, and produces results that are visually as good as a 3D-based method. Recently, a 2D-3D hybrid stabilization approach was proposed to stabilize 360° video [33]. In general, 2D stabilization methods perform efficiently and robustly, while 3D-based methods can generate visually better results.

As demonstrated in [1], directly imposing video stabilization on sped-up videos does not produce stable hyper-lapse results, because motions are significantl amplifie in the sped-up video. So, after assembling the optimal hyper-lapse frame sequence we use stabilization during video rendering to remove the remaining jitter.

III. The Hyper-Lapse Transition Graph

Given a collection of n input videos denoted $\widetilde{V} = \{V_1, \ldots, V_n\}$, where each video V_i is a sequence of frames $V_i = [f_{i,1}, \ldots f_{i,L_i}]$, we aim to sample and assemble T frames $F = \{f_{ø(1)}, \ldots, f_{ø(T)}\}$ from $\widetilde{V}$ to form the hyper-lapse output video, where $ø(k) = (i, j)$ indicates $f_{ø(k)} = f_{i,j} \in V_i$, meaning that the k-th frame of the output video is the j-th frame of the i-th input video in the collection.

To take full advantage of multiple videos, we have to find connections between them so that we can make a smooth transition from one video shot to another. We use the *hyper-lapse transition graph* structure to represent all videos and the transitions between them. Using this graph structure, the multi-video hyper-lapse problem is reduced to finding the shortest path in the graph, as described in Section IV.

The hyper-lapse transition graph is constructed once for each video collection in a pre-processing stage. It may be re-used many times to create various hyper-lapse videos on-line. We first compare all videos and filter irrelevant video pairs to avoid unnecessary computation (Section III-A). Next, we compute temporal local alignments of video frames between pairs of possibly relevant videos (Section III-B). After alignment, we build the hyper-lapse transition graph: each node represents a frame in a video and edges indicate possible transition steps in hyper-lapse paths (Section III-C).

A. Video Pair Filtering

The filtering stage aims to reduce the number of costly computations of video-pair alignment by excluding irrelevant video pairs. We represent each video in the collection by a set of sampled key-frames and find irrelevant video pairs by comparing their key-frames. Each video V_i is sampled using a skip of 29 frames (corresponding to $\sim$ 1 second), and SIFT features [34] are extracted from each key frame. Next, we cluster all extracted feature points in all key-frames into $K = 100$ categories, and compute a normalized K-dimensional bag-of-features histogram [35] for each key-frame. The distance between two key-frames is defined as the chi-square histogram distance, which can be rapidly computed for any pair of key-frames. We compare all pairs of key-frames that originate from different videos. We regard two videos as *irrelevant* if none of their compared key-frame distances is smaller a than threshold $\tau = 0.01$. Such pairs are excluded from the alignment stage.

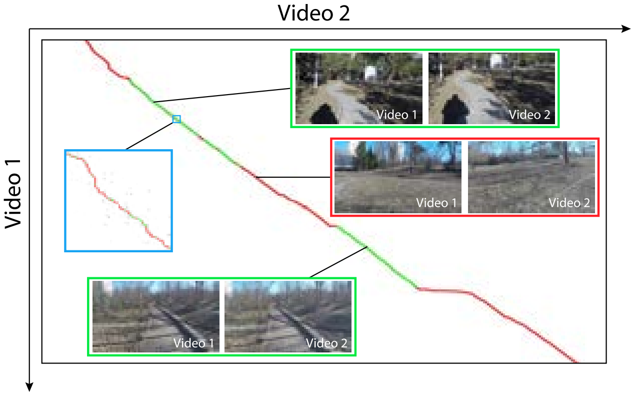


Fig. 3. Video Alignment. The global frame alignment matrix for a pair of videos is shown, where white indicates large cost and black, low cost (locally zoomed-in in the blue window). The curve represents the minimum alignment path. Colors on the path illustrate filtering of the local alignment by our approach (red is out, green is in). Representative frame pairs at various positions are shown.

Optionally, if geographic information is provided e.g. by GPS annotation, we can rapidly reject irrelevant video pairs by considering their geographic coordinates. We assume that candidate alignments only exist in relevant video pairs. This filtering strategy prunes unnecessary computations of alignments between videos.

B. Local Frame Alignments

Given a pair of relevant videos V_i and V_j, we would like to find sub-sequences of frames where good alignment between the videos exists. Along such overlapping sub-sequences we denote each pair of *matched frames* $\langle f_{i,\alpha}, f_{j,\beta} \rangle$, where $f_{i,\alpha} \in V_i$ and $f_{j,\beta} \in V_j$. These matched frames help to provide a smooth and stable transition between the two videos during creation of the multi-video hyper-lapse output, so must be chosen carefully.

First, the content of the two matching frames must be aligned spatially using a simple transformation. In this paper we use a homography to estimate the camera motion and spatial frame alignment. Second, to eliminate the possibility of producing back-and-forth camera motions, we add a constraint that the mappings of frames in an aligned sub-sequences should be temporally non-decreasing. This means that for any two pairs of matched frames $\langle f_{i,\alpha}, f_{j,\beta} \rangle$ and $\langle f_{i,\alpha'}, f_{j,\beta'} \rangle$ for the two videos V_i and V_j, with frame indices α, α' in V_i, and β, β' in V_j, if $\alpha < \alpha'$, then we require $\beta < \beta'$.

To satisfy the above criterion, we discover pairs of matched frames as follows. We start by applying the global video alignment algorithm of VideoSnapping [16] to V_i and V_j. In that work, an alignment cost is assigned to every pair of frames in the two videos, creating a dense cost matrix whose values are in the range of [0, 1]. The globally optimal alignment between the two videos is found using Dijkstra's shortest path algorithm [36], treating the matrix as a planar graph with neighboring matrix elements connected by an edge. Each point on the optimal path is regarded as an alignment between two frames, with a cost c related to the matched SIFT features of the frames. The advantage of VideoSnapping is that a global alignment between V_i and V_j is found even if irrelevant sub-paths exist. The global match also favors matches that are temporally non-decreasing. However, we do aim for a global alignment, but to discover possible transition points between V_i and V_j. Hence, after finding the global alignment path, we filter out irrelevant sub-paths from the match, as well as some unsatisfactory matched frames.

We use a two step-filtering strategy. First, we filter alignments based on the cost metric c, and second, based on the homography between the frames. The cost c of the alignment of two frames is a product of feature descriptor distance and feature position distance. We simply filter out all alignments whose cost is above a given threshold $c > 0.05$. Using this simple strategy, we are able to keep correctly matched frames with similar content. We further filter matching frames by considering the magnitude of the transformation required to align the corresponding frames. We calculate the homography T between the frame pair $\langle f_{i,\alpha}, f_{j,\beta} \rangle$, and estimate the offset of the original frame center $M_{i,\alpha}$ and transformed frame center $T(M_{i,\alpha})$ after fitting it to $f_{j,\beta}$. We filter out all alignments where this offset is above a given threshold (i.e. $\|T(M_{i,\alpha}) - M_{i,\alpha}\| > 0.5d$, where d is the length of the frame diagonal). The thresholds of the filtering strategy were determined empirically, with the same values used throughout all our results.

C. Graph Construction

We now present the *hyper-lapse transition graph*, which is a digraph data structure that plays a key role in multi-video hyper-lapse creation. The transition graph is composed of nodes and directed edges. The node $N_{i,p}$ represents the p-th video frame $f_{i,p} \in V_i$. The graph connects all input videos by densely assigning *transition edges* between nodes. Each directed edge $\langle N_{i,p}, N_{j,q} \rangle$ connects two nodes whose corresponding frames could possibly appear in the output video sequence as consecutive frames. Transition edges are of two types: if $i = j$, the edge connects two nodes from the same video, and if $i \neq j$ the edge connects nodes from different videos (Figure 4).

The first type of edges connect each node forward to all its R following nodes in the same video up to a given temporal distance away, i.e. $\langle N_{i,p}, N_{i,q} \rangle$ where $(q - p) < R$. As the movement in the hyper-lapse video should appear as smooth as possible, one cannot skip too many frames of the input video. In the naive single-video hyper-lapse methods the frame-skip is fixed, but in non-uniform sampling methods, an upper limit is always set for the temporal distance between possible consecutive frames. For nodes from the same video, the temporal distance is defined as $\pi(N_{i,p}, N_{i,q}) = (q-p)$ and is forced not to exceed twice the maximal allowed speed-up $\upsilon_{max} = 30$. This sets the maximum allowed temporal transition distance between transition nodes to be $R = 60$. Note that when using only a single video source, this graph structure and our algorithm reduce to the case of non-uniform single-video hyper-lapse creation.

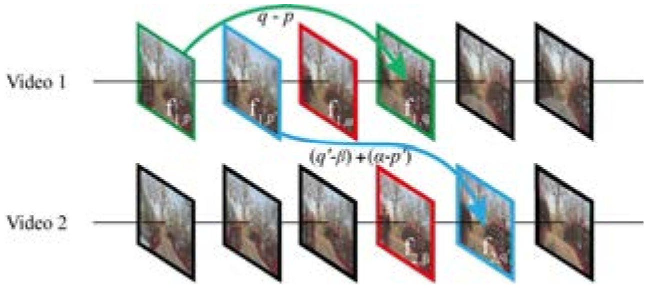


Fig. 4. Two types of transitions. Intra-video transitions between two frames inside a video are illustrated in green: two frames $f_{1,p}$ and $f_{1,q}$ in video 1 with temporal transition distance $q - p$ are connected by an edge. Inter-video transitions between frames from different videos are illustrated in blue: frame $f_{1,p'}$ in video 1 and frame $f_{2,q'}$ in video 2 are connected with temporal transition distance $(q' - \beta) + (\alpha - p')$.

The second type of edges connect nodes for one video to nodes for another. For each pair of matched frames $\langle f_{i,\alpha}, f_{j,\beta} \rangle$ from videos V_i and V_j (Section III-B), we connect an edge between their neighbors $\langle N_{i,p}, N_{j,q} \rangle$ with the following two constraints. First, we only connect neighbors where $p \leq \alpha$ and $q \geq \beta$, so that we only permit transitions between two nodes where the first node is before or at the matched frame in video V_i, and the second node is at or after the matched frame in V_j (note that the corresponding edges $\langle N_{j,q}, N_{i,p} \rangle$, when $q \leq \beta$ and $p \geq \alpha$ also point from V_j to V_i). Second, we also constrain the temporal distance between the transition nodes. However, in this case, as the two nodes represent frames from different videos, the temporal distance between them is not their index difference. We define $\pi(N_{i,p}, N_{j,q}) = (q - \beta) + (\alpha - p)$ and set the maximum allowed temporal distance using the same threshold R.

To summarize, the temporal distance π is defined as:

$$\pi(N_{i,p}, N_{j,q}) = \begin{cases} q - p & i = j \\ (q - \beta) + (\alpha - p) & i \neq j \end{cases} \tag{1}$$

where i and j indicate video sources and $\langle N_{i,\alpha}, N_{j,\beta} \rangle$ is a pair of matched frames in these videos. Figure 4 illustrates these two types of transitions. After collecting all valid transitions with $\pi(\cdot, \cdot) \leq R$, we connect each pair of transition nodes by a directed edge.

The set of nodes N and edges E define the basic structure of the transition graph. Assume for now that a starting node set $N_S \subseteq N$ and ending node set $N_E \subseteq N$ are given (later in Section IV-A we describe several methods to define them). This indicates that the hyper-lapse route can start from any frame node in N_S and end at any frame node in N_E. We introduce two additional virtual nodes: a *virtual start node* $N_{\upsilon s}$, and a *virtual end node* $N_{\upsilon e}$. We connect $N_{\upsilon s}$ to each node in N_S, and each node in N_E to $N_{\upsilon e}$ with *virtual edges* (Figure 5). Using this full graph structure, any path from $N_{\upsilon s}$ to $N_{\upsilon e}$ forms a possible hyper-lapse video F by assembling the video frames of the node sequence of the path. In the next section we introduce our method to find an optimal path for the creation of a hyper-lapse video F.

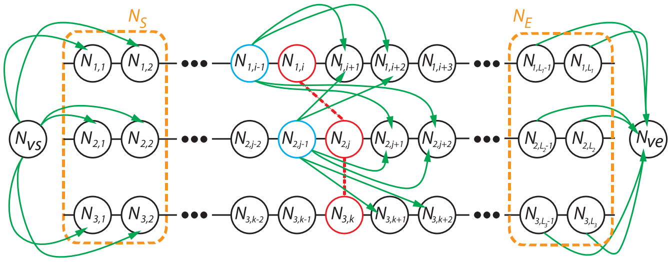


Fig. 5. Example hyper-lapse transition graph structure formed by three videos. Each node corresponds to a video frame. Nodes $N_{1,i}$, $N_{2,j}$ and $N_{3,k}$ are matched frame nodes (red). The edges between the virtual start and end nodes and the start and end node-sets ensure a complete hyper-lapse route. We illustrate only a few of the intra-video and inter-video transition edges: the edges starting from $N_{1,i-1}$ and $N_{2,j-1}$ (blue) with temporal distance $\pi \in \{2, 3\}$ are shown.

IV. Hyper-Lapse Construction

Our hyper-lapse construction method uses frame sampling and assembly from multiple video sources. To find the frames from the various videos, we minimize an energy function by computing the shortest path on the hyper-lapse transition graph. The output of the shortest path algorithm is the hyper-lapse frame sequence F.

A. Route Definition

Because we combine several videos, and each one can have a different camera route, there is a need to define the camera route for the output hyper-lapse video. We do this using user guidance.

In the simplest case, multiple input videos are given with a fully overlapping route. The hyper-lapse route in this case incorporates all the routes of the videos (Figure 2, left). The aim of hyper-lapse creation is to produce a fast-forward result which is as smooth as possible while having the target velocity. This means that the route can start in any video and finish in any video. To define this case, we set the start node sets N_S as the first τ frames of all videos, and the end node sets N_E as the last τ frames of all videos (we use $\tau = 100$). Optionally, the user can specify a specific video in which the route starts or ends.

In cases when the routes of several input videos are partially overlapping, a hyper-lapse video with a *virtual* route is generated. A virtual route is more complex as it is assembled from parts of the original video routes (Figure 2, middle). Without user constraints, our algorithm will simply find the smoothest route starting in any video and ending in any video. However, because we solve a minimization problem, such routes will tend to be shorter. In an extreme case, when a sequence of input videos only share a common view with one camera at the beginning and one camera at the end (Figure 2, right), the default output hyper-lapse will most probably choose only a single camera route.

To overcome this problem, we allow the user to constrain the output hyper-lapse results. As stated earlier, the user can specify a specific video in which the route starts or ends. This can simply solve the extreme case mentioned above by choosing the first and last videos in the sequence. The only possible route that adheres to this constraint is a long virtual route starting with the first camera and ending with the last camera. To allow more freedom, we provide an interface where the user can browse the different videos and mark specific sub-sequences of frames either on a route-map panel or on a frame time-line slider, to ensure that they are either included or excluded from the final hyper-lapse output (Figure 6, and accompanying video).

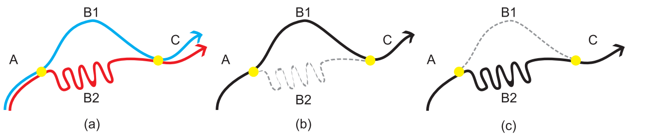


Fig. 6. Possible user constraints: (a) The input camera routes share common sub-routes A and C, and do not share sub-routes B1 and B2. (b) By default, our algorithm selects sub-route B1 during optimization, because it is shorter and smoother. (c) The user can specify that B2 must be included (or that B1 must be excluded), and in this case, the optimal hyper-lapse generated will include B2.

The user constraints are converted to penalties on the corresponding route nodes when optimizing (Section IV-D). In most cases, such penalties guarantee that the final hyper-lapse adheres to these constraints. However, if over-marking is used, there may be cases when the global solution must use excluded frames or may even fail to find a solution. In such cases the user is notified to remove some constraints.

B. Speed-Up Determination

In a single video hyper-lapse, the speed-up factor s can be simply determined by converting it to a frame skip rate $v = s$. However, in multi-source hyper-lapse, the camera movement velocity for different input videos can differ. For example, if we mix videos of running, walking and bicycling, and directly use the frame skip rate $v = s$, the actual velocity of the hyper-lapse camera will be unstable. To obtain coherent velocity for the whole hyper-lapse video, we adapt the time-varying skip rate from single video [1] to multiple videos.

We first estimate the instantaneous camera velocity $\widehat{v_{i,j}}$ of $f_{i,j}$, the j-th frame in V_i, using the average movement of the four frame corners:

$$\widehat{v_{i,j}} = \frac{1}{4}\sum_{k=1}^{4}\|T^{i}_{j,j+1}(X_k) - X_k\|_2, \tag{2}$$

where X_k, $k = 1, \ldots, 4$ are the coordinates of the four frame corners, and $T^{i}_{j,j+1}$ is the homography between frame $f_{i,j}$ and $f_{i,j+1}$. Next, we estimate the average camera velocity v_c for all videos by averaging all instantaneous velocities:

$$v_c = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=1}^{L_i}\widehat{v_{i,j}}}{L_i}, \tag{3}$$

where L_i is the length of the i-th video V_i.

Finally, the time varying frame skip rate for a local frame is defined as $v_{i,j} = \frac{v_c}{\widehat{v_{i,j}}} \cdot s$. Intuitively, this matches the speed-up at a specific frame to the average camera velocity.

To provide a more intuitive velocity setup, we also allow the user to specify a specific video V_k and use its camera velocity as the base velocity. In this case we use $v_c = \frac{\sum_{j=1}^{L_k} \widehat{v_{i,j}}}{L_k}$. Lastly, the user can still use a constant skip rate $v = s$ to allow the original camera velocity for each sub-route, if desired.

C. Formulation

We present the criteria and corresponding cost terms for the assembled output hyper-lapse frames $F = \{f_{\o(1)}, \ldots, f_{\o(T)}\}$. Following [1], we use stability, velocity and acceleration terms, and add the user preferences and minimum duration terms to define the camera route:

1) Visual Stability: The camera motion should be as small as possible between each neighboring output frames $f_{\o(i)}$ and $f_{\o(i+1)}$. To measure this, we assign a cost to each pair of neighboring frames using the offset of the frame center $M_{\o(i)}$ as follows:

$$C_s(f_{\o(i)}, f_{\o(i+1)}) = \min(\|T_{\o(i)}(M_{\o(i)}) - M_{\o(i)}\|_2, d/2),$$

where $T_{\o(i)}$ is the estimated homography which spatially aligns $f_{\o(i)}$ and $f_{\o(i+1)}$ and d is the length of the frame diagonal.

2) Velocity: The hyper-lapse output should obtain the desired speed up. Using the time-varying skip rate $v_{\o(i)}$ for every frame $f_{\o(i)}$ (Section IV-B), this means that $f_{\o(i+1)}$ should be $v_{\o(i)}$ frames away from $f_{\o(i)}$. However, these two frames can also come from different videos. Therefore, we use the temporal transition distance π to assign the following cost to each neighboring output frame:

$$C_v(f_{\o(i)}, f_{\o(i+1)}, v_{\o(i)}) = \|\pi(f_{\o(i+1)}, f_{\o(i)}) - v_{\o(i)}\|_2^2.$$

This measurement penalizes the departure from the required skip rate. In practice, we use a truncated norm, clamping it at 200, following [1].

3) Acceleration Penalty: By balancing the above two criteria, the resulting hyper-lapse may have varying velocity, harming the viewing experience. We penalize this by assigning the cost function:

$$C_a(f_{\o(i-1)}, f_{\o(i)}, f_{\o(i+1)}) = \\ \|\pi(f_{\o(i+1)}, f_{\o(i)}) - \pi(f_{\o(i)}, f_{\o(i-1)})\|_2^2$$

to every triplet of consecutive output frames. In practice, we use a truncated norm, clamping it at 200.

4) User Constraints: To allow the user to specify a beginning or ending video, as well as to include or exclude parts of other videos, we introduce a cost term $C_u(f_{\o(i)})$. Frames that must be excluded are given a penalty term $C_u(f_{i,j}) = \theta$ for some large value (we use 10^6). For frames that the user wants to include in the final hyper-lapse, we set $C_u(f_{i,j}) = -\theta$ for its first and last frames, and in addition raise the minimum duration value Υ along that part. This attracts the minimum path on the graph to include this part of the video.

In addition to these optimization terms, we impose a hard constraint of a minimum duration $\Upsilon = 10$ frames, for each sub-sequence of consecutive frames taken from each source video V_i. This constraint is expressed as $\min_i(D(V_i)) \geq \Upsilon$, and ensures that there is a transition window when switching between videos to allow gradual color transition and camera motion smoothing during post-processing (Section V).

Multiple source hyper-lapse creation is then formulated as a minimization problem over the set of frames that define F from the input videos:

$$\begin{aligned} F = \underset{\{f_{\o(1)},\ldots,f_{\o(T)}\}}{\arg\min} \sum_t & C_s(f_{\o(t)}, f_{\o(t+1)}) + C_u(f_{\o(t)}) \\ & + \lambda_v C_v(f_{\o(t)}, f_{\o(t+1)}, v_{\o(t)}) + \lambda_a C_a(f_{\o(t-1)}, f_{\o(t)}, f_{\o(t+1)}), \\ & s.t. \min_i(D(V_i)) \geq \Upsilon, \end{aligned} \tag{4}$$

where λ_v and λ_a are parameters balancing the individual cost terms. This formula sums all costs related to each selected frame $f_{\o_t} \in F$. We empirically set $\lambda_v = 100$, $\lambda_a = 40$ throughout all our examples. In practice, the results are insensitive to these parameter values, but they can be adjusted according to the user's preference in balancing visual stability and the speed-up constraint.

D. Shortest Path Optimization

We convert the global optimization of Equation 4 to finding a shortest path in our transition graph. First, we define a cost function C to measure the cost of a two-step transition from node p to node q, and from node q to node r, according to the above criteria. The definition is the weighted sum of the individual cost terms:

$$C = C_s(p, q) + C_u(p) + \lambda_v C_v(p, q, v) + \lambda_a C_a(p, q, r), \tag{5}$$

where v is the locally determined frame skip rate for frame p, and λ_v and λ_a are parameters as in Equation 4. Using this cost function, the shortest path on the transition graph from N_{vs} to N_{ve} corresponds to the optimal solution with the smallest total cost.

We solve the shortest path problem using *memorization search* [37], which is a top-down recursive search approach in the spirit of dynamic programming. From the virtual start node N_{vs}, we recursively expand and visit connected nodes under the minimum duration constraint, calculate the cost of following specific edges, and memorize the optimal sum of sub-path cost plus the corresponding successor node in a look-up table. We continue the search until reaching the virtual end node N_{ve}. The optimal path is created by iteratively collecting the optimal successor nodes starting at N_{vs} using the look-up table. The corresponding frames are assembled as the hyper-lapse video. Pseudo-code of this search process is provided in the supplementary materials. The time complexity of the algorithm is $O(Ln^2R^2\Upsilon)$, where L is the total number of input video frames, n is the number of input videos, R is the maximum sampling temporal distance and Υ is the minimum duration. In practice, the shortest path computation time taken in all our examples was less than 5 minutes for typically tens of thousands of nodes and edges in the hyper-lapse transition graph.

V. Rendering

The constructed hyper-lapse video consists of intra-source and inter-source transitions between frames. Geometric differences between the frames in inter-source transitions may require large camera motions. To solve this, we average the homography between the transition frames over several previous frames to smooth out large camera motions.

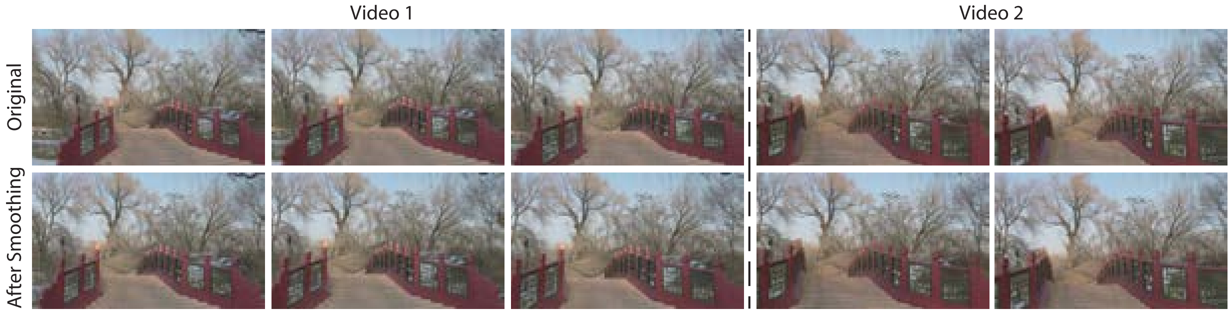


Fig. 7. Inter-source motion smoothing. Consecutive frames in the hyper-lapse sequence ‘Old Summer Palace 2’ taken from two videos are shown. Without motion smoothing, a sudden horizontal jump is seen between the transition frames. After applying motion smoothing, the frames in Video 1 gradually change towards Video 2 creating a smoother transition (bottom).

Stabilization is then performed on the output frames to eliminate the remaining jitter. In addition, as the input videos may be captured at different times using different devices, the color, lighting and movements of dynamic objects may differ in the two videos. This can lead to reduced visual quality as a result of a sudden appearance change at inter-source transitions. To solve this, we apply appearance smoothing while rendering the output.

A. Motion Smoothing

Although hyper-lapse creation optimization seeks an as-smooth-as-possible frame sequence, there may still be large inter-video camera motions between the output frames. We deal with such large inter-video motions by first distributing the motion to several frames, when there is a change in the video source, and then using standard video stabilization on all output frames.

To eliminate inter-video motions, we estimate the camera motion $T_{ø(i)}$ between two consecutive cross-video transition frames $f_{ø(i)}$ and $f_{ø(i+1)}$, and instead of applying it point-wise in the transition, we distribute the motion evenly into the ω previous frames: $\langle f_{ø(i-\omega+1)}, \ldots, f_{ø(i)} \rangle$. The original large camera motion is then broken into smaller ones that are applied from $f_{ø(i-\omega+1)}$ to $f_{ø(i+1)}$.

Specifically, assume that $T_{ø(i)}$ is the homography defined by:

$$T_{ø(i)} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

For each output frame $f_{ø(i-\omega+t)}$ $(1 \leq t \leq \omega)$, the motion smoothed frame $f'_{ø(i-\omega+t)} = T'_t(f_{ø(i-\omega+t)})$, where T'_t is the t-th motion-smoothing transformation, defined by the matrix:

$$T'_t = \begin{pmatrix} s^t_{11} & d^t_{12} & d^t_{13} \\ d^t_{21} & s^t_{22} & d^t_{23} \\ d^t_{31} & d^t_{32} & 1 \end{pmatrix}$$

As h_{11} and h_{22} in $T_{ø(i)}$ are parameters related to x and y coordinate scaling respectively, we distribute their value in ω steps using the power function:

$$s^t_{pp} = pow(h_{pp}, \frac{t}{\omega+1}) \quad 1 \leq p \leq 2$$

For all other parameters, we evenly divide them into ω parts and accumulate their values from 0.

$$d^t_{pq} = \frac{h_{pq} \cdot t}{\omega+1} \quad 1 \leq p, q \leq 3, p \neq q.$$

Figure 7 shows an inter-video smoothing example carried over several frames.

To further eliminate the remaining jitter, we follow previous methods of single video hyper-lapse video creation [1] [25]. We perform video stabilization on the motion-smoothed sequence, to further remove low frequency motion noise present in each video source. We use the single path camera trajectory smoothing method of Liu et al. [30], under an area constraint of at least 80%, and crop the central rectangle view to get the final result.

B. Appearance Smoothing

The resulting hyper-lapse depends on plausible appearance transitions for both the colors and the motions of objects in the scene. Possible appearance smoothing solutions include direct color adjustment of each frame or multiple-frame fusion based on a dissolve transition. We perform color adjustment by considering different constraints, including original spatial gradient preservation, temporal illumination smoothing and corresponding appearance preservation. Nevertheless, it is difficult to avoid visual inconsistency in extreme cases where objects suddenly appear in the scene. Complicated solutions, such as point-cloud and full 3D warping based dissolving [14], can potentially be used for appearance smoothing, but will reduce the computational efficiency of the system. Therefore, we smoothly adjust the appearance by applying simple Gaussian smoothing temporally (with a radius of 3 frames), which our results shows works well (Figure 8).

VI. Results

All the videos in our results were taken by mobile devices (either a GoPro Hero 4 or a smart phone), or taken from collections of videos from the Internet. The capture devices were handheld or fixed on a helmet. The collection of videos were captured either outdoors or indoors while touring. The videos include static scenes, moving objects and even crowds, and are of length 5–15 minutes, with 1280×720 frame size. We processed the videos on a computer with an

TABLE I

STATISTICS AND COMPUTATION TIME (IN MINUTES) FOR THE EXAMPLES USED IN OUR EXPERIMENTS. ALL INPUT VIDEOS IN EACH EXAMPLE WERE USED IN THE OUTPUT HYPER-LAPSE RESULT. CREATION TIME IS THE AVERAGE TIME FOR ALL RESULT VIDEOS PER EXAMPLE

Video Properties				Pre-Processing (min.)			Creation (min.)	
Video Name	#Videos	#Frames	Speed-Up	Filtering	Alignment	Graph Const.	Opt.	Render
Walk and Bike	2	16344	8×	3.4	32.0	22.0	0.7	2.2
‖	‖	‖	12×	‖	‖	‖	0.6	2.0
Bike	3	13913	8×	4.3	28.0	25.0	2.1	2.3
Gordon Square	10	41006	8×	17.0	22.0	37.0	2.8	3.2
Old Summer Palace 1	3	20817	10×	5.0	42.3	41.0	2.2	0.9
Old Summer Palace 2	3	31034	10×	8.5	40.0	51.0	2.8	2.5
‖	‖	‖	20×	‖	‖	‖	0.6	1.9
Old Summer Palace 3	5	11926	10×	10.7	17.3	24.0	2.3	1.5
Garden Walk 1	3	23156	10×	10.0	57.0	34.5	2.0	1.8
Garden Walk 2	4	27646	10×	8.3	41.0	25.0	3.3	2.5
Short Walk 1	5	22939	10×	12.0	39.0	8.0	0.5	2.5
Short Walk 2	3	12697	10×	4.5	7.0	12.0	0.4	2.0
Short Walk 3	4	17546	10×	17.0	33.0	35.5	4.7	4.0
Supermarket	3	10682	10×	7.6	19.2	20.5	2.2	2.0
Zoo	7	38508	10×	21.0	22.5	33.8	1.5	2.6
Museum	9	61447	10×	28.0	45.0	24.5	3.3	1.0
Park Tour 1	7	40550	15×	11.5	33.2	20.0	2.3	2.0
Park Tour 2	10	53379	15×	26.0	41.0	22.3	2.7	1.5

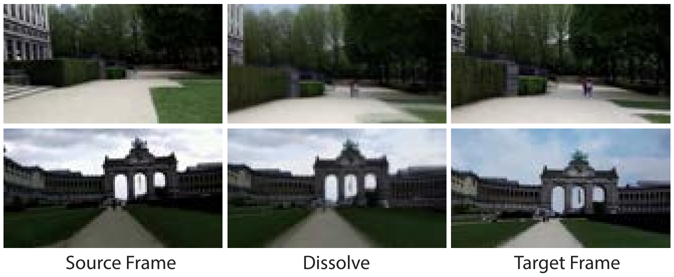


Fig. 8. Appearance smoothing processing for smooth content transition. Direct color adjustment works poorly for sudden appearance of objects (see the walkers in the first row) or obvious content change (the sky in the second row).

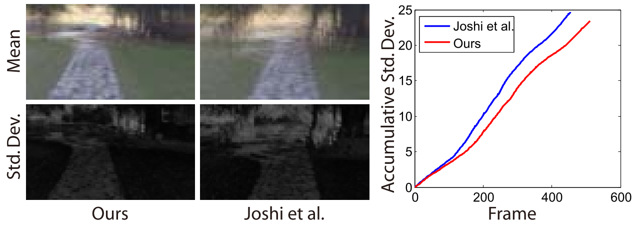
Intel Core i7-4790K CPU and 32GB RAM. Pre-processing to build the transition graph took less than two hours. Note that once the graph has been constructed, it can be reused to create multiple results. For all examples, hyper-lapse sequence optimization took less than 5 minutes, as did online rendering. Detailed timings are listed in Table I.

All parameter values were fixed as reported earlier in the text, for all experiments. The parameters that affect the visual results most are the velocity and acceleration weights λ_v, λ_a, and the minimum duration in the optimization objective. Our default parameters worked well in all experiments, but could be adjusted to match particular user preferences: for example, increasing λ_v provides greater speed consistency, while decreasing λ_v and λ_a provides more stable visual content.

One basic goal of our multi-video approach is to provide more stable hyper-lapse results for static scenes compared to using single-video hyper-lapse creation based on frame selection. A further benefit is that the proposed method provides effective hyper-lapse routes synthesis for virtual camera routes. We present several scenarios and interesting applications to demonstrate its power.

First, given several input videos of a static scene with a single overlapping route, in which each video includes drastic, joggling, camera motions, we generate more stable hyper-lapse video results than single-video frame selection. We illustrate this in Figure 9 by showing an example of the mean and standard deviation of five consecutive hyper-lapse frames and the accumulated average standard deviation of our approach compared to [1]'s method. As can be seen, our hyper-lapse result is more stable due to the optimized hyper-lapse transitions avoiding unstable camera sub-paths in the original input video. The cost of these improved results is longer run time mostly due to the need to handle multiple videos. Our computations take 105 minutes, out of which 101 minutes are pre-processing. This is compared to 4 minutes for Joshi et al.'s method on a single video. Nevertheless, the pre-processing stage must only be performed once, and the user can generate alternative results from the set of input videos in time comparable to Joshi et al.'s method. Note that if large camera parallax exists, hyper-lapse creation based on the 3D scene reconstruction technique of [24] would generate more stable results than a 2D methods like ours. However, reconstruction of multiple videos is far more time consuming than sampling and can sometimes also fail.

Our multiple video hyper-lapse creation can generate videos with consistent camera velocity given videos captured during different activities such as walking and bicycling.



(a) (b)

Fig. 9. Comparisons with *[Joshi et al. 2015]*'s method, using a challenging video example with drastic camera motions. Frames from 'Garden Walk 1' with three videos with fully overlapping routes. (a) Example for the mean and standard deviation of five consecutive hyper-lapse frames. (b) The accumulated standard deviation divided by the number of video frames for the two videos.

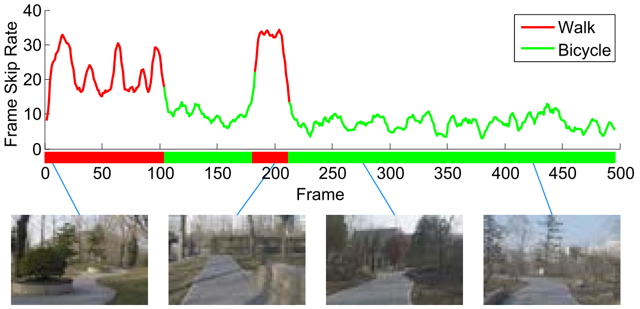


Fig. 10. Combining walking and bicycling videos at 12× the average velocity of the two videos results in higher skip rate for the walking sequences than the bicycling. As can be seen in the supplemental video, the output hyper-lapse maintains a consistent camera velocity.

Figure 10 shows the sampling rate over time of a hyper-lapse combining the content from walking and bicycling videos, where the average speed up was set to 12× (Section IV-B). As can be seen, the local skip rate can fluctuate, but to maintain a consistent rate it is lower for the bicycle parts than for the walking parts.

Our approach can find valid transitions between videos and generate plausible hyper-lapse results with dynamic content, even with crowds in the videos. Both the outdoor 'Zoo' and indoor 'Museum' examples contain many tourists. Nevertheless, we still succeeded in finding optimal hyper-lapse routes with only moderate dynamic object inconsistency during hyper-lapse transitions between videos (Figure 1 and supplemental video).

A. User Interface

As explained in Section IV-A, we allow users to specify sub-routes which must be included or excluded from the output hyper-lapse. There are two typical reasons for this: to select a sub-route from several non-overlapping routes, and to include or exclude specific content from videos. We provide an easy-to-use interface that portrays a time-line slider for every video in the collection using sampled thumbnails of its frames, a preview window, a command panel and optionally a route map panel (Figure 11). This map is created manually, but our system will immediately map the overlaps in the routes to the detected frame alignments in the videos. Note that video alignment does not require any user input assistance and the route map is only used for interaction and hyper-lapse preview. The user can label frames either on the map panel or on the video timeline slider. When the cursor is on the map panel, the selected frame is snapped to the nearest frame according to the distance between cursor and curve points. Users can easily select sections of any video by clicking the left or right button, indicating inclusion or exclusion at start position and end position; the selected sub-videos are marked as green or red, respectively. Once hyper-lapse optimization is finished, the user can play the hyper-lapse via the interface where the hyper-lapse result is displayed in the preview window and the corresponding frame is highlighted on the route map and the timeline slider. Examples of interaction are provided in the supplemental video.

For input videos with complex routes, default hyper-lapse creation seeks an optimal hyper-lapse route with the most stable camera trajectory that is also short. To create longer and more complex outputs, the user can impose sub-route constraints using the user interface, as desired. In Figure 12, three different hyper-lapse routes with the same start and end points were synthesized from ten input videos by using different user specification on the sub-routes. Please refer to the supplemental videos for the interaction process.

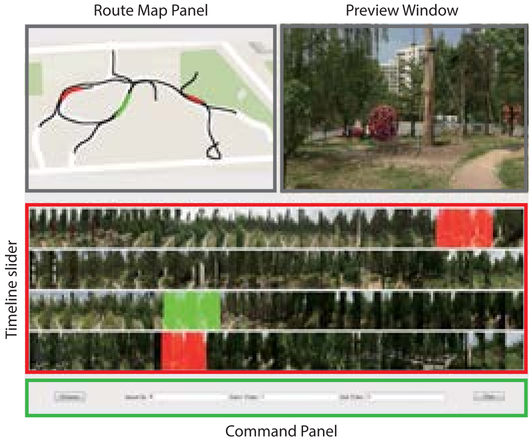


Fig. 11. User interface for adding constraints to video routes and playing hyper-lapse results. It consists of a preview window, a route map panel, a timeline slider and a command panel.

Using the interface, users can easily remove distractions from the resulting video. For example, if some distraction, such as an undesired person, comes into view while capturing a video, the user only needs to re-capture a new local video clip. She can then constrain the video to exclude the distraction (or to include the new video). Our algorithm will automatically find the path in the transition graph that follows these constraints and remove the distraction. Figure 13 shows an example of removing distracting people from a hyper-lapse.

We also have an interactive hyper-lapse navigation interface, which allows users to interactively select the next sub-route to explore while watching a hyper-lapse video. Each time a junction is encountered, the user can interactively preview frames on the next available sub-routes by exploring a user input abstract route map. Once the user selects the corresponding route, navigation with hyper-lapse continues (see Figure 14). To support this application, we perform an additional junction extraction step after discovering local alignments. We group neighboring positions along non-overlapping routes between video pairs. We then accumulate the length of the grouped non-overlapping positions by counting the corresponding frames. We mark every continuous non-overlapping route whose length exceeds 300 frames (corresponding to ∼ 10 seconds) as a *non-overlapping section* and record the start and end points of the section as junctions. The user can synthesize a hyper-lapse sub-route between any two neighboring junctions and play the clips in navigation order.

Note that to be faithful to the original video content, input videos in our experiments are not allowed to be temporally reversed. However videos could be straightforwardly reversed before processing for special user preference and visual effects.

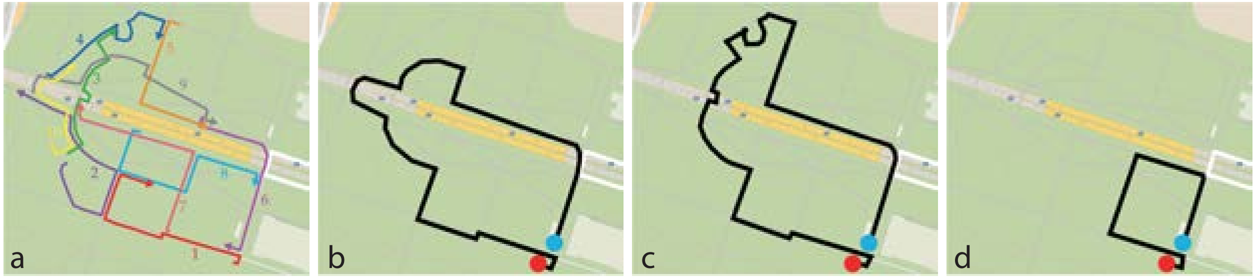


Fig. 12. Examples of virtual complex routes synthesis. Based on the input routes (a) and user-specified constraints (see supplemental video), three different hyper-lapse routes (b)–(d) are created with the same start point and end point.

TABLE II

COMPARISON TO OTHER GRAPH MODELS USED IN MULTIPLE VIDEO EDITING. FROM EACH COLUMN, *ENCODING COSTS* INDICATES WHETHER THERE ARE COSTS OR PENALTIES EMBEDDED IN THE GRAPH; *SKIPPING TRANSITIONS* INDICATES WHETHER THERE ARE EDGES CONNECTING NON-TEMPORAL NEIGHBORING NODES; *NAVIGATION* INDICATES WHETHER INTERACTIVE USER NAVIGATION IS SUPPORTED, AND *ROUTE OPTIMIZATION* INDICATES WHETHER AN OPTIMAL, VISUALLY APPEALING CAMERA ROUTE CAN BE OBTAINED FROM THE GRAPH MODEL

	Structure		*Function*	
	Encoding Costs	***Skipping Transitions***	***Navigation***	***Route Optimization***
Videoscapes [Tompkin et al. 2012]	X	X	√	X
Trellis Graph [Arev at al. 2014]	√	X	X	√
EgoSampling [Halperin at el. 2017]	√	√	X	√
Our Hyper-lapse Transition Graph	√	√	√	√

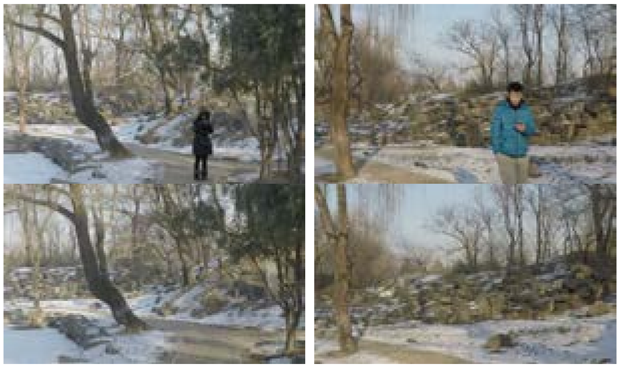


Fig. 13. Example of removing distractions. Top: the original captured video contains distracting people. Bottom: using the corresponding sub-route from other videos, the undesired people are removed from the hyper-lapse result.

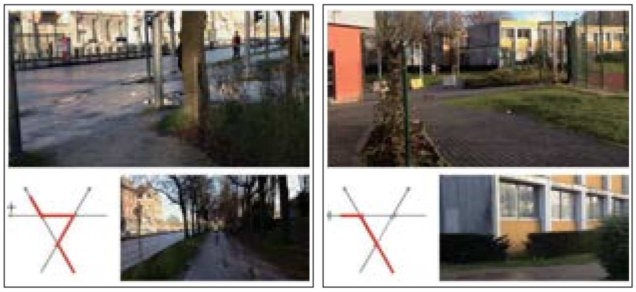


Fig. 14. Interactive hyper-lapse navigation. The user interacts with an abstract route map (bottom left) and sees thumbnails of the route in a preview window (bottom right). Following selections, the hyper-lapse is shown in the viewing window (top). We show examples of two hyper-lapse paths explored using this interface.

VII. DISCUSSION AND CONCLUSION

A. Comparison to Graph Models

Graph models have been widely used in image and video editing. Our graph methodology is similar to other formulations. However, each problem has its own specific requirements, and in our case the graph is carefully designed to solve the multi-video hyper-lapse problem. The structure of this graph models both intra-video and inter-video transitions between frames in using edges. By optimizing an objective function on the graph, our method can generate a hyper-lapse sequence. We compare graph models from recent video processing works [13], [14], [27] with respect to some key structural and functional properties in Table II. Compared to Videoscapes [14], our transition graph can be used to compute stable hyper-lapse results because of the encoded penalty for route optimization. In contrast, Videoscapes only supports interactive navigation. Compared to the trellis graph structure [13] where edges only connect temporally neighboring frame nodes, our graph contains possible transitions between frames with a temporal distance we called *skipping transitions*, which do not necessarily form a dense trellis structure. Compared to the graph structure in EgoSampling [27], our hyper-lapse transition graph can encode interactively specified user constraints, and thus can support interactive hyper-lapse creation or navigation to specified target positions on the routes.

B. Conclusion

We have presented a method to create smooth continuous hyper-lapse videos from multiple spatially-overlapping input videos. Our method is based on sampling and assembly of frames, for multiple videos with different sub-routes, camera velocities and dynamic objects. We mine the potential connections between video pairs and encode the potential hyper-lapse transitions in a transition graph for all videos in the collection. Hyper-lapse frame assembly is formulated as a minimization problem solved by a shortest path search on the transition graph. For challenging input videos with severe camera motions, our approach provides visually better results than an approach based on a single video hyper-lapse sampling, as more options to choose frames are available. Furthermore, our approach also allows numerous novel applications such as virtual hyper-lapse route synthesis, user-guided hyper-lapse creation, and interactive hyper-lapse navigation.

Our approach has its limitations: videos that share a common view or location must be available. However, as video capture these days is extremely simple, we believe that collections of similar videos will rapidly grow as have collections of images. Given a set of similar videos, our frame alignment and camera motion estimation procedures are based on feature matching rather than full scene reconstruction. Videos with large camera parallax or moving distractions may fail to be aligned properly.

In future, we wish to combine our hyper-lapse creation with high-level semantic cues. Semantic cues such as video content understanding and human activity recognition can assist in pre-filterin hyper-lapse transition edges as well as providing visual aesthetic criteria for hyper-lapse creation. In this way, human-centric video processing is addressed, which we believe will be an important research direction.

Acknowledgment

The authors would like to thank all the reviewers.

References

[1] N. Joshi, W. Kienzle, M. Toelle, M. Uyttendaele, and M. F. Cohen, "Real-time hyperlapse creation via optimal frame selection," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 63-1–63-9, 2015.

[2] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3D," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 835–846, 2006.

[3] R. Martin-Brualla, D. Gallup, and S. M. Seitz, "Time-lapse mining from Internet photos," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 62-1–62-8, 2015.

[4] G. Kim, L. Sigal, and E. P. Xing, "Joint summarization of large-scale collections of Web images and videos for storyline reconstruction," in *Proc. Int. Conf. CVPR*, Jun. 2014, pp. 4225–4232.

[5] A. Lippman, "Movie-maps: An application of the optical videodisc to computer graphics," *ACM SIGGRAPH Comput. Graph.*, vol. 14, no. 3, pp. 32–42, Jul. 1980.

[6] S. Pongnumkul, J. Wang, and M. Cohen, "Creating map-based storyboards for browsing tour videos," in *Proc. 21st Annu. ACM Symp. User Interface Softw. Technol.*, 2008, pp. 13–22.

[7] C.-X. Ma, Y. Guo, and H.-A. Wang, "VideoMap: An interactive and scalable visualization for exploring video content," *Comput. Vis. Media*, vol. 2, no. 3, pp. 291–304, Sep. 2016.

[8] L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 600–608, Aug. 2004.

[9] L. Ballan, G. J. Brostow, J. Puwein, and M. Pollefeys, "Unstructured video-based rendering: Interactive exploration of casually captured videos," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 87-1–87-10, 2010.

[10] J. Rüegg, O. Wang, A. Smolic, and M. Gross, "DuctTake: Spatiotemporal video compositing," *Comput. Graph. Forum*, vol. 32, no. 2pt1, pp. 51–61, 2013.

[11] Y. Wang, D. Tao, X. Li, M. Song, J. Bu, and P. Tan, "Video tonal stabilization via color states smoothing," *IEEE Trans. Image Process.*, vol. 23, no. 11, pp. 4838–4849, Nov. 2014.

[12] T. Chen, J.-Y. Zhu, A. Shamir, and S.-M. Hu, "Motion-aware gradient domain video composition," *IEEE Trans. Image Process.*, vol. 22, no. 7, pp. 2532–2544, Jul. 2013.

[13] I. Arev, H. S. Park, Y. Sheikh, J. Hodgins, and A. Shamir, "Automatic editing of footage from multiple social cameras," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 81-1–81-11, 2014.

[14] J. Tompkin, K. I. Kim, J. Kautz, and C. Theobalt, "Videoscapes: Exploring sparse, unstructured video collections," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 68-1–68-12, 2012.

[15] J. Tompkin, F. Pece, R. Shah, S. Izadi, J. Kautz, and C. Theobalt, "Video collections in panoramic contexts," in *Proc. ACM Symp. UIST*, 2013, pp. 131–140.

[16] O. Wang, C. Schroers, H. Zimmer, M. Gross, and A. Sorkine-Hornung, "VideoSnapping: Interactive synchronization of multiple videos," *ACM Trans. Graph.*, vol. 33, no. 4, p. 77, 2014.

[17] C. Malleson *et al.*, "FaceDirector: Continuous control of facial performance in video," in *Proc. ICCV*, Dec. 2015, pp. 3979–3987.

[18] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proc. Int. Conf. CVPR*, vol. 1. Jun./Jul. 2004, pp. 645–652.

[19] H.-I. Chen, Y.-L. Chen, W.-T. Lee, F. Wang, and B.-Y. Chen, "Integrating dashcam views through inter-video mapping," in *Proc. ICCV*, Dec. 2015, pp. 3110–3118.

[20] E. P. Bennett and L. McMillan, "Computational time-lapse video," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 102-1–102-6, 2007.

[21] Y. Shih, S. Paris, F. Durand, and W. T. Freeman, "Data-driven hallucination of different times of day from a single outdoor photo," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 200-1–200-11, Nov. 2013.

[22] S.-P. Lu, G. Dauphin, G. Lafruit, and A. Munteanu, "Color retargeting: Interactive time-varying color image composition from time-lapse sequences," *Comput. Vis. Media*, vol. 1, no. 4, pp. 321–330, 2015.

[23] K. Hasegawa and H. Saito, "Synthesis of a stroboscopic image from a hand-held camera sequence for a sports analysis," *Comput. Vis. Media*, vol. 2, no. 3, pp. 277–289, Sep. 2016.

[24] J. Kopf, M. F. Cohen, and R. Szeliski, "First-person hyper-lapse videos," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 78-1–78-10, 2014.

[25] Y. Poleg, T. Halperin, C. Arora, and S. Peleg, "Egosampling: Fast-forward and stereo for egocentric videos," in *Proc. Int. Conf. CVPR*, Jun. 2015, pp. 4768–4776.

[26] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa, "Video textures," in *Proc. ACM SIGGRAPH*, 2000, pp. 489–498.

[27] T. Halperin, Y. Poleg, C. Arora, and S. Peleg, "EgoSampling: Wide view hyperlapse from egocentric videos," *IEEE Trans. Circuits Syst. Video Technol.*, to be published, doi: 10.1109/TCSVT.2017.2651051.

[28] Y. Matsushita, E. Ofek, W. Ge, X. Tang, and H.-Y. Shum, "Full-frame video stabilization with motion inpainting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 7, pp. 1150–1163, Jul. 2006.

[29] M. Grundmann, V. Kwatra, and I. Essa, "Auto-directed video stabilization with robust L1 optimal camera paths," in *Proc. Int. Conf. CVPR*, Jun. 2011, pp. 225–232.

[30] S. Liu, L. Yuan, P. Tan, and J. Sun, "Bundled camera paths for video stabilization," *ACM Trans. Graph.*, vol. 32, no. 4, p. 78, 2013.

[31] F. Liu, M. Gleicher, H. Jin, and A. Agarwala, "Content-preserving warps for 3D video stabilization," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 44-1–44-9, 2009.

[32] F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala, "Subspace video stabilization," *ACM Trans. Graph.*, vol. 30, no. 1, pp. 4-1–4-10, 2011.

[33] J. Kopf, "360° video stabilization," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 195-1–195-9, Nov. 2016.

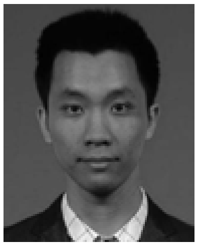
[34] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. Int. Conf. ICCV*, vol. 2. Sep. 1999, pp. 1150–1157.

[35] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Proc. ECCV Workshop*, vol. 1, 2004, pp. 1–16.

[36] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

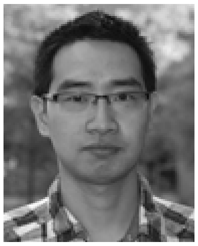
[37] D. Michie, "Memo functions and machine learning," *Nature*, vol. 218, no. 5136, pp. 19–22, 1968.

Miao Wang received the B.S. degree from Xidian University in 2011, and the Ph.D. degree from Tsinghua University in 2016. He is currently a Postdoctoral Research Fellow with Tsinghua University. His research interests include computer graphics, image/video editing and deep learning in graphics. He is a member of ACM and CCF.



Jun-Bang Liang received the B.S. degree from Tsinghua University, in 2016. He is currently pursuing the Ph.D. degree with Prof. Ming C. Lin in computer graphics at the University of North Carolina at Chapel Hill, USA. He is interested in physically based simulation and computer vision.

Song-Hai Zhang received the Ph.D. degree from Tsinghua University in 2007. He is currently an Associate Professor with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include image and video processing, geometric computing.



Shao-Ping Lu received the Ph.D. degree in computer science from Tsinghua University, China, in 2012. Since 2012, he has been a Postdoctoral Researcher with Vrije Universiteit Brussels, Belgium, where he is currently a Senior Researcher and leading a new team focusing on 3D video and visualization research. His primary research areas are image and video processing and editing.

Ariel Shamir received the B.Sc. and M.Sc. *cum laude* degrees in math and computer science and the Ph.D. degree in computer science from the Hebrew University, Jerusalem, in 2000. He is currently a Professor with The Interdisciplinary Center, Ef Arazi school of Computer Science, Israel. He spent two years as a Post-Doctoral Fellow with the computational visualization center, University of Texas in Austin. He was a Visiting Scientist with Mitsubishi Electric Research Laboratory, Cambridge, MA, USA, in 2006 and with Disney Research in 2014. His research interests include geometric modeling, computer graphics, fabrication, visualization, and machine learning. He is a member of the ACM SIGGRAPH, IEEE Computer and Eurographics societies.

Shi-Min Hu received the Ph.D. degree from Zhejiang University, in 1996. He is currently a Professor with the Department of Computer Science and Technology, Tsinghua University, Beijing. His research interests include digital geometry processing, video processing, computer animation and computer-aided geometric design. He is the Editor-in-Chief of Computational Visual Media, and on the Editorial Board of several journals, including the IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, *Computer Aided Design* and *Computer and Graphics.*