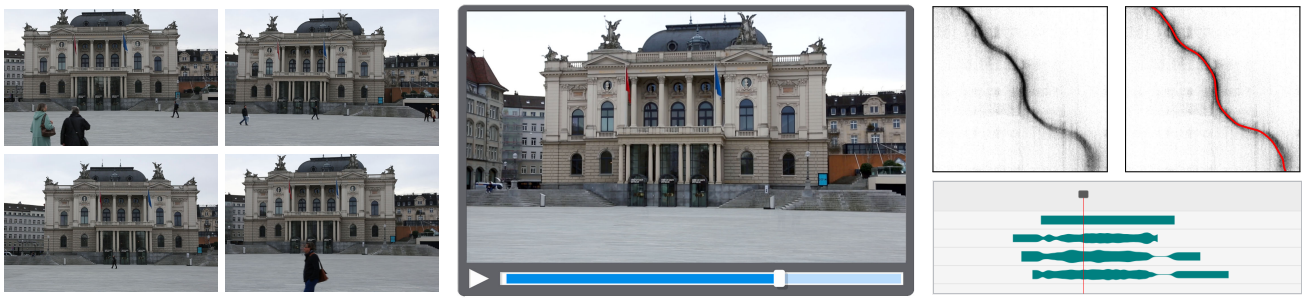
# VideoSnapping: Interactive Synchronization of Multiple Videos

Oliver Wang*, Christopher Schroers*, Henning Zimmer*, Markus Gross*†, Alexander Sorkine-Hornung*
*Disney Research Zurich, †ETH Zurich



(a) (b) (c)

**Figure 1:** *Four videos are taken of the same scene at different times from roughly similar camera trajectories* ***(a)****. Spatiotemporal alignment greatly simplifies complex video editing tasks. Here, we removed the people from all video clips with a simple median filter* ***(b)****. Our method efficiently finds a nonlinear temporal warping, synchronizing multiple videos by computing paths in a cost matrix* ***(c)*** *above. This is visualized by the squeezing of clips in the timeline* ***(c)*** *below, resulting in an intuitive, interactive video alignment tool.*

## Abstract

Aligning video is a fundamental task in computer graphics and vision, required for a wide range of applications. We present an *interactive* method for computing optimal nonlinear temporal video alignments of an arbitrary number of videos. We first derive a robust approximation of alignment quality between pairs of clips, computed as a weighted histogram of feature matches. We then find optimal temporal mappings (constituting frame correspondences) using a graph-based approach that allows for very efficient evaluation with artist constraints. This enables an enhancement to the "snapping" interface in video editing tools, where videos in a timeline are now able snap to one another when dragged by an artist based on their *content*, rather than simply start-and-end times. The pairwise snapping is then generalized to multiple clips, achieving a globally optimal temporal synchronization that automatically arranges a series of clips filmed at different times into a single consistent time frame. When followed by a simple spatial registration, we achieve high quality spatiotemporal video alignments at a fraction of the computational complexity compared to previous methods. Assisted temporal alignment is a degree of freedom that has been largely unexplored, but is an important task in video editing. Our approach is simple to implement, highly efficient, and very robust to differences in video content, allowing for *interactive* exploration of the temporal alignment space for multiple real world HD videos.

**CR Categories:** I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.4.8 [Image Processing and Computer Vision ]: Scene Analysis—Time-varying imagery;

**Keywords:** Video Editing, Alignment, Synchronization

**Links:** DL PDF WEB

## 1 Introduction

Most tasks related to handling or manipulating multiple video clips involve some form of alignment. Exemplary applications include compositing multiple takes in movie production and visual effects [Rüegg et al. 2013], video mosaicking and stitching [Agarwala et al. 2005], color grading, action recognition and video retrieval [Jiang et al. 2007], summarization [Ngo et al. 2005], and HDR video [Kang et al. 2003].

The majority of existing solutions focus on accurate *spatial* alignment, i.e., computing sparse correspondences or dense mappings between individual frames of multiple video clips. The necessary *temporal* alignment required for successful spatial matching is generally assumed to be available, e.g., as a pre-specified offset computed using global timecodes like LTC, genlock, or audio [Shrestha et al. 2007; Bryan et al. 2012].

However, in many of the above mentioned scenarios, explicit synchronization signals may not be available. In practice, a temporal ordering and alignment of multiple clips often has to be established manually using video editing software, where clips are typically represented as bars that can be dragged and positioned relative to each other on a global timeline. The assistance that such tools provide the artist is typically limited to snapping video clips at start- and endpoints, or at pre-specified markers.

The basic motivation for our work is to enable *content-based* snapping to any location where the video clips share similar content. But instead of simple temporal offsets or linear time scaling, we are interested in finding general nonlinear time warps that globally optimize the matching quality between multiple input video clips. Intuitively, we would like to temporally "squeeze-and-stretch" the input clips so that at any point in time their visual similarity is maximized. Figure 1c shows an example of such a temporal deformation, visualized as a literal *stretching* of the height of the clips in the timeline.

As potentially many such alignments exist, and only a certain amount of temporal warping is visually acceptable, our method allows the artist to interactively and intuitively explore the space of alignments and define temporal constraints with a simplicity that is comparable to the basic snapping known from existing video editing tools.

For each pair of input clips our method computes a cost matrix, whose values represent a robust metric of spatial alignment quality. A path through this cost matrix gives us a set of corresponding frames in the two respective videos with implicit temporal continuity. We compute alignments as constrained, normalized shortest paths using a modified version of Dijkstra's algorithm that allows clips to be aligned with only partial temporal and spatial overlap. Desired temporal mappings can then be interactively updated given an artist's constraints, e.g., when the artist drags a clip to a preferred temporal position, or when changing distortion limits. We additionally discuss possible parameterizations of these mappings that can be used to control warping effects. For the alignment of multiple video clips, we describe a graph-based solution based on an efficient minimum spanning tree that places all video clips aligned on a single global time axis, and is updated in real-time according to the artist's interaction.

Besides interactive video alignment, we demonstrate novel applications such as the automatic arranging of a set of clips of the same or similar events. We also show how our method can be used for synchronizing multiple cameras, aligning motion capture data, and improving the results of existing spatial alignment algorithms.

## 2 Prior Work

When addressing video alignment, a natural approach is to extend the broad range of literature on *image* alignment. The latter is a very well-studied problem in the computer vision community in the context of motion estimation (optical flow) [Baker et al. 2011] and stereo correspondence estimation [Scharstein and Szeliski 2002]. However, when aligning frames between videos, further challenges arise. For one, if we align videos taken from different camera trajectories and at different times, the scene content and appearance may vary greatly due to the change in perspective, moving scene content and lighting differences. Thus frames to align can have a arbitrarily different appearance, violating the basic assumptions of many image alignment methods. Some techniques have been proposed that address these issues directly for images [Baker and Matthews 2004]. More recently, promising results have been shown using robust pixel descriptors [Liu et al. 2011] or content adaptive weighting functions [Yücer et al. 2012] but these methods have focused on recognition and editing tasks where some degree of inaccuracy in the alignment is still tolerable. Even more important, is the additional dimension of *time* in video. Slight alignment errors that might be invisible in still images are generally much more noticeable in video due to our attention to temporal changes.

Basic temporal alignment, i.e., finding the closest frames between two or more videos, has been studied in the field of human behavior analysis where one needs to align similar human actions to study similarities and differences. Most methods are based on dynamic time warping (DTW), with extensions to improve efficiency and to accommodate various sensor modalities like skeleton tracking or accelerometer data [Zhou and la Torre 2009; Zhou and la Torre 2012]. Rao et al. [2003] perform DTW-based video alignment using estimated 3D point trajectories. As this relies on an accurate tracking of image features over time and cameras with a fixed (rigid) setup, application to real world filming scenarios is an issue. This work also considers only temporal frame-to-frame alignment, leaving out spatial alignment.

In addition to temporal alignment, most applications however also require computing spatial alignment. Spatial alignment establishes pixel correspondences between temporally aligned frames, a basic requirement for any compositing operation, e.g. blending. One of the first methods that tackled the problem of spatiotemporal alignment was proposed by Caspi and Irani [2002], which computes an affine temporal warping and a fixed spatial homography per video. Affine temporal warps are, however, not suitable for general alignment scenarios with unconstrained camera motion and scene content, and fixed homographies cannot compensate for depth variation or relative change in camera placement. The same limitations apply to the later works [Pádua et al. 2010; Ukrainitz and Irani 2006] which also assume fixed or jointly moving cameras. Notably, the alignment of Pádua et al. [2010] can handle more than two videos, however, it does so for linear temporal mappings only. Li and Chellappa [2010] allow for nonlinear time warps but require fixed time spans and full video overlap.

Targeting accurate spatiotemporal alignment for *general scenarios*, Sand and Teller [2004] proposed a method that can handle varying scene content and arbitrary camera configurations. While their main contribution is a very robust spatial alignment, they also propose an adaptive search algorithm that finds the frames that best align temporally. They achieve impressive results, but their approach is computationally intense, requiring several minutes per second of video, and only considers pairs of videos. Diego et al. [2011; 2013] and Evangelidis and Bauckhage [2013] also propose methods for spatiotemporal alignment that leverage similar frame-to-frame cost matrices. The former performs an expensive MAP inference, or relies on GPS information for robustness, while the latter computes quad descriptors and finds a multiscale spatiotemporal mapping by dynamic programming. Both approaches also assume that one sequence is contained entirely within the other, and are limited in the types of temporal and spatial warps allowed; as such they are not suitable for our main application scenarios.

In contrast to all above approaches, our method enables flexible, artist directed nonlinear temporal warping of multiple videos at *interactive* rates. All of these differences are necessary criteria for practical usage in a video editing framework where instant feedback is critical.

## 3 Synchronizing Two Videos

In this section, we first propose a method for temporally synchronizing two videos $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, before generalizing the concepts to multiple videos (Section 4). We begin by defining the requirements that have to be fulfilled.

**Requirements** In order to be useful in a video editing context, it is essential to support *partial overlap* of video clips (see Figure 1c), as solutions that require one video to be contained entirely in the other are not practical. Additionally the method should be symmetric, i.e., aligning $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ or $\boldsymbol{v}_2$ and $\boldsymbol{v}_1$ should result in the same set of frame-correspondences.

Another essential aspect for any video editing tool is interactivity. In our case this means that the artist should be able to specify a pair of frames that should be in temporal alignment. This is achieved by dragging clips in a timeline-style interface similar to existing video editing tools, s.t. frames that should be in correspondence line up under the current time indicator bar (vertical red line in Figure 1c). From an artist's perspective, the most helpful and innovative functionality required is that clips automatically *snap* into an optimal temporal alignment within a vicinity of a specified alignment constraint.

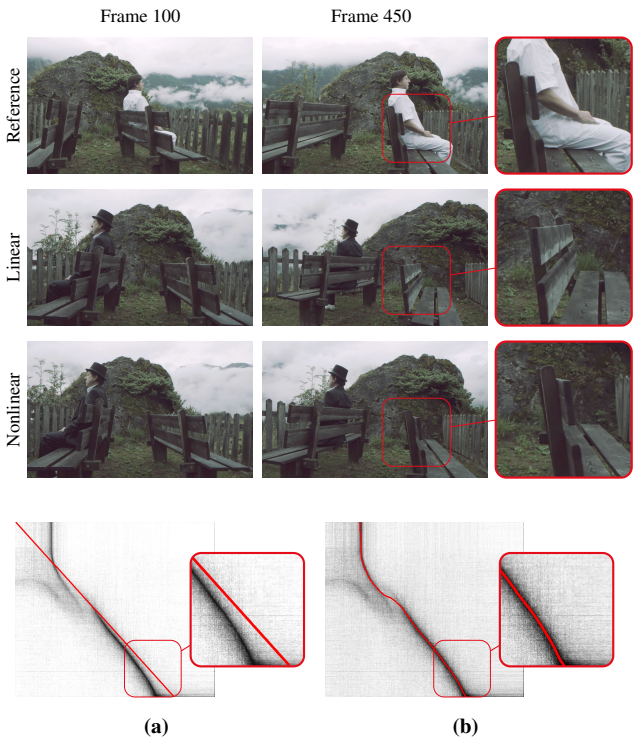



**Figure 2:** *A linear path (red line in the cost matrix shown in **(a)**) is unable to model differences in camera motion. By aligning the beginning of a sequence (left column, above), large parallax effects occur towards the end (right column, above). Nonlinear snapping matches the full range of motion **(b)**, better synchronizing the camera positions at each frame. This can be seen in the visibly reduced parallax differences of the bench (inset), which then makes the spatial alignment a much easier task.*

Additionally, the degree of temporal warping that is allowed may change depending on the context. Therefore, the artist should be able to control the limits of temporal distortion, preventing or allowing effects such as backwards motion, pausing, or playing videos at high speed.

**Notation** We use subscripts to denote the video index, while arguments correspond to frame indices, e.g., $\boldsymbol{v}_i(j)$ denotes the $j$th frame of video $i$ where $j \in [1...N_i]$ and $N_i$ is the number of frames of the $i$th video sequence. Our goal is to find a simple curve (i.e., without self intersections) $\boldsymbol{p} : \mathbb{R} \to \mathbb{R}^2$, where $\boldsymbol{p}(t) = (p_1(t), p_2(t))$ associates a global time $t$ with two corresponding frames $\boldsymbol{v}_1(p_1(t))$ and $\boldsymbol{v}_2(p_2(t))$.

As we allow both object and camera motion to be different, determining a constant temporal offset between two videos, $\boldsymbol{p}(t) = (t, t + b)$, or linear transformation, $\boldsymbol{p}(t) = (t, a \cdot t + b)$, will not allow for a good temporal synchronization in many cases. Instead, we find a constrained nonlinear mapping, allowing for variation in the timing of both scene content and camera trajectory. Figure 2 shows the difference between a linear and nonlinear mapping.

We now describe how we compute the mapping $\boldsymbol{p}$ as a continuous temporal warp that maximizes the expected spatial alignment, while adhering to constraints. First, a cost matrix is constructed that approximates alignment scores for pairs of frames, and then this cost matrix is used to compute the unknown mapping $\boldsymbol{p}$.

### 3.1 Cost Matrix

As the basis of our method, we need a robust estimate of the alignment quality for all pairs of frames. We observe that two frames are more likely to be "alignable" if they contain a large number of similar features. Using this concept, we compute a histogram from feature matches from $\boldsymbol{v}_1$ to $\boldsymbol{v}_2$ and vice versa. Each feature match contributes to one bin of the histogram based on the frames that those features come from. Subsequently, we transform the histogram into a cost matrix $\mathbf{C} \in \mathbb{R}^{N_1 \times N_2}$, in which each entry $c_{jk}$ specifies a cost for aligning a pair of frames $(\boldsymbol{v}_1(j), \boldsymbol{v}_2(k))$.

More formally, we consider a video and the set of all features found in all of its frames. We denote the $l$th feature of the $i$th video by $\boldsymbol{f}_i(l)$. It contains the image space coordinate $\boldsymbol{x}_i(l) \in \mathbb{R}^2$, the frame number $z_i(l) \in [1 \dots N_i]$ and a descriptor $\mathbf{d}_i(l) \in \mathbb{R}^d$. We generally use SIFT feature descriptors [Lowe 1999], where $d = 128$. However, the choice of descriptor can be made based on the specific application. In Section 5, we discuss other potential frame descriptors.

We then build a set of matches $M \subset \mathbb{N}^2$ by finding the nearest neighbor of each feature $\boldsymbol{f}_1(l)$ in the set of all features from the second video and vice versa, based on the $L_1$ distance of their descriptors. A match $(a, b) \in M$ indicates that $\boldsymbol{f}_1(a)$ matches to $\boldsymbol{f}_2(b)$. For a pair of frames $\boldsymbol{v}_1(j)$ and $\boldsymbol{v}_2(k)$, we write $M_{jk}$ as the set of all matches $(a, b) \in M$ such that $z_1(a) = j$ and $z_2(b) = k$, that is, where a feature in frame $j$ matches to a feature in frame $k$.

Besides $L_1$ descriptor difference, we also consider image space distance and prefer matches that are close in image space (which generally lead to more reliable spatial alignment). The affinity histogram $\boldsymbol{H}$ has entries

$$h_{jk} = \sum_{(a,b) \in M_{jk}} \rho_d \left( \|\mathbf{d}_1(a) - \mathbf{d}_2(b)\| \right) \rho_s \left( \|\mathbf{x}_1(a) - \mathbf{x}_2(b)\| \right). \quad (1)$$

We use a common decaying weighting function that gives a higher weight to closer matches

$$\rho_w(x) = \exp(-w \cdot x), \quad (2)$$

where $\rho_d$ and $\rho_s$ control the rate of decay for descriptor and spatial weights respectively. For most examples, we use $d = 1$ and $s = 2$, however for harder examples such as Figure 9, we use $s = 5$ to account for a higher number of incorrect matches.

Finally, we transform $\boldsymbol{H}$ to obtain the cost matrix $\boldsymbol{C}$ in which low values indicate frames that are likely to have a *good* match. The entries of the cost matrix $\boldsymbol{C}$ are given by

$$c_{jk} = \left(1 - \frac{h_{jk}}{h_{max}}\right)^{\alpha}, \quad (3)$$

where $h_{max}$ is the value of the maximum bin, and $\alpha$ is a scaling parameter that penalizes high cost bins. We use $\alpha = 4$ in almost all examples, the exception being for noisier matrices (e.g. Figures 9, 16) we used $\alpha = 10$.

We note that the frame number $z_i(l)$ is *not* a part of the descriptor, meaning our method has no prior bias for matching features in one frame to a specific frame in the other video. As a result, we are able to align vastly different camera timings, such as videos played in reverse, or with highly nonlinear differences.

As it does not directly rely on the accuracy of individual matches, but rather the overall statistical distribution of a large set of feature matches over the entire video, the cost matrix is highly robust

to incorrect correspondences. In principle, additional pruning such as RANSAC using epipolar geometry or homography relationships could be used to increase basic matching reliability. However, they do so at the cost of increased computation time and fragility; common effects such as rolling shutter and motion break assumptions of the above methods. Our approach has proved to work robustly even in very challenging scenarios (e.g., Figures 8, 9), where such standard feature refinement techniques fail.

### 3.2 Computing Mappings

So far we have described mappings in the continuous sense. We now first explain how to find a discrete mapping $\boldsymbol{p} : \mathbb{N} \rightarrow \mathbb{N}^2$ in our cost matrix. Subsequently, we discuss how we turn this into a continuous mapping and how we can use different curve parameterizations to embed the alignment into a global time frame.

We refer to the discrete representation of a mapping $\boldsymbol{p}$ as a "path" through the cost matrix, and consider a graph based solution to find optimal paths. Furthermore, we associate a cost

$$\phi(\boldsymbol{p}) = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{C}(\boldsymbol{p}(t)) \tag{4}$$

with each path, that is the average of all entries in the cost matrix that it crosses. Here $T$ denotes the number of steps in the path and we use the notation $\boldsymbol{C}(j,k) = c_{jk}$ for better readability. We chose the average cost to not bias our preferences to shorter paths.

Given a cost matrix $\boldsymbol{C}$, we can trivially compute the best path assuming a constant temporal offset between videos by quickly searching the space of feasible offsets for the minimal costs according to Equation 4. In a similar fashion, one can solve for the best linear transformation (slope and offset). The slope can either be computed using the ratio of frame rates of the cameras or it can be treated as a additional unknown leading to a small 2-D search problem. These concepts alone would provide a useful extension to video editing tools, and are sufficient for snapping to optimal linear paths. However, as mentioned before, linear mappings are often not enough to obtain accurate temporal alignments. Instead, we compute arbitrary nonlinear paths.

If we interpret the cost matrix as a directed graph, paths become a set of connected edges. We identify a node by its position $(j,k)$ in the cost matrix and an edge as an ordered pair of nodes. In the simplest case where backwards motion is not allowed, each node is connected to its three neighbors, i.e., has the edges $((j,k),(j+1,k))$, $((j,k),(j,k+1))$, and $((j,k),(j+1,k+1))$. The weight of each edge is the value of the cost matrix at the node that the edge points to. To allow backwards motion we add full 8-way connectivity.

Minimum distance paths can be found using Dijkstra's algorithm [Dijkstra 1959], which we describe briefly as a recap. The basic algorithm finds the shortest distance $d$ from a source node $s$ to *all* other nodes, maintaining a set of nodes that are being processed called the "working set". Until all nodes are marked as finished, the node in the current working set with the lowest $d$ is picked for processing. All of its neighbors are then checked and added to the working set if their new distance is shorter than their current best known $d$. Once all neighbors of a node have been checked, that node is set to finished and is removed from the working set.

By computing a path from the start of both clips $(1,1)$ to the end of both $(N_1, N_2)$ we can solve basic alignment problems. However, a number of our requirements would not be met. In the next few paragraphs we discuss the details of our path computation, and how they allow us to meet all the requirements.

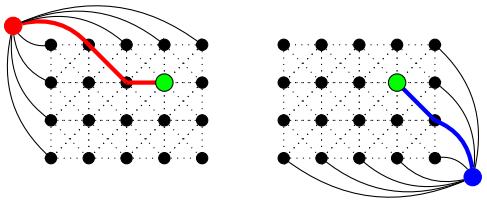


**Figure 3:** *Computing a path in the cost matrix, given a user constraint (green). Source nodes (red and blue) are connected to the first frames (top row and left column) and last frames (bottom row and right column) of each video. Min-path trees are then computed from each source node. Tracing the user constraint back to source nodes (red and blue lines) yields the final path.*

**Partial Overlap** As discussed above, when working in a video-editing context it is essential to support nonlinear alignment of clips with partial temporal overlap. Partial overlap implies that a path can start at any frame in either video, and end at any frame in either video. We construct a modified graph, adding an additional super-source node $S$ (the red circle in Figure 3). This source node is connected to a set of start nodes $\hat{S}$ that correspond to the first frames of $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, i.e.,

$$\hat{S} = \{(1,1),\ldots,(1,N_2)\} \cup \{(1,1),\ldots,(N_1,1)\}. \tag{5}$$

The shortest path is then computed from $S$ to all end nodes

$$\hat{E} = \{(N_1,1),\ldots,(N_1,N_2)\} \cup \{(1,N_2),\ldots,(N_1,N_2)\}. \tag{6}$$

This gives us a set of alignments from *any* start frame to *any* end frame from which we must select one. If we simply choose the path with the least distance, this approach will inherently prefer paths that contain *fewer* edges. Instead, we evaluate the cost of each feasible path according to Equation 4 (i.e., normalizing by the number of nodes in the path), and select the one with the lowest cost. We note that trivial solutions are possible, e.g., when a corner of $\boldsymbol{C}$ has a low cost, very short paths can be selected. To prevent this, we set a minimum path length $T_{min} = 10$, ensuring a reasonable number of samples within the cost matrix.

**Interactivity** To support interactive modification of paths, we introduce the notion of an *alignment constraint* provided by an artist. An alignment constraint $\boldsymbol{a} = (j,k)$ specifies that a pair of frames $\boldsymbol{v}_1(j), \boldsymbol{v}_2(k)$ are in correspondence, i.e., that the path must pass through $(j,k)$. Using the described graph-based approach, we could compute the shortest path from $S$ to $\boldsymbol{a}$, and then another shortest path from $\boldsymbol{a}$ to $\hat{E}$ to achieve this.

However, this would be too slow to support real-time interaction as the artist drags the clip in the timeline. We use an approach inspired by Panorama Weaving [Summa et al. 2012], which allows users to drag control points while visualizing optimal seams in a panorama. This approach involves computing two shortest-path trees, one from $S$ to $\hat{E}$ and an inverted tree from a super-sink node $E$ to $\hat{S}$. Min-cost paths that pass through $\boldsymbol{a}$ can then be computed by simply tracing the paths from $\boldsymbol{a}$ to $S$ and from $\boldsymbol{a}$ to $E$, illustrated in Figure 3. Finally, to achieve "snapping", we can simply evaluate a window around $\boldsymbol{a}$, and pick the minimum cost path that is below some threshold.

Allowing arbitrary paths could create unrealistic mappings that distort the videos when played. To allow the artist to have full control over the amount of temporal distortion that is acceptable, we also introduce *warping constraints*.

On a geometric level, we can interpret the relative warping of $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ as the *slope* of the path. The artist provides two bounds,

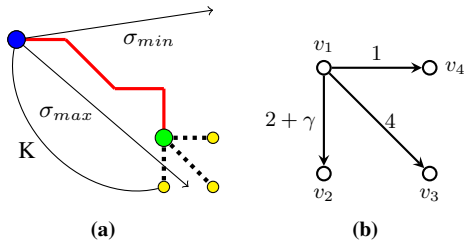



**Figure 4:** *Warping constraints. When processing the green node, neighbors (yellow) are considered for addition to the working set* ***(a)****. If the slope of the line between these candidate locations and the point $K$ steps up the tree (blue) is outside of the constraints, a large constant $\gamma$ is added to the edge weight* ***(b)****.*

$\sigma_{min}$ and $\sigma_{max}$ that specify the minimum and maximum slope allowed, respectively. As our path is discrete, we compute an estimate of the slope by using a point a fixed $K$ steps back up the shortest path tree. For all examples, we use $K = 10$.

These constraints are used in a modified version of Dijkstra's algorithm. Before adding any node to the working set we check whether the slope of the path up to that node is within $\sigma_{min}$ and $\sigma_{max}$. If these bounds are violated, we increase the edge weight by a high constant factor $\gamma$. This has the effect of enforcing that the edge appears in our final path *only* if no other paths that do not violate the slope constraint are possible. This is demonstrated in Figure 4a; the blue node is used to compute the slope at points under consideration (yellow). In this case, the bottom node is outside $\sigma_{min}$, and is therefor assigned a high penalty weight (Figure 4b). The order that the nodes will be visited in the working set of the modified Dijkstra's algorithm is: $v_4, v_3, v_2$.

For all results we used $\gamma = 1000$. The $\sigma_{max}$ and $\sigma_{min}$ are dataset dependent, but usually if no special restrictions exist we can use very generous bounds, e.g. $\sigma_{max} = 10$ and $\sigma_{min} = \frac{1}{10}$.

**Sub-frame Rendering** So far we have discussed the discrete setting of computing an optimal path. In order to achieve a continuous temporal mapping with values at sub-frame positions, we filter our discrete path using Laplacian smoothing and linearly interpolate and extrapolate. After smoothing, our path has fractional indices into the cost matrix, which we evaluate by interpolating from neighboring points in $\mathbf{C}$, and eventually when rendering by frame-interpolation using optical flow [Zimmer et al. 2011]. As computing optical flow is relatively slow, we perform frame interpolation only during final rendering, and for the purpose of visualization we show the nearest-neighbor frames in the GUI.

**Global Time Frame** A path uniquely determines a set of frames in correspondence in two videos. However, the path does *not* define the global time frame, i.e., the rate at which we walk along the path, and at which each video is played. There are several valid options for parameterizing $\boldsymbol{p}$, depending on the preferences of the artist or the requirements of the application. Sometimes it can be desirable for the temporal warping to be distributed among videos, so that no single video exhibits as extreme temporal changes. This can be achieved by using the arc length parameterization (See Figure 5a), which amounts to walking along the path in equal size steps, and rendering corresponding frames from each video. However, many times it can also be advantageous to leave one video unchanged as a *reference* video, and adapt only the speed of the second one, i.e., choosing a parameterization that maintains equal steps in one video (Figure 5b).

We provide a high-level summary of the required tasks for aligning two videos in Algorithm 1.

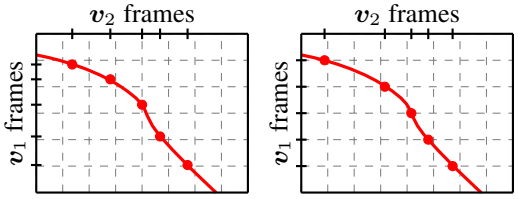



**Figure 5:** *Effect of path parameterization. Using an arc-length parameterization* ***(a)*** *causes steps along the curve at evenly spaced intervals (red circles), distributing temporal warping in both videos (black ticks on axes). Using a reference video for the parameterization* ***(b)*** *results in a constant playback speed of one video ($\boldsymbol{v}_1$ in this case), and all the distortion in the other ($\boldsymbol{v}_2$).*

**Algorithm 1** Computing an optimal temporal path between two videos $\boldsymbol{v}_1, \boldsymbol{v}_2$ and rendering corresponding frames.

$\boldsymbol{F}_1 \leftarrow$ features($\boldsymbol{v}_1$)
$\boldsymbol{F}_2 \leftarrow$ features($\boldsymbol{v}_2$)
$M \leftarrow$ match($\boldsymbol{F}_1, \boldsymbol{F}_2$)

$\boldsymbol{H} \leftarrow$ histogram($\boldsymbol{F}_1, \boldsymbol{F}_2, M$) ▷ See Eq. 1
$\boldsymbol{C} \leftarrow$ costMatrix($\boldsymbol{H}$) ▷ See Eq. 3

$u \leftarrow$ minPathTreeConstrained($\boldsymbol{C}$,$S$,$\hat{E}$) ▷ See Sec. 3.2
$v \leftarrow$ minPathTreeConstrained($\boldsymbol{C}$,$E$,$\hat{S}$)

**while** true **do** ▷ interactive loop
  $(j, k) \leftarrow$ getUserConstraint()
  $p_u \leftarrow$ trace$(u, j, k)$
  $p_v \leftarrow$ trace$(v, j, k)$
  $\boldsymbol{p} \leftarrow$ connect$(p_u, p_v)$
**end while**

$\boldsymbol{p} \leftarrow$ reparameterize($\boldsymbol{p}$)
**for all** $t \in [t_{start}, t_{end}]$ **do**
  render $\boldsymbol{v}_1(\boldsymbol{p}_1(t))$,$\boldsymbol{v}_2(\boldsymbol{p}_2(t))$
**end for**

## 4 Synchronizing Multiple Sequences

In video editing software, working with multiple clips in a single timeline is common. A significant contribution of our work is providing a practical extension to support joint synchronization for an arbitrary number of videos.

In theory it would be straightforward to extend the previously described approach to multiple videos by increasing the dimensionality of the cost matrix and finding a mapping $\boldsymbol{p} : \mathbb{R} \rightarrow \mathbb{R}^D$ , where $D$ is the number of videos. However, in practice this quickly becomes intractable due to the size of the $D$-dimensional cost matrix $(N_1 \cdot N_2 \cdot \ldots \cdot N_D)$. Let us conservatively assume that $N_i \approx 300$, which would cover 10 seconds of video at 30 frames per second. With just five videos, this would result in $300^5$ entries, or roughly 8.8 TiB of memory when using 32-bit floats, which clearly exceeds current hardware capabilities.

Furthermore, it is often undesirable to match all sequences simultaneously, as pairs of video clips can have little or no overlap. Alignment estimates for such pairs would degrade the overall alignment quality. Therefore, rather than matching "all-to-all" videos, we instead use only the best *pairwise* alignments to obtain a global relationship. This strategy relies only on the most accurate alignments, and also reduces the problem to a size that is easily tractable.

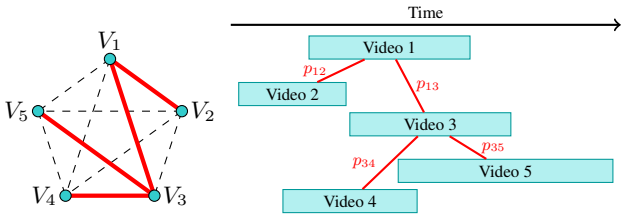



**Figure 6:** *A minimum spanning tree representing the optimal set of pairwise alignments (left), and the same tree displayed as videos on a timeline (right).*

We describe our approach in two stages. First, we define a way to measure the associated cost of how well we expect a pair of video clips to align. Second, we use these costs to find an optimal set of pairwise temporal alignments that best describes the global relationship of all clips.

**Computing Matching Costs Between Videos** The basic idea is to use the cost of the best path $\phi(\boldsymbol{p}_{ij})$ as a measure of how well videos $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ can be aligned. Using the method from Section 3 we compute cost matrices and mappings for each pair of videos. By design, our method for temporally aligning two videos is symmetric, i.e. $\phi(\boldsymbol{p}_{ij}) = \phi(\boldsymbol{p}_{ji})$. Therefore, only $\binom{D}{2}$ different cost matrices have to be considered.

**Finding Optimal Alignment Pairs** Given all matching scores between videos, the task is to find the optimal set of pairwise matches such that every video is aligned to *at least* one other. To do this, we construct a undirected graph $G$ where each node $V_i$ corresponds to a video, and each edge $E_{ij}$ defines a pairwise alignment between videos $V_i$ and $V_j$. The weight of the edge $E_{ij}$ is then equal to the cost of aligning the two videos, i.e., $\phi(\boldsymbol{p}_{ij})$. A spanning tree of $G$ gives a set videos, each aligned to at least one other, and the *minimum spanning tree* (MST) of $G$ corresponds to the one with the globally minimal error.

We use Prims algorithm for MST computation [Prim 1957]. This is efficient to compute, as our graph has only $D$ nodes and $\binom{D}{2}$ edges, where $D$ is usually quite small. The MST computation can be updated in real-time as the artist changes constraints (and therefore changes $\phi(\boldsymbol{p}_{ij})$). Figure 6 shows an example of $G$, the MST, and corresponding tree of pairwise relationships of video clips.

**Global Time Axis** Now that we have determined the optimal pairwise relationships, we need to know which frame from each of the $D$ videos to display at a given point in time $t$. In other words, we need a mapping $\tilde{\boldsymbol{p}} : \mathbb{R} \rightarrow \mathbb{R}^D$. We define this mapping by means of the individual pairwise mappings $\boldsymbol{p}_{ij}$ and the MST. To find all corresponding frames, we simply traverse the MST recursively evaluating frame positions. We note that for this to work, each path must describe the graph of a 1D function, i.e. can be expressed as $\boldsymbol{p}_{ij}(t) = (t, p_{ij}(t))$, where $p_{ij} : \mathbb{R} \rightarrow \mathbb{R}$. We can ensure this by choosing $\sigma_{min} > 0$ and using appropriate graph connectivity that prevents either video from playing backwards. This is the one restriction over the pairwise case that we introduce when processing multiple videos. For the example shown in Figure 5, the global mapping would be defined as: $\tilde{\boldsymbol{p}}(t) = (t, p_{12}(t), p_{13}(t), p_{34}(p_{13}(t)), p_{35}(p_{13}(t)))$.

**Art Directability** We have included simple and intuitive opportunities for interaction to give the artist detailed control over the multiple clip alignment. First of all, the tree obtained from the MST computation comes without the notion of a root, so we select one and use it to define the reference time, but the artist can easily change the root video simply by double clicking on it. Additionally, MST connections can be visualized in the timeline (Figure 16), and we allow the artist to manually constrain certain pairs of videos to be connected by control-clicking both of them. This is useful for example if the artist would like to synchronize a specific event occurring in one pair of videos. We achieve this interaction technically by setting the weights of these edges in $G$ to 0, and recomputing the MST as usual. When one video is dragged in the timeline, all of the children of that video will move as well, keeping their optimal synchronizations fixed. This gives an intuitive way for the artist to modify the temporal placement of one video, and see the result on the entire set immediately. Please refer to our supplemental video for an example of such an interaction.

**Figure 7:** *Cyclical motion (orbiting around a static object) results in numerous good paths, visible in the cost matrix (right). Our interface allows an artist to easily snap between them.*

**Figure 9:** *A temporal mapping for a difficult case where the image appearance differs greatly.*

## 5 Applications

In the following section we demonstrate a number of different applications using our algorithm, ranging from basic video editing interface enhancements to more complex compositing tasks. Please refer to the supplemental video for complete examples of these applications.

**VideoSnapping** Content aware video snapping provides a useful and intuitive extension to modern video editing software. Allowing the artist to specify alignments by dragging clips in a timeline has numerous advantages, besides to being an established metaphor. For one, it allows her to select from numerous potentially well aligned paths within a pair of videos (Figure 7). Additionally, it provides a useful way to specify *which* clips should be aligned at all, which is especially useful in cluttered timelines with many clips present. By inferring the artist's intents from this simple gesture, our method can be used with minimal hassle.

**Robust Temporal Alignment** Because we do not rely on the accuracy of individual feature correspondences, but rather on accumulated statistical differences over entire video sequences, our cost matrix robustly captures frame similarities. Figure 8 shows an example where a car drives down a road with a hood-mounted camera. In this case, the small regions above and through the windscreen provide all the alignment cues available. Still, a clear path can be derived from the associated cost matrix.

**Figure 8:** *Even with only a small portion of the video containing shared content, we can still find a clear path through the cost matrix, synchronizing the car's location.*

**Figure 10:** *Our solution computes an alignment globally, and can find an optimal path even with a region in the middle of the videos where no similar content exists.*

In Figure 9, we temporally synchronized two camera pans taken during different times of the day. In this case, feature correspondences were insufficient to find any kind of reliable spatial alignment between frames, but the correct path in the cost matrix could still be accurately detected. Because we compute a *global* path within the cost matrix, we can also handle large areas where *no* actual alignment is possible. In Figure 10, the middle half of the video is taken from two different routes with no common content. However, our method is able to smoothly interpolate in the unknown region, and reconnect as the content starts to be shared again.

**Multi-Camera Synchronization** Camera arrays are useful for acquiring high resolution, field-of-view, or parallax effects that would not be possible with a single camera. However, synchronizing large numbers of video cameras can be a challenging task, and often synchronization problems result in unusable data. We used our method to synchronize an array of 15 cameras entirely based on visual cues. Our method automatically determines an optimal tree of video pairs where content is the most similar (e.g., cameras that overlap). Figure 11 shows the detected temporal relationship of all input videos, and a successfully reconstructed frame of a video panorama. In addition to finding small offsets between videos, we also successfully detected one camera that completely failed.

**Figure 11:** *By finding relative temporal offsets of clips from a 15 camera array (left), it becomes possible to construct a video panorama without ghosting on moving objects (right).*

**Figure 12:** *Pixel accurate spatiotemporal alignment allows us to compute an HDR video by per-frame exposure fusion, despite significant differences in camera motion and illumination conditions.*

**Spatiotemporal Alignment** Accurate spatiotemporal alignment is an extremely important open problem in video processing and visual effects pipelines. High quality alignments make otherwise extremely complex tasks much easier, such as video segmentation, matting, blending and compositing. A major benefit of our temporal snapping is that it can greatly reduce the difficulty of finding good *spatial* alignments. In fact, we show that by applying a simple, out-of-the-box method for spatial alignment *after* temporal synchronization, we can achieve similar or better quality when compared to state-of-the-art spatiotemporal approaches.

Our spatial alignment method is simple. We first compute per-frame homography matrices, and then filter each entry independently using a temporal median filter with a width of 15 frames so as to improve temporal stability. Finally, to correct for small parallax effects, we warp the images using optical flow [Zimmer et al. 2011]. Figure 14 shows the importance of nonlinear temporal synchronization before spatial alignment. With only the best linear alignment, parallax effects are too large to correct.

We compare our method to two prior spatiotemporal alignment approaches. Figure 13 and the video show difference maps computed for similar scenes. We can see that our method performs equal

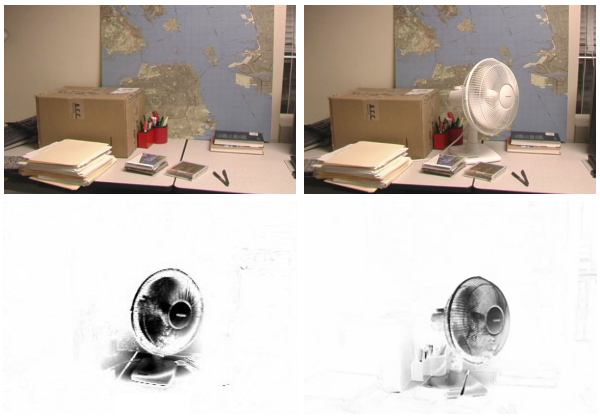


**Figure 13:** *Comparison of spatiotemporal alignment to Sand and Teller [2004] (whiter is better). (Note: The input videos made available differed slightly from publicly posted results)*

**(a)** *linear* **(b)** *nonlinear*

**Figure 14:** *Inverted difference images (whiter is better) after temporal and spatial alignment. With the best possible linear temporal mapping **(a)**, larger alignment errors are visible, most noticeably in the bench which cannot be correctly aligned due to parallax, and in the boundary, where overlapping content is not available. After nonlinear temporal alignment, much better spatial alignment is possible **(b)**. Images have been darkened for visibility.*

to, or better than these approaches. Significantly, our method uses *simple* methods for spatial alignment, not relying on complex feature refinement [Sand and Teller 2004], or MAP inference and GPS data [Diego et al. 2011].

**Additional Modalities of Data** While we propose using SIFT features to describe frames, these can be easily substituted with other kind of frame descriptors. In Figure 15b, it is possible to synchronize multiple actions performed by the same person using only SIFT features. However, when considering an action performed by different people (Figure 15d), there is not enough visual similarity to correctly determine frame compatibility. As a result, we can see that the corresponding cost matrix is largely noise, and no reasonable paths can be found.

We augment the use of SIFT descriptors with skeleton data automatically tracked from a Kinect sensor [Bloom et al. 2012]. At each frame, there are 20 joint positions, given in world space coordinates. We subtract the centroid from each frame, and concatenate the relative joint positions into a single per-frame descriptor $\mathbf{d}_i(j) \in \mathbb{R}^d$, where $d = 60$ (20 joints and one 3D position per joint). We then use the exact same pipeline as described in Section 3 to build the cost matrix and compute paths. As we have many fewer features per frame (one rather than thousands), we use

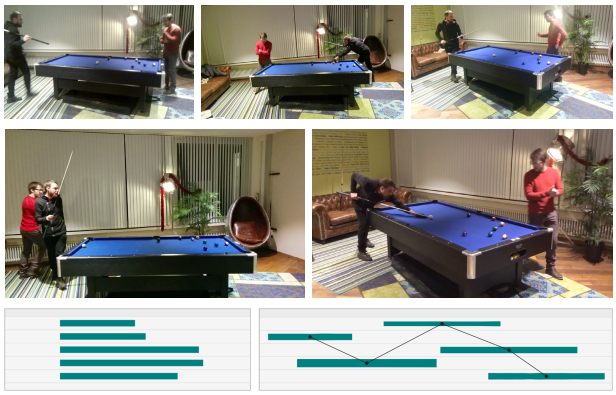


**Figure 16:** *Five frames from different input devices capturing the same event at different times (above). The timeline before and after alignment (below). Optimally determined minimum spanning tree edges are shown in black. Please see the supplemental material for a complete video.*

8-nearest neighbors when building $\boldsymbol{C}$ instead of just finding the best match. By adding these features, we can correctly synchronize the actions of the different people.

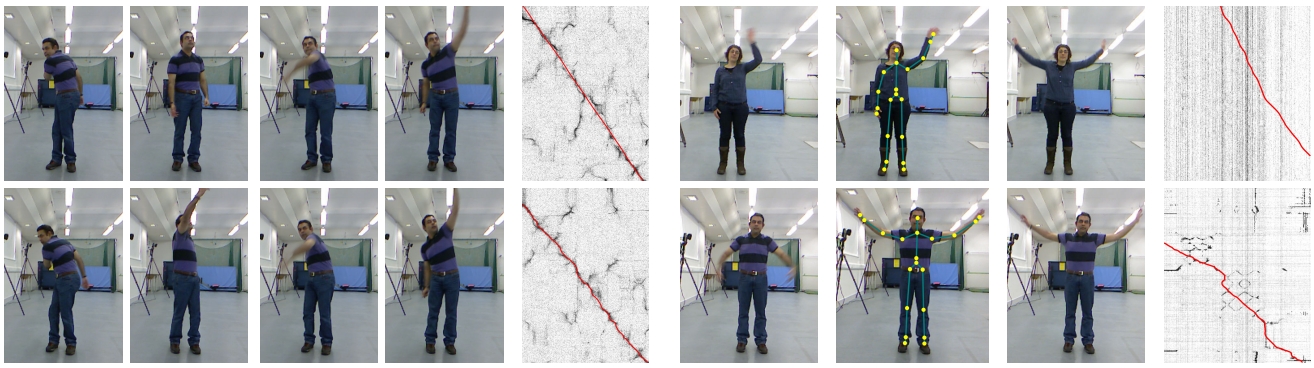
**Crowdsourced Video** Being able to synchronize multiple videos enables novel applications, such as automatically arranging a set of uncalibrated videos in the same global time frame based only on similarities of visual data. Popular events today are recorded by a large number of cameras, each filming a distinct temporal subset of the event. In this case, it is possible that no single camera records the whole sequence, and there may be little to no overlap between all pairs of videos.

In Figure 16, we show an example where we take a series of unordered videos of a pool game, filmed from different positions on different devices, each with its own framerate, resolution, and visual appearance. Our approach can place all clips in a global time axis by matching optimal pairs of videos, after which we can create a single video that cuts between available cameras (please see the supplemental material for this result).

## 6 Implementation

Our prototype interface was written in C++ using QT, and consists of parallelized, but otherwise unoptimized code. We use the VLFeat open source library to perform SIFT feature computation and feature matching [Vedaldi and Fulkerson 2008], using a KD-tree with approximate nearest neighbor queries for efficiency.

**Timing** The entire preprocessing stage, which consists of computing SIFT features, building and querying the KD-tree, and computing cost matrices and shortest path trees takes on average **295ms** per $1440 \times 810$ frame on a modern desktop computer (Intel 3.20Ghz i7-3930K, 6 Cores). Most of this running time (265ms) is spent computing SIFT features; if needed, much faster preprocessing times could be achieved by replacing SIFT with a different feature detector and descriptor. Additionally, we can handle the multiple video case with minimal overhead, as SIFT feature computation grows only linearly with the number of videos. Even though the number of cost matrices grows quadratically with the number of videos, they are efficient to compute and easy to store in memory, or cache for later interaction sessions, requiring roughly 3.4 MiB for *all* matrices considering the example size discussed in the beginning of Section 4. The cost of evaluating a path given a frame



(a) (b) (c) (d) (e) (f) (g)

**Figure 15:** *A series of actions is repeated by a person. Each column shows matched frames from a pair of videos. Using the best linear mapping, it is not possible to align all actions* ***(a)****. With nonlinear warping* ***(b)****, the entire sequence can be aligned (cost matrices shown in* ***(c)****). However, with two different people, there is not enough similarity using visual features alone, resulting in poorly aligned frames* ***(d)****, and a noisy cost matrix* ***(g)****, above. By including skeleton data from a Kinect sensor* ***(e)****, the videos can be correctly aligned* ***(f)****. The correct path and multi-modal cost matrix is visible in* ***(g)****, below.*

constraint (during the interaction stage) takes **3ms**, allowing us to query numerous constraints at interactive rates, and snap to the best one.

For longer clips, it is often not necessary to process every frame. Instead, we subsample the video temporally and compute cost matrices on these. For Figures 8 and 10, we temporally subsampled by a factor of $\frac{1}{5}$. This greatly reduces running time, allowing us to generate results for sequences with 4000+ frames. We then use the full video to compute interpolated images at fractional frame indices, without loss of temporal resolution.

**User Interface** We have developed a prototype interface that allows temporal alignment of multiple videos in a timeline, well known from standard video editing tools. We introduce a visualization of the amount of temporal warping by stretching and squeezing the bar representing each clip (see e.g. Figure 1(c)), which gives a quick and intuitive way to visualize the relative speed of the entire warped video when setting constraints.

We also provide a quick visual reference for how *well* the current path aligns the two videos at every frame. This is visible as a colored bar that exists above the overlap region of two videos (please see the supplemental video for an example), and is computed by evaluating the cost matrix at each frame-correspondence as determined by the path. This alignment-quality visualization gives artists instant feedback, allowing her to judge which parts of the video are well aligned given the current constraints.

## 7 Limitations and Future Directions

In order to synchronize object actions or camera motion with nonlinear temporal relationships, we must incur some amount of temporal warping. In certain applications this warping may be unacceptable, and lead to visible temporal distortions. In these cases (and if linear temporal alignment is not sufficient), aligning the videos without warping is not feasible. To address these concerns, we gave the artist the ability to set the bounds of allowed temporal warping per video, after which the best *possible* path will be found. Additionally, by choosing the parameterization of the path, it is possible for the artist to direct temporal distortion into different videos, or to spread it out evenly.

Currently our method supports only one frame alignment constraint. Technically it is trivial to allow the artist to select multiple points in a video, and then interpolate the alignment between these, requiring only the computation of additional min-cost trees from each added constraint point. In our interface, we tried to keep the interactions as simple as possible, e.g., in this case just dragging a video. The cost matrix and path visualization was intended only as a debug tool so far. However, for advanced artists, much more complex interactions are possible, for example drawing directly into the cost matrix could specify complex alignment constraints that could be integrated in the optimal path computation.

While our method is very robust, it still depends on the existence of usable and descriptive features. When such features do not exist, our method cannot find temporal relationships, as is visible in Figure 15g. However, we have provided a general framework for alignment, and when alternate modalities are available (such as skeletons), these can be easily incorporated into our system.

Finally, consistent spatiotemporal alignment is still an unsolved problem. Our method simplifies the task by computing a temporal mapping such that the content is as similar as possible, however if views are significantly different, alignment can still pose problems. We hope that by showing that high quality spatiotemporal results can be found using a naive spatial alignment method, future researchers will be encouraged to consider robust temporal synchronization as an essential preprocessing step to developing more advanced spatial alignment methods.

## 8 Conclusion

At the center of our work is a robust technique for computing temporal mappings in an efficient manner. We have designed our approach to operate at interactive speeds and for multiple videos, thus creating a natural fit into existing modern video editing pipelines, and allowing for a new interaction mechanism. We made relevant design decisions to present nonlinear temporal snapping to the artist in the most intuitive way possible. In addition, we present solutions to some important new problems for the video editing framework, specifically a constrained shortest path method for finding shortest path trees, and a graph construction that supports partial overlap. Additionally, we presented a novel method for computing a global temporal alignment of multiple clips, using a graph-based pairwise

alignment approach. This method enables new applications such as placing multiple independent videos into a single global time axis.

Our method is capable of robustly finding high quality non-linear temporal alignments and it has few parameters that remain fixed for almost all results presented. While user interaction is a key part of the paper, fine tweaking of the paths was *not* required. In no case was the amount of interaction needed greater than a couple of seconds for any examples presented.

We believe that our synchronization technology has the potential to enable new applications. By bringing the often ignored aspect of temporal alignment into light, we hope to encourage both researchers and video artists alike to consider the significant new possibilities that can be achieved.

## Acknowledgments

We would like to thank the reviewers for helpful comments, Victoria Bloom and Peter Sand for sharing their datasets, Federico Perazzi for stitching the panorama example, and Tze-Koong Wang for three decades of encouragement and inspiration.

## References

AGARWALA, A., ZHENG, K. C., PAL, C., AGRAWALA, M., COHEN, M. F., CURLESS, B., SALESIN, D., AND SZELISKI, R. 2005. Panoramic video textures. *ACM Trans. Graph. 24*, 3, 821–827.

BAKER, S., AND MATTHEWS, I. 2004. Lucas-kanade 20 years on: A unifying framework. *IJCV 56*, 3, 221–255.

BAKER, S., SCHARSTEIN, D., LEWIS, J. P., ROTH, S., BLACK, M. J., AND SZELISKI, R. 2011. A database and evaluation methodology for optical flow. *IJCV 92*, 1, 1–31.

BLOOM, V., MAKRIS, D., AND ARGYRIOU, V. 2012. G3d: A gaming action dataset and real time action recognition evaluation framework. In *CVPR Workshops*, 7–12.

BRYAN, N. J., SMARAGDIS, P., AND MYSORE, G. J. 2012. Clustering and synchronizing multi-camera video via landmark cross-correlation. In *ICASSP*, 2389–2392.

CASPI, Y., AND IRANI, M. 2002. Spatio-temporal alignment of sequences. *IEEE TPAMI 24*, 11, 1409–1424.

DIEGO, F., PONSA, D., SERRAT, J., AND LÓPEZ, A. M. 2011. Video alignment for change detection. *IEEE Transactions on Image Processing 20*, 7, 1858–1869.

DIEGO, F., SERRAT, J., AND LÓPEZ, A. M. 2013. Joint spatio-temporal alignment of sequences. *IEEE Transactions on Multimedia 15*, 6, 1377–1387.

DIJKSTRA, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

EVANGELIDIS, G. D., AND BAUCKHAGE, C. 2013. Efficient subframe video alignment using short descriptors. *IEEE TPAMI 35*, 10, 2371–2386.

JIANG, Y.-G., NGO, C.-W., AND YANG, J. 2007. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *CIVR*, 494–501.

KANG, S. B., UYTTENDAELE, M., WINDER, S. A. J., AND SZELISKI, R. 2003. High dynamic range video. *ACM Trans. Graph. 22*, 3, 319–325.

LI, R., AND CHELLAPPA, R. 2010. Aligning spatio-temporal signals on a special manifold. In *ECCV (5)*, 547–560.

LIU, C., YUEN, J., AND TORRALBA, A. 2011. Sift flow: Dense correspondence across scenes and its applications. *IEEE TPAMI 33*, 5, 978–994.

LOWE, D. G. 1999. Object recognition from local scale-invariant features. In *ICCV*, 1150–1157.

NGO, C.-W., MA, Y.-F., AND ZHANG, H. 2005. Video summarization and scene detection by graph modeling. *IEEE Trans. Circuits Syst. Video Techn. 15*, 2, 296–305.

PÁDUA, F. L. C., CARCERONI, R. L., SANTOS, G. A. M. R., AND KUTULAKOS, K. N. 2010. Linear sequence-to-sequence alignment. *IEEE TPAMI 32*, 2, 304–320.

PRIM, R. C. 1957. Shortest connection networks and some generalizations. *Bell system technical journal 36*, 6, 1389–1401.

RAO, C., GRITAI, A., SHAH, M., AND SYEDA-MAHMOOD, T. F. 2003. View-invariant alignment and matching of video sequences. In *ICCV*, 939–945.

RÜEGG, J., WANG, O., SMOLIC, A., AND GROSS, M. H. 2013. Ducttake: Spatiotemporal video compositing. *Comput. Graph. Forum 32*, 2, 51–61.

SAND, P., AND TELLER, S. J. 2004. Video matching. *ACM Trans. Graph. 23*, 3, 592–599.

SCHARSTEIN, D., AND SZELISKI, R. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV 47*, 1-3, 7–42.

SHRESTHA, P., BARBIERI, M., AND WEDA, H. 2007. Synchronization of multi-camera video recordings based on audio. In *ACM Multimedia*, 545–548.

SUMMA, B., TIERNY, J., AND PASCUCCI, V. 2012. Panorama weaving: fast and flexible seam processing. *ACM Trans. Graph. 31*, 4, 83.

UKRAINITZ, Y., AND IRANI, M. 2006. Aligning sequences and actions by maximizing space-time correlations. In *ECCV (3)*, 538–550.

VEDALDI, A., AND FULKERSON, B., 2008. VLFeat: An open and portable library of computer vision algorithms.

YÜCER, K., JACOBSON, A., HORNUNG, A., AND SORKINE, O. 2012. Transfusive image manipulation. *ACM Trans. Graph. 31*, 6, 176.

ZHOU, F., AND LA TORRE, F. D. 2009. Canonical time warping for alignment of human behavior. In *NIPS*, 2286–2294.

ZHOU, F., AND LA TORRE, F. D. 2012. Generalized time warping for multi-modal alignment of human motion. In *CVPR*, 1282–1289.

ZIMMER, H., BRUHN, A., AND WEICKERT, J. 2011. Optic flow in harmony. *IJCV 93*, 3, 368–388.