Tools for Placing Cuts and Transitions in Interview Video

Floraine Berthouzoz University of California, Berkeley Wilmot Li Adobe Systems Maneesh Agrawala University of California, Berkeley

Abstract

We present a set of tools designed to help editors place cuts and create transitions in interview video. To help place cuts, our interface links a text transcript of the video to the corresponding locations in the raw footage. It also visualizes the suitability of cut locations by analyzing the audio/visual features of the raw footage to find frames where the speaker is relatively quiet and still. With these tools editors can directly highlight segments of text, check if the endpoints are suitable cut locations and if so, simply delete the text to make the edit. For each cut our system generates visible (e.g. jump-cut, fade, etc.) and seamless, hidden transitions. We present a hierarchical, graph-based algorithm for efficiently generating hidden transitions that considers visual features specific to interview footage. We also describe a new data-driven technique for setting the timing of the hidden transition. Finally, our tools offer a one click method for seamlessly removing 'ums' and repeated words as well as inserting natural-looking pauses to emphasize semantic content. We apply our tools to edit a variety of interviews and also show how they can be used to quickly compose multiple takes of an actor narrating a story.

CR Categories: I.3.8 [Computer Graphics]: Applications

Keywords: Interaction, Human-Computer Interfaces

Links: 🗇 DL 🖾 PDF 🐻 WEB 📀 VIDEO 🍋 DATA

1 Introduction

Interview video frequently appears in broadcast news, documentary films, and online lectures. Such video usually focuses on the head and upper body of a person who explains a concept while looking at the camera. The challenge in editing interview video is to convert hours of raw footage into a concise clip that conveys the desired story. The process involves two steps: (1) choosing where to cut segments of the raw footage and (2) creating audio/visual transitions that connect the remaining segments into the final clip.

To place a cut, the editor must consider both the semantic content and the audio/visual attributes of the raw footage. Editors often begin by watching the entire raw footage several times and manually building an index that describes its content. Once the editor decides on the story, he can use the index to find the segments corresponding to specific content. The editor must also account for the audio/visual attributes of the segments when deciding where to cut. For example, cutting while the speaker is in the middle of saying a word or is gesturing energetically is often undesirable because such cuts interrupt the audio/visual flow of the video [O'Steen 2009].



Figure 1: Our interface. Selecting text in the transcript highlights corresponding regions of the timeline. Vertical bars in the transcript and blue bars in the timeline visualize cut suitability; places where a cut is least likely to interrupt the flow of the video.

After choosing the rough position of a cut the editor must create a transition that connects the video segments before and after the cut. Such transitions fall into two main categories;

- *Hidden Transitions* conceal the cut by smoothly joining the segments. Thus, they allow viewers to remain focused on the continuous flow of semantic content in the video.
- *Visible Transitions* introduce a noticeable change in audio/visual content (e.g. jump-cut, fade, etc.) between the segments. They usually indicate a break in the flow of semantic content.

Producing hidden transitions can be difficult as it requires either synthetically interpolating new frames or finding existing frames within the footage to seamlessly join the segments. Yet, interpolation [Mahajan et al. 2009; Brox et al. 2004] is only effective when the endpoint frames are similar in appearance, and manually searching for existing frames in the entire raw footage is impractical. As a result, editors often end up using visible transitions to join segments even when there is no break in the flow of semantic content.

In this work, we present tools for interactively placing cuts and creating hidden as well as visible transitions in interview video footage. To help editors place the cuts, we combine crowdsourcing (castingwords.com) to obtain a text transcript of the raw footage, with text-audio alignment software [Virage] to build an index that links each word in the transcript with the corresponding location in the footage. The editor can then directly edit the text script and our tools propagate the changes to the video. We also visualize the suitability of cut locations by analyzing the audio/visual features of the raw footage to find frames where the speaker is relatively quiet and still (Figure 1). Together these tools allow the editor to focus on the semantic content and quickly select cut locations that do not interrupt the flow of the video in an unnatural manner.

For each cut, our system automatically generates hidden and visible transitions so that the editor can see each possibility and choose whichever one is most appropriate. We present a new algorithm for efficiently generating hidden transitions. Our method builds on the general approach of constructing a similarity graph between



Figure 2: Example editing session with our interface. The transcript view (top) shows transcript text aligned with the video timeline (bottom). The editor considers two options (a–b) for removing the phrase "communicating that story." The cut suitability visualization in both the transcript view and timeline indicates that option (a) is a better cut because it lines up with vertical bars in the transcript and peaks in the timeline which imply high cut suitability. Our interface also highlights in red the 'ums' and repeated words that can be removed and automatically replaced with seamless transitions, even when the suitability scores are low (c).

the raw frames and walking the graph to select the smoothest set of in-between frames [Schödl et al. 2000; Kwatra et al. 2003; Kemelmacher-Shlizerman et al. 2011]. Unique to our approach is a frame matching distance that considers facial appearance, body appearance, and location of the speaker within the frame. We also use hierarchical clustering to significantly reduce the number of frameto-frame comparisons necessary to achieve good performance.

To further refine the resulting transition we apply optical flow interpolation [Brox et al. 2004] with a new data-driven technique for selecting the number of interpolated frames. Our technique sets the timing of the transitions so that any motions of the speaker appear natural. Such timing is crucial for maintaining the flow of the video.

Beyond placing simple cuts and transitions, our interface offers additional tools for increasing the quality of the final clip. The transcript view highlights 'ums' and repeated words and gives editors a one click option for seamlessly removing them. Editors can also insert natural-looking pauses, in which the speaker appears to be at rest, to further emphasize important semantic content.

We apply our tools on videos from a variety of sources including a corporate marketing department and professional journalists. In addition to interview editing, we show how our interface can be used to quickly compose multiple takes of an actor narrating a story. Finally, we demonstrate the success of our approach by providing visual comparisons to a number of other techniques.

2 Related Work

Commercial video editing software such as Adobe Premiere or Apple FinalCutPro provide low-level tools for rearranging segments of video. These tools force editors to manually review, segment and compose the video into the final clip. As a result researchers have developed higher-level techniques to facilitate editing. We focus on the subset of these techniques that are most related to our work.

Navigating, Indexing and Editing Videos: Reviewing long sequences of raw footage can be tedious. Truong and Venkatesh [2007] survey a number of techniques for indexing video by first segmenting it into shots [Boreczky and Rowe 1996] and then summarizing each shot by a static keyframe [Zhang et al. 1995]. Users can navigate and edit the video by scanning through the keyframes and rearranging them [Ueda et al. 1991; Girgensohn et al. 2000]. Researchers have also developed more sophisticated motion segmentation methods to enable navigation via direct manipulation [Dragicevic et al. 2008; Goldman et al. 2008; Karrer et al. 2008]. Another approach is to segment the video based on speech recognition [Bregler and Omohundro 1995; Potamianos et al. 2003]. Silver [Casares et al. 2002] is a complete editing system that allows users to navigate by clicking on words

in the transcript. While our system similarly allows navigation and editing through a text transcript we also provide tools to visualize suitable cut locations and to generate transitions.

Ranjan et al. [2008] propose a system designed for videos of group meetings that applies television production principles to automatically capture and then edit the video. However, their work only focuses on visible transitions and cannot generate hidden transitions.

Transitions Based on Reusing Frames: Video texture [Schödl et al. 2000; Kwatra et al. 2003; Agarwala et al. 2005] and video completion [Wexler et al. 2007] methods generate seamless loops or fill-holes by copying frames or spatio-temporal patches based on pixel-level color and motion similarity. The cost of these methods scales poorly with the number of pixels and is prohibitive even for relatively short videos of over 1000 frames. Moreover, these methods are designed primarily for stochastic video and cannot handle the highly structured appearance and motion of interview footage. Video rewrite [Bregler et al. 1997] focuses on talking head videos; it extracts the speaker's mouth position for each phoneme in the original video and can then re-composite the mouth to match the phonemes in a new audio track. Kemelmacher-Shlizerman et al.'s [2010] puppeting system matches video of a puppeteer's face with the closest frames of another person. Neither technique is designed to smoothly transition between two endpoint frames.

Our work is closely related to Photobios [Kemelmacher-Shlizerman et al. 2011] which creates a face animation from a collection of photographs of the same person. It builds a facial similarity graph to transition between two photographs. We extend their approach in two ways. First, we include body/hand appearance and face location terms in our frame comparison metric to improve the smoothness of transitions. Second, we develop a hierarchical clustering approach based on our comparison metric to accelerate our technique.

Interpolation, Warping and Morphing: General-purpose techniques for smoothly synthesizing intermediate images between a pair of endpoint frames include interpolation [Brox et al. 2004; Mahajan et al. 2009], warping [Gomes 1999] and morphing [Shechtman et al. 2010]. These methods do not account for appearance and motion constraints imposed by the visual content and therefore require the endpoint frames to be extremely similar to produce good results. Face-specific warping techniques [Dale et al. 2011] focus on transferring expressions or entire faces to a new image, but cannot handle gestures or bodies. Moreover, these techniques are not designed to synthesize new intermediate frames but rather to replace regions of existing frames. Methods for smoothly interpolating or morphing facial expressions often rely on first mapping the video to a 3D face model [Blanz and Vetter 1999; Zhang et al. 2004; Pighin et al. 1998]. As a result these techniques are computationally expensive and may require a specialized 3D capture setup.



Figure 3: We use our frame distance metric D(i, j) to select frames of the footage that are visually closest to a reference frame (columns 2, 3). When we drop the face appearance term (columns 4, 5) we identify frames that match the body, but not the face, and vice versa when we drop the body appearance term (columns 5, 6). The L_2 distance produces poor matches because it cannot distinguish whether large pixel differences are due to changes in hand positions or facial expressions. Note that we eliminate frames that are within a window of 20 frames from the reference to ensure a greater variety in the selected frames for these comparisons. Frame timestamps are given in the corners.

3 Video Editing Interface

Our interface includes several features that help editors find relevant portions of the footage, choose cut locations, and decide which transitions to use. We present these features in the context of an example editing session (Figures 1 and 2). We encourage readers to also view the paper video 1 which walks through the features.

Linked Transcript View: Interview editors often start by transcribing the footage and deciding on the story [O'Steen 2009]. To facilitate this task, our interface provides a *transcript view* that shows text obtained via a crowdsourced transcription service (castingwords.com). The service costs \$1.50 per minute of footage and, in our experience, produces far superior results to commercial speech-to-text software. We automatically align the transcript to the footage using off-the-shelf software from Autonomy [Virage]. Highlighting text selects the corresponding segment of the timeline and vice versa. In our example, the editor uses the transcript view to identify a segment about digital stories and selects the text to quickly locate the corresponding footage (Figure 2a).

Cut Suitability Visualization: Editors rarely place a cut when the speaker is in the middle of talking or gesturing actively because such cuts disrupt the audio/visual flow [O'Steen 2009]. To help editors avoid such cuts, we identify segments in which the speaker is relatively quiet and still. Our interface visualizes such cut locations in both the transcript view and timeline. The timeline plots a per-frame cut suitability score and the transcript view renders a vertical bar wherever the set of inter-word frames contains at least one frame with a suitability score greater than 0.9 (out of 1.0). Thus, vertical bars indicate places where a cut is most likely to yield a high-quality hidden transition and not disrupt the flow of the video. In our example, the editor considers how to remove the phrase "communicating that story." While Figures 2a and 2b show two ways to cut the text, the cut suitability visualizations indicate that the first option will create a better hidden transition. We also identify 'ums' and repeated words that can be cut and replaced with hidden transitions, even when the suitability scores are relatively low (Figure 2c). The editor can then remove such artifacts with a single click. Section 5 describes our techniques for computing cut suitability scores and identifying removable words. An editor may also choose to make a cut for purely semantic reasons, even though the suitability scores are low. In such cases, our system is still often able to generate reasonable hidden transitions (Figure 6 and 7). Once the editor has decided where to cut, he simply deletes the corresponding text in the transcript view, and the edit propagates to both the timeline and the resulting video.

Transition Alternatives: Our system generates hidden transitions, pauses and visible transitions for each cut and allows the editor to review these alternatives before deciding which one to apply. As discussed earlier, hidden transitions join video segments without breaking the flow of the video. Pauses are video segments in which the speaker does not talk and appears to be at rest. In radio, such pauses are frequently used to highlight semantic content [Abel and Glass 1999]. They are less common in video because natural pauses are difficult to synthesize. Section 6 describes how our system assists editors in inserting pauses into the interview.

The most basic type of visible transition is a jump-cut, which simply joins the endpoint frames without inserting anything inbetween. Usually the speaker will appear to jump from one location in the frame to another. While small differences between adjacent frames can look like skipping or stuttering artifacts, larger differences are usually less perceptually jarring. For example, a zoomin/zoom-out jump-cut, transitions to a tighter or wider crop of the next video segment and thus suggests a change in camera viewpoint. We generate such zoom-in/zoom-out jump-cuts by computing the image space bounding box of the speaker's face across the video frames following the cut and then cropping the frame either closer or farther away from this bounding box. Finally, we also generate cross-fades and fade-to-black transitions, which are typically used to indicate the passage of time. Once the editor chooses a transition, the timeline is updated to indicate the transition type with a colored vertical bar. This allows the editor to see, at a glance, which transitions he has used across the entire clip.

4 Frame Distance

Our algorithms for locating suitable cuts and generating hidden transitions rely on a new appearance-based frame comparison metric designed for interview videos. Our metric extends the face comparison metric of Kemelmacher-Shlizerman et al. [2011]. We first describe the features used in the metric and then the metric itself.

4.1 Features

As a pre-process to computing the features, we apply Saragih et al.'s [2009] face tracker to locate 66 landmark points along the contour of the face, the eyes, eyebrows, nose and mouth.

Face Appearance: Like Kemelmacher-Shlizerman et al. [2011], we capture face appearance by dividing the bounding boxes of the eyes and mouth into a small grid of cells $(2 \times 3$ for the eyes and 3×3 for the mouth) and computing a HOG feature [Dalal and Triggs 2005] for each cell. We then build separate feature vectors for the mouth and eyes by concatenating the HOG features across the cells for these two facial components.

¹http://vis.berkeley.edu/papers/vidtrans/



Figure 4: The cut suitability score $S(i) = S_a(i)S_v(i)$ identifies frames where the speaker is relatively quiet and still. The audio score $S_a(i)$ analyzes the alignment between the audio channel and the transcript to find the quiet segments. The visual score $S_v(i)$ identifies frames that are similar to a frame in which the speaker is at rest. Segments 1 and 3 (marked in red) pass the cut suitability threshold S(i) > 0.9 and the representative frames are very similar. Cutting from segment 1 to segment 3 is likely to produce high-quality seamless transition.

Body Appearance: To capture body appearance, we first define the bounding box of the body as the region of the frame centered immediately below the face contour and continuing to the bottom of the frame. We have found that scaling the width of this bounding box to 3.2 times the width of the face usually covers the entire body region. We then divide this bounding box into 4×7 cells and concatenate the HOG features for each cell into a feature vector. Since hand gestures usually occur in front of the body, this feature also encodes changes in appearance due to hand motions.

Face Location: To capture face location we average the (x, y) position of all face landmarks. We do not encode body location because we found that in practice it is strongly correlated to face location.

4.2 Distance Metric

To quantitatively compare two frames of interview footage i and j, we modify Kemelmacher-Shlizerman et al.'s [2011] distance function to include the body appearance term as follows

$$D_{app}(i,j) = 1 - (1 - \lambda^m d_{ij}^m)(1 - \lambda^e d_{ij}^e)(1 - \lambda^b d_{ij}^b)).$$
(1)

The superscripts $\{m, e, b\}$ represent the mouth, eyes and body respectively, the d_{ij} 's are the normalized χ^2 -distances corresponding to the appearance feature vectors and the λ 's are weights that quantify the relative importance of each feature. We normalize the χ^2 -distances to the range [0, 1] using the robust logistic function $L(d) = (1 + e^{-\gamma(d^x - \mu^x)/\sigma^x})^{-1}$, where $\gamma = ln(99)$. We empirically found that setting $\lambda^m = 0.3$, $\lambda^e = 0.1$, and $\lambda^b = 0.6$ produced good results for all of our examples.

We compute location-based distance $D_{loc}(i, j)$ as the L_2 difference between the face location features of frames i and j and normalize them using the robust logistic function L(d) described above. We then combine D_{app} and D_{loc} into a frame distance function

$$D(i,j) = \left[\lambda^{app} D_{app}(i,j) + \lambda^{loc} D_{loc}(i,j)\right]^{\alpha}.$$
 (2)

We set the weights $\lambda^{app} = 0.6$ and $\lambda^{loc} = 0.4$ for all our examples. The exponent α nonlinearly scales the distances and allows us to control the number of in-between frames to select in the case of seamless transitions, and we set $\alpha = 2$. We found that combining the terms in this additive way provides good control over the relative importance of appearance versus location features. Figure 3 shows the importance of using body/hand appearance and location in addition to face appearance as proposed in Photobios. It also shows that using the L_2 distance as is commonly used in general video texture methods [Schödl et al. 2000], can produce poor visual matches in the context of interview footage.

5 Cut Suitability Score

As described in Section 3, editors prefer to place cuts when the speaker is relatively quiet and still, to avoid disrupting the flow of the video. We identify such potential cut locations and visualize them in the timeline and transcript views.

We process each frame *i* of the raw footage and assign it a cut suitability score $S(i) = S_a(i)S_v(i)$ where $S_a(i)$ considers the audio channel and $S_v(i)$ considers the video channel. To identify whether the speaker is relatively quiet (e.g., between sentences, clauses or words) we rely on the alignment between the transcript and the audio. If frame *i* falls between two words we set the audio score $S_a(i)$ to 1, otherwise we set it to 0. To identify whether the speaker is actively gesturing, we compute the appearance distance $D_{app}(i, j)$ (Eqn. 1) between frame *i* and a reference frame *j* where the speaker is at rest. By default we use the first frame of the footage as rest frame since the speaker is often listening to the interviewer at the beginning of the video. The editor can optionally select a different rest frame. Finally we set the visual score $S_v(i)$ to $1 - D_{app}(i, j)$.

We identify potential cut locations by thresholding the cut suitability score S(i) > 0.9. In Figure 4, we find segments of frames at positions 1 and 3 (marked in red) that pass the threshold test. The representative frames appear very similar to one another in these segments. The editor could eliminate the extraneous part of the sentence between these two segments and our system would produce a high-quality seamless transition. Alternatively the editor could transition from segment 2 to 4 without changing the semantic content of the sentence. However suitability scores are much lower for these two segments; the representative frames are not very similar, and generating a hidden transition would be much more difficult.

Editors may also create cuts to better convey the semantic content even though the suitability score is low. For example, in Figure 4, the editor might cut between segments 4 and 5 and between 6 and 7, to remove an 'um' and a repeated 'the'. The endpoint frames for such cuts are often close to one another in time and are usually similar in appearance. Thus, our system is able to produce seamless transitions between them. Our system marks all the 'ums' and repeated words that can be removed easily; if the frame distance D(i, j) (Eqn. 2) between the endpoint frames of the segment containing the word is less than 0.1 we highlight it in red.

Once the editor selects a set of words to remove from the transcript, we further optimize the location of the cut endpoints. We compute the frame distance D(i, j) between all pairs of frames *i* and *j* where *i* falls in the segment of frames between the words at the start of the cut and *j* falls in the segment of frames between the words at the end of the cut. If both of the between-words segments contain frames that pass the cut suitability threshold we limit the computation to just those frames. Finally, we select the lowest distance pair as the cut endpoints. We have found that optimizing the cut endpoints in this manner greatly increases the quality of hidden transitions.

6 Generating Hidden Transitions

A brute-force approach for generating hidden transitions is to first build a fully-connected graph in which each node is a frame and edge weights encode frame distance (Eqn. 2). Given the start and end frames of a cut, we can apply Dijkstra's shortest path algorithm to find the smoothest set of in-between frames [Schödl and Essa 2002; Zhang et al. 2004; Kemelmacher-Shlizerman et al. 2011].

The drawback of this brute-force approach is its $O(n^2)$ complexity; it requires computing the distance between each pair of frames. Building the graph for a 10 minute video (30 fps) would require 325 million frame distance calculations. Although such graph construction is highly parallelizable, access to a large cluster of computers would be essential for this approach to be practical.

We bypass this problem by hierarchically clustering the input frames as a pre-process. Then for each cut we compute the lowest cost path in a top-down manner. While our clustering approach only approximates the minimum cost path, it requires far less computation than the brute force approach and produces good-quality results. We also provide an interpolation algorithm that hides any remaining visual jumps while maintaining natural-looking timing of the speaker's head and body motions. Finally, we compute audio transitions that seamlessly connect the segments across the cut.

6.1 Hierarchical Clustering

We cluster similar frames of the raw footage using k-means with fixed cluster centers. To start the process we select the first frame i = 1 of the video as a cluster center and assign all frames j such that D(i, j) < 0.06, to this cluster. We then pick the frame furthest from the current set of cluster centers as the next center and repeat the process until all frames are assigned. To form the next level of the hierarchy, we cut the distance threshold in half and apply the same k-means algorithm within each cluster. The subdivision terminates when a cluster contains less than 100 frames or when the distance between each frame and the center is less than 0.01.

Although the worst case complexity of this algorithm is $O(n^2)$, in practice we have found it to be far more efficient. In the first frame of the footage the speaker usually sits still. Since the first frame is also the first cluster center, our algorithm initially groups together all the frames in which the speaker is not gesturing. Such frames make up a large percentage of typical interview video. Our algorithm is order dependent, but because it immediately identifies the largest cluster of frames, it avoids many unnecessary comparisons.

Our algorithm is designed to handle our non-Euclidean distance metric. While researchers have developed efficient clustering methods, including nearest neighbor [Arya et al. 1998] and Nystrom techniques [Fowlkes et al. 2004], most assume a Euclidean space. We have tried using BoostMap [Athitsos et al. 2004] to embed the data in a Euclidean space before applying one of the efficient clustering methods. But, as shown in Section 7, this method does not capture our frame distance accurately and produces artifacts.

6.2 Hierarchical Shortest Path Computation

To find a smooth set of transition frames, we compute an approximate shortest path between the endpoints of the cut using the hierarchical cluster representation. We build the path top-down, starting from the highest-level clusters and then refining the path to the next finer level until we reach individual frames.



Figure 5: Shortest path computation to find in-between frames. We build a distance matrix between the highest-level frame clusters C_1 , C_2 , C_3 and C_4 . Dijkstra's algorithm yields C_1 , C_2 , C_4 as the shortest path. For each consecutive pair of clusters (C_1 and C_2 , then C_2 and C_4), we update the matrix to include distances between subcluster centers and then recompute the shortest path. We repeat this process until we reach the finest level of the hierarchy.

Step 1: We find the highest-level clusters containing the start and end frames and then compute the shortest path between them using Dijkstra's algorithm. This computation requires building a matrix of pairwise distances between the highest-level cluster centers. Since we already computed these pairwise distances when building the clusters, we simply reuse them. In Figure 5, the endpoint frames lie in clusters C_1 and C_4 , and the shortest path goes through C_2 .

Step 2: To build the path at the next level of the hierarchy, we first refine the distance matrix as shown in Figure 5. For each consecutive pair of higher-level clusters on the path we update the matrix to include pairwise distances between all of their subcluster centers. We again note that we already computed the pairwise distances for the subclusters, so we reuse them (orange edges). However, we must compute the pairwise distances between subcluster centers that lie in adjacent top-level clusters (gray edges). We set all other distances to a large number and then apply Dijkstra's algorithm to compute the shortest path. The structure of the distance matrix ensures that the refined path follows the sequence of clusters computed at the higher level. In our example, the refined path cannot jump from a subcluster in C_1 to a subcluster in C_4 without first passing through a subcluster in C_2 . For further efficiency we memoize all of the pairwise distance computations so that they can be reused if necessary to generate additional seamless transitions.

Iteration: We repeat Step 2 until we reach the finest level of the cluster hierarchy. At that point each cluster node represents a frame and Dijkstra's algorithm yields that sequence of transition frames.

6.3 Interpolation and Data-Driven Timing

Directly playing back the transition sequence of frames found by our hierarchical shortest path algorithm can produce unnaturally fast motions for the speaker's head and hands. To slow the motions and hide any remaining visual jumps, we synthesize additional in-between frames; for each consecutive pair of transition frames we first compute dense optical flow using the approach of Brox et al. [2004] and then interpolate the flow to form the new frames.

The number of interpolated in-between frames sets the timing of the transition. To ensure that the motions of the speaker appear natural, we have developed a data-driven approach to learn the appropriate number of in-between frames. We consider pairs of raw footage frames (i, j) and encode the relationship between the number of intermediate frames N(i, j) = |i-j| and the frame distance D(i, j) as a 2D histogram. As shown in the inset below, each row of this histogram corresponds to one bin of D(i, j) and forms a 1D histogram of the distribution of raw footage frame pairs (i, j) over



Figure 6: Examples of hidden transitions generated by our algorithm for Maria (top) and Louis (bottom). The endpoint frames are outlined in red. We show only the in-between frames found by our shortest path computation and do not show frames generated by our interpolation method. Colored under-bars indicate frames that are at most 20 frames apart in the raw footage.

N(i, j). We use 50 bins for frame distance D(i, j) and limit the maximum number of intermediate frames N(i, j) to 20. For efficiency, we generate the frame pairs (i, j) by randomly sampling 2000 frames i and considering all frames $j \in [i-20, i+20]$.

To determine the appropriate number of in-between frames for a pair of frames in our transition sequence (p, q), we first compute the frame distance D(p, q) and look up the corresponding row of the 2D histogram. The mode of this 1D row histogram corresponds to the most frequent number of intermediate frames N(i, j) for that frame distance bin. In practice we have found that when speakers



move slowly, the 1D histogram is Nr of Intermediate Frames N(ij) relatively flat as frame distance is similar for all pairs (i, j), regardless of the number of intermediate frames. In such cases, simply using the mode N(i, j) as the number of in-between frames introduces high variance in timing because a small amount of variance in the flat distribution can lead to large differences in the mode.

We use a more robust approach. We first find all the peaks in the 1D histogram that are within 10% of the mode (orange and red bars in inset). Amongst those peaks we choose the one with the smallest N(i, j) as the number of in-between frames to interpolate (red bar in inset). Although this robust formulation favors the fastest transition when the histogram is relatively flat, we have found that it produces natural looking speaker motions.

6.4 Pauses

Our system assists editors in inserting pauses of any desired length. As a pre-processing step, we identify pauses as segments of the raw footage in which the appearance of the mouth is stable i.e. changes very little within a window of 10 frames. We first mark frames i where $D_{app}(i, j) < 0.02$ for all $j \in [i-5, i+5]$. Since we are only interested in the mouth, we set $\lambda^m = 1$, and $\lambda^e = \lambda^b = 0$ to evaluate $D_{app}(i, j)$. We then label any sequence of 10 or more marked frames as a pause segment. Note that we cannot use the audio to identify pauses, since that would eliminate all sequences where the speaker is at rest, but the off-screen interviewer is talking.

The editor can then choose to insert a pause at any location in the video. We compute the average frame distance between the frames on either side of the desired pause location and the corresponding endpoint frames of every pre-marked pause segment and we select the segment that yields the minimum average frame distance. The editor can optionally specify a desired pause duration. If the desired

		# Transitions			Computation Time		
	Raw Time	Hidden	Pauses	Visible	Clustering	Hidden	Pauses
Huda	14m 47s	12	5	0	47m	7s	12s
Kate	19m 22s	14	3	0	1h 12m	9s	14s
Louis	8m 07s	18	6	2	30m	5s	14s
Maria	4m 38s	14	2	1	22m	5s	9s
Vidi	17m 29s	10	3	2	52m	8s	16s

Table 1: Number of transitions and computation times for five videos. Raw time is the length of the input footage which was captured at 30 frames per second.

duration is longer than the selected pause segment, we use the video texture algorithm [Schödl et al. 2000] to extend the pause segment. If the desired duration is shorter than the segment we simply drop the extra frames. Finally, we generate seamless hidden transitions at both endpoints to smoothly insert the segment into the video.

Audio Transitions: Pauses and hidden transitions introduce additional frames into the video. However, we do not have corresponding audio for these frames. To fill these audio gaps we isolate the background noise of the environment. We threshold the amplitude of the audio signal to find relatively quiet moments and join them using an audio cross-fade that slowly fades in the audio of the next segment. The fade is essential to prevent audible clicking artifacts.

7 Results

Table 1 and Figures 6, 7 and 8 summarize our results. We used our tools to edit five videos we obtained from several sources including a corporate marketing department (Louis), professional journalists (Maria, Vidi) and student actors (Huda, Kate). The running time of the raw footage and the number of transitions we created are given in Table 1. We used the suitability score to place most of the cuts, and we removed all of the 'ums' and repeated words, which ranged between 2 instances for Vidi and 11 for Maria. We also placed several more challenging cuts that were designed to maintain the semantic flow of the story, even though the suitability scores were low. The edited videos run 50-100s in length. We encourage readers to view the videos on our website ².

Since visible transitions are easy to create using off-the-shelf video editing software, we focused on generating hidden transitions. While our system could generate good hidden transitions in almost all instances, in about 10% of the cases we chose to either remove a cut or use a visible transition because we were unsatisfied with the results. However even for many of the challenging cuts, where the endpoint frames are dissimilar (Figure 6 and Figure 7 top), our method was able to create a hidden transition with a plausible set

²http://vis.berkeley.edu/papers/vidtrans/



Figure 7: Comparison of in-between frames found using our shortest path computation (top) versus BoostMap (middle) and optical flow interpolation (bottom). The frames found using BoostMap differ significantly from one another and from the ending frame. Interpolation between significantly different endpoints also produces poor results as Kate's right arm appears to fade in from the background.



Figure 8: To transition between the endpoint frames (red outline), our shortest path algorithm finds two in-between frames from the raw footage and generates four intermediate frames using optical flow interpolation (blue outline). However, the interpolation fails to properly account for the motion of Louis' hands and instead simply fades them in.

of in-between frames. These figures do not include the optical flow based interpolation frames. However, the number of interpolated frames sets the timing of the transition and choosing it correctly is essential. The videos show that our learning approach for choosing this number produces much better results than simply setting it to a constant (e.g., 8 frames in our examples).

Table 1 reports the computation time required to cluster the raw footage frames and the average time required to generate the hidden transitions and pauses, using our unoptimized MATLAB implementation on a 2.4GHz MacBook Pro with 8GB of memory. While clustering can take up to 1h 20m for a 20m video, the average time for computing a hidden transition is 6.8s. Thus, once the clustering pre-process is complete the editor can create and test possible transitions relatively quickly.

We have also used our interface to combine different takes of an actor narrating a story. For each take, we cued the lines of the actors and occasionally directed them to increase or decrease their expressiveness. In the final edit, we then created a number of seamless hidden transitions to merge the best segments of the different takes. The videos contain the complete resulting stories.

Figure 7 compares our hidden transition algorithm to two alternative methods. First, we use Athitsos et al.'s implementation of BoostMap [2004] to embed our frame distance function into a Euclidean space, and then construct the hierarchical clusters using approximate nearest neighbors [Arya et al. 1998]. Second, we use optical-flow interpolation to generate a set of in-between frames. However, both approaches produce visible artifacts when the endpoint frames are dissimilar. More sophisticated interpolation algorithms [Mahajan et al. 2009; Shechtman et al. 2010] might improve the results. However, since these methods do not re-use existing frames, it is difficult for them to generate natural-looking motion when the endpoint frames differ significantly. *User Feedback:* We have informally shown our tools and results to five journalists and four professional video editors who often work with interview footage. All of them appreciated the ease of finding potential cuts through our transcript and timeline views. They felt that the hidden transition would give them a new alternative to visible cuts and extend the kinds of clips they could produce. The journalists reported that they would be either comfortable or very comfortable using these hidden transitions for videos designed for the web, and the video editors indicated that our proposed interface and tools would be useful in their interview editing workflows.

Limitations: While our approach for generating hidden transitions usually works well, if the endpoint frames differ significantly in appearance it can produce less successful results. For example, in Figure 8 Louis' hands are not visible at the start of the transition and must become visible by the end. While our algorithm finds two intermediate frames, they both show most of the hands. Our interpolation algorithm for refining the transition ends up simply fading in the hands. In general, the quality of our hidden transitions improves when the raw footage is longer, since our algorithm has a greater variety of frames to choose from. We have found that about 5 minutes of raw footage is sufficient to produce good results.

Our algorithms for generating hidden transitions and pauses work best when the raw footage contains a still background and the the lighting and camera position are fixed. When these conditions are violated the transitions may contain unnatural movement in the background or variations in illumination. In practice however, interviews often take place in such stable environments.

To compute cut suitability scores, we use a single frame as the rest frame. However, if the speaker moves significantly during the interview, his resting position may change. Our algorithm does not recognize new rest positions and will give low scores to these frames. As future work, we plan to extend our algorithms to compute still frames locally, rather than globally. However, we found that in our example videos, our global method was sufficient as the speaker was roughly at the same location throughout the footage.

8 Conclusion

Producing a high-quality interview video is a laborious process even for skilled editors. We have presented a set of tools that are designed to reduce the tedious, low-level work involved in placing cuts and creating transitions so that editors can focus on the highlevel work of telling a story. As video becomes the dominant form of sharing information on the Internet we believe that such semiautomated tools for increasing production quality will be essential.

Acknowledgments We thank Douglas Anarino for helping us develop our video editing interface, and Scott Cohen for providing us with optical flow interpolation code. This work was supported by National Science Foundation grant CCF-0643552

References

- ABEL, J., AND GLASS, I. 1999. *Radio: An illustrated guide.* WBEZ Alliance Inc.
- AGARWALA, A., ZHENG, K., PAL, C., AGRAWALA, M., COHEN, M., CURLESS, B., SALESIN, D., AND SZELISKI, R. 2005. Panoramic video textures. *Proc. SIGGRAPH* 24, 3, 821–827.
- ARYA, S., MOUNT, D., NETANYAHU, N., SILVERMAN, R., AND WU, A. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of ACM 45*, 6, 891–923.
- ATHITSOS, V., ALON, J., SCLAROFF, S., AND KOLLIOS, G. 2004. BoostMap: A method for efficient approximate similarity rankings. *Proc. CVPR*, II:268–II:275.
- BLANZ, V., AND VETTER, T. 1999. A morphable model for the synthesis of 3D faces. In *Proc. SIGGRAPH*, 187–194.
- BORECZKY, J., AND ROWE, L. 1996. Comparison of video shot boundary detection techniques. *JEI* 5, 2, 122–128.
- BREGLER, C., AND OMOHUNDRO, S. 1995. Nonlinear manifold learning for visual speech recognition. *Proc. ICCV*, 494–499.
- BREGLER, C., COVELL, M., AND SLANEY, M. 1997. Video rewrite: Driving visual speech with audio. In *Proc. SIGGRAPH*, 353–360.
- BROX, T., BRUHN, A., PAPENBERG, N., AND WEICKERT, J. 2004. High accuracy optical flow estimation based on a theory for warping. *Proc. ECCV*, 25–36.
- CASARES, J., LONG, A., MYERS, B., BHATNAGAR, R., STEVENS, S., DABBISH, L., YOCUM, D., AND CORBETT, A. 2002. Simplifying video editing using metadata. In *Proc. DIS*, 157–166.
- DALAL, N., AND TRIGGS, B. 2005. Histograms of oriented gradients for human detection. In Proc. CVPR, 886–893.
- DALE, K., SUNKAVALLI, K., JOHNSON, M., VLASIC, D., MA-TUSIK, W., AND PFISTER, H. 2011. Video face replacement. *Proc. SIGGRAPH ASIA 30*, 6, 130:1–130:10.
- DRAGICEVIC, P., RAMOS, G., BIBLIOWITCZ, J., NOWROUZEZAHRAI, D., BALAKRISHNAN, R., AND SINGH, K. 2008. Video browsing by direct manipulation. *Proc. CHI*, 237–246.
- FOWLKES, C., BELONGIE, S., CHUNG, F., AND MALIK, J. 2004. Spectral grouping using the nystrom method. *PAMI 26*, 2, 214–225.
- GIRGENSOHN, A., BORECZKY, J., CHIU, P., DOHERTY, J.,

FOOTE, J., GOLOVCHINSKY, G., UCHIHASHI, S., AND WILCOX, L. 2000. A semi-automatic approach to home video editing. *Proc. UIST*, 81–89.

- GOLDMAN, D., GONTERMAN, C., CURLESS, B., SALESIN, D., AND SEITZ, S. 2008. Video object annotation, navigation, and composition. *Proc. UIST*, 3–12.
- GOMES, J. 1999. Warping and morphing of graphical objects, vol. 1. Morgan Kaufmann.
- KARRER, T., WEISS, M., LEE, E., AND BORCHERS, J. 2008. DRAGON: A direct manipulation interface for frame-accurate in-scene video navigation. *Proc. CHI*, 247–250.
- KEMELMACHER-SHLIZERMAN, I., SANKAR, A., SHECHTMAN, E., AND SEITZ, S. 2010. Being John Malkovich. *Proc. ECCV*, 341–353.
- KEMELMACHER-SHLIZERMAN, I., SHECHTMAN, E., GARG, R., AND SEITZ, S. 2011. Exploring photobios. ACM Trans. on Graph. (Proc. SIGGRAPH) 30, 4, 61:1–61:10.
- KWATRA, V., SCHODL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *Proc. SIGGRAPH* 22, 3, 277–286.
- MAHAJAN, D., HUANG, F., MATUSIK, W., RAMAMOORTHI, R., AND BELHUMEUR, P. 2009. Moving gradients: A path-based method for plausible image interpolation. *Proc. SIGGRAPH 28*, 3, 42:1–42:11.
- O'STEEN, B. 2009. *The Invisible Cut: How Editors Make Movie Magic*. Michael Wiese Productions.
- PIGHIN, F., HECKER, J., LISCHINSKI, D., SZELISKI, R., AND SALESIN, D. 1998. Synthesizing realistic facial expressions from photographs. *Proc. SIGGRAPH*, 75–84.
- POTAMIANOS, G., NETI, C., GRAVIER, G., GARG, A., AND SE-NIOR, A. 2003. Recent advances in the automatic recognition of audiovisual speech. *Proc. IEEE 91*, 9, 1306–1326.
- RANJAN, A., BIRNHOLTZ, J., AND BALAKRISHNAN, R. 2008. Improving meeting capture by applying television production principles with audio and motion detection. In *Proc. CHI*, ACM, 227–236.
- SARAGIH, J., LUCEY, S., AND COHN, J. 2009. Face alignment through subspace constrained mean-shifts. *ICCV*, 1034–1041.
- SCHÖDL, A., AND ESSA, I. 2002. Controlled animation of video sprites. In Proc. SCA, 121–127.
- SCHÖDL, A., SZELISKI, R., SALESIN, D., AND ESSA, I. 2000. Video textures. *Proc. SIGGRAPH*, 489–498.
- SHECHTMAN, E., RAV-ACHA, A., IRANI, M., AND SEITZ, S. 2010. Regenerative morphing. *Proc. CVPR*, 615–622.
- TRUONG, B., AND VENKATESH, S. 2007. Video abstraction: A systematic review and classification. ACM TOMCCAP 3, 1.
- UEDA, H., MIYATAKE, T., AND YOSHIZAWA, S. 1991. IMPACT: An interactive natural-motion-picture dedicated multimedia authoring system. *Proc. CHI*, 343–350.
- VIRAGE. Audio analysis. http://www.virage.com/.
- WEXLER, Y., SHECHTMAN, E., AND IRANI, M. 2007. Spacetime completion of video. PAMI 29, 3, 463–476.
- ZHANG, H., LOW, C., SMOLIAR, S., AND WU, J. 1995. Video parsing, retrieval and browsing: an integrated and content-based solution. *Proc. Multimedia*, 15–24.
- ZHANG, L., SNAVELY, N., CURLESS, B., AND SEITZ, S. 2004. Spacetime faces: High resolution capture for modeling and animation. *Proc. SIGGRAPH*, 548–558.