

QuickCut: An Interactive Tool for Editing Narrated Video

Anh Truong[†], Floraine Berthouzoz[†], Wilmot Li[†], Maneesh Agrawala*

[†]Adobe Research

{truong, floraine, wilmotli}@adobe.com

*Stanford University

maneesh@cs.stanford.edu

(a) story outline

[Show example of dry roasted pork ribs]
[Show dry roasted pork ribs with group of people]
One of my favorite, super-easy things to make for large groups is dry roasted pork ribs. Every time I make these, people ask for the recipe, so here we go.

[Show raw back ribs on a baking tray]
Start by picking up some back ribs. I find this recipe works better for ribs that are not too meaty, so try to find ribs with a high bone to meat ratio.

[Trim fatty bits off pork ribs]
Trim off any excess fatty bits.

[Show salt and pepper or seasoning with salt and pepper]
and then season both sides with salt and black pepper.

[Seasoning with salt]
I typically use kosher salt, which is not too salty, so don't skimp! The salt gives the ribs a nice crust once they are cooked. Plus, pork ribs should taste salty!

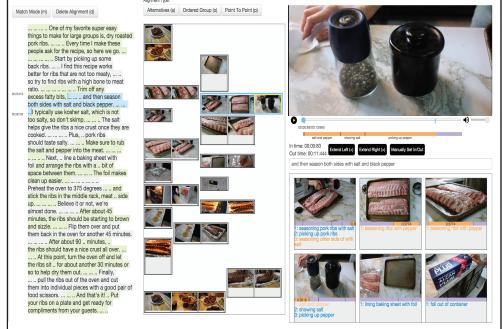


(b) capture raw video footage



(c) audio annotation

QuickCut



(d) edit with QuickCut



(e) final composition

Figure 1. Creating a narrated video starts with a pre-production phase in which authors write a story outline that includes a voiceover script (a). In the capture phase they record the narration as well as the raw video footage (b). With QuickCut, the next step is to review the raw footage and provide spoken audio annotations describing the semantically relevant actions and objects in each scene (c). The QuickCut interface (d) facilitates matching the story events in the narration with relevant segments of the raw footage and allows authors to specify high-level alignment constraints between them. QuickCut then uses a dynamic programming optimization to choose frame-level cut points between the segments to generate the final composition (e).

ABSTRACT

We present QuickCut, an interactive video editing tool designed to help authors efficiently edit narrated video. QuickCut takes an audio recording of the narration voiceover and a collection of raw video footage as input. Users then review the raw footage and provide spoken annotations describing the relevant actions and objects in the scene. QuickCut time-aligns a transcript of the annotations with the raw footage and a transcript of the narration to the voiceover. These aligned transcripts enable authors to quickly match story events in the narration with semantically relevant video segments and form alignment constraints between them. Given a set of such constraints, QuickCut applies dynamic programming optimization to choose frame-level cut points between the video segments while maintaining alignments with the narration and adhering to low-level film editing guidelines. We demonstrate QuickCut's effectiveness by using it to generate six (≤ 2 minutes) narrated videos. Each result required between 14 and 52 minutes of user time to edit (i.e. between 8 and 31 minutes for each minute of output video), which is far less than typical authoring times with existing video editing workflows.

Author Keywords

video editing; video annotation; video segmentation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

UIST 2016, October 16 - 19, 2016, Tokyo, Japan

Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-4189-9/16/10...\$15.00

<http://dx.doi.org/10.1145/2984511.2984569>

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces — Graphical user interfaces

INTRODUCTION

Narrated videos, such as documentaries, video essays, how-to tutorials, product pitches and scientific explainers are often comprised of a scripted voiceover and visuals that serve to illustrate the spoken narration. As more and more people seek information online, such videos have become a predominant form for communicating ideas, instructions and knowledge. For example, the New York Times publishes a diverse spectrum of video essays on its popular Times Video site, while a YouTube search for “how to video” returns over 200 million results, most of which are narrated tutorials. Yet, producing such video content requires skill and effort.

A common workflow for creating such narrated videos begins with a *pre-production* phase in which authors write a story outline (Figure 1a) that specifies the sequence of events in the story. The outline typically provides a script for the voiceover narration that describes the events and in some cases a description of the visuals that should accompany each event. In the *capture* phase, authors record the narration and film raw video footage for each event in the script (Figure 1b). Experienced authors often capture multiple versions of the raw video for each event (e.g., different takes, camera angles, etc.) to ensure they have enough *coverage* – high quality visual material to cover every part of the script. Although pre-production usually precedes capture, it is not uncommon for authors to start the capture phase before pre-production is complete.

In the *editing* phase, authors compose the recorded voiceover and raw footage into the final narrated video. Editing involves two main time-consuming challenges:

Match narration events with raw video segments. Authors usually start by identifying the events in the voiceover recording and roughly matching each of them with one or more semantically relevant segments of the raw video footage. Each such segment is a candidate visual for the corresponding narration event. This matching task is extremely tedious using standard video editing tools because users must navigate the audio/video recordings using a timeline-based interface. Finding the events in the voiceover and the relevant raw video segments usually requires scrubbing through and playing back the recordings multiple times. Because authors usually capture far more raw footage than they will use in the final composition, this matching process is typically the most time-consuming step in editing a narrated video.

Choose best video segments and frame-level cut points. To produce the final composition, authors must decide which alternative video segment to use at each moment in time as well as frame-level cut points and timing (start frame, end frame, length) for each chosen segment. In making these frame-level decisions, experienced authors try to balance multiple criteria based on the conventions and best practices of film editing (e.g., avoid blurry or shaky footage, avoid very short shots, include complete actions, avoid jump cuts between similar looking frames, etc.). Standard video editing interfaces force authors to inspect raw footage frame-by-frame to find cut points that satisfy the criteria.

Based on interviews with professional filmmakers and Internet forums for video professionals [16], producing each minute of narrated video with current video editing tools requires anywhere from 2-5 hours of editing.

In this paper, we present QuickCut, a video editing tool designed to help authors efficiently edit narrated videos (Figure 1d). It takes an audio recording of the voiceover narration and a collection of raw video footage as input. The QuickCut interface is based on two key ideas. First, it includes a set of transcript-based interaction tools to help authors find and match story events in the narration with semantically relevant video segments. Our system leverages audio annotations of the footage to support this semantic matching. Second, it provides a constraint-based framework that automates the process of choosing frame-level cut points for the video segments, while maintaining author-specified alignment constraints with the voiceover and adhering to low-level film editing guidelines.

Together these tools allow authors to focus on the overall story and content of their narrated video rather than working at the frame level. We demonstrate the effectiveness of QuickCut by generating 6 short narrated videos (≤ 2 minutes) with our system, including product pitches, cooking tutorials, and a professionally produced interview video promoting a research lab. Each video took between 14 and 52 minutes of user time to edit (i.e. between about 8 and 31 minutes for each minute of output video), which represents a substantial reduction in authoring effort over existing video editing workflows.

RELATED WORK

Most commercial video editing systems force users to work with a frame-based timeline. However, in recent years, researchers have developed higher-level tools for video editing. We describe the techniques most relevant to our work.

Interactive search and editing tools. Several existing systems present interactive tools to facilitate search and editing of video and audio in a specific domain. For example, researchers have developed techniques for aligning film scripts [14, 6] or source books [25, 23] to corresponding films, in order to facilitate text-based search and browsing of the visual content. Chi et al. [5] help users generate instructional videos for physical demonstrations by segmenting the steps in the demonstration via user-assisted video analysis techniques. Berthouzoz et al. [3] focus on tools for placing cuts and transitions in talking-head style interview video. Rubin et al. [18] develop tools for editing and scoring audio podcasts. Pavel et al. [15] present techniques for editing informational lecture videos into structured summaries that support browsing and skimming. Like Berthouzoz et al., Rubin et al. and Pavel et al., our approach leverages time-aligned transcripts to facilitate editing operations. However, unlike previous work, QuickCut uses such transcripts to facilitate matching and aligning raw video footage to a voiceover narration, which is a critical and time-consuming step in authoring narrated videos.

Automated video editing and camera selection. Many systems provide fully automated approaches to editing video footage. Often, these approaches exploit characteristics of a specific domain. Ranjan et al. [17] focus on videos of group meetings and use television production principles to capture and edit the footage. Heck et al. [8] edit classroom lecture videos by switching between the original footage and automatically generated 2D zooms and pans. Shin et al. [20] edit blackboard-style lectures into a readable representation combining visuals with readable transcripts. Zsombori et al. [26] rely on media annotations to automatically generate a video of an event, so as to highlight a particular person during the event. Lu and Grauman [12] edit egocentric footage by segmenting the raw footage into events and constructing chains of events that follow one another. Jain et al. [9] re-edit widescreen video to fit smaller aspect ratio screens via pans, cuts and zooms based on gaze-data. Arev et al. [1] automatically edit multiple first-person view videos taken of the same scene. They reconstruct the 3D environment and use cinematographic guidelines to select which video to show at each moment in time. Our approach is similar to several of these approaches [1, 9, 17] as we use some of the same film editing guidelines (e.g. include complete actions, avoid jump-cuts between similar looking frames) to make low-level editing decisions. However, our optimization also takes into account user-specified alignment constraints between the voiceover and footage.

Quality of raw footage. Researchers have proposed many different techniques [2, 22, 21] to assess the quality of raw video footage based on low-level frame properties such as lighting, blurriness, etc. Some of these approaches [22, 21] use this quality index to edit multiple video feeds. Our algorithm also uses low-level frame attributes to measure the quality of

each video segment. However, our algorithm also considers higher-level annotation-based semantic segmentations of the raw video as well as constraints between these segments and the voiceover to generate narrated videos.

QUICKCUT WORKFLOW

In developing QuickCut, we initially conducted informal interviews with seven professional filmmakers to better understand how they edit narrated video using existing editing tools. They often start by writing the story outline with a script and then *logging* their raw footage – watching each raw video clip and writing notes about the clip’s content and aesthetics (camera angles, lighting, etc.) with associated timestamps for each noteworthy segment. These editors consistently reported that logging takes them between 1.5 and 3 times the length of the raw footage and is extremely tedious. They typically capture 15-90 minutes of raw footage for each minute of edited video. Note that some filmmakers reverse these two steps and log their footage before writing the story outline. Following these two steps, filmmakers cut and assemble the raw video into an edited result. In general, they estimated that logging and editing together require 2-5 hours to produce each minute of edited video. However, one filmmaker reported needing multiple 8 hour days to edit a single minute of particularly complex video. These estimates are in-line with those reported in online video editing forums [16].

Our QuickCut interface is designed to help authors efficiently log raw footage and edit narrated video. It takes an audio recording of the voiceover narration and a collection of raw video footage as input. To demonstrate how QuickCut works in practice, we describe the workflow an author might use to create a narrated video (Figure 2). Our example is an instructional cooking video we created to explain how to prepare dry roasted pork ribs.

Annotating Inputs with Time-Aligned Transcripts

QuickCut facilitates navigation, segmentation and alignment of the input voiceover and raw video footage using time-aligned text transcripts. Authors often produce a complete transcript of the narration as they write the story outline and QuickCut accepts such transcripts as part of the voiceover input. However, if the recorded voiceover differs from the script in the story outline, we use the crowdsourcing service `rev.com` to obtain a verbatim transcript for the voiceover audio file at a cost of \$1 per minute. QuickCut then time-aligns the transcript to the voiceover audio using the phoneme mapping technique of Rubin et al. [18]. For the pork ribs example, we used the script we created as part of the story outline.

QuickCut supports two different methods for annotating (or logging) captured videos. Audio annotations can be recorded directly into the audio track of the footage during filming. Alternatively, filmmakers can record annotations as they review footage by describing what they see as the raw video plays. Our post-capture annotation tool lets filmmakers scrub forwards and backwards through the raw footage as necessary and maintains a map from each audio annotation to the timeline of the raw footage.

The annotations should describe the actions, objects and other relevant content in the footage to serve as a descriptive log of the raw material. For the pork ribs example we annotated the footage post-capture, making sure to describe the important actions and objects we saw (e.g., “season ribs with pepper”). Given such audio annotations for the raw video, we use `rev.com` to obtain the corresponding text transcripts and QuickCut time-aligns the text to the raw video, again using the approach of Rubin et al. QuickCut also groups together sequences of words that occur less than .25 seconds apart and treats each such group as a single annotation. Finally, QuickCut infers the subsegment of the raw video footage that corresponds to each annotation (we call these *annotation segments*) and associates the annotation text with the segment.

We have experimented with using Google’s free Web Speech API [24] for automatic speech recognition (ASR) as an alternative to `rev.com` for transcribing the annotations. Our experience is that ASR works best when the annotator enunciates clearly and speaks coherent phrases, at a relatively slow pace, and avoids specialized jargon. In practice, it is difficult to consistently meet all of these conditions, and in many cases the ASR results contain excessive errors.

Using the QuickCut Interface

As shown in Figure 2, the QuickCut interface consists of three main components: a *Transcript View* (left) lets authors navigate and select segments of the voiceover that usually correspond to story events; a *Footage Selector* (right) lets authors find relevant segments of the raw video footage for each event; and a *Structured Timeline* (middle) lets authors specify *alignment constraints* between the voiceover and the raw video segments.

Transcript View

The left pane of the interface shows the text transcript of the voiceover in a scrollable column. The text is aligned to the recorded narration so that selecting transcript text also selects the corresponding segment of the voiceover audio. The ellipses in the transcript indicate pauses in the narration, and they allow authors to make finer grained selections within the silences between spoken phrases. By allowing the user to navigate the narration by reading text rather than scrubbing through a timeline, the Transcript View makes it much easier to identify and select the story events as defined in the story outline. In our pork ribs example (Figure 2), we have selected the narration event explaining that the cook must “then season both sides with salt and black pepper”.

Footage Selector

After selecting a story event in the narration, the next step is to find semantically relevant raw video footage. The right pane of the interface lets authors browse the collection of raw footage with a representative image for each raw clip, a list of the transcribed audio annotations associated with the clip, and a small timeline that visualizes the video segments associated with the annotations. A playback monitor at the top of the right pane supports scrubbing and playback of selected clips. Clicking on a representative image plays the entire clip from the beginning, while clicking on an audio annotation plays just the corresponding segment.

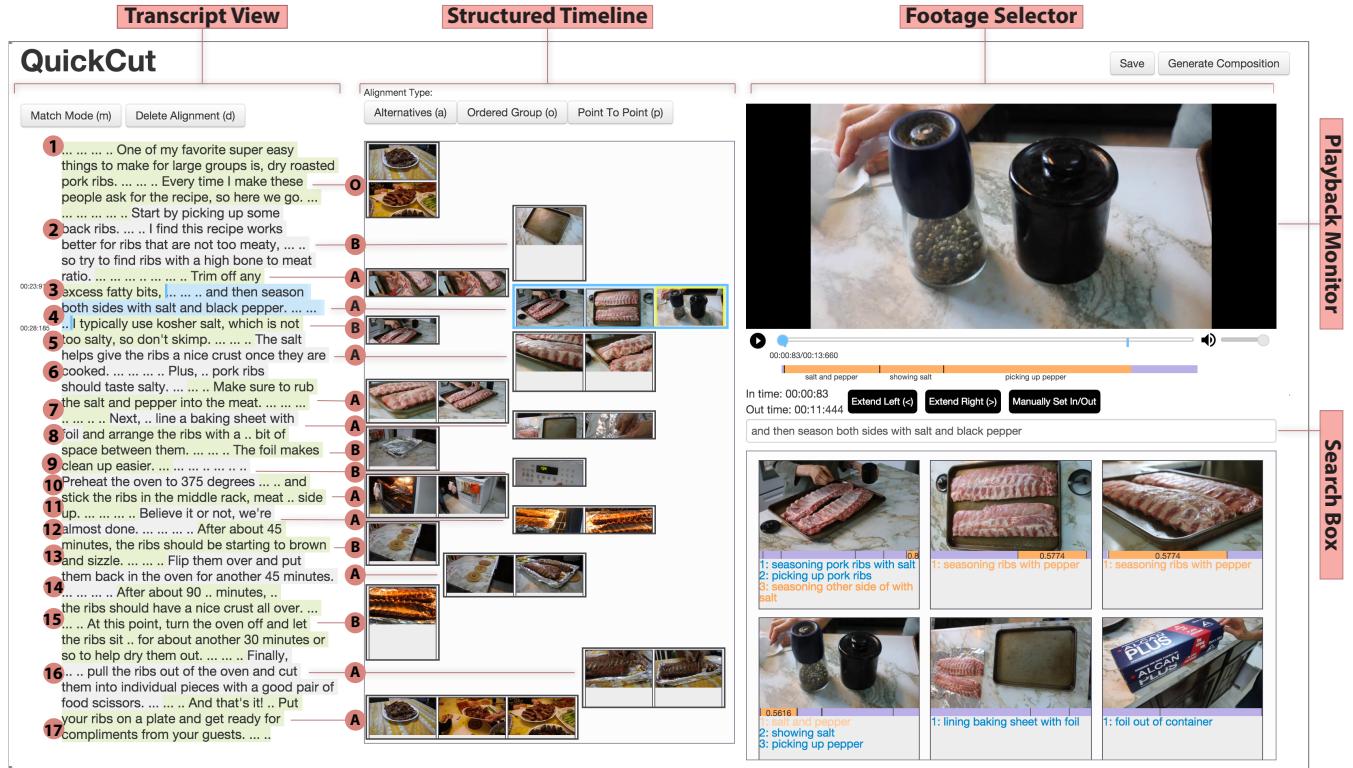


Figure 2. Making the pork ribs video with our QuickCut interface. The Transcript View (left) provides a time-aligned text transcript of the narration and lets us find and select important events in our story. We identified 17 events in this narration (alternating green and grey highlights) and used QuickCut to add alignment constraints – basic (B), alternatives (A), ordered-grouping (O) – to each one as follows. We select narration text corresponding to an event (highlighted in blue) in the Transcript View and it appears in the Footage Selector (right) search box. QuickCut then finds the raw video clips containing the top matching annotation segments and displays them below the search box (matching segments and annotation text highlighted in orange, non-matching annotation segments in blue). Here we select the “salt and pepper” segment (Footage Selector bottom left) and preview it in the Playback Monitor. We then extend the length of the selected segment using the right arrow key to include the adjacent segments “showing salt” and “picking up pepper”. Finally, we drag the clip into the Structured Timeline (middle) and add it to an alternatives constraint (narration event 3).

To help users find relevant raw video footage, the footage selector provides a text-based search interface. Users can type query text directly into a search box or select text from the Transcript View, which automatically populates the search box, as in our pork ribs example (Figure 2). Searching with selected narration text is most useful when the voiceover directly references the specific actions or objects shown in the footage. But in some cases the narration may refer to the actions in the video using different terms (either more general or more specific) than the audio annotations on the raw footage. For example, the pork ribs narration includes the phrase “stick the ribs in the middle rack”, while the audio annotation describing the relevant action is “putting ribs in oven”. In this case the narration text is very different from the annotation, but typing the search query “ribs in oven” in the search box allows us to quickly find the relevant footage.

QuickCut sorts the raw video clips based on how well the annotation text of the segments within the clip match the query text, and highlights the matching segments in orange. If the same clip contains more than one top-matching segment, it will appear multiple times in the footage selector but with different segments highlighted. Users preview the top matching segments in the playback monitor to decide which segments are most suitable. Users can also extend the length of a segment using the left/right arrow keys to add the adjacent

segments to the selection and if users require finer control, they can directly specify in- and out-times for the segment via the playback monitor timeline.

In our pork ribs example, QuickCut finds several annotation segments that match the transcript text we selected earlier. We then choose the clip containing the annotation text “salt and pepper”. Playing back that segment in the monitor we decide that it could be a little longer, so we extend it to include the adjacent “showing salt” and “picking up pepper” annotations.

Structured Timeline

As the author finds relevant video segments for each story event in the narration, he can drag the segments into the *Structured Timeline* to form alignment constraints between the selected narration event and the raw video footage. As in traditional video timelines, individual video segments are represented as rectangles. However, in our Structured Timeline, time advances from top to bottom (rather than left to right) and the height of the rectangle represents the duration of the video. As the author forms alignment constraints, QuickCut highlights the corresponding narration events with alternating green and grey backgrounds in the Transcript View. Since height represents time, these highlights also serve to visualize the duration of each event.

Associating a single raw video segment with a story event generates a ***basic alignment constraint*** and requires that some portion of the video segment should fill the duration of the corresponding narration event. This constraint is useful for aligning footage of a single action or subject (e.g., b-roll of a factory assembly line, a held shot of ingredients for a recipe) with a story event that refers to that action or subject. Our pork ribs example (Figure 2) shows six basic alignment constraints (labeled B) including one for the second narration event.

When multiple raw video segments are associated with a single story event, authors may generate an ***alternatives constraint*** which requires that a portion of exactly one of the segments fills the duration of the corresponding story event. The Structured Timeline displays all of the alternatives side-by-side in a horizontal row to indicate that any one of them could appear in the final composition. Authors can apply this constraint when portions of several different video segments could potentially illustrate the story event. Our pork ribs example contains 10 such alternatives constraints (labeled A), including one for the third narration event.

An ***ordered grouping constraint*** is another type of multi-segment constraint, which requires that some portion of all of the selected video segments must appear in their specified order over the duration of the story event. In this case the Structured Timeline displays all of the segments in a single column that lines up with the top and bottom of the corresponding story event in the Transcript View. Ordered grouping constraints are useful when several different video segments are all relevant to a longer piece of narration. By applying this constraint, the author ensures that all of the segments appear in the final composition and that they cover the duration of the story event. Our pork ribs example contains 1 such ordered grouping constraint (labeled O) for the first narration event.

The author can precisely align a word in the voiceover with a frame in a raw video segment using a ***point-to-point constraint***. To create such a constraint, the author must click a word in the Transcript View, then scrub to a point in a selected raw video clip using the playback monitor, and link the two points together. Authors can also select the video frame first and then choose the word in the transcript. The constraint requires that the corresponding word and video frame appear at the same time in the final composition and is useful for ensuring that on-screen visuals tightly match the narration. For example, many videos contain segments of a narrator speaking parts of the voiceover on-screen (e.g., talking head video). To ensure that the lip motions in the video match the voiceover audio, when the user selects a narration event and adds a video clip with matching audio to the timeline, QuickCut automatically generates a point-to-point constraint between them.

Whenever the author applies a constraint, QuickCut checks whether there is enough raw footage in the associated video segments to span the duration of the story event. Specifically, for a basic constraint and the alternatives constraint, QuickCut checks whether each possible video segment is long enough to cover the story event. For the ordered grouping constraint, it checks whether the combined length of the segments is enough to cover the story event. If a constraint fails these

length checks, QuickCut automatically extends all of the segments contained in the constraint to include adjacent segments up to the length of the entire video clip in order to attain the necessary length. It then marks the segments the user initially specified as *primary* and all other segments as *secondary*. If the constraint still fails the length requirements after the automatic extensions, we notify the author. This real-time feedback about invalid constraints can help authors determine which combinations of footage are appropriate for any given narration event. Note that while the secondary segments can include irrelevant material, they ensure that QuickCut has the raw footage it needs to cover each story event with some video. During the optimization to generate the final composition, QuickCut avoids using the secondary segments and only includes them when necessary.

Once the author has aligned footage to every portion of the narration, he clicks the *Generate Composition* button (top left corner of interface) to automatically generate an edit based on the specified alignment constraints. QuickCut can also output a standard Edit Decision List (EDL) file that lists the in- and out-time, and filename for each audio and video segment in the video project. Many video editing programs (e.g., Premiere, Final Cut) can load this EDL representation and users can further post-process the video (e.g., stabilize, color-correct) as necessary with effects QuickCut does not support.

ALGORITHMIC METHODS

Our QuickCut interface relies on automated algorithms for segmenting raw video footage into relevant actions, matching narration events to semantically related raw video segments, and choosing aesthetically pleasing cut points that respect the user-specified alignment constraints.

Segmenting Raw Video Footage

To help users find suitable footage for each narration event, QuickCut automatically segments the raw video footage into actions that are relevant to the story. Our approach combines two types of information. Since actions correspond to motion in the video, we first perform a motion-based segmentation that distinguishes between *kinetic segments*, which contain continuous motion, and *static segments*, where nothing moves. However, such motion segmentation does not capture the content-based relevance of the footage. Therefore, we use the audio annotations, which describe the content of the raw footage, to further refine the segmentation and produce a set of semantically relevant *annotation segments*.

Computing motion-based segments

Given a video clip, we start by identifying all frames that contain some motion. We consider frame f to contain motion if the luminance of at least α percent of its pixels have changed by greater than β from the previous frame. Otherwise, we consider f to be a still frame. Each contiguous run of motion frames represents a kinetic segment, and each run of still frames represents a static segment. To prevent over-segmentation due to brief pauses in a kinetic segment or subtle background movement in a still segment, we merge any segment that is less than γ seconds long with its two adjacent

segments. We find that $\alpha = 0.05\%$, $\beta = 0.12$ (where luminance ranges from 0 to 1) and $\gamma = 0.2$ seconds work well in practice and use these values for all of our results. Using this approach we typically obtain relatively long kinetic segments corresponding to one or more overlapping actions in the scene, with short static segments in between (Figure 3a).

Computing annotation segments

While motion-based segments suggest where actions start and end, each individual kinetic segment may not correspond exactly to a single semantically relevant action for the story. On the other hand, audio annotations describe semantically relevant actions, but the timing of each annotation only roughly indicates where the action starts and ends. Thus, to compute annotation segments, we refine the motion-based segmentation based on the timing of the audio annotations.

We have observed that audio annotations often start just after the corresponding action begins and end before the action finishes (e.g., annotations a_2 and a_3 in Figure 3b). That is, most audio annotations start and end within kinetic motion segments but may contain static segments within them. In such cases, where both endpoints of an audio annotation lie within kinetic motion segments, we start by extending the annotation to the nearest encompassing kinetic segment boundaries. If, however, an input audio annotation starts (or ends) within a static motion segment, we leave the starting (or ending) annotation endpoint unchanged. For example, in Figure 3c, we extend the start point of a_1 back to the starting point of kinetic motion segment m_2 but leave the end point of a_1 unchanged since it lies within a static motion segment.

These rules produce useful segments for most input audio annotations. However, as we observed earlier, our motion-based segmentation algorithm may merge several semantically distinct actions into the same kinetic segment. Such undersegmentation can create two types of problems for our extended annotation segments:

- **Overlapping annotations.** If the end point of input audio annotation a_i lies in the same kinetic segment as the start point of the next input annotation a_{i+1} , their corresponding extended audio annotation segments will overlap even though each one describes semantically different content.

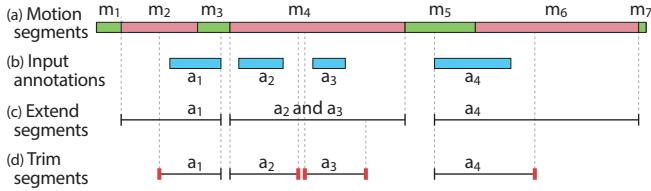


Figure 3. For each raw video clip we compute motion segments based on frame differencing to split the clip into static (green) and kinetic (pink) segments (a). To compute annotations segments we start with the grouped input audio annotations a_i (b) and initially extend each endpoint that lies within a kinetic motion segment. (c) We then trim back overlapping adjacent annotation segments (e.g., a_2 and a_3) as well as overly extended segments (e.g., start of a_1 and end of a_4) by finding boundary frames f_{bnd} (red segment ends) based on frame similarity to the origin annotation endpoints (d).

- **Overly extended annotations.** If the start frame a_i^{in} of input audio annotation a_i lies in the middle of a long kinetic segment m_j , extending the start frame back to the start frame m_j^{in} may introduce irrelevant content at the beginning of the extended annotation segment. The symmetric case where we extend the end frame of the input annotation can also introduce irrelevant content to the annotation.

We mitigate both of these issues in a final trimming step where we pull the extended endpoints of an input annotation back towards their original positions (Figure 3d). Specifically, in the overlapping annotations case, we examine the frames in order between the end frame a_i^{out} of the input annotation a_i and the start frame a_{i+1}^{in} of input annotation a_{i+1} , to find the first frame f_{bnd} that is more similar to a_{i+1}^{in} than a_i^{out} . We then use f_{bnd} as the new end frame a_i^{out} for annotation segment a_i . We similarly consider the frames between the input annotation segments in reverse order to find the frame f_{bnd} that is more similar to a_i^{out} than to a_{i+1}^{in} and set it as the new starting frame for a_{i+1} . In the case of overly extended annotations, we similarly find the boundary frame f_{bnd} between the start frame a_i^{in} of the input annotation and the start frame of the encompassing motion segment m_j^{in} and then set it as the new start frame for a_i . We use the same approach to handle the symmetric case with the end frame of the annotation. We use the phase correlation method [19] to measure frame similarity in this trimming step because it is resilient to small frame motions and image noise.

Matching Relevant Footage

As described in the Workflow Section, the Footage Selector allows users to search for relevant raw footage with text queries. For each annotation segment, we compare the query text to the annotation text using a *term frequency inverse document frequency (TF-IDF)* similarity score [13] and sort the segments in the interface from most to least similar. TF-IDF is commonly used to compare documents in a corpus based on the importance of the terms to each document. In our setting, we treat each annotation as a document and the collection of annotations as the corpus. We then represent each annotation as a TF-IDF vector in which each element is the frequency of a term (term frequency) in that annotation, divided by the frequency of the same term in the annotation corpus as a whole (document frequency). We also build a TF-IDF vector for the query, computing term frequency as the frequency of the term within the query text and the document frequency as the frequency of the term within the annotation corpus. We then compute the similarity score between the query and each annotation as the cosine distance between their TF-IDF vectors.

TF-IDF favors terms that occur frequently within a small number of annotations while penalizing terms that appear frequently in many annotations. However, common stop words (e.g., ‘the’, ‘and’, ‘a’, etc.) may appear with high-frequency in lots of annotations and can skew the TF-IDF similarity matching. Therefore, we exclude them when calculating the TF-IDF vectors using the list of stopwords from the Natural Language Toolkit (NLTK) [4]. In addition, to avoid double counting words used multiple times with different endings, we

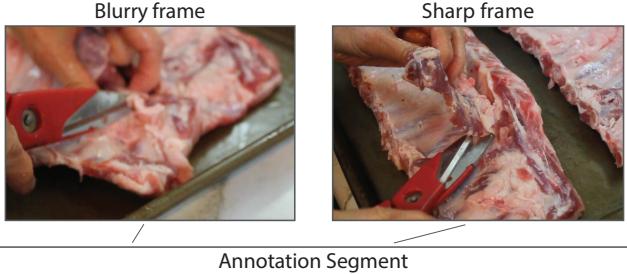


Figure 4. Our blur term penalizes the use of blurry footage from an annotated video segment. The blurry frame on the left incurs a relatively high penalty, while the sharp frame on the right incurs no cost as its blur level is below our penalty threshold.

use NLTK to first derive the part of speech for each term and then lemmatize them (remove the different endings).

Placing Aesthetically Pleasing Cuts

Given a set of alignment constraints that cover all the narration events in the voiceover, QuickCut automatically generates a final composition. We formulate the problem as a discrete optimization. We define a valid composition as a set of output video frames that originate from the annotation segments in the structured timeline and satisfy all the user-specified alignment constraints. More specifically, we express a composition as a sequence of labels $L_n = (l_1, \dots, l_n)$, one for each output frame, where a label l is a tuple (s, t) that specifies an annotation segment s and frame index t within the video clip containing s . Note that every label l uniquely identifies a frame of video footage. Each continuous run of labels that reference the same segment represents a separate shot, and the boundaries between shots are where cuts occur. We introduce a *labeling cost* $E(L_n)$ that measures how well the composition defined by labeling L_n adheres to standard film editing guidelines. The goal of the optimization is to find the labeling that minimizes E .

Labeling Cost

We define the cost for a labeling L_n as

$$E(L_n) = \sum_{i=1}^n \mathbf{F}(l_i) + \sum_{i=1}^{n-1} \mathbf{T}(l_i, l_{i+1}) \quad (1)$$

where the *frame cost* $\mathbf{F}(l_i)$ encodes the quality of the frame associated with l_i and the *transition cost* $\mathbf{T}(l_i, l_{i+1})$ encodes the quality of the transition between the frames associated with l_i and l_{i+1} . The overall labeling cost simply sums all the individual frame and transition costs for the composition.

Our frame and transition costs encode some of the conventions and best practices of film editing. In formulating these costs, our strategy is to prevent obvious artifacts without trying to finely differentiate between frames or transitions that are all of “good enough” quality. Thus, each of the terms described below includes a threshold where the cost goes to zero, indicating the frames or transitions are good enough. When the cost is greater than zero, it indicates the strength of the artifact present in the frame or transition.

Frame Cost

Our frame cost consists of three terms that operate on raw

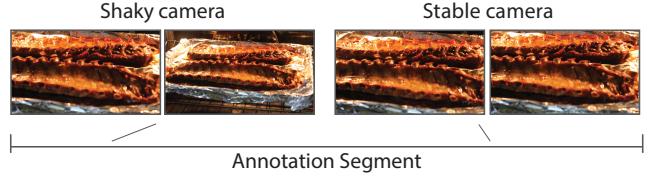


Figure 5. Our camera shake term penalizes the use of footage containing strong camera motion from an annotated video segment. The shaky footage on the left includes zooming and translational motion which incurs a high penalty, while the stable footage on the right incurs no cost as the magnitude of camera motion is below our penalty threshold.

video footage that has been pre-scaled to a width of 500 pixels. The first term penalizes the use of blurry footage:

$$\mathbf{F}_{\text{blur}}(l_i) = \begin{cases} 1/(\sigma(\mathbb{G}_i) + \epsilon), & \text{if } \sigma(\mathbb{G}_i) \leq 2.3 \times 10^{-4} \\ 0, & \text{otherwise} \end{cases}$$

where \mathbb{G}_i is the edge image obtained by convolving the luminance image of the frame associated with l_i with a Laplacian kernel, and $\sigma(\mathbb{G}_i)$ is the sum of squared pixel values for \mathbb{G}_i normalized by the frame size. The threshold for incurring the penalty (2.3×10^{-4}), which we found empirically, defines what we consider to be a sharp, in-focus frame. For example, Figure 4 shows blurry and sharp frames from the same annotation segment. Note that our blur cost does not tend to penalize frames that are generally sharp but contain moving objects since the motion blur in such frames typically involves a small number of pixels that fall below the threshold.

The second term penalizes the use of excessively shaky footage that can sometimes appear in raw, hand-held video footage:

$$\mathbf{F}_{\text{shake}}(l_i) = \begin{cases} \|\tau(\mathbb{H}_i)\|, & \text{if } \|\tau(\mathbb{H}_i)\| > 3 \\ 0, & \text{otherwise} \end{cases}$$

where \mathbb{H}_i is the homography that warps the frame associated with l_i to the next frame in the annotation segment and $\tau(\mathbb{H}_i)$ is the translational motion due to the homography. Following the approach of Joshi et al. [10] we compute \mathbb{H}_i using RANSAC [7] on SIFT feature points [11]. To compute $\tau(\mathbb{H}_i)$, we apply \mathbb{H}_i to the center point of the frame associated with l_i and then calculate how far it translates the point. This translation is an estimate of the motion of the camera look-vector and when its magnitude $\|\tau(\mathbb{H}_i)\|$, is large we assume the motion is due to camera shake. In this case the threshold for incurring the shake penalty (3 pixels) defines what we consider to be excessively shaky footage and we determined it empirically. Figure 5 shows a pair of relatively shaky frames versus relatively stable frames in the same annotation segment.

The third frame cost term encourages the use of footage that corresponds to the primary action in each annotation segment. Recall from the Workflow Section that QuickCut automatically extends any annotation segment in the Structured Timeline that is too short to cover the corresponding narration event. In these cases, the annotation segment includes the user-specified primary segments along with the adjacent secondary segments. To penalize the use of secondary versus primary segment frames, we define the following cost:

$$\mathbf{F}_{\text{sec}}(l_i) = \begin{cases} 1/d_{\text{evt}}, & \text{if } l_i \text{ refers to a secondary segment frame} \\ 0, & \text{otherwise} \end{cases}$$



Figure 6. Our transition cost penalizes jump cuts between similar looking frames using Euclidean distance as the measure of similarity. When the frames across a transition are similar in appearance (top) the transition incurs a high cost and when the frames are different they incur no cost as the similarity is below our cost threshold.

where d_{dev} is the length in frames of the narration event for the output frame under consideration. In other words, we impose a constant penalty for using a secondary segment frame, which encourages shots to include as much as possible of the primary action in each annotation segment.

The total frame cost is the sum of the blur, shake and secondary segment terms:

$$\mathbf{F}(l_i) = \mathbf{F}_{\text{blur}}(l_i) + 0.01 \cdot \mathbf{F}_{\text{shake}}(l_i) + \mathbf{F}_{\text{sec}}(l_i)$$

where we empirically determined the 0.01 weighting factor for the shake term to scale down its contribution and balance the effects of all three terms.

Transition Cost

To avoid jump cuts, we penalize transitions between very similar frames with a simple scoring function:

$$\mathbf{T}(l_i, l_{i+1}) \begin{cases} 1 - \delta(l_i, l_j) & \text{if } \delta(l_i, l_j) < 0.2, \\ 0 & \text{otherwise} \end{cases}$$

where $\delta(l_i, l_j)$ is the euclidean distance in luminance space between the frames associated with l_i and l_j , normalized by the maximum possible euclidean distance for the two frames. As with the blur cost, we determined the threshold of 0.2 empirically; above this threshold, frames tend to look different enough to avoid the appearance of a jump cut. The top row of Figure 6 shows a jump cut that results in a large transition cost; the bottom row shows the cut point that our optimization chooses when taking into account the transition cost.

Dynamic Programming

An important characteristic of our labeling score is that it satisfies the optimal substructure property. That is, the score for the optimal labeling L_n^* can be expressed as

$$\mathbf{E}(L_n^*) = \mathbf{E}(L_{n-1}^*[l_{n-1}]) + \mathbf{F}(l_n) + \mathbf{T}(l_{n-1}, l_n) \quad (2)$$

where $L_{n-1}^*[l_{n-1}]$ is the optimal labeling for the first $n-1$ output frames that ends with the label l_{n-1} (i.e., the second-to-last label of L_n^*). In other words, L_n^* can be decomposed into a sequence of optimal partial solutions $(L_1^*[l_1], \dots, L_n^*[l_n])$. This property allows us to find a globally optimal labeling via dynamic programming.

To find the optimal labeling, we incrementally construct a table of optimal partial solutions of increasing length until we have computed the optimal full solution. At every solution length k (where $1 \leq k \leq n$), we must compute a partial solution $L_k^*[l_k]$ for every possible labeling l_k of the k -th output frame. To determine $L_k^*[l_k]$, we consider appending l_k to all optimal partial solutions of length $k-1$ and choose the option that minimizes Equation 2. Note that this choice implicitly specifies a label-to-label transition from frame $k-1$ to frame k . Once we've computed all possible solutions for $k=n$, we simply pick the one with the lowest score, which represents the optimal labeling for the entire sequence of n frames.

To ensure that this procedure only considers valid labelings that satisfy the user-specified alignment constraints, we impose several restrictions on how the table is constructed:

Label restrictions. Every alignment constraint associates a set of annotation segments S with a narration event. Thus, for all output frames that correspond to that narration event, we restrict the possible labelings to $(s, t) : s \in S$. In addition, an ordered grouping constraint specifies that the event must start with the first and end with the last segment in the group, which further restricts the allowable labels for the first and last output frames in the event. Finally, a point-to-point constraint fully specifies both the segment and frame indices for the corresponding output frames.

Transition restrictions. Every shot in the final composition must play forwards in time, frame-by-frame. Moreover, an alternatives constraint requires that consecutive labels reference the same segment, while an ordered grouping constraint requires consecutive labels to either reference the same segment or transition to the next segment, as defined by the segment ordering. Finally, to prevent extremely short shots, we require each run of labels that reference the same segment to have some minimum length. Note that this restriction does violate the optimal substructure property of our labeling cost since the last shot in an optimal partial solution $L_k^*[l_k]$ may end up being too short to use in a subsequent solution. However, we have not found this potential lack of optimality to be an impediment to generating good compositions.

We enforce label restrictions by only creating the allowed entries in the partial solutions table for each solution length k . Transition restrictions limit the number of previous partial solutions that we consider when minimizing Equation 2 to compute each new partial solution. Thus, while the worst case asymptotic complexity of our optimization is $O(NM^2)$ where N is the number of output frames and M is the (very large) number of possible labels for all the annotated raw footage, the label and transition restrictions make the problem far more tractable in practice. In particular, our running time is primarily determined by the presence of ordered grouping constraints, which have the fewest transition restrictions. If there are ordered grouping constraints, the optimization is essentially quadratic in the number of possible labels in the largest ordered grouping constraint, which typically dominates the linear dependence on N . In contrast, while an alternatives constraint does compute partial solutions for each of the candidate annotation segments, it does not allow transitions between



Figure 7. We used QuickCut to produce narrated videos for five datasets obtained from students working on product pitches for a class (*Campus Tour*, *Travel Teddy*, *Umami*) and our own capture sessions (*Mexican Bagels*, *Pork Ribs*). The sixth dataset is an interview of a new member of a research lab (*Duygu*) and was captured and edited by a professional filmmaker using QuickCut. See supplemental materials for all six of the narrated videos generated using QuickCut.

| Dataset | Footage | | Annot Time | Structured Timeline | | | | Constraints | | | | Edit Time | Composition | | | User Time Per Min |
|----------------|---------|-----|------------|---------------------|-----|-----|-------|-------------|-----|-----|-----|-----------|-------------|-------|-------|-------------------|
| | Clips | Dur | | Events | Seg | Ext | Dur | Basic | Alt | Ord | P2P | | Shots | Dur | Opt | |
| Campus Tour | 21 | 5m | 7m | 10 | 16 | 6 | 2.7m | 4 | 4 | 2 | 0 | 7m | 12 | .76m | .62m | 8.26m |
| Mexican Bagels | 80 | 19m | 21m | 14 | 20 | 7 | 3.03m | 8 | 5 | 1 | 0 | 10m | 15 | 1.37m | .1m | 22.68m |
| Pork Ribs | 41 | 8m | 15m | 17 | 27 | 16 | 3.18m | 5 | 11 | 1 | 0 | 7m | 18 | 1.4m | 1.57m | 15.71m |
| Travel Teddy | 45 | 10m | 20m | 16 | 23 | 0 | 2.93m | 12 | 3 | 1 | 0 | 12m | 18 | 1.07m | .75m | 30m |
| Umami | 23 | 8m | 10m | 11 | 21 | 4 | 1.82m | 3 | 6 | 2 | 0 | 13m | 13 | .75m | .23m | 30.6m |
| Duygu | 18 | 42m | 26m | 21 | 37 | 2 | 57m | 3 | 6 | 3 | 9 | 26m | 27 | 1.7m | 4.77m | 30.59m |

Table 1. For each dataset we report the following statistics. **Footage:** number of individual clips (Clips), and total clip duration (Dur). **Annot Time** is the amount of user time spent adding annotations. **Structured Timeline:** number of narration events (Events), annotation segments added to the timeline (Seg), automatically extended annotation segments (Ext), and the total duration of annotation segments in the timeline (Dur). **Constraints:** number of basic (Basic), alternatives (Alt), ordered grouping (Ord), and point-to-point (P2P) constraints used to align footage to narration. **Edit Time** is the amount of user time spent adding constraints to the QuickCut structured timeline. **Composition:** number of shots (Shots), total duration of the composition (Dur), and time required to optimize the placement of cuts to generate the final composition (Opt). **User Time Per Minute** is the total user time required for each minute of video in the final composition – $(\text{Annot Time} + \text{Edit Time}) / \text{Composition (Dur)}$.

any of the segments. Thus, such constraints tend to have a smaller impact on the optimization time compared to ordered grouping constraints. As reported in the Opt column of Table 1, optimizing cut points required between 0.1 and 4.77 minutes for all of our results.

RESULTS

We used QuickCut to produce compositions for the first five datasets shown in Figure 7 and listed in Table 1. We obtained the *Campus Tour*, *Travel Teddy* and *Umami* datasets from students who captured the footage to generate product pitches for an undergraduate course project. We captured the *Mexican Bagels* and *Pork Ribs* footage ourselves. The sixth dataset, *Duygu*, comes from an interview promoting a research lab, and we asked the professional filmmaker who captured the footage to edit it together using QuickCut. The *Travel Teddy* and *Campus Tour* footage were captured before pre-production was completed. Conversely, pre-production for *Umami*, *Pork Ribs*, *Duygu* and *Mexican Bagels* occurred before the footage was captured. The raw input materials for all of these datasets except *Duygu* are available at the project website: <http://graphics.stanford.edu/projects/quickcut/>.

Given the captured footage, the active human time required to create each composition is the time spent annotating the footage (Annot Time column of Table 1) plus the time spent finding and aligning relevant footage to the narration events using the QuickCut interface (Edit Time column). Adding these two columns together, all of our results required between 14 and 52 minutes to create. The total user time required to produce each minute of output video varies between about 8 and 31 min (User Time Per Min column), which is significantly less than the 2-5 hours professional filmmakers suggest they require to produce 1 minute of edited video using existing tools and workflows (see Workflow Section).

To add audio annotations, we (or the professional filmmaker in the case of *Duygu*) played back the raw videos and described what we saw. For *Pork Ribs* and *Travel Teddy*, we had to watch the entire set of video clips first to understand what was happening and then replay them to add the annotations. The professional filmmaker knew the raw *Duygu* footage well and also scrubbed through the footage to quickly get to specific sections for annotation. The time she took to annotate the footage was much less than the duration of the raw footage itself. For most of our datasets, the time required to annotate the footage was only slightly longer than the total duration of the footage itself, as reported in the Dur and Annot columns. Matching the semantically relevant video segments to narration events and setting up alignment constraints between them required between 7 and 26 minutes of user interaction time in the QuickCut interface (Edit Time column). Most of this time was spent reviewing footage to determine which of the top-matching annotation segments to associate with each narration event. To search for videos, we typically used all of the text in the narration event as the query. In a few situations, we used custom search terms to find videos that we deemed a better fit for the story. The professional filmmaker similarly used a combination of narration text and search terms based on her annotations. When deciding which annotation segments to add to the Structured Timeline, we and the professional filmmaker always chose from amongst the top 5 matching segments in the Footage Selector. The professional filmmaker was still learning the QuickCut interface as she used it to create the *Duygu* video and we expect that the 26 minutes she spent editing would reduce further with additional learning time.

As indicated in the “Constraints” columns of the table, all of the videos primarily used alternatives constraints to associate multiple candidate annotation segments to individual narration events. Ordered grouping constraints were most useful

for illustrating some longer events. The only dataset that included point-to-point constraints was Duygu, which contains an onscreen speaker (e.g. talking head) whose voice matches the narration. QuickCut automatically inserted the necessary point-to-point constraints into the structured timeline whenever the professional filmmaker added such a raw talking head segment to the structured timeline (see description of point-to-point constraint in Workflow Section).

To evaluate the automatic segmentation of the raw video, we consider the variation in the differences between the start and end times of users' audio annotations and the corresponding segment boundaries generated by QuickCut. Across all our results, the difference between the start time of an audio annotation and the computed annotation segment by QuickCut varied from .02 to 18.89 seconds, while the differences in end times ranged from .03 to 55.86 seconds. As noted below, users found the computed segments to be extremely useful and these numbers suggest that QuickCut can produce valid segments even when annotations are tens of seconds off from the start/end of an action. Moreover, while QuickCut allows users to manually adjust segment in- and out-times, none of our results required such adjustments.

In terms of quality, the automatically generated compositions include semantically relevant video footage for all narration events. Moreover, the frame and transition costs effectively eliminate a variety of blurry footage, camera shake and jump cut artifacts from the results. To see specific examples, please refer to our supplemental material, which includes three versions of each composition for comparison: *no-costs* does not include the frame and transition costs; *transition-only* adds only the transition cost, which eliminates jump cuts; and *full* includes both the transition cost and frame costs, which discourages the use of blurry and shaky footage in the final edit. For example, the *no-costs Pork Ribs* composition includes both blurry and shaky footage as well as jump cuts, while the version with *full* costs avoids all three types of artifacts. Our project website (<http://graphics.stanford.edu/projects/quickcut/>) lists specific places in each of the 6 examples where artifacts are visible in the *no-cost* versions but are eliminated by our optimization in the *full* cost version.

User Feedback

In addition to the professional filmmaker who made Duygu, we have informally shown our interface to two other professional filmmakers and one amateur video editor. All of them strongly appreciated the audio annotation workflow. One of the professionals contrasted this workflow with traditional logging saying, “*A lot of people get overwhelmed when you have to sort through the video footage with all the typing and searching for timestamps. But if you can just say it out loud and the program gets it close, that's hot.*” They all rated the resulting segmentations as either useful or extremely useful and similarly found the text-based search for matching narration events to annotations segments to be extremely useful.

We have also shown our results to 10 people with video editing experience and asked whether they thought the results would be acceptable for publication. All 10 said that all of our result videos would be acceptable for publishing as a class project

video, or to a YouTube channel and in many cases on broadcast TV (e.g., Duygu, Mexican Bagels, PorkRibs, Umami). We also asked the original Duygu and Umami editors to compare their manual edits to our QuickCut edit. Both felt that the quality of the QuickCut result was very comparable to the manual edit. Aside from applying some post-processing effects (e.g., text overlays, stabilization), they felt that the QuickCut results could be published without additional editing.

To evaluate the quality of our optimization for choosing frame-level cut points, we recruited five people from the first author's workplace to examine five sets of result videos (one with our *full* costs, one with our *transition-only* cost, and one with *no-costs*). We asked them to rank the versions from highest to lowest quality and our *full* cost version was always ranked first. We found no reliable quality preference between the *no-cost* and *transition-only* versions; both were judged as worse than the *full* cost version. These results suggest that our *full* cost optimization, which avoids blurry and shaky footage as well as jump cuts, noticeably improves the quality of the resulting videos compared to not performing these optimizations.

Limitations

Our approach has two main limitations:

Fixed narration timing. QuickCut assumes that the timing of the voiceover recording remains fixed. However, there are cases where it may be useful to change the length of pauses or even remove extraneous phrases to make the narration fit better with the available footage. To address these situations, our algorithm for generating compositions could jointly optimize the audio and video cut points. One challenge here is that the extra degrees of freedom quickly explode the dynamic programming state space, which makes the computation significantly more expensive.

Constant speed video. While QuickCut automatically chooses good cut points for video segments, it does not try to speed up or slow down the video to improve its alignment with the narration. This temporal scaling is especially common for screencast videos that show users interacting with software interfaces. As with voiceover edits, temporal scaling of videos could be incorporated into our current approach at the cost of expanding the search space for the composition optimization.

CONCLUSION

While narrated video is one of the most common forms of video available on the Web, creating high-quality narrated video is challenging. We have shown that QuickCut facilitates the creation of such video in two ways. First, it provides transcript-based matching of story events in the narration with semantically relevant video segments. This approach significantly reduces the amount of time authors spend scrubbing through their raw footage to find the appropriate segments. Second, it lets authors specify high-level alignment constraints between the narration events and the relevant footage and then uses dynamic programming optimization to find suitable frame-level cut points between the segments. This approach lets authors focus on the overall story while the system handles the low-level details of assembling the video segments together. As more and more people seek to convey ideas and

information via narrated video, we believe that these kinds of tools are needed to make video creation more accessible.

ACKNOWLEDGEMENTS

We dedicate this paper in loving memory of our dear friend and collaborator Floraine Berthouzoz, with whom we began this project and who drove much of its formative groundwork.

We thank Claire Li, Avi Bagla, Ashley Ngu, Karen Wang and Kevin Zhai for providing us with the raw footage used in our results; Erik Espera, Jenny Appleton, Patrick Tulipano, David Cerf, Daniel Nguyen, Tiffany Kang, Tyler Sipe and all of our user study participants for their valuable feedback and discussions. Finally we thank Colorado Reed and Kayla DesPortes for their many contributions to an early iteration of this work. This work was partially supported by the David and Helen Gurley Brown Institute for Media Innovation.

REFERENCES

1. Arev, I., Park, H. S., Sheikh, Y., Hodgins, J., and Shamir, A. Automatic editing of footage from multiple social cameras. *ACM Trans. Graph. (TOG)* 33, 4 (2014), 81.
2. Bao, X., and Roy Choudhury, R. Movi: Mobile phone based video highlights via collaborative sensing. In *Proc. MobiSys 2010*, ACM (2010), 357–370.
3. Berthouzoz, F., Li, W., and Agrawala, M. Tools for placing cuts and transitions in interview video. *ACM Trans. Graph. (TOG)* 31, 4 (2012), 67.
4. Bird, S. Nltk: the natural language toolkit. In *Proc. of the COLING/ACL on Interactive presentation sessions*, Association for Computational Linguistics (2006), 69–72.
5. Chi, P.-Y., Liu, J., Linder, J., Dontcheva, M., Li, W., and Hartmann, B. Democut: generating concise instructional videos for physical demonstrations. In *Proc. UIST 2014*, ACM (2013), 141–150.
6. Cour, T., Jordan, C., Miltakaki, E., and Taskar, B. Movie/script: Alignment and parsing of video and text transcription. In *Proc. ECCV 2008*. Springer, 2008, 158–171.
7. Fischler, M. A., and Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.
8. Heck, R., Wallick, M., and Gleicher, M. Virtual videography. *ACM Trans. Multimedia Comput. Commun. Appl. (TOMCCAP)* 3, 1 (2007), 4.
9. Jain, E., Sheikh, Y., Shamir, A., and Hodgins, J. Gaze-driven video re-editing. *ACM Trans. Graph. (TOG)* 34, 2 (2015), 21.
10. Joshi, N., Kienzle, W., Toelle, M., Uyttendaele, M., and Cohen, M. F. Real-time hyperlapse creation via optimal frame selection. *ACM Trans. Graph. (TOG)* 34, 4 (2015), 63.
11. Lowe, D. G. Object recognition from local scale-invariant features. In *Proc. ICCV 1999*, vol. 2, Ieee (1999), 1150–1157.
12. Lu, Z., and Grauman, K. Story-driven summarization for egocentric video. In *Proc. CVPR 2013*, IEEE (2013), 2714–2721.
13. Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
14. Pavel, A., Goldman, D. B., Hartmann, B., and Agrawala, M. Sceneskim: Searching and browsing movies using synchronized captions, scripts and plot summaries. In *Proc. UIST 2015*, ACM (2015), 181–190.
15. Pavel, A., Reed, C., Hartmann, B., and Agrawala, M. Video digests: A browsable, skimmable format for informational lecture videos. In *Proc. UIST 2014*, ACM (2014), 573–582.
16. Quora.com: How much time is spent editing footage for film/video. <http://quora.com/How-much-time-is-spent-editing-footage-for-Film-Video>. Retrieved 2016-04-11.
17. Ranjan, A., Birnholtz, J., and Balakrishnan, R. Improving meeting capture by applying television production principles with audio and motion detection. In *Proc. CHI 2008*, ACM (2008), 227–236.
18. Rubin, S., Berthouzoz, F., Mysore, G. J., Li, W., and Agrawala, M. Content-based tools for editing audio stories. In *Proc. UIST 2013*, ACM (2013), 113–122.
19. Russ, J. C. *The Image Processing Handbook, Fifth Edition (Image Processing Handbook)*. CRC Press, Inc., Boca Raton, FL, USA, 2006.
20. Shin, H. V., Berthouzoz, F., Li, W., and Durand, F. Visual transcripts: Lecture notes from blackboard-style lecture videos. *ACM Trans. Graph. (TOG)* 34, 6 (Oct. 2015), 240:1–240:10.
21. Shrestha, P., Weda, H., Barbieri, M., Aarts, E. H., et al. Automatic mashup generation from multiple-camera concert recordings. In *Proc. Multimedia 2010*, ACM (2010), 541–550.
22. Sumec, S. Multi camera automatic video editing. In *Computer Vision and Graphics*. Springer, 2006, 935–945.
23. Tapaswi, M., Bäuml, M., and Stiefelhagen, R. Book2movie: Aligning video scenes with book chapters. In *Proc. CVPR 2015* (2015), 1827–1835.
24. Web Speech API. <https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html>. Retrieved 2016-04-11.
25. Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *arXiv preprint arXiv:1506.06724* (2015).
26. Zsombori, V., Frantzis, M., Guimaraes, R. L., Ursu, M. F., Cesar, P., Kegel, I., Craigie, R., and Bulterman, D. C. Automatic generation of video narratives from shared ugc. In *Proc. Hypertext 2011*, ACM (2011), 325–334.