

Computational Video Editing for Dialogue-Driven Scenes

MACKENZIE LEAKE, Stanford University

ABE DAVIS, Stanford University

ANH TRUONG, Adobe Research

MANEESH AGRAWALA, Stanford University

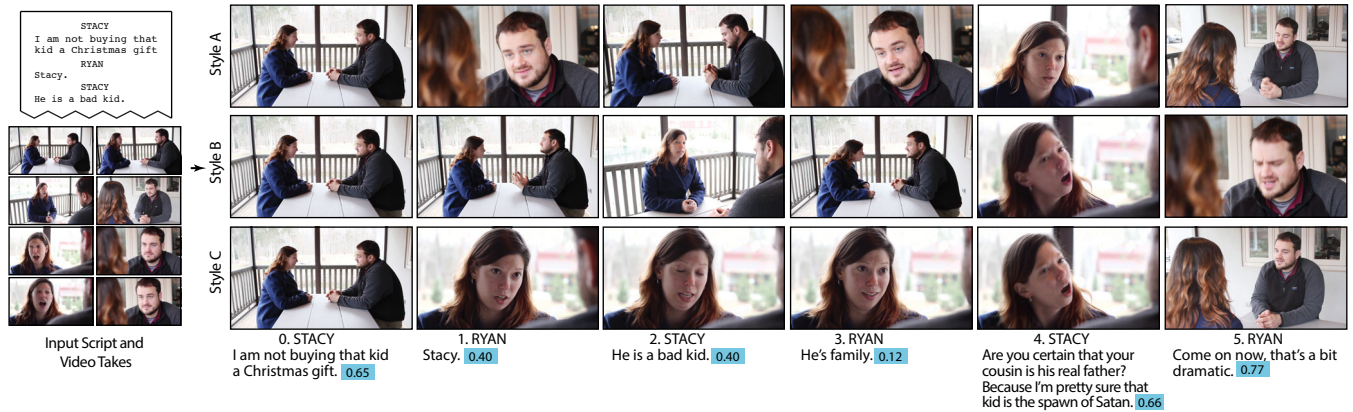


Fig. 1. Given a script and multiple video recordings, or takes, of a dialogue-driven scene as input (left), our computational video editing system automatically selects the most appropriate clip from one of the takes for each line of dialogue in the script based on a set of user-specified film-editing idioms (right). For this scene titled *Fluffles*, editing style A (top row) combines two such idioms; *start wide* ensures that the first clip is a wide, establishing shot of all the characters in the scene, and *speaker visible* ensures that the speaker of each line of dialogue is visible. Editing style B (middle) adds in the *intensify emotion* idiom, which reserves close ups for strongly emotional lines of dialogue, as in lines 4 and 5 where the emotional sentiment strength (shown in blue) is greater than 0.65. Editing style C (bottom) replaces the intensify emotion idiom with *emphasize character* that focuses on the Stacy character whenever Ryan has a particularly short line of dialogue, as in lines 1 and 3.

We present a system for efficiently editing video of dialogue-driven scenes. The input to our system is a standard film script and multiple video takes, each capturing a different camera framing or performance of the complete scene. Our system then automatically selects the most appropriate clip from one of the input takes, for each line of dialogue, based on a user-specified set of film-editing idioms. Our system starts by segmenting the input script into lines of dialogue and then splitting each input take into a sequence of clips time-aligned with each line. Next, it labels the script and the clips with high-level structural information (e.g., emotional sentiment of dialogue, camera framing of clip, etc.). After this pre-process, our interface offers a set of basic idioms that users can combine in a variety of ways to build custom editing styles. Our system encodes each basic idiom as a Hidden Markov Model that relates editing decisions to the labels extracted in the pre-process. For short scenes (< 2 minutes, 8-16 takes, 6-27 lines of dialogue) applying the user-specified combination of idioms to the pre-processed inputs generates an edited sequence in 2-3 seconds. We show that this is significantly faster than the hours of user time skilled editors typically require to produce such edits and that the quick feedback lets users iteratively explore the space of edit designs.

CCS Concepts: •Information systems → Multimedia content creation;

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.

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. 0730-0301/2017/7-ART130 \$15.00
DOI: <http://dx.doi.org/10.1145/3072959.3073653>

Additional Key Words and Phrases: video editing

ACM Reference format:

Mackenzie Leake, Abe Davis, Anh Truong, and Maneesh Agrawala. 2017. Computational Video Editing for Dialogue-Driven Scenes. *ACM Trans. Graph.* 36, 4, Article 130 (July 2017), 14 pages.
DOI: <http://dx.doi.org/10.1145/3072959.3073653>

1 INTRODUCTION

Digital cameras make it easy for filmmakers to record many versions, or *takes*, of a scene. Each new take can provide a unique camera framing or performance, and skilled editors know how to combine multiple takes to build a stronger narrative than any one recording could capture. While well-crafted edits are not always obvious to viewers [Smith and Henderson 2008], the best editors carefully cut between different framings and performances to control the visual style and emotional tone of a scene [Arijon 1976; Bowen 2013; Katz 1991; Murch 2001]. Unfortunately, the process of editing several takes together is slow and largely manual; editors must review each individual take, segment it by hand into *clips*, and arrange these clips on a timeline to tell a story. With existing frame-based video editing tools this process is especially tedious, making creative exploration of different editing styles very difficult.

In this paper we show that by focusing on a particular, but very common type of scene – namely, *dialogue-driven* scenes – we can create much more efficient tools for editing video¹. Such scenes are

¹Our results can be viewed at <http://graphics.stanford.edu/papers/roughcut>

one of the most common elements in live-action films and television. From big-screen comedies and dramas, to small-screen sitcoms and soap operas, conversational scenes take a large percentage of screen time. Even action films use dialogue-driven scenes to establish relationships between characters. A common workflow for producing such scenes is to first develop a script containing the dialogue, and then capture multiple takes of the complete scene. The script provides an overall structure of the narrative, and our editing tools are designed to let users make editing decisions based on this structure.

We build on the established concept of *film-editing idioms*, which represent rules-of-thumb for conveying a narrative through editing decisions – e.g., ensure that the speaker of each line of dialogue is visible, intensify emotion by using close ups for emotional lines, etc. (Figure 1). Previous techniques have applied film-editing idioms to the problem of virtual cinematography in 3D environments [Christianson et al. 1996; Elson and Riedl 2007; Galvane et al. 2015; He et al. 1996; Jhala and Young 2005; Karp and Feiner 1993; Merabti et al. 2015], and to automatically edit video of classroom presentations [Heck et al. 2007] or social gatherings [Arev et al. 2014]. While we are inspired by this work, our system is the first to provide idiom-based editing for live-action video of dialogue-driven scenes, thereby enabling fast creative exploration of different editing styles.

Our approach is designed to augment a typical workflow for producing dialogue-driven scenes. The input to our system is a standard film script, and several recorded takes of a scene. We begin by running these inputs through our own segmentation and labeling pre-processing pipeline, which automatically extracts high-level structural information that allows our system to apply idioms to the scene. Our tools let users explore different editing *styles* by composing basic film-editing idioms in different combinations. Finally, users can apply the resulting styles to any labeled scene, immediately and automatically producing a fully edited video.

Our work makes three main contributions:

(1) Automatic segmentation and labeling pipeline. Most film-editing idioms relate editing decisions to high-level structural information about a scene. For example, applying the *speaker visible* idiom (Figure 1) requires at least two pieces of structural information; (1) which video clips are associated with each line of dialogue, and (2) whether or not the performer speaking the line is visible in the corresponding clip. A key insight of our work is that, for dialogue-driven scenes, we can extract this kind of structural information automatically from a script and raw input videos. To this end, we present an automatic segmentation and labeling pipeline that breaks the script into lines of dialogue, then splits each take into a sequence of clips time-aligned with each line, and finally extracts enough script- and clip-level labels to support a variety of useful idioms (Section 4).

(2) Composable representation for film-editing idioms. Filmmakers usually combine multiple idioms to produce an edit that conveys the narrative in a particular style. Figure 1 shows several such idiom combinations for a scene. Flexibility in exploring different combinations of idioms is essential for filmmakers to design their own editing styles. Therefore, we propose representing individual idioms as conditional probability distributions encoded in

Hidden Markov Models (HMMs) [Rabiner 1989]. This approach lets us combine idioms through simple arithmetic operations. Given a combination of idioms, we construct the corresponding HMM so that the maximum-likelihood set of hidden variables yields an edit in the desired style (Section 5).

(3) Idiom-based editing interface. Existing video editing tools force users to work with a frame-based timeline representation of the raw video takes. Editors must therefore translate the high-level film editing idioms into low-level operations such as selecting, trimming and assembling clips into the edited result. The tediousness of scrubbing through a timeline with these tools makes it very difficult for editors to efficiently explore the space of possible edits. In contrast, we provide a prototype interface for idiom-based editing of dialogue-driven scenes. Users can drag and drop basic idioms, adjust their parameters and set their relative weights, to build custom idiom combinations. Applying the resulting combination to the scene generates an edited sequence with 2-3 seconds of processing time, and users can iteratively update the idioms and their parameters based on the result. Such immediate feedback is crucial for creative exploration of the edit design space.

We demonstrate the effectiveness of our system by generating edits for 8 short dialogue-driven scenes (< 2 minutes, 8-16 takes, 6-27 lines of dialogue) in a variety of editing styles. These scenes each required 110-217 minutes of system time to automatically segment and label. Applying a user-specified combination of idioms to such pre-processed inputs takes 2-3 seconds. Our evaluation shows that this is significantly faster than the hours of user time skilled editors typically require to produce such edits and that the quick feedback lets users iteratively explore the space of edit designs.

2 RELATED WORK

Virtual cinematography. Using the conventions of cinematography to convey animated 3D content, such as virtual actors performing in a 3D game environment, is a well-studied problem in computer graphics. This work has focused on algorithmically encoding film-editing idioms using knowledge-based planners [Jhala and Young 2005; Karp and Feiner 1993], via declarative camera control languages [Christianson et al. 1996] that can be modeled as finite state machines [He et al. 1996], or as penalty terms of an optimization that can be solved using dynamic programming [Elson and Riedl 2007; Galvane et al. 2014, 2015; Lino et al. 2011]. While our work is inspired by these techniques, they rely on access to the underlying sequence of geometry, events and actions (e.g., speaking, reacting, walking) taking place in the virtual environment and therefore cannot directly operate on live-action video. In contrast, our system automatically extracts such scene structure from the input script and the raw takes of live-action video. In addition, the earlier methods do not provide an interface for combining multiple idioms; the idioms are either baked in and cannot be changed or require users to create new idioms by programming them using specialized camera planning languages [Christianson et al. 1996; Ronfard et al. 2013; Wu and Christie 2016]. Our editing system includes an interface for combining a pre-built set of basic idioms to explore the space of editing styles.

Idiom Name	Description	Labels Required
<i>Avoid jump cuts</i>	Avoid transitions between clips that show the same visible speakers to prevent jarring transitions.	(S) (V) Speakers visible
<i>Change zoom gradually</i>	Avoid large changes in zoom level that can disorient viewers and instead change zoom levels slowly.	(S) (V) Shot type: Zoom
<i>Emphasize character</i>	Do not cut away from shots that focus on an important character unless another character has a long line. This focuses the audience's attention on the more important character.	(S) (V) Speakers visible, Clip length
<i>Intensify emotion</i>	Use close ups for particularly emotional lines to provide more detail in the performer's face.	(S) Emotional sentiment (V) Shot type: Zoom
<i>Mirror position</i>	Select clips for one performer that most closely mirror the screen position of the other performer to create a back-and-forth dynamic for a two person conversation.	(S) (V) Screen position, Shot type: NumVis
<i>Peaks and valleys</i>	Zoom in for more emotional lines and zoom out for less emotional lines to allow the audience to see more detail in the performer's faces for emotional lines.	(S) Emotional sentiment (V) Shot type: Zoom
<i>Performance fast/slow</i>	Select shorter(longer) performances of a line to control the pacing of the scene.	(S) (V) Clip length
<i>Performance loud/quiet</i>	Select louder(quieter) performances of a line to control volume of the scene.	(S) (V) Clip volume
<i>Short lines</i>	Avoid cutting to a new shot for only a short amount of time to prevent rapid, successive cuts that can be jarring.	(S) Speaker (V) Speakers visible, Clip length
<i>Speaker visible</i>	Show the face of the speaking character on screen to help the audience keep track of which character is speaking and understand the progression of the conversation.	(S) Speaker (V) Speakers visible
<i>Start wide</i>	Start with the widest shot possible to establish the scene (i.e., start with <i>establishing shot</i>) and show the relationship between performers and the surroundings.	(S) (V) Shot type: Zoom
<i>Zoom consistent</i>	Maintain a consistent zoom level throughout the scene to create a sense of balance between the performers.	(S) (V) Shot type: Zoom
<i>Zoom in/out</i>	Specify a preference for zooming in(out) throughout a scene to reveal more(less) detail in performers' faces and create more(less) intimacy.	(S) (V) Shot type: Zoom

Table 1. Basic film-editing idioms for dialogue-driven scenes as distilled from books on cinematography and filmmaking. Our computational video editing system allows users to combine these basic idioms to explore a variety of different editing styles. The “Labels Required” column describes the high-level structural information about the script (S) or the input video takes (V) required by our implementations of these idioms.

Instead of an idiom-based approach, Merabti et al. [2015] learn film editing styles from existing films. They model the editing process using a Hidden Markov Model (HMM) and hand-annotate existing film scenes to serve as training data. But manual annotation is tedious, and this approach still requires access to the underlying scene actions when applying the learned style to new animated scenes. Moreover, users cannot combine these multiple learned styles to explore the editing design space. While we similarly model the editing process with an HMM, our system directly encodes a set of basic idioms and lets users combine them to produce a variety of editing styles.

Fully-automated video editing. Fully automated techniques for editing video footage have been developed for several domains. Closest to our work are systems for editing video of group meetings [Ranjan et al. 2008; Takemae et al. 2003], educational lectures and presentations [Heck et al. 2007; Liu et al. 2001; Shin et al. 2015], and parties or other social gatherings [Arev et al. 2014; Zsombori et al. 2011]. These methods combine general-purpose film editing idioms (e.g., speaker visible, avoid jump-cuts) with domain specific guidelines to produce the edited result. For example, with classroom lectures Liu et al. [2001] suggest that the camera should occasionally show local audience members to make the video more interesting to watch. Arev et al. [2014] explain that in social gatherings the joint center of attention should be onscreen as much as possible. All of the techniques are designed to produce a single edited sequence as output. While our video editing system similarly includes general-purpose idioms with idioms specific to dialogue-driven scenes, it also provides an interface for controlling the strength with which different idioms are applied.

Interactive video editing. While commercial video editing tools are primarily frame-based, researchers have investigated the use

of higher-level editing tools. For example, Girgensohn et al. [2000] analyze the raw footage to remove low quality (e.g., shaky, blurry, or dark) segments. Users edit the video by rearranging the remaining high-quality clips. Chi et al. [2013] use video analysis techniques to segment demonstrations of physical procedures (e.g., craft projects, cooking, etc.) into meaningful steps that users can arrange into tutorial videos. Recently, a number of researchers have developed transcript-based tools for editing talking-head style interview video [Berthouzoz et al. 2012], condensing and rearranging speech for audio podcasts [Rubin et al. 2013; Shin et al. 2016], annotating video with review feedback [Pavel et al. 2016], selecting b-roll clips for narrated documentary style video [Truong et al. 2016] and generating structured summaries of informational lecture videos [Pavel et al. 2014]. Our system similarly leverages time-aligned scripts to facilitate editing of dialogue-driven scenes. But, unlike earlier methods, it also uses the structure imposed by the script to apply higher-level idioms of film editing.

3 SYSTEM OVERVIEW

It is common practice in filmmaking to start by writing a script and then capture multiple takes of the scene ensuring that there is enough *coverage* – variations in camera framings and performances – within the takes to cover the entire script. Our computational video editing system requires no modification to this standard workflow and takes the script as well as the raw takes as input. Moreover, as new video capture techniques, such as robotic cameras [Byers et al. 2003; Joubert et al. 2016; Kim et al. 2010] or multi-crop generation from a single wide-angle camera [Gandhi and Ronfard 2015; Gandhi et al. 2014], become available, our system can directly incorporate these sources for the raw takes.

The challenge of film-editing is to choose the most appropriate camera framing and performance from the available takes for each



Fig. 2. Our system automatically breaks an input script into lines of dialogue spoken by each character (left). This script contains 27 lines of dialogue. It then time-aligns the script with each input take of the scene (15 takes in this case), and uses the alignment to divide each take into a sequence of clips, such that each clip corresponds to a line of the script (right).

moment in the scene. While this is an extremely large space of editing design choices, expert editors rely on the conventions of cinematography to limit the size of this space to a manageable set of choices. For example, one of the most common conventions is to cut between lines of dialogue rather than within a line. Books on cinematography and filmmaking [Arijon 1976; Bowen 2013; Katz 1991; Murch 2001] describe a number of film-editing idioms that can further guide the editing process. We have distilled the idioms found in these books into a basic set that can be combined to produce a variety of editing styles for dialogue-driven scenes. Many of the idioms are based on higher-level structural information about the scene. Table 1 presents the set of basic idioms included in our editing tool along with the structural information that they use.

Given a script and multiple takes of a scene as input, our computational video editing system selects the most appropriate clip from one of the takes for each line of dialogue in the script based on a set of user-specified idioms. Our system operates in two stages. In the pre-processing stage, it applies an automatic segmentation and labeling pipeline (Section 4) to extract structural information from the input script and takes. Then, in the editing stage (Section 5), it uses the structural information to apply the user-specified idioms. The interface to our system (Section 6) lets users test different combinations of the idioms and thereby explore the space of film-editing styles.

4 PRE-PROCESS: SEGMENTATION AND LABELING

We pre-process the input script and takes in two steps. In the segmentation step we break the script into lines and time-align each such line with a corresponding clip from each input take. In the labeling step we automatically extract additional structure and high-level attributes for the script (e.g., emotional sentiment of each line) and clips (e.g., the location of faces in each video clip).

4.1 Segmentation

Scripts are often formatted according to AMPAS screenplay formatting standard [Rider 2016] so that the name of the character appears centered in all capitals above the dialogue spoken by that character (Figure 2). We start by parsing the input script, assuming it is in the AMPAS format, to produce an ordered sequence of lines, where each line is labeled with the name of the character who speaks it.

Next, we time-align the text dialogue in the script with the speech dialogue in each input video take using the phoneme-mapping algorithm of Rubin et al. [2013]. The aligner produces a word-level mapping between the text and each take. We use this mapping to segment each take into a sequence of clips, where each clip corresponds to a line of dialogue from the script. In practice we have found that performers often deviate from the exact wording of a line in the script, adding or removing a word or phrase. Although these deviations can cause misalignments between the script and a take, as long as the misalignments are contained within a line of dialogue, our approach properly segments each take into a sequence of clips corresponding to each line.

If performers deviate significantly from the script, misalignments may cross multiple lines of dialogue. In such cases we first use either an automatic text-to-speech tool [IBM 2016; Ochshorn and Hawkins 2016] or crowdsourcing transcription services, like rev.com, to obtain a high-quality transcript of the input take. We then time-align the accurate transcript to the take again using Rubin et al.’s phoneme-mapping algorithm. Finally we apply Levenshtein’s edit distance, using words as tokens, to align the accurate take-specific transcript to the input script, and propagate this alignment to the clips to obtain the correct clip to line correspondence.

The result of this segmentation is that the input script S is divided into a sequence of lines l_0, \dots, l_L . Similarly each input video take t_k in the set of input takes $T = \{t_0, \dots, t_N\}$ is divided into a sequence of clips c_0^k, \dots, c_L^k such that each clip c_i^k corresponds to the line l_i .

4.2 Labeling

Our labeler analyzes the lines of text from the script as well as the video clips associated with each line to generate additional structural information about the scene. Each script or video label provides one or more functions of the form $f(l_i)$ or $f(c_i^k)$ that emits the label values associated with line l_i or clip c_i^k respectively.

Script Labels

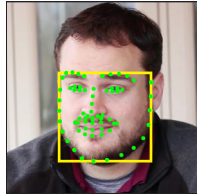
We obtain the following labels for each line l_i in the script:

Speaker. As noted in Section 4.1, our segmentation algorithm labels each line with the name of the speaking character. Provides label function $spkr(l_i)$.

Emotional sentiment. We apply the NLTK [Bird 2006] sentiment analysis algorithm to each line of dialogue to obtain *positive* and *negative* valence probabilities (these sum to 1.0) as well as an independent probability that the line is emotionally *neutral*. We treat $(1 - \text{neutral})$ as a measure of the intensity of the emotion in the line (e.g., arousal), since we expect intensely emotional lines to have low neutrality and vice versa. Provides label functions $emPos(l_i)$, $emNeg(l_i)$ and $emInt(l_i)$.

Video Labels

We apply the face detection and tracking algorithm of OpenFace [Baltrušaitis et al. 2016] to each input video take to obtain 68 facial landmark points (lying on the eyes, eyebrow, mouth, nose, and face contour) for each face detected in each video frame. We also compute the axis-aligned bounding box of the landmarks points as the face bounding box for each frame (see inset). We use this information to compute the following labels for each video clip.



Screen position. For each frame within a clip we compute the center of each face bounding box. We then average this position across all of the frames in the clip to obtain a clip-level label representing the screen position of each performer’s face. Provides label functions $pos_x(c_i^k)$ and $pos_y(c_i^k)$.

Filmmakers typically classify the shot type of a clip based on the number of performers in the frame (1-shot, 2-shot, etc.) and the relative size (or zoom level) of the closest performer in the frame (wide, medium, close up, etc.). Thus, we label each clip with:

Shot type: NumVis. We compute the number of performers in each clip as the median number of faces detected per frame across the set frames contained in the clip. Provides label function $num(c_i^k)$.

Shot type: Zoom. We identify the face with the largest median bounding box height across the clip. We then compute the zoom factor for the clip as the ratio of the median height of this face to the height of the frame. Finally, we classify the clip into one of seven of Bowen’s [2013] zoom-level categories – (1) extreme wide shot (EWS), (2) wide shot (WS), (3) medium wide shot (MWS), (4) medium shot (MS), (5) medium close up (MCU), (6) close up (CU), (7) extreme close up (ECU) – using SciKitLearn’s K-nearest neighbor classifier [Pedregosa et al. 2011] on the zoom factor. To build this classifier we ran our face detector on 74 short scenes we recorded, and randomly sampled 10 frames from each film that included at least one detected face. We hand-labeled the face bounding box and the zoom-level, and used this as training data for our classifier. The labels are ordered from widest (e.g., smallest face relative to frame) to closest so that we can use the zoom-level number in our implementations of the film-editing idioms. Provides label function $zoom(c_i^k)$.

Speakers visible. We identify the speakers that are visible in each take by grouping clips according to the character listed as the speaker of the corresponding line of dialogue and comparing average mouth motions of the faces between these groups. Consider a



Fig. 3. To compute the *speakers visible* label for each video clip, we first compute the median change in mouth area across all frame within the clip. The graph shows this clip mouth motion for the clips from take 2 (green line) and take 5 (blue line), corresponding to the first 10 lines of dialogue of this scene. For clips in take 2 the mouth motion is much higher when Stacy is speaking according to the script, than when Ryan is speaking, and vice versa for take 5.

scene with two characters Stacy and Ryan, and two takes – close ups of each character (Figure 3). For each frame i of these two takes we compute the area contained by the mouth landmark points m_i . The frame-to-frame change in mouth area $|m_i - m_{i-1}|$ is a measure of the mouth motion between frames. We compute the *clip mouth motion* as the median mouth motion across all the frames in the clip. We assume that within any take of Stacy (e.g., take 2), the clips corresponding to Stacy’s lines of dialogue will have larger mouth motions compared to the clips for which Ryan is the speaker. A similar argument applies takes of Ryan (e.g., take 5). So for each take we group together all of the clips corresponding to each speaking character based on the *speaker script* label. In this case, we group together clips for lines 0, 2, 4, 6 and 8 because they are Stacy’s lines and we group the remaining clips because they are Ryan’s lines. The speaking character associated with the group with the highest average clip mouth motion is likely to be visible, as large mouth motions imply that the mouth is visibly changing in area a lot. In our example, we see that for take 2 the average clip mouth motion for Stacy’s lines is higher than for Ryan’s lines, and the opposite holds for take 5. Therefore, we label Stacy as the speaker visible in every clip of take 2 and Ryan as the speaker visible in every clip of take 5. For takes containing multiple faces we apply the same algorithm separately considering groups for each tracked face within the take. In such cases multiple characters may be listed as visible. Provides label function $svis(c_i^k)$.

Clip volume. We compute the average root mean square (RMS) energy of the audio signal to determine the volume of the spoken performance for each clip. Provides label function $avgvol(c_i^k)$.

Clip length. We treat the total duration of the clip as a label. Provides label function $len(c_i^k)$.

5 EDITING

To generate an edited sequence, we must select a single clip from the available takes for each corresponding line of dialogue in our script. For a scene with L lines and N recorded takes, this leaves us a space of N^L alternative sequences to choose from. Our task is to

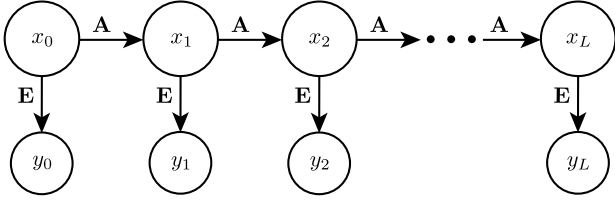


Fig. 4. We use an HMM to model the editing process. Each hidden state x_i represents the selection of a clip c_i^k from take t_k to be used for the corresponding line l_i . Each observation y_i contains any script- or clip-level labels that apply to line l_i .

find the sequence that best matches a set of user-specified idioms, or *editing style*. We build on the established machinery of Hidden Markov Models (HMMs) [Rabiner 1989], which offer a natural and efficient way to optimize over time-sequence data.

A standard HMM relates the time series of hidden states $\mathbf{x} = x_0, \dots, x_L$ to corresponding observations $\mathbf{y} = y_0, \dots, y_L$, through a probability distribution $P(\mathbf{x}|\mathbf{y})$ with the pattern of conditional dependencies shown in Figure 4. Each x_i can take a value from the state space $\mathbf{T} = \{t_0, \dots, t_N\}$ for hidden variables, and each y_i can take a value from the space $\mathbf{U} = \{u_0, \dots, u_M\}$ of observations. The HMM then describes $P(\mathbf{x}|\mathbf{y})$ through a $N \times 1$ vector of start probabilities \mathbf{b} , a $M \times M$ transition matrix \mathbf{A} , and a $N \times M$ emission matrix \mathbf{E} , where

$$b_k = P(x_0 = t_k), \quad s.t. \quad \sum_k b_k = 1 \quad (1)$$

$$A_{j,k} = P(x_{i+1} = t_k | x_i = t_j), \quad s.t. \quad \sum_k A_{j,k} = 1 \quad (2)$$

$$E_{j,k} = P(y_i = u_k | x_i = t_j), \quad s.t. \quad \sum_j E_{j,k} = 1 \quad (3)$$

Given a sequence of observations \mathbf{y} , the Viterbi algorithm [1967] offers an efficient way to calculate the maximum-likelihood sequence of unknown hidden states $\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} P(\mathbf{x}|\mathbf{y})$.

In our setting, each hidden state x_i represents the selection of a clip c_i^k from take t_k to be used for the corresponding line l_i . Each observation y_i then contains any script- or clip-level labels that apply to line l_i (e.g., emotional sentiment, speaker visibility). In this case, \mathbf{b} controls the probability of starting a scene with a particular clip, \mathbf{A} controls the probability of cutting from one clip to another, and \mathbf{E} controls the probability of using a particular clip to convey a particular line of dialogue. Unlike classic HMMs, we allow \mathbf{A} and \mathbf{E} to vary across time, noting that the Viterbi algorithm still produces the correct maximum-likelihood sequence $\hat{\mathbf{x}}$ [Forney 1973].

One concern with this formulation is that the size of the emission matrix \mathbf{E} depends on M , the number of unique possible observations. With a large space of potential labelings (possible values of y_i), \mathbf{E} becomes impractically large. We address this by noting that, in practice, calculating the maximum-likelihood sequence $\hat{\mathbf{x}}$ only requires the columns of \mathbf{E} that correspond to labelings we actually encounter. Therefore, we only construct those columns of \mathbf{E} that correspond to these observations.

Another concern is that the size and number of \mathbf{b} , \mathbf{A} , and \mathbf{E} matrices depend on specific properties of our input (e.g., the number of input takes and lines in our script). To separate the design of idioms from these scene-specific properties, we define \mathbf{b} , \mathbf{A} , and \mathbf{E} implicitly through the functions $\mathcal{B}(c_0^k)$, $\mathcal{A}(c_i^k, c_{i+1}^j)$, and $\mathcal{E}(c_i^k)$. We assume that any function of clip c_i^k has access to all labels associated with c_i^k and l_i . By taking these scene-specific properties as parameters, our functions provide a recipe for constructing HMMs that can be applied to any scene. That is, we define \mathbf{b} , \mathbf{A} , and \mathbf{E} as

$$b_k \propto \mathcal{B}(c_0^k), \quad s.t. \quad \sum_k b_k = 1 \quad (4)$$

$$A_{j,k} \propto \mathcal{A}(c_i^k, c_{i+1}^j), \quad s.t. \quad \sum_k A_{j,k} = 1 \quad (5)$$

$$E_{j,k} \propto \mathcal{E}(c_i^k), \quad s.t. \quad \sum_k E_{j,k} = 1 \quad (6)$$

Note that in Equation 6, we apply the summation constraint to columns of \mathbf{E} , rather than rows of \mathbf{E} , as we did in Equation 3. Applying the constraint in this way lets us avoid enumerating the entire observation space of possible labelings. This constraint effectively assumes a uniform prior over the space of all possible labelings. While this approach introduces a constant of proportionality to the probabilities $P(\mathbf{x}|\mathbf{y})$ calculated by Viterbi, the constant does not affect the selection of the maximum-likelihood sequence $\hat{\mathbf{x}}$.

5.1 Encoding Basic Film-Editing Idioms

To encode a film-editing idiom into an HMM we must design the functions \mathcal{B} , \mathcal{A} , and \mathcal{E} to yield distributions $P(\mathbf{x}|\mathbf{y})$ that favor edit sequences satisfying the idiom. Here we present example encodings for a few basic idioms from Table 1. Encodings for the rest of the idioms in Table 1 can be found in Appendix A.

Start wide. To encourage the use of a wide, establishing shot for the first clip in the scene, we set the start probabilities to favor clips that have smaller zoom-level values (i.e., wide shots rather than close ups). We set the start probabilities as

$$\mathcal{B}(c_0^k) = \frac{1}{\text{zoom}(c_0^k)}, \quad \mathcal{A}(c_i^k, c_{i+1}^j) = 1, \quad \mathcal{E}(c_i^k) = 1$$

We set probabilities for every zoom level so that if the scene does not contain a particular zoom level (e.g., extreme wide shots) the idiom will encourage the use of the next widest shot.

Avoid jump cuts. To avoid jarring cuts between clips of the same visible speakers, we set the transition probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{E}(c_i^k) = 1$$

$$\mathcal{A}(c_i^k, c_{i+1}^j) = \begin{cases} 1 & \text{if } k = j \\ 1 & \text{if } \text{svis}(c_i^k) \neq \text{svis}(c_{i+1}^j) \\ \epsilon & \text{otherwise} \end{cases}$$

We favor two kinds of transitions, (1) those between clips that remain on the same take $k = j$ and (2) those that switch to a take in which the set of speakers visible is different $\text{svis}(c_i^k) \neq \text{svis}(c_{i+1}^j)$.

Speaker visible. We encourage the use of clips that show the face of the speaking character. We set the emission probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{A}(c_i^k, c_{i+1}^k) = 1$$

$$\mathcal{E}(c_i^k) = \begin{cases} 1, & \text{if } \text{spkr}(l_i) \in \text{svis}(c_i^k) \\ \epsilon & \text{otherwise} \end{cases}$$

so that we favor clips in which the speaker of the line is in the set of speakers visible in the clip.

Intensify emotion. We encourage the use of close ups whenever the emotional intensity of a line is high by setting the emission probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{A}(c_i^k, c_{i+1}^k) = 1$$

$$\mathcal{E}(c_i^k) = \begin{cases} 1 & \text{if } \text{emInt}(c_i^k) > \phi \text{ and } \text{zoom}(c_i^k) \geq 6 \text{ (CU)} \\ 1 & \text{if } \text{emInt}(c_i^k) \leq \phi \text{ and } \text{zoom}(c_i^k) < 6 \text{ (ECU)} \\ \epsilon & \text{otherwise} \end{cases}$$

where ϕ is a user-specified parameter for controlling the emotional intensity threshold at which close ups (CU) or extreme closeups (ECU) are preferred.

5.2 Composing Idioms

Filmmakers typically combine several different idioms when editing a scene. To let users explore this space computationally, we need a way to combine different idioms and a way to specify the relative importance of different idioms within a combination.

To combine multiple idioms we simply take an element-wise product of their corresponding HMM parameters \mathbf{b} , \mathbf{A} , and \mathbf{E} , and renormalize according to Eq 4-6. The resulting distribution $P(\mathbf{x}|\mathbf{y})$ assigns each edit sequence \mathbf{x} a probability proportional to the product of that sequence's probabilities in the original idioms. Composing two idioms in this manner simply produces a new HMM representation for the combined idiom, which can itself be further composed with other HMM-based idioms. We let users control the relative importance of different idioms by specifying a weight w for each one. We apply each weight as an exponent to its corresponding idiom before normalization (letting $w = 1$ by default), letting the relative weights of different idioms control their relative influence on the resulting combination. Negative weights can be used to encourage edits that violate an idiom.

5.3 Tempo Control

Editors can also control the style and tone of a scene by changing the amount of time left between cuts. Most performers leave some silence between the end of one line and the beginning of the line that follows (otherwise, performers are talking over one another). When we cut between two clips, we can choose how much of this silence to take from the beginning and end of each clip, providing some additional control over speed and tempo in our final edit. We set this spacing between the lines using two *tempo* parameters, α and β . We treat α as the fraction of available silence to be used before each line, and β as the fraction of silence to be used after each line. When consecutive clips are selected from the same take, our system forces $\alpha + \beta = 1$ to avoid skipping or repeating frames. For all other cuts, we use global pair of α and β parameters set by the

user. If users set $(\alpha + \beta) < 1$ then the spacing between lines shortens, giving a sense of increased tempo and urgency. Alternatively, if users set $(\alpha + \beta) > 1$ then the spacing between lines lengthens, giving the scene a slower feel. When $(\alpha + \beta) = 1$, the tempo on average matches that of the input takes. By default we set $\alpha = 0.9$ and $\beta = 0.1$, based on a survey of such between-line spacing in dialogue-driven scenes [Salt 2011].

6 INTERFACE

Figure 5 shows the main components of our editing interface. The Idiom Builder (C) is the primary tool for exploring the space of possible edits. The basic set of idioms are built into the interface (Table 1) and appear in the “Basic Idioms List”. The color (pink, green, blue) of these basic idioms depends on whether the idiom specifies a start function \mathcal{B} , a transition function \mathcal{A} , or emission function \mathcal{E} , respectively.

Users build a custom idiom combination by dragging one or more of these basic idioms into the “Idiom Building Area.” The interface automatically combines any basic idioms of the same type by computing an element-wise product of the corresponding HMM parameters \mathbf{b} , \mathbf{A} , and \mathbf{E} , and renormalizing them (Section 5.2). The user can also specify the weight of each idiom in the combination using the weight textbox to the right of each idiom. A few of our basic idioms take user-specified parameters. For example, the **short lines** idiom takes a length threshold parameter to determine the maximum length of a “short line” (Appendix A). Users can click on any basic idiom to modify such parameters in a pop-up window. The “Idiom Properties” area lets users set a name and description for the idiom they have built. Users can also adjust the tempo parameters α and β (Section 5.3) to set the timing between lines of dialogue in the edited sequence.

After building an idiom, users click “Generate” to apply the resulting HMM and produce the edited sequence of clips that maximizes the probability for the specified combination. The generated sequence populates the Edit View. Users can then further modify the idiom or completely rebuild it to further explore the space of edits. The edit resulting from each new combination of idioms is automatically saved along with the idiom itself so that users can toggle between the different edits using the “Saved Idioms” dropdown menu. In addition, users can directly choose to swap out any of the clips in a resulting edit with another clip of the same line by selecting it from the Clip View. Users can also click on a clip once it is in the Edit View to fix it in place, turning its black border gray. With such fixed clips in place, users can click the “Generate” button to re-apply the HMM to generate the maximum-likelihood edit sequence that passes through the fixed clips. Finally, users can click the “Render” button to export an MP4 video of the edit or click “Export EDL” to save the edit as an edit decision list (EDL) file that can be loaded into traditional video editing software like Adobe Premiere, Avid Media Composer, or Final Cut Pro for further refinement.

Our interface also provides tools for viewing and manually correcting labels, both on the script and on the video clips. For example, the clip labeling interface (inset next page) includes a “Timeline” that displays all of the clips within the take as alternating yellow and green blocks. A grey block indicates a region of silence. Users

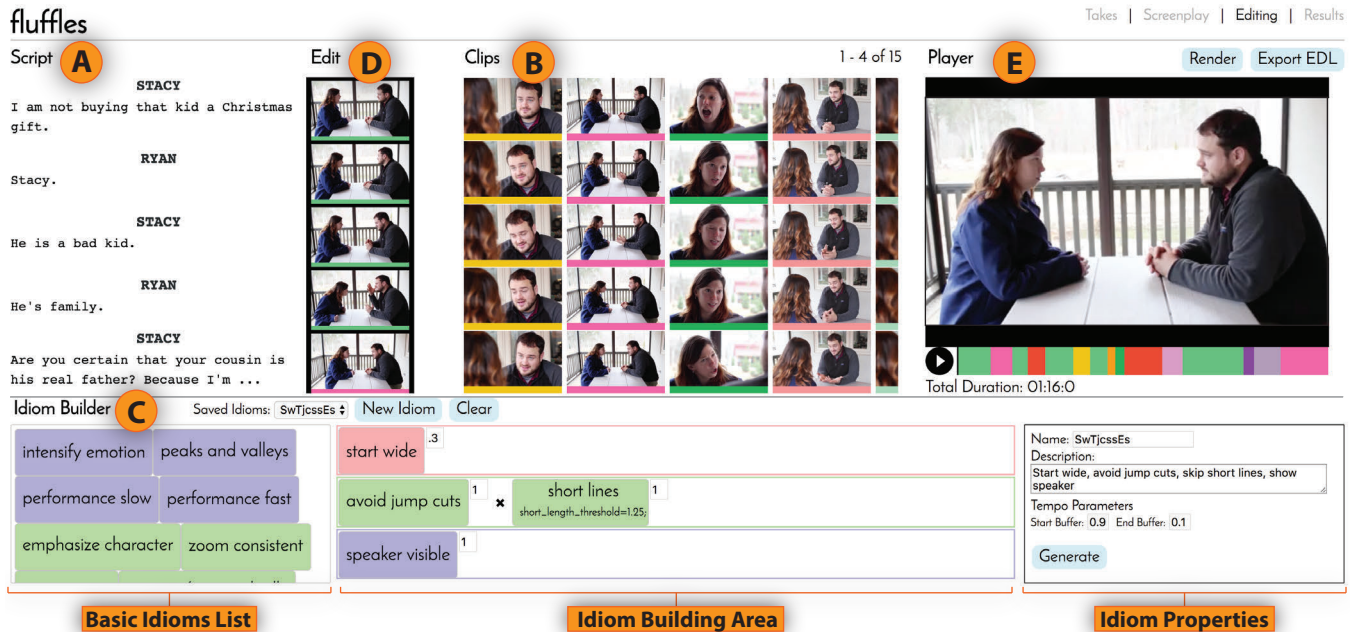


Fig. 5. Our film editing interface. The Script View (A) displays the character name and dialogue text for each line of the script. In the Clip View (B) each column shows an input take, split into clips, with each clip thumbnail horizontally aligned with the corresponding line of dialogue. Each take is assigned a unique color, and the colored bar under each clip thumbnail denotes the take it belongs to. Clicking on a clip plays it. The Idiom Builder (C) lets users combine and apply one or more basic idioms to explore the space of editing styles. The resulting edit appears in the Edit View (D), and each clip in the edit sequence is aligned with the corresponding lines of the script, just as in the Clip View. The “Player” View (E) lets users see the edited video and shows the color-coded take sequence in the timeline bar below.

can click on a block to play the corresponding clip and to bring up the “Clip Label” textbox showing all existing clip labels. Users can manually correct any errors in the labels and even add new labels to the clip. While we do not make use of this functionality for the results presented in this paper, we have found these tools to be useful for prototyping new labels and idioms.

7 RESULTS

We have used our computational video editing system to explore the edit design space for 8 dialogue-driven scenes, as listed in Table 2. Figures 1 and 6–9 show some of the different editing styles we can produce by combining our basic film-editing idioms. Supplemental materials provide a more comprehensive set of examples with both thumbnail strips and videos for each edit sequence. Our results are best experienced as videos, and we encourage readers to watch the videos provided in the supplemental materials.

We used the standard filmmaking workflow to obtain the input scenes. For some, we wrote the dialogue ourselves (*Fired*, *Fluffles*, *Friend*, *Goldfish*, *Krispies*), while others use dialogue from existing films (*Baby Steps*, *Princess Bride*, *Social Network*). We then used

Scene	Inputs			Pre-Processing			Editing	
	Takes	Lines	Dur	Align	Face	LbIs	HMM	Hand
Baby Steps	8	6	9.1m	2m	153m	11.4s	2s	105m
Fired	9	11	16.8m	5m	160m	15.0s	2s	105m
Fluffles	15	27	18.3m	4m	205m	22.3s	3s	180m
Friend	8	13	14.4m	3m	198m	14.7s	2s	135m
Goldfish	8	19	9.6m	3m	107m	14.2s	2s	105m
Krispies	15	8	14.7m	2m	169m	19.9s	2s	90m
Princess Bride	15	13	13.3m	4m	213m	25.3s	2s	135m
Social Network	13	9	7.6m	2m	147m	14.8s	2s	90m

Table 2. For each scene we report **Inputs**: number of raw takes (Takes), number of lines in script (Lines), and total duration of takes (Dur); **Pre-Processing**: time to align script to each take (Align), time to detect faces in each take (Face), and time to compute labels given face detections (LbIs); **Editing**: time required to apply an HMM and generate an edit (HMM) and time required by a skilled editor to generate an edit manually (Hand).

a single camera setup and recorded enough takes to ensure good coverage of a variety of common camera framings. We worked with amateur performers and usually captured several performances with each framing. We recorded the raw footage at 1080p resolution at 23.976 frames per second, and our system maintains this high-resolution throughout its pre-processing and editing stages. The supplemental videos were downsized as a post-process to reduce their file size. All of the results shown in this paper and in supplemental materials were generated using our fully automated segmentation and labeling pipeline without manual correction in our labeling interface. Although our editing interface allows users to manually select a clip for each line of dialogue, we did not use

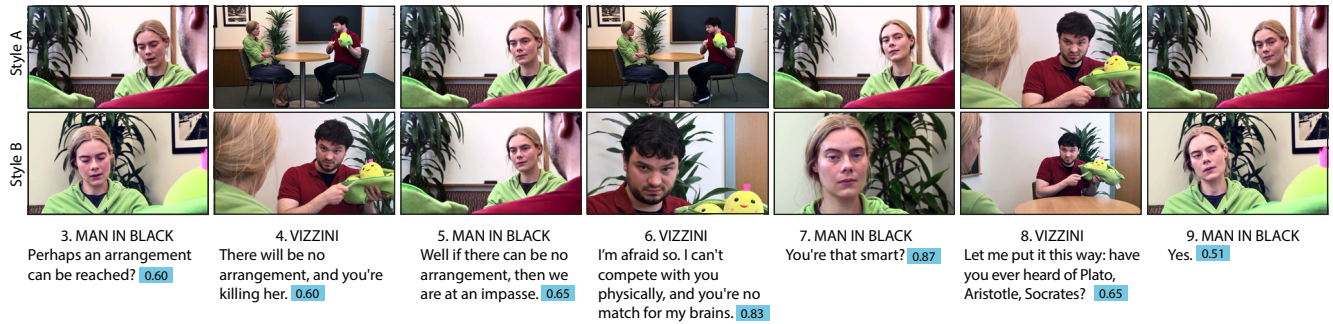


Fig. 6. Princess Bride. Editing style A combines three idioms: *start wide*, *speaker visible* and *avoid jump cuts*, while editing style B adds a fourth idiom, *peaks and valleys*, which uses close ups for the peak emotional lines 6 and 7 (emotional sentiment strength shown in blue), but zooms out to medium and wide shots for less emotional lines.



Fig. 7. Goldfish. Editing style A combines the idioms: *start wide* and *speaker visible* with *short lines*. In this case, lines 2, 3 and 6 are short, and the edit stays consistent on these lines with either of the longer adjacent lines on either side of them. In lines 2 and 3 the edit remains on the close up of Linda from line 4. For line 6 it stays on the medium shot of Craig from line 5. Editing style B replaces the *short lines* idiom with *emphasize character*, and we specify Craig as the character to emphasize on all short lines.

this functionality for the edited results we present and instead relied solely on combining idioms to produce them.

In developing the various editing styles shown in Figures 1 and 6–9, we first applied a standard editing style; we began the scene with a wide, establishing shot to reveal all the characters and ensured that the speaker of each line was visible (*Fluffles* Figure 1, style A and *Princess Bride* Figure 6, style A). We then iterated on the design of the resulting edit, adding or removing idioms or modifying idiom parameters and weightings, to see how the changes would affect the visual style and tone of the scene. For example, with *Fluffles* (Figure 1), editing style B uses close ups to intensify strongly emotional lines of dialogue, as when Stacy talks in line 4 about the kid being the “spawn of Satan” and Ryan responds in line 5. In editing style C we instead chose to emphasize Stacy and focus on her as the more important character in the scene. For *Princess Bride* (Figure 6), editing style B uses close ups for the peak emotional lines, 6 and 7, to give a sense of intimacy with the performers at emotional moments and uses wider shots for less emotional lines to contrast with the emotional peaks.

Since *Goldfish* (Figure 7) has many short lines, ensuring that the speaker is visible produces many quick cuts. Editing styles A and B use different strategies to avoid such rapid cutting by either avoiding cutting to a new take for short lines (style A) or by emphasizing a particular character (style B). In *Fired* (Figure 8) the performers varied the pace of their delivery in each take, with many long pauses within and between lines. Editing styles A and B explore how selecting either the longest or shortest performance of lines, along

with setting the tempo controls to increase or decrease the spacing between lines, affects the pacing and overall feel of the scene. The *Krispies* (Figure 9) edits explore how the weight of an idiom can be used to control its strength relative to other idioms in combination. As the weight on the *performance fast* idiom is increased, it locally takes precedence over the other idioms more and more often, and the length of the edited result becomes shorter and shorter.

7.1 Evaluation

We evaluated our system on a 3.1GHz MacBook Pro Laptop with 16GB of memory and as Table 2 shows, the slowest stage of our system is pre-processing. The pre-processing time is dominated by face detection and tracking, which can take several hours depending on the number of takes. While we report the total time for running each pre-processing step serially on each take, we could alternatively run them in parallel on all the takes at once to significantly speed up the pre-processing time. We also expect that using lower-resolution video subsampled in time would reduce this pre-processing time further without appreciably decreasing the accuracy of our labels, but we leave such optimization to future work. We note, however, that the pre-process is fully automatic and does not require any work from the user. In contrast to the pre-processing pipeline, the time required by our system to generate an edit based on a user-specified combination of idioms is 2-3 seconds. Thus, once our fully automated pre-processing is complete, users can efficiently explore different combinations of idioms.



Fig. 8. Fired. Editing styles A and B both apply the *start wide*, *speaker visible*, and *performance fast*, or *performance slow* idioms respectively to control the pacing of the scene. In style A the edit selects the shortest performance of each line subject to the other idioms to generate a faster overall sequence, while in style B the edit selects the longest performance of each line relative to the other idioms to give the scene a slower feel. Note that in this example focusing on timing is best viewed in supplemental materials.



Fig. 9. Krispies. Editing styles A and B both combine the *start wide*, *speaker visible*, and *avoid jump cuts* idioms with the *performance fast* idiom, which tries to select the fastest delivery of each line to control the pacing of the scene. In style A we set the weight of *performance fast* to 0, and, therefore, it effectively ignores the idiom. We generate an edit that is 53 seconds long. In style B we set the weight to 50 so that the idiom is enforced even when it locally conflicts with another idiom. For example, line 0 is no longer a wide, establishing shot in style B, despite the *start wide* idiom having been applied. The resulting edit is 41 seconds long. The graph (right) shows how the length of the edit becomes shorter as we increase the weight of the *performance fast* idiom from 0 to 50, and the idiom locally takes precedence over the other idioms more and more often.

Manually checking the pre-processed results, we found that our phoneme-based script-to-align would sometimes drop a syllable at the start or ends of lines and occasionally include extra words from an adjacent line. Although our tempo controls are designed to insert space between lines and can thereby correct most such errors in the edited result, it is possible to hear missing syllables and repeated words in a few cases. We also manually checked the labels generated based on our face detection and tracking (e.g., *shot-type: zoom*, *speaker visible*, etc.) and found that while our face detector could be inaccurate on a subset of frames within a clip, when we aggregated the face-based labels across an entire clip, the labels were correct for all 8 of our scenes.

7.2 Comparison to Professional Hand-Edited Results

We asked a professional film editor with 25 years of experience to manually edit each of our scenes (see Supplemental Materials). We asked him to ensure that the final edited results contained every line in the script and to make sure that the audio and video were cut together at the same point in time. We also asked him not to add in fades or any other special effects. He was free to design any editing style he thought best fit the scene and could use any editing software. He chose Avid Media Composer, a commercial frame-based video editing tool.

As shown in Table 2, in total it took him 90-180 minutes to produce each edited result. He reported that logging the raw takes to mark boundaries between lines of dialogue took 15-30% of his time, while the remainder was spent choosing an appropriate sequence of clips. Although he only produced a single edit for each scene, he estimated that re-editing a scene in a new style would require 60-90 minutes depending on the complexity of the scene and the desired style. In contrast, after our automatic pre-processing stage, our interface can apply a set of idioms to generate an edited result in 2-3 seconds. This is a substantial savings of human time and effort that can be spent iteratively designing and testing significantly more stylistic variations.

Overall, while his edited results appear more polished than those generated by our system, many of the editing decisions are similar. For example, he mostly cut between the lines of dialogue and used a variety of shot types just as in most of the edits we designed with our system. He also consistently started with a wide establishing shot. He also told us that he strategically used close ups for emotional dialogue, though the lines he considered most emotional sometimes did not match those chosen by our system.

We also showed him our interface as well as a number of edited results generated by our system. He was surprised by how efficiently our system could produce useful *rough-cuts* for a wide variety of

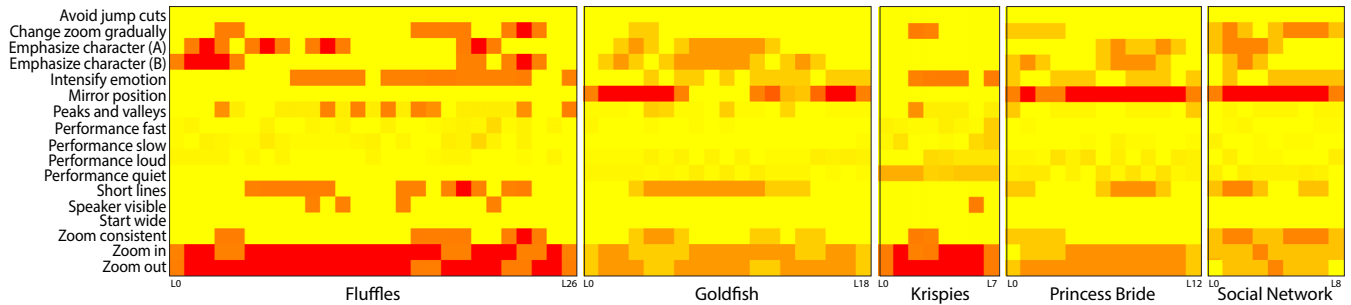


Fig. 10. These matrices show the relative probabilities of each clip according to each of our idioms for five of the professional, hand-edited videos. The probability of each clip (column) is evaluated relative to an optimal alternative, according to each idiom (row). The redness of each cell indicates how strongly an idiom has been violated. We see that *start wide* and *avoid jump cuts* are followed by every clip in all scenes. *Fluffles* and *Krispies* are the only scenes where *speaker visible* is violated - but every clip that violates this idiom satisfies an *emphasize character* idiom, suggesting that a combination of idioms was effectively used.

different editing styles. He told us he would never be able to produce and explore that much stylistic variation without such a system. While he thought that our results were excellent as a starting point and acceptable as is for online video sharing, he also suggested that they would need some refinements before he would treat them as ready for broadcast. Specifically, he would add *L cuts* and *J cuts*, i.e., where the video transition between clips leads or lags behind the audio transition [Bowen 2013]. Similarly, he would clean up alignment errors, such as missing or repeated syllables and words, and locally adjust the spacing between each adjacent pair of clips rather than globally as with our tempo controls. When we told him that he could export the edit produced by our system as an Edit Decision List (EDL) and load it into Avid for such refinement, he saw how it could fit into his existing workflow and significantly speed up his editing process.

To better understand how a human editor makes decisions, we evaluate the probability of clip selections in the professional hand-edited sequences with respect to each of our idioms. The probability of each clip is determined by the emission matrix corresponding to its respective line and the transition matrices that correspond to neighboring lines. Figure 10 visualizes the relative probabilities of each clip in five of our scenes, according to each of our individual idioms. Each row in the figure corresponds to an individual idiom, and each column corresponds to a different clip from a different scene. Probabilities (shown in yellow for high probability, red for low probability) are relative to the most probable alternative clip choice, ensuring that all clips considered optimal by an idiom will appear yellow in the idiom’s corresponding row. For example, in the hand-edit of *Fluffles*, the professional editor fully followed the *avoid jump cuts*, *mirror position*, and *start wide* idioms (row is entirely yellow). For all of the lines where he breaks the *speaker visible* idiom, he follows *emphasize character (B – Stacy)*, suggesting that his edit effectively combines these two idioms with some weight.

8 LIMITATIONS AND FUTURE WORK

While our approach for editing dialogue-driven scenes enables creative exploration of the edit design space, it does have a few limitations that offer directions for future work.

Additional video genres. Our system utilizes a line-based segmentation structure of dialogue-driven scenes to support idiom-based editing. The idioms we describe capture cinematographic conventions specific to editing this type of video. Extending our approach to other types of video (e.g., action scenes, step-by-step instructional video, etc.) would require access to an underlying segment structure (e.g., meaningful actions, each step of the instruction) and a set of idioms for editing this genre of videos.

Discretization of possible cuts. We treat lines of dialogue as the fundamental unit of time at which all editing decisions are made. While our approach produces effective edits in many cases, this limitation makes it impossible to cutaway from the speaker in the middle of a line to show a reaction from another character. Although it may be possible to work with units that are shorter than a complete line (e.g., a phrase, a short fixed time interval, a frame), working at a finer scale could significantly increase the size of the space over which we would have to compute the edit. Defining the appropriate unit is an open challenge.

Continuity. Our system does not explicitly check for continuity errors between adjacent clips. For example, hand gestures or the positions of props can differ between takes and cutting between clips containing such differences can be jarring. In professional filmmaking, skilled performers learn to minimize such gestural differences “hitting their marks” in every take. Alternatively, skilled editors learn to carefully position their cuts to hide such continuity errors. One way to correct continuity errors in our results is to export the EDL into a frame-based editing tool and refine the cut points manually. Automatically tracking differences between takes to identify potential continuity errors is an open problem.

Performance-based labels and idioms. Performances can vary from take to take as performers experiment with different voicings, facial expressions, and acting. However, our system does not currently label many such performance variations, and, therefore, our idioms cannot choose between performances based on these features. However, researchers have developed techniques for tracking pitch, timbre and other vocal variations [Rubin et al. 2015] as well as for

identifying facial expressions [Baltrušaitis et al. 2016]. We are exploring how to incorporate such labels as well as designing idioms that make use of them.

Capture-time editing and feedback. One direction for future work is to use our system to aid filmmakers during the capture process. For example, to run our system in the field so that as the raw takes are captured they are immediately pre-processed, and filmmakers can then generate complete edits within a few hours of collecting the footage. While the bottleneck in our approach is the pre-processing step, we believe it is possible to significantly speed it up. Giving filmmakers the option of editing their footage as they capture it in the field might allow them decide whether shooting another take is necessary.

9 CONCLUSION

Editing dialogue-driven scenes of live-action video is a laborious process with frame-based video editing tools. Even for skilled editors, the effort required to choose clips and arrange them into a sequence that conveys the story makes it difficult to explore the space of editing styles. Our computational video editing system significantly speeds up the editing process by letting editors specify the set of film-editing idioms they would like to enforce and then automatically generating the corresponding edit in a few seconds. This lets editors iteratively adjust the edit by trying different combinations of idioms and testing different parameters to quickly see how their scene would appear using a variety of editing styles. As video becomes an increasingly dominant form of storytelling, we believe that such automated editing systems will be essential for increasing production quality.

10 ACKNOWLEDGMENTS

We thank Lex Davis, Shannon Davis, Jane E, Alex Hall, Zhengtao Jin, Craig Leake, Linda Leake, Lily Liu, Kristen Pilgrim, Daniel Ritchie, Kaitlyn Spees, and Evan Strasnick for performing in our videos. We also thank Fluffles for giving his life so that we may have this paper. Our research is supported by The Brown Institute for Media Innovation.

REFERENCES

Ido Arev, Hyun Soo Park, Yaser Sheikh, Jessica Hodgins, and Ariel Shamir. 2014. Automatic editing of footage from multiple social cameras. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 81.

Daniel Arijon. 1976. *Grammar of the film language*. Focal Press London.

Tadas Baltrušaitis, Peter Robinson, and Louis-Philippe Morency. 2016. OpenFace: an open source facial behavior analysis toolkit. In *IEEE Winter Conference on Applications of Computer Vision*.

Floraine Berthouzoz, Wilmot Li, and Maneesh Agrawala. 2012. Tools for placing cuts and transitions in interview video. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 67.

Steven Bird. 2006. NLTK: The natural language toolkit. In *Proc. of COLING/ACL*. 69–72.

Christopher J Bowen. 2013. *Grammar of the Edit*. CRC Press.

Zachary Byers, Michael Dixon, Kevin Goodier, Cindy M Grimm, and William D Smart. 2003. An autonomous robot photographer. In *Proc. of IROS*, Vol. 3. 2636–2641.

Pei-Yu Chi, Joyce Liu, Jason Linder, Mira Dontcheva, Wilmot Li, and Björn Hartmann. 2013. Democut: Generating concise instructional videos for physical demonstrations. In *Proc. of UIST*. 141–150.

David B Christianson, Sean E Anderson, Li-wei He, David H Salesin, Daniel S Weld, and Michael F Cohen. 1996. Declarative camera control for automatic cinematography. In *AAAI/IAAI*, Vol. 1. 148–155.

David K Elson and Mark O Riedl. 2007. A Lightweight Intelligent Virtual Cinematography System for Machinima Production.. In *AIIDE*. 8–13.

G David Forney. 1973. The Viterbi algorithm. *Proc. IEEE* 61, 3 (1973), 268–278.

Quentin Galvane, Rémi Ronfard, Marc Christie, and Nicolas Szilas. 2014. Narrative-driven camera control for cinematic replay of computer games. In *Proceedings of the Seventh International Conference on Motion in Games*. 109–117.

Quentin Galvane, Rémi Ronfard, Christophe Lino, and Marc Christie. 2015. Continuity editing for 3d animation. In *AAAI Conference on Artificial Intelligence*.

Vineet Gandhi and Rémi Ronfard. 2015. A computational framework for vertical video editing. In *4th Workshop on Intelligent Camera Control, Cinematography and Editing*.

Vineet Gandhi, Remi Ronfard, and Michael Gleicher. 2014. Multi-clip video editing from a single viewpoint. In *Proceedings of the 11th European Conference on Visual Media Production*. 9.

Andreas Girsensohn, John Boreczky, Patrick Chiu, John Doherty, Jonathan Foote, Gene Golovchinsky, Shingo Uchihashi, and Lynn Wilcox. 2000. A semi-automatic approach to home video editing. In *Proc. of UIST*. 81–89.

Li-wei He, Michael F Cohen, and David H Salesin. 1996. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *Proc. of SIGGRAPH*. 217–224.

Rachel Heck, Michael Wallick, and Michael Gleicher. 2007. Virtual videography. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM-CAP)* 3, 1 (2007), 4.

IBM. 2016. IBM Speech to Text Service. <https://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/doc/speech-to-text/>. (2016). Accessed 2016-12-17.

Arnav Jhala and Robert Michael Young. 2005. A discourse planning approach to cinematic camera control for narratives in virtual environments. In *AAAI*, Vol. 5. 307–312.

Niels Joubert, Jane L E, Dan B Goldman, Floraine Berthouzoz, Mike Roberts, James A Landay, and Pat Hanrahan. 2016. Towards a Drone Cinematographer: Guiding Quadrotor Cameras using Visual Composition Principles. *arXiv preprint arXiv:1610.01691* (2016).

Peter Karp and Steven Feiner. 1993. Automated presentation planning of animation using task decomposition with heuristic reasoning. In *Graphics Interface*. 118–118.

Steven Douglas Katz. 1991. *Film directing shot by shot: Visualizing from concept to screen*. Gulf Professional Publishing.

Myung-Jin Kim, Tae-Hoon Song, Seung-Hun Jin, Soon-Mook Jung, Gi-Hoon Go, Key-Ho Kwon, and Jae-Wook Jeon. 2010. Automatically available photographer robot for controlling composition and taking pictures. In *Proc. of IROS*. 6010–6015.

Christophe Lino, Mathieu Chollet, Marc Christie, and Rémi Ronfard. 2011. Computational model of film editing for interactive storytelling. In *International Conference on Interactive Digital Storytelling*. Springer, 305–308.

Qiong Liu, Yong Rui, Anoop Gupta, and Jonathan J Cadiz. 2001. Automating camera management for lecture room environments. In *Proc. of CHI*. 442–449.

Bilal Merabti, Marc Christie, and Kadi Boutatouch. 2015. A Virtual Director Using Hidden Markov Models. In *Computer Graphics Forum*. Wiley Online Library.

W Murch. 2001. In the Blink of an Eye (Revised 2nd Edition). (2001).

Robert Ochshorn and Max Hawkins. 2016. Gentle: A Forced Aligner. <https://lowerquality.com/gentle/>. (2016). Accessed 2016-12-17.

Amy Pavel, Dan B Goldman, Björn Hartmann, and Maneesh Agrawala. 2016. VidCrit: Video-based Asynchronous Video Review. In *Proc. of UIST*. ACM, 517–528.

Amy Pavel, Colorado Reed, Björn Hartmann, and Maneesh Agrawala. 2014. Video Digests: A Browsable, Skimmable Format for Informational Lecture Videos. In *Proc. of UIST*. 573–582.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and others. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, Oct (2011), 2825–2830.

Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.

Abhishek Ranjan, Jeremy Birnholtz, and Ravin Balakrishnan. 2008. Improving meeting capture by applying television production principles with audio and motion detection. In *Proc. of CHI*. 227–236.

April Rider. 2016. For a Few Days More: Screenplay Formatting Guide. <https://www.oscars.org/sites/oscars/files/scriptsample.pdf>. (2016). Accessed 2016-12-17.

Remi Ronfard, Vineet Gandhi, and Laurent Boiron. 2013. The Prose Storyboard Language. In *AAAI Workshop on Intelligent Cinematography and Editing*, Vol. 3.

Steve Rubin, Floraine Berthouzoz, Gautham J Mysore, and Maneesh Agrawala. 2015. Capture-Time Feedback for Recording Scripted Narration. In *Proc. of UIST*. 191–199.

Steve Rubin, Floraine Berthouzoz, Gautham J Mysore, Wilmot Li, and Maneesh Agrawala. 2013. Content-based tools for editing audio stories. In *Proc. of UIST*. 113–122.

Barry Salt. 2011. Reaction time: How to edit movies. *New Review of Film and Television Studies* 9, 3 (2011), 341–357.

Hijung Valentina Shin, Floraine Berthouzoz, Wilmot Li, and Frédo Durand. 2015. Visual Transcripts: Lecture Notes from Blackboard-style Lecture Videos. *ACM Trans. Graph.* 34, 6 (2015), 240:1–240:10.

Hijung Valentina Shin, Wilmot Li, and Frédo Durand. 2016. Dynamic Authoring of Audio with Linked Scripts. In *Proc. of UIST*. 509–516.

- Tim J Smith and John M Henderson. 2008. Edit Blindness: The relationship between attention and global change blindness in dynamic scenes. *Journal of Eye Movement Research* 2, 2 (2008).
- Yoshinao Takemae, Kazuhiro Otsuka, and Naoki Mukawa. 2003. Video Cut Editing Rule Based on Participants' Gaze in Multiparty Conversation. In *Proc. of Multimedia*. 303–306.
- Anh Truong, Floraine Berthouzou, Wilmot Li, and Maneesh Agrawala. 2016. Quickcut: An interactive tool for editing narrated video. In *Proc. of UIST*. 497–507.
- Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. on Information Theory* 13, 2 (1967), 260–269.
- Hui-Yin Wu and Marc Christie. 2016. Analysing Cinematography with Embedded Constrained Patterns. (2016).
- Vilmos Zsombori, Michael Frantzis, Rodrigo Laiola Guimaraes, Marian Florin Ursu, Pablo Cesar, Ian Kegel, Roland Craigie, and Dick CA Bulterman. 2011. Automatic generation of video narratives from shared UGC. In *Proc. of Hypertext*. 325–334.

A APPENDIX: IDIOM IMPLEMENTATIONS

We detail the implementation of the basic film-editing idioms in Table 1 that were not presented in Section 5.1.

Change zoom gradually. To avoid large changes in zoom level we favor transitions in which the difference between zoom levels is at most equal to 1. We set the transition probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{E}(c_i^k) = 1$$

$$\mathcal{A}(c_i^k, c_{i+1}^j) = \begin{cases} 1, & \text{if } d_{zm}(c_i^k, c_{i+1}^j) \leq 1 \\ \frac{1}{d_{zm}(c_i^k, c_{i+1}^j)} & \text{otherwise} \end{cases}$$

$$d_{zm}(c_i^k, c_{i+1}^j) = |\text{zoom}(c_i^k) - \text{zoom}(c_{i+1}^j)|$$

If the difference in zoom is greater than 1, we lower the transition probability based on the inverse of the difference to encourage zoom transitions that are as small as possible.

Emphasize character. To avoid cutting away from an important character during short lines from the other characters, we favor two kinds of transitions; (1) transitions in which the length of both clips is long, and (2) transitions in which one of the clips is short and the important character is in the set of visible speakers for the other clip and both clips are from the same take. Note that the second case is described using two symmetric conditions in the following equations.

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{E}(c_i^k) = 1$$

$$\mathcal{A}(c_i^k, c_{i+1}^j) = \begin{cases} 1 & \text{if } \text{len}(c_i^k) > \phi \text{ and } \text{len}(c_{i+1}^j) > \phi \\ 1 & \text{if } \rho \in \text{svis}(c_i^k) \text{ and } \text{len}(c_{i+1}^j) \leq \phi \text{ and } k = j \\ 1 & \text{if } \rho \in \text{svis}(c_{i+1}^j) \text{ and } \text{len}(c_i^k) \leq \phi \text{ and } k = j \\ \epsilon & \text{otherwise} \end{cases}$$

where ρ is a parameter specifying the the name of the character to emphasize and ϕ is a parameter for setting the time threshold for short lines (1.25 seconds by default).

Mirror position. We encourage transitions between 1-shots of performers that mirror one another's horizontal positions on screen.

We set the transition probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{E}(c_i^k) = 1$$

$$\mathcal{A}(c_i^k, c_{i+1}^j) = \begin{cases} \frac{1}{d_{mir}(c_i^k, c_{i+1}^j)} & \text{if } \text{num}(c_i^k) = \text{num}(c_{i+1}^j) = 1 \\ & \text{and } \text{sign}(\text{mid}(c_i^k)) \neq \text{sign}(\text{mid}(c_{i+1}^j)) \\ \epsilon & \text{otherwise} \end{cases}$$

$$\text{mid}(c_i^k) = \text{midpt} - \text{pos}_x(c_i^k)$$

$$d_{mir}(c_i^k, c_{i+1}^j) = ||\text{mid}(c_i^k) - \text{mid}(c_{i+1}^j)||$$

where midpt is the x-position of the center of the screen.

Peaks and valleys. We encourage close ups when the emotional intensity of lines is high, wide shots when the emotional intensity is low, and medium shots when it is in the middle. We set the emission probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{A}(c_i^k, c_{i+1}^j) = 1$$

$$\mathcal{E}(c_i^k) = \begin{cases} 1 & \text{if } \text{emInt}(c_i^k) > \phi \text{ and } \text{zoom}(c_i^k) \geq 6 \text{ (CU)} \\ 1 & \text{if } \text{emInt}(c_i^k) \leq (1 - \phi) \text{ and } \text{zoom}(c_i^k) \leq 2 \text{ (WS)} \\ 1 & \text{if } (1 - \phi) < \text{emInt}(c_i^k) \leq \phi \text{ and } 2 < \text{zoom}(c_i^k) \leq 6 \\ \epsilon & \text{otherwise} \end{cases}$$

where ϕ is a user-specified parameter for controlling the emotional intensity threshold at which close ups (CU, ECU), medium (MWS, MS, MCU) and wide (EWS, WS) shots are preferred.

Performance fast. We encourage selection of the shortest clip for each line by setting the emission probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{A}(c_i^k, c_{i+1}^j) = 1$$

$$\mathcal{E}(c_i^k) = \frac{1}{\text{len}(c_i^k)}$$

Performance slow. We encourage selection of the longest clip for each line by setting the emission probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{A}(c_i^k, c_{i+1}^j) = 1$$

$$\mathcal{E}(c_i^k) = \text{len}(c_i^k)$$

Performance loud. We encourage selection of the loudest clip for each line by setting the emission probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{A}(c_i^k, c_{i+1}^j) = 1$$

$$\mathcal{E}(c_i^k) = \text{avgvol}(c_i^k)$$

Performance quiet. We encourage selection of the quietest clip for each line by setting the emission probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{A}(c_i^k, c_{i+1}^j) = 1$$

$$\mathcal{E}(c_i^k) = \frac{1}{\text{avgvol}(c_i^k)}$$

Short lines. Like the emphasize character idiom, our goal is to avoid cutting away to a new take on short lines. Thus, if one of the clips in a transition is short, we encourage a transition that

stays on the same take as the other clip as long as the speaker is visible in the other clip. We set the transition probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{E}(c_i^k) = 1$$

$$\mathcal{A}(c_i^k, c_{i+1}^j) = \begin{cases} 1 & \text{if } \text{len}(c_i^k) > \phi \text{ and } \text{len}(c_{i+1}^j) > \phi \\ 1 & \text{if } \text{spkr}(l_i) \in \text{svis}(c_i^k) \text{ and } \text{len}(c_{i+1}^j) \leq \phi \text{ and } k = j \\ 1 & \text{if } \text{spkr}(l_{i+1}) \in \text{svis}(c_{i+1}^j) \text{ and } \text{len}(c_i^k) \leq \phi \text{ and } k = j \\ \epsilon & \text{otherwise} \end{cases}$$

where ϕ is a parameter for setting the time threshold for short lines (1.25 seconds by default).

Zoom consistent. We encourage the use of a consistent zoom level throughout the scene by setting the transition probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{E}(c_i^k) = 1$$

$$\mathcal{A}(c_i^k, c_{i+1}^j) = \begin{cases} 1 & \text{if } \text{zoom}(c_{i+1}^j) = \text{zoom}(c_i^k) \\ \epsilon & \text{otherwise} \end{cases}$$

Zoom in/out. We encourage either zooming in or zooming out by setting the transition probabilities as

$$\mathcal{B}(c_0^k) = 1, \quad \mathcal{E}(c_i^k) = 1$$

$$\mathcal{A}(c_i^k, c_{i+1}^j) = \begin{cases} 2^\phi & \text{if } \text{zoom}(c_{i+1}^j) > \text{zoom}(c_i^k) \\ 2 & \text{if } \text{zoom}(c_{i+1}^j) = \text{zoom}(c_i^k) \\ 2^\phi & \text{if } \text{zoom}(c_{i+1}^j) < \text{zoom}(c_i^k) \end{cases}$$

where ϕ is a user-specified parameter that controls the aggressiveness of the zoom. Positive values of ϕ encourage zooming in while negative values of ϕ encourage zooming out.