Temporal Segmentation of Creative Live Streams

C. Ailie Fraser[1,2], **Joy O. Kim**[1], **Hijung Valentina Shin**[1], **Joel Brandt**[1], **Mira Dontcheva**[1]
Adobe Research[1]; Design Lab, UC San Diego[2]
cafraser@ucsd.edu, {joykim, vshin, jobrandt, mirad}@adobe.com

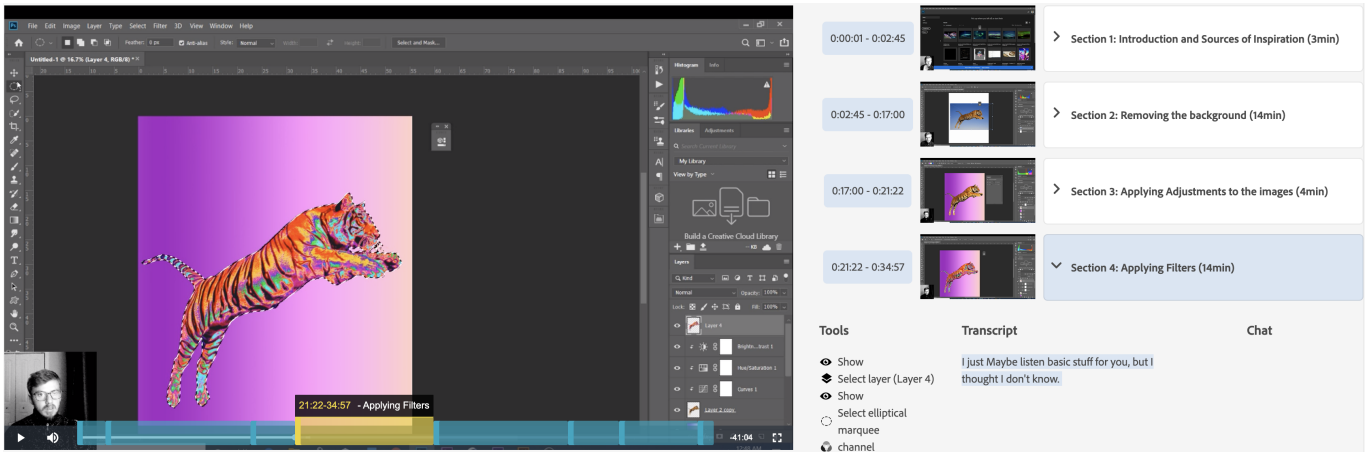

Figure 1. We present a streamer-in-the-loop approach for creating a table of contents for creative live stream videos. We pair automatic segmentation via command logs and audio transcripts with streamer labeling. We built a prototype interface to evaluate the approach with streamers, shown above.

ABSTRACT

Many artists broadcast their creative process through live streaming platforms like Twitch and YouTube, and people often watch archives of these broadcasts later for learning and inspiration. Unfortunately, because live stream videos are often multiple hours long and hard to skim and browse, few can leverage the wealth of knowledge hidden in these archives. We present an approach for automatic temporal segmentation of creative live stream videos. Using an audio transcript and a log of software usage, the system segments the video into sections that the artist can optionally label with meaningful titles. We evaluate this approach by gathering feedback from expert streamers and comparing automatic segmentations to those made by viewers. We find that, while there is no one "correct" way to segment a live stream, our automatic method performs similarly to viewers, and streamers find it useful for navigating their streams after making slight adjustments and adding section titles.

Author Keywords

live streaming; creativity; video segmentation

CHI '20, April 25–30, 2020, Honolulu, HI, USA.

ACM ISBN 978-1-4503-6708-0/20/04 ...$15.00.
http://dx.doi.org/10.1145/3313831.3376437

CCS Concepts

•**Information systems** → **Multimedia streaming;**

INTRODUCTION

Creative live streams can be a valuable learning resource, as they showcase not only the step-by-step process required to accomplish a task, but also the mistakes, decisions, and ideas that happen along the way [7]. Although watching a stream while it is live brings a host of benefits (*e.g.,* real-time interaction with the streamer and other viewers), many can only watch a stream when it is archived after the broadcast is over. Unfortunately, most creative live stream videos are very long (3-4 hours on average [7]) and by nature are not edited in any way. As a result, navigating these videos to catch up on a favorite artist, pick up some helpful tips, or learn a technique is challenging [7, 15]. Unlike tutorial videos which are often heavily produced and much shorter, live stream videos include many sections that a viewer may want to skip, such as conversation between the streamer and audience or repetitive actions that are not interesting to watch.

How might we make archived live stream videos easier to navigate? Prior work has shown that additional metadata, such as transcripts, thumbnails, and usage logs, can provide helpful ways for viewers to index into videos [9, 10, 11, 16, 19, 20, 22]. But for multi-hour live streams, organizing this information in some meaningful way is critical. Some video authors manually create a table of contents by adding labeled timestamps to their video's description, and some platforms (*e.g.,* `skillshare.com`, `linkedin.com/learning`) even require authors to divide their

videos into labeled sections. Despite the benefits of providing structure, for artists that stream for multiple hours, multiple times a week, going back through all of their content to create a table of contents can be prohibitively time-consuming.

This paper proposes a semi-automatic method for creating a table of contents for creative live stream videos. The proposed approach automatically segments videos into sections and asks the streamer to label each section with a meaningful title. To segment each video, our approach leverages a transcript of the streamer's narration and a log of their activity, combining them in a novel segmentation algorithm. This streamer-in-the-loop approach was informed by formative interviews with streamers and viewers, where we found that viewers want a quicker way to browse live stream videos based on steps in the creative process, and streamers are not willing to spend a lot of time segmenting their own streams. We also found from an online study that different viewers segment the same video differently, which suggests that there is no single "right" way to segment a creative live stream.

We demonstrate and evaluate our approach in the context of live streaming with the popular creative software Adobe Photoshop, which is used for a variety of creative tasks such as painting, illustration, design, and photo manipulation. We collected feedback on our approach from two artists who stream on Behance (`behance.net`), a social network for creative professionals. These streamers found the automatically-generated table of contents useful but wanted to make changes.

A comparison of our segmentation algorithm with segmentations done by viewers found that for some streaming styles, the algorithm generates sections as consistent as those made by different viewers, while for others it is less effective. We found that the algorithm is most consistent with viewers when both transcript and application usage are available. Transcripts are most useful when streamers describe their process while they work, and usage logs are most useful for tasks that use a variety of tools and commands.

To summarize, this paper makes the following contributions:

1. formative studies showing the potential benefits and challenges of segmenting creative live stream videos,
2. an algorithm for automatically segmenting creative live stream videos into sections that leverages multiple data streams when available,
3. feedback from streamers showing that our automatic segmentation helped them segment their videos into meaningful sections, and
4. an evaluation comparing this algorithm to segmentations by viewers that sheds light on when the algorithm works well and when it doesn't.

We discuss how our approach can generalize beyond Photoshop to other creative software, or even non-software instructional videos. As online video corpora continue to grow, there will be increased need for navigating videos at a higher level; this paper demonstrates the complexity of this challenge and proposes one method for making it possible.

RELATED WORK

Watching and Navigating Live Stream Videos

Live stream viewership has exploded in recent years, with major streaming platforms logging billions of hours of viewership each year [18]. Creative live streaming (*i.e.,* live broadcasting of an artist working on a creative project such as illustration or crafting) is similarly growing in popularity. There are many online platforms dedicated specifically to creative live streaming (*e.g.,* Behance (`behance.net/live`), Picarto (`picarto.tv`), Pixiv Sketch (`sketch.pixiv.net/lives`)), and large platforms such as Twitch (`twitch.tv`) routinely have thousands of concurrent viewers across their creative channels [7]. Creative live streams can take on different forms depending on the streamer's goal; *e.g.,* streamers aiming to teach are more likely to talk about what they are doing, whereas other streamers may not talk at all, or may focus on socializing with viewers [7]. In general, live streamed videos differ from other types of videos in that they are unedited and often less planned out; they can include unexpected moments like mistakes, confusion, and responses to real-time questions from viewers [5, 15].

A challenge with watching live streams, especially when one's goal is to learn from them, is the overwhelming amount of information [7, 14, 15]. Prior work has proposed methods for summarizing the content of knowledge-sharing streams [15] and video game streams [13]. StreamWiki [15] enables viewers to collaboratively generate a summary in real-time that helps others understand a stream's content both during and after. Helpstone [13] uses a log of game actions to show context about the streamer's gameplay and summarize what has happened so far. This paper takes an approach similar to Helpstone by using application usage to summarize a stream.

Video Navigation and Segmentation

Prior work has proposed methods for segmenting videos using application usage logs [9] and audio transcripts [19, 20], but not both together. This paper explores how these methods can apply to live streamed content, and argues that using both data sources together produces better results than either on its own.

Segmentation based on application usage

Application usage logs can reveal the type of task a user is working on and when they switch to something new, based on the tools, commands, and settings they use in their software. Chronicle [9] separates application usage logs into sections by save events, as users often save their work when they complete a subtask. Chen *et al.* [3] found that creating new layers, switching to different tools, and adjusting parameters can all be indicators of switching tasks. We build on this work and consider save events, layer selection, and the distribution of commands over time. However, usage data only reveals part of the story. Streamers often explain what they will do before they do it, and depending on how application telemetry is instrumented, usage logs may not capture all activity. Therefore, we rely not only on usage data, but audio transcripts as well.

Segmentation based on audio transcripts

Prior work has demonstrated how audio transcripts can be used to segment and navigate lecture videos [10, 20] and movies [19]. More broadly, segmenting a body of text by topic

is a longstanding problem in the natural language processing community, and many different methods exist. Pavel *et al.* [20] found that for lecture videos, Bayesian topic segmentation [4] was most successful due to its incorporation of "cue phrases"; *i.e.*, common keywords or phrases that indicate transitions, such as "now" or "next". Early experimentation with applying this method out-of-the-box to creative live stream transcripts did not show promising results, likely because streamers often rapidly switch between talking about their work and other unrelated topics guided by the live chat, all while working on the same task. In addition, any transcript-only method misses activity that the streamer does not explicitly narrate, and streamers often work in silence for long periods of time. Inspired by Bayesian topic segmentation, our algorithm does incorporate cue phrases, as we found that streamers often talk about their process when they transition between subtasks.

FORMATIVE STUDIES

To understand what it would mean to segment creative live stream recordings in a meaningful way and determine possible use cases for segmented videos, we conducted interviews with 3 creative streamers and 7 creative live stream viewers, and an online study comparing how different viewers segment the same videos. We found that viewers want a quicker way to browse live stream videos, but manually segmenting is a difficult task for streamers, and different people segment the same video in different ways.

Interviews With Streamers

The streamers (SP1-SP3) were recruited from a popular online creative community (Behance), ranged from 24 to 36 years old (all men), and reported streaming at least once a week. Two participants were new to streaming, and one was a streaming expert who in the past had streamed as his full time job. The streamers' domains of expertise included illustration, photography, and design. Each interview took place over video conferencing and lasted 30 minutes.

Method

The streamers first participated in a semi-structured interview where they were asked about their creative work, their experience streaming, their thoughts on live stream archives, and how they interact with their viewers. Then they were shown design mockups of a video player interface (see Figure 2b) and asked about different strategies for segmenting their streams and how a segmentation would affect their viewers' experiences. Our goal was to assess which metadata from the stream the streamers would find most useful and want to show to their audience. Last, they were asked to do a think-aloud activity where they walked the interviewer through one of their recent streams and described the parts of their creative process.

Results

The streamers mentioned several motivations for streaming their creative work. One stemmed from a desire to grow a relationship with their audience and build a community of fans, corroborating previous research [7]. The streamers also mentioned wanting to share parts of their process outside of just the procedural steps; SP1 and SP2 both explained wanting to share the *why* behind their artwork in addition to the *how*.

When asked about the idea of a table of contents for live stream videos, streamers responded positively, expecting that it would make their live streamed content easier for viewers to navigate. However, streamers did not want to manually create sections, as it would take too long and they were unsure whether the time spent would be worth it:

> *"the amount of time I spend on creating a table of contents would depend on whether I can monetize my time."*
> —SP1

The streamers were interested in the idea of automatically generating a table of contents. SP1 said that he would want the ability to edit it in case it was not exactly what he wanted. When asked what good sections for their videos might look like, all three streamers expressed some uncertainty. SP1 said that his more instructional-focused streams could likely be broken up based on the steps they show, but some of his other videos just involve him doing one type of task the whole time, such as inking a drawing. SP3 said he tends to bounce around a lot between different subtasks which could make higher-level sections hard to identify. Even when we asked streamers what the ideal sections for a particular stream of theirs might be, answers were not obvious. This strongly suggests that multiple segmentation schemes may be necessary depending on an individual stream's content and structure.

Interviews with Viewers

The viewers (VP1-VP7) were recruited from email distribution lists at a large software company and through posts on Twitter, and ranged from 21 to 41 years old. The viewers reported watching creative streams at least once a week, mostly in the domain of illustration. All interviews took place either in-person or remotely via video conferencing, based on each viewer's preference. Each study session lasted approximately 60 minutes and participants were compensated with a $25 USD gift card for their time.

Method

The viewers also first participated in a semi-structured interview; they were asked about their stream-watching behavior, whether they ever watch live stream archives or clips, and what they think makes live streams good or bad experiences. Next, they participated in a think-aloud activity where they were asked to walk the interviewer through a video of a stream they had recently watched (if they didn't have a link to a video ready, they were provided with one chosen by the interviewer) and describe an outline of the stream content. The interviewer asked the viewer about the reasoning behind their outline and observed the viewer as they interacted with the video recording during this task. Last, viewers were shown mockups of four different schemes we considered for segmenting a live stream video (Figure 2): creative process, working/talking, talking type, and info highlights. These schemes were loosely inspired by the four creative streaming types noted in prior work [7]. Viewers were asked about their impressions and which schemes they preferred.

Results

We found that viewers used several signals to denote whether a part of the video was of interest. For example, in the stream

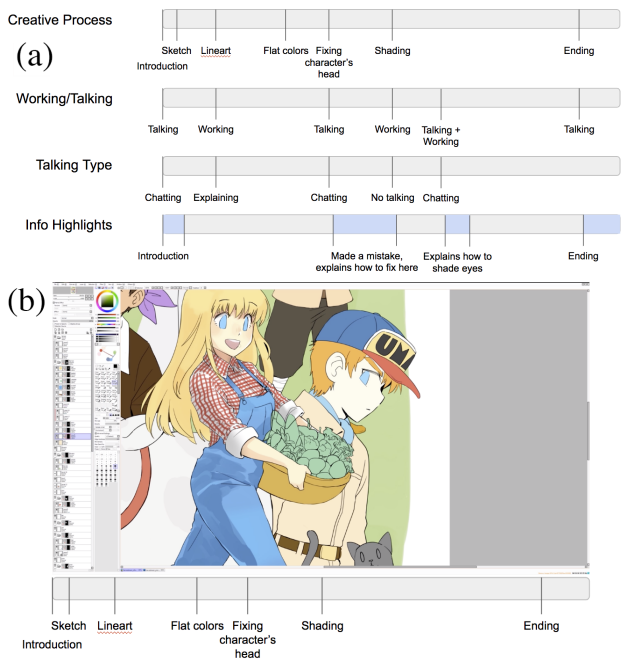

Figure 2. a) Mockups showing four possible ways to segment a live stream that were shown to formative study participants. (b) A detailed mockup showing how a segmentation might appear below a video player.

walkthrough task, most viewers turned on audio right away. When asked why, viewers said they were expecting explanation or context about what is being done and wanted to use the narration as a way to orient themselves in the video:

> *"Commentary ... sometimes really helps with backstory, like, why they're drawing this ... so I kind of wanted to see if they were gonna talk about that, or any other commentary I might need to know."* —VP4

Viewers also used video thumbnails and timeline scrubbing to compare changes between video frames to find moments of interest. This corroborated with how viewers described their stream-watching behavior in general, where they often leave a stream on in the background and look over to see if something of interest is happening:

> *"I ... don't tend to spend a lot of time watching the stream. People spend a lot of time doing the same thing ... So I tend to spend like 10 minutes or something getting some technique ... that I can adopt."* —VP7

Surprisingly, most viewers stated they currently didn't watch archived streams, because they are too long and tedious to navigate. Viewers did say they watch speedpaints (sped-up recordings of creative processes) and art tutorials on platforms like YouTube. Their reasons for watching these types of videos over live stream archives were that high visual change is easier to see, they are more efficient to learn from, and it is easy to repeat portions of the video they want to watch more carefully:

> *"I feel like with speedpaints ... you see everything coming together in a short amount of time so you can grasp what is being done easier ... with [live streams], obviously they're drawing in like real time so it's slower ... I'll click away and come back later ... I guess because I'm impatient and I just want to see start to finish quicker and I can apply it to my own drawing sooner as well."* —VP6

If live stream recordings were similarly easy to browse and navigate, they could afford similar advantages for learning, and thus would likely be watched more.

In the mockup task, viewers overwhelmingly preferred *creative process* and *info highlights*, explaining that they liked having access to a high-level view of the process. For example, VP7 stated that while watching streams, they often try to compare their own process with the streamer's in order to adopt new techniques or drop current ones. Viewers also desired the ability to skip directly to the parts of the process that were interesting to them with respect to their current learning goals:

> *The [creative process mockup] is also nice because ... if you want to see the whole process but then maybe you're like, you know, I already know sketching lineart, I want to see the shading, I can just skip to the shading.* —VP5

However, viewers also pointed out that not all streams show a holistic process; a process may be broken into several separate streams or there may even be gaps where some work was done offline. In these cases, viewers favored highlights as unique and specific signals of what small portions of the video may be worth watching. This supports our findings from the streamers, who also anticipated needing different segmentation schemes for streams depending on their format. Viewers preferred the talking mockups least, explaining that they would place less emphasis on re-watching the streamer talk in a social manner (at least in the context of creative streams) because they would be missing the experience of participating live.

Collecting and Comparing Viewer Segmentations

Since our interviews suggested that segmenting a creative live stream may not have a clear solution, we conducted an online study where human coders segmented a sample set of 23 videos. We gathered segmentations from multiple coders for each video to see how consistent different people are with each other. The coders represent potential viewers of these streams, since the goal of a segmentation is to aid viewers in browsing and navigating videos.

Data Collection

For our dataset of creative live stream videos, we selected four streamers on Behance that showcase different types of creative work commonly done in Photoshop: graphic design (S1), comic art (S2), digital illustration (S3), and image compositing (S4). For each streamer, we chose 5-7 of their publicly available videos on Behance that lie between 30 minutes and 2 hours in length, and for which we had complete application usage logs and transcripts. This produced a set of 23 videos.

We recruited 158 coders with self-reported intermediate to expert experience with Adobe Photoshop to segment each video (an average of 6.9 coders segmented each video). Coders were recruited from `usertesting.com`, an online worker platform. Coders were given up to 20 minutes to generate a segmentation for one video in our input set; this time limit was enforced to help ensure that coders would generate outlines at similar levels of granularity. Instructions were open-ended; we asked

coders to segment videos into "meaningful sections", allowing them to segment in whatever way they thought fit. We removed segmentations that had 3 or fewer segments, spanned less than half the video, or had obviously spam/gibberish section names.

Analysis

To quantify how similar different coders' segmentations were, we calculated the boundary similarity [6] between each pair of segmentations for each video. Boundary similarity computes a score representing the similarity between two segmentations based on boundary edit distance to differentiate between full and near misses. Boundary similarity scores range between 0 and 1, where a score of 1 means the two segmentations are exactly the same, and a score of 0 means they are completely different. In contrast to window-based metrics (which are calculated by comparing one manual segmentation to some ground-truth segmentation) [21], boundary similarity is symmetric, allowing its use to compute pairwise similarity means for more than two manual segmentations. This was suitable for our case since we had more than two coders segment each video and no ground-truth segmentation. We computed boundary similarity scores using a boundary edit distance window of 120 seconds, which is the maximum distance that two boundaries may span to be considered a near miss (as opposed to a full miss, which is penalized more strictly).

Results

Coders generated between 4 and 41 sections for each video, with an average of 11 sections per video. Sections ranged from 10 seconds to 1h40m long, with an average length of 6 minutes.

Table 1 shows a summary of the boundary similarity scores for each streamer's videos, which indicate how consistent different coders were to each other. Overall, coders were not very consistent; the average boundary similarity score across all streamers was 0.255, and the highest boundary similarity score was 0.679. This supports our interview findings that segmenting creative live stream videos is not straightforward, and there may be more than one "correct" way to do so.

Takeaways

Our formative studies confirmed the need for better ways to consume archived creative live stream videos, and suggested that segmenting live streams is not a straightforward task. More specifically, we learned that streamers:

- see value in a table of contents but are not willing to spend time making one manually, and
- have a hard time anticipating the best way to segment a stream.

Streamer	# scores	Mean	SD	Median
S1	127	0.243	0.108	0.236
S2	136	0.268	0.139	0.240
S3	92	0.322	0.140	0.310
S4	132	0.208	0.110	0.186

Table 1. Summary of similarity scores for pairs of coder segmentations (1 = identical, 0 = completely different). # scores refers to the number of similarity scores (*i.e.,* pairs of segmentations) for each streamer.

Meanwhile, viewers:

- see value in an overview of the creative process that allows them to skip to relevant parts of the video,
- find speedpaints and how-to videos easier to navigate with current interfaces, and
- segment the same video in different ways, suggesting there is no one "right" way to segment creative live streams.

Together these findings point to the need for an approach that requires little effort from streamers and offers viewers a meaningful way to navigate. To satisfy both of these goals, we decided to pursue a streamer-in-the-loop approach for creating a table of contents: we designed an algorithm to automatically segment videos into sections that the streamer can then label.

SEGMENTATION ALGORITHM

The goal of this algorithm is to obtain a temporal segmentation of the entire stream, where each section contains a meaningful step in the art process. We use a transcript of the streamer's audio and a log of their software usage to determine the optimal boundaries for sections, combining both input sources in a novel algorithm. We designed the algorithm in such a way that it could be extended to include additional input sources, such as visual data. We chose a heuristic approach over a data-driven approach because our formative work showed that there is no single "ground truth" way to segment a video.

Video Metadata

Transcripts

During live streams, streamers talk casually to the viewers about various topics. Each streamer has a characteristic style. For example, some streamers talk throughout the stream while others talk sparingly; some streamers talk only about their work while others also chat about unrelated topics as they work. Regardless of their style, in instructional live streams, streamers usually include some explanation of the major steps or techniques they are using. These explanations are good pointers to the main parts of the art process.

We obtain the transcript of the stream using a speech-to-text engine [1] that splits the transcript into sentences and includes the start and end time of each sentence.

Application usage logs

The log of a streamer's software usage contains a lot of information about their process [3, 9]. Some commands indicate a transition between different types of tasks. Some groups of commands are used together to achieve a single task.

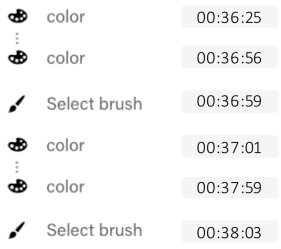

Figure 3. An example of the application usage log obtained by an Adobe Photoshop plugin.

We obtain the streamer's application usage log through an Adobe Photoshop plugin that records user actions, which the streamer enables during the live stream (Figure 3). Each event in the log comprises the name of the command (*e.g.,* `select layer`, `mask`), the

timestamp for when the command was used in the stream, and other command-specific details (*e.g.,* the ID of the layer that was selected). Similar data can also be obtained through computer vision techniques [2] or accessibility methods [8]. We categorize commands as either *navigational* or *editing*. Navigational commands do not alter the document but are used to navigate and show different parts (*e.g.,* `zoom`, `hide layer`). Editing commands, on the other hand, modify the document (*e.g.,* `brush`, `create layer`).

Pre-processing

Identifying candidate section boundaries

First, we identify all candidate section boundaries, t_i, by taking the union of all the event timestamps in the usage log and the beginning and end of each sentence in the transcript (Figure 4). Since we do not want section boundaries to occur in the middle of a streamer's sentence, we disregard any usage event that occurs mid-sentence. We designate the interval between two contiguous candidate boundaries t_i and t_{i+1} as $p_i = [t_i, t_{i+1})$. Many streamers display a *starting soon* screen at the beginning of their stream while they check their setup and wait for viewers to join. Since this part of the stream does not include any information, we do not consider it as part of any section. Instead, we begin the first interval p_0 at the start of the first candidate boundary time t_0.

Intro and Outro sections

We observed that, like tutorials [11], most live streams include an *intro* at the beginning and an *outro* at the end. The intro is the period before the streamer starts working, where they greet viewers, introduce their project, and show any preparations they have done. The outro consists of them summarizing what they did, advertising their next stream, and saying goodbye.

To identify the intro and outro sections, we look for the first and last editing commands. We exclude navigational commands because streamers often use them to show their preparatory work (*e.g.,* reference images or preliminary sketches), or to review parts of their final artwork. The intro starts at t_0 and ends at the first editing command. The outro starts after the last editing command and lasts until the end of the stream. If either section is shorter than 30 seconds, we do not split it into a separate section, but instead include it as part of the first or last main section. This is because extremely short sections are unlikely to aid navigation.

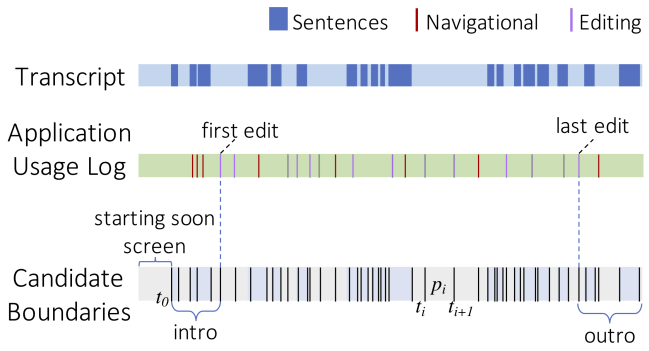

Figure 4. Pre-processing step of our segmentation algorithm. First, we identify candidate section boundaries by taking the union of the sentence boundaries and command timestamps, and removing mid-sentence boundaries. Then, we identify *intro* and *outro* sections by finding the first and last editing commands.

Dynamic Programming Segmentation

The problem of segmenting a live stream video into meaningful sections is analogous to the line-breaking problem, *i.e.,* arranging the words of a paragraph into lines. In both cases, we want to segment a sequence of discrete units (intervals or words) into an optimal set of groups (sections or lines) defined by some scoring function over candidate sections or lines. We first explain the high-level structure of our algorithm that is based on Knuth and Plass' optimal line-breaking algorithm [12], then we describe the scoring function in detail.

Algorithm overview

Given a sequence of n intervals $P = \{p_0, ..., p_{n-1}\}$, we find the optimal set of boundaries that segment the intervals into sections. Our algorithm processes the intervals in order, and for each p_i computes and records the optimal set of sections S_i formed by all intervals up to and including p_i, along with the total score $E(S_i)$ for this partial solution. To determine the optimal partial solution for interval $p_i = [t_i, t_{i+1})$, the algorithm considers each previous candidate boundary t_j, where $j \leq i$, and evaluates two possible ways of creating a section that includes intervals $P_{ji} = \{p_j, \ldots, p_i\}$: (1) Create a new section P_{ji}, or (2) merge P_{ji} with the last section in S_{j-1}. After considering both possibilities for all previous candidate boundaries t_j, we choose the segmentation with the highest total score, $E(S_i)$. Once the algorithm iterates through all intervals, S_{n-1} holds the optimal set of sections for the entire stream.

Scoring function

The algorithm described above requires a scoring function that evaluates the quality of a section formed by merging a set of contiguous intervals. The total score of a segmentation S is defined as the average score of its constituent sections, $s \in S$:

$$E(S) = \frac{1}{|S|} \sum_{s \in S} e(s)$$

where $|S|$ is the number of sections in S and each s is a set of contiguous intervals $\{p_k, ..., p_{k+m}\}$.

The scoring function for a section s takes into account four factors: (1) the duration of the section, (2) transitional commands in the application usage log, (3) coherence of the commands used, and (4) transitional phrases in the transcript.

(1) Duration of a section: Very short or very long sections are less helpful for navigation. We penalize extremely short (< 1 minute) or extremely long sections (> 10 minutes). We include a linear dropoff from 1 to 0 for sections less than 1 minute or greater than 10 minutes rather than having a strict cut-off length because these length requirements are approximate goals (a 59-second section should not be penalized significantly more than a 60-second section). However, we do have a strict cut-off for sections shorter than 30 seconds, because we believe those are too short to be useful.

$$e_{\text{length}}(s) = \begin{cases} -\infty & dur(s) \leq 0.5 \\ 2dur(s) - 0.5 & 0.5 < dur(s) < 1 \\ 11 - dur(s) & dur(s) > 10 \\ 1 & \text{otherwise} \end{cases}$$

where $dur(s)$ is the duration of section s in minutes.

(2) Transitional commands: Prior work [3, 9] found that certain types of application commands indicate that users are transitioning between tasks. We build off Chronicle [9]'s approach, using `save` commands to indicate the end of subtasks; and Chen et al. [3]'s approach, using `layer selection` commands to indicate the start of subtasks. Like Chronicle, we prefer `save`s that are followed by a longer gap until the next command. We similarly prefer `layer selection`s with longer time until the next `layer selection`, as this implies the user worked on the first selected layer for longer.

For each `save` command, c^i_{save}, we compute its importance score $I(c^i_{\text{save}})$ by considering the time gap between that command and the next command, $gap(c^i_{\text{save}})$. The longer the gap, the more important the save.

$$I(c^i_{\text{save}}) = \frac{gap(c^i_{\text{save}})}{\max\limits_{c_{\text{save}} \in \text{stream}} gap(c_{\text{save}})}$$

where the denominator is the maximum *gap* of all the `save` commands in the stream, and is included for normalization.

We prioritize sections that have an important `save` command near the end of the section. The save score is defined as:

$$e_{\text{save}}(s) = \frac{t(c_{\text{last\_save}}(s)) - t_{\text{start}}(s)}{dur(s)} \times I(c_{\text{last\_save}}(s))$$

where $c_{\text{last\_save}}(s)$ is the last `save` command in s and $t(c_{\text{last\_save}}(s))$ is its timestamp. $t_{\text{start}}(s)$ refers to the start time of section s. If there are no `save` commands in s, $e_{\text{save}}(s) = 0$.

Analogously, for a `layer selection` command c^i_{layer}, we compute its importance by considering the time gap between that command and the next `layer selection` command for a different layer, $gap(c^i_{\text{layer}})$. The longer this time, the more important the layer selection.

$$I(c^i_{\text{layer}}) = \frac{gap(c^i_{\text{layer}})}{\max\limits_{c_{\text{layer}} \in \text{stream}} gap(c_{\text{layer}})}$$

To avoid rewarding extremely short sections, we only consider layer selection events for which $gap(c^i_{\text{layer}}) \geq 30$ seconds.

We prioritize sections that have an important `layer selection` command near the beginning of the section.

$$e_{\text{layer}}(s) = \frac{t_{\text{end}}(s) - t(c_{\text{first\_layer}}(s))}{dur(s)} \times I(c_{\text{first\_layer}}(s))$$

where $c_{\text{first\_layer}}(s)$ is the first `layer selection` command in s and $t(c_{\text{first\_layer}}(s))$ is its timestamp. $t_{\text{end}}(s)$ refers to the end time of s. Again, if there are no `layer selection` commands in s, $e_{\text{layer}}(s) = 0$.

(3) Command coherence: Certain sets of commands are frequently used together for a task [3], for example the `color` and `select brush` commands for illustration (Figure 3). Separating such coherent sets of commands from other sets of commands can help segment the stream into meaningful tasks.

To estimate how coherent different commands are, we count the number of times a pair of commands appears adjacent to each other in the application usage log. The coherence, M, of a pair of commands c_a and c_b is defined as:

$$M(c_a, c_b) = \frac{\text{\# times } c_a \text{ occurs immediately before } c_b}{\text{total \# times } c_a \text{ occurs before any other command}}$$

where the denominator is used to normalize M to a range between 0 and 1. The coherence of a command with itself $M(c_a, c_a)$ is defined as 1.

We favor boundaries that lie between less-coherent commands. The command-coherence score for a section s_i is defined as:

$$e_{\text{commands}}(s_i) = 1 - M(c_{last}(s_{i-1}), c_{first}(s_i))$$

where $c_{last}(s_{i-1})$ refers to the last command in the previous section, s_{i-1}, and $c_{first}(s_i)$ refers to the first command in s_i. If none of the last three intervals in s_{i-1} have commands *and* none of the first three intervals in s_i have commands, $e_{\text{commands}}(s_i)$ is defined as 1. If only one of those two conditions hold, $e_{\text{commands}}(s_i)$ is defined as 0. This prioritizes boundaries between unrelated commands, or between a period of no application use and a period of application use.

(4) Transcript semantics: Streamers often explain the important steps in their art process. While the specific contents of these explanations vary widely and are often punctuated with other topics, key transitional phrases that indicate the start or end of a task can provide cues for segmentation [4]. For example, phrases such as "start" or "next" indicate the beginning of a new step, while phrases such as "done" or "that's all' indicate an end.

To determine the start and end phrase scores e_{start} and e_{end} for a candidate section s, we look for occurrences of (pre-defined) start and end phrases in s. We favor sections with a start phrase near the beginning and an end phrase near the end:

$$e_{\text{start}}(s) = \frac{t_{\text{end}}(s) - t_{\text{start\_phrase}}(s)}{dur(s)}$$

$$e_{\text{end}}(s) = \frac{t_{\text{end\_phrase}}(s) - t_{\text{start}}(s)}{dur(s)}$$

where $t_{\text{start\_phrase}}(s)$ is the time of the last start phrase in s, and $t_{\text{end\_phrase}}(s)$ is the time of the first end phrase in s. If there are no start or end phrases in s, the corresponding score is 0.

Final Scoring Function: We define the final scoring function for a section s as the weighted sum of the component scores:

$$\begin{aligned} e(s) = {} & \alpha_{\text{length}} e_{\text{length}}(s) \\ & + \alpha_{\text{save}} e_{\text{save}}(s) + \alpha_{\text{layer}} e_{\text{layer}}(s) \\ & + \alpha_{\text{commands}} e_{\text{commands}}(s) \\ & + \alpha_{\text{start}} e_{\text{start}}(s) + \alpha_{\text{end}} e_{\text{end}}(s) \end{aligned}$$

In our implementation, we use $\alpha_{\text{length}} = 5$, $\alpha_{\text{saves}} = 5$, $\alpha_{\text{layer}} = 3$, $\alpha_{\text{commands}} = 1$, $\alpha_{\text{start}} = 2$, and $\alpha_{\text{end}} = 2$, which we find through experimentation. However, as we explain in the Discussion, the optimal weights may depend on the type of stream and the streamer's individual style.

SEGMENTATION RESULTS

We ran our algorithm on the 23 videos for which we collected viewer segmentations in the Formative Study. Table 4 in the Appendix shows a detailed summary of its output. Our algorithm generated between 6 and 26 sections for each video, with an average of 12 sections per video. Sections ranged from 32 seconds to 13 minutes in length, with an average length of 5.5 minutes. 22/23 videos produced an Intro section, and 18/23 videos produced an Outro section.

In the following sections, we report on feedback and section labels we gathered from two of these streamers, and compare our algorithm's segmentation to viewer-generated segmentations.

FEEDBACK & LABELS FROM EXPERT STREAMERS

To investigate whether our segmentation algorithm is a good fit for the needs of artists and assess how easy it would be for them to add labels to the generated sections, we interviewed the first 2 streamers from our set of videos, S1 and S2 (both men). We showed them our segmentation in the context of a prototype interface (Figure 1) for three of their own videos, and asked them to label the sections and optionally change section timings. Our prototype interface presents a collapsed table of contents listing the sections with generic labels (Section 1, Section 2, etc.) and thumbnails showing the first frame of each section. A section can be expanded to show the application usage, transcript, and chat messages from that part of the video. The prototype was designed primarily to evaluate the algorithm, rather than as a novel interface.

Method

We recruited the streamers from Behance where they stream. Each interview lasted approximately 30-45 minutes and had three parts. First, we discussed their background and why they started live streaming their creative process. We then showed them the prototype interface and asked for their feedback on whether the automatically generated table of contents would be useful to their viewers, and discussed the segmentation, thumbnails, transcript, usage log, and chat. Last, we asked them to go through one of their videos, label each automatically generated section, and make any desired changes to the timing of the sections. Following the interview, we sent them two more of their own videos with automatically generated tables of contents and asked them to do the same task on their own. We compensated them $60 USD for their time.

Results

Both streamers found the automatic segmentation useful; they felt it made it easier to navigate the long videos and helped to organize all of the metadata associated with a long stream. Additionally, the thumbnails gave some meaning to the sections even in the absence of descriptive labels.

When asked what would make the table of contents more useful, both streamers said they wanted to add labels and edit the sections, confirming our streamer-in-the-loop approach. The two artists approached this task quite differently, though they both combined generated sections together resulting in fewer total sections (see Figure 5 for an example of each). S1 was happy with the generated sections but felt that they were too granular at times. He combined some sections together without changing any section timings. Our algorithm generated 13, 17, and 16 sections for his three videos, which he combined into 8, 11, and 10 sections respectively. 20 (69%) of these final sections were exactly the same as the generated sections. In contrast, S2 gave new start times for many of his sections. This took more effort, as he had to find exact times, but often his new start times differed only slightly from the algorithm's. Our algorithm generated 16, 9, and 21 sections for his three videos; he created 6, 5, and 6 sections respectively. 3 (18%) of these final sections were exactly the same as the generated sections, and another 5 (29%) were within 15 seconds of the generated sections. Upon closer inspection, we found that this slight misalignment was most often due to putting a boundary between two sentences on the same topic. Other times the misalignment was not really an error; both start times were equally good. Please see the Supplemental Materials for a detailed comparison.

We believe the difference in segmentation quality between the two streamers is due to three factors: type of stream, type of art, and transcript errors. One key difference between the two streamers is the type of stream they make. S2 typically streams *making* streams where he draws his online comics, while S1 makes *learning* streams where he teaches the audience how to do design [7]. We found that in learning streams, the transcript contains many more useful cues for segmentation than in making streams, because the streamer talks more about what they are doing. Another key difference between the two streamers is the type of artwork they show. S1 showed poster design work that follows a relatively linear process, with each step involving different commands. S2 showed comic illustration, which involved many of the same commands throughout the process. Last, S2 played lyrical music in the background which caused errors in the transcription, resulting in erroneous boundaries. In the end, the combination of lack of process narration, the similarity in commands, and the transcription errors came together to create a lower-quality segmentation for S2's streams. As a result he had to do more work to make a table of contents he was happy with. However, even in cases where the streamers did not agree with the output completely, they both found it helpful for creating their own sections. It was easier than segmenting from scratch as it gave them a starting point.

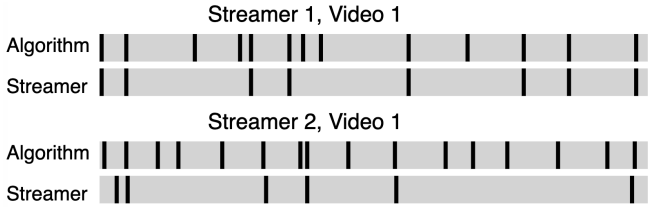

Figure 5. A comparison of our algorithm's segmentations to the streamers' final segmentations for the first video each streamer was given.

COMPARISON TO VIEWER SEGMENTATIONS

To evaluate our segmentation algorithm, we compared its results to the viewer segmentations collected in the Formative Study. Since we found that different people segment differently, our goal was to assess if our algorithm is at least as consistent with humans as they are with each other. If our algorithm can produce results at least as consistent as those made by potential viewers, this suggests it is reasonable.

Method

In addition to our algorithm's segmentations, we generated several alternate segmentations for each video to see whether the algorithm would perform better than simpler alternatives. These include versions of the algorithm that use application usage data only (*commandsOnly*) and transcript data only (*transcriptOnly*), a random segmentation (*random*), and a uniform segmentation (*uniform*) based on the average duration of sections made by coders (6 minutes).

To quantify how consistent our algorithm is with coder segmentations, we use the boundary similarity metric [6] as in the Formative Study. For each video, we first compute pairwise similarity scores between the algorithm's segmentation and each coder's segmentation. The average of these scores gives a measure of how consistent our algorithm is with coders for segmentation of a single video (*algorithm-coder* similarity). To determine whether the algorithm is as consistent with coders as they are with each other, we use a paired t-test comparing the average *algorithm-coder* similarity scores to the average *coder-coder* similarity scores across all videos for each streamer. Similarly, we compare *algorithm-coder* similarity with the other alternate segmentations' similarities to coders (*random-coder*, *uniform-coder*, *commandsOnly-coder*, *transcript-Only-coder*).

Results

Results varied across streamers, but in most cases, the algorithm was reasonably consistent with coders and better than *random* or *uniform* segmentations.

Algorithm is similar to coders for 3 out of 4 streamers. Table 2 shows the paired t-test results comparing *algorithm-coder* similarity to *coder-coder* similarity. For S1, S3, and S4, the *algorithm-coder* similarity scores were not distinguishable from *coder-coder* similarity scores, meaning that the algorithm was roughly as consistent with coders as coders were with each other. However, for S2, the *algorithm-coder* group was significantly less similar, suggesting that the algorithm performed worse than coders.

Algorithm is more consistent than other automated methods for some streamers. Table 3 shows the paired t-test results comparing *algorithm-coder* similarity to the other alternate segmentations. In most cases, our algorithm's segmentation aligned more closely with coders than *random* or *uniform* segmentations. For S1, the algorithm also aligned more closely with coders than either of the *commandsOnly* or *transcriptOnly* alternatives, and for S2, the algorithm aligned

Streamer	*n*	algorithm-coder mean similarity	algorithm-coder median similarity	coder-coder mean similarity	coder-coder median similarity	*t*	*p*	95% CI
S1	7	0.259	0.248	0.243	0.236	0.202	0.85	[-0.03, 0.04]
S2	5	0.191	0.193	0.268	0.240	**-4.563**	**0.01**	**[-0.13, -0.03]**
S3	6	0.292	0.255	0.322	0.310	-0.796	0.46	[-0.12, 0.06]
S4	5	0.203	0.202	0.208	0.186	-0.256	0.81	[-0.04, 0.03]

Table 2. Results of paired t-tests comparing the similarity of the algorithm with coders to the similarity of coders with each other, for each streamer. n refers to the number of videos for that streamer (i.e., the number of average boundary similarity scores in each group). For S1, S3, and S4, the difference between the two groups is not significant. For S2, the *algorithm-coder* group is significantly less similar than the *coder-coder* group.

Streamer	*n*	Comparison group	Mean similarity	Median similarity	*t*	*p*	95% CI
S1	7	*random-coder*	0.129	0.121	**4.413**	**0.01**	**[0.06, 0.21]**
		uniform-coder	0.165	0.168	**3.453**	**0.01**	**[0.03, 0.17]**
		commandsOnly-coder	0.196	0.196	**2.996**	**0.02**	**[0.01, 0.13]**
		transcriptOnly-coder	0.201	0.173	**2.602**	**0.04**	**[0, 0.11]**
S2	5	*random-coder*	0.081	0.082	**4.282**	**0.01**	**[0.04, 0.18]**
		uniform-coder	0.112	0.108	2.386	0.08	[-0.01, 0.16]
		commandsOnly-coder	0.161	0.184	1.461	0.22	[-0.03, 0.10]
		transcriptOnly-coder	0.143	0.138	**2.913**	**0.04**	**[0, 0.09]**
S3	6	*random-coder*	0.142	0.129	**4.160**	**0.01**	**[0.06, 0.25]**
		uniform-coder	0.179	0.162	**2.851**	**0.04**	**[0.01, 0.22]**
		commandsOnly-coder	0.242	0.223	1.846	0.12	[-0.02, 0.12]
		transcriptOnly-coder	0.204	0.184	2.129	0.09	[-0.02, 0.21]
S4	5	*random-coder*	0.143	0.135	2.700	0.05	[0, 0.12]
		uniform-coder	0.184	0.184	1.485	0.21	[-0.02, 0.06]
		commandsOnly-coder	0.194	0.190	0.240	0.82	[-0.04, 0.05]
		transcriptOnly-coder	0.193	0.179	1.059	0.35	[-0.02, 0.04]

Table 3. Results of paired t-tests comparing *algorithm-coder* similarity to the similarity of four alternate algorithms with coders, for each streamer. In all bolded rows ($p < 0.05$), the positive t scores indicate that the *algorithm-coder* group had higher similarity than the comparison group.

more closely with coders than *transcriptOnly*. For S3, the algorithm was marginally more similar to coders than those two alternatives, and for S4 there were no significant differences.

Discussion: Differences Between Streamer Styles

The performance of our algorithm varied across the four streamers, suggesting that it may work better for certain streaming styles over others. Based on our observations of the videos in this set, S1 and S3 tend to follow a more linear step-by-step process than S2 and S4, in terms of both the commands they use and the amount of instructional narration they do. This may be why the algorithm performs better on those streamers' videos. The result that the algorithm was significantly *less* consistent with coders for S2 corroborates S2's desire to more drastically change their automatically-generated segmentations.

For S1, the algorithm performed significantly better than either *commandsOnly* or *transcriptOnly*, and for S3, marginally better, indicating that both data streams are important, but the relative utility of each may depend on the streamer. It may be the case that different streamers benefit from different weights for the algorithm's scoring function components.

DISCUSSION & FUTURE WORK

The algorithm proposed in this paper performs better on some streams than on others, highlighting the complexity of live stream segmentation. We found that there is no one single solution, but we believe that leveraging additional metadata about the video could improve segmentation. Our approach is generalizable to incorporating other input sources, such as chat logs, audio, and visual data. For example, logs from the live chat could be used to find moments of high interest [17] or topic changes, audio analysis could help address transcription errors where background music is transcribed as speech, and computer vision could be used to adjust boundaries where a command is not aligned with visual change. In situations where the application usage is missing or not very discriminatory (as in S2's case), the spatial location of edits could also be useful [3]. Last, one could also ask the streamer or viewers to mark section boundaries during the live broadcast [15].

Finding Highlights

The goal of our algorithm was to create a table of contents, but we found that many streamers and viewers are also interested in highlights. Twitch allows streamers and viewers to manually create highlight clips from streams. Future work should explore how to automatically generate useful highlights. As one example, a viewer may wish to see all moments where the streamer answered a question from the chat. We have begun to develop a method for locating these moments using the transcript and chat log of a video. We find all chat messages with a question mark and search subsequent transcript sentences for an approximate match. Since streamers often read questions out loud before responding to them, finding a match helps to find the moment right before the streamer answered the question. An approach like this could be extended to find *all* moments of streamer-viewer interaction (not just question-answering) and allow viewers to watch only these moments or to skip them and focus on the instructional content.

Evaluating Algorithm Success

As is often the case with creative open-ended tasks, applying automated methods to understand them is difficult. As we found in both our formative work and our evaluations, it is not obvious how best to segment a live stream, and different people can produce very different segmentations. Notably, the streamers we interviewed generated much longer sections on average (10.5 minutes) than our algorithm (5.5 minutes) or the coders (6 minutes). This suggests that a streamer's desired segmentation may not always align with what viewers want.

The fact that our algorithm was at least as consistent with coders as they were with each other is promising, but whether and how its segmentations help viewers more easily navigate, understand, and learn from videos remains to be seen. We also hypothesize based on our feedback from streamers that having *any* segmentation for a video is better than no segmentation at all, as it gives viewers more ways to skim the contents of a video compared to a traditional interface. Future work should evaluate this hypothesis and explore the design space of interfaces for presenting a table of contents, and how they compare to traditional video-viewing interfaces.

Generalizability of Approach

Our current approach and evaluation focused on live streamed videos in Adobe Photoshop, but we believe it can extend to any creative software screencast videos. Most software include a `save` command, and other general transitional commands could be added such as opening/closing a document. Layer selection is also not specific to Photoshop; many graphic design and digital art applications also include layers as a main organizational tool. Our approach for transcript analysis may even generalize more broadly to any demonstrational video (*e.g.,* physical DIY tutorials or cooking videos). Future work should explore how our approach applies to these broader domains, as it may have implications for general live streaming platforms, not just those focused on creative work.

CONCLUSION

This paper presented a streamer-in-the-loop approach for temporal segmentation of creative live stream videos. We developed an algorithm that uses audio transcripts and application usage logs to automatically segment videos into sections based on steps of the creative process. The streamer can then add their own labels to each section to help viewers navigate. Through feedback from two streamers and a comparison of our algorithm's output with segmentations made by viewers on 23 videos, we found that our approach works well for videos where the streamers consistently talk about their process and use a variety of application commands. More generally, we find that the task of segmenting creative live streams into sections is not straightforward and has many possible solutions. We argue that our approach provides one reasonable solution, bringing viewers one step closer to the wealth of knowledge that lays hidden inside live stream videos.

ACKNOWLEDGEMENTS

We thank Sarah Rapp, Dave Stein, and Zach McCullough for their input and feedback throughout this project, and Nancy Reid for her help with statistical analyses.

REFERENCES

[1] 2019. Speech to Text API. (2019). https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/

[2] Nikola Banovic, Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2012. Waken: Reverse engineering usage information and interface structure from software videos. In *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*. ACM Press, New York, New York, USA, 83. DOI:http://dx.doi.org/10.1145/2380116.2380129

[3] Hsiang-Ting Chen, Li-Yi Wei, Björn Hartmann, and Maneesh Agrawala. 2016. Data-driven adaptive history for image editing. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '16*. ACM Press, New York, New York, USA, 103–111. DOI: http://dx.doi.org/10.1145/2856400.2856417

[4] Jacob Eisenstein and Regina Barzilay. 2008. Bayesian Unsupervised Topic Segmentation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '08)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 334–343. http://dl.acm.org/citation.cfm?id=1613715.1613760

[5] Travis Faas, Lynn Dombrowski, Alyson Young, and Andrew D. Miller. 2018. Watch Me Code: Programming Mentorship Communities on Twitch.tv. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (nov 2018), 1–18. DOI: http://dx.doi.org/10.1145/3274319

[6] Chris Fournier. 2013. Evaluating Text Segmentation using Boundary Edit Distance. In *Proceedings of 51st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Sofia, Bulgaria, 1702–1712.

[7] C. Ailie Fraser, Joy O. Kim, Alison Thornsberry, Scott Klemmer, and Mira Dontcheva. 2019a. Sharing the Studio: How Creative Livestreaming can Inspire, Educate, and Engage. In *Proceedings of the 2019 on Creativity and Cognition - C&C '19*. ACM Press, New York, New York, USA, 144–155. DOI: http://dx.doi.org/10.1145/3325480.3325485

[8] C. Ailie Fraser, Tricia J. Ngoon, Mira Dontcheva, and Scott Klemmer. 2019b. RePlay: Contextually Presenting Learning Videos Across Software Applications. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*. ACM Press, New York, New York, USA, 1–13. DOI: http://dx.doi.org/10.1145/3290605.3300527

[9] Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2010. Chronicle: Capture, exploration, and playback of document workflow histories. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology - UIST '10*. ACM Press, New York, New York, USA, 143. DOI: http://dx.doi.org/10.1145/1866029.1866054

[10] Juho Kim, Philip J. Guo, Carrie J. Cai, Shang-Wen (Daniel) Li, Krzysztof Z. Gajos, and Robert C. Miller. 2014a. Data-driven interaction techniques for improving navigation of educational videos. In *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. ACM Press, New York, New York, USA, 563–572. DOI: http://dx.doi.org/10.1145/2642918.2647389

[11] Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J. Guo, Robert C. Miller, and Krzysztof Z. Gajos. 2014b. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*. ACM Press, New York, New York, USA, 4017–4026. DOI: http://dx.doi.org/10.1145/2556288.2556986

[12] Donald E. Knuth and Michael F. Plass. 1981. Breaking paragraphs into lines. *Software: Practice and Experience* 11, 11 (nov 1981), 1119–1184. DOI: http://dx.doi.org/10.1002/spe.4380111102

[13] Pascal Lessel, Alexander Vielhauer, and Antonio Krüger. 2017. Expanding Video Game Live-Streams with Enhanced Communication Channels: A Case Study. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*. ACM Press, New York, New York, USA, 1571–1576. DOI: http://dx.doi.org/10.1145/3025453.3025708

[14] Zhicong Lu, Michelle Annett, Mingming Fan, and Daniel Wigdor. 2019. "I feel it is my responsibility to stream": Streaming and Engaging with Intangible Cultural Heritage through Livestreaming. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*. ACM Press, New York, New York, USA, 1–14. DOI: http://dx.doi.org/10.1145/3290605.3300459

[15] Zhicong Lu, Seongkook Heo, and Daniel J. Wigdor. 2018. StreamWiki: Enabling Viewers of Knowledge Sharing Live Streams to Collaboratively Generate Archival Documentation for Effective In-Stream and Post Hoc Learning. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (nov 2018), 1–26. DOI:http://dx.doi.org/10.1145/3274381

[16] Justin Matejka, Tovi Grossman, and George Fitzmaurice. 2011. Ambient help. In *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*. ACM Press, New York, New York, USA, 2751. DOI:http://dx.doi.org/10.1145/1978942.1979349

[17] Rui Pan, Lyn Bartram, and Carman Neustaedter. 2016. TwitchViz: A Visualization Tool for Twitch Chatrooms. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '16*. ACM Press, New York, New York, USA, 1959–1965. DOI: http://dx.doi.org/10.1145/2851581.2892427

[18] Jurre Pannekeet. 2018. Five Key Insights into Twitch and YouTube Gaming and the 2.4Bn Viewing Hours They Generated in Q1 2018. (apr 2018). https://newzoo.com/insights/articles/five-key-insights-into-twitch-and-youtube-gaming/

[19] Amy Pavel, Dan B. Goldman, Björn Hartmann, and Maneesh Agrawala. 2015. SceneSkim: Searching and Browsing Movies Using Synchronized Captions, Scripts and Plot Summaries. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*. ACM Press, New York, New York, USA, 181–190. DOI: http://dx.doi.org/10.1145/2807442.2807502

[20] Amy Pavel, Colorado Reed, Björn Hartmann, and Maneesh Agrawala. 2014. Video digests: a browsable, skimmable format for informational lecture videos. In *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. ACM Press, New York, New York, USA, 573–582. DOI: http://dx.doi.org/10.1145/2642918.2647400

[21] Lev Pevzner and Marti A Hearst. 2002. A Critique and Improvement of an Evaluation Metric for Text Segmentation. *Comput. Linguist.* 28, 1 (mar 2002), 19–36. DOI: http://dx.doi.org/10.1162/089120102317341756

[22] Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. 2011. Pause-and-Play: Automatically Linking Screencast Video Tutorials with Applications. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*. ACM Press, New York, New York, USA, 135. DOI: http://dx.doi.org/10.1145/2047196.2047213

APPENDIX

Streamer	Type of Work	Duration	# Sections	Shortest Section	Longest Section	Avg Section Length	Intro?	Outro?
S1	Graphic Design	1:45:01	17	1:28	9:52	6:10	✓	✗
		0:52:25	13	0:48	9:55	4:00	✓	✓
		0:59:21	11	0:59	9:55	5:20	✓	✓
		0:56:42	11	0:54	9:57	5:09	✓	✓
		1:02:28	13	1:14	9:56	4:48	✓	✓
		1:11:43	11	1:03	9:59	6:31	✓	✗
		1:01:31	11	2:17	10:00	5:33	✓	✗
S2	Comic Art	2:05:10	21	1:21	9:51	5:48	✓	✓
		0:58:06	7	1:19	10:00	8:11	✓	✗
		0:55:51	9	1:11	9:55	5:54	✓	✓
		1:20:24	10	1:19	13:15	7:41	✓	✓
		1:47:60	16	1:30	9:59	6:43	✓	✓
S3	Digital Illustration	0:48:36	7	0:37	9:57	6:32	✓	✓
		0:29:54	6	0:38	9:04	4:47	✓	✓
		0:28:22	6	1:56	9:42	4:27	✓	✓
		0:36:36	9	1:19	7:28	3:49	✓	✓
		0:33:04	6	0:37	9:40	5:05	✓	✓
		0:50:39	13	1:45	8:09	3:37	✗	✓
S4	Image Compositing	2:04:38	26	0:44	9:50	4:48	✓	✓
		1:14:08	12	1:00	10:01	6:11	✓	✓
		0:31:55	8	0:32	10:00	3:59	✓	✓
		1:22:23	18	1:47	8:10	4:34	✓	✓
		1:03:05	11	2:08	9:57	5:44	✓	✗

Table 4. Summary of our segmentation algorithm's output on a sample set of 23 videos from 4 streamers. Duration is in H:MM:SS format, and section lengths are in MM:SS format. Intro and Outro specify whether the algorithm generated an Intro and Outro section, respectively.