Swift: Reducing the Effects of Latency in Online Video Scrubbing

Justin Matejka, Tovi Grossman, George Fitzmaurice
Autodesk Research, Toronto, Ontario, Canada
{*firstname.lastname*}@autodesk.com

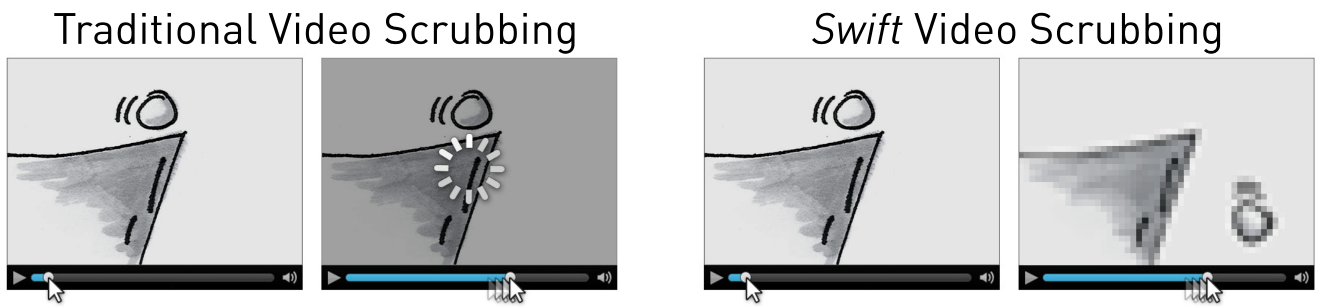

Figure 1. An illustration of the scrubbing behavior of a traditional streaming video player and the *Swift* player. With the *Swift* system a quick-to-download low resolution version of the video is displayed while scrubbing.

ABSTRACT
We first conduct a study using abstracted video content to measure the effects of latency on video scrubbing performance and find that even very small amounts of latency can significantly degrade navigation performance. Based on these results, we present *Swift*, a technique that supports real-time scrubbing of online videos by overlaying a small, low resolution copy of the video during video scrubbing, and snapping back to the high resolution video when the scrubbing is completed or paused. A second study compares the *Swift* technique to traditional online video players on a collection of realistic live motion videos and content-specific search tasks which finds the *Swift* technique reducing completion times by as much as 72% even with a relatively low latency of 500ms. Lastly, we demonstrate that the *Swift* technique can be easily implemented using modern HTML5 web standards.

Author Keywords
Video, Video Navigation, Online Streaming

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Graphical User Interfaces.

INTRODUCTION
Streaming online video players have given internet users the instant gratification of watching a video play immediately, as opposed to the previous workflow of downloading a video and then viewing it using a local player. With recent advancements in both internet download speeds, and faster CPUs, the quality of streaming videos has improved drastically. High definition videos are now standard, and full length television shows and movies are readily available on sites such as Netflix and Hulu.

Although very popular, streaming videos have limitations when it comes to navigation. In most desktop media players, the user is able to "scrub" the video by moving a slider along the timeline and the current frame of the video updates in real-time. In contrast, streaming video players request new frames from the server to update the view, introducing a significant amount of latency. This latency makes the scrubbing experience very choppy at best, and for many players, the view does not start updating until after the mouse button has been released (Figure 1).

For many usage scenarios, the ability to scrub video timelines is critical to the viewing experience. For example, a user may wish to find a particular scene in a movie, look for when a particular operation was performed in a software tutorial video, or skip past an advertisement while watching a sporting event. While scrubbing is sometimes enabled once a video is cached, fully downloading a video can take a considerable amount of time. Furthermore, it might not be desirable, or even possible, to cache a large video file for reasons such as the bandwidth costs incurred by the server and/or user, or the storage capacity of the playback device.

Many aids for navigating videos have been explored [21]. However, most require additional visual elements such as summary storyboards [25] or video analytics [22], and few enhance the ubiquitous scrubbing behavior. Furthermore, most enhancements are designed for desktop systems and assume random access availability to the video. Despite the ubiquity of online video players, we are unaware of any research to date which has empirically measured the user performance impact of latency during video scrubbing.

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. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CHI'12, May 5–10, 2012, Austin, Texas, USA.
Copyright 2012 ACM 978-1-4503-1015-4/12/05...$10.00.

In this paper, we first conduct a study to measure the impact of latency in a controlled abstract environment. We find that even extremely short delays (20ms) can cause significant difficulty when navigating videos, and with latencies users typically experience (1000ms or more), seeking tasks can take up to 10 times longer.

To combat these observed effects of latency, we present *Swift*, a technique that supports real-time scrubbing of streaming videos even in high latency conditions. *Swift* works by overlaying a small, low resolution copy of the video during video scrubbing, and snaps back to the high resolution video when the scrubbing is completed or paused. The technique has no impact on the user interface controls or layout. By fixing the resolution and number of video frames to the number of input pixels on the timeline slider, the size of the low-resolution copy can be kept at a fixed size of approximately 1MB, regardless of the duration or size of the original source video.

In a second study, using real video content, we evaluate *Swift*, against video navigation with traditional browsers in 20ms and 500ms latency environments. Swift significantly decreased task completion times, in some cases by 72%. After providing an analysis of the results, we demonstrate a simple HTML5 implementation of *Swift*. Based on these results, we believe the *Swift* technique can be integrated into today's online video players, and significantly improve the user's viewing experience.

RELATED WORK

Video Navigation

There have been many projects exploring improvements to video navigation. A number of approaches have looked at using the traditional timeline slider augmented with different dynamics such as the AV-ZoomSlider [12], which uses the additional dimension of the y-axis to enable more precise control. The PVSlider [20] provides an elastic scrubbing experience for more fluid control.

Several systems have been developed to augment the traditional video timeline with additional graphical elements and information. The Video Explorer [22] uses additional timeline strips to visualize low level video features. Bailer et. al.'s SVAT [2] visualizes scene boundaries and motion events on the timeline. Chronicle [8] uses additional rows on the timeline to display document workflow events based on captured metadata. Joke-o-mat HD [14] and emoPlayer [4] both support navigating videos based on user contributed metadata.

An alternate way to get an overview of an entire video is through a collection of scene thumbnails [11, 25]. Truong and Venkatesh [24] provide a thorough summary and classification of work in both the creation of thumbnails and video skimming. This includes techniques that display thumbnails hierarchically [6], thumbnails lists [3], and fisheye views of thumbnails [5].

While these described techniques all aid video navigation in interesting ways, they typically require a local cache of the video, or its thumbnails, to function, and do not support real-time scrubbing in an online environment.

Existing Deployed Technologies

The internet is full of sites with streaming video players and we tested over 30 unique implementations. The most popular video streaming site, YouTube, currently hosts more than 350 million streaming videos. While scrubbing, the YouTube player does not update the position in an un-cached video until the mouse button is released. This general interaction model is representative of the majority of streaming players we have found.

Several players such as the ones found on Hulu and ESPN.com support enhanced video navigation by providing a thumbnail preview while hovering over the timeline. These thumbnails are not pre-loaded so they suffer from the same latency issues as the full video, but they do present an interesting interaction model.

The PC version of the Netflix streaming video player is the only example we have found to support real-time scrubbing. When the user begins scrubbing, sequential thumbnail images are displayed in the center of the screen, and update in real-time as the user scrubs (Figure 2).

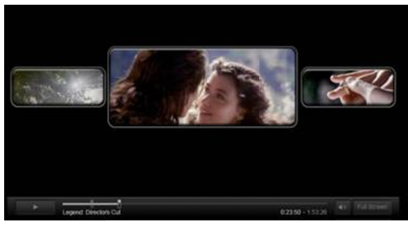

Figure 2. PC version of the Netflix streaming video player.

We are unaware of any archived publication or white paper describing its behavior, and so its technical implementation is unknown. However, the real-time scrubbing seems to only be enabled once a full set of thumbnails has been downloaded. For some movies, the scrubbing is enabled within 10 seconds, however, for others, we found it could take several minutes before the real-time scrubbing begins to work, even with a high-speed broadband connection.

Despite its limitations, the Netflix player should be considered closely related to *Swift*. That said, *Swift* does contribute a scalable technique, where real-time scrubbing only requires 1MB of data, regardless of the source video resolution or duration. Furthermore, we demonstrate a simple implementation using modern web standards consisting of less than 30 lines of javascript and HTML5 code, and provide a public dissemination of its implementation and related technical details to the research community.

DEFINITIONS

Before describing the first experiment, we will outline definitions for the two main factors being tested.

Simulated Network Latency

All computer systems have some amount of inherent input latency. Using the technique from [17] we calculated the average delay between when the mouse is moved to when the cursor actually moves on the screen to be 37ms. For our video navigation tasks we refer to *simulated network latency* (or *sn-latency*) as the time between when a frame is requested by the user and when the frame is displayed on the screen, less the inherent system latency. With an online video player this is the time it takes to request a new play position from the server, and have the server transfer enough frames to buffer the video sufficiently to begin playing again. While the effects of latency on some interaction tasks has been studied [16, 18], its impact on video scrubbing is unknown.

Classification of Video Types for Navigation Tasks

We restrict our evaluation to tasks where the user knows what scene they are looking for, and would recognize when they have successfully navigated to it. Within such a task, a user's knowledge of the target video content and scene organization provides information and orientation which may affect scrubbing behaviors. Thus we found it useful to divide videos into three main *video type* categories:

Sequential

In a *sequential* video, the scenes have a natural order which is known to the user. It is therefore possible for a user to estimate where the target scene is, and to "jump" to the approximate point in the timeline. During the navigation task the user is able to tell if they have gone too far, or not far enough, and also judge how far away from the target they are. An example of a sequential video seeking task is finding a particular time in a televised sporting event: to find "the beginning of the 4th quarter" in a basketball game, the user could jump to about 75% into the video and then looking at the game clock, adjust accordingly.

Ordered

With an *ordered* video it is possible for a user to tell if they are before or after the target scene, but not how far away they are. An example would be navigating through a commercial break to get back to the television show. After navigating, the user would know if they have gone too far, or, if they still see a commercial, had not gone far enough. Without real-time scrubbing, the optimal strategy for this type of task may be a binary search of the timeline, reducing the search space by half in each step.

Random

In a video with a completely *random* ordering, the user does not know where the target scene is likely to be located in the timeline, and has no cues as to where the target sits in relation to their current position. There are no particularly intelligent strategies for this type of task; the user is forced to try and look at each scene. A primary example of this type of task is for a user to find a particular scene in a movie which they have never seen before.

The exact same video could have different navigation characteristics for different users; finding a scene in a movie that one person is unfamiliar with would be a *random* searching task. For a person who is very familiar with the move, it would be more of a *sequential* seeking task. These classifications also do not necessarily have hard boundaries between them. For example a sporting event which only displays the clock some of the time could present a primarily *sequential* seeking task, but the segments without a clock present would represent a *random* seeking sub-task. Even though the distinctions are not always clear, this categorization of video types will be useful when studying navigation techniques over a range of potential scene finding scenarios.

EXPERIMENT ONE

To better understand how latency affects video navigation tasks we conducted a controlled experiment. Specifically, we wanted to test video scrubbing performance while navigating to a target scene with varying levels of latency and for differing video types. To ensure a controlled environment, this experiment was carried out with abstracted video content simulating three types of videos: *sequential*, *ordered* and *random*, and two different scene counts to simulate shorter (*12*) and longer (*24*) videos.

Participants and Apparatus

Twelve paid volunteer participants (8 female) were recruited though an online classified posting. Users had varying levels of computer experience, with daily usage ranging between 2 and 10 hours.

The experiment was conducted in a private office on a 3.16GHz quad-core desktop computer running Windows 7 64-bit Edition. The graphics card was an nVidia Quadro FX 5600 and was driving a 24" Dell LCD monitor with a resolution of 1920 by 1200.

Design

A repeated measures within-participant design was used with the independent variables being *video type* (*sequential, ordered, random*), *number of scenes* (*12, 24*), and *sn-latency* (*1000ms, 500ms, 100ms, 20ms, 0ms, 0ms-lowRes*). The ordering of *sn-latency* and *number of scenes* were counterbalanced and *video type* was randomized. A fully crossed design resulted in 36 combinations of variables.

Each participant performed the experiment in one session lasting approximately one hour. The study was divided into two blocks, with each condition run 4 times per block.

Video Player

A custom video player with a playback resolution of 800 by 600 pixels was used for the study. The player was programmed as a stand-alone application to allow for precise control of the latency, and to support high frequency logging capabilities. The interface was intentionally simple, with just the video playback window and a timeline slider (Figure 3). In the *0ms* condition, dragging the slider along the timeline would update the video frame to the appropriate location immediately (less the 37ms inherent system input latency). In the other conditions, the frame would only update if the slider remained stationary for the condition's associated latency duration. It was not necessary to mouse-up to trigger the update, dwelling in place was sufficient. During the latency period, the screen would display a "Loading" message on top of the previous frame to help the user understand the player's state. Playback was disabled, as we were only testing seeking behaviors.

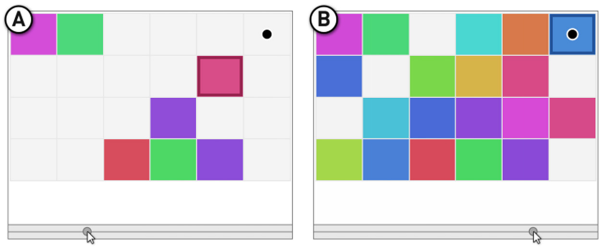

Figure 3. Video player used in the experiment near the beginning of a trial (A) and at the target scene (B). The most recently drawn square, representing the current scene, was outlined with a wide border. The goal scene is marked with a black dot.

Simulated Network Latency
Five different simulated network latency levels were used for the first study ranging from 1000ms down to 0ms. The *1000ms* condition is typical of what we have observed on a fast (~18MB/s) network when viewing a 480p video on YouTube. When viewing 720p content the delay is generally more than 2000ms. We also tested shorter delays of 500ms and 100ms, which are both faster than what we have been able to achieve on any deployed network, but might be conceivably possible. We also tested with 20ms as an approximate "theoretical limit" based on the average *network* latency, or ping times, of 20 to 40ms.

The *0ms* condition was included to simulate the baseline scenario, of watching the video on a desktop player capable of real-time scrubbing, or an online player once the video has been fully cached. To begin to understand the effects of video resolution on navigation tasks, the final condition (*0ms-lowRes*) was the same as the *0ms* condition, except that while scrubbing the timeline, the frame displayed was a 64 by 48 pixel version of the current frame which had been scaled up to 800 by 600 and bilinearly smoothed. In particular, we hoped to see if switching between a crisp view and a blurry view of the same video content would affect navigation performance.

Video Content
The video content for the study was a grid of procedurally generated squares that faded in one at a time. There were an equal number of squares as *number of scenes* (i.e., 12 squares for *12* scenes, and 24 squares for *24* scenes). During each scene a different square would be filled in. To make it easier to determine which was the last square to be drawn (i.e. the current "scene"), the most recently drawn square was outlined with a 10px wide border (Figure 3). The timeline was divided into equally spaced segments so each scene was of the same length. Because playback was disabled, the scenes, and video, did not have an absolute duration. The timeline was 800 pixels wide and each scene occupied 800/12=67 pixels in the *12 scene* conditions, and 800/24=33 pixels in the *24 scene* conditions.

To represent each of the video orderings in a controlled fashion, the order for which the squares appeared differed:

SEQUENTIAL: The squares filled in from left-to-right and top to bottom (Figure 4). This made it possible to tell how far the target scene was into the video so the user could make a large movement and "jump" to the correct area of the timeline if desired.

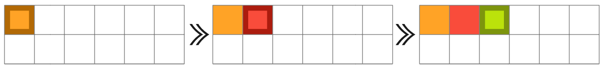
Figure 4. First three scenes in a *sequential* scene ordering. The order which the squares fill in is fixed.

ORDERED: For the *ordered* condition the squares were filled in a random order, and once a square filled in, it remained colored for the rest of the video (Figure 5). With this configuration the user could tell if they were not far enough (the target square was not filled), or too far (the target square was filled, but not outlined) but could not tell how far forward or back they needed to navigate.

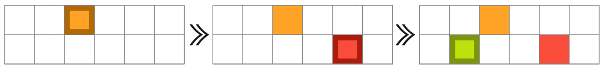
Figure 5. First three scenes in an *ordered* scene ordering. The order which the squares fill in is random.

RANDOM: With the *random* ordering, the squares would become filled in a random order, however they would only remain filled in for one scene (Figure 6). This gave no information to the user in terms of if they were before or past the target scene; they could only know if they were exactly at the target scene.

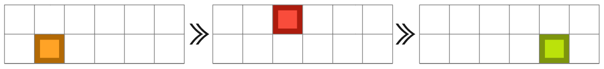
Figure 6. First three scenes in a *random* scene ordering. The order which the squares fill in is random.

Task
In all trials, the user's task was to find a target scene in the video. More specifically, the user had to find the part of the video where a target square was the last to be filled in. The target square was indicated with a black dot overlaid on the video (Figure 3). The first and last scenes were never chosen for a trial, and the same scene was never selected for consecutive trials. Additionally, the timeline was halved, and half of the trials were selected from each section.

Procedure
At the beginning of the study the examiner explained and showed the user examples of each video type. Once the differences between the video types were understood, the

participants spent two minutes becoming accustomed to the different latency conditions.

Each trial began when the cursor entered the timeline slider, and at that time the dot would appear over the target square. Participants were instructed to click and hold the mouse button down while searching for the target scene and to release the button once it was found. Errors were not possible as the trial ended only once the target scene was found. Between each block users were given a short break, and the next condition was described on the screen. If the participant was unclear how the next condition would work they had the opportunity to execute several practice trials.

Results

The primary independent variable was *completion time* for each task. Repeated measure analysis of variance showed a main effect for *video type* ($F_{2,22}$ = 79.2, p < .0001), *number of scenes* ($F_{1,11}$ = 39.4, p < .0001), and *latency* ($F_{5,55}$ = 73.5, p < .0001). Additionally, *video type* had a significant interaction with *number of scenes* ($F_{2,22}$ = 14.6, p < .0001) as well as with *ns-latency* ($F_{10,110}$ = 22.3, p < .0001).

Looking at the results for each *video type* we can see that overall the completion times increase as the latency increases (Figure 7). Of particular interest is the large jump in completion times between *0ms* and *20ms* conditions with overall mean completion times of 3210ms and 6175ms respectively (p < .0001). It is important to recall that the *20ms* latency condition represents a theoretical limit, and is much lower than any existing online video player that we are aware of. This result suggests that even optimal latency levels will not approximate the efficiency of real-time scrubbing capabilities.

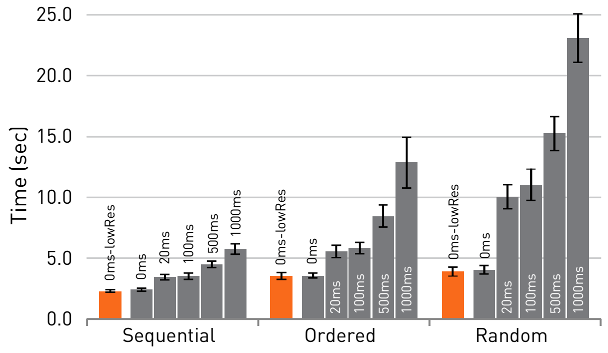

Figure 7. Average median navigation completion times for each combination of *video type* and *ns-latency.* (Note: error bars report standard error).

When we compare the *0ms* and *0ms-lowRes* conditions we see that the average *completion times* were 3.21s and 3.22s respectively. This result indicates that exploring the use of a lower resolution video while scrubbing could be a promising direction.

Figure 8 illustrates the completion times for each individual video type. It can be seen that after the initial jump from 0ms to 20ms, completion times increase in a fairly linear fashion with latency. The exception is the Random video type with 24 scenes, where the *1000ms* latency condition possesses additional difficulty.

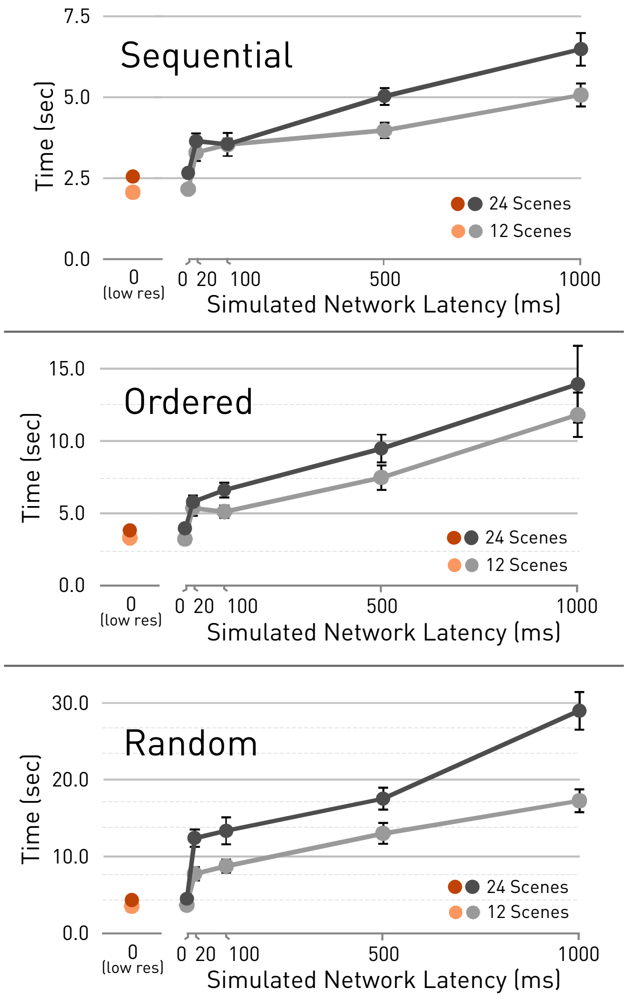

Figure 8. Average median navigation completion time divided into groups based on *video type*. (Note: error bars report standard error).

Frames Seen

In addition to completion time, it is interesting to look at how efficient users are being in their searching behaviors. One way to do this is to look at how many times a new video frame is seen by the user while completing a task. For conditions with no latency we cannot tell when a user has seen a new frame, as they are being displayed constantly. However, for the conditions with latency we can count the number of frames seen during each trial (Figure 9).

Across all *video type* and *number of scene* combinations the trend is for the number of frames seen to go down as the latency increases. This matches with the observed behavior of users being more “careful” with their movements as the penalty for each additional search step became greater. That is, when the penalty for a poor strategy is small, users were more likely to randomly search around in the video than to make a calculated decision of where to look next. So although latency is detrimental to performance times, users can partially make up some of the time in higher latency conditions with improved search strategies.

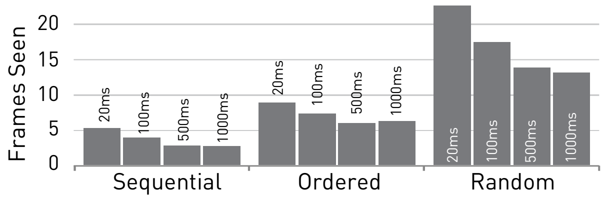

Figure 9. Average number of frames seen per trial.

Anecdotally, we found that most users were using near optimal strategies of jumping to the approximately correct position with the *sequential* videos, and linearly searching through the *random* videos. With the *ordered* videos however, few participants performed an optimal binary search of the video scenes, however, many participants performed a somewhat “partial binary” search by first seeking to near the middle of the video, and then searching linearly in one direction to find the target scene.

THE *SWIFT* TECHNIQUE

Our controlled evaluation on the effects of latency indicates a significant performance decrease when real-time scrubbing is not available. One possible way to address this limitation is to use a lower resolution version of the video that can be cached immediately and used for scrubbing. The Netflix player comes close to doing so, but its implementation details are unknown, and there is a noticeable delay before scrubbing is enabled. As such, we developed *Swift* to support *immediate* low resolution scrubbing. This technique will allow us to empirically evaluate if low-resolution scrubbing could address the performance limitations identified in our first experiment.

Overview

The idea behind *Swift* is to display a fully cached, low resolution copy of the video during video scrubbing, and snap back to the high resolution video when scrubbing is completed or paused. Since the low resolution version of the video is fully cached ahead of time, it can be scrubbed in real-time and used to find the desired scene in the video. Displaying the low resolution version overlaid onto the entire size of the high resolution version allows for spatial congruence and tracking video content while scrubbing.

User Interface

There is a large base of existing streaming videos on the internet, and a broad demographic spectrum of users who consume them [9]. As such, it is important for the navigation mechanism to be simple and have minimal impact on the existing user interfaces. Introducing advanced controls could impact ease-of-use, and hosting sites may be reluctant to adopt major changes to the interface layout.

To this end, our technique requires no changes to the traditional video player interface, and no changes to the interaction model. It would be relatively straightforward to retrofit an existing player to use our technique, and we believe any user familiar with a traditional timeline slider would be able to use the *Swift* interface on first exposure based on our results and observations from the *0ms-lowRes* condition in the first study.

Low Resolution Videos

The human visual system has an amazing tolerance to degradation in image resolution. For example, as little as 16 × 16 pixel images are suitable for face recognition [1]. Torralba et. al. found that human scene recognition on images with a resolution of 32 × 32 was 93% of the recognition at the full resolution of 256 × 256 source images, despite having only 1.5% the number of pixels as the original [23]. This findings support the idea that low resolution videos might be suitable for recognition tasks.

Increases in internet bandwidth and advances in video compression and streaming technology [10, 15] will continue to drive the movement to higher quality streaming videos. Those same advancements make lower resolution videos extremely efficient to transfer.

Video Size Analysis

The success of *Swift* depends on a low-resolution video small enough that it can be cached almost immediately after a page is loaded, but large enough to give a reasonable depiction of the video. Our hope was to use a video size of approximately 1MB, which would take less than a second to download with most broadband internet connections.

To determine appropriate parameters for the lower-resolution video, we looked at file sizes generated by a modern codec. We used the H.264/MPEG-4 AVC high profile codec, given its high quality, low file sizes, and HTML5 compatibility. Videos were converted to .mp4 files with this codec using the “mp4” option of “Miro Video Converter”, a free video conversion tool. A one hour full motion movie was used for the evaluation, at 800 × 600 resolution. For the evaluation, we varied the video resolution, ranging from 320×240 to 32×24.

In addition, we varied the total number of frames encoded. A key insight is that only a subset of the video’s frames need to be encoded for real-time scrubbing, equal to the pixel width of the timeline slider. For instance, a slider with a width of 600 pixels can only access one of 600 frames during scrubbing, regardless of the actual length of the video. We varied frame totals from 50 (representing low granularity scrubbing) to 1600 (representing high granularity scrubbing in a full screen playback mode). To convert the videos to a desired frame total, *n*, the playback speed was modified using a video editor to run exactly *n* seconds, and the video was then encoded at 1fps.

Figure 10 shows the resulting mp4 file sizes, at different resolutions and frame counts. It can be seen that video sizes drastically decrease as the resolution decreases. It can also be seen that there are a group of candidate parameters that result in video sizes close to 1MB. Based on these results, we choose to use a video size of 134 × 100, with a frame count of 800. This gave us the file size we wanted and the resolution seemed to provide adequate visual cues during navigation tasks. If the aspect ratio of a video was wider than 4:3, the height could be reduced instead of increasing the width, so that the file size would not increase.

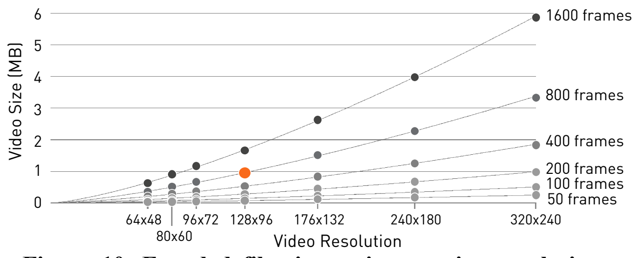

Figure 10. Encoded file sizes using varying resolutions and frame counts.

Because we fix the total frames encoded, and the frame resolution, these parameters should reduce *any* source video into the range of a 1MB file, regardless of its initial resolution, duration, or frame rate. The compressibility of the video content will have some effect on the resulting video size, however, the most complicated videos we tried still had files sizes of approximately 1MB, and for some videos we achieved sizes as small as 0.2MB.

EXPERIMENT TWO

To validate the *Swift* technique we conducted a second controlled experiment using an actual full-motion video and content-specific search tasks.

Participants and Apparatus

Twelve paid volunteer participants (7 female) were selected from the same recruiting pool as used for the first experiment. Participants reported using a computer for an average of between 2 and 14 hours per day (μ = 6.5 hours) and watching between 0 and 200 online videos per month (μ = 47 videos). The experiment was conducted in the same office and on the same machine as the first study.

Design

A repeated measures within-participant design was used with the independent variables being *video type* (*sequential, ordered, random*), target *discernibility* (*low, high*), *sn-latency* (*20ms, 500ms*), and *technique* (*traditional, swift*). The ordering of *video type*, *discernibility*, and *technique* were counterbalanced and the order of *sn-latency* was randomized. A fully crossed design resulted in 24 conditions. Each participant performed the experiment in one session and each condition was run 5 times, with the first trial discarded as practice.

For the *traditional* technique the *sn-latency* value had the same effect as in the first study (the frame would update after *x* ms), and with the *Swift* technique the low resolution version of the video was shown while scrubbing and the full resolution version would appear after the *x* ms delay.

Latency values of *20ms* and *500ms* were selected from the values used in the first study, with again the *20ms* condition serving as an approximate "theoretical limit" assuming infinite download speed and fast network ping responses and the *500ms* representing a level of latency still lower than what we have found on any existing online player.

Video Content

In video seeking tasks, the distinguishing feature of a target scene could have varying degrees of visibility. To examine this dimension, videos representing two levels of *discernibility* were selected for each *video type*; the *high discernibility* condition contained targets which were easier to recognize than the *low discernibility* conditions. We specifically chose videos that the subjects would not have seen prior to the study. Also, as described below, the experimental design tried to minimize learning effects from memorizing the video content.

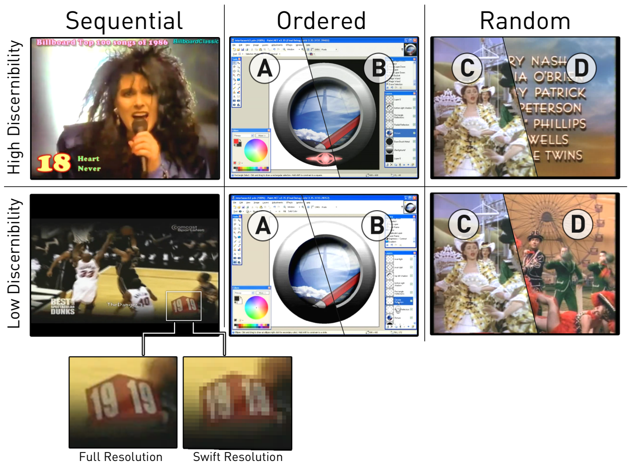

Figure 11. Frames taken from the target scenes for each *video type* and *discernibility* combination used in the study. For the *ordered* examples, (A) is from before the change occurring, and (B) is from after. In the *random* examples, (C) is a typical scene from the movie and (D) is the target scene.

SEQUENTIAL: The *sequential* videos used for the study were both "countdown" videos which presented a number of clips in decreasing numerical ranking. For the *high discernibility* condition, a countdown of the Top 25 Music Videos of 1986 was used (Figure 11). This video displayed the number of the current video prominently in the bottom left corner of the screen. Each video took the same fraction of time to play making the target size on the timeline 800/25 = 32 pixels wide. The *low discernibility* video was a countdown of the Top 50 Basketball Dunks. The decreasing numbers were shown on a slightly transparent rotating cube in the bottom right corner of the frame. The clips in this video were of varying length, but the ones used in the task each occupied 12 pixels on the timeline.

ORDERED: For the *ordered* condition we used a software tutorial video and simulated the situation where a user wants to find out how a particular piece of the design was created. Both the *high* and *low discernibility* videos were taken from a tutorial video of the drawing program Paint.NET. In the *high discernibility* condition the user needed to find when the background of the drawing changed from white to black, while in the *low discernibility* condition the user needed to find the point where the inner bevel was added to the porthole (Figure 11). Participants were not required to find the exact single frame where the change occurred, but were given a 5 pixel buffer on either side making for an 11 pixel range on the timeline.

To enable positioning the target point at different locations on the timeline, each of the *ordered* videos were constructed in three parts: a seamless loop of material before the change, a small section of video where the change occurred, and a seamless loop of material from after the change. From these "master" videos, a portion was trimmed from each end, positioning the change in the desired location; half of the trials occurred at a random location in the first half of the timeline, and the remainder occurred in the second half.

RANDOM: The video used for the *random* conditions was the 1946 movie "Till the Clouds Roll By" (Figure 11). To counter the potential learning bias of users memorizing the movie, participants were required to find one particular scene which was placed at a random location within the video. For the *high discernibility* condition the scene was the easily recognizable opening credits (taking up 32 pixels on the timeline), and for *low discernibility* the scene was a dance number where the actors were wearing red and green costumes (a 3 minute scene, taking 34 pixels).

Procedure

The examiner began by using a sample video to show each of the *technique/latency* combinations to the participant. The examiner demonstrated how each worked, and observed the user interacting with the player to ensure that they understood. The trials were ordered with *video type* at the outermost level, and *discernibility* at the second level. This created 6 occasions when a new video or target type would be introduced. At these times the examiner would verbally explain the video and target to accompany the written description presented on the screen. Four trials with a 0ms latency, full resolution video player were presented for the user to become accustomed to the new video and target content, and then the balance of the trials began.

The trial timing behavior and interaction instructions were the same as in the first study.

Results

As in the first study, the primary independent variable was *completion time* for each task. Repeated measure analysis of variance showed a main effect for *technique* ($F_{1,11}$ = 100.3, $p < .0001$) with means of 7.01s for *swift* and 12.51s for *traditional*. Additionally, significant effects were found for *video type* ($F_{2,22}$ = 91.6, $p < .0001$), *discernibility* ($F_{1,11}$ = 7.15, $p < .05$), and *sn-latency* ($F_{1,11}$ = 32.3, $p < .0001$).

Looking at the *technique* pairs for each of the *video type/discernibility/sn-latency* conditions (Figure 12) we see that in all cases the *Swift* technique performed faster than *traditional*. Post-hoc analysis shows the effect to be significant for all pairs except the *20ms* conditions in the *sequential* videos, and the *20ms/ordered/low* condition.

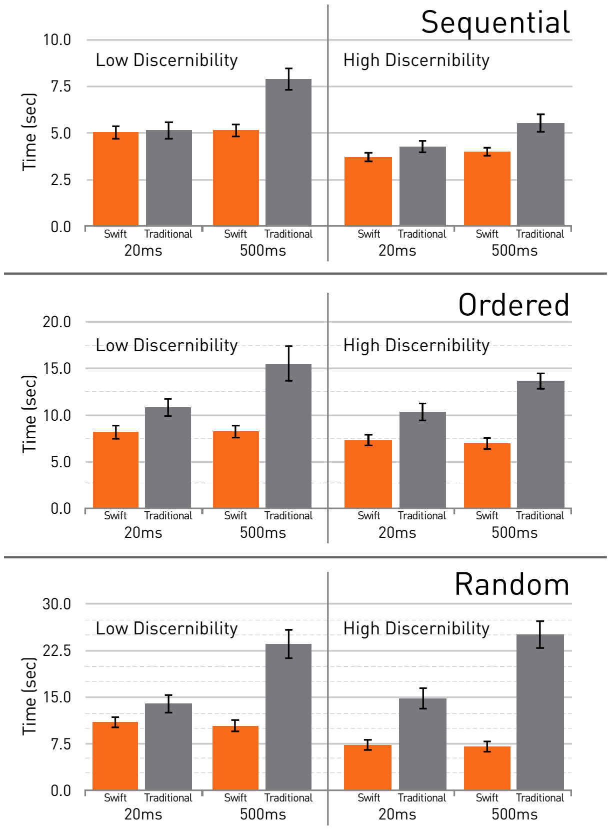

Figure 12. Results for the three video types. (Note: error bars report standard error).

It is interesting to see that for each *video type/discernibility* condition, the performance of the *Swift* technique stayed relatively constant across the two latency values. With the *traditional* player the completion times increased significantly overall from 9.9s in the *20ms* conditions to 15.1s during the *500ms* conditions ($F_{1,11}$ = 40.4, $p < .0001$). Based on the increasing trend of the results from the first study, it is reasonable to project that the gap in performance would continue to increase as the latency increased.

As in the first study, the overall task completion times increased as the tasks moved through the *video types* from *sequential* to *ordered* to *random*. As the tasks became more difficult, the benefit of the *Swift* technique became more pronounced, with *traditional* taking between 2 and 3.5 times as long as *Swift* in the *random/500ms* conditions. So as not to make the study unnecessarily hard, the target scenes were relatively long, and the movie relatively short. As the total length of the movie increases and the length of the target decreases, the benefits of *Swift* would become even more pronounced.

HTML5 IMPLEMENTATION

In this section we describe a simple HTML5 implementation of *Swift*, which demonstrates that the technique can work in today's web browsers. Two videos are rendered with the HTML5 <video> tag, with the small video above the large video, but initially invisible. The *Swift* technique is implemented with less than 20 lines of javascript code (Listing 1). A custom slider is configured to make the small-resolution video visible when sliding begins, and update its position as it slides. The position of the full resolution video is not updated until the sliding completes. The small-resolution video is not hidden until the full resolution video has finished seeking to the desired frame, resulting in a seamless transition between resolutions. Our testing of this code indicated that by default, the small and large video are downloaded in parallel, resulting in a close to instant download of the small video. However, further code could be investigated to force the initial download of the small-resolution video.

```
var small_length = 800;
var large_length = 3999.929;
var large = document.getElementById("lmovie");
var small = document.getElementById("smovie");
document.getElementById("slider").max = small_length;

function startSlide() {
  small.style.visibility = 'visible';
}

function Slide(newValue) {
  small.currentTime = newValue;
}

function endSlide() {
  var t = small.currentTime;
  large.currentTime = ((t) * (largelength)) / small_length;
}

function Seeked() {
  small.style.visibility = 'hidden';
}
```

```
<video STYLE="position:absolute;" id="lmovie" src="large.mp4"
    onseeked="Seeked()" width="800" height="600" preload controls>
</video>

<video STYLE="position:absolute; visibility:hidden" id="smovie"
    src="small.mp4" width="800" height="600" preload>
</video>

<input STYLE="position:absolute; TOP:608px; WIDTH:800px" type="range"
    id="slider" value="0" onchange="Slide(this.value)"
    onmousedown="startSlide()" onmouseup="endSlide()">
```

Listing 1. Javascript and HTML code for basic implementation of Swift technique.

Unfortunately, supported codecs and input elements still vary from one browser to the next. The listed code is fully functional in Google Chrome, and minor adjustments would be required for other HTML5 enabled browsers.

DISCUSSION AND LIMITATIONS

We have presented the empirical results from two novel experiments related to navigating online videos. Our first study demonstrated that even a small amount of network latency (20ms) can significantly hinder performance in video navigation tasks. Our second study demonstrated that real- time, low resolution scrubbing, significantly improves performance, in both high and low latency environments. The new empirical data provided by these studies will help practitioners and researchers better understand the benefits of enabling real-time scrubbing in online video players.

In addition to being beneficial in online environments, our results are applicable to desktop video players as well. Many such players still do not support real-time scrubbing, and only update their frames when a seek operation has completed. While transitioning to a lower resolution version in a desktop environment may not be necessary, this could actually improve the display rate of frames during a scrubbing operation, due to the reduced CPU load.

An important aspect of our implementation, *Swift*, is that it limits the download capacity required to enable real-time scrubbing to approximately 1MB, regardless of the source video's resolution, frame rate, and duration. As such, real-time scrubbing is available almost immediately when viewing videos with a broadband connection. We are unable to verify the Netflix implementation, but we did find it usually takes at least 10 seconds, and often several minutes before real-time scrubbing is enabled. Limiting the download size is also important as many internet providers are employing download caps and pay-per-use models.

Another advantage of *Swift* is its simple HTML5 compatibility. We demonstrated how real-time scrubbing could be enabled with less than 30 lines of HTML5 and javascript code. However, for a video sharing site to implement the technique, a service to create the low-resolution videos would be required. This should not be problematic, since sites, such as YouTube already have services to convert videos into multiple versions at different resolutions.

One potential limitation of low-resolution scrubbing is that it may be impossible to discern low-granularity details while scrubbing. Although prior research indicates very little resolution is required to identify features in images, small text fonts for example would be unreadable. Although we do not believe it is common for users to be searching for such fine grain details while navigating videos, it should be noted that low-resolution scrubbing would not aid such a task. In our future work section, we discuss possible ways for which fine grain details could be represented.

FUTURE WORK AND CONCLUSION

There are a number of other techniques in the literature that aid video navigation, although most do not focus on the scrubbing interaction. Low-resolution scrubbing could potentially be used in combination with these techniques. For example, Pongnumkul et al.'s content-aware dynamic timeline control [19] could be used while scrubbing, so that instead of frames flashing quickly by, salient scenes could be displayed at a more digestible rate. Additionally, direct manipulation video navigation systems such as DRAGON [13] and DimP [7] could utilize a low-resolution overlay.

Our implementation of *Swift* overlaid the low-resolution version of the video across the entire video player canvas. In contrast, the Netflix player displays multiple smaller thumbnails centered on the canvas. We did not become aware of the Netflix player until our studies were completed, but it would be interesting in the future to compare these two approaches. Another alternative design worth exploring is displaying a small thumbnail just above the timeline, offset from the cursor position. Some players, such as Hulu, already do this when hovering over the timeline, but do not pre-cache these thumbnails.

It would also be interesting to look at alternative low-data representations of the content while scrubbing, other than a literal down-sampling of the entire video. For example, the low resolution video could be a zoomed in view of the full resolution video, showing an area that has important details. Alternatively, metadata could be stored alongside the video and rendered instead of frames from the actual video. For example, when scrubbing through a sporting event, the current score or time remaining in the game could be overlaid. When scrubbing a movie or music video, the closed captions or lyrics could be displayed.

While our implementation used a fixed 1MB file size, our analysis of the H.264 codec performance showed that representations could be made as small as 29KB. To support low speed connections, it could be useful to have multiple low-resolution files available, and possibly progressively download and use larger versions.

Our study focused on scrubbing under uniform latency values while in practice, users may experience a range of latencies and this would be interesting to examine further.

Finally, we feel low-resolution scrubbing is particularly suited for mobile devices, as it reduces both bandwidth and CPU load. Our implementation should work with minimal modification on HTML5 supported mobile devices, such as the iPad, and it would be interesting to evaluate such an implementation.

To conclude, we have contributed empirical data demonstrating the impact of latency on online-video navigation tasks, demonstrated that low-resolution real-time scrubbing can significantly improve performance, and provided a simple HTML5 compatible implementation. Given today's prevalence of online streaming video sites, we feel these are important and timely contributions.

REFERENCES

1. Bachmann, T. (1991). Identification of spatially queatized tachistoscopic images of faces: How many pixels does it take to carry identity? *European J. of Cog. Psychology.* 3:85-103.
2. Bailer, W., Schober, C., and Thallinger, G. (2006). Video Content Browsing Based on Iterative Feature Clustering for Rushes Exploration. *TRECVID Workshop*. 230-239.
3. Chang, L., Yang, Y., and Hua, X.S. (2008). Smart Video Player. *IEEE Multimedia and Expo*. 1605-1606.
4. Chen, L., Chen, G.C., Xu, C.Z., March, J., and Benford, S. (2008). EmoPlayer: A Media Player for Video Clips with Affective Annotations. *Int. with Comp*. 20:17-28.
5. Divakaran, A. and Forlines, C. and Lanning, T. and Shipman, S. and Wittenburg, K. (2005). Augmenting Fast-Forward and Rewind for Personal Digital Video Recorders. *ICCE*. 43-44.
6. Doulamis, A.D. and Doulamis, N.D. (2004). Optimal Content-based Video Decomposition for Interactive Video Navigation. *IEEE CSVT.* 757-775.
7. Dragicevic, P., Ramos, G., Bibliowitcz, J., Nowrouzezahrai, D., Balakrishnan, R., and Singh, K. (2008). Video browsing by direct manipulation. *CHI.* 237-246.
8. Grossman, T., Matejka, J., and Fitzmaurice, G. (2010). Chronicle: Capture, Exploration, and Playback of Document Workflow Histories. *UIST*. 143-152.
9. Hanson, G. and Haridakis, P. (2008). YouTube Users Watching and Sharing the News: A Uses and Gratifications Approach. *Journal of Electronic Publishing*. 11:3.
10. Hefeeda, M. and Hsu, C.H. (2008). Rate-Distortion Optimized Streaming of Fine-Grained Scalable Video Sequences. *ACM TOMCCAP*. 1-28.
11. Holthe, O. and Ronningen, L.A. (2006). Video Browsing Techniques for Web Interfaces. *IEEE CCNC.* 1224-1228.
12. Hürst, W. (2006). Interactive Audio-Visual Video Browsing. *ACM MM*. 675-678.
13. Karrer, T., Weiss, M., Lee, E., and Borchers, J. (2008). DRAGON : A Direct Manipulation Interface for Frame-Accurate In-Scene Video Navigation. *CHI.* 247-250.
14. Janin, A., Gottlieb, L., and Friedland, G. (2010). Joke-o-Mat HD: Browsing Sitcoms with Human Derived Transcripts. *ACM MM*. 1591-1594.
15. Krasic, C. and Légaré, J.S. (2008). Interactivity and Scalability Enhancements for Quality-Adaptive Streaming. *MM*. 753-756.
16. MacKenzie, I.S. and Ware, C. (1993). Lag as a Determinant of Human Performance in Interactive Systems. *CHI*. 488-493.
17. Pavlovych, A. and Stuerzlinger, W. (2009). The Tradeoff between Spatial Jitter and Latency in Pointing Tasks. *ACM Symposium on Eng. Interactive Comp. Syst*. 33-44.
18. Pavlovych, A. and Stuerzlinger, W. (2011). Target Following Performance in the Presence of Latency, Jitter, and Signal Dropouts. *GI*. 33-44.
19. Pongnumkul, S., Wang, J., Ramos, G., and Cohen, M. (2010). Content-Aware dynamic timeline for video browsing. *ACM UIST.* 139-142.
20. Ramos, G. and Balakrishnan, R. (2003). Fluid Interaction Techniques for the Control and Annotation of Digital Video. *ACM UIST*. 105-114.
21. Schoeffmann, K., Hopfgartner, F., Marques, O., Boeszoermenyi, L., and Jose, J.M. (2010). Video browsing interfaces and applications: a review. *SPIE Reviews*. 18004:1-35.
22. Schoeffmann, K., Taschwer, M., and Boeszoermenyi, L. (2010). The Video Explorer – A Tool for Navigation and Searching within a Single Video based on Fast Content Analysis. *ACM SIGMM*. 247-258.
23. Torralba, A., Fergus, R., and Freeman, W.T. (2008). 80 million tiny images: A large data set for non-parametric object and scene recognition. *IEEE Pattern Analysis and Machine Intelligence.* 1958-1970.
24. Truong, B.T. and Venkatesh, S. (2007). Video Abstraction: A Systematic Review and Classification. *ACM TOMCCAP.* Appl. 3, 1, Article 3.
25. Uchihashi, S., Foote, J., Girgensohn, A., and Boreczky, J. (1999). Video Manga: Generating Semantically Meaningful Video Summaries. *ACM CHI.* 383-392.