Content-Aware Dynamic Timeline for Video Browsing

Suporn Pongnumkul
Computer Science & Engineering
University of Washington
Seattle WA 98195
suporn@cs.washington.edu

Jue Wang
Adobe Systems
801 N. 34th Street
Seattle, WA 98103
juewang@adobe.com

Gonzalo Ramos
Microsoft Corporation
Michael Cohen
Microsoft Research
One Microsoft Way
Redmond, WA 98052
{gonzalo, mcohen}@microsoft.com

ABSTRACT
When browsing a long video using a traditional timeline slider control, its effectiveness and precision degrade as a video's length grows. When browsing videos with more frames than pixels in the slider, aside from some frames being inaccessible, scrolling actions cause sudden jumps in a video's continuity as well as video frames to flash by too fast for one to assess the content. We propose a *content-aware dynamic timeline* control that is designed to overcome these limitations. Our timeline control decouples video speed and playback speed, and leverages video content analysis to allow salient shots to be presented at an intelligible speed. Our control also takes advantage of previous work on elastic sliders, which allows us to produce an accurate navigation control.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Algorithms, Human Factors

Keywords: Timeline, Slider, Dynamic video skims

INTRODUCTION
Traditional video players support browsing, skimming and seeking through a timeline control. The control provides users a way to directly specify the video's current frame and a direct visualization of its location in the video's timeline. While simple and direct to use, timeline sliders' functionality and effectiveness degrade particularly for longer videos where there are many more video frames than there are pixels along a slider. These cases not only make many video frames unreachable, but also translate into scrolling actions causing sudden jumps in a video's continuity or video frames to flash by too fast to be understandable.

We propose an enhanced timeline control, the *content-aware dynamic timeline*, which enables high precision navigation as well as fast browsing and seeking functionality.

Our control is influenced by elastic graphical interfaces such as [5], where playback speed is dynamically adjusted as a nonlinear function of the distance between the handle and the mouse pointer. We also borrow from the PVslider [9], which uses the vertical distance between the cursor and the control to adjust the control-display ratio. Our dynamic timeline control presents a similar interactive behavior for precise video frame selecting. However, unlike the aforementioned work, the playback speed computed by the elastic function is not used to directly determine the video frame-rate. Instead, our timeline control employs a content-aware video skimming method to present the user a meaningful abstraction of the video when the user rapidly scrubs the video. By doing this, we effectively decouple average video speed and frame-to-frame playback speed, providing users discernable frames as they skim through a video.

UIST'10, October 3–6, 2010, New York, New York, USA.

CONTENT-AWARE DYNAMIC TIMELINE
From informal observation of three users while they were watching and interacting with online video content, we found that, aside from simply watching a video, they engaged in two distinct modes of interaction: (1) fast-speed skimming through a long video, or (2) low-speed navigation in a small section of the video. Fast skimming occurred in situations where the user was unfamiliar with the video, and wanted to get a quick overview of its content and identify some interesting parts. Low-speed navigation occurred when users wanted to accurately locate a specific frame or time code in the video, e.g., the beginning of a scene, or a representative frame of a shot. In long video browsing sessions, we observed users engage in these two modes in concert and often switch between them. We also observed that at playback speeds above eight times the normal rate, many videos lost their temporal coherence to the point that they appeared more like rapid sequences of random frames. With the above observations in mind, we set our design goals for the timeline control to have the following properties:

1. Presenting a meaningful visual abstraction of the video content in the fast skimming mode.
2. Providing accurate frame location in slow speed navigation mode.
3. Being efficient at both fast- and low-speed browsing modes, and allowing for a smooth transition between them.

In addition to the above goals, we also want to maintain a clean design consisting of a single timeline slider without the added clutter that summary slides of key frames or chapter headings can introduce. In this section, we will first describe how we make use of a nonlinear function to control the *video speed* so as to allow a smooth transition between high-speed

browsing and low-speed frame seeking operations in a single drag-and-pull interaction. Note that the video speed represents a *desired* speed as indicated by the user via the slider interaction. Given the desired video speed, we are still free to choose the sequence of frames to progress through the video to achieve that speed *on average*. We show that by decoupling the video speed from the selected frames to play back, our timeline presents an intelligible summarization of the underlying video content even when quickly skimming through the video.

Elastic Timeline

Our timeline has the same appearance as a traditional timeline; when the user clicks on the slider, the video jumps to the corresponding frame. However, when the user drags the handle and moves along the timeline, the handle does not stay directly attached to the mouse pointer. Instead it follows the mouse pointer though a nonlinear speed function. At any time t, the current (desired) video speed V_s is determined by two distances: the horizontal distance between the mouse pointer and the current handle position d_x, and the vertical distance between the mouse pointer and the timeline horizon d_y. Similar to the elastic skimming slider proposed in [6], V_s is a function of d_x, similar in feel to connecting the mouse pointer and the handle with a rubber band.

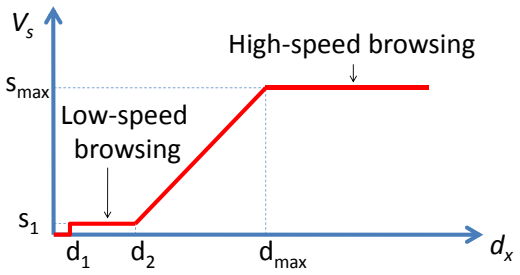

Figure 1: The function of normalized video speed V_s vs. the normalized horizontal distance between the mouse pointer and the handle d_x.

The exact shape of the response function $V_s(d_x)$ is illustrated in Figure 1. If $d_x < d_1$, we interpret it as an unintentional move, in which the speed is reduced to zero. Beyond this distance, $V_s(d_x)$ is designed to allow the user to interact with the video in the two interaction modes we identified earlier, and smoothly transition between them. In low-speed browsing, the function maintains a constant low speed s_1, as a user keeps the mouse pointer close to the handle ($d_1 < d_x < d_2$). When the user pulls the mouse pointer far away ($d_x > d_{max}$) from the handle, the timeline enters the high-speed browsing mode at a constant speed s_{max}. This allows the system to render a quick overview of the video content in a short period of time. The video speed is linearly interpolated between s_1 and s_{max}.

In our implementation d_1, d_2 and d_{max} are set to be 0.001, 0.01 and 0.25, respectively, which are normalized distances by using the total length of the slider as the normalizer. s_1 is set to be 0.2 times normal speed to allow for slow motion playback. s_{max} is normalized to the total length of the video such that the whole video would play in 20 seconds, i.e., $s_{max} = T_{max}/20$, where T_{max} equals the totals length of the video in seconds. These values are empirically chosen through observation. Normalizing most parameters to the lengths of the slider and video gives users similar browsing experiences even when the length of the video or the size of the control itself changes.

Once an initial $V_s(d_x)$ is determined, it is further modulated by the vertical distance d_y between the mouse pointer and the timeline horizon as

$$V_s = \frac{d_x + \lambda d_y}{d_x} \cdot V_s(d_x). \quad (1)$$

Since a comfortable video speed is subjective, and is dependent on the video content, this design allows the user to adjust the video speed by moving the mouse pointer up or down. This is similar to the Zoomslider interface in [6] where moving mouse up and down changes the scale of the timeline.

Dynamic Video Skims

The elastic and the traditional timelines face similar problems as the video is played at high speed. In both sliders, high-speed browsing will display a sequence of unrelated frames, as the video is uniformly sampled. Flashing images degrade the experience a video has the ability to provide, which is the smooth continuity of frames.

Our content-aware dynamic timeline extends the elastic timeline to solve this problem by decoupling the video speed from playback speed, and adding content-awareness. *Video speed*, V_s, instead is defined to control the average speedup of the video. In other words, a video speed of 4x indicates the video would take $1/4^{th}$ the time to play as a video at normal speed, 1x. *Playback speed*, in contrast, is the input frame difference between consecutively played output frames.

The decoupling of video speed and playback speed provides a degree of freedom we can leverage. While we can achieve a video speed of 10x simply by having a playback speed of 10x (i.e., the playback includes every 10^{th} frame), we can also achieve the same video speed of 10x by holding a single frame for n playback frame times, say 15 (equivalent to 1/2 second of playback time), and then jumping ahead $10n$ frames, and repeating. Holding for more than one frame provides the user a better chance to understand the content, albeit at the cost of showing fewer distinct frames.

There is a third option that we chose for our dynamic video skims. Rather than holding a single frame, we play a short (1 second) clip of video at a playback speed of 2x, i.e., for a total of 1/2 second. This is followed by a discrete jump forward to maintain an average of the desired video speed. For example, if $V_s = 10$x as in the example above, the 1/2 second (15 output frames) clip would be followed by a jump forward of approximately 150 input frames to the next clip. This approach allows selected clips to be displayed at an intelligible speed at the cost of throwing away less important information, thus providing a smoother experience of skimming the video. This approach is illustrated in Figure 2.

Selecting Key-Clips via Content Analysis

We still have one more degree of freedom in choosing the exact clip to show after each jump forward. Rather than using

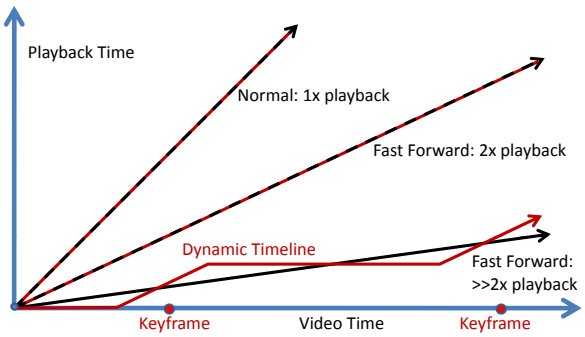

Figure 2: The illustration of dynamic timeline. Black line shows the naive way to display video given the desired video speed. Red lines shows the dynamic timeline's approach. The red-black dash lines show when the two approaches agree. At 1x, the playback time matches the video time, as illustrated with the top slope. At 2x, the playback time is half of the video time. As the video speed exceeds 8x, instead of naive uniform sampling, the dynamic timeline displays selects clips to play at 2x. This is followed by a jump to the next clip, to maintain the desired video speed.

a fixed jump, our timeline improves the online video skim generation by using the video content analysis results. As illustrated in Figure 3, we first divide the video into coherent shots (second row from the bottom). Each shot is associated with a representative keyframe. We create a *key-clip* of 1 second surrounding each keyframe. Then, instead of jumping ahead a fixed number of frames, we choose the nearest keyframe to the fixed-distance frame and play the associated key-clip. A video skim is thus dynamically generated by chaining selected key-clips together.

Using only the one layer of shot decomposition, however, is not sufficient to generate a good video skim. Just as we segment shots from which to choose key-clips, we also cluster multiple keyframes to create a higher level decomposition. This process continues to form a hierarchy of key-clips as shown in Figure 3. The lowest level, L_0 of the hierarchy contains the raw frames. The next level, L_1 contains one key-clip per shot. Each succeeding level of the hierarchy is formed from the most distinct $1/3^{rd}$ of the key-clips from the level below. This hierarchy is used to select the highest value key-clips for display during a very fast skim.

The dynamic video skim is generated in the following way: based on the dynamic slider that defines V_s and the passage of clock time we always have a *desired frame time*, F^*.

$$F^* = \Sigma_{t=0}^{T} V_s \tag{2}$$

where T is the total time since starting, and the summation is taken over each output frame time. F^* is the frame that would be played next in a standard video player. We also have the most recently played frame, F^{-1}. If the difference in these is less than 8, then we simply play frame F^*. In other words the fastest standard fast-forward is set to 8x, thus if $V_s < 8$ then the result is the same as for an elastic slider.

Otherwise, ($V_s > 8$), we look higher in the hierarchy to select a key-clip to play. Moving up from L_0 to L_1 we tentatively select the closest keyframe to F^*. If the selected keyframe involves skipping over any keyframes at L_1, in other words if there are keyframes between F^{-1} and the chosen keyframe closest to F^*, then we move up the hierarchy once again, and repeat. We again choose the closest keyframe on this higher level, and check to see that no keyframes at this level have been skipped over. Once no keyframes are skipped, or we reach the highest level of the hierarchy, we play the corresponding key-clip. At the end of this key-clip, time has passed, thus F^* is updated, and the process repeats.

Intuitively, as shown in Figure 3, when the video speed is high, the skim will be generated from key-clips at higher levels of the hierarchy, thus only very important key-clips will be chosen to play. When the video speed is low, the skim will be generated at lower levels of the hierarchy to play more scenes. If the video speed is low enough when the handle approaches the mouse pointer, the skim is essentially assembled at the bottom level where every frame is available to be chosen to display.

Note that in this design some less interesting lower level shots will be skipped when the video speed is high. However, since the skims are dynamically generated, when the user scrolls through the same portion of video at different speeds, key-clips at different levels will be revealed. This contrasts with systems that statically define a sequence of clips to play.

Building the key-clip hierarchy

We leverage existing methods to build the content hierarchy. We first employ shot boundary detection, using code from the Compression Project [1]. The middle frame in each shot is selected as its keyframe. We then treat all keyframes as a shorter video, and apply a color histogram-based clustering method [8] to compute the next level of the hierarchy such that it contains $1/3^{rd}$ the keyframes of the level below. This process iterates until the whole video is clustered to a single shot. Of course our system is not limited to any single content analysis method. Alternatively, if semantic annotations provided by users are available, they can be used as constraints for hierarchy construction as well.

RELATED WORK

Many techniques have been proposed to assist users to quickly browse through a video. One approach is to provide extra semantic information through additional graphical elements (e.g. keyframes [4]). In contrast to the work in this category, we aim to keep the look-and-feel of our slider control clean and only modify the behavior of the timeline control itself.

Another body of work for efficient video browsing is creating video skims. Smartplayer system [1] achieves this by automatically adjusting playback speed based on the complexity of the current scene, the predefined semantics, and the learnt user preferences. This approach requires minimal user interaction, but is not efficient for tasks requiring active and precise user control, and cannot avoid high frame-rate fast forward which may cause disorientation. Other systems

[1] http://compression.ru/video/quality_measure/video_measurement_tool_en.html

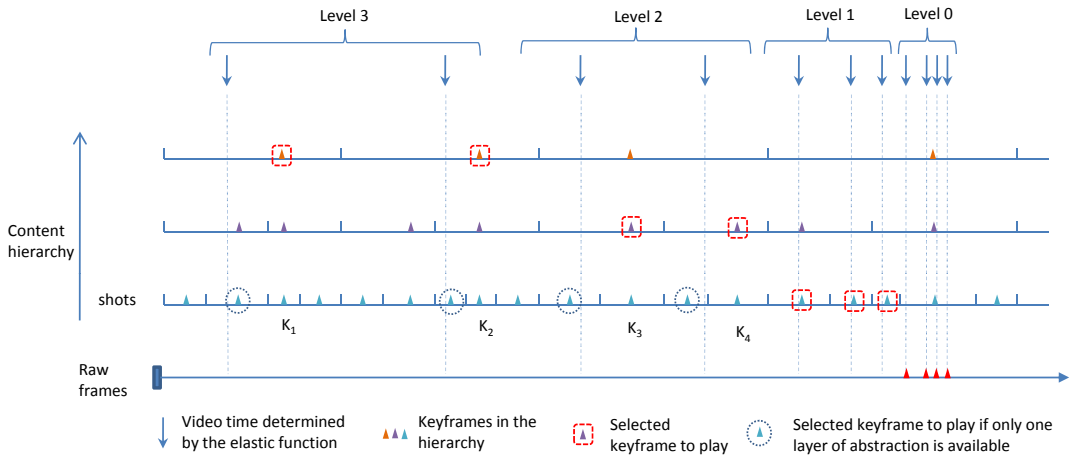

Figure 3: Dynamic skim generation. This diagram illustrates how different frames will be shown in three conditions (1). Without content analysis, video frames are sampled based on the video speed, as indicated by the blue arrow pointing down. (2). With shot segmentation and keyframe extraction, key-clips are selected to form a skim, highlighed by circular purple dots. Note that K_1, K_2, K_3 and K_4 are skipped. (3). Skim generation with a content hierarchy. Key-clips from different levels are connected together to form a skim, highlighted by red rectangular dots.

pick only "important" or "relevant" video segments to play at a normal speed, such as [2]. More complex hierarchical structure of the video content is used in [7] for skim generation. The video abstraction method embedded in our timeline falls into this category. However, the major difference between our approach and previous skimming methods is that our skim is dynamically created online as the user scrolls through the video, thus many different skims can be generated in a single user interaction session.

Direct video manipulation, such as [3] where a user can directly specify an object's position in a video frame, works well for spatial queries in short videos whose visual content in question appear in most video frames. Such techniques, while complementary, do not apply to the problem we are addressing, which is skimming very long unknown videos with many scene changes.

CONCLUSIONS AND FUTURE WORK

We present a novel dynamic timeline control for efficient video browsing. Our main contribution is the decoupling of video speed and playback speed, which allows a meaningful visual abstraction of the video content to be dynamically created, as users skim through a video. At the same time, we leverage existing elastic slider techniques to enable high precision video navigation. The two navigation modes are smoothly combined in a single timeline control.

Constraining ourselves to a single timeline affordance, we focus on how to select video frames to display when the user interacts with the timeline. However, a soundtrack can be processed in a similar way, to provide an audio abstraction of the corresponding video shot. In the supplemental video we show such an example. We are also aware that other visualization techniques can be used to create a visual abstraction of the video when the video speed is high, such as showing multiple smaller frames on the play window to give the user a fast-forward feeling. We plan to explore these possibilities for further improving the user experience of the system.

During our control's design and testing periods we were able to capture preliminary informal user feedback that underlined the potential of our dynamic timeline control. We are now in the early stages of performing quantitative studies to assess our control's performance in contrast to traditional timeline slider controls.

REFERENCES

1. K.-Y. Cheng, S.-J. Luo, B.-Y. Chen, and H.-H. Chu. Smartplayer: user-centric video fast-forwarding. In *Proceedings of CHI*, pages 789–798, 2009.
2. F. Coldefy and P. Bouthemy. Unsupervised soccer video abstraction based on pitch, dominant color and camera motion analysis. In *Proceedings of ACM Multimedia*, pages 268–271, 2004.
3. P. Dragicevic, G. Ramos, J. Bibliowitcz, D. Nowrouzezahrai, R. Balakrishnan, and K. Singh. Video browsing by direct manipulation. In *Proceedings of CHI*, pages 237–246, New York, NY, USA, 2008. ACM.
4. O.-I. Holthe and L. A. Ronningen. Video browsing techniques for web interfaces. *Consumer Communications and Networking Conference*, 2:1224–1228, 2006.
5. W. Hürst. Interactive audio-visual video browsing. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 675–678, 2006.
6. W. Hürst, G. Götz, and P. Jarvers. Advanced user interfaces for dynamic video browsing. In *Proceedings of ACM MM*, pages 742–743, 2004.
7. S. Lu, I. King, and M. R. Lyu. Video summarization by video structure analysis and graph optimization. In *Proceedings of ICME*, 2004.
8. L. Ott, P. Lambert, B. E. Ionescu, and D. Coquin. Animation movie abstraction: Key frame adaptative selection based on color histogram filtering. In *Proceedings of ICIAPW*, pages 206–211, 2007.
9. G. Ramos and R. Balakrishnan. Fluid interaction techniques for the control and annotation of digital video. In *Proceedings of UIST*, pages 105–114, 2003.