AN APPLICATION FOR INTERACTIVE VIDEO ABSTRACTION

Sangkeun Lee, Monson H. Hayes

Georgia Institute of Techonology, Center for Signal and Image Processing,
Atlanta, GA 30325, U.S.A
gte392q@prism.gatech.edu, mhh3@ece.gatech.edu

ABSTRACT

This paper proposes a novel system that can automatically create an optimal and nonrepetitive summarization and support different user requirements for video browsing and content overview by outputting both the optimal set of key frames and a summarized version of the original video with the user-specified time length. Comparing our approach to video abstraction with another algorithm, we demonstrate that our approach is fast and produces an effective video summary.

1. INTRODUCTION

Recently, we have witnessed a rapid increase in the production, transmission, and storage of multimedia information. Of all the media types, video is the most challenging as it combines all other media information into a single data stream. However, efficient access to video is not an easy task due to its length and unstructured format. In order to efficiently search a vast video library or digital archive to retrieve specific shots, scenes, or entire movies, it is important to be able to produce a summary of a video, either in terms of a set of descriptive frames or a set of short video clips. Producing such a summary is generically referred to as video abstraction. A video abstraction is a compact representation of a video sequence. There are two general types of video abstractions: video summarization and video skimming [1]. Video summarization [2], sometimes referred to as "still-image abstraction", involves the creation of a small collection of representative images that are extracted or generated from the underlying video sequence. Once the video summary has been created, it may be easily displayed or presented to the user since it consists simply of a storyboard or gallery of images. These images may be arranged in a number of ways. The second form of video abstraction is called video skimming, which is also referred to as "moving-image abstraction". In video skimming [7][8], a collection of image sequences are extracted from the video along with the corresponding audio tracks. Although generally taking more consuming than video summarization, video skimming has the advantage of using audio tracks, which may contain important information, such as in education and training videos. Video skimming also has advantages during play back, since it is usually more natural and interesting for users to watch a trailer than to watch a slide show and, in many cases, the motion that is displayed is information-bearing.

We have developed a video summarization system that is designed to quickly and efficiently produce a hierarchical visual display of key frames that are extracted from the video. In this paper, we describe the last building block of the proposed video abstraction system. The focus is on the overall speed of summary generation.

This paper is organized as follows. In Section 2, we summarize our system for video summarization. In Section 3, the proposed structure for video abstraction is presented. Section 4 illustrates how to make a video abstraction with user interaction and interface. To evaluate the overall abstraction system, this work is compared with another algorithm based on the speed of summarizing a video in Section 5, and in Section 6 we provide a summary of our paper.

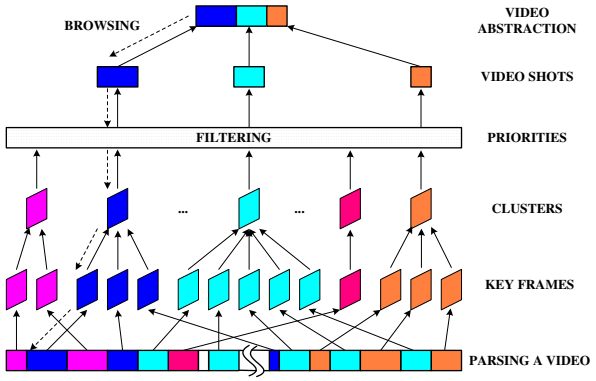


Fig. 1. Procedures for multi-level video abstraction and browsing.

2. VIDEO ABSTRACTION SYSTEM

The proposed system itself is composed of a number of building blocks. The building blocks indicate the algorithms developed for quickly summarizing a video with little redundancy. This system consists of two main parts: video analysis and video abstraction. In the video analysis part, the key-frame extraction module begins by detecting shot boundaries [3]. Each shot is analyzed for camera movement [4] and divided, if appropriate, into subshots. Key frames are then selected according to the complexity of the shots [11]. In the video abstraction part, the key frames that have been extracted in the video analysis part are used as the input to a fast clustering algorithm [11]. Here, the goal is to cluster similar key frames so that a more compact summary may be created for high-level summarization. Finally, a non-repetitive summary of the video content from the clustered key frames is created. This summarization is designed to support different user requirements for video browsing and content overview by outputting both the set of key frames and a summarized version of the original video

using a multi-resolution structure. In addition, a simple and efficient algorithm for human face detection [10] has been used to help users with more content-based searching. The procedures of summarizing and browsing contents of a video is illustrated in Figure 1. The solid arrow indicates the direction of abstraction, while the dotted arrow indicates the direction of browsing contents of a video.

3. MULTI-RESOLUTION SUMMARIES

The video abstraction architecture is shown in Figure 2 and consists of three categories: media, user, and database. The media component includes the video sequences, which are manually classified according to their genre. The user component indicates the activity of users searching and selecting relevant video on a local workstation or through the Internet. In the database component, it is assumed that the video key frames are clustered and have priorities assigned to them according to different characteristics such as length of a shot, the number of key frames of the shot, human presence, and camera motions. Let $\mathbf{\Omega}$ be a set of the refined key frames

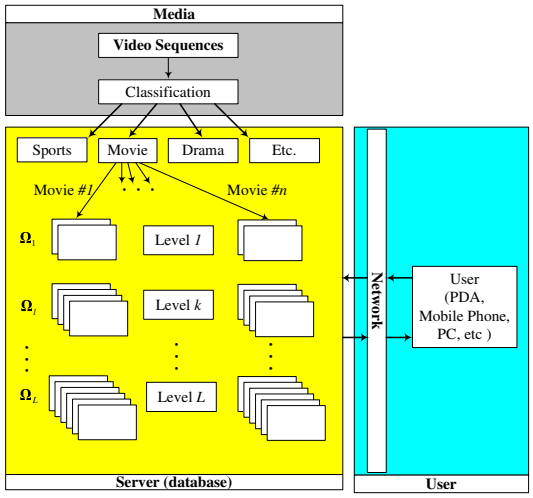


Fig. 2. Overall structure of the abstraction system.

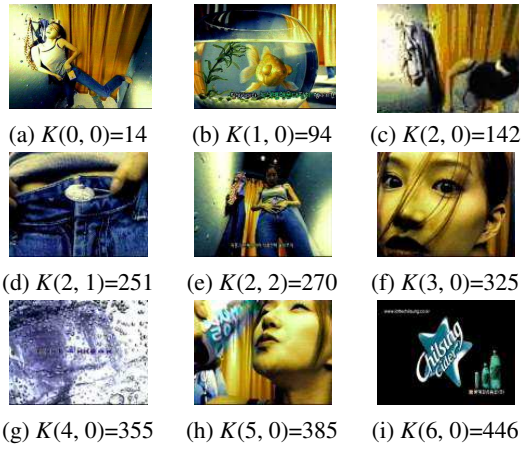
in a video arranged with temporal order, and $\mathbf{\Omega}_L = [f_1, \cdots, f_p]$ be a set of p key frames. L is the number of summary levels, and f_x, which is the temporal position of the original video, is the x^{th} key frame. At any level, the summarization can be represented as

$$\mathbf{\Omega}_k = [f_1, \cdots, f_q], \text{ where } 1 \leq k \leq L \text{ and } 1 \leq q \leq p. \quad (1)$$

The following equation defines the number of key frames to be selected at level k:

$$N_k = \lfloor p \times \tfrac{k}{L} \rfloor, \quad (2)$$

where $1 < k \leq L$, and $\lfloor \cdot \rfloor$ is the rounding operator. For the first level, this system selects the most static key frame to show users the representative image at the beginning of their video browsing. For level k, the system picks N_k key frames according to their content activities but produces a summarization according to their temporal orders. Finally, level L must have all of the key frames, i.e. $\mathbf{\Omega}_L = \mathbf{\Omega}$. The system also provides a summarized video clip, which can be available at each level, for the user's convenience. The short video clip can be understood as a temporal extension of the key frames at each representation level. Each shot or subshot, to which at least one extracted key frame belongs, is taken as a key video segment. These key segments are then concatenated to form the short video clip. It is preferable to use entire shots or subshots for making a clip due to their complete contexts. Such completeness makes the short video clip understandable. An alternative to this is to use only extracted key frames, and assign an equal or different time length depending on the duration of their shot or subshot in an original video. However, the probability of having a complete context is considerably lower in this case even though it is simple way. The following section describes the procedure to select the representative images at each level from user interactions.



(a) $K(0, 0)$=14 (b) $K(1, 0)$=94 (c) $K(2, 0)$=142

(d) $K(2, 1)$=251 (e) $K(2, 2)$=270 (f) $K(3, 0)$=325

(g) $K(4, 0)$=355 (h) $K(5, 0)$=385 (i) $K(6, 0)$=446

Fig. 3. An example of refined key frames for a commercial film.

4. USER INTERACTION/SYSTEM INTERFACE

In order to show how users (clients) interact with the database server to find their intended information, we use the result clustered key frames in Figure 3 for a commercial film (CF) sequence. Let $K(m, n)$ be the temporal position of the refined n^{th} key frame at the m^{th} shot. The characteristic of this sequence has two subshots of a zoom-in and a zoom-out camera operation, sequentially, at the third shot $K(2, 0)$. It is assumed that the characteristics of the key frames are given as shown in Table 1. In practice, the characteristics are assigned numerical values from the proposed system.

Table 1. Characteristics of the refined key frames.

Frame	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
Human	Low	No	Low	No	Low	High	No	High	No
Motion	Low	Low	High	High	High	Low	Low	Low	Low

If a client wants to have the information according to the degree of no-motion and human content in a video (in this example, the human content is weighted more heavily than the camera motion content), and chooses three levels ($L = 3$), the first

level ($k = 1$) is $\mathbf{\Omega}_1 = [f_6]$, where f_6 and f_8 were candidates but f_6 was chosen because it came first temporally. Similarly, the rest of the levels are $\mathbf{\Omega}_2 = [f_1, f_3, f_4, f_5, f_6, f_8]$ and $\mathbf{\Omega}_3 = [f_1, \cdots, f_9]$ from Equation (2). The summary example is shown in Figure 4. The elements at each level may be changed based on priorities emphasized more or less by the client.

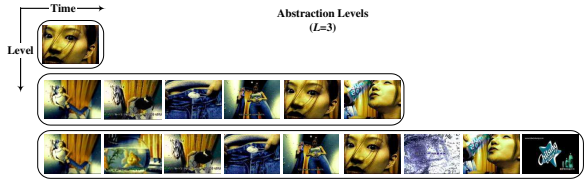


Fig. 4. Video summary example.

This architecture enables users to browse and find relevant data efficiently within the limited bandwidth, even though users go through several rounds of interactions with the system in their searching process for desired information in the worst case using the key-frame sets. In practice, the database (a server) only needs the key-frame set $\mathbf{\Omega}$ because any level of summarization can be performed at the user's request and be displayed on the user's displaying device.

The proposed system also provides an interactive interface to correct the results of the developed algorithms in Section 2 as an option, even though the default operation of the system is fully automatic. Errors produced by any one of the automatic algorithms will propagate, resulting in event boundaries that are incorrect. Therefore, it is important to provide interfaces so that a human operator (user) can verify and correct the results produced automatically at the video analysis and clustering steps. The proposed system does not use predefined video structures and domain knowledge, such as the broadcasting program and its story structure models to facilitate story segmentation. Hence, the proposed approach is not domain or program specific. The content produced from the analysis and clustering algorithm allows scalable summary generation. Thus, the video summary can be tailored according to the client's profile and needs.

Now, the tools provided to the system user in the server side are described to modify the results of automatic story segmentation at both the shot and event levels. The steps in the interactive generation of the final organized video can be summarized as follows:

1. Automatic construction of a *shotlist*, which contains the information of (sub)shot lists, from the video analysis, clustering, and human face detection algorithms on the compressed video.
2. Viewing and editing the (sub)shot change structure and clustering results to add new (sub)shots or merge (sub)shots.
3. Automatic generation of multimedia video summary, where the generated video clip is used as a short clip at the highest level of the multi-resolution summary, while the generated key frames are used as an input for clients' browsing to find their relevant information.

The second step in the list above requires user (human operator: server side) interaction, and therefore, an interface needs to be provided with the required functionality. Since these interfaces work with higher level representations of the video, a separate component is also provided to view the raw video. The proposed system provides three main interfaces to the user (server side), which communicate with each other so that changes made using one component produce the appropriate updates in the other interfaces.

Analyzer: This interface is used to analyze video contents, to show the intermediate graphs, and to produce *shotlists*. The video stream is represented as key frames containing all information such as duration of (sub)shot and human content.

Video player: This interface is used for playing the video from any point in the video. It has the functionality of a VCR including fast forward, rewind, pause, and step.

Listview: This interface is used to view and alter the automatic grouping of (sub)shots based on visual similarity. Once the similarity-based clustering results have been finalized, they are used as an interface to correct event (or story) boundaries. It also provides extra information, which includes an icon of each shot in temporal order; the length of the shot; all of the key frames of the shot; and motion, human, and cluster information. A screen-capture illustrating the *Listview* interface is shown in Figure 5. The user is allowed full freedom in changing the event boundaries and other information, since semantic information can often be missed or misinterpreted by automatic processing. The following operations are provided to facilitate changes on an item of the list:

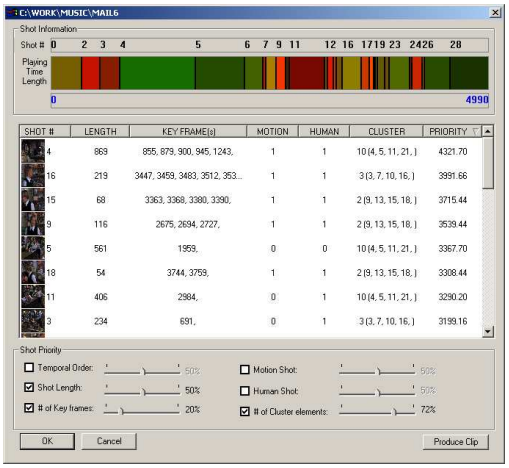


Fig. 5. A screen-capture illustrating the *Listview* interface after sorting of the priority column in descending order.

- Sort items: Each row lists all of the information extracted from the algorithm described in the previous sections. Rows can be sorted by clicking a column to sort.
- Add/delete rows: New rows can be added below the destination row. Alternatively, unnecessary rows can be removed by the user.
- Change priorities: Users can change the value of each priority as is necessary for their own application. Af-

ter changing the values of priorities and sorting a priority column in the *ListView*, they can overview or make a short clip with desired contents. For example, a short clip can be made only with two significant shots based on length of a shot and human appearance. Priorities provided in the proposed system are temporal order of a shot, length of a shot, number of key frames, camera motion, human presence, and size of cluster.

The user (server side) can invoke these operations to regroup the stories into more meaningful stories and sub-stories. The order of stories can also be changed from their usual temporal order to a more logical sequence. Though the primary function of the *Listview* interface is to interact with the high-level structure, the same interface can also be used to view and modify the groups generated by the automatic clustering process.

5. EVALUATION

To evaluate the performance of the proposed automatic video abstraction system, the overall speed of the resulting summaries is focused on.

To show that the proposed system can be applied to any type of video sequence, four different compressed MPEG-1 sequences are considered. These sequences include a movie sequence, a music video, a documentary, and a CF sequence. All algorithms were implemented in C++ and run on a Windows system with a 400 MHz Pentium-II processor.

For an overall system performance comparison, the successful system by Huang *et al.* [6] is implemented as a baseline system. They proposed extracting the key frames using an unsupervised clustering scheme. Basically, all video frames within a shot, which is segmented by a shot change detection algorithm [9], are first clustered into certain number of clusters based on the color histogram similarity comparison where a pre-defined threshold controls the density of each cluster. Next, all the clusters that are large enough are considered as the key clusters, and a representative frame closest to the cluster centroid is extracted from each of them. To implement their system, we use the color histogram with DC-images [5] to speed up the processing time and set the threshold to 0.9. The overall processing time comparison is shown in Table 2 for four different video sequences. The average speed of the proposed system is almost 9 times faster than real-time and almost 19 times faster than that of the baseline system implemented. The proposed system and the baseline system depend on the results of a shot segmentation algorithm before clustering, but the baseline system can compensate for this algorithm during key-frame extraction based on visual content similarities within a shot, where two different shots may be concatenated unintentionally by the segmentation algorithm. However, the baseline system is less effective with respect to the compactness of video summaries since it does not merge visually similar shots that are separated by other shots, such as the shot changes of dialogue shots where the camera switches from speaker to speaker.

Table 2. Overall speed comparison between the proposed system and Huang' system.

Test sequences	Time length	Proposed system (ratio)	Huang' system (ratio)
Movie	30 min.	3.38 min.(8.88)	66.05 min. (0.45)
Music video	4 min.	0.42 min. (9.52)	8.92 min (0.48)
Documentary	10 min.	1.09 min. (9.17)	20.96 min (0.48)
CF	15 sec.	1.56 sec. (9.62)	32.96 sec. (0.46)
Average	11.06 min	1.23 min (8.99)	24.12 min (0.46)

6. SUMMARY AND REMARKS

This paper describes an abstraction structure using user interaction as the last step for video abstraction. For an overall system speed comparison, another system is implemented and compared with the proposed system based on speed. Performance results show that the proposed system is almost 9 times faster than real-time (playing time) and almost 19 times faster than the compared system.

7. REFERENCES

[1] Y. Li, T. Zhang, and D. Tretter, "An overview of video abstraction techniques", *HP Laboratories Palo Alto*, HPL-2001-191, July 2001.

[2] A. Hanjalic and H. J. Zhang, "An integrated scheme for automated video abstraction based on unsupervised cluster-validity analysis," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 9, no. 8, Dec. 1999.

[3] S.-K. Lee and M. H. Hayes, "Efficient scene segmentation for content-based indexing in compressed domain," *IEEE 2001 Workshop on Multimedia Signal Processing*, Oct. 2001.

[4] S.-K. Lee and M. H. Hayes, "Real-time camera motion classification for content-based indexing and retrieval using templates," in *Proc. of IEEE int. Conf. Acoustics, Speech, and Signal Processing*, May 2002.

[5] B. L. Yeo and B. Liu, "Rapid scene analysis on compressed video," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 5, No. 6, pp. 533-544, Dec. 1995.

[6] Y. Z. Huang, Y. Rui, T. S. Huang, and Mehrotra, "Adaptive key-frame extraction using Unsupervised Clustering," *ICIP '98*, 1998.

[7] N. Omoigui, L. He, A. Gupta, J. Grudin, and E. Sanocki, "Time-compression: System concerns, usage, and benefits," *Proc. of ACM Conf. on Computer Human Interaction*, 1999.

[8] A. Amir, D. Ponceleon, B. Blanchard, D. Petkovic, S. Srinivasan, and G. Cohen, "Using audio time scale modification for video browsing," *Proc. of the 33rd Hawaii Int. Conf. on System Sciences*, vol. 1, Jan. 2000.

[9] J. S. Boreczky and L. A. Rowe, "Comparison of video shot boundary detection techniques," *Proc. IS-T/SPIE Conf. Storage and Retrieval for Image and Video Databases IV*, I. K. Sethi and R. C., Jain, Eds., vol. 2670, pp.170-179, 1996.

[10] H. Wang and S. F. Chang, "A highly efficient system for automatic face region detection in MPEG video," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, no. 4, pp. 615-628, 1997.

[11] S.-K. Lee and M. H. Hayes, "A fast clustering algorithm for video abstraction," *ICIP 2003*, 2003.