

Automatic Video Pause Detection Filter

Xiaowen Liu, Charles B. Owen, Fillia S. Makedon
Department of Computer Science
Dartmouth College

Abstract

Increasing interest in multimedia research has been drawn upon the development of video indexing and content-based image retrieval techniques. In this report, we proposed several pause detection algorithms, which instead of searching for significant visual transitions, the algorithms detect significant pauses in video streams. A realization of the algorithms was implemented using ImageTcl toolkit developed at Dartmouth Experimental Visualization Laboratory. In addition to proposing and studying the effectiveness of the pause detection algorithms, another major goal will be to incorporate our algorithms into ImageTcl and test the stability and applicability of the ImageTcl environment. Priliminary experiments showed relatively good results of our pause detection algorithms.

1 Introduction

Recently, increasing interest in multimedia research has been drawn upon the development of video indexing and content-based image retrieval. Various schemes for automatic indexing of video databases have been proposed, together with content-based image retrieval techniques. With the increasing significance of multimedia information, the development of video and image management and retrival system has become more important. While content-based video and image indexing and retrival play an increasingly important role in multimedia information management systems, we propose an alternative development in processing video sequences. Based on the schemes of detecting video cuts, pause detection, on the other hand, is to search for significant pauses in a given video stream. Video pause detection can be applied to video compression as well as processing of video archives. We propose to design and implement schemes for automatic video pause detection. We will compare difference schemes proposed and study the effectiveness of those schemes.

Our pause detection algorithms will be implemented on the basis of ImageTcl, a highly modular rapid prototyping environment for image processing. ImageTcl was developed by Charles B. Owen at the Dartmouth Experimental Visualization Laboratory. In addition to proposing and studying the effectiveness of the pause detection algorithms, another major goal will be to incorporate our algorithms into ImageTcl and test the stability and applicability of the ImageTcl environment.

In Section 2, we describe our four pause detection algorithms in detail. An implementation of these algorithms under ImageTcl environment is shown in Section 3 and Section 4. Section 5 gives some preliminary experiment results to compare the effectiveness of the algorithms, while Section 6 gives the conclusions.

2 Pause detection algorithms

Our pause detection algorithms are built upon the cut detection algorithms. Many methods of calculating difference between video frames have been proposed. Nagasaka and Tanaka [4] studied several methods to automatically detect cuts so as to index video streams. Zhang et al. [5] proposed a partitioning system to detect video segment boundaries. A *cut* in a video stream can be defined as a sudden transition (or discontinuity) of visual properties across the transition [2]. The transition of visual properties of a cut can be significant or minimal depending on how the director controls and organizes shots. Visual properties of a shot may include factors like camera motion, object shapes, color, brightness distribution, etc. On the other hand, to determine pauses can be somewhat arbitrary depending on how the users define a pause. A *pause* can be defined as a video segment with similar visual properties. However, a general threshold of a pause can be hard to obtain. The same thing is true for the duration (or minimum length) of a pause. Such factors may vary depending on the user's demand.

2.1 Template matching with absolute difference

The first pause detection algorithm measures the difference between two frames by pointwise comparison. The method is called *template matching*. The result is obtained as a sum of pointwise difference over all pixels as defined by the following formula:

$$E(f_1, f_2) = \frac{1}{hw} \sum_{y=0}^h \sum_{x=0}^w |\text{graylevel}(f_1, x, y) - \text{graylevel}(f_2, x, y)|$$

where h and w is the height and width of the images, $\text{graylevel}(f_k, x, y)$ is the gray level of the pixel (x, y) in frame f_k . The algorithm is essentially the same as the template matching algorithm, describe in [4]. However, the pointwise difference between pixels of consecutive frames is normalized. Therefore the result of the comparison must be in range of 0 to 1 inclusive.

2.2 Template matching with relative difference

Our second pause detection algorithm extends the definition of the first algorithm, where the pointwise difference is divided by the maximum of the two comparing pixels, i.e.

$$E(f_1, f_2) = \sum_{y=0}^h \sum_{x=0}^w \frac{|\text{graylevel}(f_1, x, y) - \text{graylevel}(f_2, x, y)|}{\max\{\text{graylevel}(f_1, x, y), \text{graylevel}(f_2, x, y)\}}$$

2.3 χ^2 histogram comparison between consecutive frames

Our third pause detection algorithm uses the histogram comparison algorithm developed by Nagasaka and Tanaka [4]. The essence of their algorithm rely on the color space of the frames to identify a discontinuity. Our algorithm, which is called *consecutive histogram* χ^2 , compares the

gray-scale histograms of any two consecutive frames using the following formula:

$$E(f_1, f_2) = \sum_{i=0}^{255} \frac{(H(f_1, i) - H(f_2, i))^2}{\max\{H(f_1, i), H(f_2, i)\}}$$

where $H(f_k, i)$ denotes the i th color of frame f_k 's histogram.

As pointed out in [4], the χ^2 value more strongly reflects the degree of the difference between the two frames. The use of division normalizes the significance of the difference. There is a slight difference between our algorithm and the algorithm developed in [4]. In our algorithm, instead of using fixed denominator $H(f_2, i)$, we pick the maximum of the two frames to eliminate the preference of the second frame. Also, the common divided-by-zero error must be specially taken care of. In our experiments, we use 256 gray-scale levels. This pause detection algorithm is quite robust. It will be shown by later experiments, the algorithm is quite sensitive to scene changes, which defines significant transitions or cuts. It also shows relative good results on pause detection.

2.4 χ^2 histogram comparison with key frames

One drawback of the previous pause detection algorithm is that the algorithm cannot detect incremental motions, such like slow camera panning and slow object movement. The reason for this is such increment motions do not have significant differences between consecutive frames and usually the χ^2 test results come below reasonable threshold.

Such phenomenon can be avoided by using key frames in the detection algorithm. That is, instead of comparing histograms between consecutive frames, we pick the first (or any) frame in the pause sequence as the template. Subsequent frames will compare with the template to determine their similarity.

3 Implementation under ImageTcl environment

A realization of the pause detection algorithms was implemented using ImageTcl toolkit, a highly modular rapid prototyping environment for image processing. ImageTcl was developed by Charles B. Owen at the Dartmouth Experimental Visualization Laboratory. As specified in the ImageTcl documentation, some of the features of ImageTcl include

- Support for a wide variety of image data formats including fixed and floating point, JPEG, Motion JPEG, monochrome and color, YUV, motion flow, Sun raster-image, AVI, FFT spectral representations, image sequences as a volume.
- A powerful data-flow model with no critical paths through interpreted code. ImageTcl objects are represented as nodes in the graph. They are connected by links as data packets passing through.
- A simple to use programming interface based on Tcl, the Tk Toolkit, and the Tix library. The development of user interface becomes quite simple.
- Modular design which facilitates the easy construction of new components for rapid prototyping.
- Automatic tools for assistance in creation of new data types and commands.

Our pause detection algorithms are encapsulated in an ImageTcl object, functioning as a filter with a single sink and a single source. Currently, only type Y connection is allowed. The Y data

type contains a matrix of BYTE pixel values where each pixel value represents the luminance of a pixel. That is, each pixel is represented by a BYTE value.

Data packets going through the pause detection filter will be buffered in an internal queue if the incoming image is similar to the template image. The similarity is determined by the chosen pause detection algorithm. If the computed difference is above the given threshold, the queue will be flushed and the pause detection result is negative if the number of packets in the queue has not reached the desired duration. If the number of enqueued packets is larger than the duration, a pause is therefore detected and the queue is also flushed. In this way, the pause detection will be in real time, except that there will be some jittings observed when packets are queued up in the internal buffer¹. Detecting pauses on the fly along with playing the video is an important feature of our implementation. The new ImageTcl object is defined with following list commands:

-threshold *threshold* defines the pause threshold value. The user-defined *threshold* should be a floating point value or just the keyword DEFAULT.

-duration *duration* defines the pause duration needed to detect pauses. The *duration* must be an integer or the keyword DEFAULT.

-actonpause *action* defines the action of the filter when a pause is being detected. There are three possible ways: BLANKPAUSES will output the pauses as a blank image; CUTPAUSES will cut the pauses so that the output image stream will not contain detectable pauses any more; THUMBNAILPAUSES will only output pause thumbnails. BLANKPAUSES is the default action.

-algorithm *algor* defines the pause detection algorithm to be used. Our four pause detection algorithms are represented by four corresponding keywords: TEMPLATE_ABSDIF, TEMPLATE_RELDIF, HISTO_CONSECUTIVE, and HISTO_KEYFRAME, where HISTO_KEYFRAME is the default algorithm.

4 User interface of the pause detector

We extended the ItViewer in the ImageTcl library to accommodate the use of pause detector. Figure 1 shows an snapshot the the pause detector user interface.

There are three windows in the user interface of the pause detector: an ItViewer window, a pause-filtered image window and a graph window. The ItViewer window comes along with the old ItViewer library and shows a video display and a control panel. The pause-filtered image window shows video images that have been filtered by the pause detector object. The graph window draws the results of the calculation of the pause detection algorithm in use, while playing the video. This will help the user to pick reasonable threshold and duration for the pauses.

A separate menu in ItViewer window helps the user to select the threshold, duration as well as the pause detection algorithm and the action taken whenever a pause is being detected.

5 Experiments

We performed several experiments to measure the effectiveness of our pause detection algorithms. The outcome of the first three algorithms should be independent of the chosen threshold and duration. However, for the last algorithm, the threshold and duration play an important role in determining the key frame, which will be used as the template for comparison with subsequent frames.

¹The maximum length of the queue will be less than the give pause duration. No more buffering is necessary when the pause is being detected.



Figure 1: A snapshot of the user interface of the pause detector

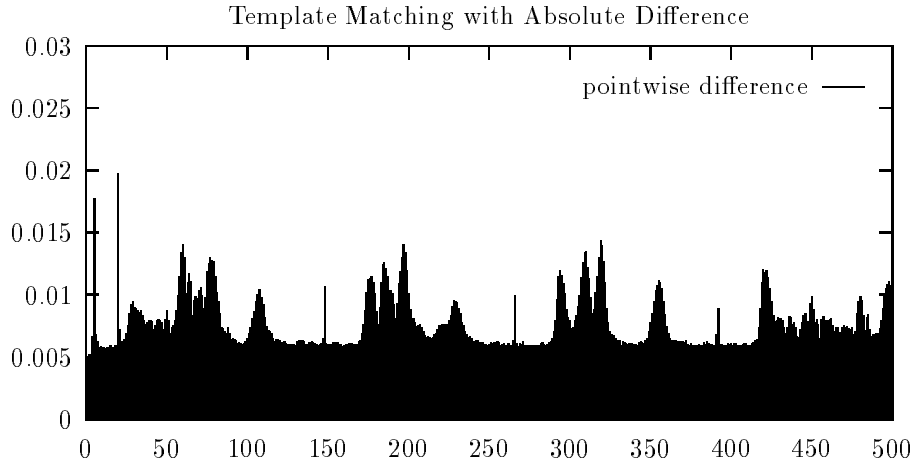


Figure 2: Output of the absolute template matching pause detection algorithm on ASL video

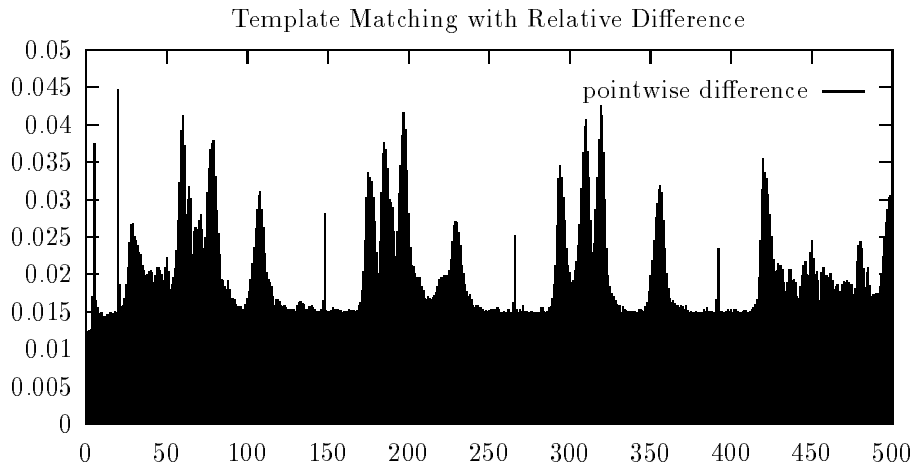


Figure 3: Output of the relative template matching pause detection algorithm on ASL video

In the first experiment, we used American Sign Language (ASL) video to study the characteristics of each of the algorithms. Figure 2 through 5 gave the results of different pause detection algorithms on the ASL video stream. Both template matching algorithms perform well in detecting the neutral positions between two sign language words, where relatively constant noise dominates the outcome of the computation. As we can see from figure 2 and 3, the pauses between sign language words are obvious. Even though template matching performs bad in detecting cuts, which are denoted by the relative low spikes in the figure, it performs quite satisfactorily in detecting pauses. Noises are normalized to a relative constant value such that the pause threshold and duration can be easily determined. Comparing these two algorithms, template matching with relative difference outperforms the other one in the sense that the values become more sensitive to object movements. As we can see from the figures, the difference between the peaks and valleys are enlarged while the neutral positions are still kept relatively constant.

Observed by Nagasaka and Tanaka [4], the χ^2 histogram comparison reduces the sensitivity to camera panning and zooming. While χ^2 algorithms obtain much clear peaks at cutting points, they also obscure the difference between pauses and movements, which are represented by object movements, camera movements like panning and zooming, etc. As shown in figure 4 and figure

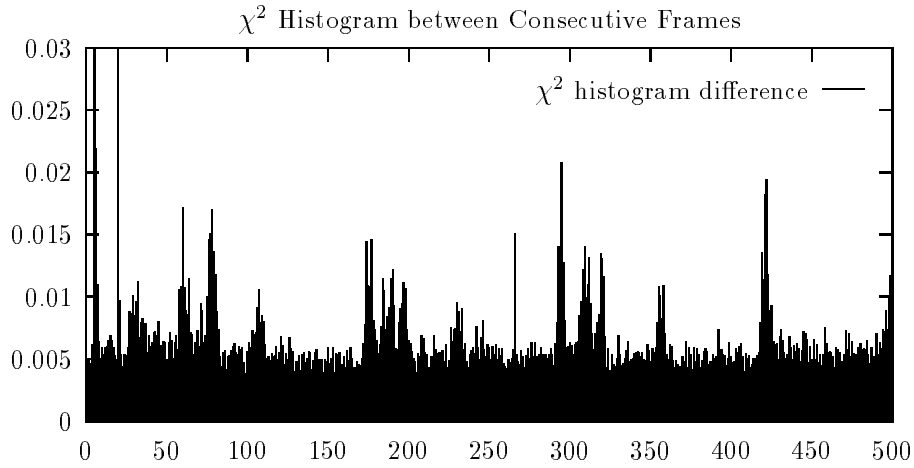


Figure 4: Output of the consecutive χ^2 histogram pause detection algorithm on ASL video

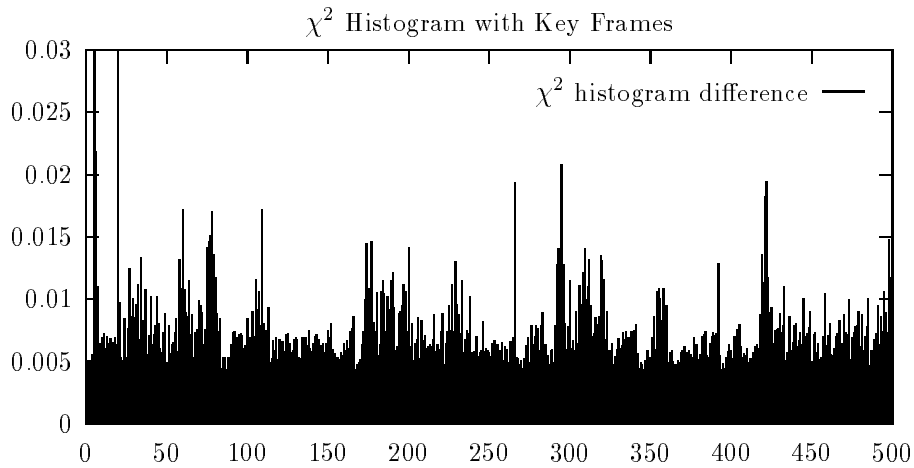


Figure 5: Output of the keyframe χ^2 histogram pause detection algorithm on ASL video, where the *threshold* is 0.08 and the *duration* is 20 frames.

5, significant visual transitions can be captured, however, they are over-emphasized at the cost of blurring insignificant visual properties which should be important in detecting pauses. Despite the above disadvantage, both χ^2 histogram algorithms perform satisfactorily on the ASL video. By choosing appropriate pause threshold and duration, the algorithms can detect all pauses in the video stream precisely.

In our second experiment, we study the advantage of using key frames as opposed to comparing consecutive frames. Two short video clips are used in this experiment: table tennis and flower garden. Both videos have no significant pauses. The first one is characterized by small-scale object movements and a camera zooming. As we mention previously, histogram difference reduces the effect of camera zooming and panning, while it will capture significant visual transitions. As shown by the figure 6, the two cuts have significantly larger values than others. The selection of the pause threshold and duration for the algorithm comparing consecutive frames can be somewhat difficult. Using our default values where we set the threshold to be 0.08 and the duration to be 20 frames, the algorithm detected three pauses. On the contrary, the algorithm comparing with selected key frames worked out to be quite robust in such circumstance. No pause is detected as expected. The reason for this is that using key frames as templates makes the algorithm more sensitive to slow

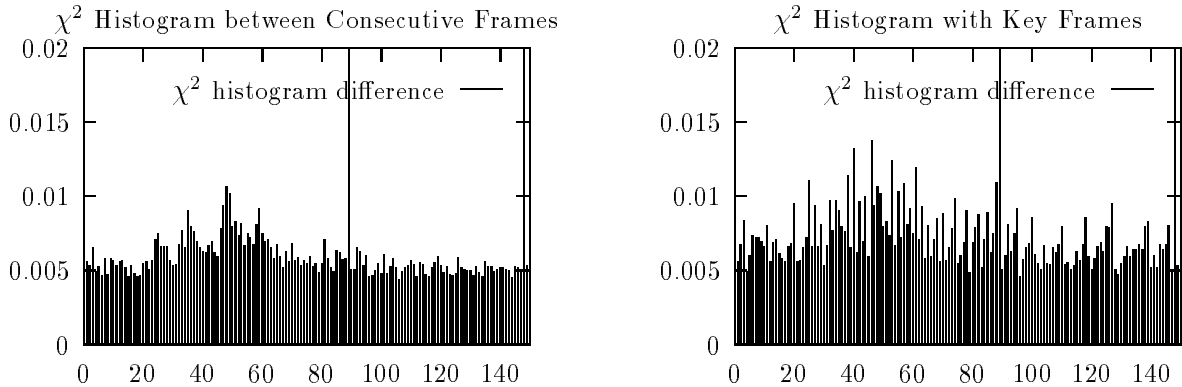


Figure 6: Comparison between the two χ^2 histogram algorithms on the table tennis video

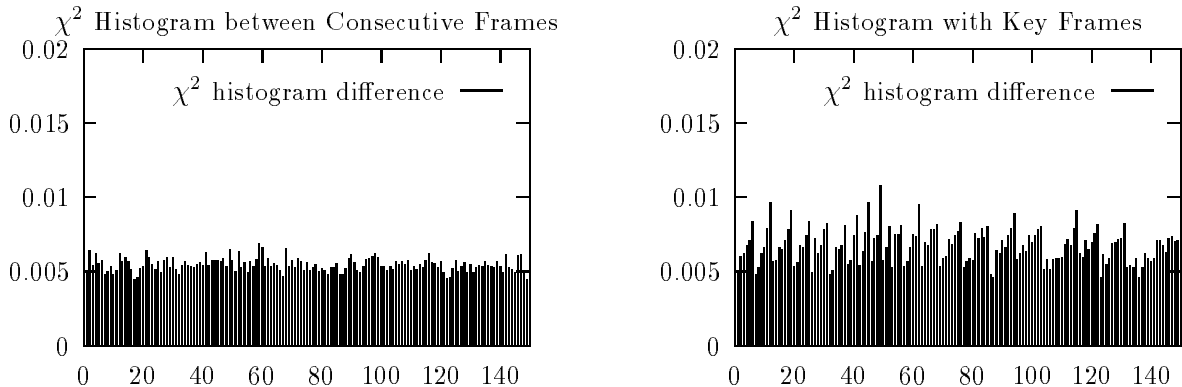


Figure 7: Comparison between the two χ^2 histogram algorithms on the flower garden video

or small-scale motions in the video since such motions will be accumulated over time and before the frames are queued up to reach the set duration, the accumulation will most likely overflow the threshold value. This explanation can be better supported by using the second video clip – flower garden. As shown in the left graph of figure 7, slow camera motions reduced the difference between consecutive frames and made almost all values below our default threshold. On the contrary, the algorithm comparing frames with the selected key frame becomes more robust against such slow motion. This is shown by the right graph of figure 7.

6 Conclusions

While template matching algorithms perform more satisfactorily, we hypothesize that the χ^2 histogram comparison algorithms should outperform their counterparts in the following way:

- χ^2 histogram comparison not only reduces the effect of camera movements but also reduces the effect of noises, which could be used against pause detection of low-quality videos.
- Slow object or camera motions can be detected by key frame comparison, which signifies the accumulation effects. On the other hand, applying the algorithms with corresponding sub-image sections should be able to capture small-scale object movements.

In a recent paper [1], Aoki et al. used both color analysis and layout analysis to calculate and compare similar shots. Although the concepts of a pause and similar shots are different, they shared the same light in attempting to detect visual insignificance between two video frames.

The future work along the track would be to study more robust pause detection algorithms with clear differentiation between noises and motions. Moreover, it will be important to devise a generic and adaptive pause detection algorithm.

References

- [1] H. Aoki, S. Shimotsuji, and O. Hori. "A shot classification method of selecting effective keyframes of video browsing." In *Proceedings of ACM Multimedia '96*. Boston, MA, November 1996.
- [2] A. Hampapur, R. Jain, and T. E. Weymouth. "Production model based digital video segmentation."
- [3] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, and C. Faloutsos. "The QBIC project: querying images by content using color, texture, and shape." In *Storage and Retrieval for Image and Video Databases*. SPIE vol. 1908, February 1993.
- [4] Akio Nagasaka and Yuzuru Tanaka. "Automatic video indexing and full-video search for object appearances." In *2nd Working Conference on Visual Database Systems*. pp. 119-133, Budapest, Hungary, IFIP WG 2.6, October 1991.
- [5] H. Zhang, A. Kankanhalli, and W. Smoliar. "Automatic partitioning of full-motion video." *Multimedia System (1993)*, Springer-Verlag, vol. 1, pp. 10-28, 1993.