Video Segmentation using Combined Cues

M.S. Toller and P.H. Lewis and M.S. Nixon

The Multimedia Research Group
Department of Electronics and Computer Science
University of Southampton, SO17 1BJ, England.

ABSTRACT

The use of video data in the multimedia environment is increasing rapidly, and so tools to handle large volumes of video data are required. One of the first steps towards creating more versatile video tools is to segment the video data, i.e. partition it into its component shots. This paper presents a novel combination of algorithms for video segmentation, utilising histogram comparison, motion vector information (focus of expansion), and edge information to detect transitions between shots. Our method can reliably detect transitions such as camera breaks, fades, dissolves and wipes, in video compressed to the MPEG-I standard.

Keywords: Video segmentation, key frame extraction

1. INTRODUCTION

With the continuing advances in multimedia technology, video is becoming commonplace in the multimedia environment. Unfortunately, tools for handling large amounts of video data are scarce, and the need for such tools is becoming increasingly important. In general, the only video handling tool available is the standard video viewer, with the typical VCR interface - i.e. fast forward/rewind, play, pause, stop, and step forward/backward. Compare this with the vast array of tools now available for image processing. We are in the process of developing MAVIS, a Multimedia Architecture for Video, Image and Sound, and have a requirement for more versatile tools for video handling. One of the first steps towards creating such tools is to segment a video into its component shots, and extract key frames for these shots. There has been much research in the area of video segmentation, and many different methods proposed.

A video sequence consists of many **frames**, these are the individual images, and are generally the smallest unit we are concerned with in a video sequence. A contiguous set of frames representing a continuous action in time and space is called a **shot**; for example, a camera following a car traveling down a road, or a fixed camera viewing a building with people walking past it. A collection of related shots is called a **scene**. i.e. news footage is usually a scene containing a shot of the anchor person followed by several shots covering the actual news footage. A **cut** or **transition** is the point at which shots change within a video sequence. Such transitions can be grouped into the following categories:

1. camera break – this is the simplest form of transition, when the current shot ends the next shot starts on the next frame - an instantaneous change;
2. fades and dissolves – these transitions take place over a number of frames, i.e. a shot is slowly faded out to black, or one shot is faded out while another is faded in;
3. spatial edit effects – these also take place over a number of frames. An example of a spatial edit effect is when the last frame of a shot slides out to the right of the screen, uncovering the next shot in the background; and
4. other approaches – there is no set number of transition effects, newer and more imaginative effects are constantly being developed.

Other author information: (Send correspondence to M.S.T.)
M.S.T.: Email: mst95r@ecs.soton.ac.uk; Telephone: +44 1703 594492; Fax: +44 1703 592865;
P.H.L.: Email: phl@ecs.soton.ac.uk; Telephone: +44 1703 593715; Fax: +44 1703 592865

Video segmentation is the process of partitioning video data into its component shots. All frames within a shot are generally very similar, or contain one or more objects that remain visible throughout the shot. By contrast, consecutive frames on either side of a camera break are usually very different. Therefore, if a quantitative measure of the qualitative difference between such frames can be obtained, this measure can be used to indicate whether a transition has occurred or not. The two most commonly used metrics for detecting transitions are based on :

- measuring differences between corresponding pixels or block regions between frames; and
- global representations such as colour or intensity histograms of the frames.

The remainder of this section describes some of the current methods and metrics for video segmentation.

2. POSSIBLE APPROACHES AND RELATED WORK

Recently Boreczky and Rowe[2] compared a number of approaches to video shot boundary detection. In this section we present our own evaluation of some of the most common techniques in use at present.

2.1. Pairwise Comparison

The simplest method for calculating a difference between consecutive frames is to compare each pixel in one frame with the corresponding pixel in the next.[19,16] If more than a certain percentage of pixels have changed between frames, then a shot transition is declared. A pixel is considered to have changed if it differs from the corresponding pixel in the previous frame by more than a set threshold t.

The major problem with this method is that it is very sensitive to camera and object motion. If the camera moves in any direction, even if only by a few pixels, then the majority of the pixels compared will be judged as different by the above metric. One solution proposed by Zhang et al.[19,20] is to smooth the image, so that each pixel has, for instance, the mean value of its 8 nearest neighbours. This also filters out some of the noise in the images, but still only compensates for minor camera or object motion.

A better method is to compare block regions, rather than individual pixels. In this case, the metrics for deciding whether a change has occurred are based on the various characteristics of the regions.

2.2. Histogram Based Methods

The most popular method for detecting transitions is by comparing the histograms of consecutive frames.[3,20,19,6,13] The theory is that two frames which have a common background and unchanging objects will show little difference in their histograms. The basic formulation for histogram comparison is as follows; the histogram (either colour or gray scale) is computed for each pair of consecutive frames, and the difference calculated as shown below:

$$D(i, i+1) = \sum_{j=0}^{N-1} |H_i(j) - H_{i+1}(j)| \tag{1}$$

where $H_i(j)$ is the j^{th} element of the histogram of the i^{th} frame, and N is the number of bins in the histogram. If colour histogram comparison is being used, j denotes a colour code derived from the three colour intensity values. Nagasaka et. al.[11] recommend using the two most significant bits from each colour channel, i.e. a six bit code, resulting in a 64 bin histogram – this is used for example by Kreyss et. al.[7] in their shot detection prior to mosaicing. A transition is declared if $D(i, i+1)$ exceeds a threshold. Determining a threshold which will work on all video data is a problem, Devadiga et al.,[3] proposed a video segmenting method called hierarchical histogram-based video indexing (HHBVI), which segments a video into several levels of granularity by using varying values for the thresholds. More sophisticated approaches to histogram comparison have been proposed by Pass, Zabih and Miller[14] and Hafner et. al.[4]

When comparing only consecutive frames, transition effects such as fades and dissolves may be missed. Zhang et. al.[20] have implemented a multi-pass, twin-comparison algorithm to overcome this problem. The twin-comparison algorithm uses two threshold values, the first, T_b is used to detect the simple camera break, and the second, lower, threshold T_s is used to detect gradual transitions. The main disadvantage of their method is that although it can detect gradual transitions, it identifies some camera operations such as panning, zooming, and tilting as false positives. To overcome this problem, the camera operations are detected through motion field analysis. Hirzalla and Karmouch[6] detect the simple camera operations by examining the hue flow pattern between consecutive frames, and comparing this to patterns formed by right-left panning, up-down tilting and in-out camera zooming operations.

Zhang's[20] multi-pass algorithm sacrifices temporal resolution in the first pass (i.e. only comparing every 10^{th} frame) to detect potential transitions. The twin comparison method is not used during this pass, instead a lower value of T_b is used. Due to this lower threshold and skip factor, both the camera breaks and the gradual transitions, as well as some other artifacts due to camera and object motion will be detected. In the second pass all computation is restricted to the vicinity of these potential boundaries. Increased resolution is used to locate all boundaries more accurately. It is claimed that this method (multi-pass and twin comparison) is faster than a single pass through a video comparing every consecutive frame.

The main sources of error encountered by Zhang et. al. during testing were:

- the motion of a very large object in a camera shot caused errors in their camera pan detection;
- combinations of pans and zooms made the detection of camera operations difficult; and
- when the camera follows a moving object that covers more than half the frame; In this case the background or foreground changes due to the camera panning may exceed T_s, and the sequence detected as a potential transition. The problem is that there is no clear mode of motion vectors and so the sequence is not correctly identified as a camera pan.

2.3. Feature Based Methods

Ramin Zabih et. al.[18] have taken a different approach to video segmentation. This one is based on detecting the appearance of intensity edges that are distant from edges in the previous frame. Their approach is based on the observation that "during a cut or a dissolve, new intensity edges appear far from the locations of old edges. Similarly, old edges disappear far from the location of new edges."

Two types of edge pixel are described:

- an **entering** edge pixel, which is one that appears far from an existing edge pixel; and
- an **exiting** edge pixel, which is one that disappears far from an existing edge pixel.

The basic measure of dissimilarity is given by

$$\varphi = max(\varphi_{in}, \varphi_{out}). \tag{2}$$

where φ_{in} denotes the fraction of edge pixels in E' (E and E' are the edge images obtained from two consecutive frames I and I') which are more than a fixed distance r from the closest edge pixel in E. φ_{in} should assume a high value during a fade-in, cut, or at the end of a dissolve. φ_{out} denotes the fraction of edge pixels in E which are more than a fixed distance r away from the closest edge pixel in E'. φ_{out} should assume a high value during a fade out, cut, or beginning of a dissolve. φ is termed the *edge change fraction*. Global motion compensation is performed to line up the edge images before computing φ_{in} and φ_{out}.

This algorithm fails under certain conditions, such as rapid changes in overall shot brightness, or shots which are very dark or very bright. i.e. on frames where the edges are not significant.

2.4. MPEG COMPRESSION AND CAMERA MOTION DETECTION

Since more video is becoming available in compressed form, there have been attempts to segment video sequences directly from the compressed data, most of which are based on MPEG compression. The advantages here are in terms of speed, as the need for decompressing each frame before any calculations are performed is removed.

Certain camera operations, such as panning and tilting, produce successive differences of the same order as gradual transitions. This causes some of the segmentation algorithms to mis-classify shots containing these operations. To overcome this problem we can try to detect the camera operations in use within a shot, which also has the advantage of supplying us with more information when it comes to indexing. Camera operations such as panning and zooming produce distinctive patterns of motion vectors, as can be seen in fig 1.

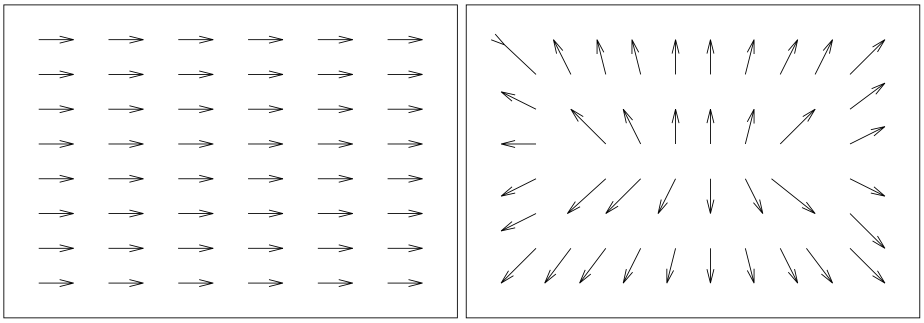

Figure 1. Motion vector patterns

The left hand figure shows the motion vector pattern associated with a camera panning operation, the majority of the motion vectors exhibit a single strong modal value which corresponds to the movement of the camera. The right hand figure shows the motion vectors typical of a camera zoom. Meng and Chang[10] use the motion vectors retrieved from the MPEG video stream to detect the global camera parameters in effect. Zhang et. al.[20] use the following formula to detect camera panning :-

$$\sum_{k}^{N} |\theta_k - \theta_m| \leq \Theta_p \tag{3}$$

where θ_k is the direction of the motion vector k, θ_m is the direction of the modal vector, and N is the total number of motion vectors in a frame. $|\theta_k - \theta_m|$ is zero when the two motion vectors are parallel. Ideally Θ_p should be zero, but a non-zero value is used to offset errors due to object motion.

From fig 1 it can be seen that the field of motion vectors associated with a zoom has a minimum value at the focus centre. Zhang et. al.[20] compare the vertical components of the motion vectors for the top and bottom rows of a frame, since during a zoom these vertical components should have opposite signs (assuming that the focus centre of the zoom lies within the boundary of the frame). This means that in every column the magnitude of the difference between these vertical components will always exceed the magnitude of either individual component. That is,

$$|v_k^{top} - v_k^{bottom}| \geq max(|v_k^{top}, v_k^{bottom}|) \tag{4}$$

The horizontal components of the motion vectors for the left-most and right-most vectors are analysed in the same way, i.e.

$$|u_k^{left} - u_k^{right}| \geq max(|u_k^{left}, u_k^{right}|) \tag{5}$$

A zoom is said to occur when both conditions are satisfied for the majority of the motion vectors. In,[20] Zhang et. al. compute the motion vectors on the frames themselves, using twenty blocks of 15 x 15 pixels. Computing more motion vectors for greater accuracy is a time consuming process. As mentioned earlier, if the video is compressed to the MPEG standard the motion vectors can be retrieved directly from the MPEG stream, and are based on 16 x 16 luminance pixel blocks . In papers[22] and,[21] Zhang et. al. use these motion vectors for their algorithms. Zhang also uses an algorithm developed for detecting global motion between frames.

3. A COMBINED APPROACH

We chose to use the MPEG-I video compression format to develop our segmentation program as this seems to be the most widely adopted standard. The MPEG video standard is portable, common, and achieves high compression, which saves heavily on disk space requirements. Another benefit of the MPEG format is that it contains motion vector information for B and P frames, which we use to aid the segmentation process.

3.1. MODIFIED HISTOGRAM SEGMENTATION

The first part of our segmentation process is an RGB histogram comparison between sub-windows from consecutive frames. The histograms are computed for the sub windows in each pair of consecutive frames, and the differences calculated as shown below:

$$D(i_w, i_w+1) = \sum_{j=0}^{N-1} |H_{i_w}(j) - H_{i_w+1}(j)| \tag{6}$$

where $H_{i_w}(j)$ is the j^{th} element from the histogram of sub window w, frame i and N is the number of bins in the histogram. This idea was proposed by Nagasaka[11] to make the comparison more robust. 16 sub windows are used, and the half with the largest differences discarded, this solves false positives due to lighting effects and small object movement covering less than half of the frame.

There is a problem with this method – when the two consecutive frames over a camera break have a similar background (of about half the frame size), but different foreground content (or vice versa), the camera break is missed as all the sub windows that differ are discarded! This is often the case when the camera switches between close up shots of peoples faces, an example of which is shown in fig 2.

Figure 2. Two frames with very similar histograms

In an attempt to solve this type of problem we selected the eight sub windows from the middle of the set, i.e. the two extremes are discarded. This new method correctly detected a number of camera breaks that suffered from the above problem. Unfortunately, many of the false detections which were corrected by using the lowest eight values are still detected as camera breaks. While over-segmentation is preferable to under-segmentation, the number of false detections introduced by this method outweighed the number of new correctly detected transitions.

3.2. GRADUAL TRANSITIONS

To detect gradual transitions, we first implemented the twin threshold algorithm developed by Zhang et. al.[20] on the histogram difference values (see section 2.2). Unfortunately, the twin threshold algorithm suffers from the fact that camera operations such as panning and zooming can introduce the same order of inter-frame difference as in gradual transitions, and so video sequences containing these camera operations (i.e. almost all at some point) are incorrectly segmented. Zhang et. al. overcame this side effect by specifically detecting the camera operations through motion vector analysis. It is possible to retrieve the motion vectors stored in MPEG-I videos, and so this seemed to be a sensible choice. Also, information retrieved about the camera operations can be used later for indexing purposes.

3.3. MOTION VECTORS

MPEG compression utilises motion compensation, and it is possible to retrieve the motion vectors from the MPEG stream. Once the motion vectors for the frames were extracted (motion vectors are only stored for P and B frames of MPEG videos), they had to be analysed to provide us with the camera operations in effect. Detecting camera panning can be achieved relatively easily by simply comparing the direction of the modal motion vector for each sub window. If the camera is indeed panning, then the modal motion vectors for each sub window should all be in the same direction, give or take a few degrees. An extra check would be to ensure that the average magnitude of the motion vectors for each sub window are similar, as they should all have moved by approximately the same amount.

This method will fail under certain circumstances :

- if there is significant object motion in the current scene, then the motion vectors for these objects will interfere with the motion vectors attributed to the camera motion; and
- if the camera is stationary, but viewing a large object that is moving, then this will be interpreted as a camera pan. For instance, if the camera is viewing a train (so that the train fills the view), and the train starts moving off to the left, then this would be interpreted as a camera pan to the right.

Detecting camera zoom operations is more difficult. Zhang et. al.[20] detected camera zoom operations by comparing the vertical components of the motion vectors for the top and bottom rows of a frame, since during a zoom these vertical components should have opposite signs (assuming that the focus center of the zoom lies within the boundary of the frame). This method was implemented, but did not perform well, especially in cases such as that depicted in figures 3, 4, and 5, in which the motion vectors for the top half of the frame do not correspond well to the actual camera operation (a zoom in). Another problem is that it only detects camera zooms when no other major camera operation is in progress; i. e. if the camera is panning as well as zooming, then the center of focus may be off center (possibly outside the frame).

We decided that a more reliable way to detect camera zooming operations would be to calculate the focus of expansion (FOE), thereby determining whether a zoom in, zoom out, or combination of zoom and panning has occurred. The FOE can be determined by finding the location where the majority of the motion vectors intersect. Akutsu et. al.[1] calculated this point using a Hough transform, but it can also be calculated by finding all the points of intersection, and then using the mean and variance of these points.

In a frame such as fig. 3, the motion vectors in the sky region are not representative of the camera operation at all (the camera is zooming in to the building in the center), this is due to the fact that MPEG compression is based on finding the best matching block, not necessarily the true match. For this very reason, Ramin et. al.[18] chose to not use the motion vectors from MPEG streams, coupled with the fact that only P and B frames contain motion vectors. In a large homogeneous region, such as the sky in fig. 3, the vectors are often not representative of the true motion, whereas when there is detail in the frame the motion vectors tend to be accurate. To overcome these problems, we decided to discard any motion vectors that may be false. Zhang et. al.[22] used the residual difference as an indication of how *true* a motion vector was, but this method will not hold true for homogeneous regions - the match is a good one, just not indicative of the true motion. Instead, we select whether or not a vector is to be used by looking at the edge map of the frame. Generally, the motion vectors around areas which have more detail are more indicative of the actual motion. We first edge detect the frame using a thresholded Laplacian edge detector. If more than a set percentage of the pixels in the 16x16 block contain edge points, the motion vector is used, otherwise it is discarded.

By experimentation, 25% was found be a reliable value for this threshold. The results are good, as can be seen from figures 4, showing all the motion vectors, and 5, showing only the selected motion vectors (fig 3 shows the original frame).

Figure 3. Original frame

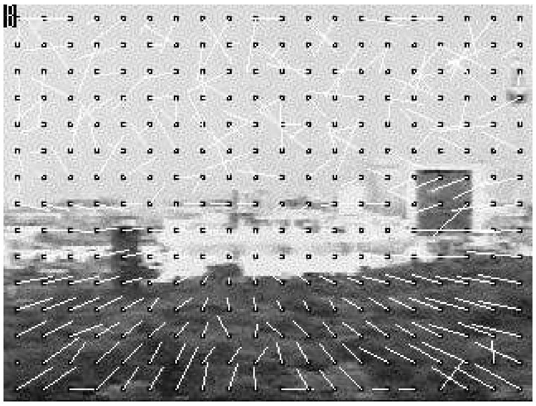

Figure 4. All the motion vectors

The following algorithm is used to locate the FOE:

1. calculate the mean and variance of the x and y co-ordinates of all the all the intersections;
2. sort the set of points in descending order of distance from the mean. If the variance in either the x or the y set is greater than a threshold T, remove the top X percent of points from the list, and return to step 1;
3. if both variance values are below T, then we check how many points of intersection remain (I). If there are more than a set number of points remaining then the FOE is located at the position $(mean(x\ set), mean(y\ set))$.

A value of 200 for T was found to give good accuracy without restricting the set of intersection points too much - too small a value for T results in the FOE being detected for only a very well defined set of motion vectors, the largest distance from the mean with a value of 200 is around 30 pixels. In step 2 we remove 25% of the points from the set at a time, this value only has an effect on the speed at which we locate threshold T, as if we drop below I by removing too many points, we go back a step and remove half as many points. In step 3 we use a value of 400

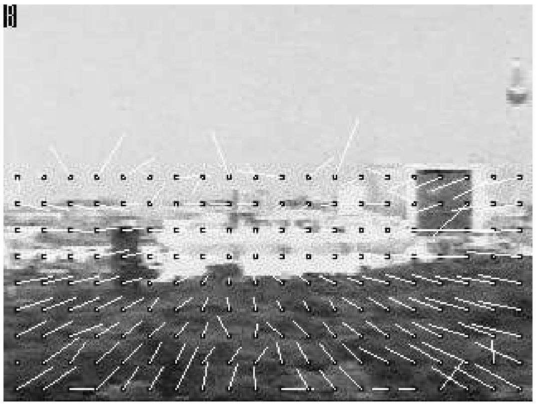

Figure 5. Selected motion vectors

for I, too low a value for I results in false detections of a FOE, where just by coincidence several motion vectors intersect. Too high a value for I will have a similar effect as too high a value for T, i. e. only a well defined set of motion vectors will result in the FOE being detected.

3.4. EDGE INFORMATION

As we are edge detecting the frames in order to locate the best motion vectors, it is efficient to use the edge information to support the shot change detection by the histogram comparison. During a transition between shots, the edge map will change, often by a large amount. The simplest method would be to count the number of edge points in two consecutive frames and calculate the difference. Unfortunately, this difference value does not always reflect a transition, as although the edge maps may be very different, the *number* of edge points may be similar.

For example, fig 6 shows the edge maps of two frames either side of a camera break. The total number of edge points in these images differ only by a small amount.

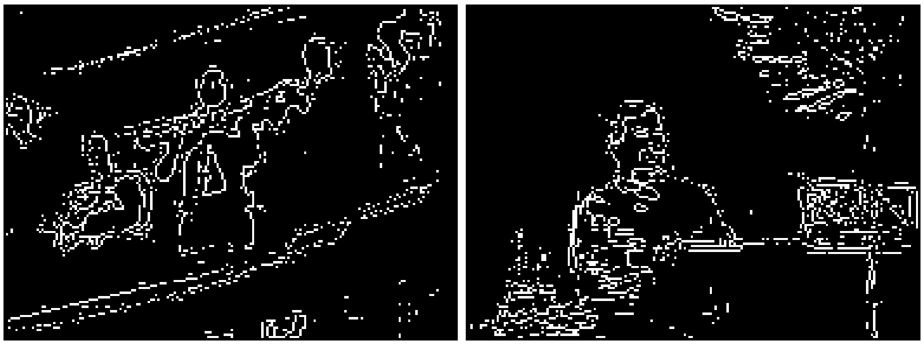

Figure 6. Difference in edge map over a camera break

To overcome this problem, we split the frame into 9 sub windows, and compare the number of edge points in corresponding sub windows. The resulting differences give a more reliable indication that a camera break has occurred. For instance, figure 7 shows the edge differences between a sequence of 800 frames from one of our test mpegs; the figure on the left shows the differences between sub windows, the figure on the right the differences between global edge values. The edge differences are calculated as follows :

$$EdgeDiff = |E_f - E_{f+1}| \tag{7}$$

where

$$E_f = 100.0 * \text{total no. of edge points}/\text{Area} \tag{8}$$

E_f is the percentage of edge points in a given area. In the sub windows method, the difference used is the sum of these differences.

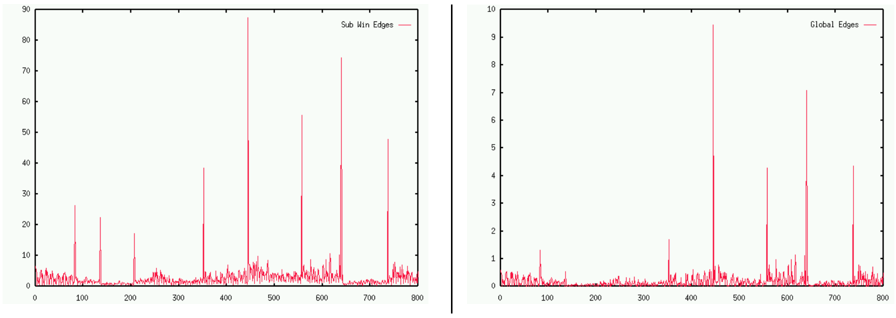

Figure 7. Difference in edge maps over 800 frames, difference between sub windows is shown on the left, global differences on the right.

As can be seen from these figures, the difference between the edges in sub windows provides a clearer indication of camera breaks (all the spikes above 10 in fig 7 are camera breaks). The first four camera breaks would have been missed by the global edge difference method, and although the four camera breaks after frame 400 would have been detected, the global edge differences are less pronounced.

4. COMBINING THE RESULTS

Since we now have several results that can be used to detect transitions and camera operations, a method of combining these results to achieve the best possible segmentation was needed. To make our decision we have the following information :

- Histogram similarity measure, H_{sim}. We use two thresholds for the histogram comparison, H_{low} and H_{high};
- Number of sub windows which reported edge change, E_{win};
- Total amount of edge change over all the sub window, E_{tot}; and
- The current state of the video.

The current state of the video can be *normal video*, *camera operation*, *gradual transition*, *camera break*, or *possible camera break*. The algorithm which integrates this information works in the following way:

- if E_{win} is zero, then we return *normal video*;
- if the $H_{sim} > H_{high}$ then:
  - if E_{win} is high we return a *camera break*;
  - if E_{win} is low and E_{tot} is low, then we return *normal video*;
  - if E_{win} is low, but E_{tot} is high, then we return a *possible camera break*;
- if $H_{low} < H_{Sim} < H_{high}$ then
  - if E_{win} is high we return a *camera break*;

– if E_{win} is low, then depending on E_{tot} and whether we are currently in a gradual transition or not, we return either *normal video*, *gradual transition*, or *possible camera break*.

- if $H_{Sim} < H_{low}$, but E_{win} is high, then we may return either *normal video*, *gradual transition*, or *possible camera break* depending on the values of E_{tot} and the current state of the video.

If the state is *camera operation* then gradual transitions effects are ignored. If the state is *gradual transition* and then reverts to *normal video*, we check the frame marked at the beginning of the state with the current frame, using the above algorithm - if this returns a *camera break* then a gradual transition has occurred. If the state was *possible camera break* then the same procedure occurs, but if less than a set number of frames have passed a camera break is declared at the first marked frame.

4.1. OUTPUT INFORMATION

The video segmenter records both the start and end frames of each shot – these frames have their frame number printed in the top left corner of the image. All the frames output are then compressed via the Berkeley MPEG Encoder to produce an MPEG file which can be used to browse through the shots detected. As an aid to debugging, it is also possible to draw the motion vectors for each frame onto the images before being recorded on disk, this enables us to see clearly what motion vectors are being used in the segmenter. A text file is also output. This contains the frame numbers of the start and end frame of each shot, and the camera operations in effect during the shots.

5. RESULTS

A preliminary test of the video segmenter was made on MPEG video clips containing a total of 236 camera breaks (as interpreted by a human viewer). The results are shown in table 8.

Name	Camera Breaks		Gradual Transitions		False Detections
	Detected	Missed	Detected	Missed	
Cliffhanger	155	9	N/A	N/A	7
Reichmix	14	0	1	2	4
Sky	31	0	N/A	N/A	2
Others	27	0	11	0	1

Figure 8. Results of video segmentation.

Defining these results in terms of recall and precision :

$$\text{Recall} = \frac{\text{Correct}}{\text{Correct + Missed}} \qquad \text{Precision} = \frac{\text{Correct}}{\text{Correct + False}} \tag{9}$$

we have *recall* $= 0.96$ and *precision* $= 0.94$, compared with a recall and precision of 0.92 and 0.96 respectively for H. Zhang et. al.,[19] and 0.90 and 0.92 for Nakajima[12] (using data from their results). The 7 false detections in the Cliffhanger video clip are due to extensive lighting changes, an example of which is shown in fig 9, which causes our algorithm to report a camera break.

The missed camera breaks were mostly in shots with fast action, with a large amount of blurring in the frames - this caused little or no edge information to be detected, and so no camera break was verified. Three of the four false detections in the Reichmix clip were due to camera motion. Unfortunately, due to the poor quality in this portion

Figure 9. Extensive lighting effect causing false camera break detection

of the video, the motion vectors were not sufficiently accurate for the camera detection algorithm to pick up this motion. Both missed dissolve effects were in very dark shots, and so very little histogram difference was detected. Sometimes, fast gradual transitions (those occurring over only a few frames) were detected as camera breaks.

The above results were collected using set threshold values. It is possible that if an adaptive threshold selection were used, such as that described by Yeo and Liu,[17] then the results may be further improved. As with spatial segmentation, it is almost impossible to automatically achieve 100% segmentation, and trade-offs have to be made, Boreczky and Rowe[2] state "In any application a trade-off must be made between recall and precision. It may or may not be acceptable to retrieve one extra shot boundary that would otherwise be missed at the expense of retrieving 100 non-boundaries incorrectly." Jain et. al.[5] state an 88% detection rate, split between false detections and missed shots, Zhang et. al.[20] achieve approximately 87% detection rate, but with more of the errors being missed transitions rather than false detections.

6. CONCLUSIONS

The video system developed so far provides a valuable test bed for the future work on video feature extraction and integration with an open hypermedia system. With no user interaction, about 95% of all transitions are detected. The driving force behind this work is that we shall eventually provide effective facilities for handling video in the MAVIS[9,8] environment, the main functions being to provide enhanced facilities for authoring and following generic links from video and content based video retrieval. Such facilities will benefit from rapid browsing facilities based on shots and scenes. We believe that it will be useful for the video interface to have access to image processing functions, such as image segmentation. Work is in progress on the integration of the video viewer with GIP,[15] to form the basis of a MAVIS/Microcosm aware video utility. This will allow us to perform such operations as freezing a frame, delineating an object within the frame, and using subsequent frames to extract the motion vectors for the object.

Methods to index and match the shots extracted with the video segmentation process and also to extract and match spatio-temporal events within those shots are being developed and will be the main processes behind the retrieval and navigation processes.

ACKNOWLEDGEMENTS

The first author wishes to thank the University of Southampton for its support, and the second author wishes to thank the EPSRC for the support of grant number GR/L03446.

REFERENCES

1. A. Akutsu, Y. Tonomura, H. Hashimoto, and Y. Ohba. Video indexing using motion vectors. In *SPIE Visual Communications and Image Processing*, volume 1818, pages 1522–1530, 1992.
2. J.S. Boreczky and L.A. Rowe. Comparison of video shot boundary detection techniques. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases IV*, pages 170–179, 1-2 February 1996.

3. S. Devadiga, D.A. Kosiba, U. Gargi, S. Oswald, and R. Katuri. A semiautomatic video database. In Waye Niblack and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases III*, pages 262–267, 9-10 February 1995.
4. J. Hafner, H. Sawhney, W. Flickner, and W. Niblack. Efficient colour histogram indexing for quadratic form distance functions. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 729–736, july 1995.
5. A. Hampapur, R. Jain, and T.E. Weymouth. Production model based digital video segmentation. In *Multimedia Tools and Applications*, volume 1, pages 9–46. Kluwer Academic Publishers, Boston, 1995.
6. N. Hirzalla and A. Karmouch. Automatic cut and camera operation detection for video. Technical report, Multimedia Information Research Laboratory, University of Ottawa, 1995.
7. J. Kreyß, M. Roper, P. Alshuth, Th. Hermes, and O. Herzog. Video retrieval by still image analysis with imageminer^{tm}. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases V*, pages 36–44, 13-14 February 1997.
8. P. H. Lewis, H. C. Davis, M. R. Dobie, and W. Hall. Towards multimedia thesaurus support for media-based navigation. In *Proceedings of the First International Workshop on Image Databases and Multi-Media Search*, pages 83–90. Amsterdam University Press, August 1996.
9. P. H. Lewis, H. C. Davis, S. R. Griffiths, W. Hall, and R. J. Wilkins. Media-based navigation with generic links. In *Proceedings of the Seventh ACM Conference on Hypertext*, pages 215–223. ACM, March 1996.
10. J. Meng and S. Chang. Tools for compressed-domain video indexing and editing. In Ishwar K. Sethi and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases IV*, pages 180–191, 1-2 February 1996.
11. A. Nagasaka and Y. Tanaka. Automatic video indexing and full-video search for object appearances. In E. Knuth and L.M. Wegner, editors, *Visual Database Systems II*, pages 113–127, 1992.
12. Y. Nakajima. A video browsing using fast scene cut detection for an efficient networked video database access. In *IEICE Transactions on Information & Systems*, pages 1355–1364, December 1994.
13. K. Otsuji, Y. Tonomura, and Y. Ohba. Video browsing using brightness data. In *SPIE Visual Communications and Image Processing*, volume 1818, pages 980–989, 1991.
14. G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In *ACM Multimedia 96*, pages 65–73. ACM, 1996.
15. S.T. Perry and P. H. Lewis. A novel image viewer providing fast object delineation for content based retrieval and navigation. In *Storage and Retrieval for Image and Video Databases VI (to be published)*. SPIE, Jan 1997.
16. W. Xiong, J. Chung-Mong Lee, and Man-Ching Ip. Net comparison: a fast and effective method for classifying image sequences. In Waye Niblack and Ramesh C. Jain, editors, *Storage and Retrieval for Image and Video Databases III*, pages 318–326, 9-10 February 1995.
17. Boon-Lock Yeo and Bede Liu. A unified approach to temporal segmentation of motion jpeg and mpeg compressed video. In *International Conference on Multimedia Computing and Systems*, pages 81–83. IEEE, Los Alamitos, CA, USA, 1995.
18. Ramin Zabih, Justin Miller, and Kevin Mai. A feature-based algorithm for detecting and classifying scene breaks. In *3rd International Multimedia Conference and Exhibition, Multimedia '95*, pages 189–200. ACM, New York, NY, USA, 5-9 November 1995.
19. H. Zhang, S.W. Smoliar, and Jian Hua Wu. Content-based video browsing tools. In *Multimedia Computing and Networking*, pages 389–398. Proc. SPIE - The International Society for Optical Engineering, 6-8 February 1995.
20. H.J. Zhang, A. Kankanhalli, and S.W. Smoliar. Automatic partitioning of full-motion video. In *Multimedia Systems*, pages 10–28, 1993.
21. H.J. Zhang, C.Y. Low, and S.W. Smoliar. Video parsing and browsing using compressed data. In *International Conference on Image & Video Processing II*, pages 142–149. Proc. SPIE - The International Society for Optical Engineering, 1994.
22. H.J. Zhang, C.Y. Low, and S.W. Smoliar. Video parsing and browsing using compressed data. In *Multimedia Tools and Applications*, volume 1, pages 89–111. Kluwer Academic Publishers, Boston, 1995.