Special effect edit detection using *VideoTrails*: a comparison with existing techniques

Vikrant Kobla, Daniel DeMenthon, and David Doermann

Laboratory for Language and Media Processing
University of Maryland
College Park, MD 20742 - 3275.

ABSTRACT

Video Segmentation plays an integral role in many multimedia applications such as digital libraries, content management systems, and various other video browsing, indexing, and retrieval systems. Many algorithms for segmentation of video have appeared in the past few years. Most of these algorithms perform well on cuts, but yield poor performance on gradual transitions or special effect edits. A complete video segmentation system must achieve good performance on special effect edit detection also. In this paper, we discuss the performance of our *VideoTrails* based algorithms with other existing special effect edit detection algorithms in literature. Results from experiments testing for the ability to detect edits from TV programs ranging from commercials to news magazine programs, and also diverse special effect edits introduced by us have been shown.

Keywords: Special effect edit detection, MPEG, *VideoTrails*, FastMap, Compressed domain analysis, Performance evaluation

1. INTRODUCTION

With the increased role of computer technology in film and video production, many types of complex special effect edits have begun to appear in video clips. Telecasts of sporting events often contain fancy edits between live footage and instant-replay, or use them as a transition for displaying game statistics. Most documentaries or biographical stories use dissolves as the primary form of edit. The editor's job is to select the best type of edit that suits the appropriate "mood" the video clip is supposed to convey.

The simplest edit between two shots is a cut or a break in which the transition is immediate between two frames. Let us define any other edit that occurs gradually over multiple frames as a special effect edit. Hence, these edits are sometimes known as gradual transitions. Examples of these include fades, dissolves, and wipes. In a fade, the luminance gradually decreases to, or increases from, zero. In a dissolve, two shots, one increasing in intensity, and the other decreasing in intensity, are mixed. Wipes are generated by translating a line across the frame in some direction, where the content on the two sides of the line belong to the two shots separated by the edit. Many other special effect edits exist that may not be simple linear transformations like the ones described above.

Recently, numerous articles have appeared in the literature detailing techniques for detecting cuts.[1–10] Many of these algorithms work directly in the compressed domain of MPEG with minimal decoding. Though some of these articles tackle the problem of detecting the complex special effect edits, very few of them do so in the MPEG compressed domain. The problem is difficult because no obvious features exist in the MPEG compressed domain that suggest that a gradual transition is taking place without looking over large numbers of consecutive frames. Even so, other types of normal scene action may begin to affect decisions. Most of the shot change detection algorithms are applied to small localized input data and hence cannot be extended to detect gradual special effect edits.

In this paper, we briefly describe our algorithm for detecting these special effect edits using the *VideoTrails* representation of video.[11] We compare in detail the performance of our algorithm with those of existing special effect edits detection techniques. Our algorithm based on *VideoTrails* uses features extracted from the reduced DC images of video derived from the compressed domain of MPEG-1 video. Our implementations of the existing algorithms were

Authors' emails: V. Kobla: kobla@cfar.umd.edu; D. DeMenthon: daniel@cfar.umd.edu; D. Doermann: doermann@cfar.umd.edu

also designed to handle the same reduced DC images as input video data. Techniques that work on the compressed domain are very fast compared to algorithms that require complete decompression to full-image sequences.

In Section 2 we discuss some related work, and in Section 3 we present our *VideoTrails* based special effect edit detection algorithm. Section 4 lists and discusses the existing special effect detection algorithm that we compare our technique with. Section 5 explains the evaluation procedure used in this paper. The experiments are detailed in Section 6, and its results are presented and discussed in Section 7. Finally, the conclusions are given in Section 8.

2. RELATED WORK

The problem of detecting changes in shots or scenes has been of interest to researchers over the past few years. Detecting cuts or abrupt changes has been comparatively an easier problem to tackle than detecting special effect edits.

Our earlier work on detection of cuts in MPEG video uses only the macroblock (MB) type information. Special effect edits cannot, however, be detected with the use of MB types alone, since they occur gradually over a series of frames and the MBs tend to remain bidirectionally predicted or intra-coded over this span. It becomes necessary therefore to use the DCT values of frames to perform this analysis. Reduced DC images are fairly easy and not too expensive to extract and have proven to be sufficient for many image-processing tasks. If the MPEG file contains P or B frames, the actual DC coefficients of their MBs need to be estimated.[12]

Some of the existing techniques utilize these reduced DC images for special effect detection, and most of them can be extended to handle the DC images, since the DC images basically are the pixel images of the individual frames at much lower scale. A paper by Yeo and Liu[2] suggests a method in which every i^{th} frame is compared to the $(i+k)^{\text{th}}$ frame. The separation parameter k should be larger than the number of frames in the edit. If that is the case, by using the sum of the absolute difference of the corresponding DC coefficients as the comparison metric, any ramp input should yield a symmetric plateau output with sloping sides.

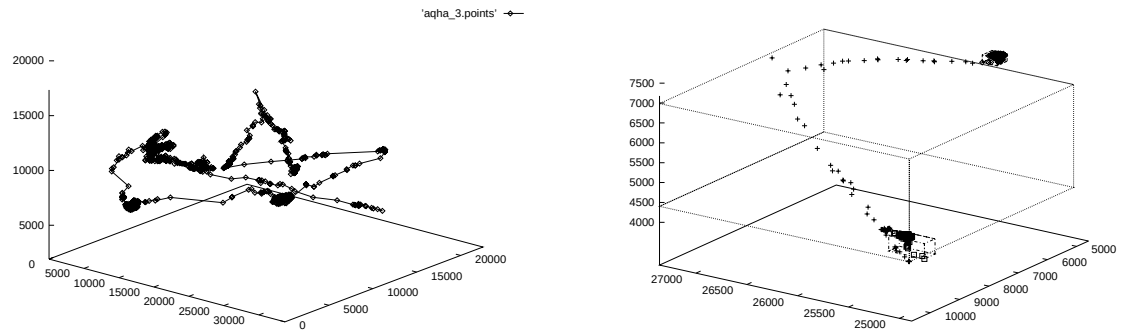
Another technique suggested by Meng et al.[3] involves using the intensity variance to detect dissolves. They measure frame variance by using the DC coefficients of the I and P frames, and observed that during a dissolve the variance curve shows a parabolic shape. Song et al.[13] suggest a technique using the chromatic video edit model. There have also been research done in this area that work on uncompressed pixel data.[6,7,10] The algorithms by Zhang et al.[10] employs a twin comparison technique using dual thresholds for gradual transition detection. Most of these earlier works on special effect detection perform well on linear transitions, but not on more general transitions. In our technique using *VideoTrails*, we look for shot consistency to detect transitions.

Gargi et al.[14] present some results of their evaluation of algorithms based on cut and gradual transition detection. Their implementations of the competing algorithms were as close as possible to the actual algorithms discussed in the original papers. Their evaluations of the various algorithms show poor performance on gradual transition detection. In our implementation, we made limited improvements to the original techniques discussed in literature.

3. *VideoTrails* ALGORITHM

Our technique for detecting the gradual transitions involves mapping each frame to a point in a low-dimensional space using a technique developed by Faloutsos and Lin[15] called *FastMap*, clustering the frames, and observing the transitions between the resulting clusters. The mapping of the features of each frame to a point in space creates a trail of points in that space called a *VideoTrail* (Figure 1). If the output space is at most 3 dimensions, we have the ability to visualize the trail. For dimensions greater than 3, we can still analyze the trails using techniques that can be extended from 3 dimensions. The features that we use for obtaining the *FastMap* points are either the YUV or the RGB values of the DC coefficients.

The advantage of our technique is that transitions are detected irrespective of whether they are linear or not. Apart from the common special effect edits such as dissolves, fades, and wipes, many kinds of complex edits are also detected. Gradual transitions in *FastMap* space appear as sparsely threaded trails. It might also be possible to distinguish between various types of special effect edits by analyzing the trajectory of these trails in geometric space. The more common linear edits such as the dissolves and fades mentioned above yield linear transitions in *FastMap* space whereas the more complex and uncommon non-linear edits generally yield non-linear trajectories in space. In



(a) (b)

Figure 1. *Video Trails*: (a) Example of a *Video Trail* containing many transitions, (b) Part of a *Video Trail* showing two stationary trails at the ends of a sparse transitional trail. Each trail is enclosed within its bounding box.

one example, where a small area of a map was high-lighted and then expanded to full-scale, the effect resembled a zoom-in operation of that small area of the map. The resulting trail had a fairly parabolic trajectory in 3-D space (Figure 1 (b)).

The difficulty with this approach is that, sometimes, activity in the clip arising from camera or large object motion also yields trails that are somewhat similar to trails resulting from gradual edits. Changes that are due to slight movements are often not detected as trails, but when fast camera motion occurs over vastly varying scene content, it becomes indistinguishable from special effect transitions. This ambiguity can be resolved by extracting the global camera motion directly from the temporal features present in the MPEG compression stream, and tagging these transitions as motion transitions.[16] We call this the "Motion Transition Removal Test". Thus, the transitions not tagged as motion transitions are detected as special effect edits.

To detect special effect edits in a video clip, we generate the *Video Trail* for the clip and split it into its constituent trails. We run a classification algorithm[11] on each trail to obtain a list of stationary and transitional trails (Figure 1 (b)). Then, by comparing the ranges of each transitional trail with the motion ranges detected by the global motion detector, we determine whether a transitional trail was due to motion or due to a special effect edit. See references for full details.[11,17]

For our evaluation, we use three versions of the *Video Trail* technique for detecting special effect edits. The only difference between these three versions is the type of features and distance function used to generate the points with *FastMap*. They are detailed below:

VT-Pix: The *Video Trails* algorithm uses feature vectors formed by all the color pixel values (YUV or RGB) of the reduced DC images. Let $V_p^i = (x_1^i, \ldots, x_N^i)$ and $V_p^j = (x_1^j, \ldots, x_N^j)$ be two Feature vectors of length N. Their pixel-wise distance $\mathcal{D}_p(V_p^i, V_p^j)$ is an Euclidean distance defined by:

$$\mathcal{D}_p(V_p^i, V_p^j) = \sqrt{\sum_{k=1}^{N} (x_k^i - x_k^j)^2} \tag{1}$$

VT-Hist: This algorithm's features are the color histogram values (YUV or RGB). We used 16 histogram bins over the range [0:255] (yielding a bin-size of 16) for each of the 3 components. This gives us $M = 16 \times 3 = 48$ values for each vector. Let $V_h^i = (y_1^i, \ldots, y_M^i)$ and $V_h^j = (y_1^j, \ldots, y_M^j)$ be two feature vectors of length M. Their histogram bin-wise distance $\mathcal{D}_h(V_h^i, V_h^j)$ is defined by:

$$\mathcal{D}_h(V_h^i, V_h^j) = \sqrt{\sum_{k=1}^{M}(y_k^i - y_k^j)^2} \tag{2}$$

VT-PixHist: This algorithm uses a weighted sum of the above two distances as the final distance. Given the vectors V_p^i, V_p^j, V_h^i, and V_h^j, we define the weighted distance $\mathcal{D}_{ph}(V_p^i, V_p^j, V_h^i, V_h^j)$ as:

$$\mathcal{D}_{ph}(V_p^i, V_p^j, V_h^i, V_h^j) = \mathcal{D}_p(V_p^i, V_p^j) + w * \mathcal{D}_h(V_h^i, V_h^j) \tag{3}$$

where w is the weighting factor between the two distances. The weighting factor is calculated as follows. Since the two pixel-wise and histogram-wise distance functions are based on different features and different lengths of feature vectors, their ranges are quite different. We wish to calculate w such that the two individual distances are more or less of the same order of magnitude. We want $w = \max(\mathcal{D}_p)/\max(\mathcal{D}_h)$.

Take the RGB case for example: $\max(\mathcal{D}_p)$ occurs when one feature vector has all values equal to 255, and the other has all values equal to 0.

$$\max(\mathcal{D}_p) = \sqrt{\sum_{k=1}^{N}(255 - 0)^2} = 255 * \sqrt{N}$$

$\max(\mathcal{D}_h)$ occurs when one histogram distribution has all pixels in one bin, and the other histogram distribution has all pixels in some other bin. If there are n pixels in each image and c color components (e.g., $c = 3$ for the RGB case), then $N = c \times n$.

$$\max(\mathcal{D}_h) = \sqrt{c \times [(n-0)^2 + (0-n)^2]} = \sqrt{2cn^2} = n\sqrt{2c}$$

Hence,

$$w = \frac{255 * \sqrt{N}}{n\sqrt{2c}} = \frac{255 * \sqrt{cn}}{n\sqrt{2c}} = \frac{255}{\sqrt{2n}}$$

If the reduced DC images are of size 44×30, then $w \approx 5$.

4. EXISTING ALGORITHMS

We have implemented four existing algorithms for comparing with the *VideoTrails* technique. These four algorithms are:

1. The plateau detection algorithm of Yeo and Liu.[2]
2. The variance curve approach of Meng et al..[3]
3. The twin comparison approach of Zhang et al..[10]
4. The chromatic edit model approach of Song et al..[13]

Each algorithm has been implemented as described in the respective papers, except for a few improvements. In this section, we briefly describe our implementations of each algorithm, and cite any improvements. The reader is referred to the cited papers for further details. All the algorithms have been implemented to handle reduced DC image sequences of all I, P and B frames. The reduced DC image sequences are obtained using the algorithm by Yeo and Liu.[12] After the transitions have been identified by the respective algorithms, they are passed to the Motion Transition Removal Test where any transitions that correspond to motion transitions are removed. Only the twin comparison approach mentioned the Motion Transition Removal Test in its original paper.

The Plateau Algorithm (Plateau)[2]: The algorithm compares every i^{th} frame to the $(i+k)^{\text{th}}$ frame using the sum of absolute difference between the two frames as difference metric. The plot of these deferred differences yields a plateau if k is greater than the length of the gradual transition. If the transition occurs from frame a_i to frame a_j, then a plateau should appear from $a_j - k$ to a_i (k should be at least slightly greater than $a_j - a_i$ for this plateau to be present) in the deferred difference plot (Figure 2). The paper suggests two criteria to detect the points on the difference plot. The first criterion tests to see that the points within a $2s+1$ window around a point on the plateau are within the neighborhood. In our implementation, we allowed a tolerance of up to 10% of its value for the test of closeness. We used the suggested value of 5 for s. The second criterion tests to see if the plateau stands out by the following:

$$D_i \geq l \times D_{i-k-1} \quad \text{or} \quad D_i \geq l \times D_{i+k+1}$$

where D_i is the difference between frame i and frame $i+k$, and l is a threshold parameter.

The original paper suggested using $k/2$ instead of k in the above tests, but from our experiments, we observed that for large transitions, performing the test with k performed better than with $k/2$. We used the suggested value of 3 for l.

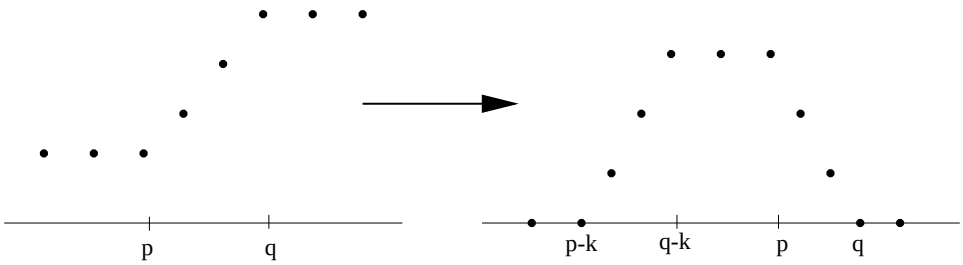


Figure 2. The Plateau method of detecting gradual transitions.

The Variance Curve Algorithm (VarCurve)[3]: Figure 3 depicts the type of variance plot that is used to detect the presence of dissolves in a video clip. The algorithm calculates the variance of the pixels (here, the DC coefficients) for each frame. These variances are examined for downward parabolic curves that suggest dissolves. In our implementation, we first smooth the curve using a Gaussian smoothing operator with $\sigma = 1.0$. We identify successive peaks and the valleys between them. For each set of two peaks and a valley, we test for two criteria to be satisfied. First, the width of the curve should be wide enough. The number of frames between peak 1 and the valley, and the number of frames between the valley and peak 2 should each be greater that T_f frames. We use $T_f = 4$. The second criterion tests to see if the valley is deep enough. For this test, we had two conditions. If either condition applies, then, the test for this criterion was passed. For the first condition, we tested to see if the average variance difference ($\Delta\sigma^2$) between the two peaks was greater than the variance on either end of the peaks by a threshold $T_1 = 3$. We used a local window of 10 frames before peak 1 and 10 frames after peak 2. The second condition requires that either of the two drops (sloping sides) from the peaks to the valley be greater than T_2 times the value of the midpoint of the drop. This condition was added because in certain cases, the variances before peak 1 and after peak 2 were large due to activity in certain special effects or motion, which did not allow the variance within the peaks to be large enough to exceed the threshold. This second condition allows a more "global" test for detection.

The Twin Comparison Algorithm (TwinComp)[10]: The idea in this algorithm is to use two thresholds, one a lower threshold (T_s) to identify potential starts of transitions, and the second, a larger threshold (T_b) that is applied to the ends of the transitions. During a dissolve, it is observed that successive frame histogram differences are slightly greater than the normal average within-shot histogram difference, while none is large enough to pass a much larger threshold for shot change detection. However, the accumulated difference over the span of frames with successive histogram differences greater than the lower threshold, is greater than the larger shot change threshold.

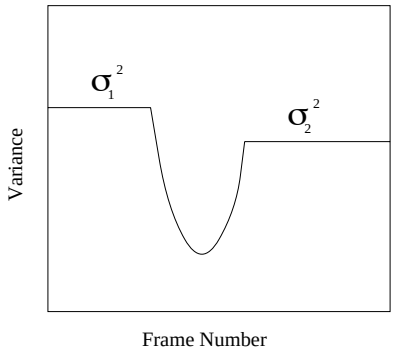


Figure 3. The Variance Curve method of detecting gradual transitions.

As suggested, we use $T_b = \mu + \alpha\sigma$, where μ and σ are the mean and variance of the frame-to-frame differences. The α is a constant and we use the suggested value of 5. For calculating T_s, we form the histogram of the difference values of the video clip and identify the peak. This peak should correspond to a low index value in the histogram. We assign T_s, the index value that corresponds to half of the peak value on the right slope of the peak. Also as suggested, we ensure that T_s is greater that the mean, by assigning a value marginally larger than the mean, if the T_s calculated from the slope is less than the mean.

The Chromatic Edit Model Algorithm (ChromEdit)[13]: The chromatic video edit model for a gradual scene change assumes the transition is composed of two piece-wise linear functions of time, one decreasing and the other increasing. From the model the authors note that within the linear transition domain, the first partial derivatives with respect to time are constant, and the second partial derivatives are zero. Testing for the latter case by itself is not sufficient since static image sequences with also yield zero second derivative values. Allowing for noise, they test for the second derivative to be less than a fraction of the first partial derivatives. The frames for which this test is satisfied are part of the transition. A series of frames, tolerating noise, for which the test is satisfied is identified as a gradual transition. The paper does not discuss appropriate values for the threshold, but from practice we observe that a high value seems to be required to detect even few transitions correctly. We used 0.8 as the fraction for the threshold.

5. EVALUATION PROCEDURE

All the video clips in our test sets were manually ground truthed for various kinds of special effect edits. The end points of the edits were identified after repeated and careful viewing of the video clips.

Using the ground truth of the clips, we wish to evaluate the performance of the candidate algorithms. The algorithms in this scenario can be evaluated on two separate criteria — (1) whether the algorithm was able to detect the presence of the special effect edits, and (2) whether the algorithm the was able to correctly localize the start and end points of the special effects edits. The next two subsections detail these two evaluation criteria.

5.1. Precision and Recall

The use of precision and recall metrics for evaluating detection performance has been widely accepted. The following expressions explain how the metrics are calculated:

$$Recall = \frac{\text{Number Of Correct Detections}}{\text{Number Of Records In Ground Truth}}$$

$$Precision = \frac{\text{Number Of Correct Detections}}{\text{Number Of Detections}}$$

The recall is therefore the percentage of the total number of ground truth records that were correctly detected, and the precision is the percentage of the total number of detections that were correct.

5.2. Temporal Localization

As part of the detection process, each ground truth target can initially be associated with a candidate match. For example, we could consider a candidate which overlaps temporally for at least one frame, as a candidate match. Temporal localization then defines how well a given object was detected in time. When matching objects temporally, we consider only metrics on the range of frames over which the target is valid. We need metrics that compare the quality of match of two ranges.

For a given pair of ranges (one from the target and one from the candidate), we will define two metrics:

Overlap Coefficient - the percentage of frames in the ground truth target range which are also in the detected candidate range. Note this measure is not symmetric. Maximum possible value is 100%. For example, if the ground truth target range is [45:52] and the candidate range is [44:56], then the Overlap Coefficient is 100%. If the candidate range is [44:49], then the Overlap Coefficient $= \frac{5}{8} = 62.5\%$.

Dice Coefficient - a normalized measure of the number of frames in common, providing a measure between [0,1]:

$$\text{Dice Coefficient} \quad = \quad \frac{2 * |\text{Range}_{target} \cap \text{Range}_{candidate}|}{|\text{Range}_{target}| + |\text{Range}_{candidate}|}$$

where the |Range| stands for the number of frames included in the range, and the $\cap$ operator on two ranges gives the range that is common to both ranges.

The Dice coefficient for the above two candidates will be $\frac{2*8}{8+13} = 0.76$, and $\frac{2*5}{8+6} = 0.71$.

Sometimes, more than one candidate will temporally overlap a ground truth target. In these cases, we can sum their individual match coefficients to yield the final performance values.

6. EXPERIMENTS

Let us denote the 7 algorithms that we are evaluating as follows — Plateau, VarCurve, TwinComp, ChromEdit, VT-PixHist, VT-Hist, and VT-Pix. The *Video Trails* based algorithms VT-PixHist, VT-Hist, and VT-Pix used *Video Trails* generated in 10 dimensions using YUV input data. The non-*Video Trails* based algorithms Plateau, VarCurve, TwinComp, and ChromEdit also used YUV input data. In all cases, the YUV input data were in the form of reduced DC images of all frames of the MPEG clips.

For evaluating the performance of the algorithms, we conducted two sets of experiments. In the first set of experiments, we wanted to evaluate the versatility of the detection techniques. We took two unrelated 10 second video clips, and combined them using 15 different types of special effect edits. These 15 edits are shown in Figure 4.

Most of these edits are non-linear edits. They span a wide range of edits from dissolves (AdditiveDissolve, Dither) to fancy wipes (ClockWipe, Curtain) to multiple grid-based edits (BandSlide, MultiSpin). Using this test set, we will be able to see which algorithms are the most versatile and which algorithms fail on certain types of edits. Moreover, at the end of the second clip, there is a beginning of special effect edit that was intentionally terminated prematurely, i.e., we did not let the edit proceed to completion. We wanted to see if the algorithms were able to detect these partial edits at the start or end of a video clip. We applied each of the algorithms to these newly edited video clips, and results and their discussion are presented in the next section.

In the second set of experiments, we applied the candidate techniques to real-world movie clips and edits. We ran the 7 algorithms on approximately 90000 frames of MPEG video amounting to almost 50 minutes of video. It contained 441 special effect edits of which a majority were dissolves. There were also many fancy special effect edits taken from sporting events present in the test set. The clips contained a wide variety of content including documentary clips, commercials, prime-time TV news magazines, sporting events etc. Within the sports clips, there were also many documentary-style features of athletes. The duration of edits ranged from extremely large (> 100 frames) to very short (5 or 6 frames). Some clips had edits in quick succession. Moreover, many edits were between similar shots. All in all, our test set was designed to be a really challenging one.

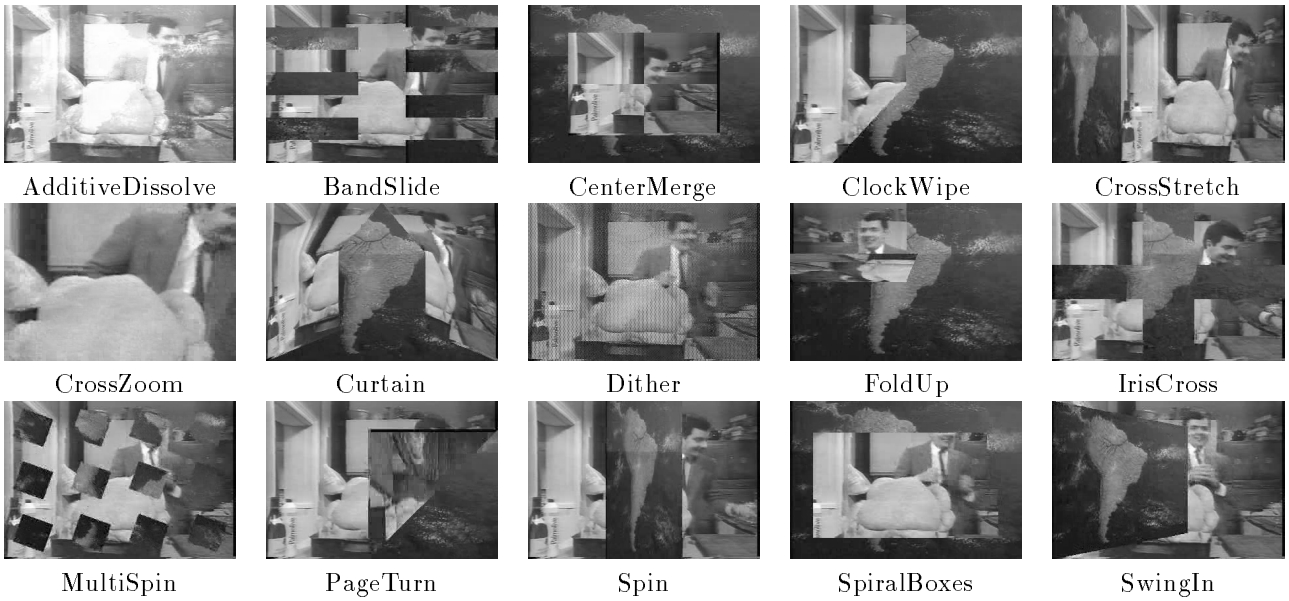


Figure 4. Various video edits used in Experiment 1.

Table 1. Correct, Partial and False Detections of various special effect edits.

	Plateau			VarCurve			TwinComp			ChromEdit			VT-PixHist			VT-Hist			VT-Pix		
	C	P	F	C	P	F	C	P	F	C	P	F	C	P	F	C	P	F	C	P	F
AddDissolve	0	0	2	1	0	1	1	0	0	0	1	1	1	0	1	1	0	1	1	0	5
BandSlide	1	0	3	0	0	2	1	0	3	0	0	1	1	0	1	1	0	2	1	0	2
CenterMerge	1	0	2	1	0	1	1	0	3	0	1	1	1	0	0	1	0	0	1	0	3
ClockWipe	0	1	2	1	0	1	1	0	4	0	0	1	1	0	0	1	0	1	1	0	3
CrossStretch	1	0	2	0	1	1	1	0	3	0	0	1	1	0	0	1	0	1	1	0	4
CrossZoom	1	0	2	0	0	1	0	1	1	0	0	1	0	0	0	0	0	1	0	1	2
Curtain	1	0	2	0	1	1	1	0	3	1	0	1	1	0	2	1	0	2	1	0	4
Dither	1	0	2	1	0	1	1	0	0	0	0	1	1	0	2	0	0	0	1	0	6
FoldUp	1	0	2	1	0	1	1	0	3	0	0	1	1	0	1	1	0	1	1	0	2
IrisCross	1	0	2	0	0	1	1	0	4	1	0	1	1	0	1	1	0	0	1	0	2
MultiSpin	1	0	2	0	0	1	1	0	3	0	0	1	1	0	1	0	1	0	1	0	4
PageTurn	1	0	2	0	1	1	1	0	4	0	0	1	1	0	0	1	0	1	1	0	2
Spin	1	0	2	0	0	1	1	0	3	0	1	1	1	0	0	1	0	0	1	0	3
SpiralBoxes	1	0	2	0	1	1	1	0	4	0	0	1	1	0	1	1	0	2	1	0	4
SwingIn	1	0	2	0	0	1	1	0	4	0	0	1	1	0	0	1	0	0	1	0	3
Total	13	1	31	5	4	16	14	1	42	2	3	15	14	0	10	12	1	12	14	1	49
Recall %	93.3			60			100			33.3			93.3			86.7			100		
Precision %	31.1			36			26.3			25			58.3			52			23.4		

7. RESULTS AND DISCUSSION

Table 1 lists the correct, partial, and false detections of the 7 algorithms under columns C, P, and F under each algorithm's heading for the first experiment. The total number of correct, partial, and false detections for each algorithm are summed up at the bottom of the table. The precision and recall percentages are also shown at the bottom. While calculating precision and recall, we took partial detections to be correct detections. Similarly, Table 2 shows the Overlap and Dice coefficients under columns Ov and Dice. The average values of the Ov and Dice coefficients are shown in the bottom row. Only the detected values were used in calculating the average. Partial detections were the result of the overlap coefficient being less than 50%. The correct and partial detections shown

Table 2. Overlap and Dice Coefficients of detections.

	Plateau		VarCurve		TwinComp		ChromEdit		VT-PixHist		VT-Hist		VT-Pix	
	Ov	Dice	Ov	Dice	Ov	Dice	Ov	Dice	Ov	Dice	Ov	Dice	Ov	Dice
AddDis	-	-	72.2	0.83	97.2	0.86	38.8	0.56	91.6	0.85	80.5	0.89	94.4	0.83
BndSld	81.5	0.81	-	-	81.5	0.81	-	-	76.3	0.82	68.4	0.81	68.4	0.81
CtrMrg	72.9	0.79	91.8	0.95	81.0	0.78	45.9	0.62	89.1	0.92	94.5	0.95	94.6	0.85
ClkWpe	45.9	0.62	100.0	1.0	100.0	0.89	-	-	94.6	0.85	81.0	0.85	94.6	0.85
CrsStr	100.0	0.98	37.8	0.54	100.0	0.89	-	-	94.6	0.85	94.6	0.88	94.6	0.85
CrsZm	67.5	0.80	-	-	18.9	0.31	-	-	-	-	-	-	13.5	0.23
Curtn	70.2	0.82	35.1	0.52	100.0	0.89	54.0	0.70	94.6	0.90	94.6	0.94	94.6	0.85
Dither	61.7	0.76	100.0	0.95	85.2	0.89	-	-	58.8	0.73	-	-	66.7	0.8
FoldUp	68.5	0.81	94.2	0.95	100.0	0.86	-	-	97.1	0.85	97.2	0.96	97.1	0.85
IrCrs	100.0	0.97	-	-	100.0	0.87	55.5	0.71	97.2	0.86	75.0	0.73	97.2	0.86
MltSpn	56.7	0.71	-	-	100.0	0.88	-	-	89.1	0.94	35.1	0.52	97.2	0.86
PgTurn	80.0	0.87	40.0	0.56	100.0	0.86	-	-	97.1	0.93	97.1	0.88	97.1	0.85
Spin	58.3	0.73	-	-	100.0	0.86	27.7	0.43	100.0	0.88	94.4	0.94	100.0	0.87
SplBox	59.4	0.74	37.8	0.54	100.0	0.89	-	-	94.6	0.85	94.6	0.86	94.6	0.85
SwngIn	97.2	0.97	-	-	100.0	0.87	-	-	97.2	0.95	94.4	0.85	97.2	0.86
Avg	72.8	0.81	67.6	0.76	90.9	0.83	44.4	0.60	90.9	0.87	84.7	0.85	86.8	0.81

Table 3. Number of correct detections of the curtailed edit at end of clip.

Plateau	VarCurve	TwinComp	ChromEdit	VT-PixHist	VT-Hist	VT-Pix
0	0	0	0	15	14	2

in Tables 1 and 2 are for the special effect edits between the two 10 second shots. The results of the curtailed edit at the end of the clip is shown in Table 3.

From Table 1, it can be clearly seen that VT-PixHist performs best as it found 14 out of 15 edits correctly, while keeping the number of false detections comparatively low. The performance of VT-Hist was the next best. Of the non-*Video Trails* based algorithms, Plateau and TwinComp gave good recall results but had many false detections (in spite of removing some due to Motion Transition Removal). VT-Pix also detected many false transitions, while VarCurve and ChromEdit gave poor recall results. Table 2 shows that the best temporal localization is obtained with VT-PixHist and TwinComp, with VT-Hist and VT-Pix a little behind.

By far the most troublesome edit as far as recall is concerned was CrossZoom. This is due to the fact that the CrossZoom edit is actually composed of a fast Zoom-In of the first shot, followed by a cut into a fully zoomed image of the second shot, followed by a Zoom-Out of the second shot to its normal size. The camera motion detection algorithm detected a major part of the two zooms and whatever detections some of the algorithms were able to make, were eventually removed.

From Table 3, it is evident that only the *Video Trails* based techniques were able to detect the curtailed edits. VT-PixHist and VT-Hist were almost perfect.

Figure 5 shows plots of the outputs of the Plateau and the Variance Curve algorithms. The plots are shown from frame 200 to 350 for the videos in the test set of the first experiment. The actual special effect edit was from frame 265 to frame 300 as indicated by the two vertical lines in each plot. For the Plateau algorithm, the plotted values are the deferred difference values ($k = 40$). Since the value of k was 40 and the length of the edits were 35, a "plateau" of length 5 (from 260 to 265) in those plots (Figures 5 (a), (c), and (e)) can be seen as predicted. But, for certain edits, the sides of the "plateau" do not drop off ideally and this leads to detection problems. Also, for smaller values of k, the detection of a plateau is probably not possible.

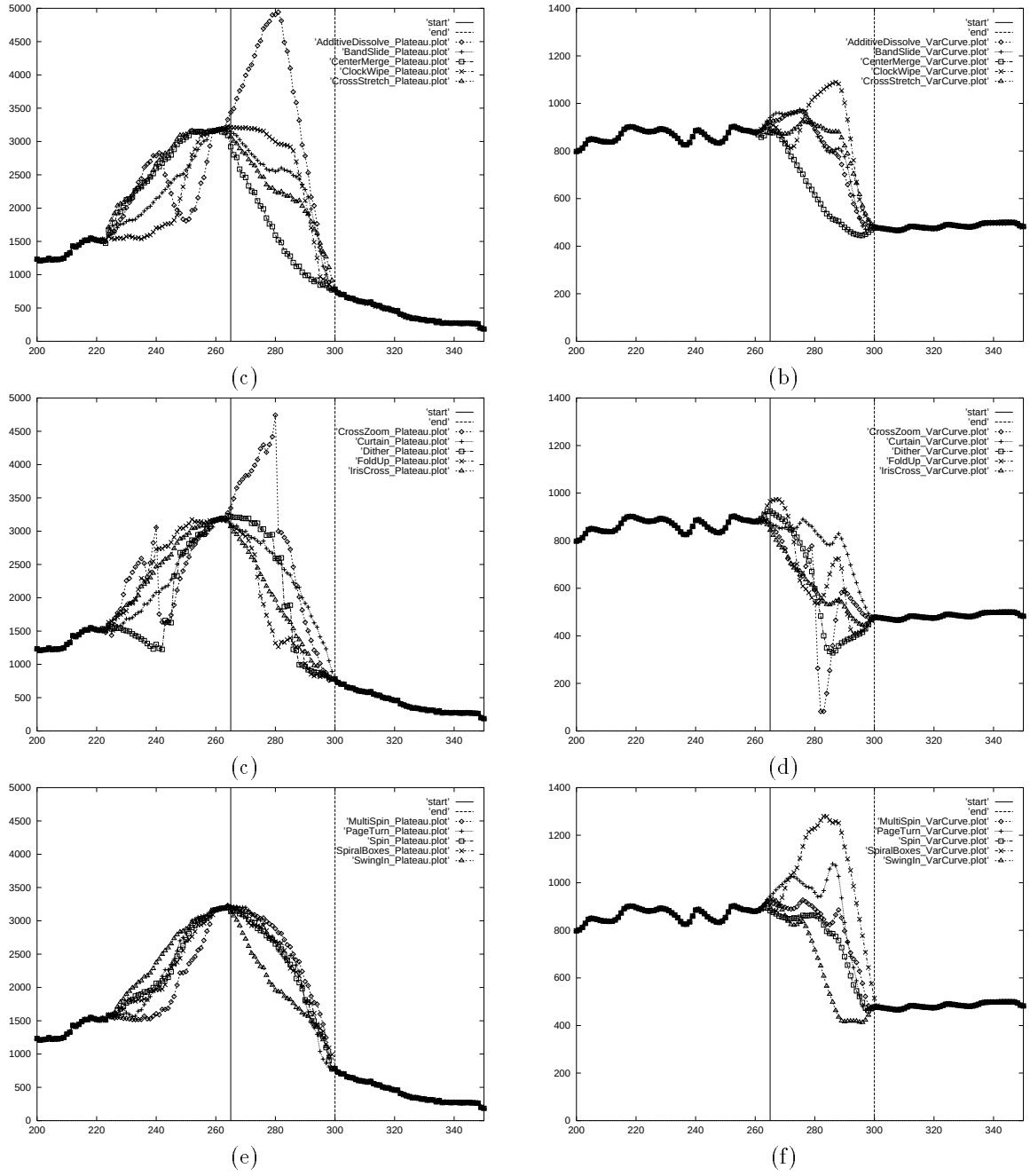


Figure 5. Plots of Plateau and VarCurve algorithms from frame 200 to 350. The actual edit was from frame 265 to frame 300 as indicated by the vertical lines: (a), (c), and (e) Deferred difference plots of the Plateau algorithm for the 15 edits; (b), (d), and (f) Variance plots of the VarCurve algorithm for the 15 edits.

For the VarCurve algorithm, the plotted values are the variances of each frame. Clearly, very few edits yield plots even remotely resembling a a parabolic curve. For many edits, it is evident that the variances are very high around the middle of the edits.

The TwinComp algorithm is heavily dependent on the "Motion Transition Removal Test". Even slight motion in the video, or, the appearance or disappearance of new objects leads to false detections.

Table 4. Number of correct and false detections, and the precision and recall percentages from experiment 2.

	# Correct	# False	Precision	Recall
Plateau	218	177	55.1	49.4
VarCurve	280	298	48.4	63.4
TwinComp	270	232	53.7	61.2
ChromEdit	81	47	63.2	18.3
VT-PixHist	275	119	69.7	62.3
VT-Hist	249	100	71.3	56.4
VT-Pix	300	225	57.1	68.0

The results of the second experiment are shown in Table 4. The first column shows the number of correct detections (out of 441), the second column shows the number of false detections, the third shows the precision percentages and the last column shows the recall values. VT-PixHist performed best overall with good precision and recall results. VT-Pix had the highest recall but also gave many false positives. VT-Hist gave the best precision while compromising on the recall. Of the non-*Video Trails* based algorithms, TwinComp and VarCurve yielded good recall percentages, but also had a large number of false detections. The poor performance of ChromEdit can be attributed partly to the input set. The DC estimation algorithm generates DC images sequences where the intensity slightly fluctuates between I, P, and B frames, and the first derivatives did not display the constancy that might be expected. Even smoothing did not offer much help. The algorithm will probably perform better with full-frame video.

Many of the false detections occured due to large motion in sports clips, or in close-up shots where the heads is moving constantly. Our motion transition removal test was able to remove only the global motion or camera motion. With better motion detection for independent object motion, or multiple motions, the performance of most of these algorithms should improve.

A typical case leading to missed detections was when an edit combined similar shots. This is the case, for example, when a transition occurs between shots from two cameras focused on the same scene. Some missed detections were due to the fact that one or both of the shots being combined with a special effect edit could be undergoing motion as the edit occurs. In such cases, the motion detector misclassified gradual transitions as motion transitions.

Finally, some of the thresholds in the algorithms could be varied to give slightly better performance with precision or recall. Though typically, when a particular threshold in varied to yield better recall performance, the precision performance becomes poorer and vice versa. Whenever possible, we chose the thresholds suggested by the authors in the implementation of the non-*Video Trails* algorithms. Otherwise, we made suitable choices.

8. CONCLUSION

From the first set of experiments, we can see that the *Video Trails* based algorithms have a lot of potential for detecting a wide variety of special effect edits with good accuracy. The temporal localization results prove that they are better able to pinpoint the starting and ending frames of the special effect edits. The ability of the *Video Trails* based techniques to detect partial edits is a further advantage of the technique. The Plateau and the Twin Comparison techniques were able to achieve good results on recall but had too many false detections to be reliable. The performance of the Plateau algorithm severly depends on the separation parameter k. The Variance Curve algorithms seems to perform excellently on edits that resemble dissolves, but fails on most special effect edits. This is evident from the second experiment where most of the 441 edits were dissolves and the Variance Curve algorithm achieved good recall percentages. The ChromEdit algorithm is probably better suited for full-frame video than the reduced DC image sequences used in our experiments.

Among the *Video Trails* based algorithms, VT-PixHist with the weighted pixel-wise and histogram-wise distances gave the best performance while the VT-Hist and VT-Pix gave good precision and recall percentages respectively while compromising a little on the other.

REFERENCES

1. V. Kobla, D. Doermann, and K. Lin, "Archiving, indexing, and retrieval of video in the compressed domain," in *Proc. of the SPIE Conference on Multimedia Storage and Archiving Systems*, vol. 2916, pp. 78–89, 1996.
2. B. Yeo and B. Liu, "Unified approach to temporal segmentation of Motion JPEG and MPEG video," in *Proc. of the International Conference on Multimedia Computing and Systems*, pp. 2–13, 1995.
3. J. Meng, Y. Juan, and S. Chang, "Scene change detection in a MPEG compressed video sequence," in *Proc. of the SPIE Conference on Digital Video Compression: Algorithms and Technologies*, vol. 2419, pp. 14–25, 1995.
4. I. Sethi and N. Patel, "A statistical approach to scene change detection," in *Proc. of the SPIE Conference on Storage and Retrieval for Image and Video Databases III*, vol. 2420, pp. 329–338, 1995.
5. H. Liu and G. Zick, "Scene decompostion of MPEG compressed video," in *Proc. of the SPIE Conference on Digital Video Compression: Algorithms and Technologies*, vol. 2419, pp. 26–37, 1995.
6. B. Shahraray, "Scene change detection and content-based sampling of video," in *Proc. of the SPIE Conference on Digital Video Compression: Algorithms and Technologies*, vol. 2419, pp. 2–13, 1995.
7. R. Zabih, J. Miller, and K. Mai, "Feature-based algorithm for detecting and classifying scene breaks," in *Proc. of the ACM Multimedia Conference*, pp. 189–200, 1995.
8. A. Hampapur, R. Jain, and T. Weymouth, "Digital video segmentation," in *Proc. of the ACM Multimedia Conference and Exposition*, pp. 357–364, 1994.
9. F. Arman, A. Hsu, and M. Chiu, "Image processing on compressed data for large video databases," in *Proc. of the ACM Multimedia Conference*, pp. 267–272, 1993.
10. H. Zhang, A. Kankanhalli, and S. Smoliar, "Automatic partitioning of full-motion video," *Multimedia Systems* **1**, pp. 10–28, 1993.
11. V. Kobla, D. Doermann, and C. Faloutsos, "Developing high-level representations of video clips using *VideoTrails*," in *Proc. of the SPIE Conference on Storage and Retrieval for Still Image and Video Databases VI*, vol. 3312, pp. 81–92, 1998.
12. B. Yeo and B. Liu, "On the extraction of DC sequence from MPEG compressed video," in *Proc. of the IEEE International Conference on Image Processing*, vol. 2, pp. 260–263, 1995.
13. S. M.-H. Song, T.-H. Kwon, and W. M. Kim, "Detection of gradual scene changes for parsing of video data," in *Proc. of the SPIE Conference on Storage and Retrieval for Still Image and Video Databases VI*, vol. 3312, pp. 404–413, 1998.
14. U. Gargi, S. Antani, and KasturiR., "Performance characterization and comparison of video indexing algorithms," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 559–565, 1998.
15. C. Faloutsos and K. Lin, "*FastMap*: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *Proc. of the ACM SIGMOD Conference*, pp. 163–174, 1995.
16. V. Kobla, D. Doermann, K.-I. Lin, and C. Faloutsos, "Compressed domain video indexing techniques using DCT and motion vector information in MPEG video," in *Proc. of the SPIE Conference on Storage and Retrieval for Still Image and Video Databases V*, vol. 3022, pp. 200–211, 1997.
17. V. Kobla, D. Doermann, and C. Faloutsos, "*VideoTrails*: Representing and visualizing structure in video sequences," in *Proc. of the ACM Multimedia Conference*, pp. 335–346, 1997.