A Feature-Based Algorithm for Detecting and Classifying Scene Breaks

Ramin Zabih
Justin Miller
Kevin Mai
Computer Science Department
Cornell University
Ithaca, NY 14853
{rdz,jmiller,kmai}@cs.cornell.edu
607-255-8413

ABSTRACT

We describe a new approach to the detection and classification of scene breaks in video sequences. Our method can detect and classify a variety of scene breaks, including cuts, fades, dissolves and wipes, even in sequences involving significant motion. We detect the appearance of intensity edges that are distant from edges in the previous frame. A global motion computation is used to handle camera or object motion. The algorithm we propose withstands JPEG and MPEG artifacts, even at very high compression rates. Experimental evidence demonstrates that our method can detect and classify scene breaks that are difficult to detect with previous approaches. An initial implementation runs at approximately 2 frames per second on a Sun workstation.

KEYWORDS

Content-based indexing and retrieval; video processing

1 INTRODUCTION

The amount of digital video that is available has increased dramatically in the last few years, but the tools available for browsing video remain quite primitive. Computer vision techniques promise to allow content-based browsing of image sequences. For example, we may be able to replace the "fast forward" button on current video browsers with a button that searches for the next dissolve, or for the next time a moving object enters the scene. This will require algorithms to automatically detect these events. This paper presents algorithms for detecting and classifying scene breaks (including cuts, fades, dissolves and wipes) in digital video sequences.

Scene breaks mark the transition from one sequence of consecutive images (or scene) to another. A cut is an instantaneous transition from one scene to the next. A fade is a gradual transition between a scene and a constant image (*fade out*) or between a constant image and a scene (*fade in*). During a fade, images have their intensities multiplied by some value α. During a fade in, α increases from 0 to 1, while during a fade out α decreases from 1 to 0. The speed with which α changes controls the fade rate. A dissolve is a gradual transition from one scene to another, in which the first scene fades out and the second scene fades in. Typically, fade out and fade in begin at the same time, and the fade rate is constant. Another common scene break is a wipe, in which a line moves across the screen, with the new scene appearing behind the line.

The detection and classification of scene breaks is a first step in the automatic annotation of digital video sequences. The problem is also important for other applications, including compression and automatic keyframing. Motion-based compression algorithms like MPEG can obtain higher compression rates without sacrificing quality when the locations of scene breaks are known. Knowledge about scene breaks can be used to look for higher-level structures (such as a sequence of cuts between cameras), or to ensure that keyframes come from different scenes.

We begin with a survey of related work on scene break detection. These methods rely directly on intensity data, and have difficulty with dissolves and with scenes involving motion. We then present our feature-based approach to the problem, which detects the appearance of new edges far from old ones. We show that our methods have substantial tolerance for compression artifacts. We present empirical evidence that our algorithm can outperform conventional approaches, especially on images involving motion. Finally, we discuss some of the current limitations of our algorithm and describe extensions which we hope will overcome them.

1.1 Existing algorithms for detecting scene breaks

Existing work has focused on cut detection, which can be done with reasonable accuracy by a variety of simple schemes. There has been a small amount of work on detecting dissolves — a much harder problem. All of these approaches have relied directly on intensity data. In section 4 we will present an empirical comparison of these methods with the feature-based approach that we propose. The evidence shows that our method can detect scene breaks in a variety of sequences that are difficult to handle with existing intensity-based approaches.[1]

Computational schemes for detecting scene breaks generally define a similarity measure between consecutive images. When two consecutive images are sufficiently dissimilar, there may be a scene break. Typically the similarity measure is smoothed and thresholded, using methods such as those of [11]. This is done to reduce the effects of noise and to prevent the detector from signaling too many scene breaks in a short period of time.

Several researchers have proposed algorithms for detecting cuts and dissolves. These methods have relied directly on intensity data, and have used such techniques as image differencing (which subtracts two consecutive images to determine changes in intensity) and intensity histogramming. Most approaches are based on intensity histograms, and concentrate on cuts.

Otsuji and Tonomura [11] discuss a variety of measures based on image differencing and changes in the image's intensity histogram. Nagasaka and Tanaka [10] present algorithms which use similar measures. Arman, Hsu and Chiu [2] have addressed the issue of change detection while operating directly on JPEG or MPEG encoded video; this approach is noteworthy for its efficiency.

The above methods have difficulty with "busy" scenes, in which intensities change substantially from frame to frame. This change often results from motion: if the camera is moving, or if an object that occupies much of the image is moving, then many pixels will change their values from frame to frame. Motion can cause intensity-based similarity measures to produce a low similarity score, which can result in a false positive from the detector.

Dissolves are more difficult to detect than cuts, especially if the scenes may involve motion. False positives resulting from motion can be supressed by setting a high threshold. Cuts usually result in a dramatic change in image intensities, so they can still be detected much of the time. However, a dissolve is a gradual change from one scene to another, and thus cannot be easily distinguished from motion. A dissolve can even occur between two scenes which each contain motion; this case is particularly difficult to detect.

Hampapur, Jain and Weymouth [6] describe a method they call chromatic scaling, which attempts to detect a variety of scene breaks based on an explicit model of the video production process. While their approach is intensity-based, it does not involve histogramming. Instead, they compute a chromatic image from a pair of consecutive images. Its value at each pixel is the change in intensity between the two images divided by the intensity in the later image. Ideally, the chromatic image should be uniform and non-zero during a fade.

The difficulties caused by motion and by dissolves are well-known. For example, Hampapur, Jain and Weymouth note in their discussion of dissolves that their measure "is applicable if the change due to the editing dominates the change due to motion"[6, page 11], and describe both object and camera motion as causes of false positives for their method. Another recent paper [17] describes motion as a major limitation of histogram-based methods.

A particularly interesting approach has been taken by Zhang, Kankanhalli and Smoliar [17]. They have extended conventional histogram-based approaches to handle dissolves and to deal with certain camera motions. They use a dual threshold on the change in the intensity histogram to detect dissolves. In addition, they have a method for avoiding the false positives that result from certain classes of camera motion, such as pans and zooms. They propose to detect such camera motion and suppress the output of their scene-break measure during camera motion.

Their method does not handle false positives that arise from object motion, or from more complex camera motions. Nor does their method handle false negatives, which would occur in dissolves between scenes involving motion. In section 4 we will provide an empirical comparison of our method with Zhang, Kankanhalli and Smoliar's algorithm, as well as Hampapur, Jain and Weymouth's chromatic scaling technique.

2 A FEATURE-BASED APPROACH

Our approach is based on a simple observation: 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. We define an edge pixel that appears far from an existing edge pixel as an *entering* edge pixel, and an edge pixel that disappears far from an existing edge pixel as an *exiting* edge pixel. By counting the entering and exiting edge pixels, we can detect and classify cuts, fades and dissolves. By analyzing the spatial distribution of entering and exiting edge pixels, we can detect and classify wipes. A more detailed description of the algorithm, with additional examples of its performance, can be found in [15].

The algorithm we propose takes as input two consecutive images I and I'. We first perform an edge detection step, resulting in two binary images E and E'. Let ρ_{in} denote the fraction of edge pixels in E' which are more than a fixed distance r from the closest edge pixel in E. ρ_{in}

[1]It is a slight misnomer to call existing approaches intensity-based, since any algorithm must be based on intensities. We will use the term in contrast with our feature-based approach.

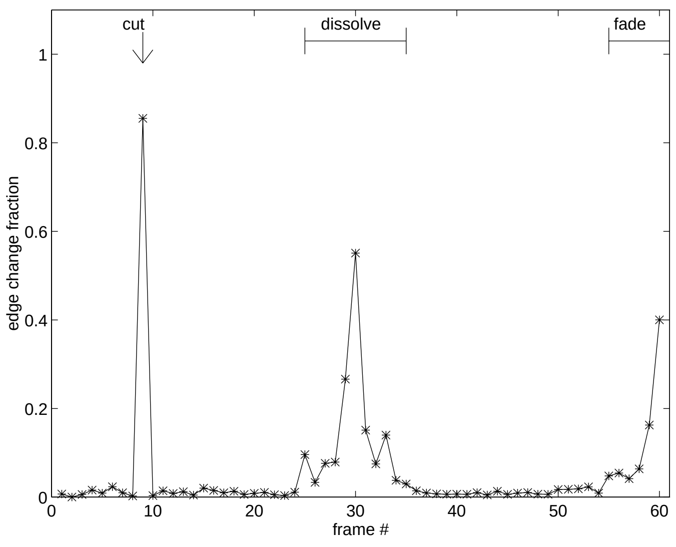


Figure 1: Results from the table tennis sequence

measures the proportion of entering edge pixels. It should assume a high value during a fade in, or a cut, or at the end of a dissolve.[2]

Similarly, let ρ_{out} be the fraction of edge pixels in E which are farther than r away from the closest edge pixel in E'. ρ_{out} measures the proportion of exiting edge pixels. It should assume a high value during a fade out, or a cut, or at the beginning of a dissolve.

Our basic measure of dissimilarity is

$$\rho = \max(\rho_{in}, \rho_{out}). \tag{1}$$

This represents the fraction of changed edges; this fraction of the edges have entered or exited. Scene breaks can be detected by looking for peaks in ρ, which we term the *edge change fraction.*

An example of the edge change fraction is shown in figure 1. The sequence we have chosen is the widely-known "table tennis" sequence which was used to benchmark MPEG implementations. The original sequence contains a fair amount of motion (including zooms) plus a few cuts. To demonstrate our algorithm, we have spliced together several parts of the sequence and inserted a few scene breaks. The modified table tennis sequence contains a cut (taken from the original sequence) between frames #9–#10. We have added a dissolve in frames #25–#35, and then a fade out starting at frame #55. On this sequence, ρ shows clear peaks at the scene breaks, and the detection and the classification algorithm described in section 3.1 performed correctly.

2.1 Motion compensation

Our method can be easily extended in order to handle motion. We can use any registration technique [3] to compute a global motion between frames. We can then apply this global motion to align the frames before detecting entering or exiting edge pixels. For example, assume that the camera is moving to the left, and so image I' is shifted to the right with respect to image I. A registration algorithm will compute the translation that best aligns I with I' (which in this example is a shift to the right). We can apply this translation to I before computing the edge change fraction as shown above.

There are a wide variety of registration algorithms reported in the literature. The ones we have used involve global similarity measures between images, and are based on correlation. We begin with some function f for comparing two images on a pixel-by-pixel basis. We search for the integer values of δx and δy that maximize the quantity

$$\sum_{x,y} f(I[x+\delta x, y+\delta y], I'[x,y]) \tag{2}$$

where the sum is taken over all pixels. We maximize over some range of possible motions

$$\|\delta x\| + \|\delta y\| \leq \Delta. \tag{3}$$

Note that we only search for a translational motion between the two images. While it is possible to handle affine or projective motions, they incur significant additional overhead, and do not necessarily result in better performance. We can then warp I by the overall motion before computing ρ_{in} and ρ_{out}.

We need a registration algorithm that is efficient, that can withstand compression artifacts, and that is robust in the presence of multiple motions. The last property is particularly important, since we will often be faced with an image with multiple motions, and our registration algorithm must compute the predominant motion.

We have explored two algorithms, both of which have given satisfactory results. We have experimented with using census transform correlation, a non-parametric approach developed in [16]. This algorithm operates by transforming the image in an outlier-tolerant manner and then using correlation. We have also used the Hausdorff distance [8], an outlier-tolerant method described in section 3.2 that operates on edge-detected images.

It is tempting to exploit the motion vectors contained in MPEG-compressed video in order to determine object or camera motion. Indeed, a number of researchers [17] have attempted to do this. There are a number of reasons that we have not taken this approach. MPEG encoders optimize for compression, and do not necessarily produce accurate motion vectors. MPEG-compressed streams do not contain motion vectors for all images; in fact, if the encoder chooses to create only I-frames, there will be no motion vectors at all.[3]

[2]Due to the quantization of intensities, new edges will generally not show up until the end of the dissolve.

[3]This is not as unusual a situation as one might imagine. About one quarter of the MPEG sequences we have come across on the World-Wide Web are compressed with only I-frames.

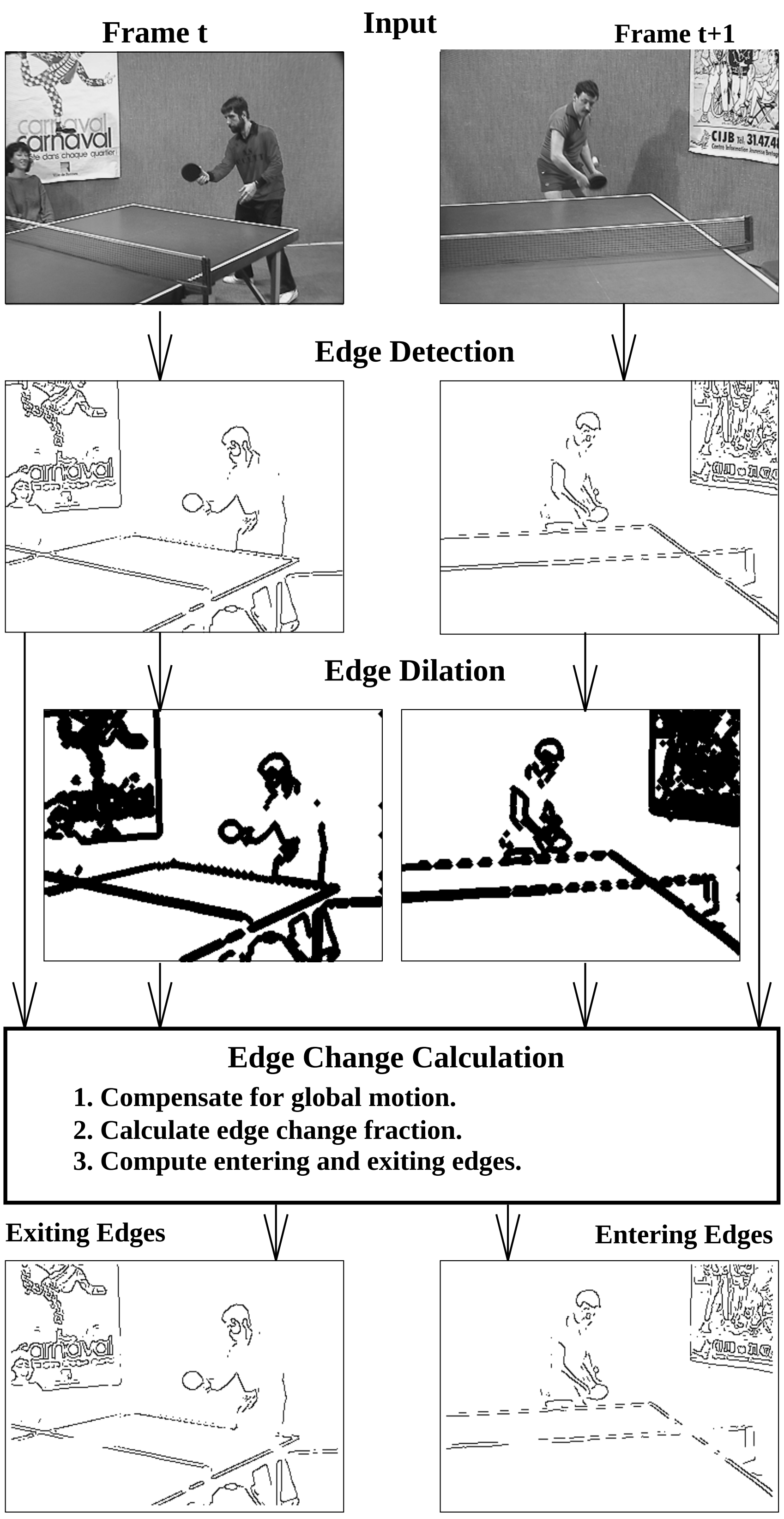


Figure 2: Main steps of the computation of the edge change fraction

3 COMPUTING THE EDGE CHANGE FRACTION

Computing the values of ρ for a sequence of images is straightforward. Let δx and δy be the translations necessary to align the images I and I', as calculated by one of the global motion compensation methods discussed in section 2.1. The first step in our algorithm is edge detection.

In our experiments, we have used an edge detector based on Canny's algorithm [5]. We first smooth the image by convolving it with a Gaussian of width σ. We next compute the gradient magnitude, which indicates how fast the local intensities are changing. The gradient magnitude is thresholded at a value of τ to detect edges, Canny-style non-maximum suppression is performed. For additional efficiency we implement Gaussian smoothing by using a small number of box filters, as described in [14].

Next, copies of E and E' are created with each edge pixel dilated by a radius r. Let us call these dilated images $\bar{E}$ and $\bar{E}'$. Thus image $\bar{E}$ is a copy of E in which each edge pixel of E is replaced by a diamond whose height and width are $2r+1$ pixels in length.[4] Similarly, image $\bar{E}'$ is a dilated copy of E'.

Consider ρ_{out}, the fraction of edge pixels in E which are farther than r away from an edge pixel in E'. A black pixel $E[x,y]$ is an exiting pixel when $\bar{E}'[x,y]$ is not a black pixel (since the black pixels in $\bar{E}'$ are exactly those pixels within distance r of an edge pixel in E'). The equation for ρ_{out} is

$$\rho_{out} \quad = \quad 1 - \frac{\sum_{x,y} E[x+\delta x, y+\delta y]\bar{E}'[x,y]}{\sum_{x,y} E[x,y]} \qquad (4)$$

which is the fraction of edge pixels which are exiting. ρ_{in} is calculated similarly

$$\rho_{in} \quad = \quad 1 - \frac{\sum_{x,y} \bar{E}[x+\delta x, y+\delta y]E'[x,y]}{\sum_{x,y} E[x+\delta x, y+\delta y]} \qquad (5)$$

similarly. The edge change fraction ρ shown in equation 1 is the maximum of these two values.

The major steps of the computation of the edge change fraction are shown in figure 2. The example shows a cut between the two frames. While ρ is being calculated the locations of the exiting and entering pixels can be saved and their spatial distribution analyzed when looking for wipes and other spatial edits.

3.1 Peak detection and classification

We propose to detect scene breaks by looking for peaks in the edge change fraction ρ. We have designed a simple thresholding scheme for peak detection. We use an *event threshold* and an *event horizon*. A frame where ρ exceeds the event threshold may be a scene break. To localize scene breaks that occur over multiple frames, we restrict scene breaks to occur only when ρ is a local maximum within a fixed window of consecutive frames. The width of this window is the event horizon.

[4]To use the Manhattan distance between edges, we dilate by a diamond. If we were to use the Euclidean distance between edges, we would dilate by a circle.

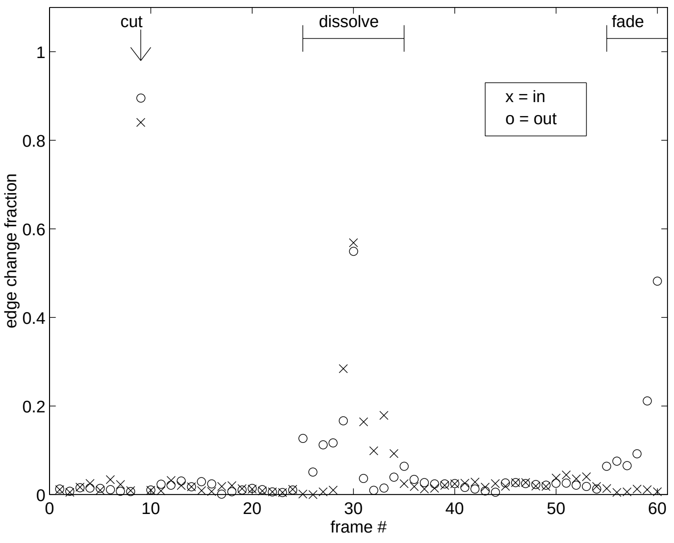


Figure 3: Values of ρ_{in} (shown as "$\times$") and ρ_{out} (shown as "o") in the table tennis sequence

3.1.1 Classification

Once a peak has been detected, the next problem is to classify it as a cut, dissolve, fade or wipe. Cuts are easy to distinguish from other scene breaks, because a cut is the only scene break that occurs entirely between two consecutive frames. As a consequence, a cut will lead to a single isolated high value of ρ, while the other scene breaks will lead to an interval where ρ's value is elevated. This allows us to classify cuts.

Fades and dissolves can be distinguished from each other by looking at the relative values of ρ_{in} and ρ_{out} in a local region. During a fade in, ρ_{in} will be much higher than ρ_{out}, since there will be many entering edge pixels and few exiting edge pixels. Similarly, at a fade out, ρ_{out} will be higher than ρ_{in}, since there will be many exiting edge pixels, but few entering edge pixels. A dissolve, on the other hand, consists of an overlapping fade in and fade out. During the first half of the dissolve, ρ_{in} will be greater, but during the second half ρ_{out} will be greater.

Figure 3 shows the values of both ρ_{in} and ρ_{out} during the table tennis sequence. During the fade out, ρ_{out} is much higher than ρ_{in}. During the dissolve, there is an initial peak in ρ_{in} followed by a peak in ρ_{out}.

3.1.2 Wipes

Wipes are distinguished from dissolves and fades by looking at the spatial distribution of entering and exiting edge pixels. During a wipe, each frame will have a portion of the old scene and a portion of the new scene. Between adjacent frames, a single strip of the image will change from the old scene to the new scene. For a horizontal

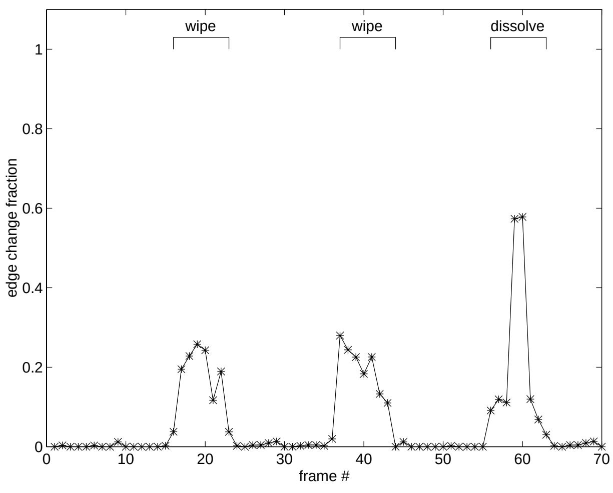


Figure 4: Results from an image sequence with two wipes and a dissolve

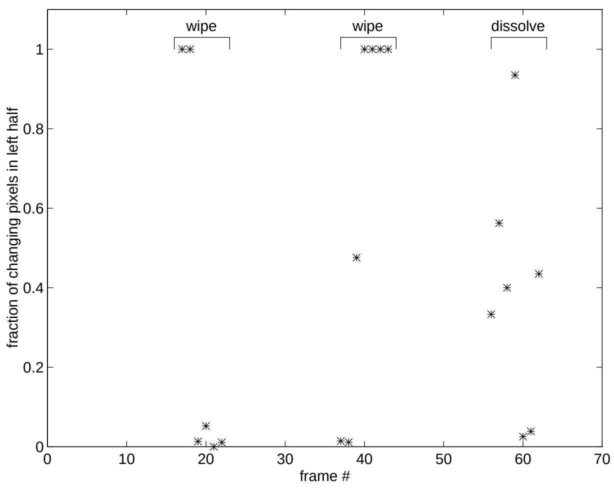


Figure 5: Spatial distribution of change pixels in an image sequence with two wipes and a dissolve (shown only where $\rho > .05$)

wipe there is a vertical strip that passes either left-right or right-left, depending on the direction of the wipe. Since the between-scene transition occurs in this strip, the number of edge pixels that either enter or exit should be higher inside the strip and lower in the other areas of the image. We will call an edge pixel that is either entering or exiting a *changing* pixel.

When computing the edge change fraction, the location of the changing edge pixels can be recorded and their spatial distribution analyzed. There are many ways to analyze the spatial distribution of changing pixels, but we have identified a simple scheme which has yielded good results. We calculate the percentage of such pixels in the top half and the left half of the images, and use this to classify vertical and horizontal wipes. For a left-to-right horizontal wipe, the majority of changing pixels will occur in the left half of the images during the first half of the wipe, then in the right half of the images during the rest of the wipe. Likewise, for a top-to-bottom vertical wipe, the majority of changing pixels will concentrate in the top half, and then in the bottom half. The other two cases (right-to-left and bottom-to-top wipes) can be handled similarly.

Our wipe detection method is aided by the ability of our motion computation follow the predominant motion. This is particularly important during a wipe, since there can be two rather different motions on the image at the same time. Another aid in discriminating wipes from other scene breaks is that there is no pattern in the values of ρ_{in} and ρ_{out} as there was with dissolves and fades. Also the relative differences between ρ_{in} and ρ_{out} will be small, since the changing pixels only occurs a limited strip in the image.

Figure 4 shows the edge change fraction in operation on an image sequence containing a left-to-right wipe, a right-to-left wipe, and a dissolve. Figure 5 shows the proportion of the change pixels that occupy the left half of the image (for clarity, this data is only shown when $\rho > .05$). Note that during the left-to-right dissolve, this fraction drop rapidly from 1 to 0, while during the right-to-left dissolve it rises rapidly from 0 to 1. In addition, the pattern during the dissolve is essentially random, as would be expected.

3.2 The Hausdorff distance

The edge change fraction is related to the Hausdorff distance, which has been used to search for the best match for a model in an image. This distance which has been used for such tasks as recognition and tracking [8]. The Hausdorff distance, which originates in point set topology, is a metric for comparing point sets.

The Hausdorff distance from the point set A to the point set B is defined as

$$h(A,B) \quad = \quad \max_{a\in A} \min_{b\in B} \|a-b\|. \tag{6}$$

Now consider the Hausdorff distance between the edge detected images E and E'. If $h(E',E) \leq r$ then every edge pixel in E' is within r of the closest edge pixel in E, there are no entering edge pixels, and so $\rho_{in} = 0$. Similarly, if $h(E,E') \leq r$ then there are no exiting edge pixels and $\rho_{out} = 0$.

Most applications of the Hausdorff distance use a generalization called the partial Hausdorff distance, which is

$$h_K(A,B) \quad = \quad K^{th}_{a\in A} \min_{b\in B} \|a-b\|. \tag{7}$$

This selects the K^{th} ranked distance from a point in A to its closest point in B. If we select the largest such distance, we have the original Hausdorff distance defined in equation 6.

Applications which use the partial Hausdorff distance for matching [7] can provide a fixed fraction $K/|A|$, which is equal to $1-\rho$. This specifies what fraction of the points in A should be close to their nearest neighbor in B at the best match. Alternatively, a fixed distance can be supplied, and the fraction of points in A within this distance of their nearest neighbor in B can be minimized. We are using a similar measure as the basis for algorithms to detect scene breaks, which is a very different task than matching.

3.3 Algorithm parameters

Our algorithm has several parameters that control its performance. These parameters include:

- the edge detector's smoothing width σ and threshold τ,
- the expansion distance r,

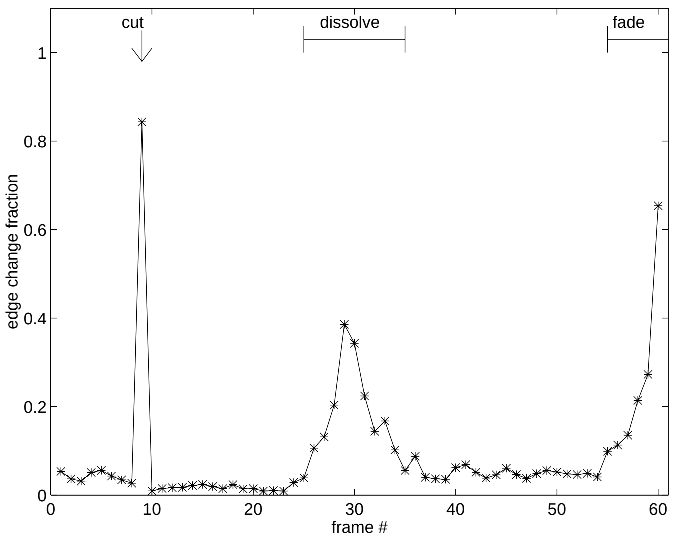
We have gotten good performance from a single set of parameters across all the image sequences we have tested. These parameters are $\sigma = 1.2$ and $\tau = 24$, for the edge detector, and $r = 6$. Except where otherwise noted, these were the parameters used to generate the data shown in this paper. We have found that our algorithm's performance does not depend critically upon the precise values of these parameters (see [15] for evidence of this).

3.4 Compression tolerance

Most video will undergo some form of compression during its existence, and most compression methods are lossy. It is therefore important that our algorithm degrade gracefully in the presence of compression-induced artifacts. While edge detection is affected by lossy compression, especially at high compression ratios, we do not rely on the precise location of edge pixels. We only wish to know if another edge pixel is with r of an edge. As a consequence, the precise location of edge pixels can be changed by image compression without seriously degrading our algorithm's performance. The experimental evidence we present in the next section comes from images that were highly compressed with the lossy MPEG compression scheme.

To demonstrate the compression tolerance of our approach, we have taken an uncompressed image sequence, added a few scene breaks, and compressed it with a variety of different compression ratios. We have used JPEG compression to benchmark the compression tolerance of our algorithm because it introduces similar artifacts to MPEG, but is more standardized. (Note that this is only for the data shown in figure 6 — the data shown in section 4 came from MPEG compressed images.)

Figure 6 shows the results from the table tennis sequence when JPEG-compressed to .18 bits per pixel (with a quality factor of 3). Our algorithm performed correctly even though the compression artifacts were so enormous as to make the sequence almost unviewable. Figure 6 also shows frame #20 at this compression rate.



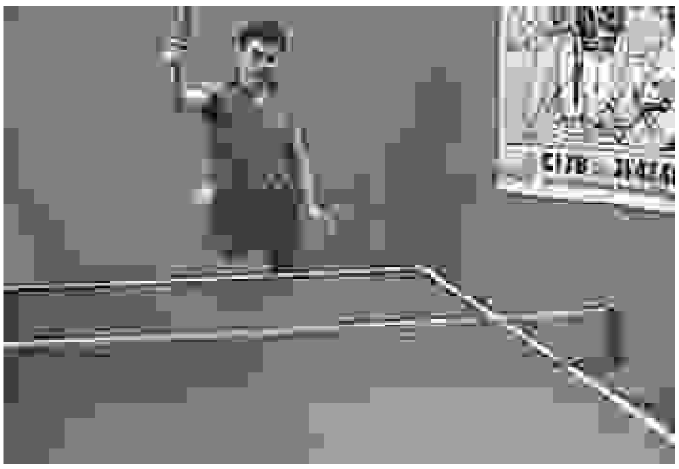


Figure 6: Table tennis sequence results at .18 bits/pixel

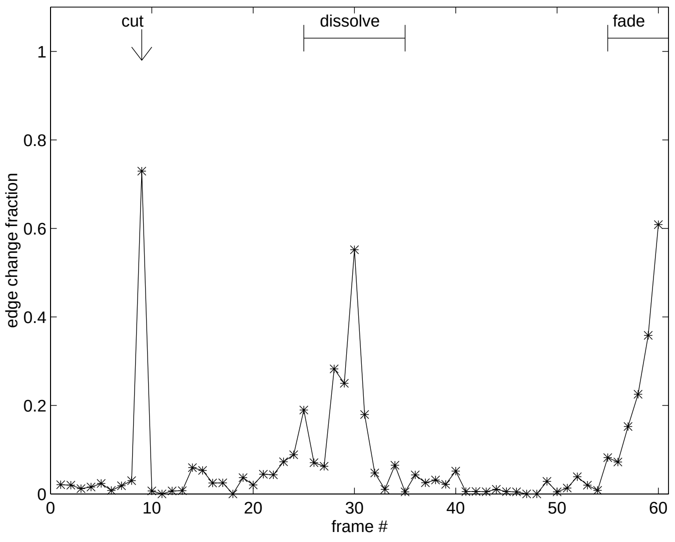


Figure 7: Table tennis sequence with 4:1 subsampling

3.4.1 Subsampling

Our algorithm also performs well when the input images are subsampled to reduced resolution. Figure 7 shows our algorithm's performance on the table tennis sequence when subjected to 4:1 horizontal and vertical subsampling. Note that the output shown in figure 7 is only a little worse than the output that results without subsampling. However the size of the images is reduced by a factor of 16. Depending on how the registration algorithm is implemented, the speedup can be even greater.

4 EXPERIMENTAL RESULTS

We have tested our algorithm on a number of image sequences, containing various scene breaks. To provide a comparison, we have also implemented two other intensity-based measures used to detect scene breaks. The first measure is the intensity histogram difference, which is used with slight variations in most work on scene breaks [10, 11, 17]. The second measure is the chromatic scaling method of Hampapur, Jain and Weymouth [6], a recent method for classifying scene breaks.

There are a number of ways to use intensity histograms. Let N denote the number of histogram buckets (which is typically a power of 2 no greater than 256), and let H_t denote the intensity histogram of the t'th frame. The sum of the histogram differences

$$\sum_{i=0}^{N-1} |H_t[i] - H_{t+1}[i]| \tag{8}$$

is one frequently used measure. Another common measure [10] is the χ^2 value

$$\sum_{i=0}^{N-1} \frac{(H_t[i] - H_{t+1}[i])^2}{H_{t+1}[i]}. \tag{9}$$

We implemented a variant of equation 8 used by Zhang, Kankanhalli and Smoliar. For each of the 3 color channels we used the 2 most significant bits, for a total of $N = 64$ bins in the histogram.

4.1 Sources of data

The image sequences used for testing are MPEG movies. We obtained a number of MPEG encoded movies from `http://www.acm.uiuc.edu/rml/Mpeg/` which include segments from a number of different sources including music videos, television advertisements, documentaries, and NASA recordings. We selected a number of MPEG movies which contained scene breaks.

In addition, we created some additional MPEG movies. Because the MPEG movies we obtained from the network did not contain enough scene breaks to generate significant data, we spliced together scenes from existing MPEG movies and inserted a variety of scene breaks. These spliced movies have several advantages: they show many different scene breaks at known locations, but the video itself was shot and compressed by third parties. Finally, we created one movie, called `andy`, from video which we shot. We inserted several scene breaks during the editing process, and then compressed it using the Berkeley MPEG encoder.

The data we used is highly compressed. The following table summarizes the compression parameters of several of the image sequences we used.

Sequence	Bits per pixel	Dimensions
`clapton`	0.91	160×120
`spacewalk`	0.82	160×120
`andy`	0.35	160×112

All these sequences are color, so the compression ratios (from 24-bit color images) range from 26:1 to 69:1. These high compression ratios probably result from using videos available on the World-Wide Web, which places a premium on compression to minimize bandwidth and storage costs. However, this makes our data set representative of the kind of video that is widely available today.

All of the test sequences shown use the parameter values mentioned above. The chromatic scaling method and the histogram difference, which we show for comparison, involve no parameters. All of these methods are intended to produce distinctive peaks at cuts and dissolves.

4.2 Some comparative results on difficult sequences

The image sequences we have collected fall into three classes. Several image sequences had easy scene breaks, which could be detected by all the methods we tried. For example, there may only be cuts, or there may be minimal motion. Another class of image sequences caused errors for conventional intensity-based methods, but were handled correctly by our feature-based method. Examples include sequences with motion, and especially ones with both motion and dissolves. Finally, certain image sequences yielded incorrect answers, no matter what method we used. Examples include commercials with very rapid changes in lighting and with fast-moving objects passing right in front of the camera.

In our discussion we will concentrate on sequences where some algorithm had difficulty detecting the scene breaks. On the 50 MPEG movies we examined, we did not find an example where our method failed but where intensity-based methods worked.

4.2.1 The Clapton sequence

One MPEG video that we obtained is part of an Eric Clapton music video. It is an interesting sequence because it contains two dissolves, as well as a moving object (the singer). It has been used to benchmark other algorithms (e.g., [6]). Figure 8 shows the performance of several measures on this sequence. Our edge change fraction detects and classifies both dissolves correctly. The image from each dissolve with the highest value of ρ is shown in figure 11 (these are the images that are at the center of the two dissolves according to our detection method described in section 3.1).

The intensity histogram difference, shown in figure 8(b), is a noisier measure on this sequence. It does show a rise during the first dissolve, and it is possible that the dual threshold scheme of [17] would detect this (depending on the exact thresholds used). However, the second dissolve appears to be indistinguishable from the noise. Their method for handling motion would not help here, since the problem is a false negative rather than a false positive.

The chromatic scaling feature of [6] is shown in figure 8(c). As the authors state, their method has difficulty with dissolves involving motion.

4.2.2 The Andy sequence

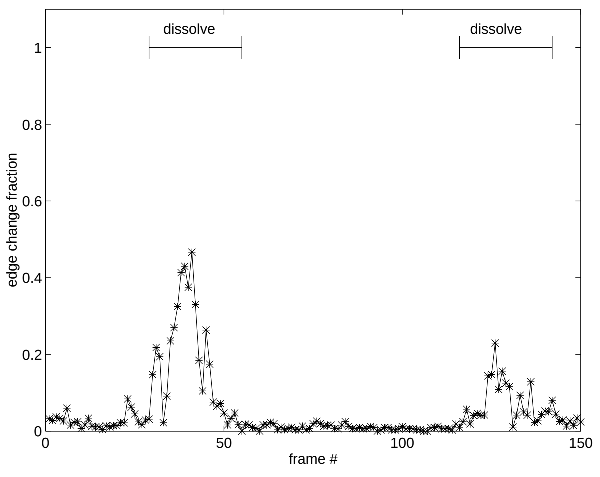
Another sequence that caused some difficulty is the `andy` MPEG. The sequence involves camera and object motion, as well as zooms. It was the most highly compressed MPEG movie that we examined and consists of five scenes separated by three cuts and one dissolve. Frames #1–#50 consist of a stationary scene with the camera panning from right to left. The sequence then cuts to a scene in frames #51–#99 during which the camera zooms in on a stationary background. There is another cut to a scene in frames #100–#133 consisting of a zoom out from a stationary background. The camera stops zooming and continues on the same stationary background for frames #133–#170 and camera remains still on the stationary background. Following the third cut, the sequence contains a scene with a moving person walking from left to right with the camera panning to the right to follow the individual during frames #171–#230. Frames #231–#240 consist of a dissolve between this scene and another in which the camera pans to the right with a stationary background.

Figure 9 presents the results of our method and the intensity histogram difference. The image from the dissolve with the highest value of ρ (frame #235) is shown in figure 10. While we have run the chromatic scaling method on `andy`, it does not produce good results because the sequence includes so much motion.

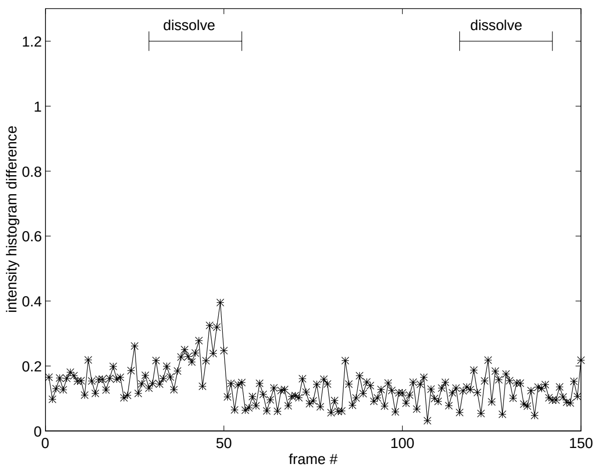
From these results it is not clear whether the histogramming method would find the first dissolve. Depending on how the data is thresholded, either the second cut would be missed or a false cut would be detected at frames #137–#138. These two frames are shown in figure 12.

4.3 Results on our data set

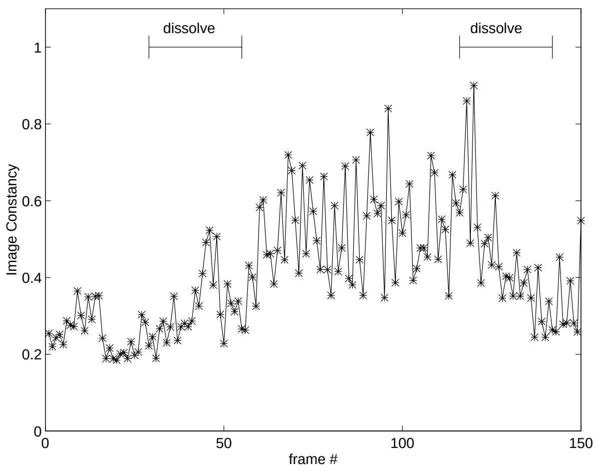
Our algorithm has been tested on an initial dataset of 50 MPEGs obtained from the website mentioned in section 4.1. These MPEGS were decoded and stored as a sequences of gray-level frames. There are 7788 frames with a total of 118 cuts, 8 dissolves, and 3 fades. Our current classification algorithm correctly identifies 115 of 118 cuts, all dissolves and all fades. The algorithm currently identifies 17 false positives, including 14 cuts and 3 fades. The falsely detected fades are all cases where the video becomes very dim and no edges are generated using a constant threshold. Since our current collection of



(a) Our edge change fraction

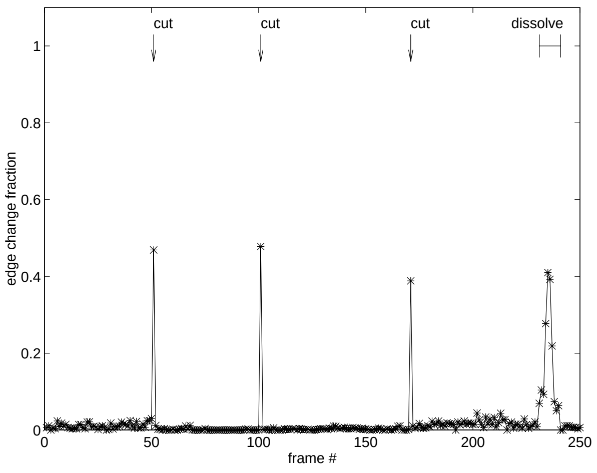


(b) Intensity histogram difference

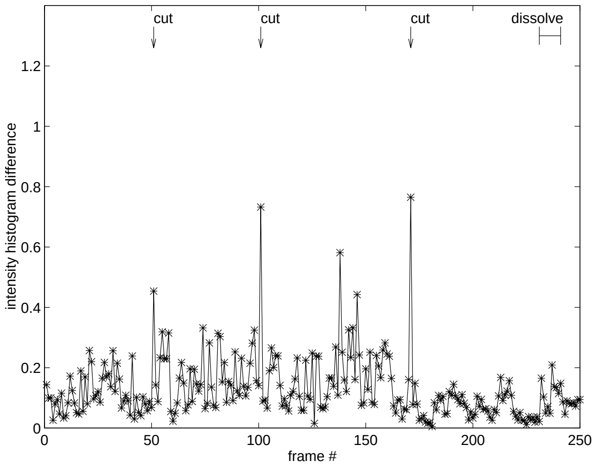


(c) Chromatic scaling feature

Figure 8: Results from the Clapton sequence



(a) Our edge change fraction



(b) Intensity histogram difference

Figure 9: Results from the `andy` sequence

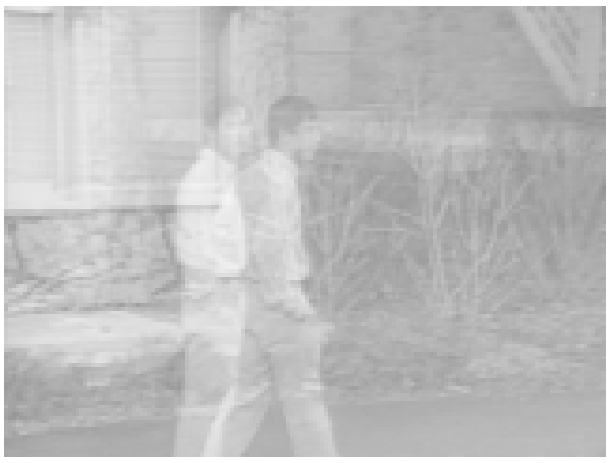


Figure 10: Image from `andy` sequence detected as dissolve by our method

MPEGs does not include any wipes, we are currently developing a test set featuring special effects that we insert ourselves.

4.4 Performance

Our initial implementation of our algorithm is optimized for simplicity rather than for speed. However, its performance is still reasonable. Most of the processing time comes from the global motion computation. We have implemented our algorithm on a Sparc workstation with a single 50-MHz processor, and with 4 50-MHz processors. Because it is so easy to perform the motion compensation in parallel, we have obtained near linear speedups.

The table below shows our implementation's performance with motion compensation disabled, when running on the table tennis sequence.

Image dimensions	1 processor	4 processors
88 × 60	11.03 Hz	44.13 Hz
176 × 120	2.91 Hz	11.63 Hz
352 × 240	.62 Hz	2.48 Hz

The next table shows the performance when using a simple Hausdorff-distance based motion compensation scheme. The running time is linear in the number of disparities considered. Data is shown for a range of disparities which has been adequate for images of these sizes.

Image dimensions	1 processor	4 processors
88 × 60	9.14 Hz	36.57 Hz
176 × 120	1.49 Hz	5.95 Hz
352 × 240	.15 Hz	.6 Hz

The performance on our corpus of MPEG movies was typically around 2 frames per second on a single processor.

A number of methods can be used to reduce the running time on large images, including performing a coarse-to-fine search. A number of methods for improving the performance of the Hausdorff-distance search are described in [13], and have given impressive speedups in practice.

Since the methods we use are fundamentally non-linear, it seems unlikely that we will be able to operate directly on compressed data streams without decompressing. However, our scheme is reasonably fast, and can be optimized further. Our method also appears to give good results on reduced resolution imagery, as shown in figure 7. Finally, much of the overhead of MPEG decompression is due to dithering (for example [12] states that dithering consumed 60% to 80% of the time in their MPEG decoder). Since our approach only uses intensity information, this phase of MPEG decompression can be bypassed.

4.5 Availability

Code for running the algorithm is available via FTP from the host `ftp.cs.cornell.edu` in the directory `/pub/dissolve`. In addition, an HTML version of this document can be found from the URL

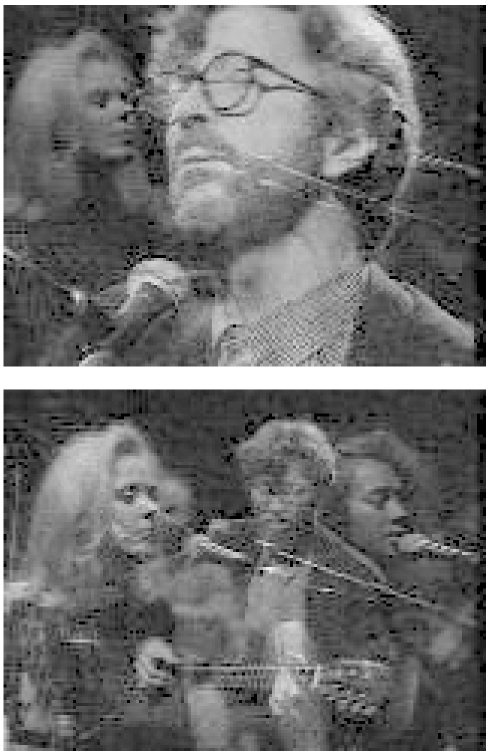


Figure 11: Images from `clapton` sequence detected as dissolves by our method

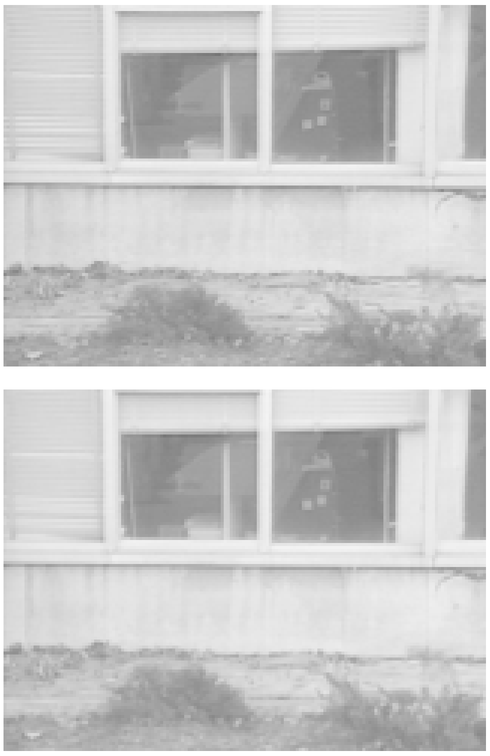


Figure 12: Consecutive images from `andy` sequence with large intensity histogram difference

`http://www.cs.cornell.edu/Info/Projects/csrvl/dissolve.html` along with the code and the image sequences we used.

5 LIMITATIONS AND EXTENSIONS

Our algorithm's failures involve false negatives, and result from two limitations in our current method. First, the edge detection method does not handle rapid changes in overall scene brightness, or scenes which are very dark or very bright. Second, our motion compensation technique does not handle multiple rapidly moving objects particularly well.

The edge detection used in our algorithm has a few limitations at present. For example, rapid changes in overall scene brightness can cause a false positive. Since a thresholded gradient-based edge detector is dependent on the relative contrast of regions in the image, large-scale scalings in image brightness will disturb the edge density of the scene. This effect sometimes occurs in scenes due to camera auto gain.

Scene break detectors based on intensity histogramming will also generate false positives when the overall scene brightness changes dramatically. Although the intensities change dramatically, the underlying edge structure of the image does not change. A more robust edge detection scheme may enable us to handle these events.

In addition, because we use a single global edge threshold τ, some scenes might have no edges, or too many edges. This has not been an issue in most of the videos we have examined, but we are working on ways to address it. One approach is to use an edge detector based on zero-crossings of the Laplacian, rather than on the intensity gradient. We have experimented with a Marr-Hildreth style edge detector [9], but have not yet obtained stable edges on MPEG compressed sequences.

It is also possible to eliminate the edge-detection threshold τ by dynamically thresholding the intensity gradient magnitude. In dynamic thresholding, a constant number of image pixels are labeled as edges. This method is appealing in theory, but has not proved successful in practice. Dynamic thresholding tends to reduce the number of entering and exiting edge pixels. This results in lower values from our similarity measure.

Another improvement, also discussed in [17], involves handling multiple moving objects. The `clapton` sequence contains some motion, while the `andy` sequence contains significant motion (both camera and object motion). As the above data shows, our algorithm handles these sequences well. However, the algorithm's handling of multiple moving objects could probably be improved by compensating for multiple motions.

A number of algorithms have been proposed for this problem, including [1, 4]. When there are two distinct motions in the scene, our motion compensation will track one of them. Edges that undergo the other motion will show up as entering or exiting pixels, assuming that the two motions are sufficiently distinct. We may be able to use these changing pixels to identify objects undergoing a different motion. A solution to this problem would allow users to search a video for the next entrance of an additional moving object.

Another interesting extension involves combining our feature-based algorithm with an intensity-based approach. For example, a conservative intensity-based scheme might be designed which can reliably determine that there are no scene breaks in some portion of a video. Our algorithm might be invoked when the intensity-based scheme indicates a potential scene break. Such a hybrid scheme could be much faster than our method, especially if the intensity-based component operated directly on compressed data.

Conclusions

We have described a new approach to detecting and classifying scene breaks. Our methods robustly tolerate motion, as well as compression artifacts. We are incorporating our algorithm into a browser for MPEG videos which allows the user to search for scene breaks. In the future, we hope to be able to add higher level search capabilities to this browser.

Acknowledgments

We are grateful to Sheila Hemami, Dan Huttenlocher and Brian Smith for suggestions that improved the presentation of this material. We also wish to thank Matt Welsh for providing help with PostScript, and Andy Wand and Robert Szewczyk for helping us generate image sequences.

References

[1] Gilad Adiv. Determining three-dimensional motion and structure from optical flow generated by several moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):384–401, July 1985.

[2] Farshid Arman, Arding Hsu, and Ming-Yee Chiu. Image processing on compressed data for large video databases. In *Multimedia Conference*, pages 267–272. ACM, 1993.

[3] Lisa Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4), December 1992.

[4] Peter Burt, James Bergen, Rajesh Hingorani, R. Kolczynski, W. Lee, A. Leung, J. Lubin, and H. Shvaytser. Object tracking with a moving camera. In *Proceedings of IEEE Workshop on Visual Motion*, pages 2–12, 1989.

[5] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[6] Arun Hampapur, Ramesh Jain, and Terry Weymouth. Production model based digital video segmentation. *Journal of Multimedia Tools and Applications*, 1:1–38, March 1995.

[7] Daniel Huttenlocher and Eric Jaquith. Computing visual correspondence: Incorporating the probability of a false match. In *5th International Conference on Computer Vision, Cambridge, MA*, pages 515–522, 1995.

[8] Daniel Huttenlocher, Greg Klanderman, and William Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[9] David Marr and Ellen Hildreth. Theory of edge detection. *Proc. of the Royal Society of London B*, 207:187–217, 1980.

[10] Akio Nagasaka and Yuzuru Tanaka. Automatic video indexing and full-video search for object appearances. In *2nd Working Conference on Visual Database Systems*, October 1991.

[11] K. Otsuji and Y. Tonomura. Projection-detecting filter for video cut detection. *Multimedia Systems*, 1:205–210, 1994.

[12] L. A. Rowe, K. Patel, and B. C. Smith. Performance of a software MPEG video decoder. In *Multimedia Conference*. ACM, 1993.

[13] William Rucklidge. Efficient computation of the minimum Hausdorff distance for visual recognition. Technical Report TR94-1454, Cornell University Department of Computer Science, September 1994.

[14] William M. Wells, III. Efficient synthesis of Gaussian filters by cascaded uniform filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2):234–239, March 1986.

[15] Ramin Zabih, Justin Miller, and Kevin Mai. Feature-based algorithms for detecting and classifying scene breaks. Technical Report CS-TR-95-1530, Cornell University Computer Science Department, July 1995.

[16] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In Jan-Olof Eklundh, editor, *3rd European Conference on Computer Vision*, number 801 in LNCS, pages 151–158. Springer-Verlag, 1994.

[17] H. Zhang, A. Kankanhalli, and S. Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1:10–28, 1993.