Feature-Based Algorithms for Detecting and Classifying Scene Breaks

Ramin Zabih
Justin Miller
Kevin Mai

Computer Science Department
Cornell University
Ithaca, NY 14853

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 algorithms we propose withstand compression artifacts such as those introduced by JPEG and MPEG, 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.

1 Introduction

One of the greatest problems with forthcoming media technologies is the difficulty of browsing and indexing image sequences. Without some information about the content of video sequences, there are only minimal advantages to having image data available in a digital format. One approach to this problem is for users to "annotate" video sequences by hand with information describing what occurs in them. Unfortunately, this approach is time- and labor-intensive. The obvious alternative is to automatically detect certain events in video sequences. This paper addresses the automatic detection and classification of 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, the images from the scene 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, which is typically constant. 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 approximately the same time. Another common scene break is a wipe, in which a line moves across the screen; the new scene appears behind it, while in front of it the old scene remains.

The detection and classification of scene breaks is thus a first step towards 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. For example, in MPEG, the first image after a cut should be encoded as either an I-frame or a B-frame; encoding it as a P-frame will produce bad output. Automatic keyframing [13] selects a set of still images to represent a video. 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, which relies directly on intensity data. These methods 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 how our algorithm can give good performance under various settings of its parameters, and give evidence of its 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. In addition, there has been a small amount of work on detecting dissolves, which is a much harder problem. All of these approaches have relied directly on intensity data. In section 3 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 thresholded, often in combination with a smoothing mechanism 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 [14] discuss a variety of measures based on image differencing and changes in the image's intensity histogram. Nagasaka and Tanaka [12] 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.

[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.

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 repressed by setting a high threshold. Cuts usually result in a very 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 [7] 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. Their approach to dissolves is novel because 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. They propose to detect dissolves by measuring the uniformity of the chromatic image.

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" [7, page 11], and describe both object and camera motion as causes of false positives for their method. Another recent paper [20] describes motion as a major limitation of histogram-based methods.

A particularly interesting approach has been taken by Zhang, Kankanhalli and Smoliar [20]. 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. They use a local motion computation to compute an optical flow field on an image by image basis, and then test if the flow field could result from a camera pan or zoom.

Their method does not handle false positives that arise from object motion, or more complex camera motions. Nor does their method handle false negatives, which would occur in dissolves between scenes involving motion. In section 3 we will provide an empirical comparison of our method against both histogram-based methods (including Zhang, Kankanhalli and Smoliar's algorithm) as well as Hampapur, Jain and Weymouth's chromatic scaling technique.

1.2 Alternatives to automatic scene break detection

A possible alternative to automatic detection of scene breaks would be for video producers to add such information during the editing process. Scene breaks are a kind of special effect. They are typically added to raw video during post-production, where they are stored in an Edit Decision List (EDL). It is possible that the EDL might be added to the resulting video, which would make it easy to find scene breaks. While this might become a viable approach at some point in the future, there are significant obstacles to be overcome. First, there is an enormous amount of existing video which does not contain EDLs. In addition, new standards would have to be defined in order to encode EDLs in video. These EDLs would need to be encoded in some manner that survived conversion between analog and digital formats, as well as a variety of compression techniques. Since billions of dollars have been invested in hardware that uses current video standards, any new standards would need to be backwards compatible. These factors suggest that encoding EDLs in video is not a viable approach at present.

2 A Feature-Based Approach to Scene Breaks

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. In addition, 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.

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.

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} 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*.

In our experiments, we have used an edge detector based on Canny's algorithm [4]. We smooth the image with a Gaussian of width σ, threshold the gradient magnitude at a value of τ, and perform Canny-style non-maximum suppression. For additional efficiency we perform smoothing by using a small number of box filters, as described in [18].

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 algorithms, 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

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

#55. Every 4th frame from this sequence is shown at the end of this paper in figure 16. On this sequence, ρ shows clear peaks at the scene breaks, and the detection and the classification algorithm described in section 2.2 performed correctly.

2.1 Motion compensation

Our basic algorithm can be extended to handle motion. We first compute the overall motion between images I and I', using some technique for image registration such as those described in [3]. In our experience, it has been sufficient to restrict the motion to be purely translational. While it is easy 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 correlation based on the census transform, a non-parametric approach developed in [19]. This algorithm operates by transforming the image in an outlier-tolerant manner and then using correlation. We have also used the Hausdorff distance [9], an outlier-tolerant method described in section 2.3 that operates on edge-detected images.

Because MPEG has emerged as the dominant cross-platform standard for compressing image sequences, 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 [20] 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]

[3]This is not as unusual a situation as one might imagine. About $\frac{1}{4}$ of the MPEG sequences we have come across on the World-Wide Web are compressed with only I-frames.

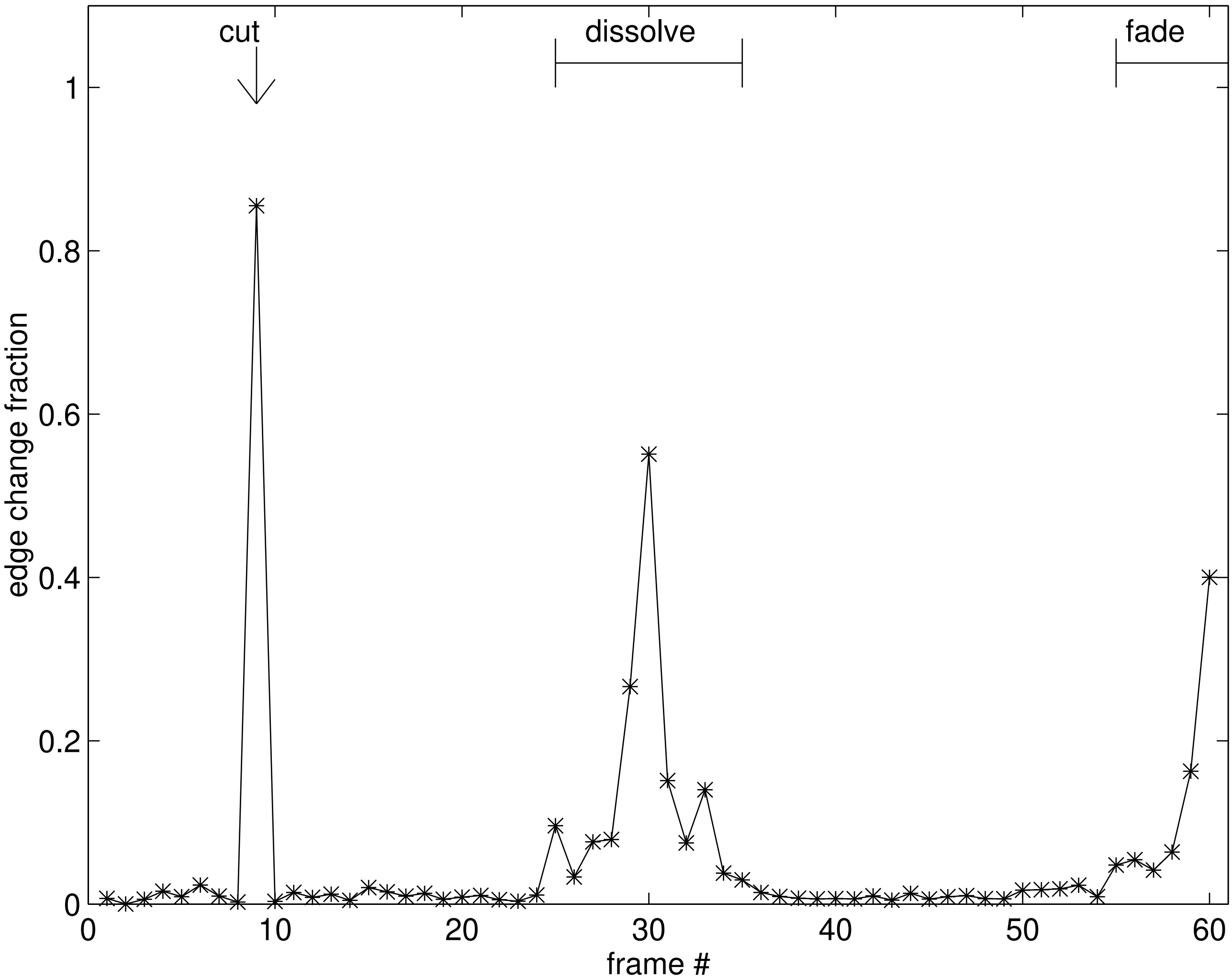


Figure 1: Results from the table tennis sequence

2.2 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.

2.2.1 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 cuts from the 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 frame with a 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 2 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}.

2.2.2 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 wipe there is a vertical strip that passes either left-right or right-left, depending on the direction of the wipe. Since

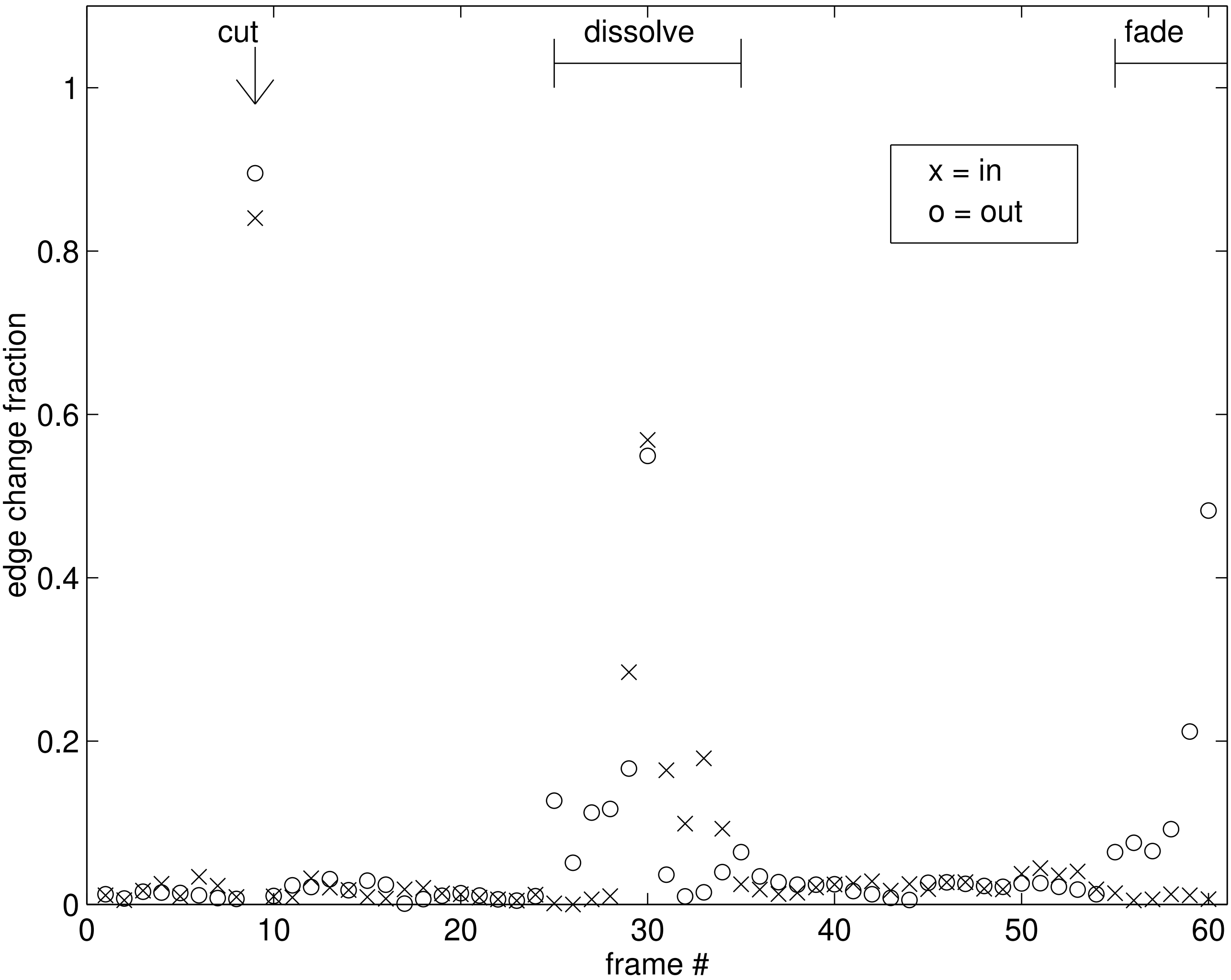


Figure 2: Values of ρ_{in} (shown as "$\times$") and ρ_{out} (shown as "o") in the table tennis sequence

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 3 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 4 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.

2.3 The Hausdorff distance

Our measure 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 [9] and tracking [10]. The Hausdorff distance, which originates in point set topology [5], is a metric for comparing

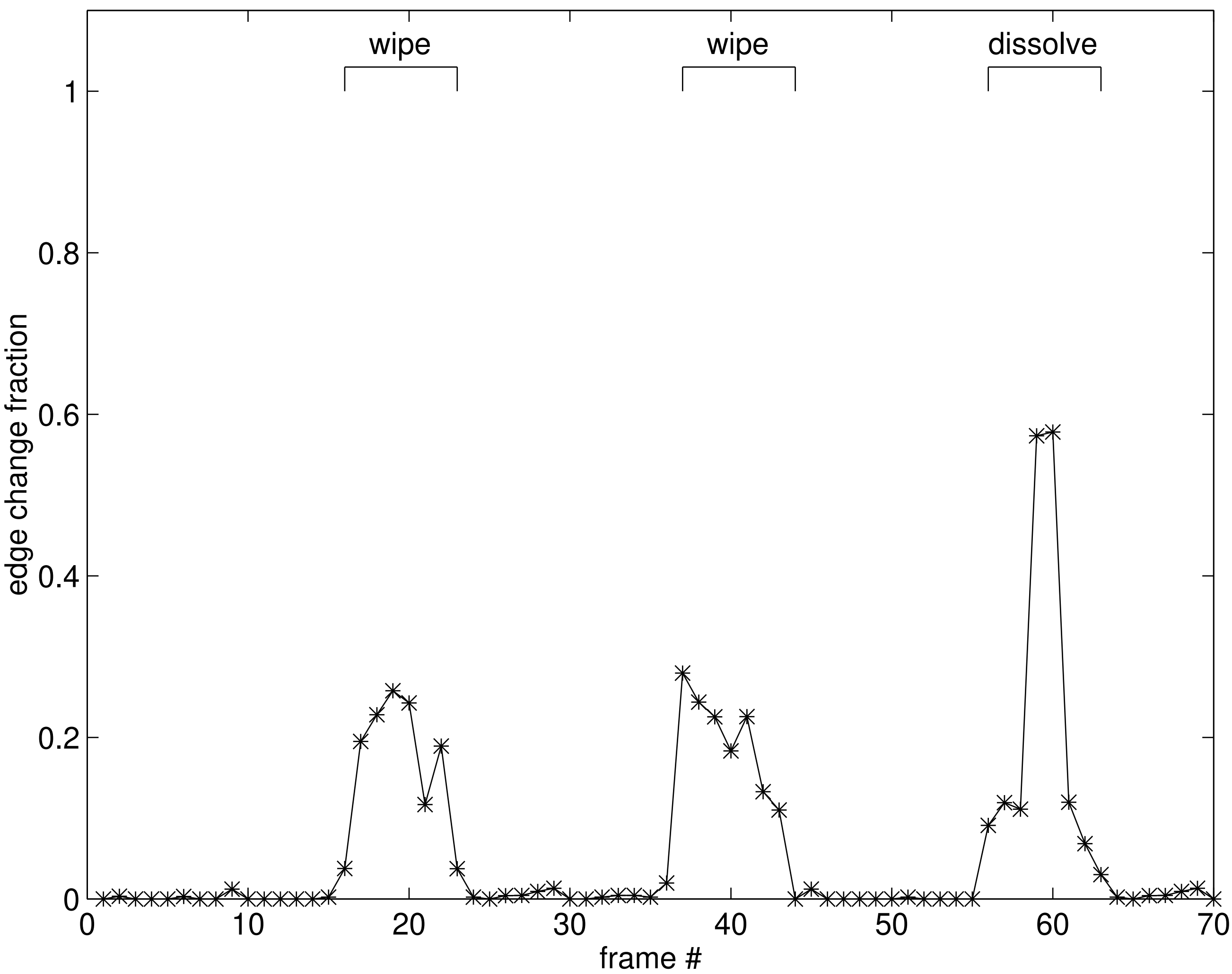


Figure 3: Results from an image sequence with two wipes and a dissolve

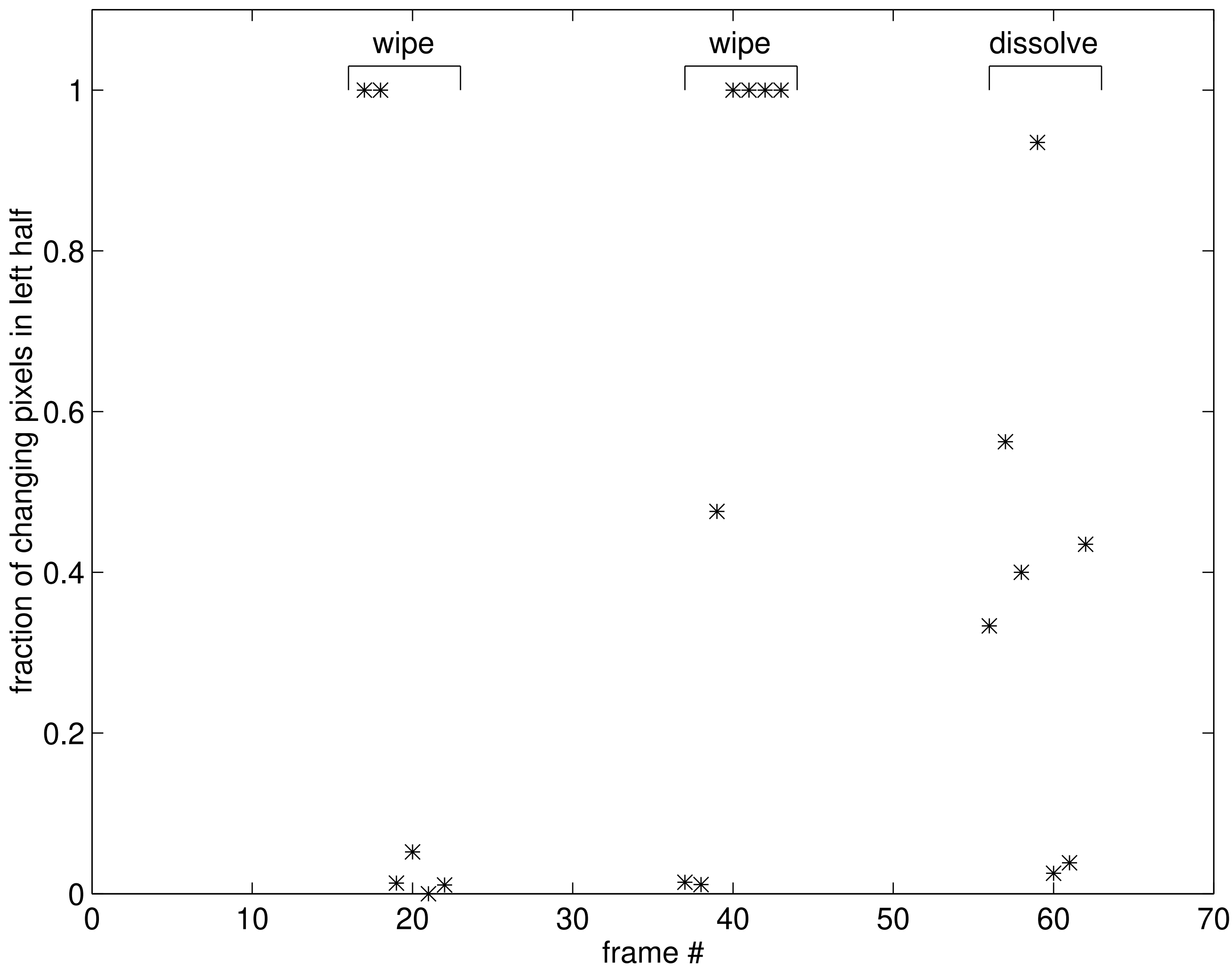


Figure 4: Spatial distribution of change pixels in an image sequence with two wipes and a dissolve (shown only where $\rho > .05$)

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

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|. \tag{2}$$

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

$$h_K(A, B) = K^{th}_{a \in A} \min_{b \in B} \|a - b\|. \tag{3}$$

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 2.

Applications which use the partial Hausdorff distance for matching [8] can provide a fixed fraction $f = K/|A|$, which is $1 - \rho$. 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.

2.4 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.

2.4.1 Robustness

We have found that our algorithm's performance does not depend critically upon the precise values of these parameters. As a demonstration, we have run the table tennis sequence with a variety of different parameters. Figure 5 shows our results with several different values of σ, τ and r.

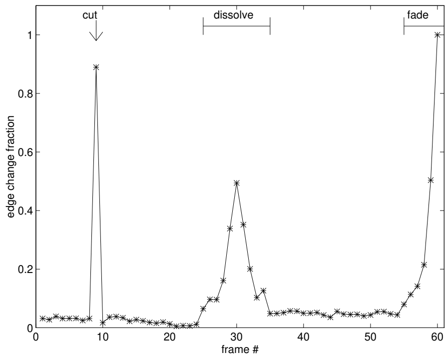
It is, of course, important that the edge detector parameters are set to obtain a reasonable number of edges. This suggests dynamically thresholding the gradient magnitude, an extension we will discuss in section 4. If r is very small the algorithm will also perform badly; small shifts in edges (due to noise, non-rigid motion, zooming or compression artifacts) will cause our detector to register false positives. As r gets larger, the value of ρ when there is no scene break decreases, but so does the value during a scene break.

2.5 Compression tolerance

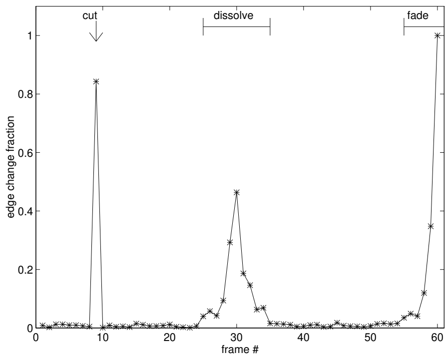
Most video will undergo some form of compression during its existence, and most compression methods are lossy. It is therefore important that our algorithm degrade gracefully in the presence of compression-induced artifacts. While edge detection is affected by lossy compression, especially at high compression 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. Although MPEG is the dominant compression scheme for video sequences, we have chosen to use JPEG compression for this benchmark. There are a number of reasons for this, but they center on the fact that the behavior of an MPEG encoder is not specified by the MPEG standard. The MPEG standard defines what MPEG-compressed streams look like, but leaves compressors considerable latitude in how they produce them. As a result, the quality of MPEG-compressed video depends greatly upon which encoder is used.

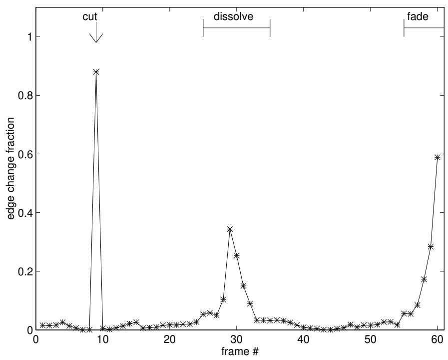
There are a large number of MPEG coders available, with very different



(a) $\sigma = 1$, $\tau = 32$, $r = 3$



(b) $\sigma = 1$, $\tau = 32$, $r = 6$



(c) $\sigma = 1.8$, $\tau = 12$, $r = 6$

Figure 5: Results from varying parameters on the table tennis sequence

schemes for computing motion vectors. In addition, an individual MPEG-encoder will produce different artifacts depending on the pattern of I-, P- and B-frames it uses during encoding. We have used JPEG compression to benchmark the compression tolerance of our algorithms 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 3 came from MPEG compressed images.)

Figure 6 shows the results from the table tennis sequence under different compression ratios. Our algorithm performed correctly even when the compression artifacts were so enormous as to make the sequence almost unviewable. Figure 6 also shows frame #20 at various compression ratios.

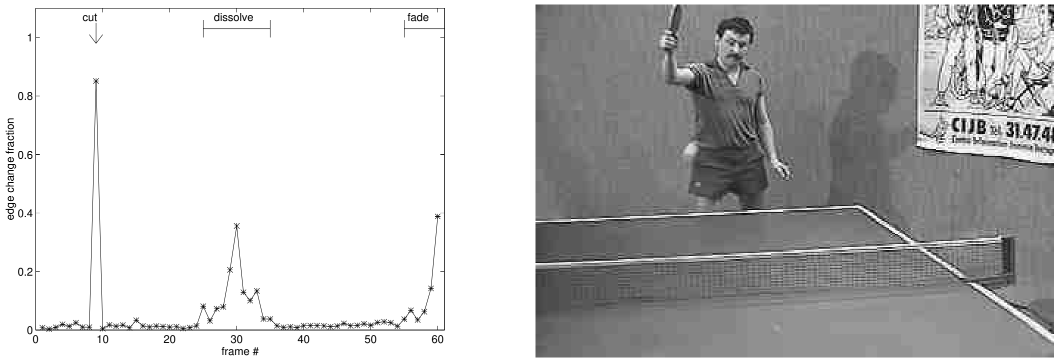
2.5.1 Subsampling

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 subsampling. (We rescaled τ for the subsampled images, to obtain roughly the same fraction of edge pixels, and reduced r as well.) Note that the output shown in figure 7(c) 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, and depending on how the registration algorithm is implemented the speedup in performance can be even greater.[4]

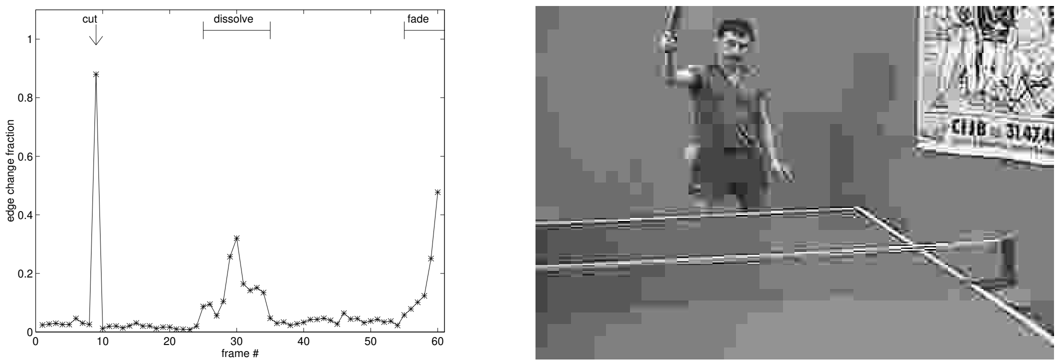
3 Experimental results

We have implemented our algorithm and tested it 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 [12, 14, 20]. The second measure is the chromatic scaling method of Hampapur, Jain and Weymouth [7], a recent method for classifying scene breaks.

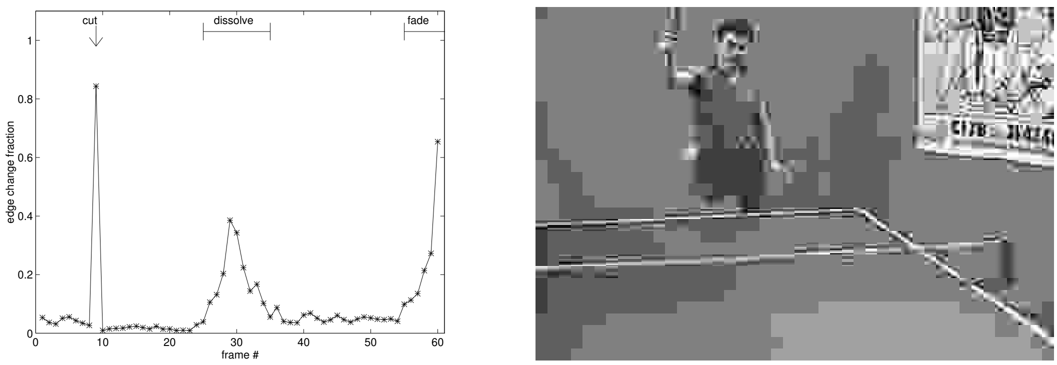
[4] If we use a straightforward search of disparity space, the search window can also be reduced by a factor of 16, leading to a possible 256-fold speedup.



(a) .48 bits per pixel JPEG compression

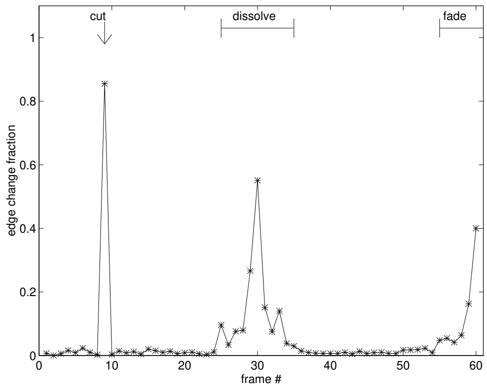


(b) .22 bits per pixel JPEG compression

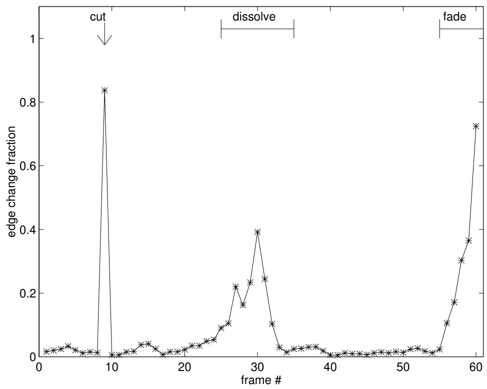


(c) .18 bits per pixel JPEG compression

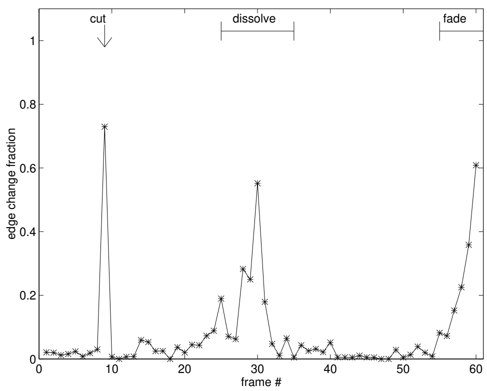
Figure 6: Table tennis sequence results at various compression ratios



(a) No subsampling



(b) 2:1 horizontal and vertical subsampling



(c) 4:1 horizontal and vertical subsampling

Figure 7: Table tennis sequence results with subsampling

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

$$\sum_{i=0}^{N-1} |H_t[i] - H_{t+1}[i]| \tag{4}$$

is one frequently used measure. Another common measure [12] is the χ^2 value

$$\sum_{i=0}^{N-1} \frac{(H_t[i] - H_{t+1}[i])^2}{H_{t+1}[i]}. \tag{5}$$

We implemented a variant of equation 4 used by Zhang, Kankanhalli and Smoliar. It makes use of all 3 color channels with $N = 4$, which produces 64 bins for the histogram.

3.1 Sources of data

The image sequences used for testing are MPEG movies. We obtained a number of MPEG encoded movies from the site

`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 [6].

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

Sequence	Bits per pixel	Megabits per second	Dimensions
`clapton`	0.91	.562129	160×120
`spacewalk`	0.82	.502259	160×120
`andy`	0.35	.187576	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. Before running any scene-break algorithms we manually determined the correct locations of the cuts and dissolves in all the sequences. All of these methods are intended to produce distinctive peaks at cuts and dissolves. Also note that we have used a single, consistent scaling of the y axis for each method that doesn't change between image sequences. For displaying the intensity histogram difference, we have truncated values larger than 1.2, to make the output legible.

3.2 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.

3.2.1 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., [7]). 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 9 (these are the images that are at the center of the two dissolves according to our detection method described in section 2.2).

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 [20] 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.

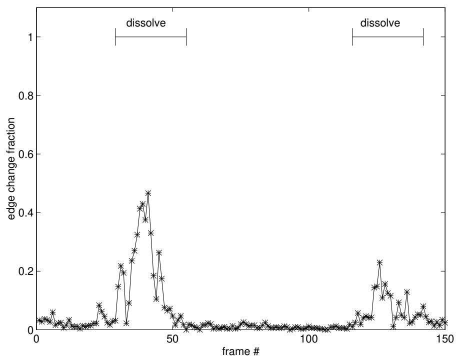
The chromatic scaling feature of [7] is shown in figure 8(c). As the authors state, their method has difficulty with dissolves involving motion.

Figure 10 demonstrates the importance of motion compensation for our results. Even on a scene without camera motion, the motion compensation is important. Without it, the edge change fraction has a larger value at the beginning of the sequence, and before the first dissolve, than it does during the second dissolve. These are due to the motion of the singer's head. Our algorithm would report false positives on this sequence if we did not perform motion compensation.

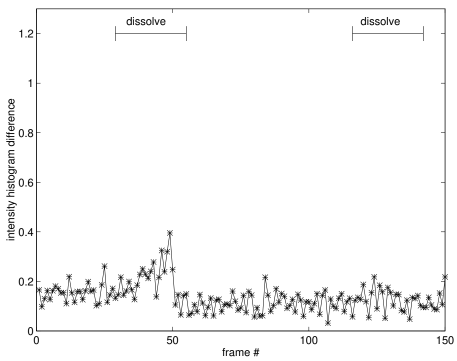
3.2.2 The Andy sequence

Another sequence that caused some difficulty is the `andy` MPEG. There are three scenes in this sequence. They first appear separated by cuts, and then appear again separated by dissolves. The sequence involves both camera and object motion, as well as zooms. It was also the most highly compressed MPEG movie that we examined.

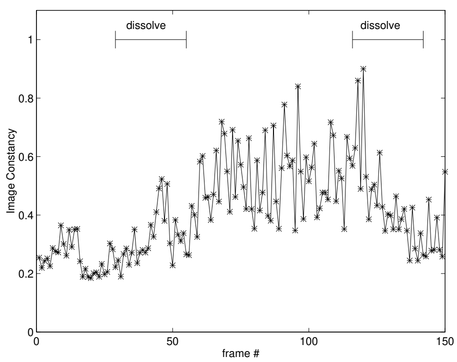
Frames #1–#140 consist of a moving person walking from left to right with the camera panning to the right to follow the individual. There is a cut to the next scene, frames #141–#250, in which the camera pans to the right with only background motion. It cuts to frames #251–#300, consisting of a



(a) Our edge change fraction



(b) Intensity histogram difference



(c) Chromatic scaling feature

Figure 8: Results from the Clapton sequence

Figure 9: Images from `clapton` sequence detected as dissolves by our method

zoom in on a stationary background, and then cuts to frames #301–#370, which consists of a zoom out from another stationary background. This sequence is then repeated, with dissolves replacing the cuts between scenes. These dissolves are located at frames #499–#509, #599–#609, and #639–#649.

Figure 11 presents the results of our method and the intensity histogram difference. From these results it is not clear that the histogramming method would find the first dissolve. Depending on how one thresholds the data, either the second cut would be missed or a false cut would be detected at frame #682. Each of the dissolves contains either camera panning or zooming that might cause a false negative from Zhang, Kankanhalli and Smoliar's approach to motion. Because the sequence includes so much motion, we do not show the chromatic scaling method on `andy`.

3.2.3 The Energizer Bunny sequence

The Energizer Bunny commercial sequence presents an interesting failure of the histogram-based method. The intensity histogram difference has a large peak in the middle of the sequence shown in figure 12. This false peak is a larger response than four of the seventeen real cuts occurring in the sequence. The value of the edge change fraction here is very small (under .012 on these four images).

This sequence also provides a false positive due to a very large object moving quickly near the camera. The images shown in figure 13 cause difficulties

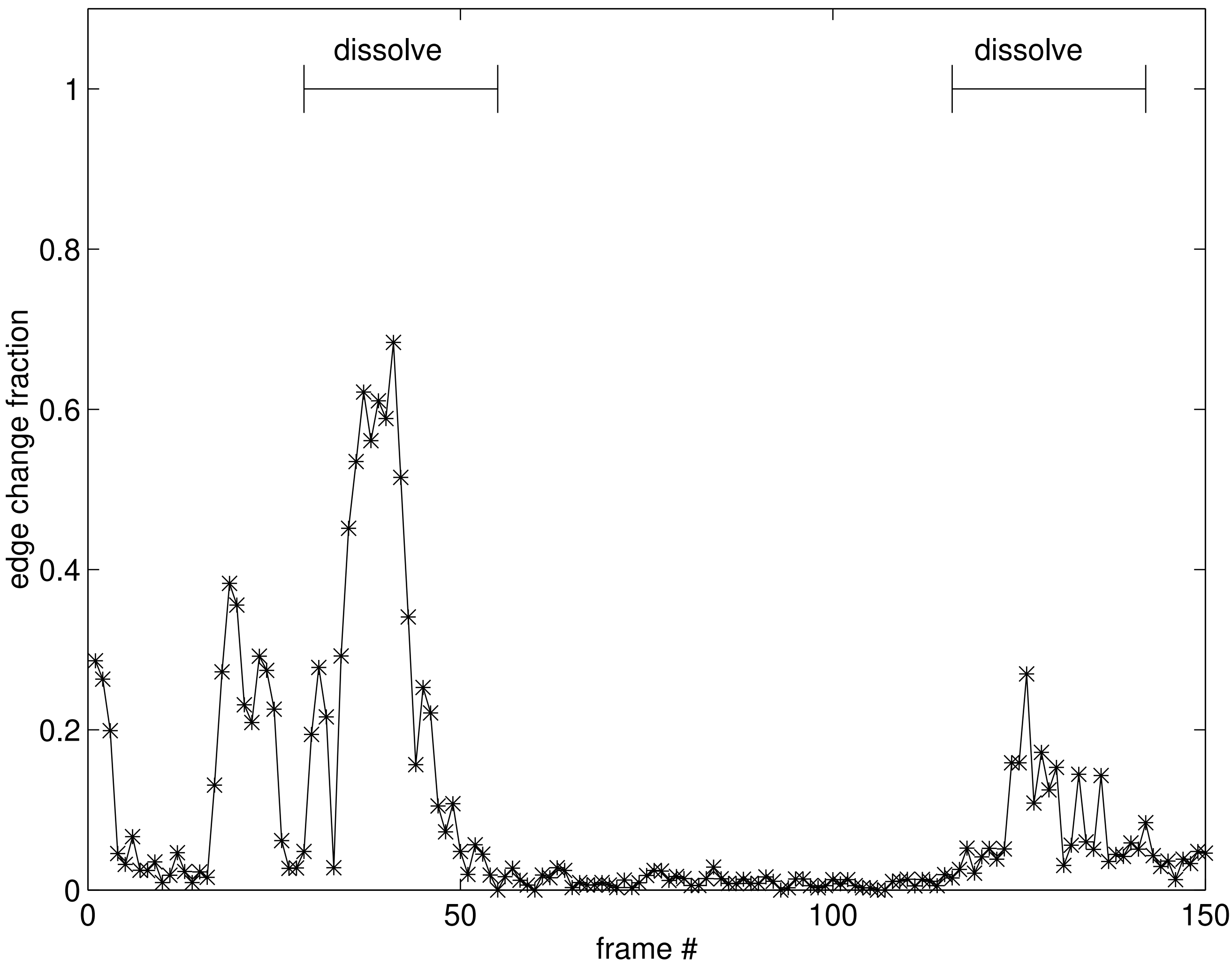
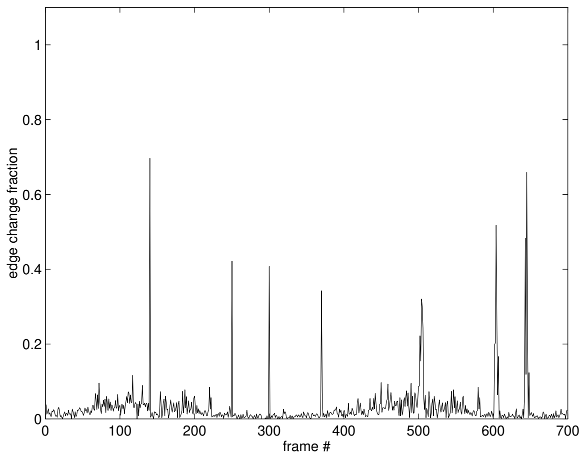
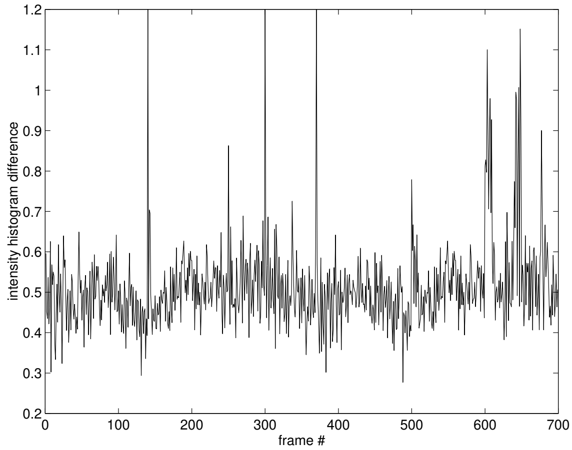


Figure 10: Results from the `clapton` sequence with motion compensation disabled



(a) Our edge change fraction



(b) Intensity histogram difference

Figure 11: Results from the `andy` sequence

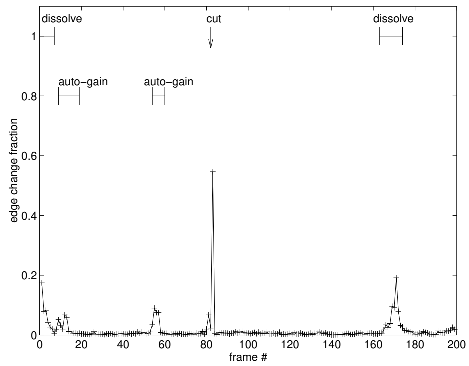
Figure 12: Images from `energizer` sequence detected as a scene break by intensity histogram differencing

Figure 13: Images from `energizer` sequence causing false positives for both methods

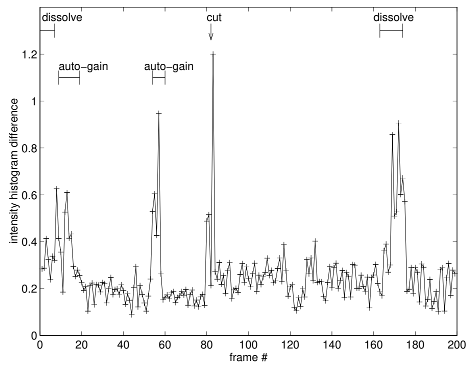
for both the intensity histogram difference method and for our approach. (It is quite hard to distinguish a wipe from a large object moving in front of the camera, especially if the object fills a large enough portion of the scene.)

3.2.4 The spacewalk sequence

The spacewalk sequence illustrates another type of scene which both our method and image histogram methods have difficulty with. The sequence begins with a dissolve occurring in frames #1–#8, a cut in frames #82–#83, and a dissolve in frames #163–#174. Frames #9–#83 is a steady scene during there is a dramatic change in the camera gain, occurring in frames #9–#19 and #54–#60. While an event threshold of ρ around 0.15 would separate the camera gain effects from the real scene breaks, this is too small a value to work on the majority of the MPEGs. Note that the intensity histogram difference misses the first dissolve, and generates a response to the second gain period slightly larger than its response to the second dissolve.



(a) Our edge change fraction



(b) Intensity histogram difference

Figure 14: Results from the `spacewalk` sequence

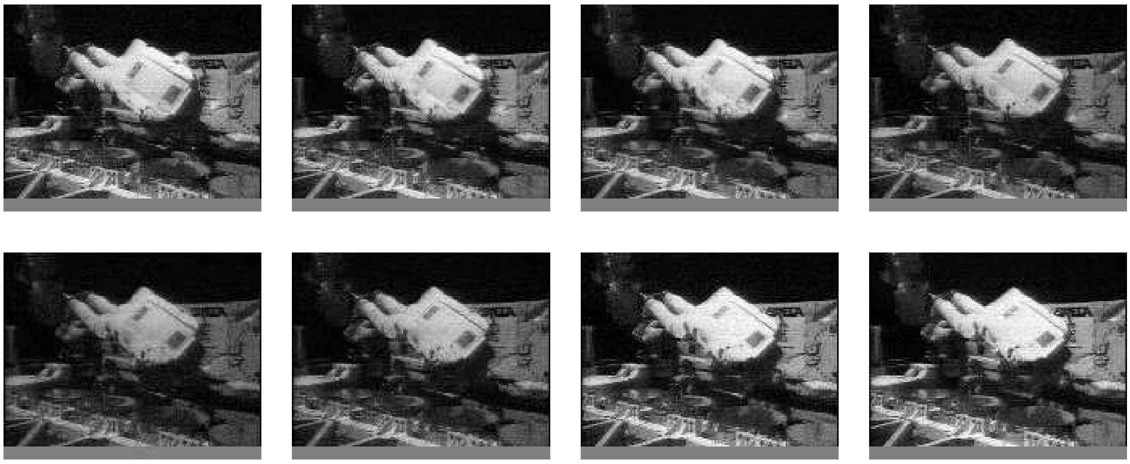


Figure 15: Images #52–#59 from `spacewalk` scene, which generate a false positive due to gain

3.3 Results on our data set

Our algorithm has been tested on an initial dataset of 50 MPEGs obtained from the website mentioned in section 3.1. 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 MPEGs does not include any wipes, we are currently developing a large test set featuring special effects that we insert ourselves.

3.4 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 the motion computation is so parallelizable, 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 (in our experience) is adequate for images of these sizes.

Image dimensions	Disparities	1 processor	4 processors
88×60	25	9.14 Hz	36.57 Hz
176×120	81	1.49 Hz	5.95 Hz
352×240	289	.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 [17], 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 [16] 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.

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

4 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 auto gain, as shown in the spacewalk sequence in figure 15.

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 [11], 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. This method is appealing in theory, but has not proved successful in practice. Dynamic thresholding tends to make consecutive images similar to each other, which results in lower values from our similarity measure.

Another improvement, also discussed in [20], 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, 15].

Conclusions

We have described a new approach to detecting and classifying scene breaks. Our methods robustly tolerate image and camera motion, as well as compression artifacts. Scene break detection is important for a number of applications, including compression, annotation, and storyboarding. We hope that our approach will prove useful for these tasks, and help fulfill some of the promise of digital video.

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

[5] A. Csaszar. *General Topology.* Adam Hilger Ltd., Bristol, 1978.

[6] K. Gong and L. A. Rowe. Parallel MPEG-1 video encoding. In *Proceedings of the Picture Coding Symposium*, 1994.

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

[8] 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.

[9] 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.

[10] Daniel Huttenlocher, Jae Noh, and William Rucklidge. Tracking non-rigid objects in complex scenes. In *Proceedings of the Fourth International Conference on Computer Vision*, pages 93–101, 1993.

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

[12] 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.

[13] B. C. O'Connor. Selecting key frames of moving image documents: A digital environment for analysis and navigation. *Microcomputers for Information Management*, 8(2):119–133, 1991.

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

[15] Shmuel Peleg and Hillel Rom. Motion based segmentation. In *International Conference on Pattern Recognition*, pages 109–113, 1990.

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

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

[18] 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.

[19] 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.

[20] HongJiang Zhang, Atreyi Kankanhalli, and Stephen William Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1:10–28, 1993.

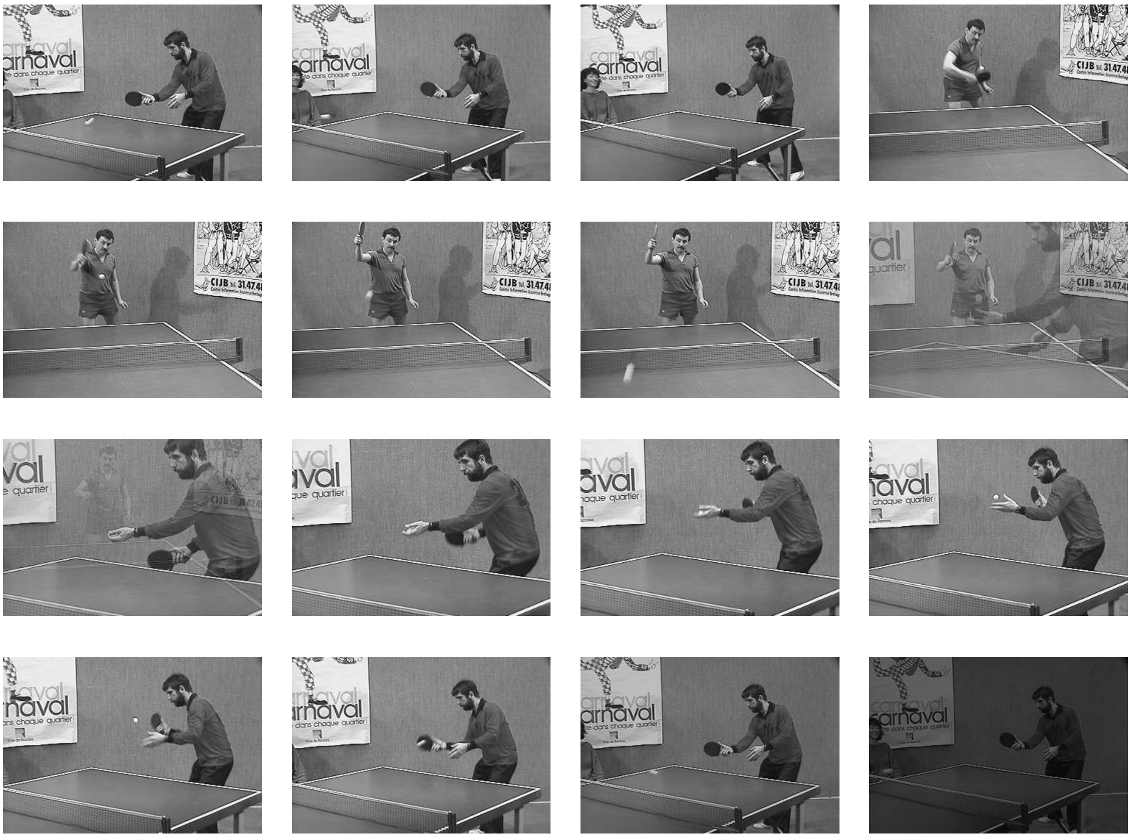


Figure 16: Every 4th frame from the table tennis sequence