Motion–based Parsing of Compressed Video

Suchendra M. Bhandarkar
Department of Computer Science
The University of Georgia
Athens, GA 30602–7404, USA
suchi@cs.uga.edu

Aparna A. Khombhadia
Netscape Communications Corporation
501 E. Middlefield Rd, Bldg 14, MV-093
Mountain View, CA 94043–4042, USA
aparna@netscape.com

Abstract

An algorithm for detecting scene changes in compressed video streams is proposed. The proposed algorithm directly exploits the motion compensation information and the prediction error signal in the MPEG1-coded video stream. By performing minimal decoding of the compressed video stream, the proposed algorithm results in significant savings in terms of execution time and memory requirement. Experimental results show that the algorithm is effective in detecting abrupt scene changes (cuts) as well as gradual scene changes (dissolves) in an MPEG1-coded video stream. The algorithm is capable of processing video frames in real time and could be used for the rapid generation of key frames for real-time browsing of video streams and for indexing to support content-based access to on-line video libraries.

1 Introduction

One of the greatest problems with emerging multimedia technologies is the difficulty of rapidly and reliably extracting "key" information from images, video and audio streams which could then be used for rapid browsing and indexing of the underlying information [16]. In this paper we deal with the problem of extracting key information from video data. Video parsing or scene change detection in a video stream is typically used to extract key frames in a video stream. These key frames are then used for rapid video browsing and automatic annotation and indexing of video streams to support content-based query access to large video databases. The video parsing operation is primarily domain-independent i.e., no assumptions are made about the semantics of the video or its underlying theme. Video parsing, therefore, is a crucial first step that precedes domain-dependent analysis of the video [16].

Due to the large amount of data involved, video streams are often compressed for efficient transmission and storage. Scene detection techniques that are capable of operating on compressed video data directly have a considerable advantage in terms of execution time and memory requirement when compared to those that require full frame decompression. In this paper we propose an algorithm for detecting scene changes in compressed video streams video. The proposed algorithm directly exploits the motion compensation information and the prediction error signal in the MPEG1-coded video stream. The algorithm entails minimal decoding of the compressed video stream, thus resulting in significant savings in terms of execution time and memory requirement.

2 Review of Video Parsing Techniques

Techniques that parse uncompressed video streams typically compute a global measure of difference between successive frames in the video stream and use a global thresholding scheme to identify and localize significant scene changes in the video stream. The difference measures are typically based on gray level sums, gray level histograms color histograms, and gray level or color statistics [7, 15], motion discontinuities [4] or edge composition [14]. In some cases a domain-specific video production model could be used to guide the video parsing [3].

More recent approaches to video parsing deal with the processing of compressed video directly (i.e., without requiring complete decompression). Most techniques use the histograms of the DC images derived from the DCT coefficients of the I frames in an MPEG1-coded video stream [8, 10, 11], the variance of the DC coefficients in the I and P frames [1, 6] or the proportion of macroblocks with valid motion vectors in the P and B frames [5, 17] to detect abrupt scene changes. Yeo and Liu [13] present algorithms for detecting abrupt and gradual scene changes, intrashot variations and flashlight scenes in MPEG1 and MJPEG video streams without requiring full-frame decompression using the DC images extracted from a compressed video stream.

However, none of the aforementioned approaches that deal with compressed video data actually correlate the motion information from successive frames in a compressed video stream. The difference measure used for scene detection relies primarily on chrominance and/or luminance values (or their averages as reflected in the DC coefficient images). Consequently, these approaches are prone to misses when there is little change in background color or luminance between successive video shots and to false positives when there

is a change in background color or luminance or change in ambient lighting within a single shot. We contend that, in such situations, the relative motion of objects between successive frames can be used more effectively to detect scene changes in a compressed video stream.

In this paper, we present a novel and efficient approach to video parsing that exploits motion information in an MPEG1-coded compressed video stream. It attempts to detect a scene change at the current frame by estimating the amount of motion that has occurred since the previous frame. This technique has the following advantages: it does not need to decompress the video stream before detecting scene boundaries, it is completely independent of the semantic content of the video frames and the computation of the difference or dissimilarity measure between successive frames, which is used to detect scene changes, can be done efficiently. In fact, the dissimilarity measure which is based solely on the relative motion of objects between frames, does not entail any floating-point computation. The dissimilarity measure computation involves only integer addition/subtraction which makes it particularly suitable for real-time applications. The motion-based dissimilarity measure proposed and described in this paper could be used in conjunction with chrominance-based and luminance-based dissimilarity measures [10, 13, 17] for more robust scene change detection in compressed video streams.

3 Motion Compensation in MPEG1

The MPEG1 video compression algorithm [2] relies on two basic techniques for compression: block-based motion compensation for the reduction of temporal redundancy and Discrete Cosine Transform(DCT)-based compression for the reduction of spatial redundancy. *Intracoded* (or I) frames in the MPEG1 video stream use only DCT-based compression i.e., without any motion compensation. Motion vectors are defined for each 16×16 pixel region of the image, called a *macroblock*. *Predictive coded* (or P) frames, have macroblocks that are *motion compensated* with respect to the I frame in the immediate past (i.e., causal prediction). Macroblocks coded in this way are said to have a *forward predicted* motion vector (FPMV). *Bidirectionally coded* (or B) frames, have macroblocks that are motion compensated with respect to, either a reference frame in the immediate past, immediate future or both (i.e., noncausal prediction or interpolative coding). Such macroblocks are said to have a *forward predicted* motion vector (FPMV), *backward predicted* motion vector (BPMV) or both, respectively. The difference signal (or prediction error) is further compressed using spatial redundancy reduction techniques based on the DCT. The DCT-coded *prediction error signal* consists of 16×16 pixel macroblocks that are transmitted along with the rest of the spatial information. The motion information is compressed using a variable-length entropy code to achieve maximum compression efficiency.

4 Detecting Scene Changes in MPEG1 Video

Our algorithm is currently designed to detect two important types of scene changes in MPEG1-coded video streams: *cuts* and *dissolves*. A *cut* in a video stream is characterized by an abrupt scene change. At a cut, a typical motion estimator will need to intracode almost all the macroblocks. Thus, by computing the *motion distance* or the extent of motion within each macroblock in the current frame relative to the corresponding macroblock in the previous frame and setting it to ∞ if such correspondence cannot be determined, instantaneous scene breaks or cuts can be detected. The algorithm for detection of scene cuts is detailed in Section 4.1.

A *dissolve* in a video stream is characterized by a gradual scene transition in which the present (outgoing) scene is gradually faded out and the next (incoming) scene gradually fades in. In our algorithm, we detect dissolves by using the prediction error signal coded in the video stream. An absolute error histogram is computed for each frame and the difference between the error histograms of two consecutive frames is used to detect a dissolve in the video stream. In most cases, a dissolve can be detected by observing large values for the error histogram difference in the immediate vicinity of the dissolve. Detection of dissolves is detailed in Section 4.2.

4.1 Algorithm for Detection of Cuts

The MPEG1 standard does not coerce the encoder to compress the video stream into a specific number of P and B frames. It is possible to encounter any of the frame organizations shown in Figs. 1 and 2. Every non-intracoded frame depends on its reference frame(s) for motion compensation. The number of B frames between two P frames or between an I and a P frame may vary. In the extreme case, there may not be any B frames at all (Fig. 2). Also, the number of P frames between two I frames may vary. In the extreme case, the entire MPEG1 video may be intracoded as in Motion JPEG (MJPEG) video. In MJPEG video no motion estimation is employed by the encoder. Since motion estimation results in high compression due to the large temporal redundancy in most video streams, it is rare for a good MPEG1 encoder to intracode the entire video stream. Since the proposed algorithm exploits motion compensation information in an MPEG1-coded video stream, it is not applicable in situations where all frames in the video stream are intracoded. Chrominance and luminance-based techniques would be more appropriate in such situations [10, 13, 17].

Motion estimation in MPEG1 video can be of two types: *Forward Prediction* (FP) specifies the amount of displacement (or motion) of a macroblock with respect to the corresponding macroblock in the *previous* reference frame. Both P and B frames can have forward predicted motion vectors. *Backward Prediction*

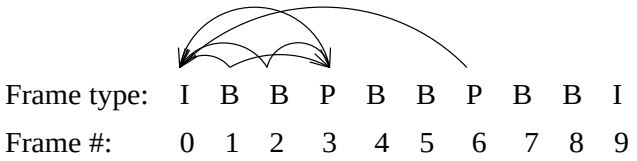

Figure 1: MPEG1 frame organization: A typical case

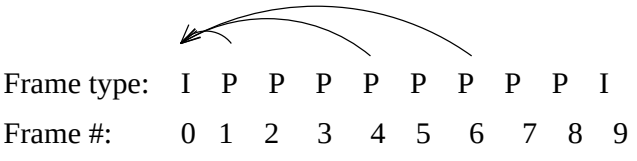

Figure 2: MPEG1 frame organization: without B frames

(BP), on the other hand, specifies the displacement (or motion) of a particular macroblock with the corresponding macroblock in the *following* reference frame. Only B frames can have backward predicted motion vectors.

The proposed algorithm proceeds by comparing each macroblock of every non-intracoded frame with the corresponding macroblock of the previous frame. It attempts to detect a scene break at the current frame by estimating the extent of motion since the last frame. It does so by computing, what we term, the *motion distance (MD)* for each macroblock in the current frame relative to the corresponding macroblock in the previous frame.

4.1.1 Motion Distance Computation: Two Level Indirection

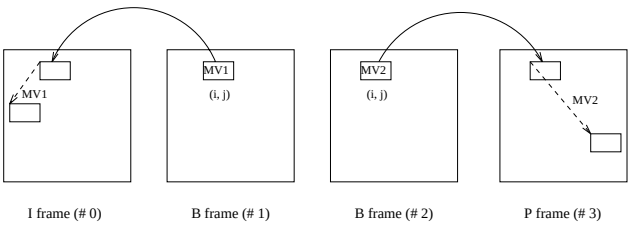

Figure 3: Motion Distance Computation: Possible scenario

The computation of MD is complicated by the fact that the two macroblocks under consideration may have motion vectors (MVs) based on different reference frames. As an example, consider Fig. 3. Let the macroblock under consideration be (i, j). Also let $MV_1 = (u_1, v_1)$ be the FPMV of the $(i, j)^{th}$ macroblock of frame #1, and $MV_2 = (u_2, v_2)$ be the BPMV of the $(i, j)^{th}$ macroblock of frame #2. Here the $(i, j)^{th}$ macroblock of frame #2 has a BPMV based on the $(i, j)^{th}$ macroblock of frame #3. However, the corresponding macroblock in frame #1 has a FPMV based on the $(i, j)^{th}$ macroblock of the reference frame #0. In order to calculate the MD or the motion in frame #2 relative to frame #1, we need to find the MVs for the corresponding macroblocks of adjacent frames based on the *same* reference frame. Thus, we require a *two-level indirection* for the $(i, j)^{th}$ macroblock of frame #2 to obtain its MV based on a different reference frame (i.e., frame #0). This two-level indirection is shown using dotted arrows in Fig. 4. The order of computation is indicated by the number on the arrows. This two-level indirection yields $MV_4 = (u_4, v_4)$, which is the MV of the $(i, j)^{th}$ macroblock of frame #2 based on the reference frame #0. In the above case, the MD can be computed as:

$$MD = \sqrt{(u_4 - u_1)^2 + (v_4 - v_1)^2} \qquad (1)$$

Since the MD computed in equation (1) involves floating point computation, in practice it could be approximated by MD_a the computation of which entails only integer arithmetic:

$$MD_a = |u_4 - u_1| + |v_4 - v_1| \qquad (2)$$

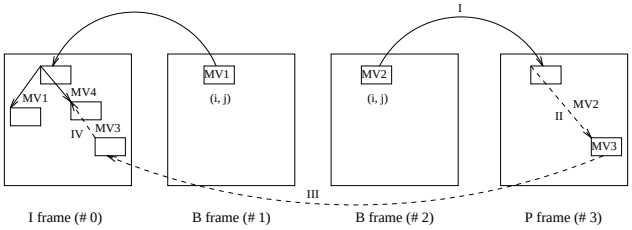

Figure 4: Motion Distance Computation: Two-level indirection

4.1.2 Motion Distance Computation: Some Notation

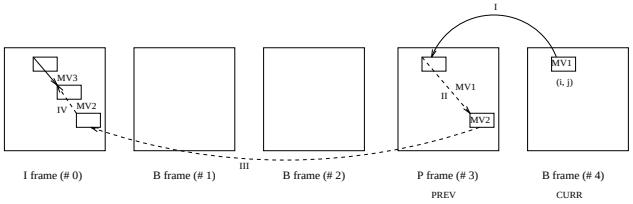

Figure 5: Another Example of Motion Distance Calculation: Two level indirection

Let *Curr* and *Prev* denote the present frame and the immediately preceding frame respectively. With reference to Fig. 5 let $MV_1 = (u_1, v_1)$ and $MV_2 = (u, v)$. Also, let $(u, v) = FPMV_{Curr,FP}(u_1, v_1)$ be defined as

the FPMV (u, v) of the P-type macroblock in the reference frame that the FP of the macroblock of frame *Curr* is dependent on. This notation implies a two-level indirection that is used to obtain an MV based on a different reference frame.

For example, with reference to Fig. 5, let $Curr = 4$, let $Prev = 3$, let the current macroblock under consideration be (i, j) and let $(u, v) = FPMV_{Curr,FP}(23, 68)$. In the above example the macroblock $M(i + 23/16, j + 68/16)$ in frame #3 is first located, which is the reference frame on which *Curr* depends for its FP. The division by 16 arises due to the fact that each macroblock is of size 16×16 pixels. The FPMV of M i.e., $MV_2 = (u, v)$ is then computed. We refer to this process as a two-level indirection since $(u + 23, v + 68)$ is now the equivalent MV for the current macroblock of *Curr* with respect to a different reference frame, which in Fig. 5 is frame #0.

Referring to Fig. 4, if $Prev = 2$ which has only a BPMV, then $MV_3 = (u_3, v_3)$ is the MV obtained by the two-level indirection indicated by the dotted arrows in Fig. 4. This two level indirection is denoted by $(u_3, v_3) = FPMV_{Prev,BP}(u_2, v_2)$. This effectively means that (u_3, v_3) is the FPMV of the reference frame P (frame #3) that the BP of the macroblock of *Prev* is dependent on.

Using the above argument, the proposed algorithm attempts to correlate the MVs of corresponding macroblocks of consecutive frames to the same reference frame, and computes the value of MD.

4.1.3 Motion Distance Computation: Possible Cases

The macroblocks may have a FPMV, a BPMV, both or none. Thus, depending on the type of MVs the macroblocks possess, the algorithm consists of sixteen cases. We describe the computation of motion distance for each case in the remainder of this subsection. The value of MD is used as a *dissimilarity function* for detecting cuts. Unless otherwise mentioned, let the MV of the macroblock in *Curr* be (u_1, v_1) and that in *Prev* be (u_2, v_2). Also, let the macroblock under consideration be (i, j).

Case 1: Both *Curr(i, j)* and *Prev(i, j)* have only FPMVs

In this case, it is possible for *Curr* and *Prev* to be P or B frames. Thus we have the following four subcases:
Subcase A: $Curr = \mathbf{B}$; $Prev = \mathbf{B}$
Subcase B: $Curr = \mathbf{P}$; $Prev = \mathbf{B}$
Subcase C: $Curr = \mathbf{P}$; $Prev = \mathbf{P}$
In the above subcases, since both the frames have their MVs based on the same reference frame, we find the MD between these two macroblocks as,

$$MD_a(i, j) = |u_2 - u_1| + |v_2 - v_1| \quad (3)$$

Subcase D: $Curr = \mathbf{B}$; $Prev = \mathbf{P}$
In this subcase, the motion vector of *Curr* is based on *Prev*, and that of *Prev* is based on a previous I frame (frame #0). We find, $(u_3, v_3) = FPMV_{Curr,FP}(u_1, v_1)$. If the FPMV (u_3, v_3) exists (i.e., if the macroblock in the P frame is not intracoded), then the MD can be determined as:

$$MD_a(i, j) = |(u_1 + u_3) - u_2| + |(v_1 + v_3) - v_2| \quad (4)$$

If (u_3, v_3) does not exist and if $(u_1/16)$ or $(v_1/16)$ is not zero, then $MD_a(i, j) = \infty$ since it is not possible to compute the MD between two macroblocks in consecutive frames without having their MVs based on a common reference frame. This is consistent with previous observations [17] that the number of macroblocks with valid MVs in P or B frames tends to be low when these frames lie on opposite sides of an abrupt scene change. By setting $MD_a = \infty$ in such situations, we account for abrupt scene changes. We compute the quantities $u_1/16$ and $v_1/16$ since we are interested in locating the macroblock corresponding to the motion vector (u_1, v_1). In the MPEG1 standard [2], each macroblock is of size 16×16. Note that if the FPMV (u_3, v_3) does not exist, i.e. the macroblock in the P frame is intracoded, and $(u_1/16)$ and $(v_1/16)$ are both zero, then the current macroblock is the same as that in the previous frame. In this case, $MD_a(i, j) = 0$.

Case 2: *Curr(i, j)* has only a FPMV and *Prev(i, j)* has only a BPMV

Here there are two subcases to discuss.
Subcase A: $Curr = \mathbf{B}$ **and** $Prev = \mathbf{B}$
We find that $(u_3, v_3) = FPMV_{Prev,BP}(u_2, v_2)$. In Fig. 4, if $Prev = 2$, then (u_3, v_3) returned is the motion vector (MV_3) obtained using the two-level indirection indicated by dotted arrows. If (u_3, v_3) exists, i.e. the macroblock $(i + u_2/16, j + v_2/16)$ in the P frame (frame # 3) is not intracoded, then

$$MD_a(i, j) = |(u_2 + u_3) - u_1| + |(v_2 + v_3) - v_1| \quad (5)$$

However, if (u_3, v_3) does not exist, then $MD_a(i, j) = \infty$.
Subcase B: $Curr = \mathbf{B}$ **and** $Prev = \mathbf{B}$
The motion distance is computed in the same way as subcase A. Note, however, that if the forward predicted motion vector (u_3, v_3) does not exist, i.e. the macroblock in the P frame is intracoded, and $(u_2/16)$ and $(v_2/16)$ are both zero, then the current macroblock is the same as that in the previous frame. In this case, $MD_a(i, j) = 0$.

Case 3: *Curr(i, j)* has only a FPMV and *Prev(i, j)* has both FPMVs and BPMVs

Here again, it is not possible for *Prev* to be a P frame. Accordingly we have the following two subcases. Let the BPMV of *Prev* be $Prev_{BP} = (u_3, v_3)$.

Subcase A: *Curr* **= B and** *Prev* **= B**
Subcase B: *Curr* **= P and** *Prev* **= B**
In both the above subcases we determine $(u_4, v_4) = FPMV_{Prev,BP}(u_3, v_3)$. If (u_4, v_4) exists,

$$MD_a(i,j) = \left|\frac{u_2+u_4}{2} - u_1\right| + \left|\frac{v_2+v_4}{2} - v_1\right| \quad (6)$$

else if (u_4, v_4) does not exist, we use an optimistic approximation to calculate motion distance as,

$$MD_a(i,j) = |u_2 - u_1| + |v_2 - v_1| \quad (7)$$

Case 4: *Curr(i, j)* **has a FPMV and** *Prev(i, j)* **has neither a FPMV nor a BPMV**

The only subcase in which MD can be computed is when *Curr* is a B frame, *Prev* is a P frame and $(u_1/16)$ and $(v_1/16)$ are both zero. In this subcase, the MV of *Curr(i, j)* is based on *Prev* and the corresponding macroblock of *Prev* is intracoded. Hence in this subcase, $MD_a(i,j) = 0$ For all other subcases, $MD_a(i,j) = \infty$

Case 5: *Curr(i, j)* **has only a BPMV and** *Prev(i, j)* **has only a FPMV**

Subcase A: *Curr* **= B and** *Prev* **= B**
Subcase B: *Curr* **= B and** *Prev* **= P**
In both subcases, we compute (u_3, v_3) as $(u_3, v_3) = FPMV_{Curr,BP}(u_1, v_1)$. If (u_3, v_3) exists, then

$$MD_a(i,j) = |(u_1+u_3) - u_2| + |(v_1+v_3) - v_2| \quad (8)$$

else $MD_a(i,j) = \infty$.

Case 6: *Curr(i, j)* **has only a BPMV and** *Prev(i, j)* **has only a BPMV**

Only a single subcase is possible; the one when both *Curr* and *Prev* are B frames.

$$MD_a(i,j) = |u_2 - u_1| + |v_2 - v_1| \quad (9)$$

Case 7:*Curr(i, j)* **has only a BPMV and** *Prev(i, j)* **has both, a FPMV and a BPMV**

Here there is only one subcase possible; the one when both *Curr* and *Prev* are B frames. Let $Curr_{BP} = (u_1, v_1)$, $Prev_{FP} = (u_2, v_2)$, $Prev_{BP} = (u_3, v_3)$. We find the motion vectors of both *Curr* and *Prev* in terms of a FP reference frame. $(u_4, v_4) = FPMV_{Curr,BP}(u_1, v_1)$. If (u_4, v_4) does not exist, then

$$MD_a(i,j) = |u_1 - u_3| + |v_1 - v_3| \quad (10)$$

else if (u_4, v_4) exists then we find (u_5, v_5) as $(u_5, v_5) = FPMV_{Prev,BP}(u_3, v_3)$. If (u_5, v_5) does not exist, then,

$$MD_a(i,j) = |u_2 - u_4| + |v_2 - v_4| \quad (11)$$

else if (u_5, v_5) exists, then,

$$MD_a(i,j) = \left|\frac{u_2+u_5}{2} - u_4\right| + \left|\frac{v_2+v_5}{2} - v_4\right| \quad (12)$$

Case 8: *Curr(i, j)* **has both FPMVs and BPMVs and** *Prev(i, j)* **has only a FPMV**

Let $Curr_{FP} = (u_1, v_1)$, $Prev_{FP} = (u_2, v_2)$ and $Curr_{BP} = (u_3, v_3)$.
Subcase A: *Curr* **= B and** *Prev* **= B**
We first find (u_4, v_4) as $(u_4, v_4) = FPMV_{Curr,BP}(u_3, v_3)$. If (u_4, v_4) exists, then

$$MD_a(i,j) = \left|\frac{u_1+u_4}{2} - u_2\right| + \left|\frac{v_1+v_4}{2} - v_2\right| \quad (13)$$

else if (u_4, v_4) does not exist, then

$$MD_a(i,j) = |u_2 - u_1| + |v_2 - v_1| \quad (14)$$

Subcase B: *Curr* **= B and** *Prev* **= P**
MD in this subcase can be computed using the previous two subcases 1(D) and 8(A). Here, we first find the FPMV (u_4, v_4) from (u_3, v_3). Next we find the FPMV (u_5, v_5) from (u_1, v_1) where (u_5, v_5) represents the motion relative to the previous reference frame. MD can be given as:

$$MD_a(i,j) = \left|\frac{u_5+u_4}{2} - u_2\right| + \left|\frac{v_5+v_4}{2} - v_2\right| \quad (15)$$

If (u_4, v_4) cannot be determined, we set $u_4 = v_4 = 0$ and continue finding (u_5, v_5). However, if (u_5, v_5) cannot be determined, and if $(u_1 + u_4)/16 = (v_1 + v_4)/16 = 0$, $MD_a(i,j) = 0$ else $MD_a(i,j) = \infty$.

Case 9: *Curr(i, j)* **has both a FPMV and a BPMV and** *Prev(i, j)* **has only a BPMV**

In this case, the frames *Curr* and *Prev* must be B frames. MD is computed in a similar fashion as in Case 7. Let $Curr_{FP} = (u_1, v_1)$, $Prev_{BP} = (u_2, v_2)$ and $Curr_{BP} = (u_3, v_3)$. Also let, $(u_4, v_4) = FPMV_{Curr,BP}(u_3, v_3)$ and $(u_5, v_5) = FPMV_{Prev,BP}(u_2, v_2)$. We compute MD as,

$$MD_a(i,j) = \left|\frac{u_1+u_4}{2} - u_5\right| + \left|\frac{v_1+v_4}{2} - v_5\right| \quad (16)$$

Case 10: *Curr(i, j)* **has both a FPMV and a BPMV and so does** *Prev(i, j)*

Let $Curr_{FP} = (u_1, v_1)$, $Curr_{BP} = (u_2, v_2)$, $Prev_{FP} = (u_3, v_3)$ and $Prev_{BP} = (u_4, v_4)$. We first find $(u_5, v_5) = FPMV_{Curr,BP}(u_2, v_2)$. If (u_5, v_5) does not exist, we set $u_5 = v_5 = 0$. Next we find, $(u_6, v_6) = FPMV_{Prev,BP}(u_4, v_4)$. If (u_6, v_6) does not exist, set $u_6 = v_6 = 0$. MD can now be computed as,

$$MD_a(i,j) = \left|\frac{u_1+u_5}{2} - \frac{u_3+u_6}{2}\right| + \left|\frac{v_1+v_5}{2} - \frac{v_3+v_6}{2}\right| \quad (17)$$

Case 11: *Curr(i, j)* **has both a FPMV and a BPMV and** *Prev(i, j)* **has neither**

If *Curr* is a B frame, *Prev* is a P frame and $(u_1/16) = (v_1/16) = 0$, then using optimistic approximation, $MD_a(i,j) = 0$ else, $MD_a(i,j) = \infty$.

Case 12: *Curr(i, j)* **has neither a FPMV nor a BPMV and** *Prev(i, j)* **has only a BPMV**

Case 13: *Curr(i, j)* **has neither a FPMV nor a BPMV and** *Prev(i, j)* **has both**

In both, cases 12 and 13, correlation between frames is possible only when *Curr* = P and *Prev* = B and $(u_2/16) = (v_2/16) = 0$. In this case $MD_a(i,j) = 0$. Else, it is not possible to compute the motion between the macroblocks and, $MD_a(i,j) = \infty$.

Case 14: *Curr(i, j)* **has only a BPMV,** *Prev(i, j)* **has neither a FPMV nor BPMV**

Case 15:*Curr(i, j)* **has neither a FPMV nor a BPMV and** *Prev(i, j)* **has only a FPMV**

Case 16: *Curr(i, j)* **has neither a FPMV nor a BPMV and** *Prev(i, j)* **likewise**

In cases 14, 15 and 16, it is not possible to compute the relative motion between the macroblocks of the two frames. Thus $MD_a(i,j) = \infty$.

Figure 6: Table Tennis: Frame# 89

4.1.4 Scene Cut Detection and Classification

In our experiments, we compute the MD for each macroblock in a non-intracoded frame using the above algorithm. We used the approximate distance measure MD_a (eqn. (2)) instead of MD (eqn. (1)) in our experiments. The famous video sequence *Table tennis* shows a scene cut at frame #90 (Figs. 6 and 7). In order to detect cuts, we calculate the *motion edge* magnitude for each frame which is defined as the sum of MDs of all the macroblocks in the frame. We detect

Figure 7: Table Tennis: Frame# 90

scene cuts by thresholding the peaks in the plot of the motion edge magnitude versus frame number (Fig. 8).

4.2 Algorithm for Detection of Dissolves

Most videos often contain special editing effects like *fades* and *dissolves*. 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) [14]. A *dissolve* is a gradual transition from one scene to another, in which the first scene fades out and the second scene fades with the fade-out and fade-in begining at approximately the same time. The gradual nature of a dissolve is what makes it difficult to detect using motion vectors alone. Instead, we use the prediction error information encoded in the MPEG1 video stream to detect dissolves.

We observed that during a typical fade-in, fade-out or dissolve operation the total prediction error (the sum of the absolute values of the prediction error for all the macroblocks) is high and the macroblocks with high values of prediction error are spatially distributed in a random manner throughout the frame(s) that comprise the fade-in, fade-out or dissolve operation. The absolute error for intracoded macroblocks is zero. The proposed algorithm detects dissolves by computing the histogram of the absolute error values for each frame. The difference between the error histograms of two consecutive frames is used to detect a dissolve in the video stream. The difference between two histograms H_1 and H_2 is defined as $\sum_{i=1}^{B} |H_1[i] - H_2[i]|$ where B is the number of bins in each histogram. We generate a *histogram difference diagram* [10] by computing the difference between the histograms of two consecutive frames throughout the video sequence and plotting them in order of the frame number. The dissolves can be detected by thresholding the peaks in the histogram difference diagram. This approach is based on the rationale that within a shot the total prediction error for each frame is small and the prediction error histogram for each frame shows a large percentage of macroblocks to have small values of prediction error. In the vicinity of the dissolve, not only is the total pre-

diction error for each frame large, but the prediction error histogram for each frame shows a large percentage of macroblocks to have large values of prediction error. This shift in the prediction error histogram is captured in the histogram difference diagram.

5 Experimental Results

We used 30 MPEG1-coded video streams in our experiments. Each video had approximately 150-200 frames. The video sources included music, science fiction and sports video sequences. The algorithm was tested on a dataset of 30 MPEG1 files including *Table tennis*, *Energizer*, *Spacewalk*, *Clapton* etc. most of which are used as benchmarks for testing video parsing algorithms. There were over 4,500 frames in all with a total of 51 cuts and 10 dissolves. The algorithm identified 48 of the 51 cuts and all of the dissolves. The motion edge plot to detect cuts in the *Table tennis* sequence is shown in Fig. 8. In this sequence, there are 2 cuts: frames 90 and 149. Both were accurately detected using our algorithm. Dissolves are detected by thresholding the peaks in the histogram difference diagram. For example, the *Clapton* video sequence has one dissolve which is successfully detected by our algorithm (Figs. 9 and 10).

The fact that our algorithm can analyze compressed video streams directly gives it a significant advantage in terms of processing time and memory. Our algorithm can analyze video streams at rates of 44 frames/sec on a SUN SPARC5 workstation. It is is much faster than the edge-based scene change detection algorithm proposed by Zabih *et al.* [14] (3–5 frames/sec) which deals with decompressed video. It is also faster than the algorithms proposed by Yeo and Liu [13] and Shen and Delp [10] (24–28 frames/sec) which uses the luminance and chrominance values of the DC images derived from MPEG1-coded video data. The memory requirement of our algorithm is an order of magnitude lower (1/8 – 1/10) than the algorithm in [14] and comparable to the algorithm in [10]. The above performance statistics were obtained by implementation of the algorithms described in [10] and [14] on the same SUN SPARC5 workstation and comparison of run-time results.

Our algorithm is robust to changes in background or ambient lighting as compared to the algorithm in [10]. In summary, our algorithm performs better or as well as existing approaches in terms of scene change detection accuracy (i.e., misses and false alarms) while delivering real-time performance (44 frames/sec) with low memory requirement. A limitation of our algorithm is that it relies on motion compensation information in the compressed video making it unsuitable for MJPEG video data wherein all the frames are intracoded. Our motion dissimilarity-based approach complements luminance-based and chrominance-based approaches for analysis of MPEG1-coded data and could be used in conjunction with these approaches for more efficient and robust parsing of compressed video data.

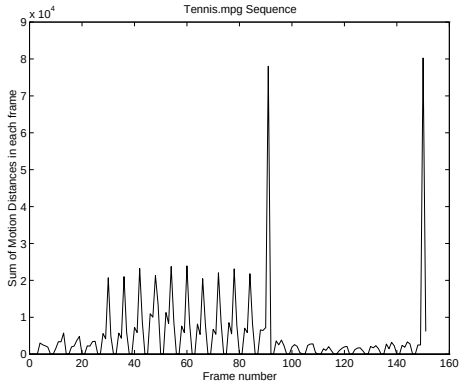

Figure 8: Plot of Motion Edges for *Table tennis* sequence

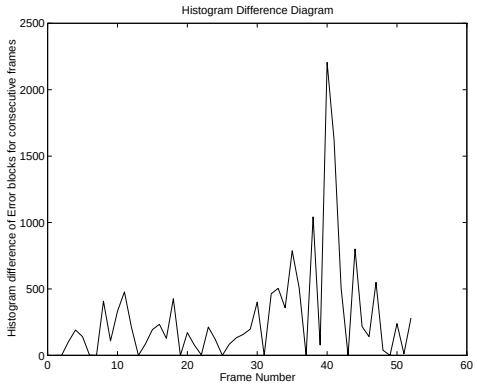

Figure 9: Histogram Difference Diagram for *Clapton* sequence

6 Conclusions and Future Work

The rapid progress in multimedia information systems and digital libraries has generated a great deal of interest in content-based indexing, content-based retrieval, browsing and interactive navigation of video streams. Video parsing is a fundamental operation in several digital video applications. The accuracy and execution speed of the video parsing algorithm is critical for real-time applications that involve the processing and analysis of video data. In this paper we have described a scheme for detecting scene transitions in a MPEG1-coded compressed video stream. Abrupt scene changes (cuts) and gradual scene transitions (fades and dissolves) are detected in a domain-independent manner to automatically segment a compressed video stream. Faster execution time and reduced memory requirement were achieved by using compressed MPEG1 video. The technique proposed and described in the paper has the following advantages: it does not need to decompress the video

Figure 10: Video frame from *Clapton* sequence detected as dissolve

stream before detecting scene boundaries, it is completely independent of the semantic content of the video frames and the computation of the difference or dissimilarity measure between successive frames, which is used to detect scene changes, can be done very efficiently. The dissimilarity measure which is based solely on the relative motion of objects between frames, does not entail any floating-point computation and involves only integer addition/subtraction making it particularly suitable for real-time applications. The dissimilarity measure however, is limited by the fact that it requires that motion compensation be used in the video compression process and is therefore unsuitable in cases where all the frames in the video stream are strictly intracoded. The motion-based approach complements chrominance-based and luminance-based approaches to the parsing of compressed video streams. The motion-based approach in conjunction with chrominance-based and luminance-based approaches would result in more robust and efficient algorithms for parsing of compressed video streams. The work in this paper could be further enhanced to detect zooms, wipes and pans using motion information in the compressed video stream. The key frames (denoting scene changes and special cinematographic effects) could be used to index into a video database and also provide for interactive browsing, navigation and content-based query access to the video database.

References

[1] H. Ching, H. Liu, and G. Zick, Scene decomposition of MPEG compressed video, *Proc. SPIE Conf. Dig. Video Comp: Alg. and Tech.*, San Jose, CA, Vol. 2419, Feb. 1995, pp. 26–37.

[2] D.L. Gall, MPEG: A video compression standard for multimedia applications, *Comm. ACM*, Vol. 34, No. 4, April 1991, pp. 46–58.

[3] A. Hampapur, R. Jain and T.Weymouth, Production model–based digital video segmentation, *Jour. Multimedia Tools and Appl.*, Vol. 1, March 1995, pp. 1–38.

[4] P.R. Hsu and H. Harashima, Detecting scene changes and activities in video databases, *Proc. IEEE Intl. Conf. Acoustics, Speech, Signal Process.* Vol. 5, April 1994, pp. 33–36.

[5] V. Kobla, D. Doermann, K.I. Lin and C. Faloutsos, Compressed domain video indexing techniques using DCT and motion vector information in MPEG video, *Proc. SPIE Conf. Storage and Retrieval for Image and Video Database V*, Vol. 3022, Feb. 1997, pp. 200–211.

[6] J. Meng, Y. Juan, and S.F Chang, Scene change detection in a MPEG compressed video sequence, *Proc. SPIE Conf. Dig. Video Comp.: Alg. and Tech.*, San Jose, CA, Vol. 2419, Feb. 1995, pp. 14–25.

[7] K. Otsuji and Y. Tonomura, Projection detecting filter for video cut detection, *Proc. ACM Multimedia*, Aug. 1993, pp. 251–257.

[8] N.V. Patel and I.K. Sethi, Video shot detection and characterization for video databases, *Pattern Recog.*, Vol. 30, No. 4, 1997, pp. 583–592.

[9] L.A. Rowe, K. Patel and B.C Smith, Performance of a software MPEG video decoder, *Proc. ACM Multimedia*, Vol. 34, No. 4, April 1991, pp. 46–58.

[10] K. Shen and E. Delp, A fast algorithm for video parsing using MPEG compressed sequence, *Proc. IEEE Intl. Conf. Image Process.*, Washington, DC, Oct. 23–26, 1995, pp. 252–255.

[11] B. Shen and I.K. Sethi, Block-based manipulations on transform–compressed images and videos, *Multimedia Sys.*, Vol. 6, No. 2, 1998, pp. 113–124.

[12] G.K. Wallace, The JPEG still picture compression standard, *Comm. ACM*, Vol. 34, No. 4, April 1991, pp. 30–44.

[13] B.L. Yeo and B. Liu, Rapid scene analysis on compressed video, *IEEE Trans. Circuits and Sys. for Video Tech.*, Vol. 5, No. 6, Dec. 1995, pp. 533–544.

[14] R. Zabih, J. Miller, and K. Mai, Video browsing using edges and motion, *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, San Francisco, CA, June 18–20, 1996, pp. 439–446.

[15] H.J. Zhang, A. Kankanhalli, and S.W. Smoliar, Automatic partitioning of full-motion video, *Multimedia Sys.*, Vol. 1, No. 1, 1993, pp. 10–28.

[16] H.J. Zhang and S.W. Smoliar, Content–based video indexing and retrieval, *IEEE Multimedia*, Vol. 1, No. 2, Summer 1994, pp. 62–72.

[17] H.J. Zhang, C.Y. Low, and S.W. Smoliar, Video parsing and browsing using compressed data, *Jour. Multimedia Tools and Appl.*, Vol. 1, No. 1, March 1995, pp. 89–111.