Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Information Systems 31 (2006) 638–658

www.elsevier.com/locate/infosys

# A motion-based scene tree for browsing and retrieval of compressed videos

Haoran Yi, Deepu Rajan, Liang-Tien Chia*

*Center for Multimedia & Network Technology, School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore*

Received 15 March 2005; received in revised form 12 July 2005; accepted 1 October 2005

Recommended by Mei-Ling Shyu, Shu-Ching Chen

**Abstract**

This paper describes a fully automatic content-based approach for browsing and retrieval of MPEG-2 compressed video. The first step of the approach is the detection of shot boundaries based on motion vectors available from the compressed video stream. The next step involves the construction of a scene tree from the shots obtained earlier. The scene tree is shown to capture some semantic information as well as to provide a construct for hierarchical browsing of compressed videos. Finally, we build a new model for video similarity based on global as well as local motion associated with each node in the scene tree. To this end, we propose new approaches to camera motion and object motion estimation. The experimental results demonstrate that the integration of the above techniques results in an efficient framework for browsing and searching large video databases.
© 2006 Elsevier B.V. All rights reserved.

*Keywords:* Shot boundary detection; Video indexing; Video browsing; Video similarity; Video retrieval

## 1. Introduction

State of the art video compression and communication technologies have enabled a large amount of digital video to be available online. Storage and transmission technologies have advanced to a stage so that they can accommodate the demanding volume of video data. Encoding technologies such as MPEG, H.263 and H.264 [1–4] provide for access to digital videos within the constraints of current communications infrastructure and technology. Even production of digital videos has become available to the masses with introduction of high performance, low-cost digital capture and recording devices. As a result, a huge volume of digital video content is available in digital archives on the World Wide Web, in broadcast data streams, and in personal and professional databases. Such a vast amount of content information calls for effective and efficient techniques for finding, accessing, filtering and managing video data. While search engines and database management systems suffice for text documents, they simply cannot handle the relatively unstructured, albeit information rich, video content. Hence, building a content-based video indexing system turns out to be a difficult problem. However, we can identify three tasks that are fundamental to building an efficient video management system: (i) the entire video sequence

*Corresponding author. Tel.: +65 6790 6241.
*E-mail address:* asltchia@ntu.edu.sg (L.-T. Chia).

0306-4379/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2005.12.005

must be segmented into shots, where a shot is defined as a collection of frames recorded from a single camera operation. This is akin to a tuple which is the basic structural element for retrieval in conventional text-based database management system; (ii) even though a shot determines a physical boundary in a video sequence, it does not convey any meaningful semantics within it. Hence, shots that are related to each other must be grouped together into a scene [5–7]; (iii) finally, a robust retrieval method depends on a model that captures similarity in the semantics of video content.

In this paper, we address the above fundamental tasks to provide an integrated approach for managing *compressed* video content. Since video is mostly available in compressed form, there is a need to develop algorithms to process compressed video directly without paying for overheads to decode them before processing. Tasks such as restoration, resolution enhancement, tracking, etc. in compressed videos have been reported recently [8–11]. Our objective is to develop a fully automatic technique for content-based organization and management of MPEG-compressed videos. To this end, the paper

1. describes a novel shot boundary detection (SBD) algorithm that is capable of detecting both abrupt and gradual shot changes like dissolve, fade-in/fade-out;
2. describes a scene tree that acts as an efficient structure to facilitate browsing;
3. presents a video similarity measure that enables efficient indexing of video content based on motion and which is shown to be useful in video retrieval.

As noted earlier, these three tasks provide for an integrated approach to browsing and retrieval in large video databases.

The remainder of this paper is organized as follows. In Section 2, we describe a novel SBD algorithm in the compressed domain. The procedure for building an adaptive scene tree is described in Section 3. In Section 4, the motion-based indexing and retrieval techniques are discussed. The experimental results are presented in Section 5. Finally, we give concluding remarks in Section 6.

## 2. Shot boundary detection algorithm

SBD is the first fundamental step to video content analysis. It segments video data into the basic unit, shot, for indexing and retrieval. A shot in a video sequence refers to a contiguous recording of one or more video frames depicting a continuous action in time and space. In a video database, the isolation of shots is of interest because the shot level organization of video sequences is considered to be the basic unit in video indexing and is appropriate for video browsing and content-based video retrieval [12]. In general, there are two types of shot changes: (1) abrupt change or hard cut, and (2) gradual change due to the various video editing effects, such as fade-in, fade-out, dissolve and wipe. There is a rich literature of algorithms for detecting video shot boundaries [13–15]. In the initial phase of research in SBD, the data mostly consisted of uncompressed video [13,16]. They can be broadly classified into methods that use one or more of the following features extracted from the video frames: (i) pixel differences [16], (ii) statistical differences [17], (iii) histogram [18], (iv) compression differences [19], (v) edge tracking [20,21], and (vi) motion vectors [22], etc. With the emergence of the MPEG compression standard, most of the videos are stored using the MPEG compression standard now. While the above approaches work in the spatial domain, it is prudent to develop algorithms in the compressed domain so as to save time in fully decoding the sequence. Some algorithms have been proposed to parse a video sequence directly from the compressed data [19,23,24]. Most of these methods use either the DC information [19] or the bit-rate of different MPEG picture types [24] in their algorithms; the methods using DC coefficients involve significant decoding of the MPEG-compressed video, while the methods using bit-rate do not yield satisfactory detection. Moreover, these methods are incapable of detecting gradual transitions. Recently, many researchers also address the problem of gradual transition detection [25]. In [26], the authors proposed a rule-based refinement postprocessing scheme to detect gradual transitions in digital video. However, these algorithms cannot detect all types of shot changes. They are incapable of handling both hard cuts and gradual transitions (i.e. fade, dissolve, etc.) simultaneously [25]. Lastly, most of them involve tight thresholds that are largely dependent on the video sequence [15]. In this section, we describe a method of detecting not only hard cuts but also gradual transitions [27]. Our algorithm requires only partially decoding of the video stream. Without the need to perform inverse DCT, our algorithm works fast. We exploit the number of macroblock (MB) types in each of the I-, P-, and B-frames to derive the SBD algorithm.

### 2.1. Abrupt shot change detection

We note that not all the MBs in a P-frame are forward motion predicted and not all the MBs in a B-frame are bidirectionally predicted [24]. Each MB in the P-frame is either forward predicted, skipped or intracoded. Similarly, each MB in the B-frame is either forward predicted, backward predicted, bi-directionally predicted, skipped or intracoded. However, a skipped MB is encoded in the same way as the first MB in the slice to which it belongs [28]. Hence, we are left with basically four types of MBs which we abbreviate as *In*, *Fw*, *Bk* and *Bi* for intracoded, forward predicted, backward predicted and bidirectionally predicted, respectively.

The number of each type of MBs in a frame is an indication of the similarity/dissimilarity of that frame with its neighboring frames. Since only the B-frame has all the four types of MBs, we use it to compare the dissimilarity through a frame dissimilarity ratio (FDR), which is defined as

$$FDR_n = \begin{cases} \dfrac{Fw_{n-1}}{Bi_{n-1}} & \text{for reference frame,} \\ \max\left(\dfrac{Fw_n}{Bi_n}, \dfrac{Bk_n}{Bi_n}\right) & \text{for B-frame,} \end{cases} \tag{1}$$

where the reference frame refers to either an I-frame or a P-frame and the index $n$ denotes the frame number. If a shot change takes place at a reference frame, most of the MBs in the previous frame (which has to be a B-frame) are predicted from the previous reference frame. In other words, *Fw* in the previous frame will be high resulting in a high FDR. On the other hand, if the shot change takes place in a B-frame, all the frames lying between the previous and the following reference frames are either forward predicted or backward predicted. If the number of bi-directionally predicted MBs is small, it once again results in a high FDR for these B-frames. The advantage of using only the B-frames for comparison is that we can avoid the need for normalizing the FDR as well as choosing a different threshold on the FDR for each type of frame. However, we observe that if a shot change takes place at a B-frame, all the B-frames lying between the previous and the next reference frames will have high FDRs. Consider the following frame structure in an MPEG bit stream:

$$\ldots I_1 B_2 B_3 B_4 P_5 B_6 B_7 B_8 P_9 \ldots .$$

If the shot change takes place at $B_3$, FDRs for $B_2$, $B_3$ and $B_4$ will be very high. In order to determine the exact location of the shot boundary, we observe that $B_2$ is mostly forward predicted while $B_3$ and $B_4$ are mostly backward predicted. Thus, at the shot boundary there is a change in the dominant MB type of the B-frame. So, we define a dominant MB change (DMBC) for frame $n$ as

$$DMBC_n = \begin{cases} 1 \\ 0, & \textit{if } (Bk_n - Fw_n)(Bk_{n-1} - Fw_{n-1}) > 0, \\ 1, & \textit{if } (Bk_n - Fw_n)(Bk_{n-1} - Fw_{n-1}) \leqslant 0, \end{cases} \tag{2}$$

where the first line in the above equation applies to the reference frames and the other two lines apply to the B-frames. The modified FDR (MFDR) for frame $n$ is then defined as

$$MFDR_n = FDR_n \times DMBC_n. \tag{3}$$

While the MFDR is the same as the FDR for a reference frame, there will be one or two consecutive high MFDRs for a shot change in a B-frame. If there is only one high MFDR, the corresponding frame indicates the beginning of a new shot. If there are two consecutive high MFDRs, the former indicates the end of a shot and the latter indicates the beginning of a new shot. In the example frame structure considered earlier, the MFDR is retained for frames $B_2$ and $B_3$, while it drops to zero for frame $B_4$.

An adaptive threshold mechanism based on a sliding window as proposed in [19] is used to detect hard cuts from the MFDR. The MFDR values are compared over a window whose length $l$ is set to be less than the duration of a shot, typically 10 frames. The presence of an abrupt change is detected at each window position, in the middle of the window, using the following adaptive threshold:

$$T = \frac{\max(MFDR_n, \ldots, MFDR_{n+l})}{\max(T_g, \operatorname{submax}(MFDR_n, \ldots, MFDR_{n+l}))} \geqslant \alpha. \tag{4}$$

Here, max and submax refer to the largest and the third largest values, respectively, of their arguments, the parameter $\alpha$ can be thought of as a shape parameter of the boundary pattern characterized by an isolated sharp peak in a series of discontinuity values [29] and $T_g$ is a global threshold which prevents the detected MFDR to be very small. At first glance, it would appear that the presence of $\alpha$ and $T_g$ imposes a tight constraint diluting the adaptive nature of the thresholding process. It is interesting to note, though, that an accurate choice of $\alpha$ and $T_g$ is not crucial to the success of the algorithm. We have observed that an empirical choice of $\alpha = 5$ and $T_g = 1$ works well with a wide variety of video sequences.

### 2.2. Detection of gradual transitions

We consider two types of gradual transitions, viz., fade and dissolve. A fade is characterized by a gradual darkening of a frame and its replacement by another image which appears either gradually or abruptly [30]. A fade-in occurs when the image gradually appears from a blank screen and a fade-out occurs when the image disappears gradually, leaving a blank screen. A dissolve occurs when one image fades away while another image appears. It can be viewed upon as a linear transition from the ending frame in one shot to the starting frame in the next shot with pictures from both the shots occupying the region of transition. From this perspective, fade-in and fade-out can be considered as special instances of a dissolve in which case, the former is a dissolve that begins with a blank frame and the latter is a dissolve that ends in a blank frame.

We use the DMBC defined in (2) to detect dissolves. Consider the following frame structure from an MPEG video: $\ldots I_1B_2B_3P_4B_5B_6P_7\ldots$ . If a dissolve takes place in this sequence, the dominant MB type in frame $B_2$ will be $Fw$ since $B_2$ is nearer to the reference frame $I_1$, and that in frame $B_3$ will be $Bk$ because it is nearer to $P_4$. Thus, we observe that during a dissolve the DMBC changes rapidly during a dissolve and consequently a high local sum of the DMBC indicates the presence of a dissolve. If $w$ is the length of a window centered around frame $n$, then the local sum of the DMBC for that frame is given by

$$F_n = \sum_{n-w}^{n+w} DMBC_k. \tag{5}$$

If $F_n$ is greater than a threshold $\tau$, then frame $n$ belongs to the set of frames that makes up the dissolve. It is heuristically found that the value of $\tau$ can be chosen to be 0.8 times the width of the window for *several* kinds of video sequences (news, movie, cartoon, sports). This choice is a tradeoff between the *precision* (the ratio of the number of shot changes detected correctly over the total number of shot changes detected) and *recall* (the ratio of the number of shot changes detected correctly over the actual number of shot changes). If we lower the threshold, we are able to increase the recall but the precision decreases. On the other hand, if we increase the threshold, the precision increases but the recall decreases. The width of the window is taken to be slightly less than the width of a dissolve, which is typically 10 frames. We choose our threshold to be about 80% of the window length. If the dissolve lasts longer than the window, the algorithm is still capable of detecting the high plateau as shown in Fig. 6 in Section 5. If the dissolve lasts only a few frames, for example 3–4 frames, it would be missed by Eq. (5). But it might result in a high MFDR in Eq. (3) due to the relatively large amount of changes between frames, which would then be potentially detected by Eq. (4). We do not set the window size too narrow, which would cause many false detections. Thus, we note that the presence of the threshold $\tau$ does not render the algorithm inflexible over different types of sequences, as we show in our experiments. A further refinement of the detected gradual transitions can be achieved by merging those transitions whose temporal distance is less than the width of a typical dissolve.

## 3. Scene tree

The objective of content-based indexing of videos is to facilitate easy browsing and retrieval. Ideally, a non-linear browsing capability is desirable as opposed to standard techniques like fast forward or fast reverse. This can be achieved, preferably, using a structure that represents video information as a hierarchy of various semantic levels. Such a multilayer abstraction makes it not only more convenient to reference video information but also simplifies video indexing and storage organization. Several multilevel structures have been proposed in the literature [6,7,12,31,32]. However, their tree structure is not adaptive. The underlying theme is to group together frames that are 'similar' to each other where similarity could be defined in such primitive terms as temporal adjacency [33] or in terms of video content. The latter results in the entity called *scene*, which should convey semantic information in the video being viewed. Hence, our objective is to build a browsing hierarchy whose shape and size are determined only by the semantic complexity of the video. We call such a hierarchical structure as an *adaptive scene tree*. The scene tree algorithm described in this section is motivated by the work of Oh and Hua [34] who build a scene tree for uncompressed videos. Our algorithm described in Section 3.1 is an extension and an improvement over [34] that works for compressed videos. Section 3.2 provides an illustrative example of the scene tree, which helps the reader to understand the scene tree construction algorithm. Section 3.3 describes the representation of the scene tree as an MPEG-7 compliant XML document.

### 3.1. Scene tree building algorithm

The main idea of the scene tree building algorithm is to sequentially compare the similarity of the key frame from the current shot to the key frames from the previous $w$ shots. Based on a measure of similarity, the current shot is either appended to the parent node of the previous shot or appended to a newly created node. The result of the scene tree building algorithm is a browsing tree whose structure is adaptive, i.e. the number of levels of the tree is larger for complex content and smaller for simple content. The details of the algorithm, which takes in a sequence of video shots and outputs the scene tree are shown below:

1. Initialization: Create a scene node $SN_i^0$ at the lowest level (i.e., level 0) of the scene tree for each $shot_i$. The subscript indicates the shot (or scene) from which the scene node is derived and the superscript denotes the level of the scene node in the scene tree.
2. Initialize $i \leftarrow 3$ (since the current shot has to be compared with at least two previous shots).
3. Check if $shot_i$ is similar to shots $shot_{i-1}, \ldots, shot_{i-w}$ (in descending order) using a function *isSimilar*(), which will be described later. The comparisons stop when a similar shot, say $shot_j$, is found. If no related shot is found, a new empty node is created and connected to $SN_i^0$ as its parent node; then proceed to Step 5.
4. For scene nodes $SN_{i-1}^0$ and $SN_j^0$,

   (a) If $SN_{i-1}^0$ and $SN_j^0$ do not currently have a parent node, we connect all scene nodes, $SN_i^0$ through $SN_j^0$ to a new empty node as their parent node.
   (b) If $SN_{i-1}^0$ and $SN_j^0$ share an ancestor node, we connect $SN_i^0$ to this ancestor node.
   (c) If $SN_{i-1}^0$ and $SN_j^0$ do not currently share an ancestor node, we connect $SN_i^0$ to the current oldest ancestor of $SN_{i-1}^0$, and then create a new empty node and connect the oldest ancestor of all nodes from $SN_j^0$ to $SN_{i-1}^0$ to it as its children.

5. If there are more shots, we set $i \leftarrow i+1$, and go to Step 3. Otherwise, connect all the nodes currently without a parent to a new empty node as their parent.
6. For each scene node at the bottom of the scene tree (it represents a shot), we select the key frame as its representative frame by choosing the I-frame in a shot whose DC value is closest to the average of the DC values of all the I-frames in the shot. Since we do not desire to decode all the frames in the video sequence (recall that we would like to process in the compressed domain), we choose the DC value of the I-frame to make the algorithm more efficient. We then traverse all the nodes in the scene tree from the bottom to the top. For each empty node visited, we identify the child node which contains the largest number of frames and assign its representative frame as the representative frame for this node.

We now return to the function *isSimilar*() used in Step 3 to compute the similarity between key frames. The computation of the similarity involves dividing the frames into background area and foreground area. Instead of using a fixed background and a fixed object area as in [34], we use a byproduct of the camera motion estimation algorithm described in Section 4. This byproduct, which we call a "Block Rejection Map (BRM)" consists of those MBs which corresponds to moving foreground object region in the scene. The reason for such a nomenclature will be evident in Section 4; however, for the moment, it will suffice to understand the BRM as a partition of the frame into foreground and background areas. Fig. 4 shows an example of a BRM. Operating on the HSV color space, we compute the color histograms of the foreground and background area of two key frames $k_1$ and $k_2$. The Hue(H) and Saturation(S) values are quantized into eight bins and four bins, respectively, while we ignore the Illumination(V) component.

The similarity measure is defined as

$$Sim = w_1 \times |hist_b(k_1, k_2)| + w_2 \times |hist_o(k_1, k_2)|, \quad (6)$$

where $|hist_b(.)|$ and $|hist_o(.)|$ are the Euclidean distances of histograms of the background and object areas, respectively, between frames $k_1$ and $k_2$ and $w_1, w_2$ are weights for the background and object areas, respectively. We choose $w_1 = 0.7$ and $w_2 = 0.3$ so as to give more weight to the background since a change in background area is a stronger indication of a change in the scene. The value of *Sim* in Eq. (6) is compared with a threshold to determine if two key frames are similar.

Although the above adaptive scene tree building algorithm is motivated by Oh and Hua [34], there are some important differences, which we highlight now. As mentioned earlier, our algorithm works directly on MPEG-2 compressed video in the sense that we need only partially decode the video to extract the motion vectors and DC images. Moreover, a full decoding is required only for one frame per shot that represents the key frame. Secondly, the similarity between shots is ascertained through a meaningful predefined window, $w$, as opposed to all the shots from the beginning of the video. We now compare the computational complexity of our algorithm to [34]. The shot similarity determination can be done in $O(w * s)$, where $w$ is the length of the window and $s$ is the number of shots. Generally, the number of frames in a shot, $f$ is much larger than $s$ and $s$ is larger than $w (f \gg s > w)$. The scene tree construction algorithm involves traversal in Step 4 and Step 6. The computational complexity of transverse a tree is of $O(\log(s))$. Therefore, the worst-case computational complexity of building the tree is of $O(s * \log(s))$. Thus, the total computational complexity of our algorithm is of $O(s * \log(s))$, while that in [34] is $O(s * f^2)$. Finally, we partition the frame into background and foreground areas using BRM dynamically, while the frame is partitioned into fixed background/object area partition in [34]. The usage of BRM gives more accurate partition of object and background areas, hence results more accurate scene similarity calculation.

## 3.2. Example of a scene tree

In this section, we walk through the steps leading to the construction of a scene tree, using a symbolic video sequence consisting of 10 shots. The shots denoted by the same gray color in Fig. 1(a) are regarded as similar by the *isSimilar*() function.

The scene tree construction algorithm is initialized by attaching a scene node, $SN_{0,i}$ for each shot $i$, $i = 1, \ldots, 10$ (Step 1) as in Fig. 1(a). The shots are scanned to check the similarity between scene node pairs. $SN_{0,4}$ is found to be similar to $SN_{0,2}$. Since the

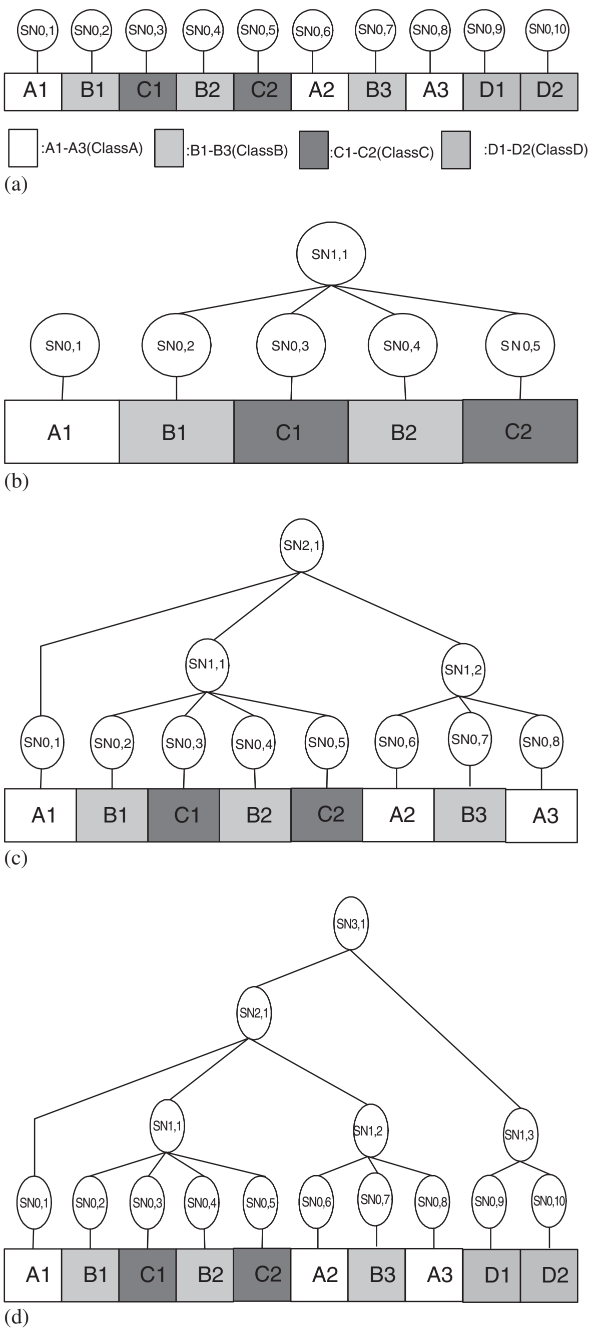

Fig. 1. Illustration of scene tree construction algorithm.

previous node $SN_{0,3}$ and $SN_{0,2}$ do not have a parent node, $SN_{1,1}$ is created in the upper level and all scene nodes from $SN_{0,2}$ to $SN_{0,4}$ are connected to $SN_{1,1}$ (condition (a) of Step 4). Next, $SN_{0,5}$ is found to be similar to $SN_{0,3}$. Since the previous $SN_{0,4}$ and $SN_{0,3}$ share a common ancestor node, $SN_{1,1}$, $SN_{0,5}$ is connected to $SN_{1,1}$ as the child node (condition (b) of Step 4). Fig. 1(b) shows the constructed scene tree at this stage. Since $SN_{0,6}$ is not similar to any of the previous three shots, $SN_{1,2}$ is created as its parent. However, $SN_{0,7}$ is similar to $SN_{0,4}$ and since $SN_{0,6}$ and $SN_{0,4}$ do not share a common parent, a new node $SN_{2,1}$ is created with children $SN_{0,1}$, $SN_{1,1}$ and $SN_{1,2}$ (condition (c) of Step 4). Next, $SN_{0,8}$ is similar to $SN_{0,6}$ and since $SN_{0,6}$ and $SN_{0,7}$ share the same parent node, $SN_{0,8}$ is also connected as their sibling. The scene tree at this stage is illustrated in Fig. 1(c). We observe that if the comparison of the scene nodes are not restricted to within a window, $SN_{0,6}$ would be deemed similar to $SN_{0,1}$ causing loss of the inherent semantic information in the tree. $SN_{0,9}$ and $SN_{0,10}$ are found to be different from all other shots considered so far and a new parent node $SN_{1,3}$ is created for them. Finally, the scene node $SN_{2,1}$ and $SN_{1,3}$ are connected to the root node $SN_{3,1}$ of the scene tree, which is shown in Fig. 1(d).

### 3.3. Scene tree representation

The algorithm in Section 3.1 outputs a description for the video content in the form of a tree structure called ‘Scene Tree’. It is an important tool for video content management. Here, we address the issue of representing the ‘Scene Tree’. If the ‘Scene Tree’ is stored in different ways, it would not be possible to access content across several repositories to enable content exchange using different description methods. These are interoperability issues. We can define a standardized description to solve the interoperability issue so that the scene tree information can be shared and reused. The MPEG-7 standard is aimed at addressing this problem. Formally known as “Multimedia Content Description Interface”, it provides a set of “Descriptors” and “Description Schemes”, along with a “Description Definition Language”, to describe multimedia content [35]. The scope of the standard covers various forms of media including audio, video and images, both in digital and analog format, although most applications involve only digital content. MPEG-7 allows a standard description scheme of various aspects of multimedia material that can be used by MPEG-7 enabled applications aimed at end user, or automatic systems. Thus, we choose to represent the ‘scene tree’ information in MPEG-7 compliant XML document.

XML (eXtensible Markup Language) [36] is a simple and flexible text format derived from SGML(ISO 8879). It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification introduced by the World Wide Web Consortium (W3C). XML schema has been widely used as a schema language for constraining the structure and content of XML document. After a detailed evaluation of XML and XML schema, MPEG-7 chose to adopt and extend XML and XML schema as the description definition language (DDL) for MPEG-7 document. Thus the MPEG-7 documents are XML documents that conform to particular MPEG-7 schemas (expressed in DDL) and that describe audiovisual content. In MPEG-7, there are a collection of schemas called “Multimedia Description Schema (MDS)”. The MDS provides a set of standardized tools to facilitate multimedia content description. In MDS, there is a specific schema for temporal video decomposition, which matches our scene tree-based video description.

An example of an MPEG-7 compliant XML document is shown in Fig. 2. The XML document starts with <Mpeg7> as the root node. This node has two children: <DescriptionMetadata> and <Description>. The <DescriptionMetadata> is used to describe the metadata information about the XML document, such as author, confidence, version, etc. and the <Description>, which is of ContentEntityType, is used for multimedia content description. The scene tree information is contained in the <TemporalDecompostion> node under “Mpeg7/Description/MultimediaContent/Video/”. Since there is no gap in the scene tree decomposition and parent scene nodes are overlapped with child scene nodes, the *gap* and *overlap* attributes of <TemporalDecompostion> are set to false and true, respectively. Each node in the scene tree is represented by a <VideoSegment>, which contains a <StartFrame>, a <EndFrame> and one or more <VideoSegment>(s) depending on number of child scene nodes that the current scene node has. Note that other MPEG-7 visual descriptors such as color, texture, motion, etc. can be inserted to the <VideoSegment> too.

We will illustrate an XML representation of an actual scene tree derived from a video sequence in Section 5.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Mpeg7>
  <DescriptionMetadata>
    .......
  </DescriptionMetadata>
  <Description xsi:type="ContentEntityType">
    <MultimediaContent xsi:type="VideoType">
      <Video>
        <MediaLocator>
          <MediaUri>file:///c:/example.mpg</MediaUri>
        </MediaLocator>
        <TemporalDecomposition gap="false" overlap="true">
          <VideoSegment>
            <StartFrame>0</StartFrame>
            <EndFrame>2999</EndFrame>
            <Keyframe>1020</Keyframe>
            <VideoSegment>
              <VideoSegment> ... </VideoSegment>
                 ...
            </VideoSegment>
                 ......
          </VideoSegment>
        </TemporalDecomposition>
      </Video>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

Fig. 2. Example of XML representation for scene tree.

## 4. Motion-based indexing and retrieval

Video retrieval and browsing can be envisioned as an integrated and interactive process. This process can be seen to be analogous to browsing through books with a table of contents in the front and an index at the back of the book. In order to search for a specific topic of interest, one may consult the index to locate the pages where it appears. Consequently, if one wants to know more about the topic, he may refer to the table of contents to browse through chapters, sections and so on. Similarly, for video data, the feature vectors serve as the index and the scene tree serves as the table of contents. The search for relevant content in the video is initiated by locating the video shot by feature matching/retrieval and then carried forward by browsing up and down along the scene tree (table of content) to further explore the relevant content. Hence, the feature vector and the scene tree are closely related tools for the management of the video content. Many visual features like color, texture, shape, edge, etc. have been extracted to be used as indices [37–41]. However, such indexing techniques do not take into account the essential characteristic of video, viz., its temporal dimension. Recent works have used the spatio-temporal relationship among video frames by extracting motion information inherent in them [42–45]. The authors in [42] use texture features extracted from temporal slices to index motion content. While patterns in spatio-temporal slices reveal camera motions (pan and zoom) and direction of motion, they do not indicate the *intensity* of motion in a video sequence. The authors in [43] use the trajectory of the moving object as the index to motion content. However, it is very difficult to extract the trajectories of moving objects under complex scene. In [44], the authors use Markov random fields to characterize the optical flow field of video clips. This method is computationally intensive and is, therefore, not suitable for long video clips in large video databases. In [45], the authors use block-based motion vectors and principal component analysis to represent motion content, but this is done at a very coarse level. The MPEG-7 standard provides motion descriptor to describe motion activity. The extraction of this descriptor is based on aggregate motion vectors contained in the compressed bitstream [46]. In this section, we describe a new motion feature for video indexing and retrieval. The motion content is represented by three components, viz., the camera motion (*CM*), the object motion (*OM*), and the total motion (*TM*). Each of $CM, OM$ and $TM$ are matrices and it is these matrices that serve as indices for video retrieval.

### 4.1. Motion extraction

Recall that all the videos considered in this paper are compressed according to MPEG-2 format. Hence, the MV for frame $n$ is obtained as

$$MV_n = \begin{cases} -Bk_{n-1} & \text{for I-frame,} \\ Fw_n - Fw_{n-1} & \text{for P-frame,} \\ (Fw_n - Fw_{n-1} & \\ \quad +Bk_n - Bk_{n-1})/2 & \text{for B-frame.} \end{cases} \tag{7}$$

Refer to Section 2 for meanings of the notations. The MV fields (MVFs) obtained are then smoothened using a spatial $(3 \times 3)$ median filter followed by a temporal $(1 \times 1 \times 3)$ median filter. The smoothened MVFs are then used to calculate the camera motion and object motion. The total motion component at each MB is just the smoothened motion vector at its center.

We use a six parameter affine model to estimate the camera motion which is modelled as

$$mv_x(i,j) = p_1 \cdot i + p_2 \cdot j + p_3,$$
$$mv_y(i,j) = p_4 \cdot i + p_5 \cdot j + p_6, \tag{8}$$

where $mv_x(i,j)$ and $mv_y(i,j)$ are the $x$ and $y$ components of the motion vector for a MB centered at $(i,j)$, $p$'s are the affine parameters. Consider a group of MBs $G$ whose affine parameters $P = \{p_1, \dots, p_6\}$ need to be determined. We will explain the significance of $G$ shortly, when we present the iterative algorithm for camera motion estimation. Using the method of least squares, from Eq. (8), $P$ is obtained by minimizing

$$\sum_G (mv_x(i,j) - p_1 \cdot i - p_2 \cdot j - p_3)^2 + (mvy(i,j) - p_4 \cdot i - p_5 \cdot j - p_6)^2. \tag{9}$$

The two terms inside the summation can be minimized independently. Considering the first term only and differentiating it with respect to $p_1$, $p_2$ and $p_3$ and setting the resulting equations to zero, we get

$$\sum_{(i,j)\in G} (mv(i,j) - p_1 \cdot i - p_2 \cdot j - p_3)i = 0, \tag{10}$$

$$\sum_{(i,j)\in G} (mv(i,j) - p_1 \cdot i - p_2 \cdot j - p_3)j = 0, \tag{11}$$

$$\sum_{(i,j)\in G} (mv(i,j) - p_1 \cdot i - p_2 \cdot j - p_3) = 0. \tag{12}$$

If the origin of the frame is taken to be at its center (instead of the top left corner as is conventionally done), $i' = i - (V+1)/2$ and $j' = j - (U+1)/2$, $U \times V$ is the size of the motion vector field, then $\sum_G i' = 0$ and $\sum_G j' = 0$ and the affine parameters can be easily shown to be

$$p_1 = \frac{IX \cdot YY - JX \cdot XY}{A}, \quad p_2 = \frac{JX \cdot XX - IX \cdot XY}{A},$$
$$p_3 = \frac{\sum_G mv_x(i,j)}{g}, \quad p_4 = \frac{IY \cdot YY - JY \cdot XY}{A},$$
$$p_5 = \frac{JY \cdot XX - IY \cdot XY}{A}, \quad p_6 = \frac{\sum_G mv_y(i,j)}{g}, \tag{13}$$

where

$$IX = \sum_G i' \cdot mv_x(i',j'), \quad JX = \sum_G j' \cdot mv_x(i',j'),$$

$$IY = \sum_G i' \cdot mv_y(i',j'), \quad JY = \sum_G j' \cdot mv_y(i',j'),$$

$$XX = \sum_G i'^2, \quad YY = \sum_G j'^2, \quad XY = \sum_G i' \cdot j',$$

$$A = XY \cdot XY - XX \cdot YX,$$

and $g$ is the number of MBs in the group $G$. $mv(i',j') = mv(i,j)$ since they refer to the same MB.

The algorithm to estimate the affine parameters starts by labelling all the MBs in a frame as 'inliers'. Then the parameters are estimated for all the inliers using Eq. (13). A new set of motion vectors are reconstructed for each inlier using the estimated parameters. If the magnitude of the residual motion vector $R_{mv}$, calculated as the difference between the original motion vector and the reconstructed one, is greater than an adaptive threshold $T = \max(\text{median}(R_{mv}), \beta)$ for a particular MB, then that MB and the one situated diagonally opposite it are marked as 'outliers'. The role of $\beta$ is to prevent the rejection of a large number of MBs if the median of the residuals is very small. We choose $\beta$ to be 1. Fig. 3 shows the histogram of the magnitudes of residual motion vectors between two frames in an 'example' soccer video. Note that the diagonally opposite MBs are also marked as 'outliers' due to the shifting of the co-ordinate axes to the center of the frame. After each iteration, some MBs are

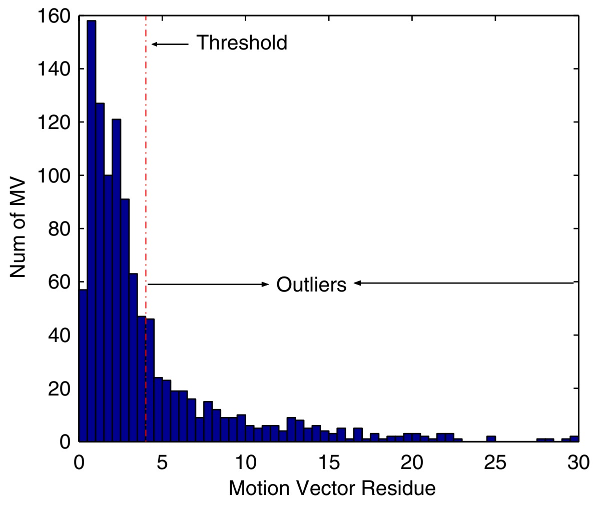

Fig. 3. Histogram of magnitudes of residual motion vectors between two frames in an example ‘soccer’ video.

marked as ‘outliers’; these MBs correspond to areas in which the motion is associated with moving objects. The steps in the algorithm to estimate the camera motion are shown in Algorithm 1.

**Algorithm 1.** Camera Motion Estimation

```
1: Mark all the MBs as ‘inliers’.
2: Estimate the motion affine parameters for ‘inliers’ (Eq. (13)).
3: Reconstruct the global motion vector at each MB with the estimated affine parameters.
4: Calculate residual motion vector ($R_{mv}$) as the difference between the original and reconstructed motion vector.
5: If $R_{mv}$ is greater than max(median($R_{mv}$), $\beta$), then mark this MB and its opposite diagonal MB as ‘outliers’. median($R_{mv}$) is the median of all the $R_{mv}$s.
6: Go to Step 2 until there are no more new ‘outliers’ or more than two thirds the MBs are marked as ‘outliers’.
7: If more than two thirds of the MBs are ‘outliers’, the affine parameters are set to zero.
```

When Algorithm 1 is terminated, the MBs that are rejected as outliers correspond to moving foreground regions. The background area consisting of MBs labelled as inliers are marked with zero intensity. Such a partition of frame into foreground and background regions results in the block rejection map; an example of which is shown in Fig. 4. As mentioned in the previous section, we can utilize the BRM for the calculation of the similarity between two scenes since the meaningful calculation of scene similarity requires different treatments of background and foreground areas.

After estimating the camera motion, we now describe the $CM$, $OM$, $TM$ matrices for motion content indexing. Since the computed camera motion vector should not be greater than the total motion vector at a MB,

$$|cm(i,j)| = \min(|mv(i,j)|, |cm(i,j)|), \tag{14}$$

where $cm(i,j)$ and $mv(i,j)$ are the camera motion vector and total motion vector, respectively, at the MB centered at $(i,j)$. For the same reason

$$|om(i,j)| = \max(0, |mv(i,j) - cm(i,j)|), \tag{15}$$

where $om(i,j)$ is the object motion vector at the MB centered at $(i,j)$. The $CM$, $OM$ and $TM$ matrices are now formed for a shot by accumulating $|cm(i,j)|$, $|om(i,j)|$ and $|mv(i,j)|$, respectively, over all the frames in the shot, i.e., $CM(i,j) = \sum_{l=1}^{n} cm_l(i,j)$, $OM(i,j)=\sum_{l=1}^{n} om_l(i,j)$, and $TM(i,j)=\sum_{l=1}^{n} mv_l(i,j)$, where $n$ is the number of frames in the shot. These matrices serve as indices for retrieval, as explained in the next subsection.

## 4.2. Video similarity measure

Each shot is characterized by the three matrices $CM$, $OM$ and $TM$. In order to compute the similarity

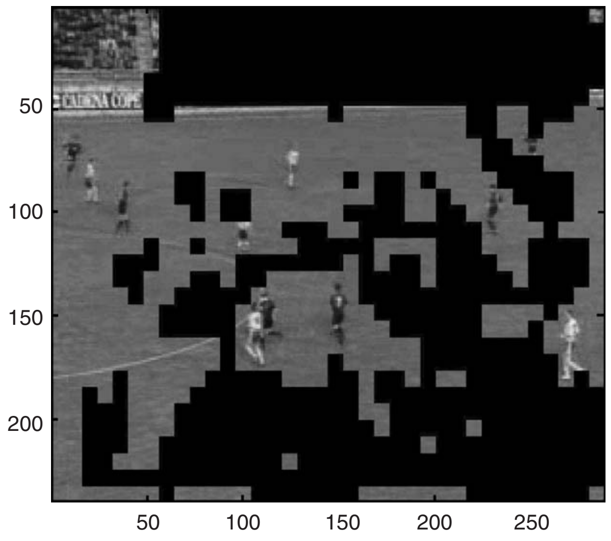

Fig. 4. Block rejection map.

between two shots, we assume that the matrices belong to the Hilbert sequence space $l^2$ with metric defined by

$$d(M_x, M_y) = \left[ \sum_{u=1}^{U} \sum_{v=1}^{V} |M_x(u,v) - M_y(m,n)|^2 \right]^{1/2}, \tag{16}$$

where $M_x$ and $M_y$ are the motion matrices of order $U \times V$. However, instead of comparing matrices, we reduce the dimension of the feature space by projecting the matrix elements along the rows and the columns to form one dimensional feature vectors $M_x^r(v) = \sum_{u=1}^{U} M_x(u,v)$ and $M_x^c(u) = \sum_{v=1}^{V} M_x(u,v)$. The metric in Eq. (16) can then be rewritten as

$$d(M_x, M_y) = \left[ \sum_{v=1}^{V} \frac{1}{U} |M_x^r(v) - M_y^r(v)|^2 + \sum_{u=1}^{U} \frac{1}{V} |M_x^c(u) - M_y^c(u)|^2 \right]^{1/2}. \tag{17}$$

This leads us to the distance function between two shots $s_x$ and $s_y$ to be defined as

$$\hat{d}(s_x, s_y) = \omega_C \cdot d(CM_x, CM_y) + \omega_O \cdot d(OM_x, OM_y) + \omega_T \cdot d(TM_x, TM_y), \tag{18}$$

where the $d(\cdot)$'s are computed according to Eq. (17) and the $\omega$'s are the weights for each motion component. The weights are assigned in such a way that there is equal contribution from each of the components to the distance function. Without prior information, it is a common strategy to assign equal weight to different components. This mechanism is widely used and known as the "normalization" scheme in database literature and the "pre-whitening" in signal processing literature. However, if there are preferences for certain features, then the weights can be chosen accordingly, e.g., if the user is more interested in objection motion, then more weight can be assigned to OM feature. The interactive relevance feedback from the user can also be used to guide to assign the weights of different features. Use of distance function enables current matrix-based indexing techniques, like M-tree, R-tree, etc. to be applied as the index to the feature vectors to facilitate efficient search.

## 5. Experimental results

Our experiments were designed to assess the performance of the proposed techniques for SBD, scene tree construction, motion estimation and motion-based indexing and retrieval of shots. In the following subsections, we present the experimental results for SBD, scene tree construction, motion estimation and motion-based indexing and retrieval of shots.

### 5.1. Shot boundary detection

We have tested our algorithm on a wide variety of video sequences like soccer, news, movie and cartoons which were encoded in the MPEG-2 format with a GOP length of 12 frames. The encoding pattern was

*...IBBPBBPBBPBB...* .

There were a total of 105 hard cuts and 25 gradual transitions of which 20 are dissolves, three are fade-ins and two are fade-outs. Fig. 5 shows the MFDR, (see Eq. (3)) for 373 frames of a 'news' video. As expected, the hard cuts at frames 120 and 300 have a very high MFDR, which enables easy detection of the shot boundary. As for gradual transitions, local sum of DMBC, (see Eq. (2)) for a 'movie' sequence is shown in Fig. 6. We note that the beginning of the dissolve takes place at frame 43 and ends at frame 68. Table 1 summarizes the results of SBD. In the experiment, the global threshold $T_g$ in Eq. (4) for detecting hard cuts was set to 1. For detecting gradual transitions, the length of the window over which the local sum is computed is set to 12, with $\tau$ (the threshold for $F_n$ in Eq. (5)) set to 10. The video sequences are chosen so as to include various kinds of camera and object motions. We use *precision* (the

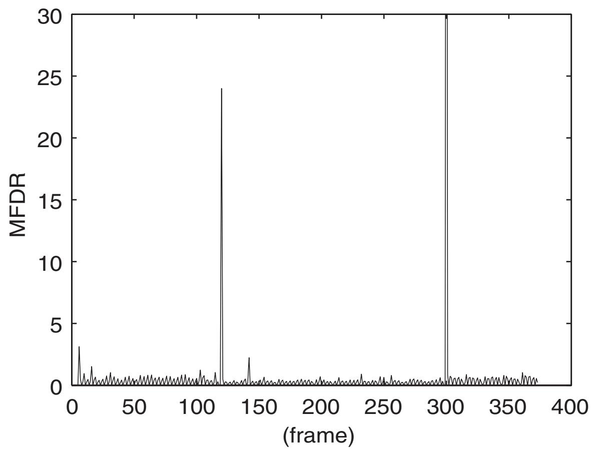

Fig. 5. MFDR for a news sequences.

ratio of the number of shot changes detected correctly to the number of shot changes detected) and *recall* (the ratio of the number of shot changes detected correctly over the actual number of shot changes) to measure the performance of the SBD algorithm.

The recall rates for hard cuts are consistently above 95% over all the sequences reinforcing the fact that the choice of the thresholds are not critical to the success of the method. The missed cuts are mostly due to large motion at the shot boundary. The overall recall rate for gradual transitions is 92% (only two of the 25 transitions were missed). Note that $\tau$ is chosen the same for all the sequences. The false detection of hard cuts is attributed to the sudden change in brightness of the frame due to flash in the 'news' sequence, a lightning bolt in the 'movie' sequence and the sudden appearance of text in the 'cartoon' sequence. We must mention that the 'news' sequence has 45 instances of flashes which were potential candidates for shot boundaries, out of which only four were falsely detected, and yielding the precision rate of 91.3%. However, the overall precision obtained for hard cuts is 91.75%. The false detection of gradual transitions is mainly due to the undesirable shaking of the camera which pulls down the camera motion characterization parameters to near zero. This is evident from the relatively low precision obtained for 'soccer' and 'movie' sequence where such jerky movements of the camera are present.

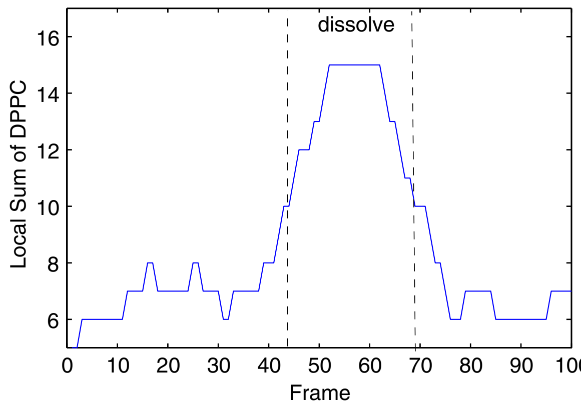

Fig. 6. Local sum of DMBC in a dissolve.

## 5.2. Adaptive scene tree construction

To evaluate the scene tree building algorithm, we run the algorithm described in Section 3 for various videos. Since we cannot quantify the effectiveness of the algorithm to produce the scene trees, we show four examples of scene trees in Figs. 7–10 and describe the main events occurring in the video sequences from which the trees are constructed.

The scene tree in Fig. 7 is built from a 3-min video clip available from the MPEG-7 test video set (CD 21 Misc2.mpg). In this sequence, two women are watching the TV, and talking to each other (the first eight shots, leaf nodes). Then one of the women leaves while the other continues watching TV (next four shots). After a while, this woman also leaves (starting from frames 3300). Then both the women appear together. One of them puts on a coat and leaves while the other starts cleaning the table (the

Table 1
Summary of SBD results

| Video (frames) | Cut (correct,false) | Gradual (correct,false) | Recall/precision Cut | Recall/precision Gradual |
|---|---|---|---|---|
| News (32640) | 43 (42, 4) | 12 (11,2) | 97.67/91.3 | 91.67/84.62 |
| Movie (29010) | 17 (16, 2) | 3 (3 ,1) | 94.12/88.89 | 100/75 |
| Cartoon (1498) | 13 (13, 2) | 0 (0,0) | 100/86.67 | — |
| Soccer (21310) | 32 (31, 2) | 10 (9, 3) | 96.88/93.94 | 90/75.00 |
| Total (84458) | 105 (102, 10) | 25 (23, 6) | 97.41/91.75 | 92.00/79.31 |

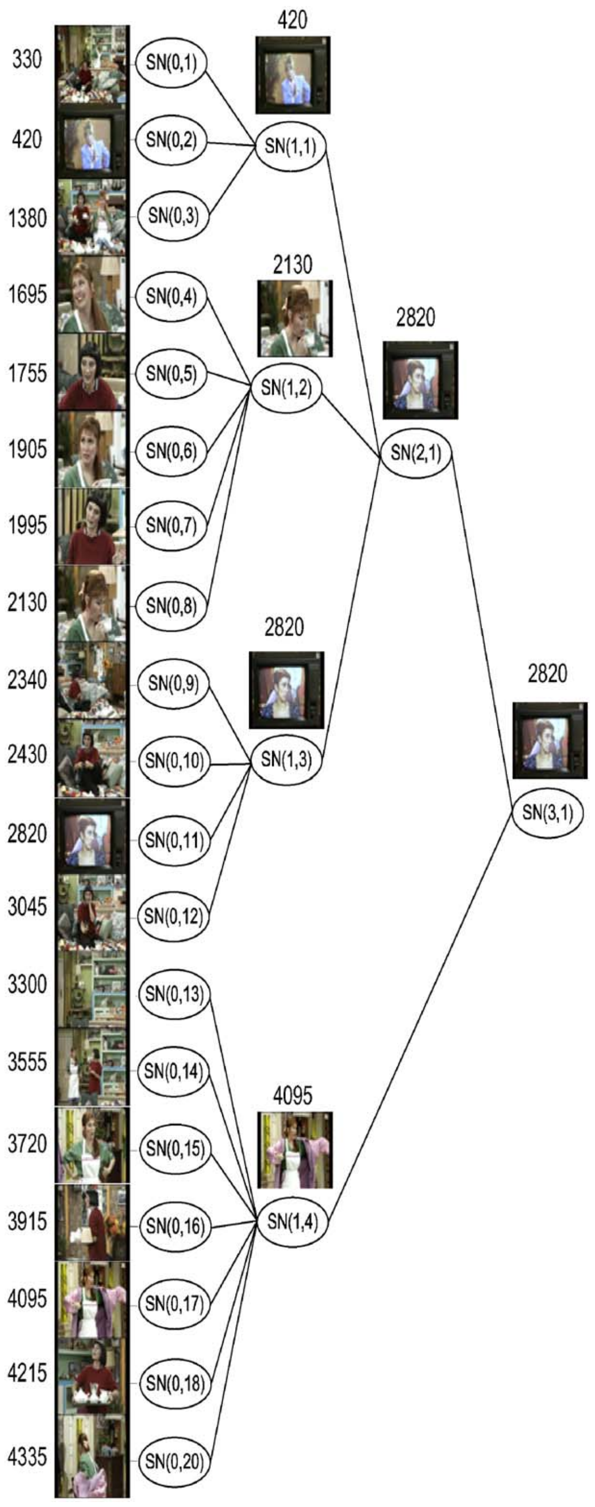

Fig. 7. Scene tree of MPEG-7 test video.

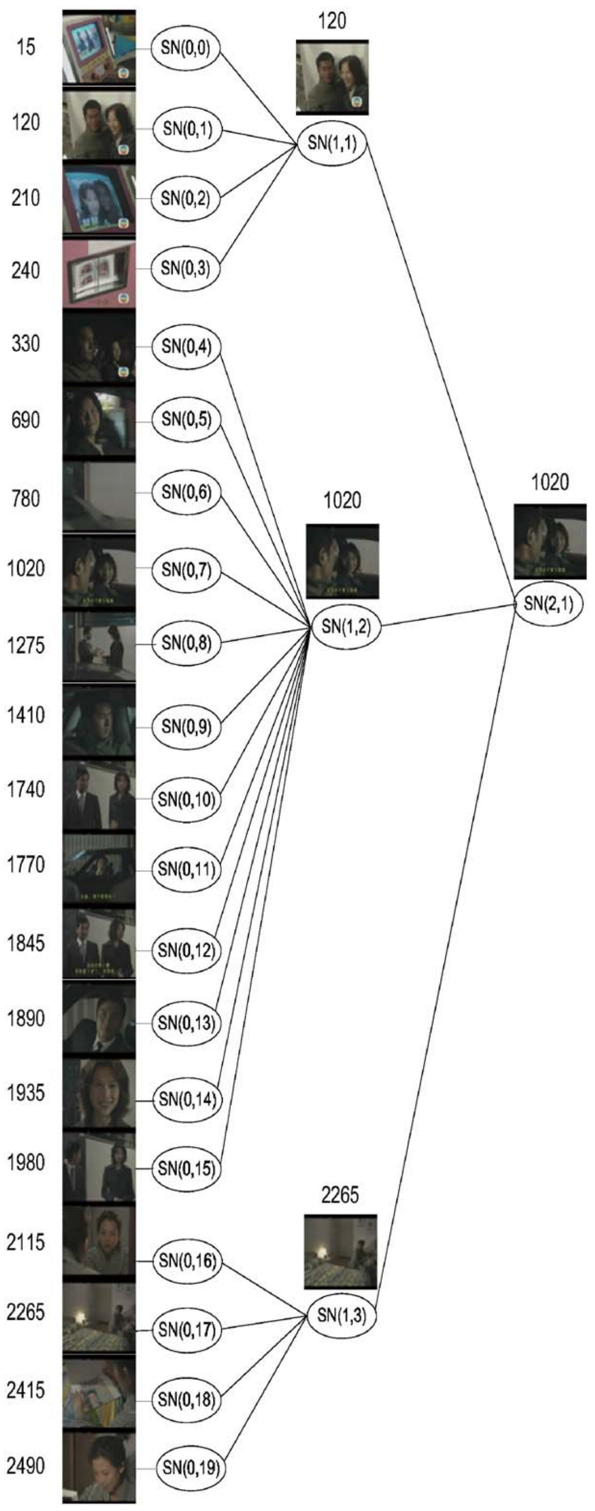

Fig. 8. Scene tree of TV program "Opening The New Era".

last shots, leaf nodes). The output XML file for the scene tree is shown below:

```
<?xml version = "1.0" encoding = "iso-
8859-1"?>
<Mpeg7>
  <DescriptionMetadata>
    <Confidence>0.85</Confidence>
    <Version>1.4</Version>
    <LastUpdate>2003-09-
1T10:00:00+00:00
    </LastUpdate>
    <Comment>
      <FreeTextAnnotation>
       Scene Tree.
      </FreeTextAnnotation>
    </Comment>
  </DescriptionMetadata>
  <Description
xsi:type = "ContentEntityType">
    <MultimediaContent xsi:type =
"VideoType">
      <Video>
        <MediaLocator>
          <MediaUri>
           file:///c:/MPEG-7_tv.mpg
          </MediaUri>
        </MediaLocator>
        <TemporalDecomposition gap =
"false" overlap = "true">
          <VideoSegment>
            <StartFrame>0</
StartFrame>
            <EndFrame>4432</
EndFrame>
            <Keyframe>2820</
Keyframe>
            <VideoSegment>...</
VideoSegment>
              ......
          </VideoSegment>
        </TemporalDecomposition>
      </Video>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

The <DescriptionMetadata> carries the meta data information of the scene tree XML document(Confidence, Version, LastUpdate, Comments). The main content of the scene tree is stored in the <TemporalDecomposition> descriptor under "/Description/MultimediaContent/Video/". Each scene node is represented by a <VideoSegment>. In total, there are 26 <VideoSegment>s which correspond to the number of scene nodes in the scene tree shown in Fig. 7. The <MediaLocator> descriptor under the <Video> indicates the source of the video.

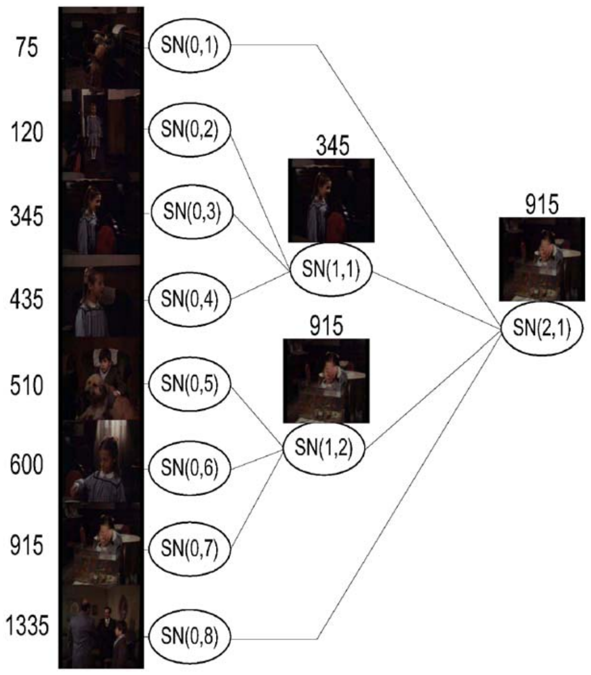

Fig. 9. Scene tree of MPEG-7 test video (CD20#1).

The scene tree in Fig. 8 is built from a 2-min video clip from the TV program "Opening The New Era". This video sequence consists of three scenes. A man and a woman are taking a picture together. Then the man drives the woman to her home. The woman goes inside her home and reads the newspaper. If we traverse the scene trees from the top to bottom, which in essence is a non-linear browsing of the video, we get the above stories. The scene trees give a hierarchical view of the scenes as well as provide a summary of the video sequence. The output XML document for the scene tree is shown below:

```
<?xml version = "1.0" encoding = "iso-
8859-1"?>
<Mpeg7>
  <DescriptionMetadata>
    <Confidence>0.85</Confidence>
    <Version>1.4</Version>
    <LastUpdate>2003-09-
1T10:00:00+00:00</LastUpdate>
    <Comment>
      <FreeTextAnnotation>
       Scene Tree.
      </FreeTextAnnotation>
    </Comment>
  </DescriptionMetadata>
  <Description
xsi:type = "ContentEntityType">
    <MultimediaContent xsi:type =
"VideoType">
      <Video>
        <MediaLocator>
          <MediaUri>
           file:///c:/csj_tv.mpg
          </MediaUri>
        </MediaLocator>
        <TemporalDecomposition gap =
"false"overlap = "true">
          <VideoSegment>
            <StartFrame>0</
StartFrame>
            <EndFrame>2999</
EndFrame>
             <Keyframe>1020</
Keyframe>
            <VideoSegment> ... </
VideoSegment>
              ......
          </VideoSegment>
        </TemporalDecomposition>
      </Video>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

The scene tree information is stored in the <TemporalDecomposition> descriptor. There are 24 nodes in the scene tree, thus there are 24 <VideoSegment> under the <TemporalDecomposition> descriptor.

Fig. 9 shows the scene tree extracted from another MPEG-7 test video (CD20 Misc1.mpg). The length of the video clip is 1 min. This video clip contains two different events. The first event happens between a boy and a girl. The boy sits on a chair and plays with a dog in the room. A girl enters the room and sees the boy. She begins to talk to him. Then the girl begins to feed the goldfish. The next event in this video clip happens between the boy and two other men. One man introduces the boy to the other. The output XML document for the scene tree is as follows:

```
<?xml version = "1.0" encoding = "iso-
8859-1"?>
<Mpeg7>
  <DescriptionMetadata>
    <Confidence>0.85</Confidence>
    <Version>1.4</Version>
    <LastUpdate>2003-09-
1T10:00:00+00:00</LastUpdate>
    <Comment>
      <FreeTextAnnotation>
       Scene Tree.
      </FreeTextAnnotation>
    </Comment>
  </DescriptionMetadata>
  <Description xsi:type =
"ContentEntityType">
    <MultimediaContent xsi:type =
"VideoType">
      <Video>
        <MediaLocator>
          <MediaUri>
           file:///c:/MPEG-7_CD20_1.mpg
          </MediaUri>
        </MediaLocator>
        <TemporalDecomposition gap =
"false"overlap = "true">
          <VideoSegment>
            <StartFrame>0</
StartFrame>
            <EndFrame>1499</
EndFrame>
            <Keyframe>915</Keyframe>
            <VideoSegment> ... <
/VideoSegment>
              ......
          <
/VideoSegment>
        </TemporalDecomposition>
       </Video>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

Fig. 10 shows another scene tree extracted from the same video as in Fig. 9. This video clip lasts 1 min. The story happens in the classroom. There are three events in this video clip. The first event is that the teacher walks into the classroom and begins the class. The second event is that the students hear

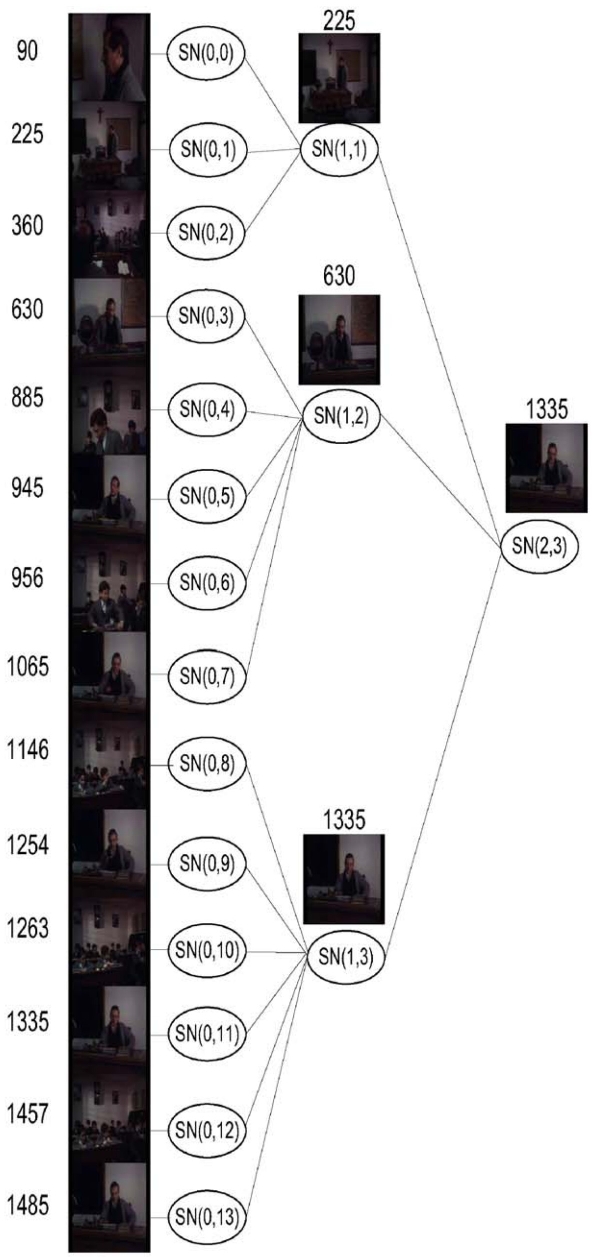

Fig. 10. Scene tree of MPEG-7 test video (CD20#2).

the bell and think the class is over. They begin to leave the classroom. The teacher gets angry and orders the student to go back to their seats and continues the class. The corresponding XML document of the scene tree is shown below:

```
<?xml version="1.0" encoding="iso-
8859-1"?>
<Mpeg7>
  <DescriptionMetadata>
    <Confidence>0.85</Confidence>
    <Version>1.4</Version>
    <LastUpdate>2003-09-
1T10:00:00+00:00</LastUpdate>
    <Comment>
      <FreeTextAnnotation>
        Scene Tree.
      </FreeTextAnnotation>
    </Comment>
  </DescriptionMetadata>
  <Description xsi:type=
"ContentEntityType">
    <MultimediaContent xsi:type=
"VideoType">
      <Video>
        <MediaLocator>
          <MediaUri>
            file:///c:/MPEG-7_CD20_2.mpg
          </MediaUri>
        </MediaLocator>
        <TemporalDecomposition gap=
"false"overlap="true">
          <VideoSegment>
            <StartFrame>0</
StartFrame>
            <EndFrame>1499</
EndFrame>
            <Keyframe>1335</
Keyframe>
            <VideoSegment> ... <
/VideoSegment>
              ......
          </VideoSegment>
        </TemporalDecomposition>
      </Video>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

As described in previous sections, the scene tree can be viewed as "the table of content" of a video. It can used as a guide for video content browsing and navigation. It has a very wide range of practical applications. We have integrated the "scene tree" into a semiautomatic video annotation tool, called "VideoBuddy". The interface of VideoBuddy is shown in Fig. 11. The user selects the input video to be annotated. Our algorithm automatically performs the SBD, key frame extraction and builds up

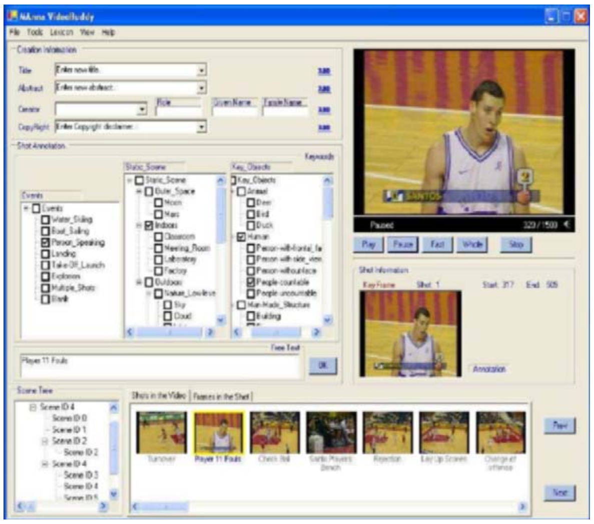

Fig. 11. Video annotation tool (VideoBuddy) interface.

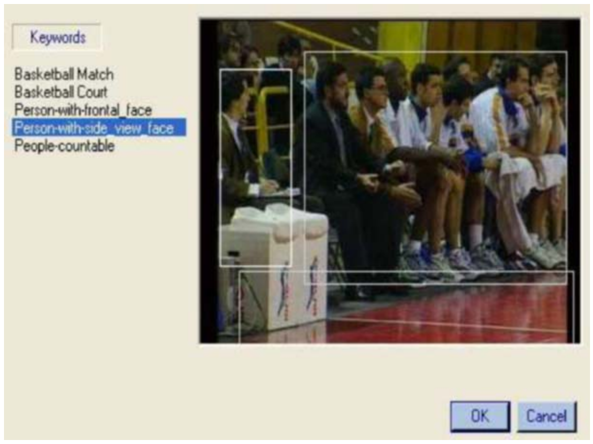

Fig. 12. Key frame annotation.

the scene tree. The outputted scene tree is shown in the bottom left part in the interface. The key frames from each shot are shown as thumbnail images on the right panel besides the scene tree panel. Then, the user can navigate the "scene tree" and select interested scene node for annotation. When the user select one scene node, the corresponding scene node information from the scene tree is shown in "shot information" window. The user can view the selected scene in the embedded media player. After that, he can annotate the video with key words by ticking the appropriate checkbox. The user can also perform the key frame region annotation by drawing a rectangle over the annotated regions (Fig. 12). Although "VideoBuddy" is still a semiautomatic video annotation tool, the automatically extracted "scene tree" provides valuable information for video navigation and facilitates the annotation of the video content. With guidance of the "scene tree", the user can annotate the video more efficiently with "VideoBuddy".

### 5.3. Motion estimation

In this subsection, we evaluate the proposed algorithm for motion estimation. We consider video sequences that are compressed using the MPEG-2 standard. Fig. 13 shows four examples of such videos. Each row consists of the key frame in a particular shot, camera motion, object motion and total motion. The motion information contained in the $CM$, $OM$ and $TM$ matrices are represented as monochrome images for visualization purposes; the brighter pixels indicate higher motion activity. The first row in Fig. 13 shows the shot of a news anchor person in which the camera is stationery. This is reflected in the black $CM$ image while the motion of the object (face of the person) is clear in the $OM$ image. Thus the total motion consists essentially of the object motion only. Similarly in the second row which shows a scene with a moving camera while the objects in the scene are stationery, the $OM$ image is black (indicating no motion) while the $CM$ image is uniformly bright. The third row illustrates the case of both object motion and camera motion contributing to the total motion in the shot from a soccer game. We can clearly see the zooming in of the camera in the $CM$ image while object motion is indicated by some bright pixels in the $OM$ image. Finally, we show a shot of the camera tracking a person and its corresponding motion images which are all mostly bright implying that the motion contains both camera as well as object motion. We see that the proposed motion estimation algorithm has been able to extract the motion information quite well.

### 5.4. Motion-based retrieval

We noted earlier that the video retrieval process is implicitly contained within the process of browsing the adaptive scene tree. However, the retrieval method proposed in this paper can also be considered as a 'standalone' process. To evaluate the performance of our proposed method for motion indexing and retrieval, we build a database consisting of video shots using the SBD algorithm

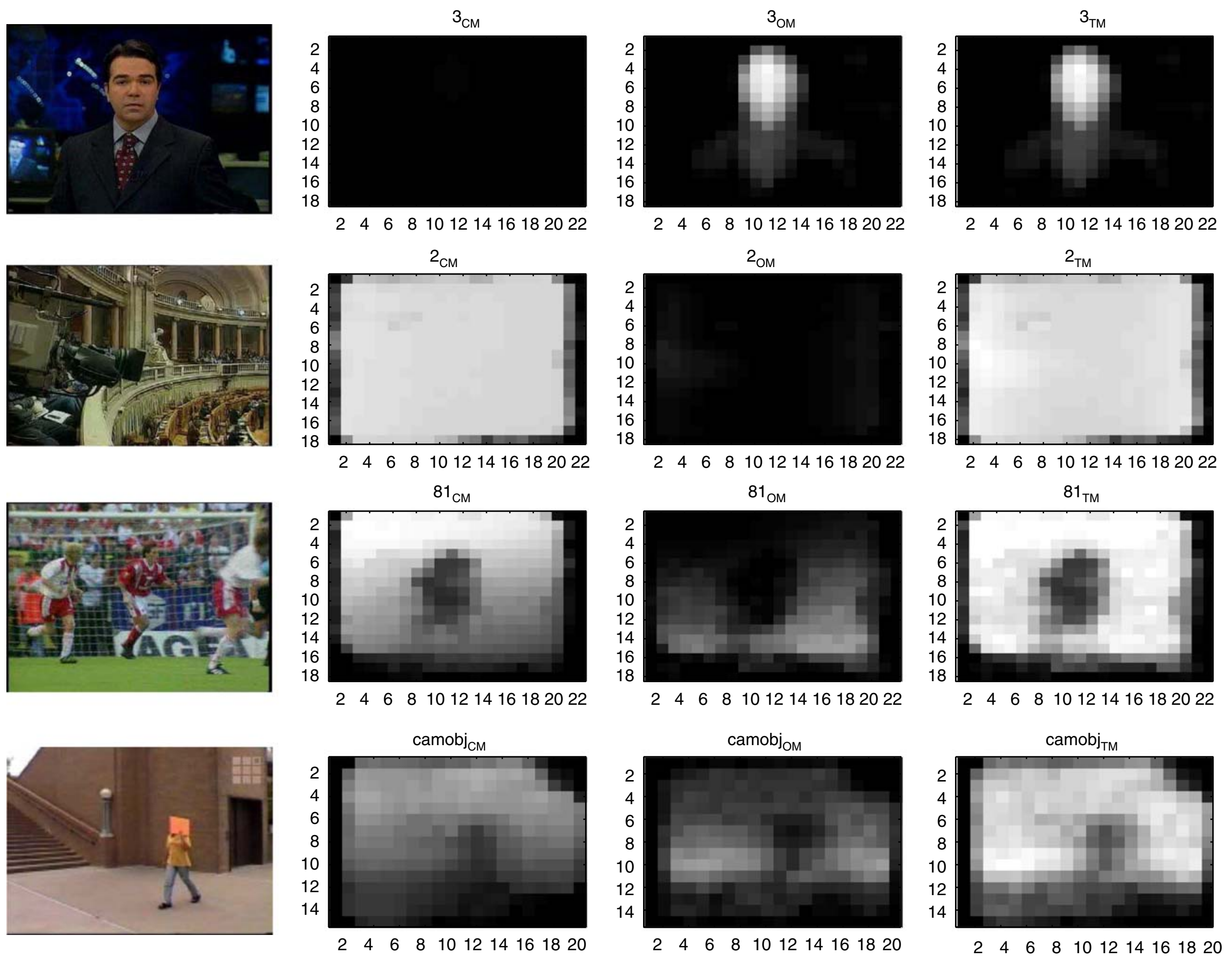

Fig. 13. Motion estimation. Each row consists of a key frame of the shot, the *CM* image, the *OM* image and the *TM* image of news anchor person (first row), gallery (second row), soccer (third row) and moving person (fourth row).

on the 643 video sequences of the MPEG-7 test set. The lengths of the video clips range from 5 to 30 s.

Using the *CM*, *OM* and *TM* matrices as indices and the video similarity measure developed in Section 4, we retrieve the top *N* video shots from the database. In Fig. 14, we show the top three results of the query shown in the first column. We compare the retrieval results when only *TM* is used with the case when all the three motion matrices, viz., *CM*, *OM* and *TM*, are used to retrieve. Fig. 14(a) shows the retrieval results with the motion feature extracted from *TM* matrix only, while Fig. 14(b) shows the retrieval result when *CM*, *OM* and *TM* matrices are considered. The first columns of Fig. 14(a) and (b) are the key frames from the query video shots. The second, third and fourth columns of Fig. 14(a) and (b) are the first, second and third retrieval results. From these examples, we observe that both models retrieve video clips which have similar motion content. However, we see that the video similarity model with *CM*, *OM* and *TM* matrices gives better retrieval result in both the motion and semantic sense than the model that uses only *TM* matrix.

We randomly choose 50 video shots from the video database as queries and retrieved the top 30 videos. *Precision* is used as the measure for performance, which is defined as the ratio of the number of relevant shots retrieved to the total number of shots retrieved. We also compare the efficiency of the proposed features with *PCA* + motion vector features used in [45]. The frame size is 320 by 240. The motion vector

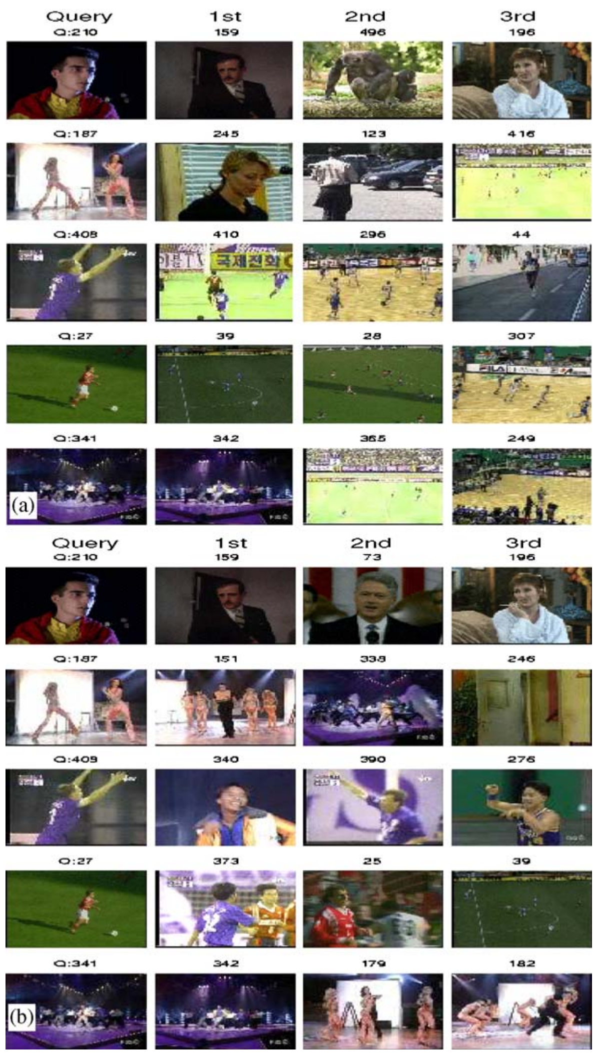

Fig. 14. Motion-based video retrieval. (a) Five example query results using *TM* matrix. (b) The same five query results using *CM*, *OM* and *TM* matrices.

field size is 20 by 15. Since we are using projection vector of the motion vector field, the actual dimension of our feature vector is $20 + 15 = 35$, the dimension of the $PCA$ + motion vector feature [45] is set to 35 in the comparison. We plot *precision* as a function of the number of retrieved video clips are shown in Fig. 15. The average precision (computed as average of the precision with the number of top return video clips varying from 1 to 30) when $CM$, $OM$ and $TM$ are considered is 91% compared to 85% when only $TM$ is used as an index and 87% when $PCA + MV$ feature is used as an index. The performance of the features is $TM + OM + CM > MV + PCA > TM$ only.

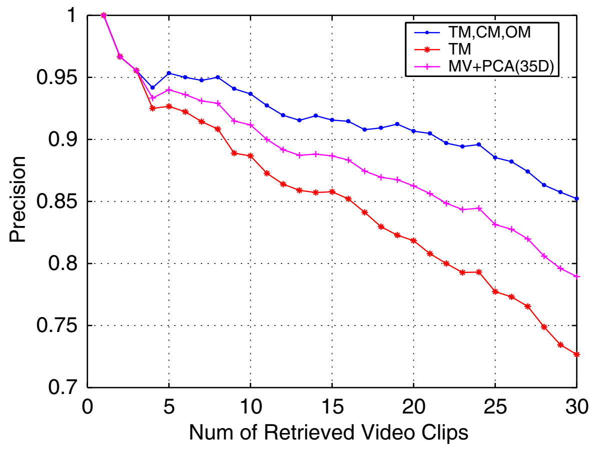

Fig. 15. Precision as a function of the number of video clips retrieved.

## 6. Conclusion

In this paper, we have presented a fully automatic content-based approach to organizing and indexing *compressed* video data. The three main steps in our approach are (i) shot boundary detection using motion prediction information of MPEG-2, (ii) development of a browsing hierarchy called adaptive scene tree and (iii) video indexing based on camera motion, object motion and total motion. The SBD algorithm can detect both abrupt cuts and gradual transitions. Unlike existing schemes for building browsing hierarchies, our technique builds a scene tree automatically from the visual content of the video. The size and shape of the tree reflect the semantic complexity of the video sequence. A video similarity measure based on the $CM$, $OM$ and $TM$ matrices is shown to give good retrieval results. The estimation of these matrices using the affine parameter model is also shown to perform well.

The hunt for highly discriminating features is a continuous one. From our study, we find that the $CM$, $OM$ and $TM$ matrices have excellent discriminatory characteristics. However, during computation of video similarity, the conversion of the matrices to a one dimensional vector might result in loss of valuable information. A more elegant and robust use of these matrices is one of the future directions for this research. The proposed scene tree is integrated into the MPEG-7 standard description scheme to generate a MPEG-7 compliant XML metadata. In the MDS of MPEG-7, there is a descriptor to describe the temporal segment of the video. We use this segment descriptor to represent the scene tree since each node in the scene tree is a video segment. To check the effectiveness of the scene tree building algorithm, it will be useful to conduct user studies in the form of presenting the summarized video to several users and asking them to narrate the story as they understand it. However, it remains to be seen if a meaningful scene tree can be built for sports videos.

## References

[1] International Organization for Standardization, Overview of the MPEG-7 Standard, ISO/IEC/JTC1/SC29/WG11 N4031 Edition, March 2001.

[2] International Organization for Standardization, MPEG21 Overview (CODING OF MOVING PICTURES AND AUDIO), ISO/IEC JTC1/SC29/WG11/N4318 Edition, July 2001.

[3] ITU-T, Video Coding for Low Bitrate Communication ITU-T Recommendation, H.263 Edition, February 1998.

[4] ITU-T, Joint Final Committee Draft (JFCD) of Joint Video Specification (ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC), H.264 Edition, July 2002.

[5] E. Katz, The Film Encyclopedia, second ed., Harper Collins, New York, 1994.

[6] J.R. Kender, B.-L. Yeo, Video scene segmentation via continuous video coherence, in: Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, Santa Barbara, CA, 1998, pp. 367–373.

[7] B.T. Truong, S. Vekatesh, C. Dorai, Scene extraction in motion picture, IEEE Trans. Circuits Syst. Video Technol. 13 (1) (2003) 5–15.

[8] M.A. Robertson, R.L. Stevenson, Restoration of compressed video using temporal information, in: Proceedings of SPIE Conference on Visual Communications and Image Processing, vol. 4310, San Jose, CA, 2001, pp. 21–29.

[9] B. Gunturk, Y. Altunbasak, R. Mersereau, Super-resolution reconstruction of compressed video using transform-domain statistics, IEEE Trans. Image Process. 13 (1) (2004) 33–43.

[10] D. Schonfeld, D. Lelescu, VORTEX: video retrieval and tracking from compressed multimedia databases—multiple object tracking from MPEG-2 bitstream, J. Visual Commun. Image Representation 11 (2000) 154–182 (special issue on Multimedia Database Management).

[11] H. Wang, A. Divakaran, A. Vetro, S.-F. Chang, H. Sun, Survey of compressed-domain features used in audio-visual indexing and analysis, J. Visual Commun. Image Representation 14 (2) (2003) 50–183.

[12] H. Zhang, H. Liu, A hierarchical organization scheme for video data, Pattern Recognition 35 (2002) 2381–2387.

[13] A. Hanjalic, Shot-boundary detection: unraveled and resolved?, IEEE Trans. Circuits Syst. Video Technol. 12 (2) (2002) 90–105.

[14] R.M. Ford, C. Robson, D. Temple, M. Gerlach, Metrics for shot boundary detection in digital video sequences, Multimedia Syst. 8 (1) (2000) 37–46.

[15] R. Lienhart, Comparison of automatic shot boundary detection algorithms, in: Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases VII, vol. 3656, San Jose, CA, 1999, pp. 290–301.

[16] H. Zhang, A. Kankanhalli, S. Smoliar, Automatic partitioning of full-motion video, Multimedia Syst. 1 (1993) 10–28.

[17] I.K. Sethi, N.V. Patel, Statistical approach to scene change detection, in: Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases, vol. 2420, San Jose, CA, 1995, pp. 329–338.

[18] C.-C. Lo, S.-J. Wang, A histogram-based moment-preserving clustering algorithm for video segmentation, Pattern Recognition Lett. 24 (14) (2003) 2209–2218.

[19] B.L. Yeo, B. Liu, Rapid scene analysis on compressed video, IEEE Trans. Circuits Syst. Video Technol. 5 (1995) 544–553.

[20] R. Zabih, J. Miller, K. Mai, A feature-based algorithm for detecting and classifying scene breaks, in: Proceedings of ACM Multimedia 95, vol. 1, San Francisco, CA, 1995, pp. 10–28.

[21] W.J. Heng, K.N. Ngan, An object-based shot boundary detection using edge tracing and tracking, J. Visual Commun. Image Representation 12 (3) (2001) 217–239.

[22] N.V. Patel, I.K. Sethi, Video shot detection and characterization for video databases, Pattern Recognition 30 (1997) 607–625.

[23] J. Feng, K.-T. Lo, H. Mehrpour, Scene change detection algorithm for MPEG video sequence, in: Proceedings of IEEE International Conference on Image Processing, vol. 1, 1996, pp. 821–824.

[24] J. Meng, Y. Juan, S.-F. Chang, Scene change detection in a MPEG video sequence, in: Proceedings of SPIE Conference on Multimedia Computing and Networking, vol. 2417, San Jose, CA, 1995, pp. 180–191.

[25] R. Brunelli, O. Mich, C.M. Modena, A survey on the automatic indexing of video data, J. Visual Commun. Image Representation 10 (2) (1999) 78–112.

[26] W. Heng, K. Ngan, Post shot boundary detection technique: transition type-independent shot boundary refinement, IEEE Trans. Multimedia 4 (4) (2002) 434–445.

[27] H. Yi, D. Rajan, L.-T. Chia, A unified approach to detection of shot boundaries and subshots in compressed video, in: Proceedings of IEEE International Conference on Image Processing, Barcelona, Spain, 2003, pp. 10005–10008.

[28] B.G. Haskell, A. Puri, A.N. Netravali, Digital Video: An Introduction to MPEG-2, Chapman & Hall, New York, 1997.

[29] A. Hanjalic, R.L. Lagendijk, J. Biemond, Automated segmentation of movies into logical story units, IEEE Trans. Circuits Syst. Video Technol. 9 (4) (1999) 580–588.

[30] A. Daniel, Grammar of the Film Language, Focal Press, London, 1976.

[31] C. Saraceno, R. Leonardi, Identification of story units in audio-visual sequences by joint audio and video processing, in: Proceeding of International Conference on Image Processing, Chicago, IL, USA, 1998, pp. 358–362.

[32] Z. Rasheed, M. Shah, Scene boundary detection in hollywood movies and TV show, in: Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, Madison, WI, 2003, pp. 343–348.

[33] J. Nam, A.H. Tewfik, Combined audio and visual streams analysis for video sequence segmentation, in: Proceedings of ICASSP-97, vol. 4, Munich, Germany, 1997, pp. 2665–2668.

[34] J. Oh, K.A. Hua, An efficient and cost-effective technique for browsing and indexing large video databases, in:

Proceedings of 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, 2000, pp. 415–426.

[35] B. Manjunath, P. Salembier, T. Sikora, Introduction to MPEG-7: Multimedia Content Description Interface, Wiley, New York, 2002.

[36] G. Robertson, J. Machinlay, S. Card, Cone trees: animated 3D visualization of hierarchical information, in: Proceedings of ACM SIGCHI Conference on Human Factors in Computing System'91, 1991.

[37] S. Smoliar, H. Zhang, Content-based video indexing and retrieval, IEEE Trans. Multimedia 1 (1994) 62–72.

[38] Y. Deng, B.S. Manjunath, C. Kenney, M.S. Moore, H. Shin, An efficient color representation for image retrieval, IEEE Trans. Image Process. 10 (1) (2001) 140–147.

[39] B.S. Manjunath, W.Y. Ma, Texture features for browsing and retrieval of image data, IEEE Trans. Pattern Anal. Mach. Intell. 18 (8) (1996) 837–842.

[40] Y. Rui, A.C. She, T.S. Huang, Modified Fourier descriptors for shape representation a practical approach, in: Proceedings of First International Workshop on Image Databases and Multimedia Search, 1996.

[41] D.K. Park, Y.S. Jeon, C.S. Won, Efficient use of local edge histogram descriptor, in: Proceedings of the ACM Workshops on Multimedia, Los Angeles, California, USA, 2000, pp. 51–54.

[42] C.-W. Ngo, T.-C. Pong, H.-J. Zhang, On clustering and retrieval of video shots through temporal slices analysis, IEEE Trans. Multimedia 4 (4) (2002) 446–458.

[43] S.F. Chang, W. Chen, H.J. Meng, H. Sundaram, D. Zhong, A fully automatic content-based video search engine supporting multiple object spatio-temporal queries, IEEE Trans. Circuit Syst. Video Technol. 8 (1998) 602–615.

[44] R. Fablet, P. Bouthemy, P. Perez, Statistical motion-based video indexing and retrieval, in: International Conference on Content Based Multimedia Information Access, 2000, pp. 602–619.

[45] E. Sahouria, A. Zakhor, Content analysis of video using principal components, IEEE Trans. Circuit System Video Technol. 9 (1999) 1290–1298.

[46] MPEG, Multimedia content description interface-part 8: extraction and use of MPEG-7 descriptors, ISO/IEC 15938-8:2002, 2002.