Efficient and Cost-effective Techniques for Browsing and Indexing Large Video Databases*

JungHwan Oh *Kien A. Hua*

Computer Science Program, School of EECS
University of Central Florida, Orlando, FL 32816-2362
E-mail: {*oh, kienhua*}*@cs.ucf.edu*

Abstract

We present in this paper a fully automatic content-based approach to organizing and indexing video data. Our methodology involves three steps:

- **Step 1**: We segment each video into shots using a Camera-Tracking technique. This process also extracts the feature vector for each shot, which consists of two statistical variances Var^{BA} and Var^{OA}. These values capture how much things are changing in the background and foreground areas of the video shot.
- **Step 2**: For each video, We apply a fully automatic method to build a browsing hierarchy using the shots identified in Step 1.
- **Step 3**: Using the Var^{BA} and Var^{OA} values obtained in Step 1, we build an index table to support a variance-based video similarity model. That is, video scenes/shots are retrieved based on given values of Var^{BA} and Var^{OA}.

The above three inter-related techniques offer an integrated framework for modeling, browsing, and searching large video databases. Our experimental results indicate that they have many advantages over existing methods.

KEYWORDS: Shot detection, Video indexing, Video browsing, Video similarity model, Video retrieval.

1 Introduction

With the rapid advances in data compression and networking technology, video has become an inseparable part of many important applications such as digital libraries, distance learning, public information systems, electronic commerce, movies on demand, just to name a few. The proliferation of video data has led to a significant body of research on techniques for *video database management systems* (VDBMSs) [1]. In general, organizing and managing video data is much more complex than managing text and numbers due to the enormous size of video files and their semantically rich contents. In particular, *content-based browsing* and *content-based indexing* techniques are essential. It should be possible for users to browse video materials in a non-sequential manner and to retrieve relevant video data efficiently based on their contents.

In a conventional (i.e., relational) database management system, the *tuple* is the basic structural element for retrieval, as well as for data entry. This is not the case for VDBMSs. For most video applications, video clips are convenient units for data entry. However, since an entire video stream is too coarse as a level of abstraction, it is generally more beneficial to store video as a sequence of shots to facilitate information retrieval. This requirement calls for techniques to segment videos into *shots* which are defined as a collection of frames recorded from a single camera operation. This process is referred to as *shot boundary detection* (SBD).

Existing SBD techniques require many input parameters which are hard to determine but have a significant influence on the quality of the result. A recent study [2] found that techniques using color histograms [3, 4, 5, 6] need at least three threshold values, and their accuracy varies from 20% to 80% depending on those values. At least six different threshold values are necessary for another technique using edge change ratio [7]. Again, these values must be chosen properly to get satisfactory results [2]. In general, picking the right values for these thresholds is a difficult task because they vary greatly from video to video. These observations indicate that today's automatic SBD techniques need to be more reliable before they can be used in practice. From the perspective of an end user, a DBMS is only as good as the data it manages. A bad video shot, returned as a query result, would contain incomplete and/or extra irrelevant information. This is a problem facing today's VDBMSs. To address this issue, we propose to detect shot boundaries in a more direct way by tracking the

* This research is partially supported by the National Science Foundation grant ANI-9714591.

MOD 2000, Dallas, TX USA

camera motion through the background areas in the video. We will discuss this idea in more detail later.

A major role of a DBMS is to allow the user to deal with data in abstract terms, rather than the form in which a computer stores data. Although shot serves well as the basic unit for video abstraction, it has been recognized in many applications that *scene* is sometimes a better unit to convey the semantic meaning of the video to the viewers. To support this fact, several techniques have been proposed to merge semantically related and temporally adjacent shots into a *scene* [8, 9, 10, 11]. Similarly, it is also highly desirable to have a complete hierarchy of video content to allow the user to browse and retrieve video information at various semantic levels. Such a multi-layer abstraction makes it more convenient to reference video information and easier to comprehend its content. It also simplifies video indexing and storage organization. One such technique was presented in [12]. This scheme abstracts the video stream structure in a *compound unit, sequence, scene, shot* hierarchy. The authors define a *scene* as a set of shots that are related in time and space. Scenes that together give meaning are grouped into a *sequence*. Related sequences are assembled into a *compound unit* of arbitrary level. Other multilevel structures were considered in [13, 14, 15, 16, 17]. All these studies, however, focus on modeling issues. They attempt to design the best hierarchical structure for video representation. However, they do not provide techniques to automate the construction of these structures.

Addressing the above limitation is essential to handling large video databases. One attempt was presented in [18]. This scheme divides a video stream into multiple segments, each containing an equal number of consecutive shots. Each segment is then further divided into sub-segments. This process is repeated several times to construct a hierarchy of video content. A drawback of this approach is that only time is considered; and no visual content is used in constructing the browsing hierarchy. In contrast, video content was considered in [19, 20, 21]. These methods first construct a priori model of a particular application or domain. Such a model specifies the scene boundary characteristics, based on which the video stream can be abstracted into a structured representation. The theoretical framework of this approach is proposed in [19], and has been successfully implemented for applications such as news videos [20] and TV soccer programs [21]. A disadvantage of these techniques is that they rely on explicit models. In a sense, they are application models, rather than database models. Two techniques, that do not employ models, are presented in [11, 22]. These schemes, however, focus on low-level scene construction. For instance, given that shots, groups and scenes are the structural units of a video, a 4-level video-scene-group-shot hierarchy is used for all videos in [22].

In this paper, we do not fix the height of our browsing hierarchy, called *scene tree*, in order to support a variety of videos. The shape and size of a scene tree are determined only by the semantic complexity of the video. Our scheme is based on the content of the video. Our experiments indicate that the proposed method can produce very high quality browsing structures.

To make browsing more efficient, we also introduce in this paper a variance-based video similarity model. Using this model, we build a content-based indexing mechanism to serve as an assistant to advise users on where in the appropriate scene trees to start the browsing. In this environment, each video shot is characterized as follows. We compute the average colors of the foreground and background areas of the frames in the shot, and calculate their statistical variance values. These values capture how much things are changing in the video shot. Such information can be used to build an index. To search for video data, a user can write a query to describe the impression of the degree of changes in the primary video segment. Our experiments indicate that this simple query model is very effective in supporting browsing environment. We will discuss this technique in more detail.

In summary, we present in this paper a fully automatic content-based technique for organizing and indexing video data. Our contributions are as follows:

1. We address the reliability problem facing today's video data segmentation techniques by introducing a camera-tracking method.

2. We fully automate the construction of browsing hierarchies. Our method is general purpose, and is suitable for all videos.

3. We provide a content-based indexing mechanism to make browsing more efficient.

The above three techniques are inter-related. They offer an integrated framework for modeling, browsing, and searching large video databases.

The remainder of this paper is organized as follows. We present our SBD technique [23], and discuss the extensions required to support our browsing and indexing mechanisms in Section 2. The procedure for building scene trees is described in details in Section 3. In Section 4, we discuss the content-based indexing technique for video browsing. The experimental results are examined in Section 5. Finally, we give our concluding remarks in Section 6.

2 A Camera Tracking Technique for SBD and Its Extension

To make the paper self-contained, we first describe our SBD technique [23]. We then extend it to include new features required by our browsing and indexing techniques.

2.1 A Camera Tracking Approach to Shot Boundary Detection

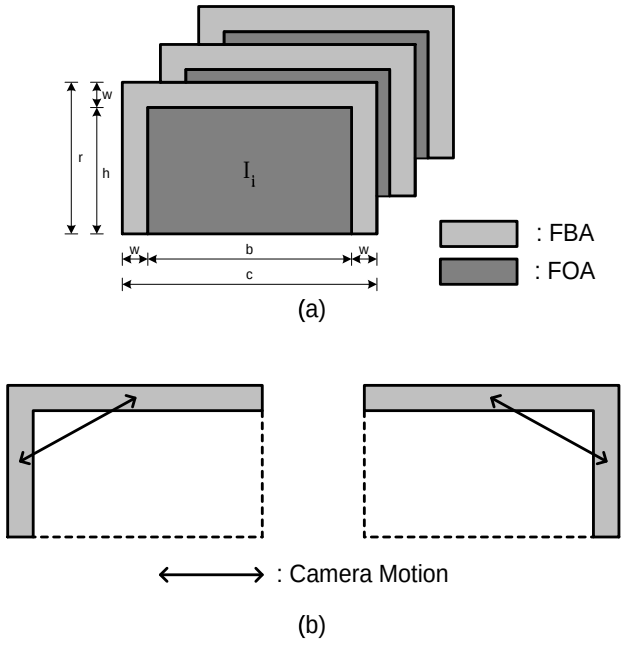

Figure 1: Background Area

Since a shot is made from one camera operation, tracking the camera motion is the most direct way to identify shot boundaries. This can be achieved by tracking the background areas in the video frames as follows. We define a *fixed background area* (FBA) for all frames as illustrated by the lightly shaded areas in Figure 1(a). The rationale for the ⊓ shape of the FBA is as follows:

- The bottom part of a frame is usually part of some object(s).
- The top bar cover any horizontal camera motion.
- The two columns cover any vertical camera motion.
- The combination of the top bar and the left column can track any camera motion in one diagonal direction. The other diagonal direction is covered by the combination of the top bar and the right column. These two properties are illustrated in Figure 1(b).

The above properties suggest that we can detect a shot boundary by determining if two consecutive frames share any part of their FBAs. This requires comparing each part of one FBA against every part of the other FBA. To make this comparison more efficient, we rotate the two vertical columns of each ⊓ shape FBA outward to form a *transformed background area* (TBA) as illustrated in Figure 2. From each TBA, which is a two-dimensional array of pixels, we compute its *signature* and *sign* by applying a modified version of the image reduction technique, called *Gaussian Pyramid* [24]. The idea of 'Gaussian Pyramid' was originally introduced for reducing an image to a smaller size. We use this technique to reduce a two-dimensional TBA into a single line of pixels (called *signature*) and eventually a single pixel (called *sign*). The complexity of this procedure is $O(2^{\log(m+1)})$, which is actually $O(m)$, where m is the number of pixels involved. The interested reader is referred to [23] for the details. We illustrate this procedure in Figure 3. It shows a 13×5 TBA being reduced in multiple steps. First, the five pixels in each column are reduced to one pixel to give one line of 13 pixels, which is used as the signature. This signature is further reduced to the *sign* denoted by $sign_i^{BA}$. The superscript and subscript indicate that this is the sign of the *background area* of some frame i. We note that this rather small TBA is only illustrative. We will discuss how to determine the TBA shortly.

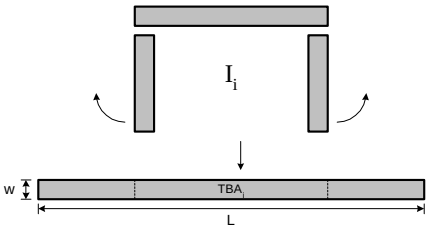

Figure 2: Shape Transformation of FBA

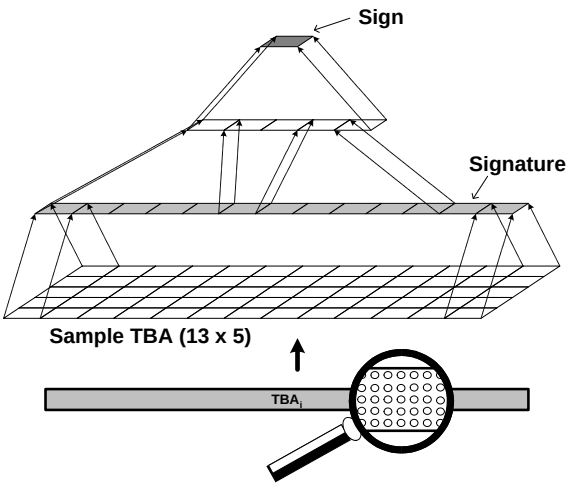

Figure 3: Computation of Signature and Sign

We use the signs and signatures to detect shot boundaries as illustrated in Figure 4. The first two stages are quick-and-dirty tests used to quickly eliminate the easy cases. Only when these two tests fail, we need to track the background in Stage 3 by shifting the two signatures, of the two frames under test, toward each other one pixel at a time. For each shift, we compare the overlapping pixels to determine the longest run of matching pixels. A running maximum is maintained for these matching scores. In the end, this maximum value indicates how much the two images share the common background. If the score is larger than a certain threshold, the two video frames are determined to be in the same shot.

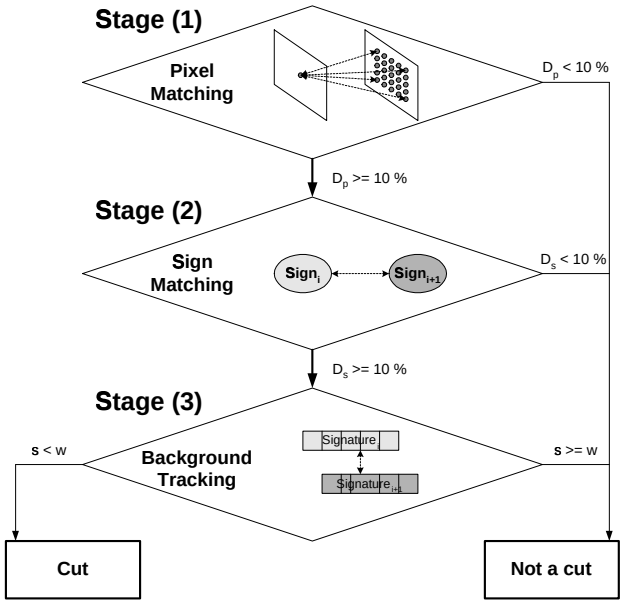

Figure 4: Shot Boundary Detection Procedure

2.2 Extension to the Camera Tracking Technique

We define the *fixed object area* (FOA) as the foreground area of a video frame, where most primary objects appear. This area is illustrated in Figure 1 as the darkly shaded region of a video frame. To facilitate our indexing scheme, we need to reduce the FOA of each frame i to one pixel. That is, we want to compute its sign, $sign_i^{OA}$, where the superscript indicates that this sign is for an FOA. This parameter can be obtained using the Gaussian Pyramid as in $sign_i^{BA}$. This computation requires the dimensions of the FOA. Given r and c as the dimensions of the video frame (see Figure 1), we discuss the procedure for determining the dimensions of TBA and FOA as follows.

Let the dimensions of FOA be h and b, and those of TBA be w and L as illustrated in Figure 1. We first estimate these parameters as h', b', w', and L', respectively. We choose w' to be 10% of the width of the video frame, i.e., $w' = \lfloor \frac{c}{10} \rfloor$. This value was determined empirically using our video clips. They show that this value of w' results in TBAs and FOAs which cover the background and foreground areas, respectively, very well. Using these w', we can compute the other estimates as follows: $b' = c - 2 \cdot w'$, $h' = r - w'$, and $L' = c + 2 \cdot h'$.

In order to apply the Gaussian Pyramid technique, the dimensions of TBA and FOA must be in the *size set* {1, 5, 13, 29, 61, 125, ...}. This is due to the fact that this technique reduces five pixels to one pixel, 13 pixels to five, 29 pixels to 13, and so on. In general, the jth element (s_j) in this size set is computed as follows:

$$s_j = 1 + \sum_{i=2}^{j} 2^i \quad for \quad j = 1, 2, 3, ... \tag{1}$$

Using this size set, the proper value for w is the value in the size set, which is nearest to w'. This nearest number can be determined as follows. We first compute $j = 2 + \left\lfloor \log_2(\frac{w'+3}{6}) \right\rfloor$. Substituting this value of j into Equation (1) gives us the desired value for w. Similarly, we can compute L, h, and b. This approximation scheme is illustrated in Table 1. As an example, let $c = 160$. We have $w' = \lfloor \frac{160}{10} \rfloor = 16$. The corresponding j value is 3. Substituting j into Equation (1) gives us 13 as the proper value for w.

h', b', w' or L'	Nearest value	h, b, w or L
1, 2	1	1
3, 4, ..., 8	5	5
9, 10, ..., 20	13	13
21, 22, ..., 44	29	29
45, 46, ..., 92	61	61
...	...	...

Table 1: Approximate the dimensions using the nearest value from the size set.

In this section, we have described the computation of the two sign values $sign_i^{BA}$ and $sign_i^{OA}$, and provided the procedure to determine the video shots. In the next two sections, we will discuss how these shots and signs are used to build browsing hierarchies and index structures for video databases.

3 Building Scene Trees for Non-linear Browsing

Video data are often accessed in an exploring or browsing mode. Browsing a video using VCR like functions (i.e., fast-forward or fast-reverse) [25], however, is tedious and time consuming. A hierarchical abstraction

allowing nonlinear browsing is desirable. Today's techniques for automatic construction of such structures, however, have many limitations. They rely on explicit models, focus only on the construction of low-level scenes, or ignore the content of the video. We discuss in this section our Scene Tree approach which addresses all these drawbacks.

In order to automate the tree construction process, we base our approach on the visual content of the video instead of human perception. First, we obtain the video shots using our camera-tracking SBD method discussed in the last section. We then group adjacent shots that are related (i.e., sharing similar backgrounds) into a *scene*. Similarly, scenes with related shots are considered related and can be assembled into a higher-level scene of arbitrary level. We discuss the details of this strategy and give an example in the following subsections.

3.1 Scene Tree Construction Algorithm

Let A and B be two shots with $|A|$ and $|B|$ frames, respectively. The algorithm to determine if they are related is as follows.

1. Set $i \leftarrow 1$, $j \leftarrow 1$.
2. Compute the difference D_s of $Sign_i^{BA}$ of shot A and $Sign_j^{BA}$ of shot B using the following equation. We use the number 256 since in our RGB space red, green and blue colors range from 0 to 255

$$D_s = \left(\frac{max.\ difference\ in\ Sign^{BA}s}{256} \right) \times 100(\%) \quad (2)$$

3. If D_s is less than 10%, then stop and return that the two shots are related; otherwise, go to the next step.
4. Set $i \leftarrow i + 1$.
   - If $i > |A|$, then stop and return that two shots are not related; otherwise, set $j \leftarrow j + 1$.
   - If $j > |B|$, then set $j \leftarrow 1$.
5. Go to Step (2).

For convenience, We will refer to this algorithm as *RELATIONSHIP*. It can be used in the following procedure to construct a browsing hierarchy, called *scene tree*, as follows.

1. A scene node SN_i^0 in the lowest level (i.e., level 0) of scene tree is created 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. Set $i \leftarrow 3$.
3. Apply algorithm *RELATIONSHIP* to compare *shot#i* with each of the shots *shot#(i-2)*, $\cdots$, *shot#1* (in descending order). This sequence of comparisons stops when a related shot, say *shot#j*, is identified. If no related shot is found, we create a new empty node, connect it as a parent node to SN_i^0, and proceed to Step 5.
4. We consider SN_{i-1}^0 and SN_j^0. Three scenarios can happen:
   - 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.
   - If SN_{i-1}^0 and SN_j^0 share an ancestor node, we connect SN_i^0 to this ancestor node.
   - 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 connect the current oldest ancestors of SN_{i-1}^0 and SN_j^0 to a new empty node as their parent node.
5. If there are more shots, we set $i \leftarrow i + 1$, and go to step 3. Otherwise, we 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, we select from the corresponding shot the most "repetitive" frame as its *representative frame*, i.e., this frame shares the same sign with the most number of frames in the shot. We then traverse all the nodes in the scene tree, level by level, starting from the bottom. For each empty node visited, we identify the child node, say SN_m^c, which contains *shot#m* which has the longest sequence of frames with the same $Sign^{BA}$ value. We rename this empty node as SN_m^{c+1}, and assign the representative frame of SN_m^c to SN_m^{c+1}.

We note that each scene node contains a representative frame or a pointer to that frame for future use such as browsing or navigating. The criterion for selecting a representative frame from a shot is to find the most frequent image. If more than one such image is found, we can choose the temporally earliest one. As an example, let us assume that *shot#5* has 20 frames and the $Sign^{BA}$ value of each frame is as shown in Table 2. Since $Sign^{BA}$ is actually a pixel, it has three numerical values for the three colors, red, green and blue. In this case, we use frame 1 as the representative frame for *shot#5* because this frame corresponds to an image with the longest sequence of frames with the same $Sign^{BA}$ values (i.e., 219, 152, 142). Although, the sequence corresponding to frames 15 to 20 also has the same sequence length, frame 15 is not selected because

it appears later in the shot. Instead of having only one representative frame per scene, we can also use $g(s)$ most repetitive representative frames for scenes with s shots to better convey their larger content, where g is some function of s.

Frames	Sign		
	Red	Green	Blue
No. 1	219	152	142
No. 2	219	152	142
No. 3	219	152	142
No. 4	219	152	142
No. 5	219	152	142
No. 6	219	152	142
No. 7	226	164	172
No. 8	226	164	172
No. 9	213	149	134
No.10	213	149	134
No.11	213	149	134
No.12	213	149	134
No.13	200	137	123
No.14	200	137	123
No.15	228	160	149
No.16	228	160	149
No.17	228	160	149
No.18	228	160	149
No.19	228	160	149
No.20	228	160	149

Table 2: Frames in the *shot#5*

Now, let us evaluate the complexity of the two algorithms above. The complexity of *RELATIONSHIP* is $O(|A| \times |B|)$. The average computation cost, however, is much less because the algorithm stops as soon as it finds the two related scenes. Furthermore, the similarity computation is based on only one pixel (i.e., $Sign^{BA}$) of each video frame making this algorithm very efficient.

The cost of the tree construction algorithm can be derived as follows. Step 3 can be done in $O(f^2 \times n)$, where f is the number of frames, and n is the number of shots in a given video. This is because the algorithm visits every shot; and whenever a shot is visited, it is compared with every frame in the shots before it. In Step 4 and Step 6, we need to traverse a tree. It can be done in $O(log(n))$. Therefore, the whole algorithm can be completed in $O(f^2 \times n)$.

3.2 Example to explain Scene Tree

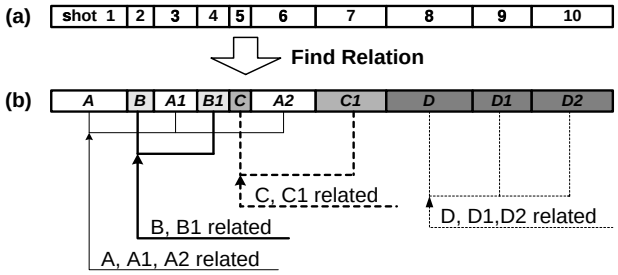

Figure 5: A video clip with ten shots

The scene tree construction algorithm is best illustrated by an example. Let us consider a video clip with ten scenes as shown in Figure 5. For convenience, we label related shots with the same prefix. For instance, *shot#1*, *shot#3* and *shot#6* are related, and are labeled as A, $A1$ and $A2$, respectively. An effective algorithm should group these shots into a longer unit at a higher level in the browsing hierarchy. Using this video clip, we illustrate our tree construction algorithm in Figure 6. The details are discussed below.

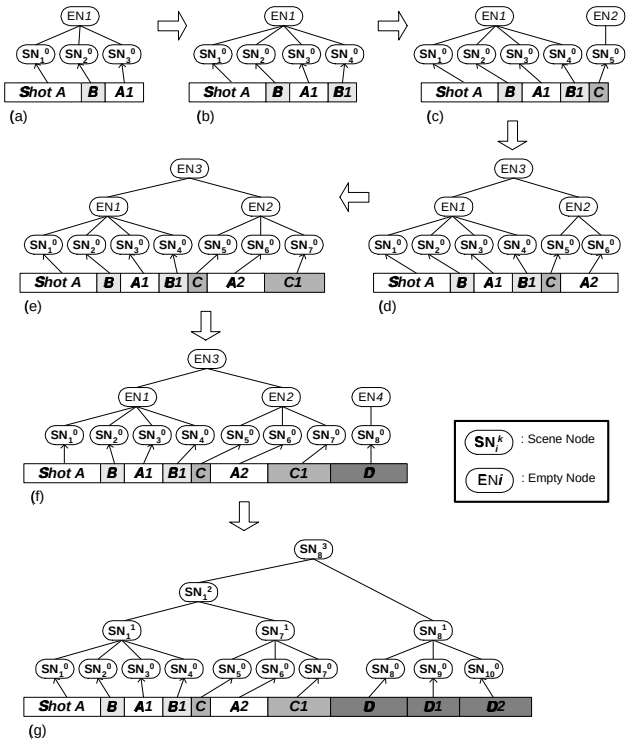

Figure 6: Scene Tree Building

- **Figure 6(a)**: We first create three scene nodes SN_1^0, SN_2^0 and SN_3^0 for *shot#1*, *shot#2* and *shot#3*, respectively. Applying algorithm *RELATIONSHIP* to *shot#3* and *shot#1*, we determine that the two shots are related. Since they are related but neither currently has a parent node, we connect them to a new empty node called $EN1$. According to our algorithm, we do not need to compare *shot#2* and *shot#3*. However, *shot#2* is connected to $EN1$ because *shot#2* is between two related nodes, *shot#3* and *shot#1*.

- **Figure 6(b)**: Applying the algorithm *RELATIONSHIP* to *shot#4* and *shot#2*, we determine that they are related. This allows us to skip the comparison between *shot#4* and *shot#1*. In this case, since SN_2^0 and SN_3^0 share the same ancestor (i.e., $EN1$), we also connect *shot#4* to $EN1$.

- **Figure 6(c)**: Comparing *shot#5* with *shot#3*, *shot#2*, and *shot#1* using *RELATIONSHIP*, we determine that *shot#5* is not related to these three

shots. We, thus create SN_5^0 for *shot#5*, and connect it to a new empty node $EN2$.

- **Figure 6(d)**: In this case, *shot#6* is determined to be related to *shot#3*. Since SN_5^0 and SN_3^0 currently do not have the same ancestor, we first connect SN_6^0 to $EN2$; and then connect $EN1$ and $EN2$ to a new empty node $EN3$ as their parent node.

- **Figure 6(e)**: In this case, *shot#7* is determined to be related to *shot#5*. Since SN_7^0 and SN_5^0 share the same ancestor node $EN2$, we simply create SN_7^0 for *shot#7* and connect this scene node to $EN2$.

- **Figure 6(f)**: This case is similar to the case of Figure 6(c). *shot#8* is not related to any previous shots. We create a new scene node SN_8^0 for *shot#8*, and connect this scene node to a new empty node $EN4$.

- **Figure 6(g)**: *shot#9* and *shot#10* are found to be related to the immediate previous node, *shot#8* and *shot#9*, respectively. In this case, according to the algorithm, both *shot#9* and *shot#10* are connected to $EN4$. Since *shot#10* is the last shot of the video clip, we create a root node, and connect all nodes which do not currently have a parent node to this root node. Now, we need to name all the empty nodes. $EN1$ is named SN_1^1 because *shot#1* contains an image which is "repeated" most frequently among all the images in the first four level-0 scenes. The superscript of "1" indicates that SN_1^1 is a scene node at level 1. As another example, $EN3$ is named SN_1^2 because *shot#1* contains an image which is "repeated" most frequently among all the images in the first seven level-0 scenes. The superscript of "2" indicates that SN_1^2 is a scene node at level 2. Similarly, we can determine the names for the other scene nodes. We note that the naming process is important because it determines the proper representative frame for each scene node, e.g., SN_7^1 indicates that this scene node should use the representative frame from *shot#7*.

In Section 5, we will show an example of a scene tree built from a real video clip.

4 Cost-effective Indexing

In this section, we first discuss how $Sign^{BA}$ and $Sign^{OA}$, generated from our SBD technique, can be used to characterize video data. We then present a video similarity model based on these two parameters.

4.1 A Simple Feature Vector for Video Data

To illustrate the concept of our techniques, we use the same example video clip in Figure 5, which has 10 shots. From this video clip, let us assume that our SBD technique generates the values of $Sign^{BA}$s and $Sign^{OA}$s for all the frames as shown in the 4th and 5th columns of Table 3, respectively. The 6th and

Shots	No. of start frame	No. of end frame	$Sign^{BA}$	$Sign^{OA}$	Var^{BA}	Var^{OA}
# 1 (A)	1	75	$Sign_1^{BA}$, ..., $Sign_{75}^{BA}$	$Sign_1^{OA}$, ..., $Sign_{75}^{OA}$	Var_A^{BA}	Var_A^{OA}
# 2 (B)	76	100	$Sign_{75}^{BA}$, ..., $Sign_{100}^{BA}$	$Sign_{75}^{OA}$, ..., $Sign_{100}^{OA}$	Var_B^{BA}	Var_B^{OA}
# 3 (A1)	101	140	$Sign_{101}^{BA}$, ..., $Sign_{140}^{BA}$	$Sign_{101}^{OA}$, ..., $Sign_{140}^{OA}$	Var_{A1}^{BA}	Var_{A1}^{OA}
# 4 (B1)	141	170	$Sign_{141}^{BA}$, ..., $Sign_{170}^{BA}$	$Sign_{141}^{OA}$, ..., $Sign_{170}^{OA}$	Var_{B1}^{BA}	Var_{B1}^{OA}
# 5 (C)	171	290	$Sign_{171}^{BA}$, ..., $Sign_{290}^{BA}$	$Sign_{171}^{OA}$, ..., $Sign_{290}^{OA}$	Var_C^{BA}	Var_C^{OA}
# 6 (A2)	291	350	$Sign_{191}^{BA}$, ..., $Sign_{350}^{BA}$	$Sign_{191}^{OA}$, ..., $Sign_{350}^{OA}$	Var_{A2}^{BA}	Var_{A2}^{OA}
# 7 (C1)	351	415	$Sign_{351}^{BA}$, ..., $Sign_{415}^{BA}$	$Sign_{351}^{OA}$, ..., $Sign_{415}^{OA}$	Var_{C1}^{BA}	Var_{C1}^{OA}
# 8 (D)	416	495	$Sign_{416}^{BA}$, ..., $Sign_{495}^{BA}$	$Sign_{416}^{OA}$, ..., $Sign_{495}^{OA}$	Var_D^{BA}	Var_D^{OA}
# 9 (D1)	496	550	$Sign_{496}^{BA}$, ..., $Sign_{550}^{BA}$	$Sign_{496}^{BA}$, ..., $Sign_{550}^{BA}$	Var_{D1}^{BA}	Var_{D1}^{OA}
#10 (D2)	551	625	$Sign_{551}^{BA}$, ..., $Sign_{625}^{BA}$	$Sign_{551}^{OA}$, ..., $Sign_{625}^{OA}$	Var_{D2}^{BA}	Var_{D2}^{OA}

Table 3: Results from Shot Boundary detection

7th columns of Table 3, which are called Var^{BA} and Var^{OA}, respectively, are computed using the following equations:

$$Var_i^{BA} = \frac{\sum_{j=k}^{l} (Sign_j^{BA} - \overline{Sign_i^{BA}})^2}{l-k} \quad (3)$$

where k and l are the first and last frames of the ith shot, respectively. $\overline{Sign_i^{BA}}$ is the mean value for all the signs, and is computed as follows:

$$\overline{Sign_i^{BA}} = \frac{\sum_{j=k}^{l} Sign_j^{BA}}{l-k+1} \quad (4)$$

Similarly, we can compute Var_i^{OA} as follows:

$$Var_i^{OA} = \frac{\sum_{j=k}^{l} (Sign_j^{OA} - \overline{Sign_i^{OA}})^2}{l-k} \quad (5)$$

$$\overline{Sign_i^{OA}} = \frac{\sum_{j=k}^{l} Sign_j^{OA}}{l-k+1} \quad (6)$$

We note that Var^{BA} and Var^{OA} are the statistical variances of $Sign^{BA}$s and $Sign^{OA}$s, respectively, within a shot. These variance values measure the degree of changes in the content of the background or object area of a shot. They have the following properties:

- **If Var^{BA} is zero**, it obviously means that there is no change in $Sign^{BA}$s. In other words, the background is fixed in this shot.

- **If Var^{OA} is zero**, it means that there is no change in $Sign^{OA}$s. In other words, there is no change in the object area.

- **If either value is not zero**, there are changes in the background or object area. A larger variance indicates a higher degree of changes in the respective area.

Thus, Var^{BA} and Var^{OA} capture the spatio-temporal semantics of the video shot. We can use them to characterize a video shot, much like average color, color distribution, etc. are used to characterize images.

Based on the above discussions, we may be asked if just two values, Var^{BA} and Var^{OA}, are enough to capture the various contents of diverse kinds of videos. To answer this concern, we note that videos in a digital library are typically classified by their genre and form. 133 genres and 35 forms are listed in [26]. These genres include 'adaptation', 'adventure', 'biographical', 'comedy', 'historical', 'medical', 'musical', 'romance', 'western', etc. Some examples of the 35 forms are 'animation', 'feature', 'television mini-series', and 'television series'. To classify a video, all appropriate genres and forms are selected from this list. For examples, the movie 'Brave Heart' is classified as 'adventure and biographical feature'; and 'Dr. Zhivago' is classified as 'adaptation, historical, and romance feature'. In total, there are at least 4,655 (133×35) possible categories of videos. If we assume that video retrieval is performed within one of these 4,655 classes, our indexing scheme using Var^{BA} and Var^{OA} should be enough to characterize contents of a shot. We will show experimental results in the next section to substantiate this claim.

Unlike methods which extract keywords or keyframe(s) from videos, our method extracts (Var^{BA} and Var^{OA}) for indexing and retrieval. The advantage of this approach is that it can be fully automated. Furthermore, it is not reliance on any domain knowledge.

4.2 A Video Similarity Model

To facilitate video retrieval, we build an index table as shown in Table 4. It shows the index information relevant to two video clips, 'Simon Birch' and 'Wag the Dog.' For convenience, we denote the last column as D^v. That is $D^v = \sqrt{Var^{BA}} - \sqrt{Var^{OA}}$.

A	B	C	D	E	F
1	1	19	20.49	18.05	2.45
2	20	50	24.03	11.59	12.44
3	51	59	30.11	16.07	14.04
4	60	98	22.90	25.73	-2.82
5	99	116	16.59	21.78	-5.19
6	117	153	34.23	17.81	16.42
7	154	172	18.67	19.37	-0.70
8	173	199	25.59	38.01	-12.41
9	200	205	13.10	13.97	-0.88
10	206	237	8.88	13.31	-4.43
			.		
			.		
			.		

(a) Simon Birch

A	B	C	D	E	F
1	1	8	4.40	0.89	3.52
2	9	14	2.65	21.73	-19.09
3	15	22	5.30	6.54	-1.24
4	23	33	11.42	5.84	5.58
5	34	73	7.57	20.89	-13.32
6	74	89	11.24	8.31	2.93
7	90	96	2.81	35.07	-32.26
8	97	103	11.24	7.54	3.69
9	104	116	1.88	17.23	-15.35
10	117	118	8.01	7.16	0.84
			.		
			.		
			.		

(b) Wag the Dog

A : Shot No.
B : Start Frame No.
C : End Frame No.
D : $\sqrt{\text{var}^{BA}}$
E : $\sqrt{\text{var}^{OA}}$
F : $\sqrt{\text{var}^{BA}}$ - $\sqrt{\text{var}^{OA}}$

Table 4: Index Information for the two Clips

To search for relevant shots, the user expresses the impression of how much things are changing in the background and object areas by specifying the Var_q^{BA} and Var_q^{OA} values, respectively. In response, the system computes $D_q^v = \sqrt{Var_q^{BA}} - \sqrt{Var_q^{OA}}$, and return the ID of any shot i that satisfies the following conditions:

$$(D_q^v - \alpha) \leq D_i^v \leq (D_q^v + \alpha) \quad (7)$$

$$(\sqrt{Var_q^{BA}} - \beta) \leq \sqrt{Var_i^{BA}} \leq (\sqrt{Var_q^{BA}} + \beta) \quad (8)$$

Since the impression expressed in a query is very approximate, α and β are used in the similarity computation to allow some degree of tolerance in matching video data. In our system, we set $\alpha = \beta = 1.0$. We note that another common way to handle inexact queries is to do matching on quantized data.

In general, the answer to a query does not have to be shots. Instead, the system can return the largest scenes that share the same representative frame with one of the matching shots. Using this information, the user can browse the appropriate scene trees, starting from the suggested scene nodes, to search for more specific scenes in the lower levels of the hierarchies. In a sense, this indexing mechanism makes browsing more efficient.

5 Experimental Results

Our experiments were designed to assess the following performance issues:

- Our camera tracking technique is effective for SBD.
- The algorithm, presented in Section 3, builds reliable scene trees.
- The variance values Var^{BA} and Var^{OA} make a good feature vector for video data.

We discuss our performance results in the following subsections.

5.1 Performance of Shot Boundary Detection Technique

Two parameters 'recall' and 'precision' are commonly used to evaluate the effectiveness of IR (Information Retrieval) techniques [27]. We also use these metrics in our study as follows:

- *Recall* is the ratio of the number of shot changes detected correctly over the actual number of shot changes in a given video clip.
- *Precision* is the ratio of the number of shot changes detected correctly over the total number of shot changes detected (correctly or incorrectly).

In a previous study [23], we have demonstrated that our Camera Tracking technique is significantly more accurate then traditional methods based on color histograms and edge change ratios. In the current study, we re-evaluate our technique using many more video clips. Our video clips were originally digitized in AVI format at 30 frames/second. Their resolution

Type	Name	Duration (min:sec)	Shot Changes	Recall (H_r)	Precision (H_p)
TV Programs	Silk Stalkings (Drama)	10 : 24	95	0.97	0.87
	Scooby Dog Show (Cartoon)	11 : 38	106	0.87	0.75
	Friends (Sitcom)	10 : 22	116	0.88	0.75
	Chicago Hope (Drama)	9 : 47	156	0.96	0.84
	Star Trek(Deep Space Nine)	12 : 27	111	0.78	0.81
	All My Children (Soap Opera)	5 : 44	50	0.89	0.81
	Flinstone (Cartoon)	6 : 09	48	0.89	0.84
	Jerry Springer (Talk Show)	4 : 58	107	0.77	0.82
	TV Commercials	31 : 25	967	0.95	0.93
News	National (NBC)	14 : 45	202	0.95	0.93
	Local (ABC)	30 : 27	176	0.94	0.91
Movies	Brave Heart	10 : 03	246	0.90	0.81
	ATF	11 : 52	224	0.94	0.90
	Simon Birch	11 : 08	164	0.95	0.83
	Wag the dog	11 : 01	103	0.98	0.81
Sports Events	Tennis (1999 U.S. Open)	14 : 20	114	0.91	0.90
	Mountain Bike Race	15 : 12	143	0.96	0.95
	Football	21 : 26	163	0.94	0.88
Documentaries	Today's Vietnam	10 : 29	93	0.89	0.84
	For all mankind	16 : 50	127	0.90	0.81
Music Videos	Kobe Bryant	3 : 53	53	0.86	0.78
	Alabama Song	4 : 24	65	0.89	0.84
	Total	278 : 44	3629	0.90	0.85

Table 5: Test Video Clips and Detection Results for Shot Changes

is 160 × 120 pixels. To reduce computation time, we made our test video clips by extracting frames from these originals at the rate of 3 frames/second. To design our test video set, we studied the videos used in [28, 7, 9, 10, 29, 30, 2]. From theirs, we created our set of 22 video clips. They represent six different categories as shown in Table 5. In total, this test set lasts about 4 hours and 30 minutes. It is more complete than any other test sets used in [28, 7, 9, 10, 29, 30, 2]. The details of our test video set and shot boundary detection results are given in Table 5. We observe that the recalls and the precisions are consistent with those obtained in our previous study [23].

5.2 Effectiveness of Scene Tree

In this study, we run the algorithms in Section 3 to build the scene tree for various videos. To assess the effectiveness of these algorithms, we inspected each video and evaluated the structure of the corresponding tree and its representative frames. Since it is difficult to quantify the quality of these scene trees, we show one representative tree in Figure 7. This scene tree was built from a one-minute segment of our test video clip "Friends." The story is as follows. Two women and one man are having a conversation in a restaurant, and two men come and join them. If we travel the scene tree from level 3 to level 1, and therefore browsing the video non-linearly, we can get the above story. We note that the representative frames serve well as a summary of important events in the underlying video.

5.3 Effectiveness of Var^{BA} and Var^{QA}

To demonstrate that Var^{BA} and Var^{QA} indeed capture the semantics of video data, we select arbitrary shots from our data set. For each of these shots, we compute its Var^{BA} and Var^{QA}, and use them to retrieve similar shots in the data set. If these two parameters are indeed good feature values, the shots returned should resemble some characteristics of the shot used to do the retrieval.

We show some of the experimental results in Figure 8, Figure 9 and Figure 10. In each of these figures, the upper, leftmost picture is the representative frame of the video short selected arbitrarily for the retrieval experiment. The remaining pictures are representative frames of the matching shots. The label under each picture indicates the shot and the video clip the representative frame belongs to. For instance, #12W represents the representative frame of the 12th shot of 'Wag the dog'. Due to space limitation, we show only the three most similar shots in each case. They are discussed below.

- **Figure 8** The shot (#12W) is from 'Wag the dog'. This shot is a close-up of a person who is talking. The D^v_{12} and Var^{BA}_{12} for this shot are 5.86 and 17.37, respectively, as seen in Table 4(b). The shot #102 from 'Wag the dog', and the shots #64 and #154 from 'Simon Birch' were retrieved and presented in Figure 8. The results are quite impressive in that all four shots show a close-up view of a talking person.

- **Figure 9** The shot (#33W) is from 'Wag the dog', and the content shows two people talking from some distance. The D^v_{33} and Var^{BA}_{33} for this shot are 1.46 and 9.37, respectively, as seen in Table 4(b). The shot #11 from 'Wag the dog', and the shots #93, and #108 from 'Simon Birch' were retrieved and presented in Figure 9. Again, the four shots are very similar in content. All show two people talking from some distance.

- **Figure 10** The shot (#76S) is from 'Simon Birch.' The content is a person running from the kitchen to the window. The D^v_{76} and Var^{BA}_{76} for this shot are -0.78 and 23.55, respectively, as seen in Table 4(a). The shot #87 from 'Wag the dog', and the shots #1 and #4 from 'Simon Birch' were retrieved and presented in Figure 9. Two people are riding a bike in shot #1S. In shot #4W, one person is running in the woods. In shot #87, one person is picking a book from a book shelf and walking to the living room. These shots are similar in that all show a single moving object with a changing background.

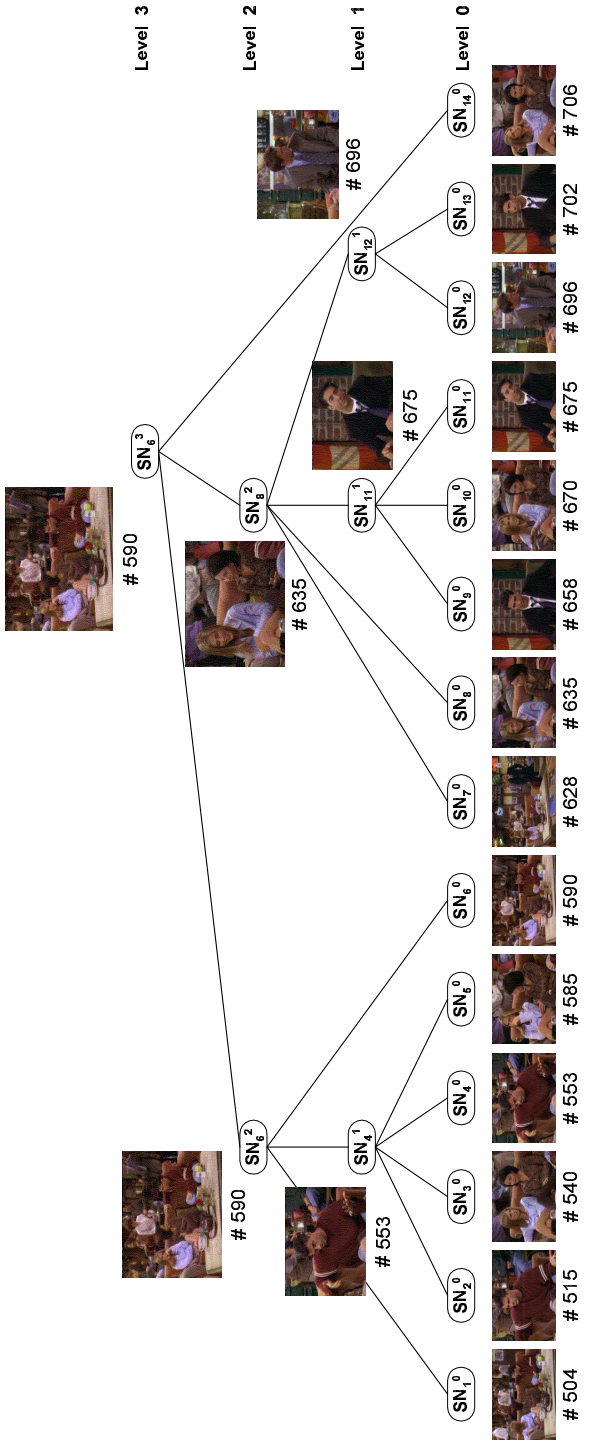

Figure 7: Scene Tree of 'Friends'

Figure 8: Shots with similar index values - Set 1.

Figure 9: Shots with similar index values - Set 2.

Figure 10: Shots with similar index values - Set 3.

6 Concluding Remarks

We have presented in this paper a fully automatic content-based approach to organizing and indexing video data. There are three steps in our methodology:

- Step 1: A Camera-Tracking Shot Boundary Detection technique is used to segment each video into basic units called shots. This step also computes the feature vector for each shot, which consists of two variances Var^{BA} and Var^{OA}. These two values capture how much things are changing in the background and foreground areas of the shot.

- Step 2: For each video, a fully automatic method is applied to the shots, identified in Step 1, to build a browsing hierarchy, called *Scene Tree*.

- Step 3: Using the Var^{BA} and Var^{OA} values obtained in Step 1, an index table is built to support a variance-based video similarity model. That is, video scenes/shots are retrieved based on given values of Var^{BA} and Var^{OA}.

Actually, the variance-based similarity model is not used to directly retrieve the video scenes/shots. Rather, it is used to determine the relevant scene nodes. With this information, the user can start the browsing from these nodes to look for more specific scenes/shots in the lower level of the hierarchy.

Comparing the proposed techniques with existing methods, we can draw the following conclusions:

- Our Camera-Tracking technique is fundamentally different from traditional methods based on pixel comparison. Since our scheme is designed around the very definition of shots, it offers unprecedented accuracy.

- Unlike existing schemes for building browsing hierarchies, which are limited to low-level entities (i.e., scenes), rely on explicit models, or do not consider the video content, our technique builds a scene tree automatically from the visual content of the video. The size and shape of our browsing structure reflect the semantic complexity of the video clip.

- Video retrieval techniques based on keywords are expensive, usually application dependent, and biased. These problems remain even if the dialog can be extracted from the video using speech recognition methods [31]. Indexing techniques based on spatio-temporal contents are available. They, however, rely on complex image processing techniques, and therefore very expensive. Our variance-based similarity model offers a simple and inexpensive approach to achieve comparable performance. It is uniquely suitable for large video databases.

We are currently investigating extensions to our variance-based similarity model to make the comparison more discriminating. We are also studying techniques to speed up the video data segmentation process.

References

[1] A. Elmagarmid, H. Jiang, A. Helal, A. Joshi, and M. Ahmed. *Video Database Systems - Issues, Products, and Applications.* Kulwer Academic Publishers, 1997.

[2] R. Lienhart. Comparison of automatic shot boundary detection algorithms. In *Proc. SPIE Vol. 3656, Storage and Retrieval for Image and Video Databases VII*, pages 290–301, San Jose, CA, January 1999.

[3] M. A. Smith and M. G. Christel. Automating the creation of a digital vidoe library. In *Proc. of ACM Multimedia '95*, pages 357–358, 1995.

[4] R. Lienhart, S. Pfeiffer, and W. Effelsberg. The moca workbench: Support for creativity in movie content analysis. In *Proc. of the IEEE Int'l Conf. on Multimedia Systems '96*, June 1996.

[5] M. Abdel-Mottaleb and N. Dimitrova. Conivas: Content-based image and video access system. In *Proc. of ACM Int'l Conf. on Multimedia*, pages 427–428, Boston, MA, November 1996.

[6] H. Yu and W. Wolf. A visual search system for video and image databases. In *Proc. IEEE Int'l Conf. on Multimedia Computing and Systems*, pages 517–524, Ottawa, Canada, June 1997.

[7] R. Zabih, J. Miller, and K. Mai. A feature-based algorithm for detecting and classifying scene breaks. In *Proc. of ACM Multimedia '95*, pages 189–200, San Francisco, CA, 1995.

[8] P. Aigrain, P. Joly, and V. Longueville. Medium knowledge-based macro-segmentation of video into sequences. In *IJCAI Workshop on Intelligent Multimedia Information Retrieval*, pages 5–14, 1995.

[9] H. Aoki, S. Shimotsuji, and O. Hori. A shot classification method of selecting effective key-frame for video browsing. In *Proc. of ACM Int'l Conf. on Multimedia*, pages 1–10, Boston, MA, November 1996.

[10] M. M. Yeung, B. Yeo, and B. Liu. Extracting story units from long programs for video browsing and navigation. In *Proc. of the IEEE Int'l Conf. on Multimedia Systems '96*, pages 296–304, Hiroshima, Japan, June 1996.

[11] D. Zhong, H. Zhang, and S-F Chang. Clustering methods for video browsing and annotation. Technical report, Columbia University, 1997.

[12] R. Hjelsvold and R. Midtstraum. Modeling and querying video data. In *Proc. of 20th Int'l Conf. on Very Large Database (VLDB '94)*, 1994.

[13] G. Davenport, T. Smith, and N. Pincever. Cinematic primitives for multimedia. In *Proc. IEEE Computer Graphics & Applications*, pages 67–74, July 1991.

[14] R. Hamakawa and J. Rekimoto. Object composition and playback models for handling multimedia data. In *Proc. of ACM Multimedia*, pages 273–281, Anaheim, CA, August 1993.

[15] R. Weiss, A. Duda, and D. Gifford. Content-based access to algebraic video. In *Proc. of IEEE Int'l Conf. Multimedia Computing and Systems*, Los Alamitos, CA, 1994.

[16] J. M. Corridoni, A. D. Bimbo, D. Lucarella, and H. Wenxue. Multi-perspective navigation of movies. *Journal of Visual Languages and Computing*, 7:445–466, July 1996.

[17] H. Jiang and A. K. Elmagarmid. Wvtdb - a semantic content-based video database system on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 10(6):947–966, 1998.

[18] H. J. Zhang, S. W. Smoliar, and J. Wu. Content-based video browsing tools. In *Proc. of IS&T/SPIE Con. on Multimedia Computing and Networking*, 1995.

[19] D. Swanberg, C. F. Shu, and R. Jain. Knowledge guided parsing in video databases. In *Proc. of SPIE Symposium on Electronic Imaging: Science and Technology*, pages 13–24, February 1993.

[20] H. Zhang and S. W. Smoliar. Developing power tools for video indexing and retrieval. In *Proc. of SPIE Storage and Retrieval for Image and Video Database*, San Jose, CA, Jan. 1994.

[21] Y. Gong, H. Chua, and X. Guo. Image indexing and retrieval based on color histogram. In *Proc. of Int'l Conf. Multimedia Modeling*, pages 115–126, Singapore, Nov. 1995.

[22] Y. Rui, T. S. Huang, and S. Mehratra. Constructing table-of-cont for videos. *ACM Multimedia Systems*, 7(5):359–368, 1999.

[23] JungHwan Oh, Kien A. Hua, and Ning Liang. A content-based scene change detection and classification technique using background tracking. In *SPIE Conf. on Multimedia Computing and Networking 2000*, San Jose, CA, Jan. 2000.

[24] P. J. Burt and E. H. Adelson. The laplacian pyramid as a compact image code. In *IEEE Transactions on Communications V COM-31*, pages 532–540, April 1983.

[25] Kien A. Hua, W. Tavanapong, and J. Wang. 2psm: An efficient framework for searching video information in a limited-bandwidth environment. *ACM Multimedia Systems*, 7(5):396–408, September 1999.

[26] B. Taves, J. Hoffman, and K. Lund. The moving image genre-form guide. In *Motion Picture/Broadcasting/Recoreded Sound Division Library of Congress*, 1997.

[27] W. B. Frakes and R. Baeza-Yates. *Information Retrieval - Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, 1992.

[28] A. Hampapur, R. Jain, and T. Weymouth. Digital video segmentation. In *Proc. of ACM Multimedia*, pages 357–364, October 1994.

[29] S. Chang, W. Chen, H. J. Meng, H. Sundaram, and D. Zhong. Videoq: An automated content based video search system using visual cues. In *ACM Proc. of the conf. on Mutimedia '97*, pages 313–324, Seattle Washington, November 1997.

[30] Y. Rui, T. S. Huang, and S. Mehratra. Exploring video structure beyond the shots. In *Proc. of 98 IEEE Conf. on Multimedia Computing and Systems*, pages 237–240, Austin Texas, June 1998.

[31] H. D. Wactlar, M. G. Christel, Y. Gong, and A. G. Hauptmann. Lessons learned from building terabyte digital video library. *Computer*, pages 66–73, February 1999.