Indexing, Query Interface and Query Processing for Venus: A Video Database System*

Tony C.T. Kuo and Arbee L.P. Chen

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
Email: alpchen@cs.nthu.edu.tw

Abstract

A content-based video database system requires new technologies to fit the new requirements and characteristics of videos. They include video modeling, query specification, video indexing, and query processing. In this paper, we present Venus, a prototype content-based video database system. Venus consists of a video manger, a query manager, and an indexing tool. The technologies developed for each component are discussed, which include class hierarchies for managing videos and content symbols, a content-based query language for video retrieval, the algorithms for semi-automatic index construction from MPEG coded videos, and the strategies for query processing.

1 Introduction

A video contains rich information in its contents. There are two types of characteristics for a video: (1) the image features (such as color, shape, and texture), and (2) the features of the content symbols (such as the spatial and temporal relationships between them). The goal of a video database is to support an efficient and friendly way for users to retrieve video data. Traditional query capabilities allow only textual and/or numerical specification. It cannot satisfy the requirements of video retrievals, and new technologies are necessary for the requirements.

The content-based retrieval is a natural way to meet the characteristics of video data. There are several key issues of a content-based video database: (1) the modeling of video data, (2) the specification of video queries, (3) the construction of video indexes, and (4) the video query processing. Many approaches [5,8,11,12,13,16,19] were proposed for these issues but few of them supported a complete solution. In this paper, we present Venus, a prototype content-based video database system. Technologies for addressing the above issues are discussed, which are integrated in the implementation of Venus.

The concepts of content-based retrieval were adopted from image databases [4,14,18]. For video modeling and querying, previous works [12,16] focused on the representations of temporal and spatial relationships of content symbols. Two types of content-based retrieval were investigated. One is querying based on the image features of video contents [19] and the other querying based on content object specifications [5,9,13].

Information implied in the video contents has to be extracted in advance as the video index for query processing. A video sequence can be organized as shot, scene, episode and video [6]. It can be processed on uncompressed domain or compressed domain. Many works segmented video sequences into video shots [2,15,17,20,21,22] on uncompressed domain. Based on compressed domain, [1] proposed an approach for video segmentation by analyzing the Discrete Cosine Transform (DCT) coefficients of video frames.

For video querying, we designed a content-based video query language CVQL [9] for Venus. CVQL is powerful in predicate specifications. The temporal and spatial relationships can be flexibly specified to describe events or status of the video contents. Moreover, an indexing tool is supported for the construction of video indexes which are required for the CVQL processing. An index construction method based on the compressed domain was designed to avoid the large amount of analysis time on raw video data. Elimination-based query processing was used in Venus to avoid frame by frame predicate evaluations.

This paper is organized as follows. In Section 2 the system architecture of Venus, and its

* This work was partially supported by the Republic of China National Science Council under Contract No. NSC 86-2213-E-007-017.

components are introduced. In Section 3, we describe the main ideas for video index construction. The CVQL is introduced in Section 4. The elimination-based query processing is depicted in Section 5. The last section concludes our works and discusses the future works.

2 System Architecture

In this section, we introduce the system architecture of Venus. Venus is designed in the X environment. In this system, video queries with predicate specifications of the temporal and spatial relationships of content objects are provided.

The system architecture of Venus is shown in Figure 1. It consists of three components: (1) Video manager, (2) Query manager, and (3) Indexing tool, as presented below.

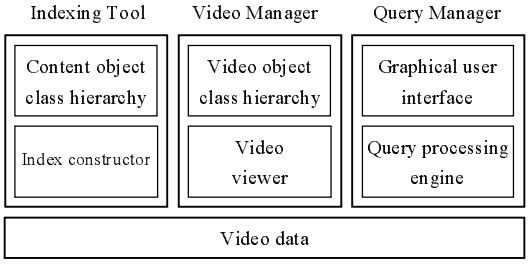


Figure 1: The system architecture of Venus.

2.1 Video manager

The video manager manages video objects and provides structural video browsing. The video objects are organized in a class hierarchy which supports the classification of videos and a flexible search space for query processing. Figure 2 illustrates a video class hierarchy. **Basketball** and **Tennis** are subclasses of video class **Sports**. The user who is interesting in **Sports** videos need not access or query the videos in class **Politics** since the possible results are all in the class **Sports**. In this paper, the name of a video will be presented in bold face Roman beginning with a lowercase letter, and a video class will be presented in bold face Roman beginning with a capital letter.

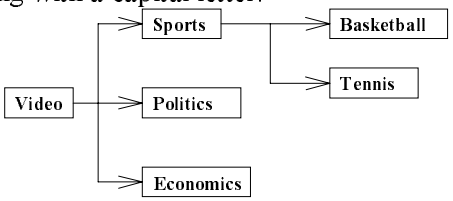


Figure 2: A video class hierarchy.

A video consists of a set of alpha-numeric attributes and raw video sequences. The raw video sequence is in the format of MPEG [7] coded stream, which can be displayed by the video viewer. The attributes are name, length, and description of the video. The description attribute stores the keywords associated with the video. The keywords are used for video classification. For each class, a keyword is associated with it. When the keyword of a video is matched with the keyword of a class, the video is classified into this class. The class hierarchy and their associated keywords are constructed manually. When a video contains multiple keywords, the video belongs to the associated classes.

Video objects can be inserted, deleted, classified and browsed by the video manager.

2.2 Query manager

In Venus, CVQL is used for describing queries by video contents. Since the CVQL may be complex for naïve users, a graphical user interface (GUI) is supported for simplifying query specifications.

A *content object* is a symbol extracted from videos (a content symbol), which represents a real world entity. For example, an anchorperson is a content object in a news video. The content object is used to specify predicates in the video database. To classify various kinds of content objects, a class hierarchy is constructed. For example, a user may be interested in the video containing some animals, the class Animal can then be used for this specification. The predicate of a content-based video query can be specified by the temporal and spatial relationships of the content objects.

There are three steps for posing a video query on the GUI. First, use the video browser to set the search range of the query. Since the video objects are classified in the class hierarchy, users can choose a suitable set of classes as the search range. Second, use the sketch board to specify the query predicates. The temporal and spatial relationships of content objects can be specified. Third, choose the form of query results. The query results can be the whole video, a set of continuous frames, or the content objects.

When the query specification is completed, it will be transformed into the format of CVQL for processing. The qualified video sequence will be shown as query results. Figure 3 shows the GUI.

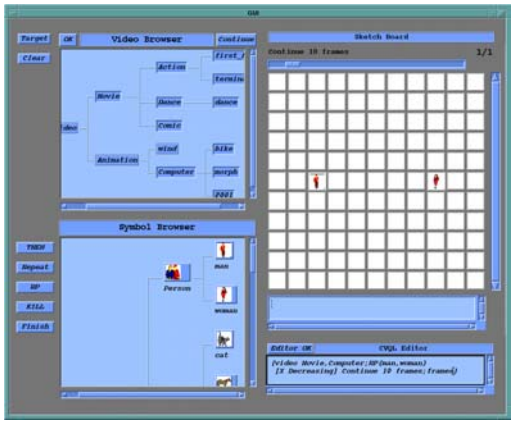


Figure 3: The graphical user interface.

The concept of CVQL and predicate specifications will be presented in section 4.

2.3 Indexing Tool

For CVQL query processing, the indexes of video contents are required. The content objects as well as their spatial and temporal relationships have to be extracted from the video contents. Since the video data are often stored in the compressed format, the algorithms of index construction are based on the analysis of MPEG coded video streams. The indexing tool consists of three parts. An incoming video object can be analyzed by these three components step by step for index construction.

- Shot change detector: A video is first divided into a sequence of shots. The detector analyzes MPEG coded streams to detect the possible shot change locations. The detected shot change locations can be confirmed manually. The shot change locations are stored for structural video browsing. Moreover, a shot is a basic processing unit for the detection of content objects.
- Content object detector: The second phase is the content object detection. For each shot, content objects are detected by the content object detector. The content objects which may be involved in the user queries are chosen. The content objects are assigned a name and classified into the content object class hierarchy.
- Motion track constructor: Finally, the motion tracks of the content objects are constructed. For each content object, the positions in the frames it appears are computed. Content object positions on the continuous frames are chained to form the motion tracks.

The algorithms of these detectors are presented in the next section.

The detected content objects are managed by the content object class hierarchy. A content object is presented in Arial font beginning with a lowercase letter and a content object class is presented in Arial font beginning with a capital letter. In Figure 4, we illustrate the class hierarchy of content objects.

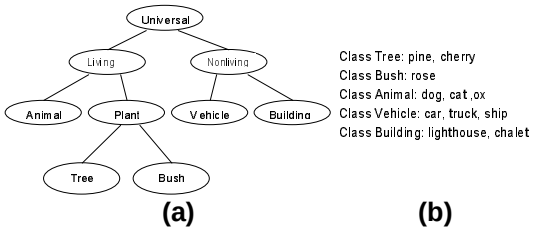


Figure 4: A class hierarchy of content objects.

3 Video Indexing

For processing a content-based video query, the matching between the query and video indexes is required. Since on-line processing on raw video data is time consuming, the content objects as well as their spatial and temporal information have to be extracted in advance. In this section, we illustrate the approaches for segmenting video sequences into shots, detecting content objects and extracting their motion tracks from MPEG coded video sequences.

3.1 MPEG Data Format

In MPEG coding, a frame is divided into macroblocks. Each macroblock is a 16 by 16 image as a basic coding unit. A macroblock can be coded by DCT or references to its adjacent frames when it matches the similar image patterns of these frames. A macroblock coded by DCT is named *intra-coded* macroblock. A macroblock referencing to similar image patterns is named *forward-prediction coded*, *backward-prediction coded* or *bidirectional-prediction coded* macroblocks when it references to the image patterns of the preceding frame, subsequent frame, or both preceding and subsequent frames, respectively. A reference to the preceding frame is named *forward reference*, and to the subsequent frame *backward reference*.

There are three types of frames, named *I* frame, *P* frame and *B* frame. All macroblocks in an I frame must be intra-coded macroblocks. Macroblocks of the P frame may have forward references to its preceding I or P frame. The displacement between the current macroblock and the similar image pattern is named *motion vector* and is encoded in the MPEG video stream. A B

frame may have references to its adjacent I or P frames. Bidirectional references are allowed. The macroblock in a B frame can be an intra-coded, bidirectional-prediction coded, forward-prediction coded, or backward-prediction coded macroblock.

The references among MPEG frames can be used to evaluate if an image area is similar to the image area of other frames or if a frame is similar to another frame. This feature can be used for video index construction, as presented below.

3.2 Video Segmentation

In a video sequence, the contents of continuous frames in the same shot are similar. Shot change detection is based on the similarity measurement between video frames. The references among coded frames are analyzed to measure the similarity of two frames.

We have proposed a mask matching algorithm for shot change detection [10]. The references of each coded frame is examined to judge if it is a shot change or not. For each frame, the analysis contains three steps.

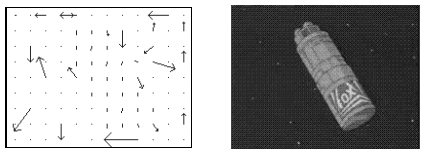
1. According to the frame type, apply a mask for it: The mask denotes the conditions that the current frame and its adjacent frames have to satisfy if the current frame is a shot change.
2. Examine each condition: A condition can be examined and a value between 0-1 is given.
3. Compute the shot change probability: The values of the conditions are input to a *shot change probability function*. It results in a shot change probability of the current frame.

After each frame is analyzed, the frame with the probabilities higher than a threshold will be considered as the shot change frames.

3.3 Detection of Content Objects

The detection of content objects is by analyzing the motion vectors of the referencing macroblocks of the video frames. A moving object in the current frame should be at different positions in the reference frames. If the moving object does not change a lot in its shape, color and size, it should be able to match itself in the reference frames while encoding. The macroblocks of the moving object should have the same or similar motion vectors.

This concept is shown in Figure 5. Notice that the motion vectors around the moving object are heterogeneous. They are helpful for separating the moving object from the background.

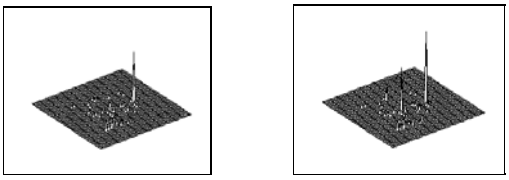


(a) (b)

Figure 5: (a) The motion vectors of P frame 28 in 2001.mpg; (b) Reconstructed image of the corresponding frame.

Based on the above observation, we propose an object detection method. The method takes an MPEG-coded frame as input. and the following steps are applied.

1. Motion vector extraction: For each frame, extract the motion vectors directly from the MPEG coded video.
2. Classification: By counting the number of macroblocks which have the same motion vector, we can obtain the distribution of motion vectors. An example is shown in Figure 6(a). If the motion vectors with similar numbers of macroblocks are adjacent, they are merged to form the *similar motion vectors*. Among the similar motion vectors, the one with the largest number of macroblocks is taken as the *representative motion vector*. An example of the distribution of the representative motion vectors is shown in Figure 6(b).



(a) (b)

Figure 6: The distributions of (a) motion vectors; (b) representative motion vectors.

3. Object finding: In the distribution of representative motion vectors, a peak indicates that there might be objects with the similar motion vectors. The similar motion vectors belonging to a peak are compared to the motion vectors in the frame. If a set of macroblocks are adjacent and their motion vectors are in the similar motion vectors, they are collected as an object candidate. Note that a frame often contains more than one peak and a peak may indicate more than one object.

3.4 Motion Track Extraction

After detecting the content objects, a motion track is constructed for each interested object. We present the motion track construction method in this section.

An object's motion track is a series of connections between the same object in the consecutive frames. In order to make a connection, we determine which are the same objects in the current frame and the previous frame on the basis of the objects' spatial information. The spatial information of objects refers to their positions, sizes and motion vectors. Two objects coming from consecutive frames are considered the same if they have similar positions and sizes when they are mapped into the same reference frame. Please refer to [23] for the details of building the connections on the three types of frames.

4 Query Language

Users retrieve video data by specifying features of video frames. In this section, we present the video query language CVQL. The features of video contents including the existence of content objects and their spatial and temporal relationships are used in this language.

A CVQL query can be expressed by the form: *{ range; predicate; target }*. The range clause defines the search space of a query. The target clause specifies the results which users want to retrieve. The qualification of a query is specified in the predicate clause.

A predicate is specified by the description of the temporal and/or spatial relationships of content objects. In CVQL, we use the function-based specification. It is easy for users to describe an event or state which appears in the video contents. A set of functions and modifiers are defined, as illustrated in the following sections.

4.1 Video functions

A video function returns the information of content objects in frames, such as location, motion of a content object and relative location of two content objects. We introduce the video functions as follows:

- AP(): It returns the location of a content object in a frame.

  Ex1: AP(dog)[X<3 ∧ Y<3]

  Ex1 describes whether a dog in a frame stays at the left-bottom corner of the frame grid, specified by "X<3 ∧ Y<3."
- RM(): The motion of a content object will be returned. A sequence of frames showing a bird flying to the right can be specified as Ex2:

  Ex2: RM(bird)[X>=1]
- RP(): RP() requires two content objects as parameters. By RP(), relative location between two content objects can be retrieved. Ex3 shows the application of RP().

  Ex3: RP(tree, car)[X<0 ∧ Y<0]

  In this example, the predicate "the car located at the left-bottom of the tree" is specified.
- Exist(): Exist() function examines whether a content object exists in a frame and returns a Boolean value (True/False). Q1 shows a simple video query which retrieves frames from video **nthu-campus**, in which a content object in content object class Person exists.

  Q1: {video **nthu-campus**; Exist(Person); frames}
- Distance(): Distance() is used to compute the geometric distance for a two-dimensional value produced by a video function. Q2 illustrates an example to find the frames which include two content objects fish and crab in a short distance.

  Q2: {video **sea**; Distance(RP(fish, crab))<3; frames}

4.2 Modifiers

In addition to the video functions, a set of modifiers can be used to enhance the power of predicate specifications.

- Increasing, Decreasing and Equal: We have introduced five video functions for retrieving the information of content objects from video frames. However, the basic comparative operators is not enough. For example, it cannot specify that two content objects keep the same relative location in two or more continuous frames. The Increasing, Decreasing and Equal modifiers (named XY modifiers) are proposed for supporting the temporal expressions for a frame sequence. Q3 illustrates an example which shows two content objects keep the same relative location in two or more continuous frames. In Q4, video objects containing a content object ball which is falling faster and faster will be retrieved.

  Q3: {video *; RP(o_1, o_2)[X Equal, Y Equal]; video}

  Q4: {video *; RM(ball)[Y Decreasing]; video}

  For the three modifiers, temporal specifications such as the variance of speed of moving content objects and the variance of

relative position of two content objects are supported.

- Continue: In a query, we may have to specify the time interval for a state or event. The Continue is used to indicate the minimum time interval for a state. The Continue follows a video function. For instance, Q5 accesses videos which have at least 30 sequential frames containing a content object in Person.

  Q5: {video *; Exist(Person) Continue 30 frames; video}

  For convenience, the number of seconds can be used instead of the number of frames as the unit for time interval specification.
- Then: This modifier concatenates two states into a compound state in time sequence. Q6 retrieves frames containing a ball moving left and then moving right. The former state describes a ball moving left and the latter state describes the ball moving right.

  Q6: {video **sport**; RM(ball)[X<0] Continue 10 frames Then RM(ball)[X>0] Continue 10 frames; frames}
- Repeat: The Repeat modifier specifies a repeated complex state. For example,

  Q6.1: {video **sport**; (RM(ball)[X<0] Continue 10 frames Then RM(ball)[X>0] Continue 10 frames) Repeat 2 times; frames}

We have introduced the operations for specifying spatial relationships and temporal relationships between content objects in the query predicate. In the following, we illustrate some more complex query examples.

Q7: {video **Disaster**; Distance(RP(train, car)) Decreasing to 0 Then !Exist(train, car); frames}

Q8: {video **sport**; RM(barrier)[X=0 ∧ Y=0] and RM(horse)[X>=0 ∧ Y>=0] and RP(barrier, horse)[X<=0] Then RM(horse)[X>=0 ∧ Y<=0] and RP(barrier, horse)[X>=0]; frames}

In Q7, users want to retrieve video frames of a train crashing with a car from disaster videos. In the predicate of Q7, first we describe the state of a train approaching a car. After the crash, both the car and train disappear from the next frame.

Q8 retrieves the frame sequence containing a horse jumping over a barrier from a sport video. There are two states in the predicate of the query. The first one describes a horse running close to the barrier and jumping up, and the second describes the moving track and the spatial relationships between the horse and barrier after the horse jumps over the barrier.

It may be difficult for a user to specify a complex state by primitive video functions and modifiers. For example, to describe a crash, instead of the complex predicate specification of Q7, it will be more convenient if a crash function is predefined.

A macro can be defined by primitive video functions and modifiers or the other macros with a set of parameters. For example, Ex4 defines the Near function.

Ex4: Near(o_1, o_2) = Distance(RP(o_1,o_2)) < 3;

Ex5 illustrates the simplification of Q7 by defining a crash macro function, as shown below.

Ex5: Approach(o_1, o_2) = Distance(RP(o_1, o_2)) Decreasing;

Crash(o_1, o_2) = (Approach(o_1, o_2) to 0 Then !Exist(o_1, o_2)) and Class(o_1, o_2) = Vehicle;

Therefore, Q7 can be simplified to:

{ video **Disaster**; Crash(car, train); frames }

The Class function is used to limit the parameters since any content object may be applied as a parameter. For the use of macro, a set of macros are predefined. Users can define new macros by themselves. The definition of a macro can be changed by user profiles. For example, Ex4 defines two near content objects by their relative distance "3". A user may change it to be "2" for a stricter criterion.

The thirteen temporal relationships proposed in [3] can be defined as macros using our video functions and modifiers. We show the definition of Meet function in the following.

Meet(o_1, o_2) = Exist(o_1) and !Exist(o_2) Then !Exist(o_1) and Exist(o_2);

5 Query Processing

For CVQL, users can pose their queries by combining temporal and spatial relationships of content objects. A large amount of video indexes may have to be evaluated. Therefore, an efficient index structure and processing strategy are required for query processing.

A video query may contain a set of videos as range clause and each video consists of a large amount of frames. It is time-consuming to evaluate predicates frame by frame. the query processing is based on a three phase elimination. The processing steps are introduced as follows.

1. Video elimination: A video cannot be a query result if this video does not contain the content objects specified in a predicate. For example, a query retrieves videos which contain a dog staying in the left-bottom corner of the frame

grid. If a video does not contain a dog, it cannot be a query result.

2. Frame elimination: Before the evaluation of predicates, a video can be filtered by the life time of the content objects. Frames not containing the content objects specified in a predicate can be skipped.
3. Video function evaluation: In the final phase, the motion tracks of content objects are accessed to evaluate the remaining frames. Video functions are actually evaluated in this phase. Qualified frames or videos are returned as the query results.

For the above three steps of elimination, we design the index structure for them.

- In the video object class hierarchy, each node represents a video class or a video. A node maintains a *CO* (content object) table which denotes the possible content objects that may exist in the videos in this class. The CO of a video class is the union of all the CO of its children. For example, Figure 7 shows the possible content objects for each class: CO(V1)={dog, car}, CO(V2)={car, tree}, and CO(C2)=CO(V1) ∪CO(V2)={dog, car, tree}. For video elimination, the predicate is first converted into the CO pattern. For instance, if the predicate is Exist(car, tree), the CO pattern is CO(Q)={car, tree}.

  The CO information is used to eliminate the search of video objects which are impossible as query results. The CO(Q) is first computed and compared to the CO of the video class C in the query range. The comparison will be propagated to the subclasses of C when CO(Q) ⊆CO (C), otherwise the class C will be pruned. For the above example, CO(Q)={car, tree}, V2 is the only candidate result video after the elimination process.

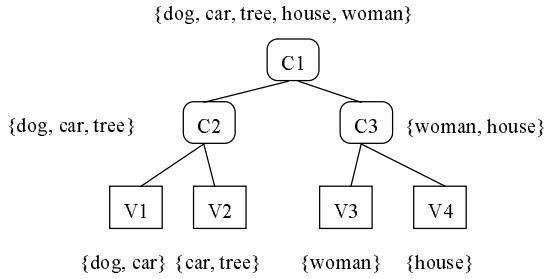


Figure 7: Index for video elimination.

- For the frame elimination, each video object maintains a content object life time table, named as CLVT. For example, a video contains a dog and a cat; the life time of these two content objects are { dog: 10-20, 40-45; cat: 15-25, 44-70, 90-102 }; and the predicate is "RP(dog, cat)[X>2] Continue 5 frames." Frames do not contain both a dog and a cat will be skipped. The frames (15-20) and (44-45) are possible frames. Furthermore, the predicate specifies to "Continue 5 frames." Frames (44-45) are then skipped since they do not satisfy the Continue clause.

6 Conclusions and Future Work

This paper presents an overall design and implementation of a content-based video database system, Venus. Key technologies for video retrieval, index construction as well as strategies of query processing are discussed. The CVQL with a graphical user interface provides flexible and powerful content-based video queries. The indexing tool supports for semi-automatic extraction of video shots, content objects and their motion tracks. For query processing, the concept of multiphase elimination is proposed to avoid the large mount of index accesses for predicate evaluation.

Approximate query capabilities are required when users cannot precisely specify the query predicate. The study of approximate query answering are currently in progress.

The Venus system will be integrated with an alpha-numeric database. The content objects are linked with the objects in the alpha-numeric database representing the same real-world entity. The information of the video database and alpha-numeric database can then be integrated to provide a more powerful query specification and a higher degree of resource sharing.

References

[1] Farshid Arman, Arding Hsu, and Ming-Yee Chiu. "Image Processing on Compressed Data for Large Video Databases," *in Proc. of First ACM Int'l Conf. on Multimedia, 1993.*

[2] Donald A. Adjeroh and M.C. Lee, "A Principled Approach To Fast Partitioning of Uncompressed Video," *in Proc. of IEEE Workshop on Multimedia Database Management systems*, pages 115-122, August 1996.

[3] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, pages 832-843, Nov. 1983.

[4] S.K. Chang, Q.Y. Shi, and C.W. Yan, "Iconic Indexing by 2-D strings," *IEEE Tran. Pattern*

Analysis and Machine Intelligence, pages 413-428, 1987.

[5] Y.F. Day, S. Pagtas, M. Iino, A. Khokhar, and A. Ghafoor, "Object-Oriented Conceptural Modeling of Video Data," *In Proc. of IEEE Data Engineering*, pages 401-408, 1995.

[6] G. Davenport, T.A. Smith, and N. Pincever. "Cinmatic Primitives for Multimedia," *IEEE Computer Graphics & Applications*, pages 67-74, July 1991.

[7] D. Le Gall. "MPEG: A video compression standard for multimedia applications," *Communications of ACM*, 34(4):46-58, April 1991.

[8] R. Hjelsvold and R. Midtsraum, "Modeling and Querying Video Data," *In Int'l Conf. on Very Large Data Bases*, pages 686-694, 1994.

[9] Tony C.T. Kuo and Arbee L.P. Chen, "A Content-based Query Language for Video Databases," *in Proc. of IEEE Multimedia Computing and Systems, June 1996.*

[10] Tony C.T. Kuo, Y. B. Lin and Arbee L.P. Chen, "Efficient Shot Change Detection on Compressed Video Data," *in Proc. of IEEE Workshop on Multimedia Database Management systems*, pages 101-108, August 1996.

[11] T.D.C. Little and A. Ghafoor, "Interval-Based Conceptual Models for Time-Dependent Multimedia Data," *IEEE Transaction on Knowledge and Data Engineering*, pages 551-563, August 1993.

[12] Suh-Yin Lee and Huan-Ming Kuo, "Video Indexing: An Approach Based on Moving Object and Track," *In Proc. of SPIE - The Int'l Society for Optical Engineering*, Vol. 1908, pages 25-36, 1993.

[13] John, Z. Li, et al., "Modeling of video Spatial Relationships in an Object Database Management," *in Proc. of IEEE Workshop on Multimedia Database Management systems*, pages 124-132, August 1996.

[14] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, and D. Petkovic, "The QBIC Project: Querying Images By Content Using Color, Tecture, and Shape," *In Proc. of SPIE - The Int'l Society for Optical Engineering*, Vol. 1908, pages 173-187.

[15] A. Nagasaka and Y. Tanaka. "Automatic Video Indexing and Full-video Search for Object Appearances," *in 2nd Working Conference on Visual Database Systems*, pages. 119-133, Budapest, Hungary, October 1991, IFIP WG 2.6.

[16] Eitetsu Oomoto and Katsumi Tanaka, "OVID: Design and Implementation of a Video-Object Database System," *IEEE Transactions on Knowledge and Data Engineering*, pages 629-643, August 1993.

[17] Kiyotaka Otsuji and Yoshinobu Tonomura. "Projection Detecting Filter for Video Cut Detection", *Proc. of ACM Multimedia*, pages. 251-257, 1993.

[18] E.G.M. Petrakis and S.C. Orphanoudakis, "Methdology for the Representation, Indexing and Retrieval of Image by Content," *Image and Vision Computing*, 1993.

[19] Stephen W. Smoliar and HongJiang Zhang, "Content-Based Video Indexing and Retrieval," *IEEE Multimedia Magazine*, pages 62-72, 1994.

[20] Y. Tonomura and S. Abe. "Content Oriented Visual Interface Using Video Icons for Visual Database Systems," *Journal of Visual Languages and Computing*, 1:183-198, 1990.

[21] H. Ueda, T. Miyatake, and S. Yoshizawa, "Impact: An Interactive Natural-motion-picture Dedicated Multimedia Authoring System," *In Proc. of Human Factors in Computing Systems (CHI91)*, pp. 343-354, New Orleans, Louisiana, 1991.

[22] H. Zhang, A. Kankanhalli, and S.W. Smoliar. "Automatic Partitioning of Video," *IEEE Multimedia System* Vol. 1, No. 1, pages. 10-28, 1993.

[23] Y.B. Lin, "An Efficient Method to Build Video Indexes from Compressed Data," Master Thesis, Computer Science Department of National Tsing Hua Univ., Taiwan, R.O.C., June 1996.