A Content-Based Query Language for Video Databases*

Tony C.T. Kuo and Arbee L.P. Chen
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
alpchen@cs.nthu.edu.tw

Abstract

This paper presents a content-based video query language CVQL for video databases. Spatial and temporal relationships of content objects are used for the specification of query predicates. Queries of realism are illustrated to show the power of CVQL. Macro definitions are supported to simplify query specification. Index structures and query processing for CVQL are considered, and a prototype video database system is implemented, which consists of a GUI and a CVQL processor. Users can sketch a query and its corresponding predicate by the GUI, and the query can then be converted to CVQL for processing.

1. Introduction

With the progress of computer hardware and storage technologies, databases for managing multimedia data, such as audio, video, image, animation and graphics, were investigated in the past years. Although video is rich in the temporal and spatial relationships between its content objects, there has been few research which provides suitable accessing interfaces based on these characteristics. The goal of the video database is to support an efficient and easy way for users to retrieve video data. Traditional query capabilities can only support textual and numerical-based evaluation. For example, we can retrieve video data by specifying video identifier, title or their descriptions. However, users cannot specify predicates to retrieve parts of video data and the characteristics of video data are not fully used for query specifications. Consider the following example, a user may want to retrieve parts of a video object named **sport**, which shows a 100M runner passing the finish line. A new query model should be designed to meet the requirements of video queries.

Many researchers have investigated the enhancement of video query capabilities [9,7,6,4,11,3]. In the past, content-based retrieval was applied in image databases [2,10,8]. Similar concepts are extended to enhance query capabilities in video databases. In [11], video data can be queried by image features, such as color, texture and shape. The query capabilities are limited. [9] proposed a video query language VideoSQL. It applied a new inheritance mechanism based on interval inclusion relationship between video objects for specifying queries. In [6], a set of temporal operators were designed for video queries. However, the temporal relationships can be evaluated between frame sequences only. Temporal relationships of content objects were not considered. [7] considered sixteen primitive types of motions for specifying the tracks of content objects in queries. However, the spatial relationships between content objects were not considered. In [3], the spatial/temporal semantics of video data were studied. Conceptual Spatial Object (CSO), Conceptual Temporal Object (CTO), Physical Object (PO) and a set of predicate logics were defined to express queries. Since spatial and temporal semantics are only captured by CSOs and CTOs, semantics that haven't been defined in CSOs and CTOs cannot be applied in queries.

In this paper, we propose a new mechanism of content-based retrieval to access video data. A content-based video query language CVQL is presented. In CVQL, both temporal and spatial relationships of content objects are considered. A set of operations for specifying temporal and spatial relationships in queries is defined. By these operations, characteristics of video data can be used for query qualification. Users can express the semantics of demanded video data by combining the proposed temporal and spatial operations in CVQL. The indexes and the processing of CVQL are considered. Macros are proposed to simplify query specification. We also implement a GUI and a query processor of CVQL in our prototype video database system.

This paper is organized as follows: in section 2, our video query model is introduced, which includes the representation of video data, the specification of content-based video query and the index structures. Section 3 presents the syntax of CVQL and introduces the operations for video predicate specifications. We briefly describe query processing of CVQL in section 4. The implementation of our prototype video database system is presented in section 5. The last section leads our

* This work was partially supported by the Republic of China National Science Council under Contract No. NSC 85-2213-E-007-024.

conclusions and future works.

2. Video Query Model

2.1. Video objects

A video is viewed as an object (named *video object*), which consists of raw data and descriptions. The raw data part is composed of a set of continuous image frames, which can be displayed by video devices. The description part provides descriptive information of a video, such as the video title, the number of frames and the introduction of the video.

Various kinds of video may exist in the database. Videos are organized as a class hierarchy for easy retrievals. For example, in Figure 1, **Basketball** and **Tennis** are subclasses of video class **Sports**. In this paper, the name of a video will be presented in bold face Roman, and a video class will be presented in bold face Roman beginning with a capital letter.

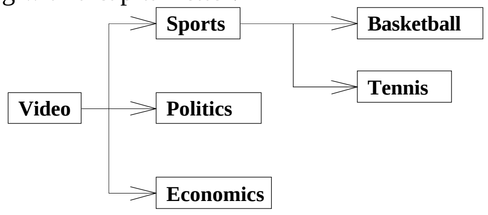


Figure 1: A video class hierarchy.

2.2. Content-based video queries

A content-based video query is a query which specifies predicates by describing the contents of video for retrieving video data. The contents can be color histogram values, textures, image shapes or *symbol objects* of video or video frames. A symbol object is a symbol extracted from video, which represents a real world entity. For example, an anchorperson is a symbol object in a news video. Our video query language is based on the predicate specification of the temporal and spatial relationships of symbol objects. It is more natural for users to retrieve video data by specifying video contents since users often remember some snapshots of the required videos. Moreover, the query capabilities can be enhanced since users can flexibly specify various kinds of predicates. In CVQL, various functions are supported for describing the spatial and temporal relationships of symbol objects. For naive users, system predefined macros are provided to simplify the predicate specification.

A state is a specification of temporal/spatial relationships or existence of symbol objects, which can be used as a query predicate. We introduce various types of state descriptions as follows:

I. Existence of symbol objects: The simplest predicate specification is to search videos which contain the user-specified symbol objects.

II. Spatial relationships of symbol objects: Descriptions of spatial relationships are based on the location that symbol objects appear in video frames. A frame can be viewed as a two-dimensional space. The location of a symbol object can then be denoted as (x,y). There are two types of spatial relationship descriptions: one is relevant to a single symbol object in which the location of a symbol object is specified. The other is relevant to two or more symbol objects in which the relative locations of two or more symbol objects are specified.

III. Temporal relationships of symbol objects: A video stream is a sequence of continuous image frames. The relationships of symbol objects among video frames are considered as temporal relationships.

IV. Compound relationships of symbol objects: A more complex state can be specified by combining the descriptions of spatial and temporal relationships of symbol objects. According to the two types of spatial relationships described in type II, we explain their combinations with temporal relationships. In type one, the motion of a symbol object can be specified by considering the location of the symbol object in a continuous frame sequence. In type two, the variance of the relative location of two symbol objects can be specified by comparing the difference of the relative location of two objects between two continuous frames. For example, a compound relationship describes a video shot where two dogs moving closer and closer by specifying the variance of the relative location of these two dogs in continue frames.

V. Compound state: Since a state is a description of predicate with temporal/spatial relationships, when the relationships are changed, it is called a *state change*. Combining two or more states in a sequential order is named a *compound state*. For example, a ball jumping up and then falling down needs to be described by a compound state: the first state denotes the ball moving up and the second denotes the ball falling down.

VI. Semantic descriptions: We have introduced various types of queries with the use of temporal/spatial relationships. However, it may be inconvenient for users to issue queries by describing complex predicates, especially for naive users. A semantic description is a way of relieving the difficulty by allowing users to specify the complex predicates by simple functions. For example, instead of the complex specifications of temporal/spatial relationships, a simple operation near(ol, o2) can be used to specify two symbol objects with a short distance.

In CVQL, a set of functions and modifiers are defined for specifying predicates. Macros are also provided to support semantic descriptions.

2.3. Content-based video indexing

For content-based video queries, symbol objects have to be detected from video contents. The motion tracks of symbol objects have to be derived too. [12] extracts content-based indexes by the analysis of video raw data. They first detects shot changes by comparing color histogram values of video frames. The moving objects are then detected by the difference of two continuous frames and the static symbol objects are detected by edge detection based image processing routines. In [5], we propose a method to extract content-based indexes from MPEG coded video data. Shot change detection is performed by analyzing the reference ratios of macroblocks of video frames.

There are four types of indexes for CVQL:

- Symbol object hierarchy: Symbol objects are managed in a class hierarchy. When a symbol object class is used in a query, it represents all symbol objects belonging to this class. A symbol object is presented in Arial font and a symbol object class is presented in Arial font beginning with a capital letter. In Figure 2, we illustrate a class hierarchy of symbol objects.

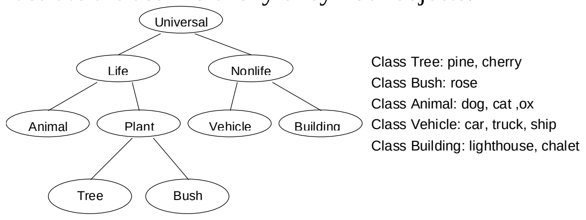


Figure 2: A class hierarchy of symbol objects.

- Video-symbol_object table (VST): This table records the videos that a symbol object appears. An example of VST is shown as : {car: **race, headline-news**; tree: **picnic**; sun: **supermanII**}. From this example we know a symbol object car can be found in videos **race** and **headline-news**.
- Symbol_object life-time table (SLT): Each video has its own SLT. The SLT records the frame duration a symbol object appears in a video. For example, a video **race** contains three symbol objects: a dog, a bird and a cat. The SLT of video **race** can be: { dog: 3-6, 20-30; bird: 15-25; cat: 1-6, 12-22, 27-30 }. It shows that the dog appears in frame 3 to 6 and 20 to 30 in video **race**.
- Symbol_object spatial_information table (SST): Each video has an SST. The SST records the locations of the symbol objects in each frame. From this table, motion tracks of symbol objects and relative positions between symbol objects can be derived.

3. A Video Query Language CVQL

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

3.1. Syntax of CVQL

A CVQL query can be expressed by the following structure:{ range; predicate; target }.

- range: The range clause defines the search space of a query. It can be a set of videos or video classes. If users have no idea about the possible sources where the target may come from, the symbol "*****" can be used to represent all videos.
- target**:** The target clause specifies the results which users want to retrieve. The target can be a whole video, some frames or some symbol objects.
- predicate**:** The qualification of a query is specified in the predicate clause. Objects in the range clause are evaluated by the specified predicate to get the result.

3.2. Predicate specification

A predicate specification has the following basic form: video-function(parameters)[xy-expression]. It can be parsed into two parts: *video function* and *xy-expression*. The video function part specifies the type of relationships, and the xy-expression part specifies the restriction for a predicate. In an xy-expression, the X variable and Y variable represent the x component and y component of the returned value of the video function, respectively. Comparative operators such as "<", ">", "=" and "!=" can be used in the xy-expression. Either X or Y variable can be omitted in an xy-expression when the value of X or Y is careless in the predicate. In the following, video functions for predicate specifications will be introduced.

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

- FP(): It returns the location of a symbol object in a frame. FP(dog)[$X<3 \wedge Y<3$] describes whether a dog in a frame stays at the left-bottom corner of the frame grid, specified by the xy-expression "$X<3 \wedge Y<3$."
- Om(): The motion of a symbol object will be returned. A sequence of frames showing a bird flying to the right can be specified as: Om(bird)[X>=1]. In this example, the Y variable is omitted. That is, there is no restriction on the y component of the returned value from Om(). By Om(), users can describe the moving direction and speed of a symbol object as well as whether a symbol object is static or not. For example, Om(bird)[$X=0 \wedge Y=0$] describes a static bird.
- Oom(): It calculates the movement of a symbol object between the frame in which the symbol object originates and the current frame. Oom(cat)[$|X|<2 \wedge |Y|<2$] demonstrates a cat bounded in a 3×3 area. FP() can act similarly to Oom(). For example, FP(cat)[$3<X<7 \wedge 1<Y<5$] tests if the cat is bounded in (3,1) to (7,5). The difference between FP() and Oom() is that Oom() bounds the cat in a certain area while FP() in an exact area, as shown in Figure 3.
- RP(): RP() requires two symbol objects as parameters. By RP(), relative location between two symbol objects can be retrieved. For RP(tree, car)[$X<0 \wedge Y<0$], the predicate "the car located at the left-bottom of the tree" is specified. A complex state can be expressed by using these functions together. RP(tree, car)[$X<0 \wedge Y<0$] and FP(tree)[$X=9 \wedge Y=8$] and Om(car)[$X>=0 \wedge Y<=0$]

further specifies the tree is located statically at (9,8) and the car moves to the right-bottom of the frame, as shown in Figure 4.

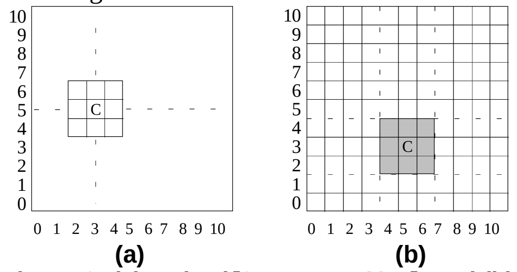


Figure 3: (a) FP(cat)[3<X<7 ^ 1<Y<5], and (b) Oom(cat)[|x|<2 ^ |Y|<2].

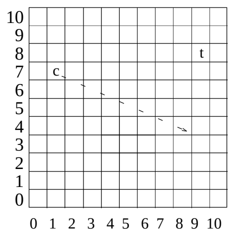


Figure 4: The state of RP(tree, car)[X<0 ^ Y<0] and FP(tree)[X=9 ^ Y=8] and Om(cat)](X>=0 ^ Y<=0] .

- Exist(): Exist() function examines whether a symbol object exists in a frame and returns a Boolean value. Q1 shows a simple video query which retrieves frames from video **nthu-campus**, in which a symbol object in symbol object class Person exists. Q1={video **nthu-campus**; Exist(Person); frames}. We show the symbol object hierarchy and some frames of video **nthu-campus** in Figure 5. From Figure 5(b), we find that symbol objects mary, john and alex belong to Person. Therefore, frames of video **nthu-campus** in which symbol objects mary, john or alex appear will be retrieved as the result of Q1. In Figure 5(a), frames *i* and *i+2* are retrieved since symbol objects mary and alex exist in frame *i* and john in frame *i+2*.

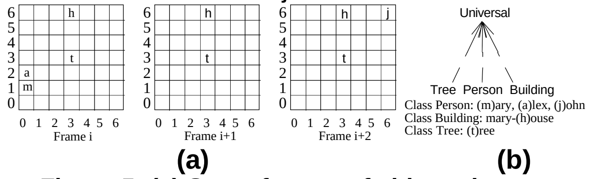


Figure 5: (a) Some frames of video nthu-campus, and (b) the symbol object hierarchy.

- Distance(): Distance() is used to compute the distance for a two-dimensional value produced by a video function. For example, in frame *i* of Figure 5(a), RP(t, a) = (0,2) - (3,3) = (-3,-1) and Distance(RP(t, a)) = $((-3)^2 + (-1)^2)^{0.5} = 10^{0.5}$. Q2 illustrates an example to find the frames which include two symbol objects fish and crab in a short distance. Q2={video **sea**; Distance(RP(fish, crab))<3; frames}. Distance() makes different semantics when it is applied to different video functions, as depicted below:
  1. FP(): It returns the distance of a symbol object from its current location to (0,0).
  2. Om(): It returns the moving distance of a symbol object from the previous frame to the current frame.
  3. Oom(): It returns the distance of a symbol object from the current frame to the frame where the symbol object first appears.
  4. RP():It returns the distance of two symbol objects.

By using the video functions, the spatial and temporal relationships among symbol objects can be specified.

3.2.2. 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 symbol objects from video frames. However, the basic comparative operators for the xy-expression is not enough. For example, it cannot specify that two symbol 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 xy-expression for a frame sequence. Q3 illustrates an example which shows two symbol objects keep the same relative location in two or more continuous frames. In Q4, video objects containing a symbol 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 *****; Om(ball)[Y Decreasing]; video}. Therefore, the temporal expression power of the xy-expression is enhanced by these three modifiers, such as the variance of speed of moving symbol objects and the variance of relative position of two symbol objects. Moreover, a complex xy-expression can be simplified by XY modifiers. For example, the following predicates are equivalent: (1)Om(o)[X>0] and FP(o)[X>3]; (2)FP(o)[X>3 $\wedge$ X Increasing].
- Continue: In a query, we may have to specify the time interval for a state. The Continue is used to indicate the minimum time interval for a state. The Continue follows an xy-expression or a video function. For instance, Q5 accesses videos which have at least 30 sequential frames containing a symbol 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**; Om(ball)[X<0] Continue 10 frames Then Om(ball)[X>0] Continue 10 frames; frames}.
- Repeat: The Repeat modifier specifies a repeated complex state. For example, Q6.1={video **sport**; (Om(ball)[X<0] Continue 10 frames Then Om(ball)[X>0] Continue 10 frames) Repeat 2 times; frames}.

Modifiers are used to enhance the capabilities of the temporal relationship of symbol objects among the frame sequence. Increasing, Decreasing and Equal are used to specify the variance of spatial relationships; Continue is used to depict the time duration that a state has to be kept; and Then and Repeat modifiers are used to arrange the time order of each state.

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

Q7:{video *; RP(dolphin, ball)[X=0 $\wedge$ Y=1] Continue 3 seconds; video}.

Q8:{video *; Om(bird)[|X|<=1 $\wedge$ |Y|<=1] Continue all frames; video}.

Q9:{video *; FP(bird)[|X|<=3 $\wedge$ |Y|<=3] Continue all frames; video}.

Q10:{video **Disaster**; Distance(RP(train, car)) Decreasing to 0 Then !Exist(train, car); frames}.

Q11:{video **sport**; Om(barrier)[X=0 $\wedge$ Y=0] and Om(horse)[X>=0 $\wedge$ Y>=0] and RP(barrier, horse)[X<=0] Then Om(horse)[X>=0 $\wedge$ Y<=0] and RP(barrier, horse)[X>=0]; frames}.

Q7 retrieves video objects containing a frame sequence which has a dolphin crowned with a ball. Such a frame sequence has to be kept for a duration of three seconds.

The target of Q8 are those video objects containing birds flying slowly. Since the "Continue all frames" is specified, all frames of a result video must have a symbol object bird satisfying "Om(bird)[|X|<=1 $\wedge$ |Y|<=1]."

Q9 retrieves videos containing birds and these birds are bounded in the area (0,0) to (3,3).

In Q10, users want to retrieve video frames of a train crashing with a car from disaster videos. In the predicate of Q10, first we describe the state of a train approaching a car. After the crash, both the car and train disappear from the next frame.

Q11 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.

3.3. Macros

Macros are defined for the simplification of predicate specifications. 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 as shown in Q10, 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, A Near function can be defined as: Near(o_1, o_2) = Distance(RP(o_1,o_2)) < 3.

The thirteen temporal relationships proposed in [1] 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, o_2) and !Exist(o_1, o_2) Then !Exist(o_1, o_2) and Exist(o_1, o_2).

The following example illustrates the simplification of Q10 by defining a crash macro function: 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, Q10 can be simplified to: { video **Disaster**; Crash(car, train); frames }.

The Class function is used to limit the parameters since any symbol 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, the Near() defines two near symbol objects by their relative distance "3". A user may change it to be "2" for a stricter criterion.

4. Elimination-based Video 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. For CVQL, 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 symbol 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. The Video-symbol_object table is used for the video elimination.
2. Frame elimination: Before the evaluation of predicates, a video can be filtered by the symbol_object life-time table. Frames not containing the symbol objects specified in a predicate can be skipped. For example, a video contains a dog and a cat; the SLT={ 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 have "Continue 5 frames." Frames (44-45) are then skipped since they do not satisfy the Continue clause.
3. Video function evaluation: In the final phase, the symbol_object spatial_information table is accessed to evaluate the remaining frames. Video functions with the xy-expressions are actually evaluated in this phase. Qualified frames or videos are returned as the query results.

5. Implementation

A prototype system is implemented to illustrate the capabilities of CVQL. In this system, a graphical user interface is provided. The system architecture is briefly introduced in the following .

5.1. System architecture

In Figure 6, the architecture of our prototype system is shown. It can be divided into five components: graphical user interfaces (GUIs), manager tools, indexes, query processors and video data. Users can access this system through GUIs or manager tools. Users may issue queries by sketching the predicates from the graphical query interface or by directly inputting a query to CVQL Editor. The query is converted into CVQL format and passed to query processor for processing. The query results are shown by Result Viewer. When a new video is inserted, it can be classified the new video into a video class by Video Manager. Also, for this video, symbol objects as well as their indexes have to be constructed. It is performed by Symbol Object Manager and Index Constructor.

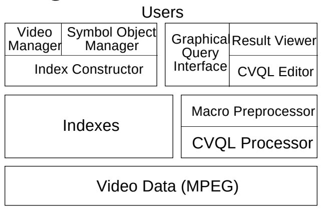


Figure 6: Architecture of the prototype system.

5.2. Query specification by the GUI

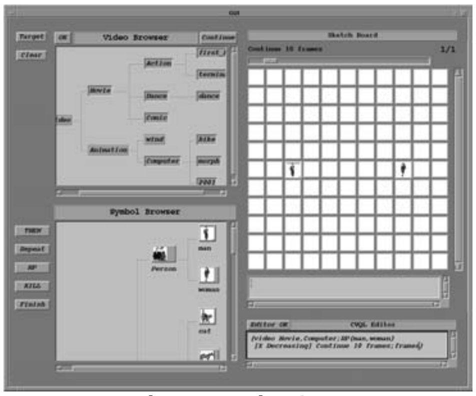
A query can be specified by the GUI. It contains three steps: (1) describes the range clause, (2) describes the query predicate, and (3) describes the target clause. The range clause can be described by selecting videos or video classes from Video Browser. A predicate is specified by selecting symbol objects from Symbol Browser and setting the spatial and temporal qualifications among them from the sketch board. The Target button is used to describe query targets. The GUI is shown in Figure 7.

6. Conclusions and Future Work

In this paper we propose a content-based video query language for video databases. The video contents are preprocessed to obtain the indexes of content objects. Users retrieve video data by specifying the state of video contents. A set of video functions is provided for describing the spatial and temporal relationships between content objects in the query predicate. Moreover, modifiers are supported for specifying complex states of desired video contents. The macros allow users to define semantic operations for simplifying query specification.

Four kinds of indexes are used for query processing. To avoid evaluating functions video by video and frame by frame, video and frame elimination are used to save the processing cost. A prototype video database system is implemented. Graphical query interface is provided and queries can automatically be converted into CVQL for processing.

Query relaxations will be considered in our future work. It is based on the relaxation of the evaluation of time intervals, function values and symbol object class hierarchies. Moreover, the CVQL will be extended for the integration of video databases and nonmedia databases.



```
{Video Movie,Computer;HP(man,woman)
 [X Decreasing] Continue 10 frames;frame}
```

Figure 7: The GUI.

Reference

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

[2]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.

[3]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.

[4]R. Hjelsvold and R. Midtsraum, “Modeling and Querying Video Data,” *In Int‘l Conf. on Very Large Data Bases*, pages 686-694, 1994.

[5]Tony C.T. Kuo, Y. B. Lin and Arbee L.P. Chen, “An approach for efficient shot change detection on compressed video data,” submitted for publication.

[6]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.

[7]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.

[8]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.

[9]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.

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

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

[12]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.