Advanced Video Information System: Data Structures and Query Processing *

Sibel Adalı, **Kasım S. Candan**, **Su-Shing Chen**[†]
Kutluhan Erol, and **V.S. Subrahmanian**
Institute for Advanced Computer Studies
Institute for Systems Research
Department of Computer Science
University of Maryland, College Park, MD 20742.
{sibel, candan, kutluhan, vs}@cs.umd.edu, schen@nsf.gov

Abstract

In this paper, we describe how video data may be organized and structured so as to facilitate queries. We develop a formal model of video data and show how spatial data structures, suitably modified, provide an elegant way of storing such data. We develop algorithms to process various kinds of video queries and show that in most cases, the complexity of these algorithms is linear. We develop algorithms to update these video databases. A prototype system called AVIS ("Advanced Video Information System") has been designed at the University of Maryland based on these concepts.

1 Introduction

During the last few years, the advent of the CD-Rom, and the introduction of high bandwidth communications networks has caused a spectacular explosion in the availability of large video-libraries. While a great deal of effort has been invested in problems of how to effectively utilize bandwidth to communicate large bodies of data across the network, relatively little effort has gone into how to organize, and access, video databases. The primary aim of this paper is to study methods of indexing video databases so as to efficiently store and retrieve video data in a number of diverse ways. Some example queries to be used for such retrievals may be:

(Query 1) find all video-frames in which John Wayne appears,

(Query 2) find all video-frames in which someone is being murdered, or more complicated ones such as

*This research was supported by the Army Research Office under grant DAAL-03-92-G-0225, by the Air Force Office of Scientific Research under grant F49620-93-1-0065, by ARPA/Rome Labs contract Nr. F30602-93-C-0241 (Order Nr. A716), and by an NSF Young Investigator award IRI-93-57756.

†Su-Shing Chen is with: Information Technology and Organizations Program, National Science Foundation, 4201 Wilson Blvd, Arlington, VA 22230.

(Query 3) find all video-frames in which Dean Martin appears and Fred Astaire is dancing with Ginger Rogers,

(Query 4) find all people who appear in the frames where Gene Kelly and Ginger Rogers are getting married,

(Query 5) find all video-frames showing Ginger Rogers dancing with someone other than Fred Astaire when he is in the same room, (note that Fred Astaire may not be visible in all these frames, but the fact that he is in the room during that time may be known in advance)

(Query 6) find all video-frames in which Alfred Hitchcock appears after someone is killed.

These are only a few of the examples of the kinds of complex queries that may be asked of a multimedia database system. When designing indexing schemes to store multimedia data, we need to take into account the kinds of queries that will be asked of such a system.

The primary contributions of this paper are the following:

1. First of all, we will show that certain kinds of existing spatial data structures are adequate for efficiently storing certain kinds of information. In particular, we will show that the problem of storing which objects occur in which frames may be viewed as an equivalent problem of storing line segments.
2. Second, we will show how we may make use of the above "spatial database" intuition to efficiently index multimedia data so as to be able to handle all the kinds of queries listed above – in particular, we will show how a combination of spatial database technology and relational database technology can be "merged" together to efficiently solve all the kinds of queries above.
3. Third, we will describe how updates to information about video data may be efficiently implemented using the data structures defined above.
4. Finally, we will describe a prototype implementation, called AVIS ("Advanced Video Information System") of the above data structures and algorithms. The implementation suggests that the theoretical algorithms work very well in practice.

2 Video Databases and Spatial Databases

2.1 A Motivating Example - A Movie by A. Hitchcock

Before we go on any further, we will give an example which will be used throughout the paper. In this example, we will introduce an Alfred Hitchcock movie, namely "The Rope (1948)." This movie is of 80 minutes duration.

In this movie, two friends, Philip and Brandon (Farley Granger and John Dall) decide to commit the perfect crime. They want to prove they are of the privileged group of people who are allowed to kill just for the sake of killing and not to receive any punishment for it. Hence, they kill their friend David and hide him inside the chest in the living room. To sign their masterpiece, they give a party where they invite friends of David (David's girlfriend Janet - Joan Chandler, Janet's old boyfriend Kenneth - Douglas Dick), and his parents (David's father Mr. Kentley - Sir Cedric Hardwicke, David's aunt Mrs. Atwater - Constance Collier). These individuals would talk about David, little suspecting that David's body is in the same room as them. In addition to these people, they invite, as a challenge, their old mentor Rupert Cadell (James Stewart) who is known to be very intelligent and suspicious. Rupert will prove worthy of his reputation and he will immediately understand the extraordinary circumstances. As the movie progresses, Rupert will keep asking questions and gather clues to find out what is wrong.

We will come back to this example in the following sections, and use it as our main example. We have distribute clues about the progression and the end of the movie throughout the paper. As the paper progresses, the "careful" reader will learn what Rupert learns and how, and what happens at the end.

2.2 Entities in a Video Database

If we look at the queries given in the introduction section, we will notice that two types of information are being queried. The first type is a set of *entities* – things of interest to us in the movie. The second type of information is the video-frames in which these entities are present. In this section, we will list the types of entities in a video that are of interest and later we will explain how the notion of the occurrence of an entity can be mapped onto the spatial domain.

Entities in a video database may have very different structures. For example, if they correspond to the visible objects and characters in a video we can answer the first question given in the introduction. Query 5 involves objects that are present in the scene, but not visible. Queries 3 and 4, on the other hand, deal with entities that have a certain set of attributes (the activity of getting married has the related attributes such as Groom,Bride,Best Man, etc.) some of which may be specified. Similarly, there may be entities for different media such as audio. The list of different types of entities is given below (we do not claim in any way that this is an exhaustive list, the reader is invited to add his or her own entities to this list):

- **Video Objects:** These are the entities that are present in the video frames. The video objects may be characters in the movie such as Philip, Rupert, etc. or they may be objects such as the chest where the body was hidden or the murder weapon. Similarly, objects may be invisible in some video frames, but nonetheless present. For example, the body of David is in the same room as the guests in all the movie although it is never seen. In fact, this fact is what constitutes the ultimate irony in the plot. We will restrict ourselves to visible objects for the time being, but we will shortly explain how invisible objects can be handled later on in the paper.
- **Activity Types:** Informally speaking, an activity describes the subject of a given video frame-sequence. Thus, for instance, in the movie "The Rope," `murder` is an activity, similarly `giving a party` is also an activity. There is no reason why multiple

activities may not simultaneously occur in a video frame-sequence – for instance, during the party someone may `open the chest` containing the body.

- **Event:** An `event` may be thought of as an `instantiation` of an activity type. For instance, the activity type `opening the chest` may refer to two separate events – Philip and Brandon opening the chest to put the body and Mrs. Wilson (the housekeeper) opening the chest to put some books in it, are two different events (both involving the same activity, viz. `opening the chest`). Note that activity types are general groups containing many events, and they will be stored implicitly in the form of a set of events of the same activity type.
  - **Roles:** The events that are of the same activity type are differentiated from each other by the set of objects involved in them. A `role` is a description of certain aspect of an activity. Roles may involve objects – for example, `victim` and `murderer` are roles in the activity `murder`. Roles may also be descriptions, such as the `murder motive` and `murder weapon`.
  - **Teams:** A `team` is a set of objects/descriptions that jointly describe an event, i.e. they are the instantiation of the roles in an activity type. Thus, for instance, if we talk of the event `murder` of David by Philip and Brandon, then the team involved consists of David in the role `victim`, Philip and Brandon, both in the role `murderer`, The rope plays the role of the `murder weapon`, while "`proving that they committed the perfect crime`" is the role `murder motive`. Each of these members of a team is called a `player`. Note that, conceptually video objects may be considred to be a special activity type in which the teams are always empty. Because of this, and as they will be the entities accessed most frequently, we have decided to represent them separately.

Example 1 For our example movie, we have chosen the following video objects and activity types with their corresponding events (the numbers in parentheses show the unique number given to them in our video database.)

Video objects: **1** - Philip, **2** - Brandon, **3** - Mrs. Wilson, **4** - Rupert, **5** - Janet, **6** - Kenneth, **7** - Mr. Kentley, **8** - Mrs. Atwater, **9** - The chest, **10** - The rope – murder weapon.

Events: **1** - Murdering of David with a rope by Philip and Brandon, **2** - giving a party where Philip and Brandon are the hosts and other people (objects 4 through 8) are the guests; there are three different events of the activity type "opening the chest" where **3** - the actors are Philip and Brandon in the first one where the body is placed in the chest; **4** - Mrs. Wilson opens it to put some books, but she doesn't see the body; finally, **5** - Rupert opens it at the end of the movie to see if the body was really there as he suspected. Events **6** through **13** are of type "find clues" and they will be given later in the paper.

2.3 Frame Sequences

In the area of video-databases, we observe that video-data may be classified as follows: a body of *raw video data*, `RVD`, for short, is a set of entities called *frames*. For the purposes of this paper, a frame may be thought of as a piece of video of short duration. In addition to the raw video data, there is a set, `ENT`, of "entities" – objects that are of interest as given above within the domain of a given `RVD`. Last but not least, there is an association map, λ, that specifies which objects occur in which video-frames. Marcus and Subrahmanian [12] have, in a series of papers, specified how multimedia database systems may be formally modeled along the lines briefly outlined above. In order to formally specify the association map, we need some elementary definitions and assumptions:

Assumption: We will assume that the set of frames, `RVD`, is the set $\{1, \ldots, n\}$ for some arbitrarily chosen, but fixed integer n. Likewise, we will assume that the set of objects is also the set $\{1, \ldots, k\}$ for some arbitrarily chosen, but fixed integer k. There is no loss of generality in making this assumption; it is convenient in providing a formal mathematical framework for reasoning about multimedia systems.

A *frame-sequence* is a pair $[i, j)$ where $1 \leq i, \leq j \leq n$. $[i, j)$ represents the set of all frames between i (inclusive) and j (non-inclusive). In other words, $[i, j) = \{k \mid i \leq k < j\}$. i (resp. j) is said to be the *start* (resp. *end*) of the frame-sequence $[i, j)$.

We may define a partial ordering, $\sqsubseteq$, on the set of all frame sequences as follows: $[i_1, j_1) \sqsubseteq [i_2, j_2)$ iff $i_1 < j_1 \leq i_2 < j_2$. Intuitively, $[i_1, j_1) \sqsubseteq [i_2, j_2)$ means that the sequence of frames denoted by $[i_1, j_1)$ precedes the sequence of frames denoted by $[i_2, j_2)$. As usual, we use $[i_1, j_1) \sqsubset [i_2, j_2)$ to denote that $[i_1, j_1) \sqsubseteq [i_2, j_2)$ and $j_1 \neq i_2$.

A set, X, of frame-sequences is said to be *well-ordered* iff:

1. X is finite, i.e. $X = \{[i_1, j_1), \ldots, [i_r, j_r)\}$ for some integer r, and
2. $[i_1, j_1) \sqsubseteq [i_2, j_2) \sqsubseteq \ldots \sqsubseteq [i_r, j_r)$.

For example, the set $X = \{[1, 4), [9, 13), [33, 90)\}$ is a well-ordered set of frame-sequences because $[1, 4) \sqsubseteq [9, 13) \sqsubseteq [33, 90)$. On the other hand, $X' = \{[1, 7), [5, 8)\}$ is not well-ordered because $[1, 7) \not\sqsubseteq [5, 8)$ and $[5, 8) \not\sqsubseteq [1, 7)$.

A set, X, of frame-sequences is said to be *solid* iff

1. X is well-ordered, and
2. there is no pair of frame-sequences in X of the form $[i_1, i_2)$ and $[i_2, i_3)$.

For instance, if we take $X = \{[1, 5), [5, 7), [9, 11)\}$, then X is not solid, even though it is well-ordered. It may be converted into an equivalent solid set of frame-sequences by replacing X by $\{[1, 7), [9, 11)\}$ which is solid. Intuitively, the requirement of solidity says that if we have two "abutting" frame-sequences $[i, j)$ and $[j, k)$, then these should be merged to form the frame-sequence $[i, k)$.

An *association-map* is a function, λ, such that for each entity $ent \in$ `ENT`, $\lambda(ent)$ is a solid set of frame-sequences. Intuitively, $\lambda(ent) = \{[i_1, j_1), [i_2, j_2)\}$ means that entity ent occurs in all frames in the frame-sequences $[i_1, j_1)$ and $[i_2, j_2)$.

Example 2 Recall the movie "The Rope". For purposes of annotation, we have divided the movie into 40 frames, 2 minutes each. Note that the granularity of this division is much coarser than the ideal case. We then determined where each object and each event occurs, thus specifying the association map for this movie. Object 1 (Philip) appears in most of the movie, hence the association map for Philip is $\lambda(o1) = \{[2,19), [20,21), [24,40)\}$ which indicates that Philip is seen in frames $2,\ldots,18, 20$ and $24,\ldots,39$. Mr. Kentley (7) and Mrs. Atwood (8) come to the party together, they stay together and they leave together. Hence, $\lambda(o7) = \lambda(o8)$.

Other examples of the frame sequences and the objects/events that appear in them are: Brandon: $\lambda(o2) = \{[2,21), [22,23), [26,40)\}$; Mrs. Wilson, the housekeeper: $\lambda(o3) = \{[7,11), [12,14), [15,18), [22,24), [26,29), [30,31)\}$; Rupert: $\lambda(o4) = \{[15,21), [22,29), [32,40)\}$; Philip and Brandon opening the chest: $\lambda(e3) = \{[2,3), [29,30)\}$; Mrs. Wilson opening the chest: $\lambda(e4) = \{[27,28)\}$; Rupert opening the chest: $\lambda(e5) = \{[37,38)\}$, etc.

2.4 Association Maps and Segment Trees

In this section, we show that association maps correspond exactly to line segments on the x-axis of the Cartesian plane, and hence, association maps may be stored using any method to store such collinear line segments, such as the well known segment tree (cf. Samet [15]). As an example consider the association map depicted in figure 1 corresponding to the movie "The Rope" according to the definitions given earlier in this section. In the figure, two types of entities that were discussed can be seen: video objects and events. As can be seen from the figure, we may represent the set of all line segments associated with `RVD` as a segment tree. The one difference between these segment trees, and the ordinary segment trees (cf. Samet[15]) is that a single object name may label multiple line segments. To differentiate between the two, we will refer to our data structure as a frame segment tree.

The observation that association maps correspond to line segments is not new – it has been made by several individuals in the multimedia community. In this diagram, the fact that video object number 1 (which, in this example, is Philip), appears in video-frames $2,\ldots,18$,20 and $24,\ldots,39$ is represented by the three shaded horizontal lines associated with object number 1. One of the key observations in our paper is that this object-frame map is just a set of line segments all of which are horizontal. Consequently, the frames in which a given object appears may be viewed as a set of line segments on the x-axis, obtained by projecting the shaded line segments shown in the figure onto the x-axis.

Hence, an appropriate data structure for this problem would contain:

- a segment tree representing the set of all line segments in `RVD`, and
- additional arrays, such as `OBJECTARRAY`,`EVENTARRAY` and `ACTIVITYARRAY` which provide additional indices to the nodes in the tree denoting where the corresponding data is located.

Figure 2 specifies this data structure for the interval $[1, 11)$, as applied to the association map shown in Figure 1. We will now explain the construction of the data structure shown in Figure 2.

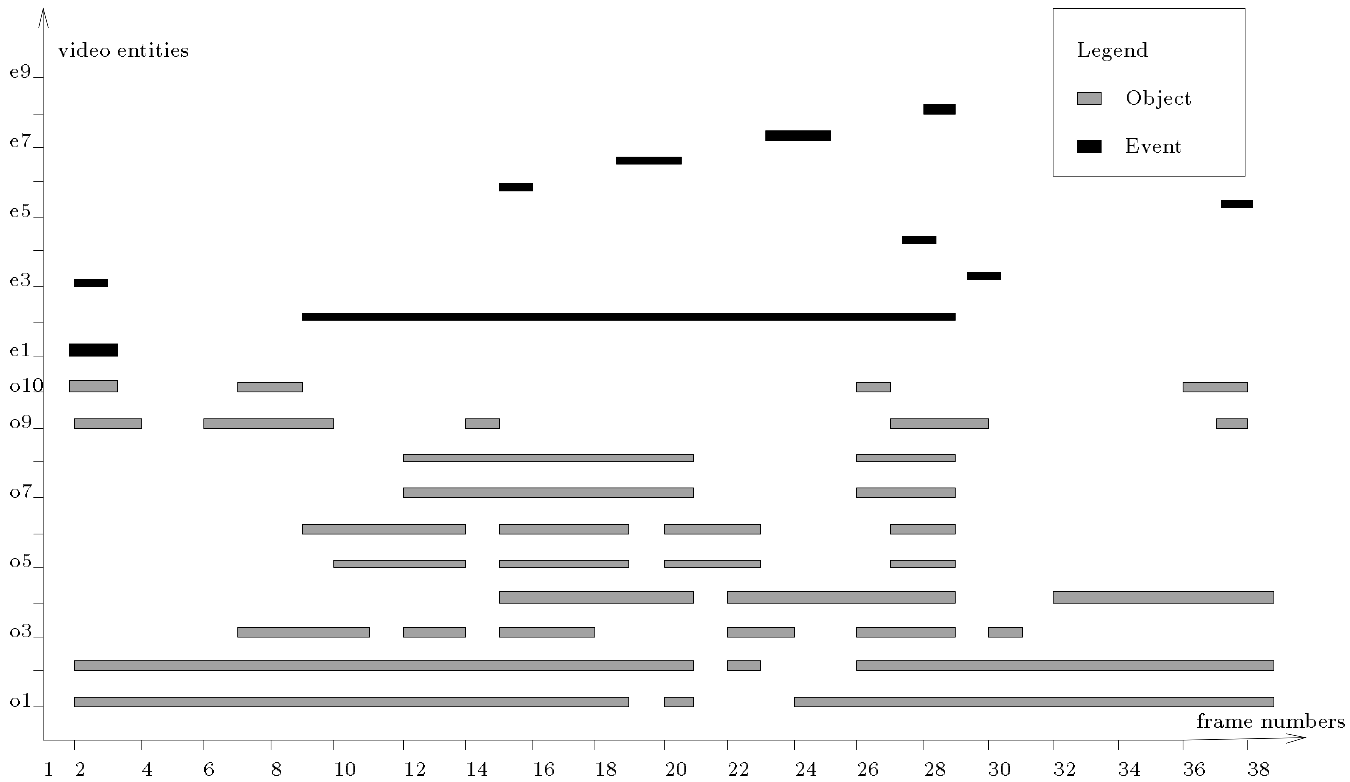

Figure 1: The Association Map for “The Rope”

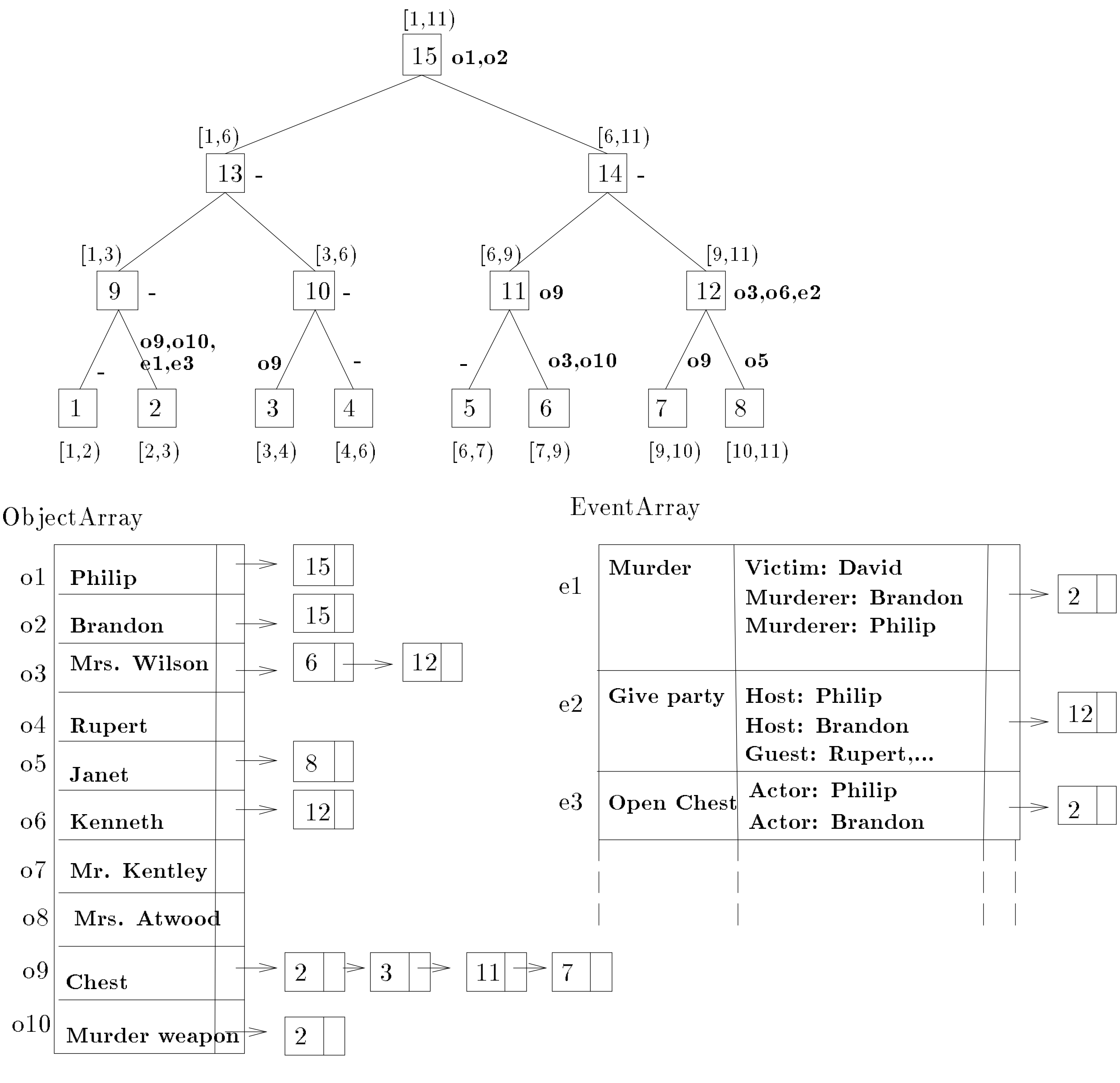

Figure 2: The Object-Array-Linked Segment Tree

Frame Segment Tree Representation of Video-Frames:

1. Each node in the frame segment tree represents a *frame-sequence* $[x, y)$ starting at frame x and including all frames up to, but not including, frame y.

2. The number inside each node may be viewed as a pointer to that node.

3. The set of numbers (in boldface) placed next to a node denotes the id-numbers of video objects and events that appear in the entire frame-sequence associated with that node. Thus, for example, if a node N represents the frame sequence $[i, j)$ and object o occurs in all frames in $[i, j)$, then object o labels node N (unless object o labels an ancestor of node N in the tree).

The `OBJECTARRAY`: The object array is an array whose i'th element denotes video object number i (denoted by oi). Associated with any element of this array is an *ordered linked list* of pointers to nodes in the frame segment tree. For example, Figure 2 shows that the linked list associated with object number 3 (o3) contains two pointers – to nodes **6** and **12**, respectively. These are the nodes in the frame segment tree described above, that are labelled with object number 3. Nodes **6** and **12** represent, respectively, the frame sequences $[7, 9)$ and $[9, 11)$. The nodes are stored in the order shown in Figure 2 because

$$[7, 9) \sqsubseteq [9, 11).$$

The astute reader may now wonder: the frame segment tree, by itself, contains all the information necessary in order to capture the association map. Therefore, why is it necessary to store an object array? The answer is that queries may be formed on multiple keys, and the object array facilitates efficient handling of certain kinds of queries. Consider, for instance, elementary membership queries. Here, we wish to find all frames in which object o occurs. Using the object array, it is a relatively simple matter – we just return the entire list associated with video object o; locating the entry for o is a constant time operation in this case.

The `EVENTARRAY`: As in the case of the `OBJECTARRAY` the event array stores the frame sequences, in the form of links to the frame segment tree, for each different event. Events are uniformly numbered from e1 to eN. For each event, the following information is stored: the activity type, the team as a list of pairs of the form (role: player) and finally, a list of tree nodes that indicates the frame sequences in which this event occurs.

The `ACTIVITYARRAY`: This is another index which simply stores, for each activity type, the list of events of that type. This will facilitate queries about a certain activity type as opposed to a certain event of that type. Similarly, the `ROLEARRAY` simply lists the name of the roles, since the roles are also stored by their unique id number in the video database. In addition to these, we assume that we have hash tables for objects, activity types and roles that map their names to their unique id numbers. We will not give details of these tables, since they're only used to locate the id numbers.

3 A Full Description of Video Databases

We are now in a position to formally describe a video-database. First, we assume $\mathtt{OBJ}$ is a set of video objects and $\mathtt{EVT}$ is a set of events found in the video database. Hence, $\mathtt{ENT} = \mathtt{OBJ} \cup \mathtt{EVT}$ ($\mathtt{ENT}$ is the set of all entities). The association map λ maps elements of $\mathtt{ENT}$ onto sets of frame sequences. For each event $e \in \mathtt{EVT}$, there exists a many to one mapping $\wp$ that maps each event to an activity type. For example, in our movie example, $\wp\mathtt{(e1)=murder}$.

Next, we assume that given an activity type A, there is a set, $\mathsf{ROLE}(\mathsf{A})$ associated with it; the members of the set $\mathsf{ROLE}(\mathsf{A})$ are strings denoting the names of the roles associated with activity A. Thus, for example, $\mathsf{ROLE}(\mathtt{murder}) = \{\mathtt{victim}, \mathtt{murderer}, \mathtt{murder\,motive}\}$, and $\mathsf{ROLE}(\{\mathtt{giving\,a\,party}\}) = \{\mathtt{host}, \mathtt{guest}\}$.

Given any event E of activity type A, we assume that there is a mapping, $\mathsf{PLAYERS}$ that maps $\mathsf{ROLE}(\mathsf{A})$ to the set, $\mathtt{OBJ} \cup \mathtt{EVT} \cup \mathtt{STR}$, $\mathtt{STR}$ is the set of all possible strings. The range of the map $\mathsf{PLAYERS}$ specifies the team involved in the event.

Definition 1 A video-database is a 9-tuple

$$(\mathtt{RVD}, \mathtt{OBJ}, \mathtt{EVT}, \lambda, \mathsf{ACT}, \Re, \wp, \mathsf{ROLE}, \mathsf{PLAYERS})$$

where

- $\mathtt{RVD}$ is a set of integers $\{1, \ldots, n\}$ for some n;
- $\mathtt{OBJ}$ is a set whose members are called "objects";
- $\mathtt{EVT}$ is a set whose elements are called *events*;
- λ is a that assigns to each entity *ent* $\in (\mathtt{OBJ} \cup \mathtt{EVT})$, a solid set of frame-sequences;
- ACT is a set whose elements are called *activity types*;
- $\Re$ is a set whose elements are called *roles*;
- $\wp$ is a map that assigns, to each event an activity type;
- ROLE is a map that takes as input, a frame-sequence $[i,j) \in \mathtt{EVT} \cup \bigcup_{o \in \mathtt{OBJ}} \lambda(o)$, an activity A in $\wp([i,j))$, and a role in $\Re(\mathsf{A})$, and assigns as output, a member of $\mathtt{OBJ}$;
- $\mathsf{PLAYERS}$ is a map that takes as input an event and its activity type, as output returns a mapping from the roles of the activity to the entities in the database and to strings.

4 Data Structures

In this section, we will define data structures that are necessary in order to store the video data that we are interested in dealing with. There are four types of structures that need to be defined: a *frame segment tree* that allows us to specify which events (and their relevant players, teams, etc.) and video objects occur in which segments of video; an `OBJECTARRAY` that allows accessing video-segments using objects as keys, an `ACTIVITYARRAY` that allows accessing video-segments using activity types as keys, and an `EVENTARRAY` that facilitates accesses using events as keys. Of these, the key component is a special kind of segment tree.

4.1 Frame Segment Trees

In this section, we define a special kind of segment tree called a frame segment tree (FS-Tree, for short) that is used for multimedia data. The structure of nodes in the frame segment tree may be defined as follows. Note that in this data structure, the players are restricted to video objects It is straightforward to augment the data structures to allow players of any type.

```
type treenode = record of
     start    : integer;  /* starting frame number */
     finish   : integer;  /* ending frame number */
     objlist  : ^objnode ; /* list of objects associated with the node */
     evtlist  : ^evtnode;  /* list of activities associated with the node */
     lchild   : ^treenode ; /* left child */
     rchild   : ^treenode;  /* right child */
     end;

type objnode = record of
     objid  : integer ; /* object number */
     next    : ^objnode; /* next node in list */
     end;

type evtnode = record of
     evtid : integer; /* activity number */
     next   : ^evtnode; /* next node in activity list */
     end;
```

Assumption: We will assume that for any node N of type `treenode`, N.`objlist` and N.`evtlist` are maintained in ascending order w.r.t. object number and event number, respectively. Thus, for example, if we consider node 9 in Figure 2, then the `objlist` associated with this node will be stored in the order **o3,o10**.

4.2 The Other Arrays

The `OBJECTARRAY`, the `ACTIVITYARRAY`, and the `EVENTARRAY` may be defined as arrays whose elements have record structures `objectrec`,`activityrec` and `eventrec` respectively. Below, we assume that `Val1,Val2,Val3,Val4` are some constants.

```
type objectrec = record
     name   : string; /* name of object/
     frames : ^frameseqlist; /* pointer to list of tree nodes */
     end;

type activityrec = record
     name   : string;
     events : ^evtnode;
     end;

type eventrec = record
     acttype:  integer;
     teamlist: ^playernode;
     frames : ^frameseqlist;
end;

type playernode = record of
     player: integer; /* object number */
     role  : integer; /* role number */
     next : ^playernode; /* next player in the team */
     end;

type frameseqlist = record
     segments : ^treenode; /* points to a tree node */
     next : ^frameseqlist; /* next node in list */
     end;

eventarray : array[1..Val1] of eventrec;
activityarray : array[1..Val2] of activityrec;
objectarray : array[1..Val3] of objectrec;
rolearray: array[1..Val4] of string;
```

Using this formal description of the data structures, we may now proceed with:

1. listing of the elementary queries that can be posed to our system,
2. development of the algorithms to execute these queries,
3. complexity analysis of the algorithms for each query, and
4. a formal rendering of algorithms to efficiently update the above data structure when either new entities in existing videos become important.

5 Query Processing

In the preceding section, we specified a data structure to store video information. In this section, we will try to show that this data structure facilitates the efficient execution of different types of queries. We will give examples of such queries and explain how each can be answered using our data structure. In addition to developing algorithms for handling such queries, we will analyze the algorithmic complexity of these procedures.

5.1 Elementary Object Queries

This is a query of the form: "Given the name of an object, find all video frames in which a given object occurs."

Example: "Find all the video frames in which the murder weapon (object 10) is seen." This query will then return the answer: video frames 2,7,8,26,37,38, or it may simply return a set of frame sequences capturing the above video frames, namely { [2,3),[7,9),[26,27),[37,39)}. Similarly, the query "find all video frames in which Brandon is seen" will return the answer: { [2,19),[20,21),[24,40)}.

Method: The query may be processed by first finding the id of the object in question in the `OBJECTARRAY` using the name given in the query and the hash table mentioned earlier. Then, follow the link `OBJECTARRAY`.frames and create a set of frame sequences corresponding to the start and end points of the tree nodes for this object, and finally merge the frame sequences to obtain a solid set.

Complexity: As finding the object in `OBJECTARRAY` can be done in constant time, the time required is proportional to (i.e. linear in) the number of segments in which the object appears, which is also the size of the output.

5.2 Elementary Activity Type Queries

This is a query of the form: "Find all the video frames in which events of a given activity type occurs."

Example: "Find all the video frames in which someone is opening the chest." There are three different events corresponding to this activity type, (in practice activity types will be more general, such as `open an object`, but we will use "open the chest" for our example movie.) where the actor is either Philip and Brandon, or Mrs. Wilson, or Rupert. Hence, we have to find the frame sequences for all these events and merge them. Finally, we get the answer: { [2,3),[27,28),[29,30),[38,39) }.

Method: The query may be solved by first locating all the events corresponding to the given activity type. This can be done by examining `ACTIVITYARRAY` and the hash table, then following the list pointed to by `ACTIVITYARRAY`.events. The set of frame sequences for all such events are obtained one by one by following the links in the `EVENTARRAY` and merging the results into solid sets of frame sequences. These sets of frame sequences are then merged into a final solid set to give the final output.

Complexity: As in the case of Elementary Object queries, the time required is proportional to (i.e. linear in) the number of segments associated with the events of the activity type under consideration.

5.3 Detailed Activity Queries – Event Queries

This is a query of the form: "Find all video frames in which one of a given set of events occurs, where the events are specified by the activity type and the roles of specific objects involved in this activity" Since the players provided in the query may be a partial list of all the players of an event, there may be more than one event satisfying such a query.

Example: "Find all the video frames in which Rupert is given a clue by Philip." Although both Philip and Brandon are both cold blooded murderers, Brandon is the brains behind the team with Philip only being his helper. Philip is a very nervous person and he will inadvertently give away some clues to Rupert who closely scrutinizes his every move. First, when Brandon tells a story about his strangling a chicken when they were kids, he shows an unexpected reaction. Rupert, on the other hand, knows this story is true (cf. frame 17). Second, Brandon gives some books to Mr. Kentley and ties them up with the rope – the murder weapon. When, Philip sees this rope, he is petrified (frame 26), and of course he is questioned by Rupert about both these events. Hence, the original query should return: { [17,18),[26,27)}.

Method: This query is similar to an Elementary Activity Query except that the search to locate relevant events is more complex. For this, first the activity type in question is located, ex. `Finding Clue`. Then, all the events listed under this activity type are searched. For all such events, it is checked to see whether teams contain all of the necessary players, ex. `Finder:Rupert` and `Giver:Philip`. Then, for all these events, the link to the frame segment tree is followed and the solid set of frame sequences is formed. Finally, all these sets are merged to give the final result.

Complexity: The time required is proportional to the number of segments in the events satisfying the conditions, plus the cost of checking each event of a given activity type for role/object pairs in the query. The cost of this check is the number of role/object pairs in the events of given activity type.

5.4 Object Occurrence Queries

This is a query of the form: "Find all objects that occur in a given set of frame sequences."

Example: "Find all the objects that are present in frame sequence [10,29)." Incidentally, this frame sequence corresponds to the whole duration of the party in the movie. Hence, it will give us everybody involved in the party one way or another. This query will return all the video objects in the database including the chest and the murder weapon. The reader may notice that if the invisible object `David's body` were stored in the database, then it would be returned by this query. Ironically, it is also a part of the party.

Method: The query may be solved by searching the frame segment tree starting from the root for the given set of frame sequences. If the node being visited intersects with any of the frame sequences in the set, all the objects stored in this node are added to the output set of objects, and then both the left and the right children of the node are visited. It is possible to split the set of frame sequences for the children so that only those sequences that may possibly intersect with the corresponding child are included in that call. Each time a set of new objects are added to the output list, duplicates need to be removed. This is a costly operation. Using a bag which clusters objects is a step towards reducing the complexity of this operation.

Complexity: As this information can be computed via the segment tree, the cost involved is proportional to the number of those nodes in the segment tree whose parents' interval overlaps some given frame sequence in the query times $\log(numberofinputframesequences)$ plus the number of objects in each of those nodes. It cannot be computed any faster because each such node needs to be checked. (The log factor is the cost of checking whether a given node's interval overlaps with some frame sequence in the query).

5.5 Activity Type Occurrence Queries

This is a query of the form: "Find all the activities that occur in a given set of frame sequences."

Example: "Find all the activities that occur in the first 30 frames." This will give the activities of murder, opening the chest, giving a party, playing the piano.

Method/Complexity: This is very similar to the object occurrence query. The tree is searched using the same algorithm. This time the events are collected and at the end of the search, the collected events are grouped into their respective activity types and this list is returned.

5.6 Conjunctive Queries

If one examine the queries given above, one can see that the above queries can be grouped into a few query types, namely (1) queries that take as input, a specification, and return as output, a set of frame sequences, (2) queries that take as input, a set of frame sequences, and return a set of objects, (3) same type of input as 2, but returns as output, a set of events, (4) there are some other queries that we have not discussed, which take the same types of input as 2 and 3, but they return as output return activity types, role of a specific object and etc.

Examples of such queries are the query that takes as input, a set of frame sequences, and returns as output, all the objects and events in those frames. Similarly, the query that takes as input, an object and an event, and returns as output, the set of all frames in which this object is seen and the event occurs.

Example: "Find all the frames where people are eating and at the same time the chest can be seen." This actually happens in the movie, since Brandon moves the appetizers from the table to the top of the chest before the guests arrive. This makes Rupert's even more suspicious when he learns of this from his talk with Mrs. Wilson.

Method: The algorithms depends on the specific query posed. For the above conjunction, we execute the simple object and event queries for the relevant parts of the query. Then we intersect these frame sequences to obtain the result. Note that the frame sequences returned by these queries are ordered, hence the intersection operation is not very costly. This approach is particularly useful since it makes use of the previous code.

Complexity: The time required equals the sum of the time required for each component plus the cost of taking the intersection of the answers. If answers from components happen to be sorted, their intersection can be found in the order of their size. Otherwise taking the intersection is $O(n \log(n))$ where n is the sum of the sizes of the individual queries.

5.7 Compound Queries

These queries involve the relationship between different objects and events. In this section, we will only consider non-temporal queries. For example, suppose we consider the query "Find all the objects present in the video when a certain event (specified by the activity type and a team) occurs." In general, answers to compound queries are found by directing the output of one function as input to another function.

Example: "Find all the people present in the scene when Brandon and Rupert discuss whether murder is a privilege reserved for a small group of people." This happens in frames 19,20. Hence, the objects seen in these frames are Philip, Brandon, Rupert, Kenneth, Janet, Mr. Kentley and Mrs. Atwater.

5.8 Other Types of Queries

As we have discussed earlier in the paper, we may have different types of video objects, such as invisible objects. We may also have audio events, i.e. events referring to the dialogues in the movie such as someone singing. The queries involving these objects will be very similar to the ones given above since, at the end, invisible objects are objects and audio events are still events with the same properties. We do not specify how these can be stored, however one may easily think of rather straightforward extensions of our data structure to handle these objects. The query processing algorithms may then be modified slightly to process queries involving these entities as well as the previous ones. Another query type could be a general query where the user specifies a complex condition such, as the combination of several of the queries above. Solving these queries is beyond the scope of this paper.

In addition, one may ask temporal queries such as query number 6 in the Introduction. The semantics of such queries has not yet been specified by the authors, and will be addressed in a future work. For example, operators such as after, right after, before and overlaps should be included (cf. work of Ghafoor et. al. [10]). In addition to this, the semantics of operators such as "before" and "after" has to be established clearly. For instance consider the query: "Find all the events Rupert is involved in after he finds out that Brandon (murderer of David) told Kenneth (Janet's old boyfriend) that he feels Kenneth may have better chances with Janet (David's girlfriend) after all." Suppose that the specified event happens in two places, then after may mean after the first occurrence or after the last occurrence of this event. We have deferred the treatment of such queries to a future paper.

6 Updates

In the real-world, it is likely that a video database system organized along the lines described thus far will change over a period of time. We have classified all changes into the following two categories.

1. **Insertion or Deletion of an Entity:** Examples of such insertions are the insertion of a new object, or a new activity type, or a new event into the database, or the insertion of a new player into an event. Similarly, some objects, events, etc. may also need to be deleted with all the corresponding information stored about this entity in the video database.

   There may be various reasons for such operations. For example, it may be the case that an object o occurred in some frame f. However, the significance of o may not been realized earlier, and therefore, the need to store o, and the fact that it occurred in f may not have been realized before. It is not far fetched to assume that, given the above annotation of the movie "The Rope," an experienced movie watcher will immediately ask the query "Find me the frames in which Hitchcock appears personally." Since Hitchcock is not an annotation in the video, this query cannot be answered. Hence, someone may decide later on that this object needs to added to the database.

   Another reason for such a change in the movie database is the possibility that some of the entities were wrongly annotated. For instance, one of the objects may have identified incorrectly. Hence, this object may need to be deleted altogether and re-inserted into the data structure. Consider the case where an Elvis "look-alike" has been categorized as Elvis Presley, but later inspection may show that it was not, in fact, Elvis Presley, and hence, this may necessitate the deletion of the object "Elvis Presley".

2. **Modification of Frame Sequences:** Since, the databases may contain incomplete information, we may later find out that an object or an event actually occurs in some additional frames which weren't in the initial list. Thus, we need to modify the database by inserting new frame sequences into the necessary data structures. Similarly, we may find out that by mistake, some objects (events) have been listed as occurring in certain frames even though they do not, in fact, occur therein. For this reason, we may need to delete, from the frame segment tree, some of the frame sequences associated with an object, as well as the corresponding array(s).

It is easy to see that some of the above updates are pretty straight forward. In particular, some updates are of a "relational" nature and do not involve temporal properties. The updates that require modifications to the frame segment tree can be broken up into more elementary operations. Such updates include: (1) Inserting a set of frame sequences into the frame segment tree and the `OBJECTARRAY` for a specific object, (2) deleting a set of frame sequences for a specific object from the database, (3-4) inserting and deleting as above for an event. The updates listed earlier in this section can be encoded by treating these operations as the elementary ones. For example, deleting an object can be done by locating the object, retrieving the set of frame sequences for this object, and then deleting all of these sequences. In the sections below, we will give two of these operations, (1) and (2) in detail. Subsequently, we will quickly outline the methods for implementing (3) and (4) – they are similar to (1) and (2).

6.1 Insertion of a set of frame sequences for an object

For this operation, the name of the object being inserted, and the solid set of frame sequences in which this object appears, are given as input. The data structure is modified in such a way that the given object appears in the specified frames.

Method:

1. If the given object is not in `ObjectArray` then create an entry for it in `ObjectArray`.
2. For each frame sequence in the input list do:
   (a) Put the frame sequence into the tree and mark the necessary tree nodes with the ID of the object. This step is further described below.
   (b) For each tree node N marked by the object do:

      Insert a node F into the `frames` list of the object in `ObjectArray` such that F has a pointer to the tree node N. Note that the `frames` lists must be preserved in sorted order.

Recall that, we restrict the boundaries of the nodes in the frame segment tree to the start and end points of frame-sequences in which entities occur. Hence, a frame sequence that is being inserted may not fit into any of the nodes in the tree completely. In this case, we need to split the boundaries of some nodes. Let the frame sequence we are inserting into the tree have the boundaries $[a, b)$. There are three cases to consider:

- **Case 1:** "There is a leaf l in the tree with boundaries $[c, d)$, where $a <= c < b$ and $d > b$." In this case, node l will have two new children with the boundaries: $[c, b)$ and $[b, d)$.
- **Case 2:** "There is a leaf l in the tree with boundaries $[c, d)$, where $c < a < d$ and $d <= b$." In this case, the two new children of the node l will have the boundaries: $[c, a)$ and $[a, d)$.

- **Case 3:** " There is a leaf l in the tree with boundaries $[c, d)$, where $c < a$ and $b < d$." In this case, l will have two children with the following boundaries; child(1): $[c, a)$ and the child(2): $[a, d)$. In addition to this, child(2) will have two children also with the boundaries: $[a, b)$, and $[b, d)$.

Note that as the boundaries of the parent node are not changed in any of the above cases, none of the other objects pointing to the parent node is affected by this change in the tree. Hence, the updates can be easily accomplished. On the other hand, the tree may become unbalanced if there are many insertions of this type. Standard tree balancing techniques may be easily adapted to balance such trees when necessary.

Complexity: This task can be performed in time $O(nh)$ where h is the height of the segment tree, and n is the cardinality of the (solid) set of frame-sequences associated with the object being inserted.

6.2 Deletion of a set of frame sequences for an object

As in the case of insertion, this operation takes as input, the name of the object, and the solid set of frame sequences (in which this object appears) that are to be deleted. The algorithm for this operation is given below:

Method: The deletion algorithm may then be implemented as follows:

1. If the object is not in `ObjectArray` then halt and report failure.
2. Otherwise, for each frame sequence $[a, b)$ in the input list do:
   (a) If there exists a node N in the segment tree with a frame sequence of the form $[c, d)$ in the `frames` list where $c <= a$ and $b <= d$ then:
      i. Remove the object from the `objlist` of the corresponding tree node.
      ii. Remove N from the fs list.
      iii. Insert the object into the tree with the frame sequence set {[c,a),[b,d)}. Note that if $c = a$ $(b = d)$ then $[c, a)$ $([b, d))$ is omitted from the set.
   (b) If there exists a frame sequence of the form $[c, d)$ in the corresponding `frames` list where $c > a$ and $b <= d$ then:
      i. Remove the object from the `objlist` of the corresponding tree node.
      ii. Remove N from the `frames` list of the corresponding `ObjectArray` node.
      iii. Insert the object into the tree with frame sequence set {[b,d)}. Note that if $b = d$ then $[b, d)$ is omitted from the set.
   (c) If there exists a frame sequence of the form $[c, d)$ in the corresponding `frames` list where $c <= a$ and $b > d$ then:
      i. Remove the object from the `objlist` of the corresponding tree node.
      ii. Remove N from the `frames` list of the corresponding `ObjectArray` node.
      iii. Insert the object into the tree with the frame sequence set {[c,a)}. Note that if $c = a$ then $[c, a)$ is omitted from the set.

(d) If there exists a frame sequence of the form $[c, d)$ in the corresponding `frames` list where $c > a$ and $b > d$ then:

   i. Remove the object from the `objlist` of the corresponding tree node.

   ii. Remove N from the `frames` list.

3. If all the frame sequences in `frames` have been removed, then release the corresponding `ObjectArray` slot.

6.3 Insertion/Deletion of a set of frame sequences for an event

In this case, an event id is provided as the input. An event may have been identified by its activity type and its team, but for the correct execution of this operation, the unique id of the event in question must be provided. After the event is identified and the set of frame sequences are given, the operations will be exactly the same, except that they will access the `EVENTARRAY` and the `ACTIVITYARRAY` instead of the `OBJECTARRAY` as in the previous cases. We will not give the details of these operations here.

6.4 Other Operations

We will briefly mention other simple updates that may be performed on the video database.

Player Insertions: To insert a new player into the database, the name of the object, the role of the object and the ID of the relevant event is given as input. Then the corresponding event e is located in `EventArray`, and a new node p is inserted into the `players` list of e with the ID of the object (found from the hash table) and the given role.

Player Deletions: In order to delete a player from certain events in the video database, the name and the role of the player, together with the corresponding event ID ,are given as input. The entry for the event is then located in `EventArray`. The player playing the given role is then searched for in the `players` list with its ID. If it is found, the node containing this information is deleted from the list of players.

6.5 Storing Multiple Videos

Thus far, we have only focused on storing a single video. In the future, users will need the ability to query a video library for certain objects. The user may ask queries such as: "Find all the video frames in your database (not necessarily from a specific movie) in which Alfred Hitchcock appears." To handle such queries, it is possible to maintain a global list of objects, events, activity types in a kind of array structure like the `OBJECTARRAY` and the other similar arrays. This array will contain a list for each object, for example, a list of videos that contain that object. Similarly, global event arrays, global role arrays, etc. will be maintained as in the case of a single video. Hence, a query such as the one above may be answered by first locating relevant videos, and then directing an elementary object query to the indices associated with these individual videos. This also has the possible advantage of different processes sharing a global hash name server for all entities in the database. Our system can implement this capability quite easily.

7 Implementation of AVIS: An Advanced Video Information System

The algorithms described in this paper have been implemented within a prototype system called AVIS ("Advanced Video Information System"). Subsequently, AVIS has been hooked up to a much more powerful system for software integration called HERMES ("Heterogeneous Reasoning and Mediator System") [16]. An advantage of hooking up AVIS and HERMES is that all the parsing capabilities of HERMES are automatically available to AVIS. Furthermore, as HERMES allows complex queries that integrate data and reasoning across multiple domains, it is now possible to ask queries of the form *Find all frame sequences F and an actor A such that actor A also appeared in some movie M, and such that actor A appears in the frame sequences F.* This query may require access to a relational database (say INGRES) that specifies which actors occured in which movie, as well as access to the video-content of the "The Rope" which is provided by AVIS. The video content of "The Rope" may be stored using the data structures described in this paper.

We now quickly describe the implementation of AVIS, followed by the integration of HERMES and AVIS.

AVIS: AVIS implements all the query processing algorithms described in this paper. It consists of approximately 4,000 lines of C code, and runs on the Unix/XWindows platform. The basic functionality of AVIS is captured by the following pre-defined predicates each of which access functions (having the same name as the predicate described below).

1. `event_to_frames(Event, Framelist)`: Given an event (which is a complex type), this predicate succeeds just in case `Framelist` is the list of frames in which the specified event occurs. In particular, an event consists of the activity type, together with players' names and their roles. The output frame list is a solid set of frames,

2. `frames_to_objects(Frame_ranges, Objects)` : This predicate succeeds just in case `Object` is the set of all objects that occur in a frame occurring in the list, `Frame_ranges`, of frame ranges. An example of such a call could be

   $$\leftarrow \texttt{frames\_to\_objects}([<5\ 10><20\ 30>], \texttt{Obj})$$

   which asks for all objects that occur in one of the frames $5, \ldots, 9, 20, \ldots, 29$.

3. `frame_lists_to_objects(Frame_range_list, Objects)`: Given a *list of lists* of ranges of frames, this function returns the objects which occur in all of the lists. Thus, for instance, if we call this predicate with the query:

   $$\leftarrow \texttt{frame\_lists\_to\_objects}([[<2\ 5><24\ 28>][<11\ 14><31\ 39>]], \texttt{Objs})$$

   then this would find all objects that occur in

   - one of the frames $2, \ldots, 4, 24, \ldots, 27$ **and**
   - one of the frames $11, \ldots, 13, 31, \ldots, 38$.

4. `object_to_frames(Objname, Frames)` : This predicate find the frames in which a specified object occurs. For instance, the query

$$\leftarrow \texttt{object\_to\_frames}(\text{"}\texttt{Rupert}'', \texttt{Frames})$$

returns a list, say $[< 15\ 24 >< 30\ 35 >]$ specifying that Rupert appears in frames $15, \ldots, 23$ and $30, \ldots, 34$.

5. `objects_and_events_to_frames(Objectlist, Eventlist, Framelist)`: This predicate takes a list of objects and a list of events (possibly with specified players in specified roles). It succeeds just in case `Framelist` is the list of frames in which these events occur (with specified characters, if any, in specified roles) and the specified objects occur in these frames as well. As before, the arguments to this predicate are complex types.

6. `frames_to_activities(Frame_range, Activities)`: Similar to the `frames_to_objects` predicate.

7. `frame_lists_to_activities(Frame_range_lists, Activities)`: Similar to the `frame_lists_to_objects` predicate.

HERMES: When multiple databases, data structures and/or software packages need to be integrated, this is typically accomplished by writing a chunk of program code (called the "mediator" program) that uses the source code of the databases and/or data structures involved in order to "glue" them together.

HERMES is a system that has been developed at the University of Maryland [16] for creating mediators. In HERMES, a mediator is a program, written in a logic-based language, that we shall now describe. When a particular set of software packages and/or databases are being integrated, the *mediator author* is charged with the responsibility of writing the mediator, using as his "tools", the mediator development toolkit.

As the mediator language must be able to "extract" information from different databases and/or software packages, and, as details of the implementation of these packages may be very hard to fathom/discover, our integration model will use the *existing operations implemented within the packages being integrated*, and use the results of these computations (which are performed within the software packages involved), in further logical manipulations.

AVIS + HERMES: We now quickly describe, through an example, how AVIS and HERMES "work together", providing functionality that neither could provide by itself. To see this, consider the query: *Find all frame-sequences (from a movie such as "The Rope") in which an actor who also acts in a specified movie M, appears.* Let us suppose that there is a relation called `portfolio` stored under the INGRES DBMS that has the schema `(name,movie)` specifying the movies acted in by a given actor. Furthermore, suppose we have a flat file relation called `cast` (associated with just the single movie, "The Rope") with the schema `(name,role)` specifying which actor played which role in "The Rope". This query can be encoded as a special predicate (let us call it `spquery(Actor, Movie, Results)` that succeeds just in case `Results` is in the appropriate set of frame sequences associated with this query. This can be encoded as a rule in the mediator of the form:

$$\begin{aligned}\texttt{spquery(Movie,Results)} \leftarrow\ & \texttt{in(P,ingres:equal('portfolio',movie,Movie))}\ \& \\ & \texttt{in(Q,flatfile:equal('cast',name,P.name))}\ \& \\ & \texttt{object\_to\_frames(Q.role,Results):1}.\end{aligned}$$

The query illustrates some features of the integrated HERMES + AVIS system. In particular:

- The constraint `in(P, relation:equal ('portfolio', movie, Movie ))` performs a select operation on the INGRES relation `portfolio`, producing a set of records related to the given movie.
- The constraint `in(Q, flatfile:equal ('cast', name, P.name ))` performs a select operation on the flat file relation `cast`, identifying all tuples associated with the objects appeared in the given movie.
- Finally, we use AVIS to search our data structures for the appropriate video-frame-ids from "The Rope" that contain the character (role) played by this actor. Note that the predicate `object_to_frames(P.role,Results):1` accesses the AVIS algorithms and AVIS data structures, in the way described within this paper.

Figure 3 shows the window when the query `spquery` is issued. Figure 4 shows the solution returned by this query. In particular, when video frames are returned as part of the answer, the system "merges" them together into a composite (low-resolution image) answer which the user may view at his/her convenience either as a composite picture, or as a sequence of video. For instance, if the user "clicks" with his mouse button on the answer `comp0.pnm`, then a list of 2 video frames is showed on-screen. Figure 5 shows this. The user may subsequently request to see a specific image (say `frame2.ras.gz`) – he does this again by clicking with his mouse button on `frame2.ras.gz` (cf. Figure 4). This brings that figure up as shown in Figure 6.

8 Related Work

Over the last couple of years, there has been a small, but noticeable, spurt of activity in the area of video databases. The primary aim of this paper is to develop techniques by which video may be organized and queried. Two works that are closely related are [5] and [9],

Gibbs et. al. [5] study how stream-based temporal multimedia data may be modeled using object based methods. However, concepts such as roles and players, the distinction between activities and events, and the integration of such video systems with other traditional database systems are not addressed.

Hjelsvold and Midtstraum [9] develop a "generic" data model for capturing video content and structure. Their idea is that video should be included as a data type in relational databases, i.e. systems such as PARADOX, INGRES, etc. should be augmented to handle video data. In particular, they study temporal queries – something we do not do in this paper. In contrast, the innovation in our approach is the use of well studied spatial (rather than temporal)' data structures, suitably modified, to query video data. We have used those intuitions to develop techniques for processing queries as well as for *updating* these video data structures – the first algorithms we know of for incorporating updates to video data.

Oomoto and Tanaka [14] have developed a language called VideoSQL for accessing video data. In contrast to their effort, in this paper we develop indexing structures and query processing algorithms for querying video data, and use a logical query language (cf. the work on HERMES [16]) to query the data.

Other work on video includes work by Davenport et. al. [4] who argue that segmenting video should not be done at the frame level. This is consistent with our rendition – segementing video at the frame level corresponds to a well-known data structure called the *unit* segement tree (cf. Samiet [15]) which is just like the segment tree described here except that leaves always must represent unit intervals, i.e. intervals of the form $[i, i+1)$. In contrast, by using segment trees instead, we allow leaves to have whatever granularity is needed to best represent the content of the video being annotated.

Other works on video indexing include specific disk-based techniques [3, 2], methods dealing with compression/image processing [1], and synchronization structures [10].

There has also been a good deal of work on picture retrieval systems [8, 7, 13] – however, these works deal with static images, whereas in contrast we have studied ways of organizing temporal sequences of such images (i.e. video).

9 Conclusions

Over the last few years, the ability to deliver movies to homes has increased substantially. The cable TV phenomenon is being enhanced slowly, but steadily, by systems that support the "pay-per-view" paradigm. Over the next few years, with the rapid concomitant evolution of the information super-highway, we expect to see widespread video-on-demand services becoming available. Such services will, in all likelihood, not just encompass commercial movies from the entertainment industry, but are likely to be broad and include movies about travel destinations, movies about educational items, movies and advertising material on consumer products, training manuals, etc.

Once an electronically accessible video library exists (and some such libraries exist even today, e.g. at the National Archives in College Park, Md.), the need to access these database "by content" assumes great importance. The primary aim of this paper is to develop techniques by which the content of these movies can be stored on-line so as to facilitate such accesses.

To accomplish this, we have presented a succinct theoretical description of video databases. We have then shown how such video databases may be stored electronically, and furthermore, we have designed polynomial-time query processing algorithms that traverse these data structures. In addition, as the "content" of these video databases may be updated from time to time, we have developed methods to implement such updates. Finally, we have developed an implemented system called AVIS ("Advanced Video Information System") that implements the various video-based queries described here.

References

[1] F. Arman, A. Hsu and M. Chiu. (1993) *Image Processing on Compressed Data for Large Video Databases*, First ACM Intl. Conf. on Multimedia, pps 267–272.

[2] S. Berson, S. Ghandeharizadeh, R. Muntz and X. Ju. (1994) *Staggered Striping in Multimedia Information Systems*, Proc. 1994 ACM SIGMOD Conf. on Management of Data, pps 79–90.

[3] H.J. Chen and T. Little. (1993) *Physical Storage Organizations for Time-Dependent Multimedia Data*, Proc. of the Foundations of Data Organization and Algorithms (FODO) Conf., Oct. 1993.

[4] G. Davenport, T.A. Smith and N. Pincever. (1991) *Cinematic Primitives for Multimedia*, IEEE Comp. Graphics and Applications, July 1991.

[5] S. Gibbs, C. Breiteneder and D. Tsichritzis. (1994) *Data Modeling of Time-Based Media*, Proc. 1994 ACM SIGMOD Conf. on Management of Data, pps 91–102.

[6] S. Gibbs and D. Tsichritzis. (1994) *Multimedia Programming: Objects, Environments and Frameworks*, ACM Press/Addison Wesley.

[7] V.N. Gudivada and V.V. Raghavan. (1993) *Design and Evaluation of Algorithms for Image Retrieval by Spatial Similarity*, to appear in: ACM Transactions on Information Systems.

[8] V.N. Gudivada, V.V. Raghavan and K. Vanapipat. (1994) *A Unified Approach to Data Modeling and Retrieval for a Class of Image Database Applications*, to appear in: "Multimedia Database Systems" (eds. S. Jajodia and V.S. Subrahmanian), Springer.

[9] R. Hjelsvold and R. Midtstraum. (1994) *Modeling and Querying Video Data*, Proc. 1994 Intl. Conf. on Very Large Databases, pps 686–694, Santiago, Chile.

[10] M. Iino, Y.F. Day and A. Ghafoor. (1994) *An Object-Oriented Model for Spatio-Temporal Synchronization of Multimedia Information*, Proc. 1994 Intl. Conf. on Multimedia Computing and Systems, pps 110-120, IEEE Press.

[11] J. Lu, A. Nerode, V.S. Subrahmanian, Hybrid Knowledge Bases, *IEEE Transactions on Knowledge and Data Engineering*, 1994. To appear. Available as technical report CS-TR-3037, University of Maryland.

[12] S. Marcus, V.S. Subrahmanian, *Multimedia Database Systems*, 1994. Submitted for publication.

[13] W. Niblack, et. al., The QBIC Project: Querying Images by Content Using Color, Texture and Shape, IBM Research Report, Feb. 1993.

[14] E. Oomoto and K. Tanaka. (1993) *OVID: Design and Implementation of a Video-Object Database System*, IEEE Trans. on Knowledge and Data Engineering, 5, 4, pps 629–643.

[15] H. Samet *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1989.

[16] V.S. Subrahmanian. (1995) *HERMES: A Heterogeneous Reasoning and Mediator System*, submitted for journal publication.

[17] G. Wiederhold, Mediators in the Architecture of Future Information Systems, *IEEE Computer*, pp. 38–49, March 1992.

[18] G. Wiederhold, Intelligent Integration of Information, *Proceedings of the ACM Conference on Management of Data*, pp. 434–437, 1993.