vldb manuscript No.
(will be inserted by the editor)

Rule-Based Spatio-Temporal Query Processing for Video Databases*

Mehmet Emin Dönderler[1], Özgür Ulusoy[2,], Uğur Güdükbay[2]**

[1] Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-5406, USA
[2] Department of Computer Engineering, Bilkent University, Bilkent 06800 Ankara, Turkey

Received: date / Revised version: date

* This work is supported by the Scientific and Research Council of Turkey (TÜBİTAK) under Project Code 199E025. This work was done while the first author was at Bilkent University.

** Corresponding author - e-mail: oulusoy@cs.bilkent.edu.tr, Tel: +90 (312) 290 1577, Fax: +90 (312) 266 4047

Abstract In our earlier work, we proposed an architecture for a Web-based Video Database Management System (VDBMS) providing an integrated support for spatio-temporal and semantic queries. In this paper, we focus on the task of spatio-temporal query processing, and also propose an SQL-like video query language that has the capability to handle a broad range of spatio-temporal queries. The language is rule-based in that it allows users to express spatial conditions in terms of Prolog-type predicates.

Spatio-temporal query processing is carried out in three main stages: query recognition, query decomposition, and query execution.

Key words spatio-temporal query processing – content-based retrieval – inference rules – video databases – multimedia databases.

1 Introduction

The interest for multimedia databases, especially for video databases, is growing rapidly. The research that was first started tackling the issue of content-based image retrieval by low-level features (color, shape, and texture) and keywords, [4,6,12,35], has progressed in time towards video databases dealing with spatio-temporal and semantic features of video data [5,16,20,27,29,41]. There has also been some work on picture retrieval systems to enhance their query capabilities using the spatial relationships between objects in images [6,7].

First attempts for supporting content-based video retrieval were initiated by applying the techniques devised for image retrieval to video databases since video can basically be regarded as a consecutive sequence of images ordered in time [12, 39]. Some prototype systems were designed and implemented, such as *VideoQ*, *KMED*, *QBIC*, and *OVID* [5,7,12,31]. Furthermore, querying video objects by motion properties has also been studied extensively [13,22,24,30,38]. Some examples on using semantic properties of video data for querying video collections can be found in [1,16,18]. Nonetheless, to the best of our knowledge, no proposal has been made thus far for a generic, application-independent Video Database Management System (VDBMS) that targets to support spatio-temporal, semantic, and low-level queries on video data in an integrated manner.

In our earlier work, we proposed a novel architecture for a VDBMS, which provides an integrated support for both spatio-temporal and semantic queries on video data [9]. A spatio-temporal query may contain any combination of directional, topological, 3D-relation, external-predicate, object-appearance, trajectory-projection, and similarity-based object-trajectory conditions. The system responds to spatio-temporal queries using its knowledge-base, which consists of a fact-base and a comprehensive set of rules implemented in Prolog, while semantic queries are handled by an object-relational database. The query processor interacts with both the knowledge-base and object-relational database to respond to user queries that contain a combination of spatio-temporal and semantic queries. Intermediate query results returned from these two system components are integrated seamlessly by the query processor, and sent to Web clients. The architecture is extensible in that it can be augmented easily to provide integrated support for low-level video queries as well, in addition to spatio-temporal and semantic queries on video data.

The focus and contributions of this paper are on the spatio-temporal video query processing; therefore, issues related to semantic and low-level video queries are not discussed. Our rule-based spatio-temporal video query processing strategy is explained in detail. Moreover, an SQL-like textual query language is proposed for spatio-temporal queries on video data. The language can be used to query the knowledge-base of the system, proposed in [9], for object trajectories, spatio-temporal relations between video objects, external predicates, and object-appearance relations. It is very easy to use even by novice users. In fact, it is relatively easier to use compared with other proposed query languages for video databases, such as *CVQL*, *MOQL*, and *VideoSQL* [19,25,31]. Furthermore, it offers great expressiveness for creating complex spatio-temporal queries, thanks to its rule-based structure. Similarity-based object-trajectory and trajectory-projection query conditions are processed separately from spatio-temporal, object-appearance, and external-predicate query conditions. The latter type of conditions are grouped together to form the maximal subqueries. Given a query, a *maximal subquery* is defined as a longest sequence of conditions that can be processed by Prolog without changing the semantics of the original query. Grouping the spatial conditions in a query into maximal subqueries minimizes the number of subqueries to be processed by our inference engine Prolog, thus reducing the interval processing time, and improving the overall performance of the system for spatio-temporal query processing. Our approach can be seen as reducing spatio-temporal video retrieval to meta-data queries on a rule-based facts-base; nonetheless, interval and similarity-based trajectory processing is carried out outside of the Prolog engine. Spatio-temporal query processing is carried out in three main stages: query recognition, query decomposition, and query execution.

In [9], we also proposed a novel video segmentation technique specifically for spatio-temporal modeling of video data, which is based on the spatio-temporal relations between salient video objects. In our approach, video clips are segmented into shots whenever the current set of relations between video objects changes, thereby helping us to determine parts of the video, where the spatial relationships do not change at all. Spatio-temporal relations are represented as Prolog facts partially stored in the knowledge-base, and those relations that are not stored explicitly can be derived by our inference engine Prolog, using the rules in the knowledge-base. The system has a comprehensive set of rules, which reduces the storage space needed for the spatio-temporal relations considerably, while keeping the query response time at interactive rates, as proven by our performance tests conducted using both synthetic and real video data [9]. Our rule-based spatio-temporal query processing strategy and query language take advantage of this segmentation technique to provide precise (fine-grained) answers to spatio-temporal video queries. Consequently, the smallest unit of retrieval is not a scene (a single camera shot), but a single frame in our VDBMS that we call *BilVideo*.

To the best of our knowledge, all VDBMSs proposed in the literature associate the spatio-temporal relations between video objects, and also object trajectories, with scenes defined as single camera shots. Hence, these systems are unable to return arbitrary segments of video clips in response to user queries that consist of spatio-temporal conditions. Nonetheless, users may not be interested in seeing an entire scene as a result of a query, if the query conditions are satisfied only in some parts of the scene. Moreover, since object trajectories are conventionally defined within the scenes, and thereby, do not span over the entire video as one entity, trajectory matching is restricted to the subtrajectories of objects that fall into scenes in the entire video. We believe that such a restriction limits the flexibility and power of a VDBMS for spatio-temporal query processing: users should be able to retrieve arbitrary video segments, if there is a matching for a given query trajectory with a part of an object trajectory, where the object trajectory spans over the entire video. To the best of our knowledge, only *BilVideo* provides this support due to its unique video segmentation technique that is based on the spatio-temporal relations between video objects.

The rest of the paper is organized as follows: Section 2 presents a discussion of some of the VDBMS and query languages proposed in the literature, and their comparison to *BilVideo* and its query language. Overall architecture of *BilVideo* and our rule-based approach to represent spatio-temporal relations between salient video objects are briefly mentioned in Section 3. Section 4 presents the proposed SQL-like textual query language, and demonstrates the capabilities of the language with some query examples on three different application areas; *soccer event analysis*, *bird migration tracking*, and *movie retrieval systems*. Section 5 provides a detailed discussion on the proposed rule-based spatio-temporal query processing strategy with some example queries. The results of our preliminary performance and scalability tests conducted on the knowledge-base of *BilVideo*, which are presented in detail in [9], are summarized in Section 6. We draw our conclusions and state possible future work areas in Section 7. Finally, the grammar of the proposed query language is given in Appendix A.

2 Related Work

In this section, we compare *BilVideo* and its query language with some other systems and query languages proposed in the literature. One point worth noting at the outset is that *BilVideo* query language, to the best of our knowledge, is unique in its support for retrieving any segment of a video clip, where the given query conditions are satisfied, regardless of how video data is semantically partitioned. None of the systems discussed here can return a subinterval of a scene as part of a query result because video features are associated with scenes defined to be the smallest semantic units of video data. In our approach, object trajectories, object-appearance relations, and spatio-temporal relations between video objects are represented as Prolog facts in a knowledge-base, and they are not explicitly related to semantic units of videos. Thus, *BilVideo* query language can return precise answers for spatio-temporal queries in terms of frame intervals. Moreover, our assessment for the directional relations between two video objects is also novel in that two overlapping objects may have directional relations defined for them with respect to one another, provided that center points of the objects' Minimum Bounding

Rectangles (MBRs) are different. It is because Allen's temporal interval algebra, [2], is not used as a basis for the directional relation definition in our approach: in order to determine which directional relation holds between two objects, center points of the objects' MBRs are used [9]. Furthermore, *BilVideo* query language provides three aggregate functions, *average*, *sum*, and *count*, which may be very attractive for some applications, such as sports statistical analysis systems, to collect statistical data on spatio-temporal events. Moreover, *BilVideo* query language provides full support for spatio-temporal querying of video data.

VideoSQL: VideoSQL is an SQL-like query language developed for OVID to retrieve video objects [31]. Before examining the conditions of a query for each video object, target video objects are evaluated according to the interval inclusion inheritance mechanism. A VideoSQL query consists of the basic *select*, *from*, and *where* clauses. Conditions may contain attribute/value pairs and comparison operators. Video numbers may also be used in specifying conditions. In addition, VideoSQL has a facility to merge the video objects retrieved by multiple queries. Nevertheless, the language does not contain any expression to specify spatial and temporal conditions on video objects. Thus, VideoSQL does not support spatio-temporal queries, which is a major weakness of the language.

MOQL and MTQL: In [26], multimedia extensions to the Object Query Language (OQL) and TIGUKAT Query Language (TQL) are proposed. The extended languages are called Multimedia Object Query Language (MOQL) and Multimedia TIGUKAT Query Language (MTQL), respectively. The extensions made are spatial, temporal, and presentation features for multimedia data. MOQL has been used in the STARS system [23] as well as in an object-oriented SGML/HyTime compliant multimedia database system [32], both developed at the University of Alberta.

MOQL and MTQL support content-based spatial and temporal queries, as well as query presentation. Both languages include support for 3D-relation queries, as we call them, by *front*, *back*, and their combinations with other directional relations, such as *front_left*, *front_right*, etc. *BilVideo* query language has a different set of third-dimension (3D) relations, though. The 3D relations supported by the *BilVideo* query language are *infrontof*, *behind*, *strictlyinfrontof*, *strictlybehind*, *touchfrombehind*, *touchedfrombehind*, and *samelevel*. Definitions of these 3D relations are given in Section 4.2.2. The moving object model integrated in MOQL and MTQL, [22], is also different from our model. *BilVideo* query language does not support similarity-based retrieval on spatial conditions as MOQL and MTQL do. Nonetheless, it does allow users to specify separate weights for the directional and displacement components of the trajectory conditions in queries, which both languages lack.

AVIS: In [28], a unified framework for characterizing multimedia information systems is proposed. Some user queries may not be answered efficiently using these data structures; therefore, for each media-instance, some feature constraints are stored as a logic program. Nonetheless, temporal aspects and relations are not taken into account in the model. Moreover, complex queries involving aggregate operations as well as uncertainty in queries require further work to be done. In addition, although the framework incorporates some feature constraints as facts to extend its query range, it does not provide a complete deductive system as we do.

The authors extend their work defining feature-subfeature relationships in [27]. When a query cannot be answered, it is relaxed by substituting a subfeature for a feature. This relaxation technique provides some support for reasoning with uncertainty.

In [1], a prototype video information system, called Advanced Video Information System (AVIS), is introduced. The authors propose a special kind of segment tree, namely *frame segment tree*, and a set of arrays to represent objects, events, activities, and their associations. The proposed data model is based on the generic multimedia model described in [28]. Consequently, temporal queries on events are not addressed in AVIS.

In [15], an SQL-like video query language, based on the data model developed by Adalı et al. [1], is proposed. Thus, the language does not provide any support for temporal queries on events. Nor does it have any language construct for spatio-temporal querying of video clips since it was designed for semantic queries on video data. In *BilVideo* query model, temporal operators, such as *before*, *during*, etc., would also be used to specify order in time between events just as they are used for spatio-temporal queries.

VideoSTAR: VideoSTAR proposes a generic data model that makes it possible to share and reuse video data [14]. Thematic indexes and structural components might implicitly be related to one another since frame sequences may overlap, and may be reused. Therefore, considerable processing is needed to explicitly determine the relations, making the system complex. Moreover, the model does not support spatio-temporal relations between video objects.

CVQL: A content-based logic video query language, *CVQL*, is proposed in [20]. Users retrieve video data specifying some spatial and temporal relationships for salient objects. An elimination-based preprocessing for filtering unqualified videos, and a behavior-based approach for video function evaluation are also introduced. For video evaluation, an index structure, called *M-index*, is proposed. Using this index structure, frame sequences satisfying a query predicate can be efficiently retrieved. Nevertheless, topological relations between salient objects are not supported since an object is represented by a point in two-dimensional (2D) space. Consequently, the language does not allow users to specify topological and similarity-based object-trajectory queries.

3 BilVideo VDBMS

This section is intended only to provide a very brief overview of the *BilVideo* system architecture. Further information and details can be found in our earlier publication [9].

3.1 Overall System Architecture

Figure 1 illustrates the system architecture of *BilVideo*. In the heart of the system lies the query processor, which is responsible for processing and responding to user queries in a multi-user environment. The query processor communicates with a knowledge-base and an object-relational database. The knowledge-base stores fact-based meta data used for spatio-temporal queries, whereas semantic and histogram-based (color, shape, and texture) meta data is stored in the feature database maintained by the object-relational database. Raw video data and video data features are stored separately. Semantic meta data stored in the feature database is generated and updated by a Video-Annotator tool, and the facts-base is populated by a Fact-Extractor tool, both developed as Java applications [3, 8]. The Fact-Extractor tool also extracts the color and shape histograms of objects of interest in video keyframes to be stored in the feature database [37].

BilVideo can currently handle only spatio-temporal queries on video data, which is the focus of this paper; however, we are in the process of extending it to provide an integrated support for semantic and low-level (color, shape, and texture) queries, as well.

Figure 1

3.2 Knowledge-base Structure

In the knowledge-base, each fact[1] has a single frame number that is of a keyframe. This representation scheme allows our inference engine Prolog to process spatio-temporal queries faster and easier compared to using frame intervals for the facts. It is because the frame interval processing to form the final query results is carried out efficiently by some optimized code, written in C++, outside the Prolog environment. Therefore, the rules used for querying video data, which we call *query rules*, have frame-number variables associated. A second set of rules that we call *extraction rules* was also created to work with frame intervals so as to extract spatio-temporal relations from video data. Extracted spatio-temporal relations are then converted to be stored as facts with frame numbers of the keyframes in the knowledge-base, and these facts are used by the query rules for query processing in the system.

The rules in the knowledge-base significantly reduce the number of facts that need to be stored for spatio-temporal querying of video data. Our storage space savings was about 40% for some real video data we experimented on. Moreover, the system's response time for different types of spatio-temporal queries posed on the same data was at interactive rates. We provide a brief summary of our performance tests conducted on the knowledge-base of *BilVideo* in Section 6. Details on the knowledge-base structure of *BilVideo*, our fact-extraction (video segmentation) algorithm, types of the rules/facts used, their definitions, and the detailed discussion of our performance tests involving spatial relations can be found in [9].

[1] Except for *appear* and *object-trajectory* facts, which have frame intervals as a component instead of frame numbers because of storage space, ease of processing, and processing cost considerations.

4 BilVideo Query Language

Retrieval of video data by its spatio-temporal content is a very important and challenging task. Query languages designed for relational, object and object-relational databases do not provide sufficient support for spatio-temporal video retrieval; consequently, either a new language should be designed and implemented, or an existing language should be extended with the required functionality.

In this section, we present a new video query language that is similar to SQL in structure. The language can be used for spatio-temporal queries that contain any combination of directional, topological, 3D-relation, external-predicate, object-appearance, trajectory-projection, and similarity-based object-trajectory conditions.

4.1 Features of the Language

BilVideo query language has four basic statements for retrieving information:

`select` *video* `from all` [`where` *condition*];
`select` *video* `from` *videolist* `where` *condition*;
`select` *segment* `from` *range* `where` *condition*;
`select` *variable* `from` *range* `where` *condition*;

The target of a query is specified in the `select` clause. A query may return videos (*video*), or segments of videos (*segment*), or values of variables (*variable*) with/without segments of videos. Regardless of the target type specified, video identifiers for videos are always returned as part of the query answer. The aggregate functions (*sum*, *average*, and *count*), which operate on segments, may also be used in the `select` clause. Variables might be used for the object identifiers and trajectories. Moreover, if the target of a query is videos (*video*), users may also specify the maximum number of videos to be returned as a result of a query. If the keyword `random` is used, video fact-files to process are selected randomly in the system, thereby returning a random set of videos as a result. The range of a query is specified in the `from` clause, which may be either the entire video collection or a list of specific videos. The query conditions are given in the `where` clause. In *BilVideo* query language, the *condition* is defined recursively, and consequently, it may contain any combination of spatio-temporal query conditions.

Supported Operators: *BilVideo* query language supports a set of logical and temporal operators to be used in the query conditions. The logical operators are *and*, *or*, and *not*, while the temporal operators are *before*, *meets*, *overlaps*, *starts*, *during*, *finishes*, and their inverse operators.

The language also has a trajectory-projection operator, *project*, which can be used to extract subtrajectories of video objects on a given spatial condition. The condition is local to *project*, and it is optional. If it is not given, entire object trajectories rather than subtrajectories of objects are returned.

The language has two operators, "=" and "!=", to be used for assignment and comparison. The left argument of these operators should be a variable, whereas the right argument may be either a variable or a constant (atom). Operator "!=" is used for inequality comparison, whilst operator "=" may take on different semantics depending on its arguments. If one of the arguments of operator "=" is an unbound variable, it is treated as the assignment operator. Otherwise, it is considered as the equality-comparison operator. These semantics were adopted from the Prolog language.

Operators that perform interval processing are called *interval operators*. Hence, all temporal operators are interval operators. Logical operators are also considered as interval operators, when their arguments contain intervals.

In *BilVideo* query language, precedence values of the logical, assignment, and comparison operators follow their usual order. Logical operators assume the same precedence values when they are considered as interval operators, as well. Temporal operators are given a higher priority over logical operators, when determining the arguments of operators, and they are left associative as are logical operators.

BilVideo query language also provides a keyword, `repeat`, that can be used in conjunction with a temporal operator, such as *before*, *meets*, etc., or a trajectory condition. Video data may be queried by repetitive conditions in time using `repeat` with an optional repetition number given. If a repetition number is not given with `repeat`, then it is considered indefinite, thereby causing the processor to search for the largest intervals in a video, where the conditions given are satisfied at least once over time. The keyword `tgap` may be used for the temporal operators and a trajectory condition. However, it has rather different semantics for each type. For temporal operators, `tgap` is only meaningful when `repeat` is used, because it specifies the maximum time gap allowed between the pairs of intervals to be processed for `repeat`. Therefore, the language requires that `tgap` be used along with `repeat` for temporal operators. For a trajectory condition, it may be used to specify the maximum time gap allowed for consecutive object movements, as well as pairs of intervals to be processed for `repeat` if `repeat` is also given in the condition.

Aggregate Functions: *BilVideo* query language has three aggregate functions, *average*, *sum*, and *count*, which take a set of intervals (segments) as input. *Average* and *sum* return a time value in minutes, whilst *count* returns an integer for each video clip satisfying given conditions. *Average* is used to compute the average of the time durations of all intervals found for a video clip, whereas *sum* and *count* are used to calculate the total time duration for, and the total number of all such intervals, respectively. These aggregate functions might be very useful to collect statistical data for some applications, such as sports event analysis systems, motion tracking systems, etc.

External Predicates: *BilVideo* query language is generic, and is designed to be used for any application that requires spatio-temporal query processing capabilities. It has a condition type *external* defined for application-dependent predicates, which we call *external predicates*. This condition type is generic; consequently, a query may contain any application-dependent predicate in the `where` clause of the language with a name different from any predefined predicate and language construct, and with at least one argument that is either a variable or a constant (atom). External predicates are processed just like spatial predicates as part of the maximal subqueries. If an external predicate is to be used for querying video data, facts and/or rules related to the predicate should be added to the knowledge-base beforehand.

In our design, each video segment returned as an answer to a user query has an associated importance value ranging between 0 and 1, where 1 denotes an exact match. The results are ordered with respect to these importance values in descending order. Maximal subqueries return segments with importance value 1 because they are exact-match queries, whereas the importance values for the segments returned by similarity-based object-trajectory queries are the similarity values computed. Interval operators *not* and *or* return the complements and union of their input intervals, respectively. Interval operator *or* returns intervals without changing their importance values, whilst the importance value for the intervals returned by *not* is 1. The rest of the interval operators takes the average of the importance values of their input interval pairs for the computed intervals. Users may also specify a time period in a query to view only the parts of videos returned as an answer. The grammar of the *BilVideo* query language is given in Appendix A.

4.2 Basic Query Types

This section presents the basic query types that the *BilVideo* query language supports. These types of queries can be combined to construct complex spatio-temporal queries without any restriction, which makes the language very flexible and powerful in terms of expressiveness. In this section, we provide some examples for the object and similarity-based object-trajectory queries; examples of the other types used in combination are introduced later in Sections 4.3 and 5.5.

4.2.1 Object Queries This type of queries may be used to retrieve salient objects for each video queried that satisfies the conditions, along with segments if desired, where the objects appear. Some example queries of this type are given below:

Query 1: “Find all video segments from the database in which object o_1 appears.”

```
select segment
from all
where appear(o1);
```

In this query, `appear` predicate returns the frame intervals (segments) of each video in the database, where object o_1 appears. The segments returned are grouped by videos, and each group is sorted in the linear timeline based on the starting frames, where smaller segments appear before larger ones if the starting frames of the intervals are the same.

Query 2: “Find the objects that appear together with object o_1 in a given video clip, and also return such segments.” (Video identifier for the given video clip is assumed to be 1.)

```
select segment, X
from 1
where appear(o1, X) and X != o1;
```

4.2.2 Spatial Queries This type of queries may be used to query videos by spatial properties of objects defined with respect to each other. Supported spatial properties for objects can be grouped into mainly three categories: directional relations that describe order in 2D space, topological relations that describe neighborhood and incidence in 2D space, and 3D relations that describe object positions on the z-axis of three-dimensional space.

There are eight distinct topological relations: *disjoint*, *touch*, *inside*, *contains*, *overlap*, *covers*, *coveredby*, and *equal*. The fundamental directional relations are *north*, *south*, *east*, *west*, *northeast*, *northwest*, *southeast*, and *southwest*. Furthermore, our 3D relations consist of *infrontof*, *strictlyinfrontof*, *touchfrombehind*, *samelevel*, *behind*, *strictlybehind*, and *touchedfrombehind*.

Definitions of the topological and 3D relations are based on Allen's temporal interval algebra [2]. Table 1 presents the semantics of our 3D relations. We, however, do not provide in this paper the semantics for the topological relations since they are given in a number of papers in the literature (e.g. [11] and [33]). We also include the relations *left*, *right*, *below*, and *above* in the group of directional relations, and these relations are defined in terms of the fundamental directional relations. However, directional components of the object trajectories can only contain the fundamental directional relations in query specifications. Our definitions for the directional relations are given in [9].

Table 1

4.2.3 Similarity-Based Object-Trajectory Queries In our data model, for each moving object in a video clip, a trajectory fact is stored in the facts-base. A trajectory fact is modelled as tr(ν, φ, ψ, κ), where

ν: object identifier,

φ (list of directions): [φ_1, φ_2, ..., φ_n], where $\varphi_i \in$ F[2] (1≤i≤n),

ψ (list of displacements): [ψ_1, ψ_2, ..., ψ_n], where $\psi_i \in Z^+$ (1≤i≤n),

κ (list of intervals): [[s_1, e_1], ..., [s_n, e_n]], where s_i, $e_i \in$ N and $s_i \le e_i$ (1≤i≤n).

A trajectory query is modeled as

`tr`(α, λ) [`sthreshold` σ [`dirweight` β | `dspweight` η]][`tgap` γ]

or

`tr`(α, θ) [`sthreshold` σ] [`tgap` γ]

where

α: object identifier,

λ: trajectory list ([θ, χ])

θ: list of directions,

χ: list of displacements,

`sthreshold` (similarity threshold): 0< σ <1,

`dirweight` (directional weight): 0≤ β ≤1 and 1-$\beta = \eta$,

`dspweight` (displacement weight): 0≤ η ≤1 and 1-$\eta = \beta$,

`tgap`: time threshold, $\gamma \in$ N, for the gap between consecutive object movements.

[2] set of fundamental directional relations

In a trajectory query, variables may be used for α and λ, and the number of directions is equal to the number of displacements in λ just like in trajectory facts, because each element of a list is associated with an element of the other list that has the same index value. The list of directions specifies a path an object follows, whilst the displacement list associates each direction in this path with a displacement value. However, it is optional to specify a displacement list in a query in which case no weights are used in matching trajectories. Such queries are useful when displacements are not important to the user.

In a trajectory query, similarity and time threshold values are also optional. If a similarity threshold is not given, the query is considered as an exact-match query. A query without a `tgap` value implies a continuous motion without any stop between consecutive object movements. The time threshold value specified in a query is considered in seconds. A trajectory query may have either a directional or a displacement weight supplied because the other is computed from the one given. Moreover, for a weight to be specified, a similarity threshold value must also be provided. If a similarity value is supplied, and no weight is given, then the weights of the directional and displacement components are considered equal by default. The key idea in measuring the similarity between a pair of trajectories is to find the distance between the two, and this is achieved by computing the distances between the directional and displacement components of the trajectories when both lists are available. If a displacement list is not specified in a query, then trajectory similarity is measured by comparing the directional lists. Furthermore, when a weight value is 0, its corresponding list is not taken into account in computing the similarity between trajectories.

Directional Similarity:

Definition 1 *A directional region is an area between neighboring directional segments in the directional coordinate system depicted in Figure 2.*

Definition 2 *Let d_a and d_b be two directions in the directional coordinate system. The distance between d_a and d_b, denoted as D(d_a, d_b), is defined to be the minimum number of directional regions between d_a and d_b.*

Definition 3 *The directional normalization factor, ω, is defined to be the number of directional regions between two opposite directions in the directional coordinate system (w = 4).*

Figure 2

Let A and B be two directional lists each having n elements such that A = [A_1, A_2, ..., A_n] and B = [B_1, B_2, ..., B_n]. The directional similarity between A and B is specified as follows:

$$\varsigma(A, B) = 1 - \frac{1}{w}\sqrt{\frac{1}{n}\sum_{i=1}^{n} D(A_i, B_i)^2} \qquad (1)$$

Displacement Similarity:

Definition 4 *The displacement normalization factor of a displacement list A is defined to be the maximum displacement value in the list, and it is denoted by A_μ.*

Let A and B be two displacement lists each having n elements such that A = [A_1, A_2, ..., A_n] and B = [B_1, B_2, ..., B_n]. Furthermore, let us suppose that $D_{nr}(A_i, B_i)$ denotes the normalized distance between A_i and B_i for 1≤i≤n. Then, the displacement similarity between A and B is specified as follows:

$$\varsigma(A,B) = 1 - \sqrt{\frac{1}{n}\sum_{i=1}^{n} D_{nr}(A_i,B_i)^2} \quad ,where \quad D_{nr}(A_i,B_i) = \frac{B_\mu A_i - A_\mu B_i}{A_\mu B_\mu} \tag{2}$$

Trajectory Matching:

Similarity-based object-trajectory queries are processed by the trajectory processor, which takes such queries as input and returns a set of intervals each associated with an importance value (similarity value), along with some other data needed by the query processor for forming the final set of answers to user queries, such as variable bindings (values) if variables are used. Here, we formally discuss how similarity-based object-trajectory queries with no variables are processed by the trajectory processor. In doing so, it is assumed without loss of generality that trajectory queries contain both the directional and displacement lists. Moreover, we restrict our discussion to such cases as those where the time gaps between consecutive object movements in trajectory facts are equal to or below the time threshold given in a query. These assumptions are made just for the sake of simplicity, because our main goal here is to explain the theory that provides a novel framework for our similarity-based object-trajectory matching mechanism rather than presenting our query processing algorithm in detail.

Let Q and T be a similarity-based object-trajectory query and a trajectory fact for an object that is stored in the facts-base for a video clip, respectively, such that Q = `tr`(α, λ) `sthreshold` σ `dirweight` β and T = (ν, φ, ψ, κ), where λ = [θ, χ]. Let us assume that there is no variable used in Q or all variables are bound, $\alpha = \nu$, $\|\varphi\|$ = n, and $\|\theta\|$ = m. Let us also assume that there is no gap between any consecutive pairs of intervals in κ such that $\kappa_{e_i} = \kappa_{s_{i+1}}$ (1≤i<m).

Case 1 (n=m): The similarity between the two trajectories Q_t = (θ, χ) and T_t = (φ, ψ) is computed as follows:

$$\varsigma(Q_t, T_t) = \beta\, \varsigma(\theta,\varphi) + \eta\, \varsigma(\chi,\psi), \quad where \quad \beta = 1 - \eta \tag{3}$$

In this case, the trajectory processor returns only one interval, ξ = [κ_{s_1}, κ_{e_n}], iff ς(Q_t, T_t)≥σ. Otherwise (ς(Q_t, T_t)<σ), the answer set is empty because there is no similarity between Q_t and T_t with a given threshold σ.

Case 2 (n>m): In this case, the trajectory processor returns a set of intervals ϕ such that

$$\phi = \{[s_i, e_i] | 1 \le i \le n-m+1 \ \wedge \ s_i = \kappa_{s_i} \ \wedge \ e_i = \kappa_{e_{i+m-1}} \ \wedge \ \varsigma(Q_t, T_{t_{[i,i+m-1]}}) \ge \sigma\} \tag{4}$$

where

$$T_{t_{[i,i+m-1]}} = ([\varphi_i, ...,\varphi_{i+m-1}], [\psi_i, ...,\psi_{i+m-1}]) \tag{5}$$

If there is no match found for any T_{t_i} for 1≤i≤n-m+1, where T_{t_i} = $T_{t_{[i,i+m-1]}}$, then the answer set is empty.

Case 3 (n<m): As in Case 1, the trajectory processor returns only one interval, $\xi = [\kappa_{s_1}, \kappa_{e_n}]$,

$$\text{iff} \quad \exists\, \varsigma(Q_{t_{[i,i+n-1]}}, T_t) \geq \frac{m}{n}\,\sigma \quad \text{for} \quad 1 \leq i \leq m-n+1,$$

where

$$Q_{t_{[i,i+n-1]}} = ([\theta_i, \ldots, \theta_{i+n-1}], [\chi_i, \ldots, \chi_{i+n-1}])$$

The importance value (similarity value) associated and returned with ξ is

$$\varsigma = \frac{n}{m}\, MAX\ \{\varsigma | \varsigma(Q_{t_{[i,i+n-1]}}, T_t)\ (1 \leq i \leq m-n+1)\}$$

If there is no match found, the answer set is empty because there is no similarity between Q_t and T_t with a given threshold σ.

Following is an example similarity-based object-trajectory query specification in *BilVideo* query language. In this example query, we are interested in retrieving the segments of a video, whose video identifier is specified as 1, where object o_1 follows a similar path to the query trajectory with no time gap value given (continuous movement). For the sake of simplicity, let us assume that the trajectory of object o_1 stored in the knowledge-base for the video queried is

```
tr(o1, [east, north, east, north, south], [10, 20,
10, 30, 15], [[1, 100], [100, 150], [150, 200], [200,
250], [250, 300]]).
```

```
select segment
from 1
where tr(o1, [[east, north, east, northwest],
    [10, 20, 15, 25]]) sthreshold 0.6 dirweight 0.7;
```

Hence, for this query example, $\alpha = \nu = o_1$, $\|\varphi\|$ = n = 5, $\|\theta\|$ = m = 4, σ = 0.6, and β = 0.7 (η = 1 - β = 0.3). Moreover, T = (ν, φ, ψ, κ) and Q = `tr`(α, λ) `sthreshold` σ `dirweight` β, where

φ = [east, north, east, north, south],
ψ = [10, 20, 10, 30, 15],
κ = [[1, 100], [100, 150], [150, 200], [200, 250], [250, 300]],
$\lambda = [\theta, \chi]$
θ = [east, north, east, northwest],
χ = [10, 20, 15, 25].

Since n $>$ m, this query falls into Case 2. Thus, from Equation 5,
$T_{t_{[1,4]}}$ = [[east, north, east, north], [10, 20, 10, 30]] and
$T_{t_{[2,5]}}$ = [[north, east, north, south], [20, 10, 30, 15]].

According to Equation 4, $\varsigma(Q_t, T_{t_{[1,4]}})$ and $\varsigma(Q_t, T_{t_{[2,5]}})$ are computed using the formula given in Equation 3. Therefore,

$\varsigma(Q_t, T_{t_{[1,4]}})$ = 0.7 $\varsigma(\theta, \varphi_{T_{t_{[1,4]}}})$ + 0.3 $\varsigma(\chi, \psi_{T_{t_{[1,4]}}})$
$\varsigma(Q_t, T_{t_{[2,5]}})$ = 0.7 $\varsigma(\theta, \varphi_{T_{t_{[2,5]}}})$ + 0.3 $\varsigma(\chi, \psi_{T_{t_{[2,5]}}})$,

where

$\varphi_{T_{t_{[1,4]}}}$ = [east, north, east, north],
$\varphi_{T_{t_{[2,5]}}}$ = [north, east, north, south],
$\psi_{T_{t_{[1,4]}}}$ = [10, 20, 10, 30],
$\psi_{T_{t_{[2,5]}}}$ = [20, 10, 30, 15].

$\varsigma(\theta, \varphi_{T_{t_{[1,4]}}})$ and $\varsigma(\theta, \varphi_{T_{t_{[2,5]}}})$ are computed using Equation 1, while $\varsigma(\chi, \psi_{T_{t_{[1,4]}}})$ and $\varsigma(\chi, \psi_{T_{t_{[2,5]}}})$ are computed using Equation 2. After the computations, $\varsigma(\theta, \varphi_{T_{t_{[1,4]}}})$ = 0.875, $\varsigma(\theta, \varphi_{T_{t_{[2,5]}}})$ = 0.427, $\varsigma(\chi, \psi_{T_{t_{[1,4]}}})$ = 0.949, and $\varsigma(\chi, \psi_{T_{t_{[2,5]}}})$ = 0.156. Thereby, $\varsigma(Q_t, T_{t_{[1,4]}})$ = 0.897 and $\varsigma(Q_t, T_{t_{[2,5]}})$ = 0.346.

Since $\varsigma(Q_t, T_{t_{[1,4]}}) > 0.6$, but $\varsigma(Q_t, T_{t_{[2,5]}}) < 0.6$, the only interval, $[s, e]$, returned as a result of this query is $[\kappa_{s_1}, \kappa_{e_4}]$, where $\kappa_{s_1} = 1$ and $\kappa_{e_4} = 250$. Hence, $\phi = \{[1, 250]\}$.

Projection Operator:

BilVideo query language provides a trajectory-projection operator, *project*(α [, β]), to extract subtrajectories from the trajectory facts, where α is an object identifier for which a variable might be used, and β is an optional condition. If a condition is not given, then the operator returns the entire trajectory that an object follows in a video clip. Otherwise, subtrajectories of an object, where the given condition is satisfied, are returned. Hence, the output of *project* is a set $\vartheta = \{\lambda \mid \lambda = [\theta, \chi]\}$, where λ is a trajectory, and θ and χ are the directional and displacement components of λ, respectively. The condition, if it is given, is local to *project*, and it is of type `<spatial-condition>` as specified in Appendix A.

4.2.4 Temporal Queries This type of queries is used to specify the order of occurrence of conditions in time. Conditions may be of any type, but temporal operators process their arguments only if they contain intervals. *BilVideo* query language implements all temporal relations, defined by Allen's temporal interval algebra, as temporal operators, except for *equal*: our interval operator *and* yields the same functionality as that of *equal* because its definition, given in Section 5.4, is the same as that of *equal* for interval processing. Supported temporal operators, which are used as interval operators in *BilVideo* query language, are *before*, *meets*, *overlaps*, *starts*, *during*, *finishes*, and their inverse operators. A user query may contain repeating temporal conditions specified by `repeat` with an optional repetition number given. If `tgap` is not provided with `repeat`, then its default value for the temporal operators (equivalent to one frame when converted) is assumed. Definitions of the temporal relations can be found in [2].

4.2.5 Aggregate Queries This type of queries may be used to retrieve statistical data about objects and events in video data. The *BilVideo* query language supports three aggregate functions, *average*, *sum*, and *count*, as explained in Section 4.1.

4.3 Example Applications

To demonstrate the capabilities of the *BilVideo* query language, three application areas, *soccer event analysis*, *bird migration tracking*, and *movie retrieval systems*, have been selected. However, it should be noted that the *BilVideo* system architecture and *BilVideo* query language provide a generic framework to be used for any application that requires spatio-temporal query processing capabilities.

4.3.1 Soccer Event Analysis System A soccer event analysis system may be used to collect statistical data on events that occur during a soccer game, such as finding the number of goals, offsides and passes, average ball control time for players, etc., as well as to retrieve video segments, where such events take place. *BilVideo* query language can be used to answer such queries, provided that some necessary facts, such as players and goalkeepers for the teams, as well as some predicates, such as *player* to find the players of a certain team, are added to the knowledge-base. This section provides some query examples based on an imaginary soccer game fragment between England's two teams *Liverpool* and *Manchester United*. The video identifier of this fragment is assumed to be 1.

Query 1: "Find the number of direct shots to the goalkeeper of *Liverpool* by each player of *Manchester United* in a given video clip, and return such video segments."

This query can be specified in *BilVideo* query language as follows:

```
select count(segment), segment, X
from 1
where goalkeeper(X, liverpool) and
  player(Y, manchester) and touch(Y, ball)
  meets not(touch(Z, ball)) meets touch(X, ball);
```

In this query, the external predicates are *goalkeeper* and *player*. For each player of *Manchester United* found in the specified video clip, the number of direct shots to the goalkeeper of *Liverpool* by the player, along with the player's name and video segments found, is returned provided that such segments exist. In *BilVideo* system architecture, semantic meta data is stored in an object-relational database. Hence, video identifiers can be retrieved from this database querying it with some descriptional data.

Query 2: "Find the average ball control (play) time for each player of *Manchester United* in a given video clip."

This query can be specified in *BilVideo* query language as follows:

```
select average(segment), X
from 1
where player(X, manchester) and touch(X, ball);
```

In answering this query, it is assumed that when a player touches the ball, it is in his control. Then, the ball control time for a player is computed with respect to the time interval during which he is in touch with the ball. Hence, the average ball control time for a player is simply the sum of all time intervals where the player is in touch with the ball divided by the number of these time intervals. This value is computed by the aggregate function *average*.

Query 3: "Find the number of goals of *Liverpool* scored against *Manchester United* in a given video clip."

This query can be specified in *BilVideo* query language as follows:

```
select count(segment)
from 1
where samelevel(ball, net) and
  overlap(ball, net);
```

In this query, 3D relation *samelevel* ensures that an event, which is not a goal because the ball does not go into the net, but rather passes somewhere near the net, is not considered as a goal. The ball may overlap with the net in 2D space while it is behind or in front of the net on the z-axis of three-dimensional space. Hence, by using the 3D relation *samelevel*, such false events are discarded.

4.3.2 Bird Migration Tracking System A bird migration tracking system is used to determine the migration paths of birds over a set of regions in certain times. In [30], an animal movement querying system is discussed, and we have chosen a specific application of such a system to show how the *BilVideo* query language might be used to answer spatio-temporal, especially object-trajectory, queries on the migration paths of birds.

Query 1: "Find the migration paths of *bird* o_1 over *region* r_1 in a given video clip."
This query can be specified in *BilVideo* query language as follows:

```
select X
from 2
where X = project(o1, inside(o1, r1));
```

In this query, X is a variable used for the trajectory of *bird* o_1 over *region* r_1. The video identifier of the video clip, where the migration of *bird* o_1 is recorded, is assumed to be 2. This query returns the paths *bird* o_1 follows when it is inside *region* r_1.

Query 2: "How long does *bird* o_1 appear inside *region* r_1 in a given video clip?"
This query can be specified in *BilVideo* query language as follows:

```
select sum(segment)
from 2
where inside(o1, r1);
```

The result of this query is a time value, which is computed by the aggregate function *sum* adding up the time intervals during which *bird* o_1 is inside *region* r_1.

Query 3: "Find the video segments where *bird* o_1 enters *region* r_1 from west, and leaves from north, in a given video clip."
This query can be specified in *BilVideo* query language as follows:

```
select segment
from 2
where (touch(o1, r1) and west(o1, r1)) meets
  overlap(o1, r1) meets coveredby(o1, r1) meets
  inside(o1, r1) meets
  coveredby(o1, r1) meets overlap(o1, r1) meets
  (touch(o1, r1) and north(o1, r1));
```

Query 4: "Find the names of birds following a similar path to that of *bird* o_1 over *region* r_1 with a similarity threshold value 0.9 in a given video clip, and return such segments."

This query can be specified in *BilVideo* query language as follows:

```
select segment, X
from 2
where Y = project(o1, inside(o1, r1)) and
   inside(X, r1) and X != o1 and
   tr(X, Y) sthreshold 0.9;
```

Here, X and Y are variables representing the bird names and subtrajectories of *bird* o_1 over *region* r_1, respectively. Projected subtrajectories of *bird* o_1, where the given condition is to be inside *region* r_1, are used to find similar subtrajectories of other birds over the same region.

4.3.3 Movie Retrieval System A movie retrieval system contains movies and series from different categories, such as cartoon, comedy, drama, fiction, horror, etc. Such a system may be used to retrieve videos or segments from a collection of movies with some spatio-temporal, semantic, and low-level conditions given. In this section, a specific episode of *Smurfs* (a cartoon series), titled as *Bigmouth's Friend*, is used for the two spatio-temporal query examples given. The video identifier of this episode is assumed to be 3.

Query 1: "Find the segments from *Bigmouth's Friend* where *Bigmouth* is below *RobotSmurf*, while *RobotSmurf* starts moving towards west, and then goes to east, repeating this as many times as it happens in the video clip."

```
select segment
from 3
where below(bigmouth, robotsmurf) and
   (tr(bigmouth, [west, east])) repeat;
```

Query 2: "Find the segments from *Bigmouth's Friend* where *robotsmurf* and *bigmouth* are disjoint, and *robotsmurf* is to the right of *bigmouth*, while there is no other object of interest that appears."

```
select segment
from 3
where disjoint(robotsmurf, bigmouth) and
   right(robotsmurf, bigmouth) and
   appear_alone(robotsmurf, bigmouth);
```

In this query, *appear_alone* is an external predicate defined in the knowledge-base as follows:

```
appear_alone(X, Y, F) :- keyframes(L1),
   member(F, L1), findall(W, p_appear(W, F), L2),
   length(L2, 2),
   forall(member(Z, L2), (Z = X; Z = Y)).
```

5 Spatio-temporal Query Processing

This section explains our rule-based spatio-temporal query processing strategy in detail. The query processing is carried out in three phases, namely, *query recognition*, *query decomposition*, and *query execution*. These phases are depicted in Figure 3, and they are explained in Sections 5.1 through 5.3. The interval processing is performed in the query execution phase, and it is discussed in Section 5.4 through some case studies.

In *BilVideo* query model, the conditions are evaluated in a single timeline. For each internal node in the query tree, the child nodes are evaluated first, and the results obtained from the child nodes are propagated to the parent node for interval processing, going up in the query tree until the final query results are obtained.

Figure 3

5.1 Query Recognition

The lexical analyzer and parser for the *BilVideo* query language were implemented using Flex and Bison that work under a Linux operating system [10, 34], which are the GNU versions of the original Lex&Yacc [17, 21] compiler-compiler generator tools. The lexical analyzer partitions a query into tokens, which are passed to the parser with possible values for further processing. The parser assigns structure to the resulting pieces and creates a parse tree to be used as a starting point for query processing. This phase is called *query recognition phase*.

5.2 Query Decomposition

The parse tree generated after the query recognition phase is traversed in a second phase, which we call the *query decomposition phase*, to construct a query tree. The query tree is constructed from the parse tree decomposing a query into three basic types of subqueries: *plain Prolog subqueries* or *maximal subqueries* that can be directly sent to the inference engine Prolog, *trajectory-projection subqueries* that are handled by the trajectory projector, and *similarity-based object-trajectory subqueries* that are processed by the trajectory processor. Temporal queries are handled by the interval-operator functions such as *before*, *during*, etc. Arguments of the interval operators are handled separately because they should be processed before the interval operators are applied. Since a user may give any combination of conditions in any order while specifying a query, a query is decomposed in such a way that a minimum number of subqueries are formed. This is achieved by grouping the Prolog-type predicates into maximal subqueries without changing the semantic meaning of the original query.

5.3 Query Execution

The input for the *query execution phase* is a query tree. In this phase, the query tree is traversed in postorder, executing each subquery separately and performing interval processing in internal nodes so as to obtain the final set of results. Since it would be inefficient and very difficult, if not impossible, to fully handle spatio-temporal queries by Prolog alone, the *query execution phase* is mainly carried out by some efficient C++ code. Thus, Prolog is utilized only to obtain intermediate answers to user queries from the facts-base. The intermediate query results returned by Prolog are further processed, and the final answers to user queries are formed after the interval processing. Figure 4 illustrates the *query execution phase*. Figure 4

The *BilVideo* query language is designed to return variable values, when requested explicitly, as part of the query result, as well. Therefore, the language not only supports video/segment queries but also variable-value retrieval for the parts of videos satisfying given query conditions, utilizing a knowledge-base. Variables may be used for the object identifiers and trajectories.

One of the main challenges in query execution is to handle such user queries where the scope of a variable used extends to several subqueries after the query is decomposed. It is a challenging task because subqueries are processed separately, accumulating and processing the intermediate results along the way to form the final set of answers. Hence, the values assigned to variables for a subquery are retrieved and used for the same variables of other subqueries within the scope of these variables. Therefore, it is necessary to keep track of the scope of each variable for a query. This scope information is stored in a hash table generated for the variables. Dealing with variables makes the query processing much harder, but it also empowers the query capabilities of the system and yields much richer semantics for user queries.

5.4 Interval Processing

In *BilVideo* query model, intervals are categorized into two types: *non-atomic* and *atomic* intervals. If a condition holds for every frame of a part of a video clip, then the interval representing an answer for this condition is considered as a non-atomic interval. Non-atomicity implies that for every frame within an interval in question does the condition hold. Hence, the condition holds for any subinterval of a non-atomic interval, as well. This implication is not correct for atomic intervals, though. The reason is that the condition associated with an atomic interval does not hold for all its subintervals. Consequently, an atomic interval cannot be broken into its subintervals for query processing. On the other hand, subintervals of an atomic interval are populated for query processing, provided that conditions are satisfied in their range. In other words, the query processor generates all possible atomic intervals for which the given conditions are satisfied. This interval population is necessary since atomic intervals cannot be broken into subintervals, and all such intervals, where the conditions hold, should be generated for query processing. The intervals returned by the *plain Prolog subqueries* (*maximal subqueries*) that contain directional, topological, object-appearance, 3D-relation, and external-predicate conditions are non-atomic, whereas those obtained by applying the temporal operators to the interval sets, as well as those returned by the similarity-based object-trajectory subqueries are atomic intervals. Since the logical operators *AND*, *OR* and *NOT* are considered as interval operators when their arguments contain intervals to process, they also work on intervals. The operators *AND* and *OR* may return atomic and/or non-atomic intervals depending on the types of their input intervals. The operator *AND* takes the intersection of its input intervals, while the operator *OR* performs a union operation on its input intervals. The unary operator *NOT* returns the complement of its input interval set with respect to the video clip being queried, and the intervals in the result set are of type non-atomic, regardless of the types of the input intervals. Semantics of the interval intersection and union operations are given in Tables 2 and 3, respectively.

Table 2
Table 3

The rationale behind classifying the video frame intervals into two categories as atomic and non-atomic may be best described with the following query example: “*Return the video segments in the database, where object A is to the west of object B and object A follows a similar trajectory to the one specified in the query with respect to the similarity threshold given*”. Let us assume that the intervals [10, 200] and [10, 50] are returned as part of the answer set for a video for the trajectory and spatial (directional) conditions of this query, respectively. Here, the first interval is of type atomic because the trajectory of object A is only valid within the interval [10, 200], and therefore, trajectory similarity computation is not performed for any of its subintervals. However, the second interval is non-atomic since the directional condition given is satisfied for each frame in this interval. When these two intervals are processed to form the final result by the *AND* operator, no interval is returned as an answer because there is no such an interval, where both conditions are satisfied together. If there were no classification of intervals, and all intervals were to be breakable into subintervals, then the final result set would include the interval [10, 50]. However, as obvious, the two conditions cannot hold together in this interval due to the fact that the trajectory of object A spans over the interval [10, 200]. As another case, let us suppose that the intervals [10, 200] and [10, 50] are returned as part of the answer set for the spatial (directional) and trajectory conditions of this query, respectively, and the intervals were to be unbreakable to sub-intervals. Then, the result set would be empty for these two intervals. This is not correct since there is an interval, [10, 50], where both conditions hold. These two cases clearly show that intervals must be classified into two groups as atomic and non-atomic for query processing. Following is a discussion with another example query that has a temporal predicate provided to make all these concepts much clearer.

Let us suppose that a user wants to find the parts of a video clip satisfying the following query:

Query: (A before B) and west(x, y), where A and B are Prolog subqueries, and x and y are atoms (constants).

The interval operator “before” returns a set of atomic intervals, where first A is true and B is false, and then, A is false and B is true in time. If A and B are true in the intervals [4, 10] and [20, 30], respectively, and if these two intervals are both non-atomic, then the result set will consist of [10, 20], [10, 21], [9, 20], [10, 22], [9, 21], ..., [4, 30]. Now, let us discuss two different scenarios:

Case 1: west(x, y) holds for [9, 25]. This interval is non-atomic because west(x, y) returns non-atomic intervals. If the operator "before" returned only the atomic interval [4, 30] as the answer for "A before B", then the answer set to the entire query would be empty. However, the user is interested in finding the parts of a video clip, where "(A before B) and west(x, y)" is true. The intervals [10, 20], [10, 21], ..., [4, 29] also satisfy "A before B"; however, they would not be included in the answer set for "before". This is wrong! All these intervals must be a part of the answer set for "before" as well. If they are included, then the answer to the entire query will be [9, 25] because [9, 25] (atomic) and [9, 25] (non-atomic) $=>$ [9, 25] (atomic). Nonetheless, make a note of that such intervals as [10, 19], [11, 25], etc. are not included in the answer set of "A before B" since they do not satisfy the condition "A before B".

Case 2: west(x, y) holds for [11, 25]. Let us suppose that "before" returned non-atomic intervals rather than atomic intervals, and that the answer for "A before B" were [4, 30]. Then, the answer to the entire query would be [11, 25] for [4, 30] (non-atomic) and [11, 25] (non-atomic) $=>$ [11, 25] (non-atomic). Nevertheless, this is wrong due to the fact that "A before B" is not satisfied within this interval. Hence, "before" should return atomic intervals so that such incorrect results are not produced.

These two cases clearly show that the temporal operators should return atomic intervals, and that the results should also include the subintervals of each largest interval that satisfy the given conditions, rather than consisting only of the set of largest intervals. It also demonstrates why such a classification of the intervals as atomic and non-atomic is necessary for interval processing.

5.5 Query Examples

In this section, three example spatio-temporal queries are given to demonstrate how the query processor decomposes a query into subqueries. Intermediate results obtained from these subqueries are integrated step by step to form the final answer set.

Query 1:
```
select segment, X, Y
from all
where west(X, Y) and west(Y, o1) and west(o1, o2)
  and tr(o2, [[west, east], [24, 40]])
  sthreshold 0.4 dspweight 0.3 and
  disjoint(X, Y) before touch(X, Y) and
  disjoint(Y, o1);
```

This example query is decomposed into following subqueries:

Subquery 1:
```
tr(o2, [[west, east], [24, 40]])
  sthreshold 0.4 dspweight 0.3
```

Subquery 2:
```
disjoint(X, Y)
```

Subquery 3: `touch(X, Y)`
Subquery 4: `west(X, Y) and west(Y,` o_1`) and west(`o_1`,` o_2`),`
`and disjoint(Y,` o_1`)`

The directional conditions `west(X, Y)`, `west(Y,` o_1`)`, and `west(`o_1`,` o_2`)` can be grouped together with the topological condition `disjoint(Y,` o_1`)` using the *and* operator without changing the semantics of the original query, as shown in the example decomposition. It should be noted here that if the topological condition `disjoint(Y,` o_1`)` were connected in the query with the operator *or* or a temporal operator, then such a grouping would not be possible. In this example, subqueries 2 through 4 are the maximal subqueries. Subqueries 2 and 3 are linked to each other by the temporal operator *before*. The rest of the internal nodes in the query tree contains the operator *and*. Figure 5 depicts the query tree constructed for this example query.

Figure 5

Query 2:
```
select segment, Y
from all
where west(X, Y) and west(Y, o1) and
  tr(o2, [[west, east], [24, 40]])
  sthreshold 0.4 dirweight 0.4 and
  disjoint(Y, o1);
```

Query 2 is decomposed into following subqueries:

Subquery 1: `tr(`o_2`, [[west, east], [24, 40]])`
`sthreshold 0.4 dirweight 0.4`
Subquery 2: `west(X, Y) and west(Y,` o_1`) and disjoint(Y,` o_1`)`

To answer Query 1, the query processor computes each subquery traversing the query tree in postorder performing interval processing at each internal node and taking into account the scope of each variable encountered. Here, the scope of object variables X and Y is subqueries 2, 3 and 4. Hence, for each value-pair of variables X and Y, a set of intervals is computed in subquery 2. Another reason for computing a set of intervals for each value-pair is that the values obtained for variables X and Y are also returned in pairs, along with the video segments satisfying the query conditions, as part of the query results. Hence, even if the scope of these variables were to be only subquery 2, the same type of interval processing and care must be provided. Nonetheless, if an object variable is bound by only one subquery, and its values are not to be returned as part of the query result as in the case of object variable X in Query 2, then it is possible to combine consecutive intervals, where the variable takes different values, while the rest of the conditions is satisfied for the same set of value-sequences for the rest of the variables. Query 3 better explains this concept of interval processing and variable value computation:

Query 3: "Return video segments in the database where object o_1 is first disjoint from object o_2 and then touches it repeating this event 3 times while it is inside another object."

```
select segment
from all
where inside(o1, X) and (disjoint(o1, o2) meets
  touch(o1, o2)) repeat 3;
```

In this query, we do not care which object object o_1 is inside, but we are only interested in the video segments where object o_1 is first disjoint from object o_2, and then touches it, repeating this event 3 times, while it is inside another object. Thus, the consecutive intervals for different objects that contain object o_1 may be combined, provided that the given conditions are satisfied.

6 Discussion on Performance

The running time of our algorithms for processing spatio-temporal queries depend on many parameters that are very hard to formulate nicely. The reason of this is mostly due to the possible existence of variables in user queries. As explained in Section 5.3, allowing variables in a user query makes the query processing much harder; nonetheless, it also empowers the query capabilities of the system, and results in much richer semantics for user queries. In *BilVideo*, when a variable is unified (bound to some values previously computed within its scope), these values are transferred and used for a condition (containing that variable) that comes next within the variable's scope. The query processor uses these values, instead of finding all the values of the variable that satisfy the condition regardless of the previous condition(s) and eliminating those that cannot be included in the result set because they do not satisfy the previous condition(s) in the variable's scope. This speeds up the query processing with unified variables, even though there is also an overhead for transferring the previously computed values for the variables. The reason is that the query domains of the variables for the next condition are narrowed down (restricted to the previously computed values for the unified variables). Since a condition may contain any number of variables, and some of these variables might have been unified previously in executing the query, the query processor has to take into account for that condition a set of variable-value lists. For this reason, it is very hard to formalize the running time behaviors of our spatio-temporal query processing algorithms as they depend on many parameters, such as the number of variables used, their scope within the entire query, the query domains of the variables for each condition, the overhead involved in transferring the variable-value lists, etc., in addition to the database size. Therefore, we instead provide a brief summary of our preliminary performance results which are presented in detail in [9].

These performance results show that the system is scalable for spatio-temporal queries in terms of the number of salient objects per frame and the total number of frames in a video clip. The results also demonstrate the space savings achieved due to our rule-based approach. For the time efficiency tests, queries were given to the knowledge-base as Prolog predicates. For the scalability and space savings, program-generated synthetic video data was used. These tests constitute the first part of our overall tests. In the second part, the performance of the knowledge-base

was tested on some real video fragments with the consideration of space and time efficiency criteria to show its applicability in real-life applications. Real video data was extracted from *jornal.mpg*[3] and a *Smurfs* cartoon episode named *Bigmouth*'s
Table 4 *Friend*. Table 4 presents some information about these video fragments.

For the space efficiency tests with the program-generated synthetic data, the number of objects per frame was selected as 8, 15 and 25, while the total number of frames was fixed to 100. To show the system's scalability in terms of the number of objects per frame, the total number of frames was chosen to be 100, and the number of objects per frame was changed from 4 to 25. For the scalability test with respect to the total number of frames, the number of objects was fixed to 8, whilst the total number of frames was varied from 100 to 1000.

In the tests conducted with the program-generated video data, there was a 19.59% savings from the space for the sample data of 8 objects and 1000 frames. The space savings was 31.47% for the sample video of 15 objects and 1000 frames, while it was 40.42% for 25 objects and 1000 frames. With the real data, for the first video fragment *jornal.mpg*, our rule-based approach achieved a savings of 37.5% from the space. The space savings for the other fragment, *smurfs.avi*, was 40%.

The space savings obtained from the program-generated video data is relatively low compared to that obtained from the real video fragments. We believe that the reason behind such a behavior is due to the random simulation of the motion of objects in our synthetic test data: while creating the synthetic video data, the motion pattern of objects was simulated randomly changing the objects' MBR coordinates by choosing only one object to move at each frame. Nevertheless, objects generally move slower in real video, causing the set of spatial relations to change over a longer period of frames. During the tests with the synthetic video data, it is also observed that space savings do not change when the number of frames is increased as the number of objects of interest per frame is fixed. The test results obtained for the synthetic data comply with those obtained for the real video. Some differences seen in the results stem from the fact that synthetic data was produced by a program, thereby not being able to perfectly simulate a real-life scenario.

The time efficiency tests performed on the program-generated synthetic data show that the system is scalable in terms of the number of objects and the number of frames, when either of these numbers is increased while the other is fixed. Moreover, the knowledge-base of the system has a reasonable response time as the results of the time efficiency tests on the real video data show. Therefore, we can claim that the knowledge-base of *BilVideo* is reasonably fast enough for answering spatio-temporal queries.

7 Conclusions and Future Work

We proposed an SQL-like textual query language for spatio-temporal queries on video data, and demonstrated the capabilities of the language through some example queries given on different application areas. Our novel rule-based spatio-

[3] from MPEG-7 Test Data set CD-14, Port. news

temporal query processing strategy has also been explained with some query examples.

The *BilVideo* query language is designed to be used for any application, which needs spatio-temporal query processing facilities. It is extensible in that any application-dependent predicate with a different name from those of predefined predicates and constructs of the language, and with at least one argument, can be used in user queries. For that, it suffices to add some necessary facts and/or rules to the knowledge-base a priori. Hence, the language provides query support through *external predicates* for application-dependent data.

The *BilVideo* query language currently supports a broad range of spatio-temporal queries. However, the *BilVideo* system architecture is designed to handle semantic (keyword, event/activity, and category-based) and low-level (color, shape, and texture) video queries, as well. We completed our work on semantic video modeling, which has been reported in [3]. As for the low-level queries, our *Fact-Extractor* tool also extracts color and shape histograms of the salient objects in video keyframes [37], and it is currently being extended to extract texture information from the video keyframes, as well. We are currently working on integrating the support for semantic and low-level video queries into *BilVideo*, by extending its query processor and query language, without affecting the way the spatio-temporal query conditions are specified in the query language, and processed by the query processor. Furthermore, we also completed our initial work on the optimization of the spatio-temporal video queries [40]. In an ideal environment, the *BilVideo* query language will establish the basis for a Web-based visual query interface and serve as an embedded language for users. Hence, we developed a Web-based visual query interface for specifying spatio-temporal video queries, visually, over the Internet [36]. We are currently working on enhancing the interface for the semantic and low-level video query specification support. We will integrate the Web-based visual query interface to *BilVideo*, and make it available on the Internet in future, when we complete our work on semantic and low-level video queries.

References

1. S. Adalı, K.S. Candan, S. Chen, K. Erol, and V.S. Subrahmanian. Advanced video information systems: Data structures and query processing. *ACM Multimedia Systems*, 4:172–186, 1996.
2. J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of ACM*, 26(11):832–843, 1983.
3. U. Arslan, M. E. Dönderler, E. Saykol, Ö. Ulusoy, and U. Güdükbay. A semi-automatic semantic annotation tool for video databases. In *Proc. of Workshop on Multimedia Semantics (SOFSEM'2002)*, pages 1–10, Czech Republic, 2002. Available online at http://www.cs.bilkent.edu.tr/ ediz/bilmdg/papers/sofsem02.pdf.
4. N.S. Chang and K.S. Fu. Query by pictorial example. *IEEE Transactions on Software Engineering, SE6*, 6:519–524, 1980.
5. S. Chang, W. Chen, H.J. Meng, H. Sundaram, and D. Zhong. VideoQ: An automated content-based video search system using visual cues. In *Proc. of ACM Multimedia*, pages 313–324, Seattle, Washington, USA, 1997.

6. S.K. Chang, Q.Y. Shi, and C.W. Yan. Iconic indexing by 2-d strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:413–428, 1987.
7. W.W. Chu, A.F. Cardenas, and R.K. Taira. A knowledge-based multimedia medical distributed database system - KMED. *Information Systems*, 20(2):75–96, 1995.
8. M. E. Dönderler, E. Saykol, Ö. Ulusoy, and U. Güdükbay. BilVideo: A video database management system. *IEEE Multimedia*, 1(10):66–70, January/February 2003.
9. M.E. Dönderler, Ö. Ulusoy, and U. Güdükbay. A rule-based video database system architecture. *Information Sciences*, 143(1–4):13–45, 2002.
10. C. Donnelly and R. Stallman. Bison: The yacc-compatible parser generator. Online manual, `http://www.combo.org/bison/`, 1995.
11. M. Egenhofer and R. Franzosa. Point-set spatial relations. *Int'l Journal of Geographical Information Systems*, 5(2):161–174, 1991.
12. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, 28:23–32, 1995.
13. R.H. Guting, M.H. Bohlen, M. Erwig, C.S. Jensen, N.A.Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Tran-sactions on Database Systems*, 25(1):1–42, 2000.
14. R. Hjelsvold and R. Midtstraum. Modelling and querying video data. In *Proc. of the 20th Int. Conference on VLDB*, pages 686–694, Santiago, Chile, 1994.
15. E. Hwang and V.S. Subrahmanian. Querying video libraries. *Journal of Visual Communication and Image Representation*, 7(1):44–60, 1996.
16. H. Jiang, D. Montesi, and A.K. Elmagarmid. VideoText database systems. In *Proc. of IEEE Multimedia Computing and Systems*, pages 344–351, 1997.
17. S.C. Johnson. Yacc: Yet another compiler compiler. Computing Science Technical Report 32, Bell Laboratories, Murray Hill, NJ, 1975.
18. J. Koh, C. Lee, and A.L.P. Chen. Semantic video model for content-based retrieval. In *Proc. of IEEE Multimedia Computing and Systems*, volume 1, pages 472–478, 1999.
19. T.C.T. Kuo and A.L.P. Chen. A content-based query language for video databases. In *Proc. of IEEE Multimedia Computing and Systems*, pages 209–214, 1996.
20. T.C.T. Kuo and A.L.P. Chen. Content-based query processing for video databases. *IEEE Transactions on Multimedia*, 2(1):1–13, 2000.
21. M.E. Lesk. Lex - a lexical analyzer generator. Computing Science Technical Report 39, Bell Laboratories, Murray Hill, NJ.
22. J.Z. Li. Modeling and querying multimedia data. Technical Report TR-98-05, Department of Computing Science, The University of Alberta, Alberta, Canada, 1998.
23. J.Z. Li and M.T. Özsu. Stars: A spatial attributes retrieval system for images and videos. In *Proc. of the 4th Int. Conf. on Multimedia Modeling*, pages 69–84, Singapore, 1997.
24. J.Z. Li, M.T. Özsu, and D. Szafron. Modeling of moving objects in a video database. In *Proc. of IEEE Multimedia Computing and Systems*, pages 336–343, Ottawa, Canada, 1997.
25. J.Z. Li, M.T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. In *Proc. of the 3rd Int. Workshop on Multimedia Information Systems*, pages 19–28, Como, Italy, 1997.
26. J.Z. Li, M.T. Özsu, D. Szafron, and V. Oria. Multimedia extensions to database query languages. Technical Report TR-97-01, Department of Computing Science, The University of Alberta, Alberta, Canada, 1997.
27. S. Marcus and V.S. Subrahmanian. Foundations of multimedia information systems. *Journal of ACM*, 43(3):474–523, 1996.

28. S. Markus and V.S. Subrahmanian. *Multimedia Database Systems: Issues and Research Directions (eds. V.S. Subrahmanian and S. Jajodia)*, chapter Towards a Theory of Multimedia Database Systems, pages 1–35. Springer-Verlag, 1996.
29. S. Mehrotra, K. Chakrabarti, M. Ortega, Y. Rui, and T.S. Huang. Multimedia analysis and retrieval system (MARS project). In *Proc. of the 3rd Int. Workshop on Information Retrieval Systems*, pages 39–45, Como, Italy, 1997.
30. M. Nabil, A.H. Ngu, and J.Shepherd. Modeling and retrieval of moving objects. *Multimedia Tools and Applications*, 13:35–71, 2001.
31. E. Oomoto and K. Tanaka. OVID: Design and implementation of a video object database system. *IEEE Transactions on Knowledge and Data Engineering*, 5:629–643, 1993.
32. M.T. Özsu, P. Iglinski, D. Szafron, S. El-Medani, and M. Junghanns. An object-oriented sqml/hytime compliant multimedia database management system. In *Proc. of ACM Multimedia*, pages 233–240, Seattle, WA, 1997.
33. D. Papadias, Y. Theodoridis, T. Sellis, and M. Egenhofer. Topological relations in the world of minimum bounding rectangles: A study with R-trees. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 92–103, San Jose, CA, USA, 1996.
34. V. Paxson. Flex: A fast scanner generator. Online manual, `http://www.combo.org/flex/`, 1995.
35. E.G.M. Petrakis and S.C. Orphanoudakis. Methodology for the representation, indexing and retrieval of image by content. *Image and Vision Computing*, 11(8):504–521, 1993.
36. E. Saykol. Web-based user interface for query specification in a video database system. M.S. thesis, Department of Computer Engineering, Bilkent University, Ankara, Turkey, September 2001.
37. E. Saykol, U. Güdükbay, and Ö. Ulusoy. A histogram-based approach for object-based query-by-shape-and-color in multimedia databases. *submitted journal paper, and also available as a Technical Report (BU-CE-0201) at http://www.cs.bilkent.edu.tr/tech-reports/2002/BU-CE-0201.ps.gz*.
38. A.P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proc. of IEEE Data Engineering*, pages 422–432, 1997.
39. S.W. Smoliar and H. Zhang. Content-based video indexing and retrieval. *IEEE Multimedia Magazine*, 1(2):62–72, 1994.
40. G. Ünel, M. E. Dönderler, Ö. Ulusoy, and U. Güdükbay. An efficient query optimization strategy for spatio-temporal queries in video databases. *Journal of Systems and Software (in press)*.
41. Y. Zhuang, Y. Rui, T.S. Huang, and S. Mehrotra. Applying semantic association to support content-based video retrieval. In *Proc. of IEEE VLBV98 Workshop*, pages 45–48, Urbana, IL, 1998.

A Grammar Specification of the Query Language

```
<query> := select <target> from all [where <condition>] ‘;’
    | select <target> from <videolist> where <condition> ‘;’
    | select segment [‘,’ <variablelist>] from <range>
      where <condition> ‘;’
    | select <variablelist> from <range>
      where <condition> ‘;’
    | select <aggregate> ‘(’ segment ‘)’ [‘,’ segment]
       [‘,’ <variablelist> ]
      from <range> where <condition> ‘;’

<target> := <video> [‘:’ (<number>
    | random ‘(’ <number> ‘)’)]

<aggregate> := average | sum | count

<range> := all | <videolist>

<video> := video [[last] <time> [seconds]]

<videolist> := [<videolist> ‘,’] <vid>

<condition> := ‘(’ <condition> ‘)’ | not ‘(’ condition ‘)’
    | <condition> and <condition> | <condition> or
<condition>
    | <condtype1> | <condtype2> | <condtype3> | <condtype4>

<condtype1> := <appearance> | <directional> | <topological>
    | <tdimension> | <external-predicate>

<condtype2> := <variable> <cop> (<atom> | <variable>)
    | <variable> ‘=’ <tprojection>

<condtype3> := <condition> <tmpred> <condition>
    | ‘(’ <condition> <tmpred> <condition> [<timegap>] ‘)’
<trepeat>

<condtype4> := <trajectory-query>
    | ‘(’ <trajectory-query> ‘)’ <trepeat>

<appearance> := appear ‘(’ <objectlist> ‘)’

<directional> := <direction> ‘(’ <object> ‘,’ <object> ‘)’

<topological> := <tpred> ‘(’ <object> ‘,’ <object> ‘)’

<tdimension> := <tdpred> ‘(’ <object> ‘,’ <object> ‘)’

<external-predicate> := <predicate-name>
    ‘(’ <objectlist> ‘)’

<tprojection> := project ‘(’ <object>
    [‘,’ <spatial-condition>] ‘)’
```

```
<trajectory-query> := tr ‘(’ <object> ‘,’ (<trajectory1> ‘)’
    [<similarity>] | <trajectory2> ‘)’
    [<simthreshold>]) [<timegap>]

<trajectory1> := <variable> | ‘[’ <dircomponent> ‘,’
    <dispcomponent> ‘]’

<trajectory2> := ‘[’ <dircomponent> ‘]’

<dircomponent> := ‘[’ <dirlist> ‘]’

<dispcomponent> := ‘[’ <displist> ‘]’

<similarity> := <simthreshold> [dirweight <dirweight>
    | dspweight <dspweight>]

<simthreshold> := sthreshold <threshold>

<timegap> := tgap <time>

<displist> := [<displist> ‘,’] <dspvalue>

<dirlist> := [<dirlist> ‘,’] <fdirection>

<trepeat> := repeat [<number>]

<spatial-condition> := ‘(’ <spatial-condition> ‘)’
    | not ‘(’ <spatial-condition> ‘)’
    | <spatial-condition> and <spatial-condition>
    | <spatial-condition> or <spatial-condition>
    | <appearance> | <directional> | <topological>
    | <tdimension> | <variable> <cop> <object>
    | <external-predicate>

<direction> := left | right | above | below | <fdirection>

<fdirection> := west | east | north | south | northeast
    | southeast | northwest | southwest

<tpred> := equal | contains | inside | cover | coveredby
    | disjoint | overlap | touch

<tdpred> := infrontof | behind | sinfrontof | sbehind
    | tfbehind | tdfbehind | samelevel

<tmpred> := before | meets | overlaps | starts | during
    | finishes | ibefore | imeets | ioverlaps | istarts
    | iduring | ifinishes

<object> := <variable> | <atom>

<objectlist> := [<objectlist> ‘,’] <object>

<variablelist> := [<variablelist> ‘,’] <variable>

<vid> := (1-9)(0-9)*
```

<number> := (1-9)(0-9)*

<time> := (1-9)(0-9)*

<variable> := (A-Z)(A-Za-z0-9)*

<atom> := (a-z)(A-Za-z0-9)*

<predicate-name>[4]:= (a-z)(A-Za-z0-9_)*

<cop> := ‘=’ | ‘‘!=’’

<threshold> := 0 ‘.’ (0* (1-9) 0*)+

<dspweight> := 0 [‘.’ (0-9)*] | 1

<dirweight> := 0 [‘.’ [0-9]*] | 1

<dspvalue> := (1-9)(0-9)*

[4] Lexer recognizes such a character sequence as an external predicate name iff it is different from any predefined predicate and construct in the language.

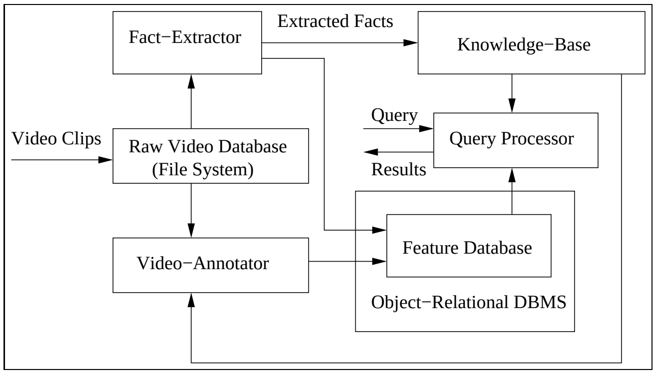

Fig. 1 *BilVideo* System Architecture

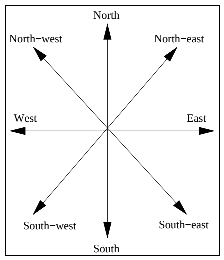

Fig. 2 Directional Coordinate System

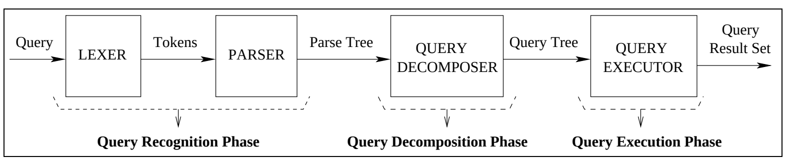

Fig. 3 Query Processing Phases

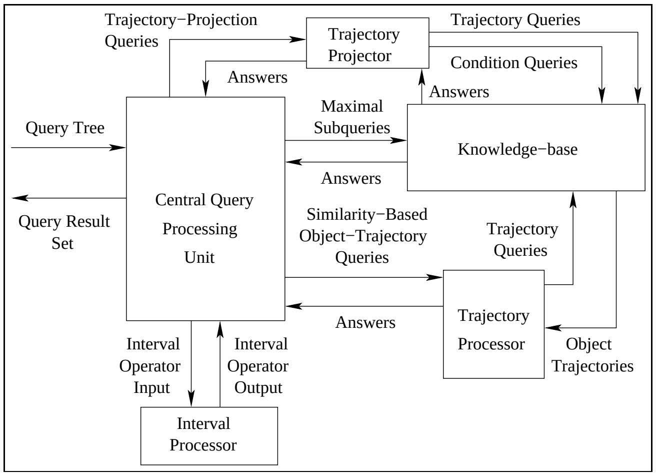

Fig. 4 Query Execution

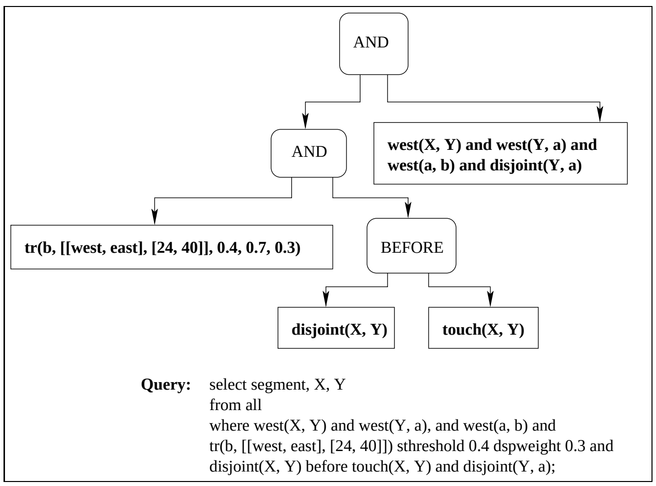

Fig. 5 The query tree constructed for Query 1

Table 1 Definitions of our 3D relations on the z-axis of three-dimensional space

Relation	Inverse	Meaning
A infrontof B	B behind A	AAA BBB (A overlaps B) or AAABBB (A meets B) or AAA BBB (A before B)
A strictlyinfrontof B	B strictlybehind A	AAA BBB (A before B) or AAABBB (A meets B)

A samelevel B	B samelevel A	AAA
		BBBBBB (A starts B)
		or
		   AAA
		BBBBBB (A finishes B)
		or
		  AAA
		BBBBBB (A during B)
		or
		AAA
		BBB (A equal B)

A touchfrombehind B	B touchedfrombehind A	BBBAAA (B meets A)

Table 2 Interval Intersection (AND)

Input Interval 1	Input Interval 2	Result Set	Result Interval Type
I_1 (Atomic)	I_2 (Atomic)	I_1 iff $I_1 \supseteq I_2$ $I_{1_s} \leq I_{2_s} \wedge I_{1_e} \geq I_{2_e}$ I_2 iff $I_1 \subset I_2$ $I_{2_s} < I_{1_s} \wedge I_{2_e} > I_{1_e}$ otherwise, Ø	Atomic
I_1 (Atomic)	I_2 (Non-atomic)	I_1 iff $I_2 \supseteq I_1$ otherwise, Ø	Atomic
I_1 (Non-atomic)	I_2 (Atomic)	I_2 iff $I_1 \supseteq I_2$ otherwise, Ø	Atomic
I_1 (Non-atomic)	I_2 (Non-atomic)	$[I_s, I_e]$ iff I_1 overlaps I_2 $I_s = I_{1_s}$ iff $I_{1_s} \geq I_{2_s}$ otherwise, $I_s = I_{2_s}$ $I_e = I_{1_e}$ iff $I_{1_e} \leq I_{2_e}$ otherwise, $I_e = I_{2_e}$ otherwise, Ø	Non-atomic

Table 3 Interval Union (OR)

Input Interval 1	Input Interval 2	Result Set	Result Interval Type
I_1 (Atomic)	I_2 (Atomic)	$\{I_1, I_2\}$	Atomic
I_1 (Atomic)	I_2 (Non-atomic)	$\{I_1, I_2\}$	Atomic and Non-atomic
I_1 (Non-atomic)	I_2 (Atomic)	$\{I_1, I_2\}$	Non-atomic and Atomic
I_1 (Non-atomic)	I_2 (Non-atomic)	$[I_{1_s}, I_{2_e}]$ if $I_{2_s} = I_{1_e} + 1$ $[I_{2_s}, I_{1_e}]$ if $I_{1_s} = I_{2_e} + 1$ $[I_s, I_e]$ if I_1 overlaps I_2 $I_s = I_{1_s}$ iff $I_{1_s} \geq I_{2_s}$ otherwise, $I_s = I_{2_s}$ $I_e = I_{1_e}$ iff $I_{1_e} \leq I_{2_e}$ otherwise, $I_e = I_{2_e}$ otherwise, $\{I_1, I_2\}$	Non-atomic

Table 4 Specifications of Real Video Data

Video	# of Frames	# of Objects	Max. # of Objects in a Frame
Jornal.mpg	5254	21	4
Smurfs.avi	4185	13	6