A Visual Query Language for Identifying Temporal Trends in Video Data*

Stacie Hibino and Elke A. Rundensteiner
EECS Department, Software Systems Research Laboratory
1301 Beal Avenue, University of Michigan, Ann Arbor, MI 48109-2122
hibino@eecs.umich.edu, rundenst@eecs.umich.edu

Abstract

Video is becoming a popular medium for data collection. The challenge of analyzing video data is to abstract and conclude quantitative results from such a rich, qualitative medium. The focus of our research is to support video analysis *by developing a user-friendly interactive visualization environment to query video data using spatio-temporal characteristics and to review visual results for trend analysis. In this paper, we present our approach for identifying temporal trends in video data via querying for* relationships *between video annotations. Our approach allows users to analyze the video in terms of* temporal *relationships between events (e.g., events of type B frequently follow events of type A). We present a temporal visual query language for specifying relative temporal queries between sets of annotations. This query language builds on the notion of dynamic query filters and significantly extends them with temporal query support. It is tailored for temporal* analysis*—allowing users to pose queries as well as to* browse *the data in a* temporally continuous *manner, thereby aiding them in the discovery of temporal trends.*

1 Introduction

Video is becoming a more common form of data collection. It is being collected by educational researchers for classroom studies, by software evaluators during usability testing, by scientists studying microscopic entities, etc. An advantage of video data is that it preserves detailed information which would be difficult to capture by other means of data collection. The challenge of analyzing video data is to abstract and conclude quantitative results from such a rich, qualitative medium.

A few video analysis[1] tools exist [13, 20, 9], but they are limited in that they focus more on the creation of annotations rather than the analysis of the video. They require users to explicitly code relationships within the video *a priori* rather than discover them during analysis. In contrast, our work focuses on the *analysis* process.

Similar to previous work [13, 16], we are also using annotations for coding the video data to avoid the high computation costs of image recognition, and to work at a higher level of abstraction. Rather than using annotations to *pre-code* relationships, however, we restrict them to only abstracting information about *atomic* objects and events. Coding only the atomic content requires less storage space and less time than coding every relationship between each atom, especially when users are interested in several different types of relationships. In addition, it provides more flexibility in the analysis process, aiding users in *discovering relationships* rather than requiring them to predefine or pre-code all possible relationships. Thus, once atomic information has been encoded, our environment can be used for exploring the data in search of data trends. This paper focuses on identifying *temporal* trends in the video data.

Consider the case where educational researchers analyze classroom video data to study social interactions in the classroom. Once the video data has been coded with annotations (e.g., to indicate in which segments of video the teacher is speaking, which segments the student is speaking, etc.), the researchers can begin exploring the relationships between different types of classroom events. In particular, they could search for temporal trends in the video data by specifying queries to locate occurrences of, and relationships between subsets of annotations (e.g., do students generally speak for shorter periods of time than the teacher?).

In order to support the trend searching process, we are designing an interactive visualization environment consisting of a visual query language (i.e., a direct manipulation query interface) linked to a spatio-temporal visualization. The visualization is *dynamically* updated as users incrementally adjust query parameters via direct manipulation of buttons and sliders. This paper describes our overall environment and then presents the details of our *temporal* visual query language (TVQL). Our TVQL allows users to specify relative temporal queries and can be used to *browse* video data in a *temporally continuous* manner. We introduce the use of temporal diagrams as visual clarifications complementing our VQL approach.

Section 2 describes our annotation model. Section 3 presents an overview of our approach, while Section 4 describes our TVQL for specifying relative temporal queries. Some preliminary evaluation of the query

*This work was supported in part by UM Rackham Fellowship, NSF NYI #94-57609, and equipment support from AT&T.

[1]In this paper, video analysis refers to the process of identifying trends and relationships between events in the video data. This is in contrast to bit-level video analysis such as that used for object extraction.

language is provided in Section 5. Section 6 discusses related work. Finally, Section 7 presents our current status along with a summary of contributions.

2 Annotation model

This section provides an overview of the annotation model we assume by defining the terms video, video segment, event, and video annotation as they are used in this paper. While this section primarily describes the *temporal* aspects of our annotation model, the spatial aspects of the model are described elsewhere [10].

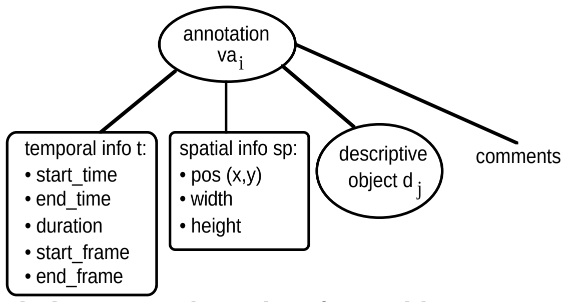
Video. In this paper, a *video* object v has the following temporal characteristics: a start time, v.t.start_time (defined as time 0); an end time, v.t.end_time, defining the finite (maximum) length of the video in seconds (up to two decimal points); a start frame, v.t.start_frame, indicating the physical video frame number corresponding to the start time of the video; a corresponding end frame v.t.end_frame. We define a function VF(Vtime) which maps a given video time (in seconds) to the corresponding frame: VF(Vtime) = round ((Vtime * fps) + f_offset), where fps = the number of frames per second of the video and f_offset = the frame number corresponding to time 0 of the video. Similarly, we define a function VT(frameNum) —> time: VT(frameNum) = (frameNum - f_offset)/fps, where the resulting time will be rounded to two decimal points. We assume that a video object is one *continuous* medium, including the start and end frames as well as *every* frame between them. v is a complex object decomposed of an ordered sequence F of frame objects, $F = [f_0, f_1, ..., f_z]$.[2]

Video segment. A *video segment* vs is a continuous subset of frames of the video object (i.e., vs is defined over frames $[f_i, . . ., f_j]$, where $0 \leq i, j \leq z$). Similar to the video object, a video segment has a start frame (vs.start_frame ≥ v.start_frame) and an end frame (vs.end_frame ≤ v.end_frame) as well as starting and ending times. In addition, the length of a video segment must be less than or equal to that of the video (i.e., vs.end_frame-vs.start_frame ≤ v.end_frame-v.start_frame).

Video annotations. A *video annotation* va is used to mark real-world situations of the video, also referred to as *events*. Each va temporally links a descriptive object to a video segment and spatially links it to a position on top of the video (i.e., to a position on the display relative to the video window). An annotation can consist of a text, audio or graphical (e.g., square, circle, arrow, etc.) object. Each annotation also has comments and a link to content-based characteristics, stored in the descriptive object.

More formally, for a given video v_p, we denote the set of annotations $Ann(v_p) = \{va_1, va_2, . . . , va_n\}$. Each annotation va_i ($1 \leq i \leq n$) has the following characteristics:

[2]We assume that we are dealing with uncompressed video, so that (unlike MPEG files) every frame is complete.



Each descriptive object d_j, referenced by va_i, contains semantic information about the event being annotated. This includes information such as the name of the object, action, and category which describe the event. The descriptive object also contains a reference to a media object md_k. A media object is the object which is visually displayed or orally played when referenced by an annotation through the descriptive object. For example, if md_1 is a talk bubble icon and d_1 describes events where the teacher is talking, then each va_i which references d_1 indicates a segment of the video when the teacher is talking. More details about the descriptive and media objects are provided elsewhere [10]. Although we are currently requiring researchers to manually input annotations, previous work by others in the area of object extraction (e.g., [14]) could eventually be used to generate annotations automatically.

3 Overview of our approach

3.1 Overall project goals and approach

Our overall goal is to support users in analyzing video data by providing tools for querying video annotations in search of relationships and data trends. The results retrieved from relative queries need to be visually presented to facilitate this searching for trends. The query and review part of the video analysis process is similar to the process used in visual information seeking (VIS) [3]. VIS is a process for *browsing* database information; it is characterized by rapid filtering, progressive refinement, continuous reformulation of goals, and visual scanning to identify results. These characteristics make VIS particularly well suited for searching for trends in video annotations. We are thus adopting this methodology as our underlying framework.

Figure 1 presents our overall approach to applying and extending VIS to the problem of video data analysis. In our MultiMedia VIS (MMVIS), users first code the video data with annotations. These annotations are stored in an underlying database. Users can then explore and analyze the video through iteratively specifying queries using a visual query language (VQL) and reviewing the visualization of results presented. Similarly to VIS, our interface will be composed of *dynamic query filters* to allow rapid adjustment of query parameters via the use of buttons and sliders. This is in contrast to text-based query languages, where query specification and modification are typically much more complicated and non-intuitive.

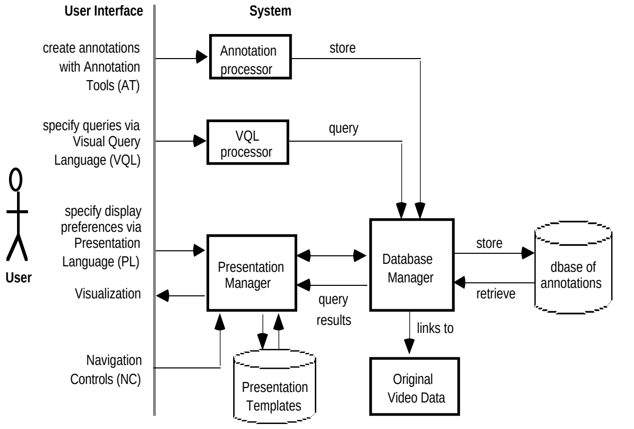


Figure 1. Overall framework for the multimedia interactive visualization environment (MMVIS).

The queries are interpreted by a VQL processor and then forwarded to the Database Manager. The retrieved results are passed to a Presentation Manager. The Presentation Manager takes the query results, along with any user-defined display preferences and updates a visualization. In this way, the visualization is updated every time users make changes to any query filter. Users can visually scan the results to look for data trends. If no trends are found, they can use the presentation language (PL) to clarify the visualization, the navigation controls to further explore query results, or the VQL to incrementally adjust the query. In addition, if users wish to test a new hypothesis or explore different characteristics of the data, they can use the VQL to do so. Thus, queries are expressed *incrementally* as users specify desired values for each query parameter. In such an environment, users can gain a sense of causality between adjusting a query filter and the corresponding changes presented in both the other query filters and the visualization.

3.2 Specific aims of this paper

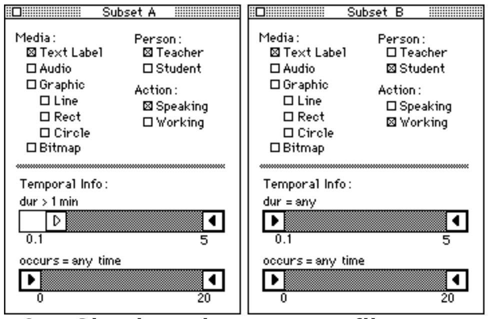
Specifically, the goals of this paper are (1) to define a general temporal visual query language (TVQL) that can be easily integrated into a variety of applications and (2) to design a TVQL interface that is easy and intuitive to use by application (i.e., video annotation and analysis) users who typically are query language novices. Thus, the specific problem addressed in this paper is that of the users' need for a simple interface for specifying relative temporal queries to video data. *Relative* queries are necessary for examining *relationships* between events and thus for identifying trends. A *visual* language is desired to correspond with the graphical nature of the objects in the database and is much more suitable for the type of novice query users we are targeting. Finally, a *temporal* language is required to facilitate searching for temporal data trends.

3.3 The problem of specifying *relative* queries

Currently, the VIS methodology does not handle relative queries [3]. That is, while we can use the approach to identify and specify interesting *subsets* of data, we cannot specify *relationships* between these subsets. In order to better understand this problem, consider a query such as "show me all annotations where the student is working *while* the teacher is speaking." While we can use dynamic queries (DQs) to select individual subsets such as the "student working" and "teacher speaking" annotations, we cannot use DQs to specify a desired relationship between members of these subsets. In the case of the example, we need to be able to specify a relative condition that holds true for each (teacher-speaking, student-working) pair returned. Rather than specifying a range of *absolute* conditions, we need to specify *relative* ones.

3.4 Our solution to specifying relative queries

Extending the use of DQs to handle *relative* temporal queries of video data requires binding subsets of the data to variables and specifying temporal relationships between these subsets. Two subsets can be specified through the use of two sets of standard query filters. We thus provide a query palette for specifying each subset. Parameters specified in the Subset A query palette automatically bind the subset formed to the variable A. The corresponding functionality is provided for binding the second subset to variable B.


Subset A
Media: [✓ Text Label] Audio Graphic Line Rect Circle Bitmap
Person: [✓ Teacher] Student
Action: [✓ Speaking] Working
Temporal Info:
dur > 1 min
0.1 5
occurs = any time
0 20

Subset B
Media: [✓ Text Label] Audio Graphic Line Rect Circle Bitmap
Person: Teacher [✓ Student]
Action: Speaking [✓ Working]
Temporal Info:
dur = any
0.1 5
occurs = any time
0 20

Figure 2. Simple subset query filters.
The Subset A palette binds the subset "teacher talking" to variable A, and the Subset B query palette binds the "student working" subset to variable B.

The simple example in Figure 2 illustrates how we can use query filters in the Subset A palette to bind the subset "teacher talking" to variable A, and those in the Subset B query palette to bind the "student working" subset to variable B. The actual subset query palettes allow the user to set more parameters and parameter values (e.g., to specify annotation names, categories, additional actions, etc.). The sliders at the bottom of each subset palette allow users to specify *absolute* temporal information for each subset. This is in contrast to the temporal relationship *R between* sets *A* and *B* that will be specified with *specialized temporal query filters* (see

Section 4). Progressive refinement of queries will be preserved by allowing the adjustment of query filters for *A*, *B*, or *R* at any time and in any order. Note that the query filters for specifying numerically valued ranges (e.g., for duration in Figure 2) are double-thumbed sliders. This type of filter was introduced in [3] for specifying ranges of values. Each thumb has an arrow, pointing towards the range being specified. The thumb arrow is filled to indicate that the endpoint of a range is included, or empty to indicate that the endpoint of a range is excluded. Ranges between slider thumbs are filled in for further clarification. The text above the slider thumbs indicates exact values. The label to the left of the slider indicates the parameter being adjusted. The values below the slider indicate the range of valid values.

4 The Temporal Visual Query Language

This section presents a temporal visual query language (TVQL) for specifying relative temporal queries. TVQL is composed of temporal query filters, a disjunctive operator, and a macro operator for saving and reusing query specifications. Our goal here is to gracefully integrate the temporal filters and operators with the standard DQs and VIS properties introduced in Section 3.

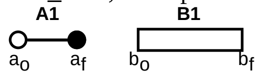
4.1 Relative temporal position

Relative temporal queries have two components—relative temporal position and relative duration. Discussion of relative duration specification of TVQL is omitted due to space limitations but can be found elsewhere [10].

Given two events A1 and B1, the relative temporal position between these events refers to the relationship between the temporal starting and ending points of the events. It describes information such as whether A1 starts before B1 or whether A1 and B1 finish at the same time.

4.1.1 Primitive temporal relationships. Allen [1] describes thirteen primitive temporal relationships between two events: *before, meets, during, starts, finishes, overlaps,* the symmetric counterparts to these six relationships, and the *equals* relationship (see first column of Table 1). These temporal relationships can be described in terms of the relationships between the temporal starting and ending points of each of the events.

Consider two events A1 and B1, with starting and ending points a_o,a_f and b_o,b_f, respectively (i.e., where a_o = A1.t.start_time, a_f = A1.t.end_time, b_o = B1.t.start_time, and b_f = B1.t.end_time):



There are four pairwise endpoint relationships between these events:

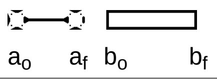
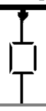
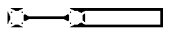
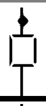
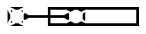
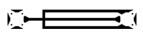
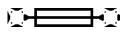
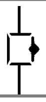
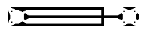
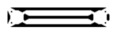
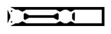
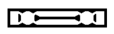
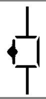
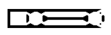
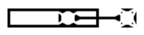
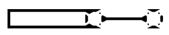
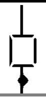
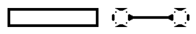
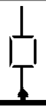
$a_o\ \theta\ b_o$, $a_o\ \theta\ b_f$, $a_f\ \theta\ b_o$, and $a_f\ \theta\ b_f$,

where $\theta \in \{ <, >, = \}$. We assume that the conditions $a_o < a_f$ and $b_o < b_f$ always hold true. These relationships can be redefined in terms of differences:

$a_o - b_o\ \theta\ 0$, $a_o - b_o$ = A1.t.start_time - B1.t.start_time;
$a_o - b_f\ \theta\ 0$, $a_o - b_f$ = A1.t.start_time - B1.t.end_time;
$a_f - b_o\ \theta\ 0$, $a_f - b_o$ = A1.t.end_time - B1.t.start_time;
$a_f - b_f\ \theta\ 0$, $a_f - b_f$ = A1.t.end_time - B1.t.end_time;

where $\theta \in \{ <, >, = \}$. The benefit of describing these relationships in terms of differences is that it allows us to

Table 1. Primitive temporal relationships
[In last four columns, double-lined borders and bold values indicate minimum endpoint relationships required.]

r_t	graphical description	Freksa's icons [6]	a_o - b_o	a_o - b_o	a_f - b_f	a_f - b_o
< before			< 0	< 0	< 0	**< 0**
m meets			< 0	< 0	< 0	**= 0**
o overlaps			< 0	**< 0**	**< 0**	**> 0**
fi finished by			< 0	**< 0**	**= 0**	> 0
di contains			< 0	**< 0**	**> 0**	> 0
si started by			< 0	**= 0**	**> 0**	> 0
= equals			< 0	**= 0**	**= 0**	> 0
s starts			< 0	**= 0**	**< 0**	> 0
d during			< 0	**> 0**	**< 0**	> 0
f finishes			< 0	**> 0**	**= 0**	> 0
oi over-lapped by			**< 0**	**> 0**	**> 0**	> 0
mi met by			**= 0**	> 0	> 0	> 0
> after			**> 0**	> 0	> 0	> 0

specify quantitative and continuous ranges of values, which is a natural specification to be handled by DQ sliders. All of Allen's thirteen temporal relationships [1] can be uniquely defined by specifying one to three of these endpoint relationships. Double-lined boxes in the last four columns in Table 1 summarize the minimum endpoint relationships required to uniquely specify a corresponding primitive temporal relationship (r_t). The relationships not double-lined are automatically inferred.

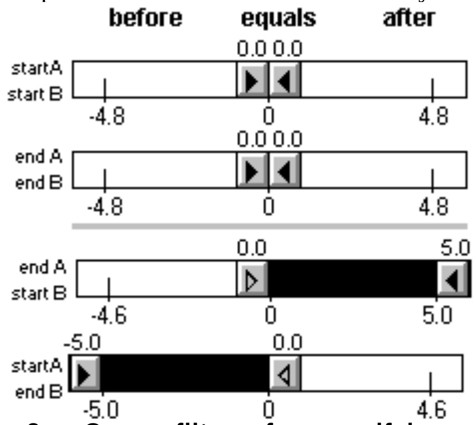


Figure 3. Query filters for specifying relative temporal position queries.
Example: Given a subset A bound to "student S1 working" and a subset B bound to "student S2 working," we can specify the query "Show all situations where students S1 and S2 start and finish working at the same time" by setting startA-startB = 0 and endA-endB = 0.

Given four dynamic query filters for the four endpoint relationships, we can thus specify any *primitive* temporal relationship. Therefore, we incorporate these filters into TVQL as a query palette. Because a_o - b_o and a_f - b_f query filters can be used to uniquely define over half of the primitive relationships, they are placed as the top two filters (see Figure 3).

Rather than requiring users to translate their queries into notations (e.g., a_o) and algebraic equations, we provide full labels to the left of the filters and qualitative English text above the filters. In this way, users can "read" the query they have specified. In Figure 3, the top query filter can be read as "the start of A equals the start of B." Similarly, the bottom query filter reads "the start of A is before the end of B."

As in the case of other query filters (i.e., those used to specify a subset), these four temporal query filters are bound to one another so that users cannot specify invalid queries. That is, when the user changes one query filter, the other three are automatically updated to exclude any invalid ranges, if necessary. In the example in Figure 3, the user needs to set both of the top two filters. However, only one filter can be set at a time. Suppose the user first specifies the top filter (i.e., setting it 0). As soon as this new value is set, the bottom two filters are automatically updated so that endA-startB > 0 and startA-endB < 0. These latter two filters are updated because these are the only valid ranges for them when startA-startB= 0 (see the last four columns of Table 1). The endA-endB filter remains unchanged, because it can have any valid value when startA-startB = 0. After the user sets endA-endB = 0, neither of the other three filters are updated because they are already set to include valid values. The automatic updating is described in more detail elsewhere [10].

4.1.2 Combining temporal primitives. In addition to the primitive temporal relationships, we may also need to specify a combination of these primitives. For example, we may wish to find all events where student S1 finishes working 0-1 minutes before student S2 starts. Because the filters allow us to specify *continuous* ranges of values for the endpoint relationships, we can use them to specify such a combination of temporal relationships. That is, we can set the endA-startB filter to values between -1 and 0 (Figure 4). Note that this one query palette now specifies a disjunctive combination of related primitive temporal relationships, corresponding to the disjunction of several primitive temporal predicates (e.g., as possible in [18]). This part of the example is equivalent to requesting events of type A which are "before" or "meet" events of type B.

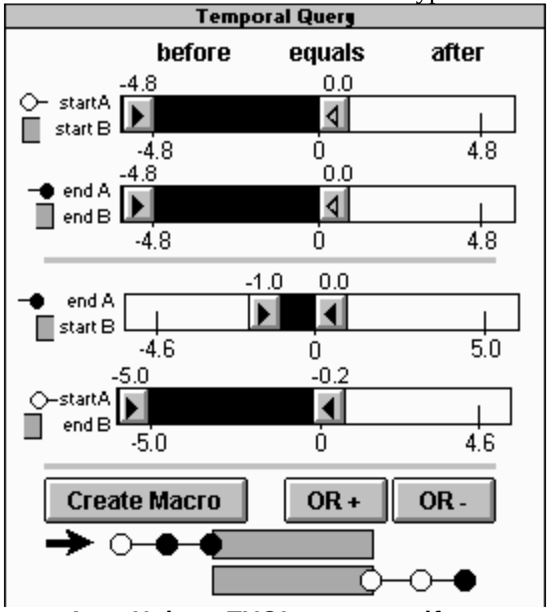


Figure 4. Using TVQL to specify temporal neighborhoods and other disjunctions.
Example: Student S1 finishes working 0-1 minutes before student S2 starts, OR Student S1 starts working 0-1 minutes after S2 finishes.

Moreover, because the filters can specify ranges and are bound to one another to prevent invalid combinations, we can use the filters to specify single "neighborhoods" of related temporal primitives. Freksa [6] defines two primitive temporal relationships between two events to be (*conceptual*) *neighbors* if a continuous change (e.g., shortening, lengthening, or moving of the duration of the events) to the events can be used to transform either relation to the other [without passing through an additional primitive temporal relationship]. Thus, the “overlaps” ( ) and “finished by” ( ) relations in Table 1 *are* neighbors, because we can move the ending point of A from the middle of B to the end of B without specifying any additional primitive relationship. On the other hand, the “overlaps” ( ) and “contains” ( ) relations are *not* neighbors. This is because we cannot move the ending point of A past the ending point of B without first passing through the “finished by” relation. A set or combination of temporal relationships between two events then forms a (*conceptual*) *neighborhood* if it consists of relations that are path-connected through conceptual neighbors. The use of continuous dynamic query filter *sliders* thus allows us to capture meaningful disjunctions of the primitives.

4.2 Disjunctive operator

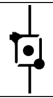
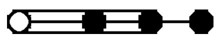
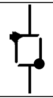
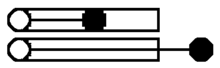
The disjunctive operator allows users to create a temporal query consisting of any combination of primitive and neighborhood queries. The operator consists of two buttons—an “OR+” and an “OR-” button (see Figure 4). When users click on OR+, the current state of the temporal query filters is saved in the current temporal diagram (see Section 4.3.1), and a copy of that diagram is added to the bottom of the palette. An arrow to the left of the diagrams indicates the current part of the OR query that is being specified or edited. If users wish to edit a previously specified part, they can simply click on its corresponding diagram, which moves the arrow to indicate that that part of the OR query is being edited. In addition, the dynamic query filters are also automatically updated to match the part of the disjunctive temporal query being edited. The *two* temporal diagrams shown in Figure 4 indicate that the user is specifying a disjunctive query. The arrow pointing to the top diagram indicates that the values of the query filters correspond to the top diagram. The OR- button can be used to remove the currently selected part of the disjunction.

4.3 Temporal diagrams

Although users can quickly adjust the query filters to specify any temporal query, they may not be able to easily determine whether or not the query specified corresponds to the desired query semantics. Part of this ambiguity may be inherent in the finely-grained definitions of the primitives (e.g., three different primitives meet the specification that events “start” at the same time). While the additional English text included at the top of the palette should provide some help, it only provides information about individual parts of the query rather than the whole query. In order to increase understandability, we have incorporated temporal diagrams for quick visual confirmation of the temporal query specified.

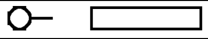
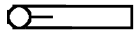
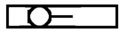
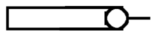
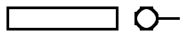
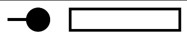
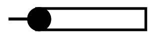
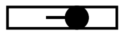
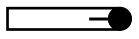
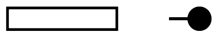
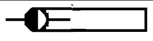
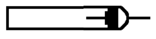
Temporal diagrams are visual representations of the specified temporal query which are displayed at the bottom of the temporal query palette (see Figure 4). In the diagrams, the rectangle refers to B events, while the connected circles are used to designate potential relative positions of A events. The open circles represent possible starting points for set A, while filled circles represent possible ending points. Freksa [6] uses similar diagrams to describe individual relationships but then uses different icons to describe combinations of temporal relationships (see columns 2 and 3 of Table 1). While the use of Freksa’s icons provides a compact visual description, it is more difficult to decipher than our temporal diagrams. This is partly due to the fact that individual endpoint relationships (e.g., differences between starting vs. ending points) are obscured. That is, temporal queries model relationships over time, and this continuous temporal dimension is not explicitly captured in Freksa’s icons. Table 2 compares the use of Freksa’s icons to our temporal diagrams.

Table 2. Freksa’s icons vs. temporal diagrams

Freksa’s icons	Temporal Diagrams
	
	

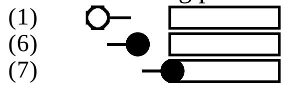
We have developed a transformation function to automatically generate the temporal diagrams from the relative temporal query filters. The primitive operators of this transformation function are defined using Table 3. We introduce a half-filled circle (e.g., (11) and (12) in Table 3) to denote the overlap of starting and ending points of A.

Table 3. Transformation function used to define temporal diagrams

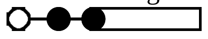
Filter Value(s)	Visualized by	
a_o-b_o < 0		(1)
a_o-b_o = 0, a_f-b_o ≠0		(2)
a_o-b_o > 0, a_o-b_f < 0		(3)
a_o-b_f= 0, a_f-b_f ≠ 0		(4)
a_o-b_f> 0		(5)
a_f-b_o < 0		(6)
a_f-b_o = 0, a_o-b_o ≠ 0,		(7)
a_f-b_o > 0, a_f-b_f < 0		(8)
a_f-b_f = 0, a_o-b_f≠ 0		(9)
a_f-b_f > 0		(10)
a_o-b_o = 0, a_f-b_o = 0		(11)
a_f-b_f = 0, a_o-b_f = 0		(12)

The transformation function is used to determine which of the five starting points of A are specified, which of the five ending points of A are specified, and whether there is a case where a starting and ending point of A overlap at either the starting or ending points of B. The function is complete because it tests for each possible endpoint position.

In general, the relative temporal position query filters pass values to the transformation function. Each part (i.e., line) of the function is evaluated and the visual results of each valid part are then collapsed into one temporal diagram. Consider the first part of the example in Figure 4: "student S1 finishes working 0-1 minutes before student S2 starts working." In this example, we have $a_o-b_o < 0$, $a_f-b_f < 0$, $-1 \leq a_f-b_o \leq 0$, and $a_o-b_f < 0$. Using the transformation operators depicted in Table 3, we see that the following parts evaluate to true:



This leads to the following composite diagram:



Because only continuous ranges are specified by the query filters, all valid starting and ending point combinations of the diagrams are included. A new temporal diagram is derived for each part of a query formed using the disjunctive OR+ operator (see Figure 4).

The temporal diagrams presented in this section represent a valuable aid for confirming the specified query to the users in a visually intuitive manner. This should increase the speed and certainty that users modify or construct complex queries. We consider these diagrams to be unique augmentations to our temporal query language, increasing the utility of our approach to novice users.

5 Preliminary evaluation

```
range of descr(Ai).content.name = "student S1"
range of descr(Ai).content.action = "working"
range of descr(Bj).content.name = "student S2"
range of descr(Bj).content.action = "working"
retrieve into A_R_B (a=Ai.id, b=Bj.id)
where Ai.id ≠ Bj.id
when  Ai.t.start_time - Bj.t.start_time = 0
      & Ai.t.end_time - Bj.t.end_time = 0
      & (Ai.t.start_time - Bj.t.end_time ≥ -300
        & Ai.t.start_time - Bj.t.end_time < 0)
      & (Ai.t.end_time - Bj.t.start_time > 0
        & Ai.t.end_time -Bj.t.start_time ≤ 300)
      & Ai.t.duration - Bj.t.duration = 0;
```

Figure 5. Example mapping of TVQL query from Figure 3 to text-based query.

TVQL queries can be automatically mapped into text-based queries [10]. Figure 5 shows the results of mapping the TVQL example of Figure 3. The contrast between the TVQL and text-based queries illustrates the advantage of our visual approach. By using TVQL, users can specify temporal constraints by simply adjusting the ranges of the temporal query filters. They can use the filters to *browse* the database without having to remember the syntax of a text-based language. Moreover, they can use *direct manipulation* to incrementally adjust their queries. In the text-based language, they would have to retype the query over and/or regain and then edit the text.

On the other hand, a forms-based query interface might address some of the drawbacks of a text-based query language. We hypothesize, however, that our visual query approach will be more effective than a forms-based language for the purpose of searching for *temporal trends* in video data. This hypothesis is supported by a study comparing the use of dynamic queries to a forms-based query language [2]. In this study, users performed significantly better using dynamic queries over a forms-based approach for three out of five tasks, one of which involved looking for trends in the data.

We are currently in the process of evaluating users' conceptual understanding of the TVQL interface through user testing. Preliminary results indicate that the dynamically updated temporal diagrams are strong visual aids explaining the semantics of the query to the users.

6 Related work

Research in temporal queries has focused more on historical (discrete) databases rather than on databases of temporal media (e.g., [19]). They focus on *discrete* changes entered into the databases as an effect of changes made in the real world (e.g., John broke his leg on April 10, 1994) rather than changes in continuous, dynamic media such as video.

While some work has begun in temporal queries of continuous, dynamic media, such work has tended to focus on issues other than analysis. For example, work by [12] focuses on synchronization and multimedia playback. Work by Davis [5] focuses on repurposing video, using iconic annotations. While his iconic language supports temporal encodings including *relative* ones, it requires users to select icons for each type of temporal relationship individually. In addition, it is not clear whether the language is restricted to only retrieving pre-coded relationships. In contrast, our TVQL allows users to *discover* temporal relationships and to explore the video data in a *temporally continuous* manner.

Besides the work in dynamic query filters by Ahlberg et al. [2], previous work in graphical or visual query languages has focused on diagrammatic, forms-based, or iconic approaches. Diagrammatic query languages allow users to specify queries by using direct manipulation of schema-type constructs to represent predicates on objects and their relationships (e.g., [21, 17]). This requires not only precise knowledge of the schema, but it also lacks the continuous browsing support provided by our TVQL. Similarly, while novel techniques have been used to combine forms-based queries with automated object extraction from videos [15], such an interface requires users to specify temporal primitives as well as ranges of temporal values by hand, thus disrupting the ability to review video data in a temporally continuous manner.

Previous work in multimedia databases (MMDBs) has focused on semantic-based retrieval, image analysis, and video indexing using bit-level analysis (e.g., [11, 7, 8]). Such approaches have focused on *locating* multimedia objects or events (typically at the pattern or object representation level) rather than *analyzing* the relationships between these events. We distinguish our work from other work in MMDBs by using *both temporal and spatial* characteristics[3] of the media, as well as by addressing needs for both retrieval and (relative) analysis.

7 Conclusion, status and future work

In this paper, we have presented a framework for video analysis based on applying and adapting a visual information seeking (VIS) approach [3] to video data. In our extensions to VIS, we have defined a temporal visual query language (TVQL) for specifying *relative temporal* queries on video data. The four query filters used for relative temporal position enable users to specify any *primitive* temporal relationship as well as *conceptual neighborhood* combinations of these primitives. Together, the temporal query filters provide users with a visual paradigm for *browsing* the video data through direct manipulation and in a *temporally continuous* manner. We also introduced the use of temporal diagrams for enhancing the clarity of the specified query.

The primary contributions of this work include a VIS approach to video analysis, the temporal visual query language (TVQL) for specifying *relative* temporal queries and for facilitating *temporal analysis* (i.e., searching for temporal trends in video data), and a transformation function for deriving temporal diagrams. This work is general, and can easily be applied to different domains and other dynamic media.

The work presented in this paper is just one component of the design of an integrated environment for video analysis. We have implemented our TVQL in a Windows-based multimedia pc (MPC) environment and are currently conducting a usability study, comparing the effectiveness of TVQL with more traditional forms-based interfaces. We are continuing work on the TVQL processor, the temporal visualizations, and the spatial and motion analysis aspects of our MMVIS environment.

References

[1] Allen, J.F. (1983). Maintaining knowledge about temporal intervals. *Comm. of ACM*, 26(11), 832-843.

[2] Ahlberg, C., Williamson, C., & Shneiderman, B. (1992). Dynamic Queries for Information Exploration: An Implementation and Evaluation. *CHI'92 Conference Proceedings*, (pp. 619-626). : ACM Press.

[3] Ahlberg, C., & Shneiderman, B. (1994). Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. *CHI'94 Conference Proceedings*: ACM Press, 619-626.

[4] Chakravarthy, A.S. (1994). Toward Semantic Retrieval of Pictures and Video. *AAAI'94 Workshop on Indexing and Reuse in Multimedia Systems*, 12-18.

[5] Davis, M. (1994). Knowledge Representation for Video. *Proceedings of the Twelfth National Conference on Artificial Intelligence*: AAAI Press, 120-127.

[6] Freksa, C. (1992). Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1992), 199-227.

[7] Gevers, T. and Smeulders, A.W.M. (1992). Indexing of Images by Pictorial Information. *Visual Database Systems, II* (E. Knuth and L.M. Wegner, Eds.), North Holland: Amsterdam, 93-100.

[8] Hampapur, A., Weymouth, T., & Jain, R. (1994). Digital Video Segmentation. *ACM Multimedia'94 Proceedings*: ACM Press, 357-364.

[9] Harrison, B.L., Owen, R., & Baecker, R.M. (1994). Timelines: An Interactive System for the Collection of Visualization of Temporal Data. *Proc. of Graphics Interface '94.* Canadian Info. Processing Society.

[10] Hibino, S. & Rundensteiner, E. (1994). A Graphical Query Language for Identifying Temporal Trends in Video Data. *University of Michigan, EECS Technical Report* CSE-TR-225-94 (Dec 94).

[11] Lenat, D. & Guha, R.V. (1994). Strongly Semantic Information Retrieval. *AAAI'94 Workshop on Indexing and Reuse in Multimedia Systems*, 58-68.

[12] Little, T.D.C. & Ghafoor, A. (1993). Interval-Based Conceptual Models for Time-Dependent Multimedia Data. *IEEE Trans. on Knowl. and Data Engin.*, 5(4), 551-563.

[13] Mackay, W. E. (1989). EVA: An experimental video annotator for symbolic analysis of video data. *SIGCHI Bulletin*, 21(2), 68-71.

[14] Nagasaka, A. and Tanaka, A. (1992). Automatic Video Indexing and Full-Video Search for Object Appearances. *Visual Database Systems, II* (E. Knuth and L.M. Wegner, Eds.). Elsevier Sci. Publ., 113-127.

[15] Oomoto, E. & Tanaka, K. (1993). OVID: Design and Implementation of a Video-Object Database System. *IEEE Trans on Knowl. and Data Engin.*, 5(4), 629-643.

[16] Roschelle, J., Pea, R., & Trigg, R. (1990). VIDEONOTER: A tool for exploratory analysis (Research Rep. No. IRL90-0021). Palo Alto, CA: Institute for Research on Learning.

[17] Santucci, G. & Sottile, P.A. (1993). Query by Diagram: a Visual Env. for Querying Databases. *Software—Practice and Exper.*, 23(3), 317-340.

[18] Snodgrass, R. (1987). The Temporal Query Language TQuel. *ACM Trans. on Database Sys., 12*(2), 247-298.

[19] Snodgrass, R. (1992). Temporal Databases. *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space* (A.U. Frank, I. Campari, and U. Formentini, Eds.), Springer-Verlag: New York, 22-64.

[20] Weber, K. & Poon, A. (1994). Marquee: A Tool for Real-Time Video Logging. *CHI'94 Conference Proceedings*: ACM Press, 58-64.

[21] Whang, K., et al. (1992). Two-Dimensional Specification of Universal Quantification in a Graphical Database Query Language, *IEEE Trans. on Software Engin.*, 18(3), 216-224.

[3] Due to space limitations in this paper, we only describe the temporal aspect of our proposed interface.