New Data Types and Operations to Support Geo-streams *

Yan Huang and Chengyang Zhang

University of North Texas
{huangyan,chengyang}@cs.unt.edu

Abstract. The volume of real-time streaming data produced by geo-referenced sensors and sensor networks is staggeringly large and growing rapidly. Queries on these geo-streams often require tracking spatio-temporal extent (e.g. evolving region) continuously in real time. The notion of real-time monitoring and notification requires support from a database capable of tracking and querying dynamic and transient spatio-temporal events as well as static spatial objects and sending out real-time notifications. In this paper, we leverage the work in data type based spatio-temporal databases and propose new data types called STREAM and their abstract semantics to support geo-stream applications. New operations on STREAM data types are defined and illustrated by embedding them into SQL.

1 Introduction

The advent of technologies in sensors and sensor networks is increasing human being's ability to interact with physical spaces in real time. Software tools that allow people to flexibly fuse, query, and make sense out of the data provided collectively by sensors in real time are very useful in many applications such as natural hazard monitoring, transportation, environmental modeling, and weather services.

Many sensors are geo-referenced and are excellent data sources for real-time situation monitoring and notification. The notion of real-time situation monitoring and notification requires support from a database capable of tracking and querying dynamic and transient spatio-temporal extents as well as static spatial objects and sending out real-time notifications. However, databases today lack this support.

In this paper, we deal with a special kind of data stream called geo-stream. A geo-stream refers to the moving and changing geometry of an object that is continuously fed into a geo-spatial data stream management system in real time. For example, the real time location of a moving object is a point geo-stream and the evolving spatial extent of a hurricane monitored in real time is a region

* This work was partially supported by the Texas Advanced Research Program under Grant No. 003594-0010-2006 and by the National Science Foundation under Grant No. CNS-0709285.

geo-stream. We give motivating examples to illustrate the benefits of having a geo-stream database management system.

Motivation Example (Hazard Weather Notification System) Hazard weather warnings and watches such as flood watches and tornado warnings are continuously produced by National Weather Services and posted to its website using text messages as well as visual graphs to represent the spatial extents of the warnings. These geo-streams can be combined with a static spatial database to answer the following queries:

Q1: Notify me when my house is within 50 miles of the mandatory evacuation area of a forest fire.
Q2: Continuously list the addresses of all the houses traversed by flood in the past 2 days in Denton county.
Q3: Continuously list road segments that have been completely under flood-water for the past 24 hours.

Without a geo-stream database management system, sophisticated programing will be required to repeatedly issue queries to a spatial database and combine the results to answer the above questions. There are several problems with this approach: (1) it is difficult to decide how frequently the program should issue the queries to the spatial databases; (2) it is computationally expensive to find the latest data from all the historical data each time when the query is issued; (3) an integrated query optimization can not be performed by the database system and kept transparent from the user.

Modeling data streams collected by sensors in real time as a database system has been proven to be successful over the past years [5, 19, 4, 3, 2]. Several data stream management systems (DSMS) have been developed with some of them leading to startup companies [18]. Similar to traditional database systems, a DSMS system supports a declarative language which allows a user to express queries in a statement like fashion. These queries are processed and results are returned by the DSMS without exposing the details of the physical data organization to the user. A DSMS uses various query optimization techniques to ensure the quality of service.

However, current DSMS systems only include very rudimentary support for simple point locations [14]. Very little has been done to support moving and evolving spatial objects in REAL time such as trajectories and evolving regions. Spatio-temporal predicates such as traversed area and their implications to query optimization have not been investigated. Furthermore, most DSMS systems employ a simple relational table based paradigm and treat an individual tuple as the basic unit of operation. However, a data type based approach [8] where an entity's spatial extent over time can be managed as a data type and participate in common spatio-temporal predicates may be semantically simple. The paper makes the following contributions:

– We propose several new streaming data types to a spatio-temporal database system to support geo-streaming applications;

– We propose to represent window semantics using data types as well;
– We investigate the semantics of common spatio-temporal predicates and operations, and propose new ones meaningful on the proposed streaming data types;
– We illustrate the embedding of the new data types, operations, and predicates to a common database query language, i.e. SQL.

2 Related Work

Related work can be classified into four categories: spatio-temporal databases, moving object databases, streaming data management systems, and sensor network databases.

Our work is closely related to spatio-temporal databases [8, 11, 17, 12, 9] and to a data type based spatio-temporal database approach [8, 11, 12] in particular. This approach focuses on designing spatio-temporal data types, operations, and predicates and embedding them into a query language to support the storage and query of spatio-temporal data. The design goal of this approach is to find a set of data types and operations that are succinct, representative, consistent, and self-closed to support many applications in spatio-temporal domains. Spatial data types, e.g. points, lines, and polygons, their abstract semantics, and their discrete implementations in a computer system have been proposed. Spatial predicates and operations have been precisely defined. Together with non-spatial data types, e.g. string, spatial data types have been associated with time and elevated into temporal data types. Spatial predicates and operations have also been lifted. Spatio-temporal query languages that have SQL-like syntax have also been developed to embed the spatio-temporal data types and operations. However, to the best of our knowledge, supporting geo-streams in real time has not been addressed in this line of work. Instead, most assume a complete view of the spatio-temporal phenomena without considering their streaming nature. Prototype spatio-temporal database systems, such as Concert [17], Secondo [12], and Dedale [9], have been built with various levels of support for spatio-temporal data.

Moving object databases handle streaming location updates and support queries on them. They can be seen as streaming point data for *now* window in our proposed framework in most cases. There is a large body of work on indexing and query optimization on moving objects [13]. The indexing methods can be classified into two groups: disk based and memory based [16, 15]. For volatile and dynamic objects, the maintenance cost of a disk based indexing could be prohibitive and memory based kinetic data structures are a better choice. Many query processing algorithms have been proposed for window queries and nearest neighbor queries. However, little has been done to support a suite of spatial streaming data types and their operations which is the focus of this work.

In a stream data management system, data arrives in the form of concurrent and continuous data streams. Queries on these data streams are typically continuous monitoring queries, involving both persistent relations and other time-varying streams, and emitting streaming data in real time as results. Data stream

management systems have several significant characteristics different from those of traditional transactional and decision support systems. First, streams are generated on a regular, irregular, or bursty basis from many sources. A large number of streams from multiple sources and large volumes of data emitted from each source challenge traditional persistent relation and input/output reduction-based query processing paradigms. Second, stream-based applications require real-time response from the query processing system in order to trigger further actions despite bursty and unpredicted system loads. Third, queries on data streams involve not only data streams but also traditional relations. This requires an integrated model on both relations and data streams. Furthermore, queries on data streams are typically continuous queries, which require an intuitive, and semantically simple and clear query language/interface to specify queries and incorporate time window semantics. New processing paradigms and methods have been proposed and implemented in several stream processing systems [5, 19, 4, 3, 2] to achieve similar objectives. However, they can only handle streaming point locations naively [14] and do not have adequate support for evolving spatio-temporal extents. Efforts in developing data stream query languages ignore the support for dynamic and evolving geo-spatial objects from geo-streams.

Work in sensor (mote sized) databases [1, 6, 7] is focused more on reducing energy consumption and hiding the inherent heterogeneity and unreliability of sensor networks. Data input is modelled as relations instead of data types. In terms of spatio-temporal data and operation support, most sensors are static and location queries are limited to temporal range queries, e.g. find the average temperature in a given window. Some recent work addresses the in-network processing of spatial join of readings from sensors in spatial proximity [20]. We argue that treating individual and discrete sensor readings as the unit of operation is not sufficient. We need to support construction of spatio-temporal phenomena from raw sensor readings. For example, identifying soaked *regions* from distributed soil moisture sensors; and monitoring and tracking of the regions would be more useful than obtaining individual sensor readings. While abstracting spatio-temporal phenomena from raw sensor readings in real time is out of the scope of this paper and will be addressed in our future work, supporting data types such as moving regions as well as operations on them will be very useful.

In summary, current database systems DO NOT have a comprehensive set of spatial streaming data types and operations to support continuous queries involving both geo-streams and static geo-spatial objects.

Thus, the goal of this paper is to develop streaming data types, operations, predicates, and their language embeddings that allow a user to express continuous queries on both static and dynamic objects and monitor sensed phenomena in real-time. Towards this goal, we resolve two important research issues: (1) the definition of window semantics through a data type based approach; (2) the design of streaming data types, operations, and predicates under the proposed window semantics and the closure and consistency of the data types.

3 Streaming Data Types

For representing spatio-temporal objects and operations, many-typed algebra has been developed to ensure the closure of the operations among data types. Because it also supports both static spatial and historical spatio-temporal data, a geo-stream database system should also support the basic data types available in these systems. We extend the work presented in [8, 11, 12] to represent the data types and operations. The data types marked by a ∘ at the end represent traditional data types in a non-streaming spatio-temporal database system:

	→ BASE	*int, real, string, bool*	∘
	→ SPATIAL	*point, points, line, region*	∘
	→ TIME	*instant*	∘
	→ WINDOW	*now, unbounded, past*	*
BASE ∪ TIME	→ RANGE	*range*	∘
BASE ∪ SPATIAL	→ TEMPORAL	*intime, moving*	∘
(BASE ∪ SPATIAL ∪ RANGE) × WINDOW	→ STREAM	*streaming*	*

In this notation, the abstract semantics of each data type is defined by its carrier set denoted by A_α where α is the data type. For example, the semantics of a point is represented by $A_{point} \equiv \{\mathbb{R}^2\} \cup \{\perp\}$ where $\perp$ represents the undefined value and the semantics of a point set is represented by $A_{points} \equiv \{P \subseteq \mathbb{R}^2 | P\, is\, finite\}$.

The detailed abstract semantics of BASE, SPATIAL, TIME, RANGE, and TEMPORAL data types have been defined elsewhere [8, 11, 12]. Briefly, the *line* data type is defined as a collection of simple curves (intersection of curves yields a finite number of intersection points). The *region* type is a collection of *faces* where a *face* is a set of points divided into interior, boundary, and exterior parts. The *instant* type is used to represent time and is isomorphic to the real numbers. Range is defined on BASE and TIME data types if a total order exists. A range has a pair of starting and ending values and includes all the values in between. The *intime* data type applied to a BASE or SPATIAL type α associates a time stamp with α. The *moving* data type applied to a BASE or SPATIAL data type α is a mapping from time to α. Please also note that all of the data types include an undefined value represented by $\perp$.

We assume that the abstract meaning of non-stream data types are properly defined and will focus on the new types namely WINDOW and STREAM in this paper.

3.1 WINDOW Types

In a data type based approach, the moving observation windows of a data stream are represented as data types. These data types can later be applied to BASE, SPATIAL, and RANGE data types to construct new streaming data types. We first define the abstract semantics of a WINDOW type that includes *now, unbounded*, and *past*.

At any time, the carrier of the type constructor *now* only has one value obtained from the system. Thus, $A_{now} \equiv$ current system time. We use *now* to refer to this value for simplicity.

At any time, an *unbounded* time window denotes the time interval starting from the system initial time until *now*. Thus, $A_{unbounded} \equiv (-\infty, now]$.

At any time, a *past* window specifies a time interval proceeding *now* and the semantics is defined by: $A_{past} \equiv \{X \subseteq A_{instant} | \forall x \in X(x \leq now) \wedge \forall y \in A_{instant}(x < y \leq now \Rightarrow y \in X)\}$.

3.2 STREAM Types

For a given BASE, SPATIAL, or RANGE type α and a WINDOW type ω, the type constructor *streaming* yields a mapping: $A_{streaming(\alpha,\omega)} \equiv \{f^c | f^c : A_{instant} \rightarrow A_\alpha$ is a partial function that is undefined for instants not in the window specified by ω and is continuously updated over time as the window changes according to the semantics of window type $\omega\}$.

For the *now* window and the *point* data type, $streaming(point, now)$ is a streaming pair consisting of a point location and the current time. A new pair replaces the old pair continuously over time. Thus, $A_{streaming(point,now)} \equiv \{f^c | f^c : A_{now} \rightarrow A_{point}$ where $\forall t \neq now, f^c$ is undefined $\}$.

For the *unbounded* window and the *point* data type, $streaming(point, unbounded)$ is an ever increasing mapping from any time instant in the past to some point in space. Specifically, $A_{streaming(point,unbounded)} \equiv \{f^c | f^c : A_{instant} \rightarrow A_{point}$ where $\forall t \in (now, \infty), f^c$ is undefined $\}$.

For the *past* window and the *point* data type, $streaming(point, past)$ is a "streaming" mapping from any time instant in the time interval $(now - i, now]$ to some point in space where i is a time interval that can be optionally specified by the *past* data type. This is similar to the array data type in a programming language where the size of the array can be either specified at data type construction or left to the system. Let us assign the current system time *now* to a constant C at some time once. After time δ, the mapping for the interval $(C - i, C - i + \delta]$ becomes undefined and the mapping for $(C, C + \delta]$ is added (note that $C + \delta$ is the current value of *now*). Thus, $A_{streaming(point,past)} \equiv \{f^c | f^c : A_{instant} \rightarrow A_{point}$ where $\forall t \notin (now - i, now], f^c$ is undefined where i is a time interval$\}$.

We also assume if a window is not provided in the type constructor, the default is the *now* window. The abstract definitions for other streaming data types, e.g. *real* or *region*, can be defined in the similar manner. For all *streaming* types, we use the following naming convention: prefixing the BASIC or SPATIAL type with an *s* to represent its streaming version, e.g. *sint, sreal, sstring, sbool, spoint, spoints, sline, srange* and *sregion*.

4 Geo-stream Operations

Now the work boils down to the design of spatio-temporal operations and predicates under the window semantics. The desired properties of the operations are:

meaningful so that they are useful, representative so that we do not have a cluttered predicate set, and general so that they can be used for many applications.

4.1 Windowing Operations on Streams

Streams can be formed from streams of different windows. For example, from a stream of an *unbounded* window, a stream of *now* can be formed. From a stream of *now*, a stream of the *past 2 hours* can also be formed although initially the two hours window is not complete. Thus, the operation **windowing:** $streaming(\alpha, \omega) \times \psi \rightarrow streaming(\alpha, \psi)$ maps one stream of window ω to a stream of the same data type α of another window ψ. To be consistent with window operations in general stream processing systems, we use the notion $streaming(\alpha, \omega)[\psi] \rightarrow streaming(\alpha, \psi)$ to represent the *windowing* operation in our language embedding later.

4.2 Projection to Domain and Range

Operations that apply on TEMPORAL data types and yield domain and range were provided in a spatio-temporal database. For example, the domain function **deftime**: $moving(\alpha) \rightarrow periods$ (*periods* is a shorthand of the range of time $range(instant)$) returns the time intervals in which a function is defined. The **rangvalues**:$moving(\alpha) \rightarrow range(\alpha)$ operation returns values assumed over time as a set of intervals for a BASE type α in 1D space. Other operations in this category include *trajectory* which projects a moving object into lines in space and *traversed* which projects an evolving region into space and aggregates all projected regions into a larger region.

The projection to domain and range for streaming data types and functions have similar semantics except that the projection is continuously re-evaluated over the moving windows and the result is of streaming data types.

For example, the semantics of projection of a streaming region with window *unbounded* is to restrict the time to the window *unbounded* and apply the projection up-to *now*. This process is continuously applied as the window moves over time. In 1D space, the operation **rangevalues:** $streaming(\alpha, \omega) \rightarrow streaming(range(\alpha), now)$ for a BASE type α and a WINDOW type ω returns the values α assumed over streaming window ω for the current time *now*. For 2D space, streaming version of the operations such as *trajectory* and *traversed* can be defined. For example, the streaming operation **trajectory:** $streaming(\alpha, \omega) \rightarrow streaming(\alpha+, now)$ aggregates/elevates a set of streaming points of window ω to a high level object lines to represent a trajectory for the streaming window *now*.

Now suppose we are tracking caribous by satellite radio collars and the satellite collars turn on for some hours every day. The locations of the caribous are streamed back and modeled as caribou(name *string*, location *spoint*). Please be reminded that by default, the location of type *spoint* is of *now* window. And applying windowing operation on *location* is represented by $location[\psi]$ where ψ is a new window. The operation **deftime(caribou.location[unbounded])**

Operations	Signature
atinstant	$streaming(\alpha, \omega) \times instant \rightarrow intime(\alpha)$
atperiods	$streaming(\alpha, \omega) \times periods \rightarrow moving(\alpha)$
present	$streaming(\alpha, \omega) \rightarrow streaming(bool, \omega)$
at	$streaming(\alpha, \omega) \times \alpha \rightarrow streaming(\alpha, \omega)$
at	$streaming(\alpha, \omega) \times range(\alpha) \rightarrow streaming(\alpha, \omega)$
passes	$streaming(\alpha, \omega) \times \beta \rightarrow streaming(bool, now)$

Table 1. Interaction of Streaming Values with Values in Domain and Range

(note: *now* can be omitted since the default window of the location is *now*) returns the times when the caribou is tracked and the result is updated continuously. The operation **deftime(caribou.location[now])** returns the current time if the caribou is tracked now and $\perp$ otherwise. The operation **trajectory(caribou.location[past 2 hours])** returns the trajectories (represented by lines) of the locations of caribous projected to space in the past two hours continuously.

4.3 Intersection with Points and Point Sets in Domain and Range

Many queries need to use operations that relate the function values to their time domain or range. For TEMPORAL types, operations such as **initial** and **final** are useful to find the spatial extent at starting and ending times. For streaming applications, the starting and ending time of an event are difficult to determine. One solution is to ask users to explicitly associate each object of a streaming spatio-temporal extent with an **initialtime** and **finaltime**. However, these two attributes will not be part of the spatio-temporal or streaming data types and do not participate in spatio-temporal or streaming operations.

The **atinstant** and **atperiods** operation for TEMPORAL data type returns the spatial extent of the object for a given instant or a range of time. They may be useful for streaming data types if one wants to "intercept" the function values of a streaming data type at a specific time or time period. The result will be a constant data type, not a stream.

The **present** operation for a streaming data type α of window ω checks if the value of α exists during the time interval specified by the moving time window and returns a *sbool* data type of the same window. The **at** operation restricts the streaming data type's range to specific values. When the value of ranges is out of the given range, the operation returns an undefined value. For example, we can restrict the streaming real to the times when its values is between 0 and 1. The operation **passes** checks if the streaming value ever assumed the given values over the moving window. Table 4.3 summarizes the operations meaningful for streaming data types.

For the sample table of caribou, if we want to track for once where the caribous are for a given time t, we can use **atinstant(caribou.location[now], t)**.

Once the time has passed t, this operation will not be useful anymore. The operation **present(caribou.location[past 3 days])** tells continuously at what times in the past three days the caribous are tracked. Let city(name *string*, location *point*) be a table storing the cities and their point locations. The operation **at(caribou.location[past 7 days],city.location)** returns continuously the caribous who had passed a city in the past 7 days.

4.4 Lifting Operations to Streaming Operations

For a spatial predicate or operation, its streaming version can be defined by converting one or more of the participating data types into its streaming version. The semantics of such a straightforward conversion will need to be investigated. For example, the spatial predicate **intersect** with signature $region \times region \rightarrow bool$ can be first lifted into three streaming versions of **intersect** (we will discuss time shifted streaming operation shortly):

1. $streaming(region, \omega) \times region \rightarrow streaming(bool, \omega)$
2. $region \times streaming(region, \omega) \rightarrow streaming(bool, \omega)$
3. $streaming(region, \omega) \times streaming(region, \omega) \rightarrow streaming(bool, \omega)$

For **intersect** case 1, the predicate is to check if a streaming region intersects with a static region for each possible time instant in the valid moving window specified by ω and will produce a streaming *bool* as the result for the same moving window. Please note that the result is a streaming bool instead of a single TRUE or FALSE value. We further assume, when all the values in the given window ω is TRUE, the value of the result is TRUE. Similarily, the result is FALSE when all the values in the given window ω is FALSE. Otherwise, the result is unknown and further operations such as **at** or **duration** will be needed if a TRUE or FALSE is desired as the result which will be shown shortly using an example.

For **intersect** case 2, since **intersect** is a symmetric predicate, the semantics will be the same as that of case 1. For asymmetric operations and predicates, the semantics can be inferred straightforwardly from the static versions of the predicates. For **intersect** case 3, it returns a streaming bool of window ω representing the intersect results of two streaming regions of the same streaming window ω at each instant in the window. The lifting of other spatial predicates can be defined in the similar manner.

In addition, if we allow the time shifted (specified by an *instant* data type representing the time shift) **intersect**, the predicate can be more meaningful. For example, one may want to continuously see if the zones of high ozone intersect with the zones of high ozone for the same time last year. A time shifted **intersect** can be represented as:

$$streaming(region, \omega) \times moving(region) \times instant \rightarrow streaming(bool, \omega)$$

where the moving region represents a moving and evolving region (usually in the past). Similarily, operations between two streaming data types can be parameterized with a time shift as well.

However, instead of adding a time shift parameter to each operation, it would be more flexible and elegant to introduce a separate time shift operator on windows which means the WINDOW types need to be enhanced with time shifted windows in addition to the three basic ones introduced in this paper. As a result, all operations involving windows will need to be revisited to include time shifted windows. We plan to explore this alternative in our work in the near future.

Now we give several examples to illustrate these streaming operations. Let hurricane(name *string*, extent *sregion*) be a table of hurricanes with streaming spatial extent. Let county(name *string*, extent *region*) be a table of counties. Then the following statement continuously emits the names of the counties that intersect with hurricane K any time during the past 2 hours:

```
SELECT c.name
FROM   hurricane h, county  c
WHERE  duration(at(intersect(h.extent[past 2 hours], c.extent), TRUE)) <> 0
       AND h.name = K
```

Please note that the **intersect** operation returns a streaming bool type for the past two hours and is continuously updated. The **at** operation returns the time intervals where the value of intersect is TRUE. Then the duration (it is a shorthand for **rangevalues** of time) returns the duration of the time when the intersect operation is true. If this duration is not 0, then the county name is returned as it intersects with hurricane K some time during the past 2 hours.

Now let us look at the lifting of another spatial predicate **intersection** which returns the intersection part of two regions. The following statement returns the county name together with the intersection of the extent of the county and the current extent of the hurricane K continuously for all counties that intersect with hurricane K:

```
SELECT c.name, intersection(h.extent[now], c.extent)
FROM   hurricane h, county  c
WHERE  intersect(h.extent[now], c.extent) AND h.name = K
```

For a streaming data type and a temporal data type, the spatial predicate is applied to a pair of spatial extents from the two data types respectively with a time shift. For example, let hurricaneA(name *string*, extent *mregion*) be the archived hurricane information for the past years where *mregion* represents a moving region data type. The following statement is used to retrieve the names of the hurricanes whose current spatial extent has at least 80% overlap with a hurricane at the same time a year ago:

```
SELECT h.name
FROM   hurricane h, hurricaneA ha
WHERE  area(intersection(h.extent[now], ha.extent, 1 year))
       >80% * area(h.extent[now])
```

The operation between two streaming data types with a time shift can be illustrated by the following example. Let heavyRain(name *string*, extent *sregion*) be a table representing the heavy rain regions monitored in real time.

Let flashFlood(name *string*, extent *sregion*) be a table representing flash flood. Then we can use the following statement to find the flash floods with heavy rain proceeding them two hours before:

```
SELECT ff.name
FROM   flashFlood ff, heavyRain hr
WHERE  duration(at(intersect(ff.extent[past 2 hours], hr.extent[past 4 hours], 2 hours)) <> 0
```

In the above statement, the **intersect** statement is applied between the extent of a flash flood and the extent of a heavy rain two hours before.

5 Language Embedding of the STREAM Data Type

We have already seen several examples of language embedding of the STREAM data type and operations into SQL. To summarize, the stream data type is typically associated with an object. The object has static properties, e.g. name, and is modeled as a tuple in a table. In the FROM clause of a SQL statement, a table with streaming attribute can participate. The combination of stream operations among streaming attributes of the tables that result in a boolean TRUE or FALSE can appear in the WHERE clause. Objects that satisfy all the predicates including the streaming predicates in the WHERE clause will be returned as the results. All streaming operations and their combinations can appear in the SELECT clause and the results are subject to the condition that the associated objects satisfy the predicates in the WHERE clause. The following statement summarizes this paragraph.

```
SELECT STREAM operation, ...
FROM   table with streaming attribute, ...
WHERE  STREAM predicate, ...
```

To answer the three queries introduced in the introduction of the paper, we use the following static and dynamic spatial objects:

forestFire(firename: *string*, evaArea: *sregion*)
flood(floodname: *string*, extent: *sregion*)
house(owner: *string*, location: *point*)
road(roadname: *string*, extent: *line*)

The first query (**Q1**) regarding houses currently in the mandatory evacuation area of a forest fire may be answered with the following statement:

```
SELECT h.address
FROM   house h, forestFire ff
WHERE  distance(h.location,ff.evaArea [now])<50;
```

The second query (**Q2**) regarding houses traversed by a flood in the past 2 days may be answered with the following statement:

```
SELECT h.address
FROM   house h, flood f
WHERE  inside(h.location,traversed(f.extent [past 2 days]))
```

The third query (**Q3**) regarding roads immersed in flood for more than the past 24 hours may be answered by:

```
SELECT r.name
FROM   road r, flood f
WHERE  duration(deftime(at(inside(r.extent,f.extent [past 1 day]),TRUE)))=1 day
```

6 Conclusion and Future Work

In this paper, we proposed new data types and operations to support geo-streaming applications. We discussed their semantics and their embedding into a database language such as SQL. The proposed system is compatible with a data type based spatio-temporal database system. The system can be used to support continuous real-time queries on evolving spatio-temporal extents through the combination of these new data types and operations with SQL. Many real time monitoring applications will benefit from such a system.

This work is just the first step towards a full-fledged geo-stream database management system. We plan to first investigate the discrete representations for each new stream data type. For example, efficient data models and structures are needed to represent a streaming point in the past two hours.

Work also needs to be done in choosing efficient data structures and algorithms for query optimization. Please note that such queries may involve multiple geo-streams or a combination of geo-streams and spatio-temporal data types. For example, in streaming applications, indexing schemes are less sufficient since the cost of building the indexes and maintaining them will be high. Kinetic data structures deal with the fundamental geometric problems such as the maintenance of a convex hull from dynamic data. Algorithms were proposed to maintain geometrics and kinetic properties such as the closest pairs (to avoid collision) and minimal spanning tree (to maintain connections among moving sensors). In many cases, memory-based-kinetic data structures [10] will need to be considered for geo-stream applications instead.

Acknowledgment: We thank the anonymous reviwers of GIScience for their valuable insights, suggestions, and very detailed comments and corrections on this paper.

References

1. Tinydb. `http://telegraph.cs.berkeley.edu/tinydb/`.
2. D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

3. M. H. Ali, W. G. Aref, R. Bose, A. K. Elmagarmid, A. Helal, I. Kamel, and M. F. Mokbel. Nile-pdt: a phenomenon detection and tracking framework for data stream management systems. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1295–1298. VLDB Endowment, 2005.
4. S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: continuous dataflow processing. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668. ACM, 2003.
5. J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq: a scalable continuous query system for internet databases. *SIGMOD Rec.*, 29(2), 2000.
6. J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of the 20th International Conference on Data Engineering*, 2004.
7. A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of VLDB*, pages 588–599, 2004.
8. L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 319–330, New York, NY, USA, 2000. ACM.
9. S. Grumbach, P. Rigaux, and L. Segoufin. The dedale system for complex spatial queries. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 213–224. ACM, 1998.
10. L. Guibas. Kinetic data structures: A state of the art report. In *The 3rd Workshop on Algorithmic Foundations of Robotics*, 1998.
11. R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.
12. R. H. Güting, V. T. de Almeida, D. Ansorge, T. Behr, Z. Ding, T. Höse, F. Hoffmann, M. Spiekermann, and U. Telle. Secondo: An extensible dbms platform for research prototyping and teaching. In *ICDE'05: Proceedings of the 21st International Conference on Data Engineering*, pages 1115–1116. IEEE Computer Society, 2005.
13. R. H. Guting and M. Schneider. *Moving Objects Databases* . Morgan Kaufmann, 2005.
14. N. Jain, L. Amini, H. Andrade, R. King, Y. Park, P. Selo, and C. Venkatramani. Design, implementation, and evaluation of the linear road bnchmark on the stream processing core. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 431–442. ACM, 2006.
15. M. F. Mokbel and W. G. Aref. Sole: scalable on-line execution of continuous queries on spatio-temporal data streams. *VLDB Journal*, accepted for publication, 2008.
16. M. F. Mokbel, X. Xiong, and W. G. Aref. Sina: scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 623–634, 2004.
17. L. Relly and U. Röhm. Plug and play: Interoperability in concert. In *INTEROP '99: Proceedings of the Second International Conference on Interoperating Geographic Information Systems*, pages 277–291. Springer-Verlag, 1999.
18. S. Systems. StreamBase Server. `http://www.streambase.com/`.

19. P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):555–568, 2003.
20. M. L. Yiu, N. Mamoulis, and S. Bakiras. Retrieval of spatial join pattern instances from sensor networks. In *SSDBM*, page 25, 2007.