SQLST: A Spatio-Temporal Data Model and Query Language

Cindy Xinmin Chen and Carlo Zaniolo

Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90095, U.S.A.
{cchen,zaniolo}@cs.ucla.edu

Abstract. In this paper, we propose a query language and data model for spatio-temporal information, including objects of time-changing geometry. Our objective is to minimize the extensions required in SQL, or other relational languages, to support spatio-temporal queries. We build on the model proposed by Worboys where each state of a spatial object is captured as a snapshot of time; then, we use a directed-triangulation model to represent spatial data, and a point-based model to represent time at the conceptual level. Spatio-temporal reasoning and queries can be fully expressed with no new constructs, but user-defined aggregates, such as AREA and INSIDE for spatial relationships, DURATION and CONTAIN for temporal ones, and MOVING_DISTANCE for spatio-temporal ones. We also consider the implementation problem under the assumption that, for performance reasons, the representation at the physical level can be totally different from the conceptual one. Thus, alternative physical representations and mappings between conceptual and physical representations are discussed.

1 Introduction

Spatio-temporal data models and query languages have received much attention in the database research community because of their practical importance and the interesting technical challenges they pose. Because of space limitation, we will only discuss previous research that most influenced our approach.

Much previous work focuses on either temporal information or spatial information, rather than both. For instance in the temporal domain, interval-based time models [17] were followed by TSQL2's implicit-time model [21], and point-based time models [22, 23]. SQL extensions to express spatial queries were proposed by several authors, including [8] and [9].

In a seminal paper, Worboys [25] defines a spatio-temporal object as a unified object which has both spatial and temporal extents, and is represented by attaching a temporal element to the components of a collection of non-overlapping spatial objects that include points, straight line segments and triangular areas. This model is extended in [5] to allow the vertices of triangles to be linear functions of time. The constraint database framework in [16] is used to characterize the expressiveness of the data model.

In [12] and [13], a spatio-temporal data model also based on linear constraints is proposed. The model restricts the orthographic dimension of an object, then it processes queries independently on each dimension of the components. A $d-$dimensional object is stored as constraints on d variables, with an upper bound on the number of variables that can occur in a single constraint. A spatio-temporal query language based on this model will have to add many new constructs to existing query languages such as SQL.

A topic of growing interest is that of modeling and storing moving objects. A framework for specifying spatio-temporal objects is presented in [6]. The paper defines a number of classes of spatio-temporal objects and studies their closure properties. However, implementation requires the database to store functions as tuple components (function objects). In [4], a model based on parametric rectangles is proposed for representing spatio-temporal objects, where the vertices of triangles are linear functions of time. In [10], a design of moving points and moving regions is discussed, which focuses on generality, closure and consistency, but only discusses the abstract level of modeling. The approach discussed in [10] introduces new spatio-temporal data types, such as *mregion* and *mpoint*, for moving region and moving points, respectively.

In this paper, we propose a minimalist's solution to the problem of supporting spatio-temporal data models and queries in databases, insofar as we want to achieve the desired functionality and expressive power with minimal extensions to current SQL3 standards. Indeed, we want to minimize the effort needed to implement spatio-temporal extensions on the new generation of object relational databases, and thus facilitate the incorporation of such extensions into commercial systems and SQL standards. Toward this goal, we apply several lessons learned with SQL^{T}, where we were able to support valid-time queries by minimal extensions of SQL [7]—specifically, by extensions that could be supported in current Object-Relational systems with the help of user-defined functions and aggregates [15].

Thus our paper is organized as follows. In the next section we give an overview of our data model SQLST. Then, in Sections 3 and 4, we introduce the temporal, spatial and spatio-temporal operators supported in SQLST in a way that is conducive to their implementation through user-defined aggregates or user-defined functions. Then, in Section 5, we show how these operators are used to provide a simple and expressive formulation for complex spatio-temporal queries. The final two sections discuss implementation issues and the opportunities for further research.

2 The Data Model of SQLST

Many applications only use temporal information, other only use spatial information, finally many use both. Therefore, following Worboys' suggestion [25], we define an SQL^{T} component that is effective at supporting temporal information, and a SQL^{S} component that is effective at supporting spatial information; then, we combine the two representations into SQLST and show that this is effective in supporting spatio-temporal information, including two-dimensional spatial objects whose geometry and shape change with time.

To model time at the conceptual level, we use a point-based time model [22], where information is repeated for each time granule where it is valid, thus eliminating the need for temporal coalescing that besets interval-based approaches [17]. Moreover, temporal aggregates can easily express interval operators in a point-based model [7].

A point-based representation of two-dimensional spatial objects was initially considered for SQL^S, but a polygon-oriented representation was finally selected for two reasons. One is that coalescing is needed much less frequently than in temporal queries. The second is that two dimensional shapes offer a more natural representation for many application domains. For instance a region in a GIS system can be drawn, enlarged, moved, or split; it is hard to associate these behaviors and operations with the points in the region. On the other hand, temporal intervals do not represent any concrete application object; in fact, an interval-based representation is often less appealing than a point-based representation (that models snapshots of reality), or an event-based representation (that models transitions between two successive states of reality). Therefore, SQLST views reality as a sequence of snapshots of objects that are moving and/or changing in shape.

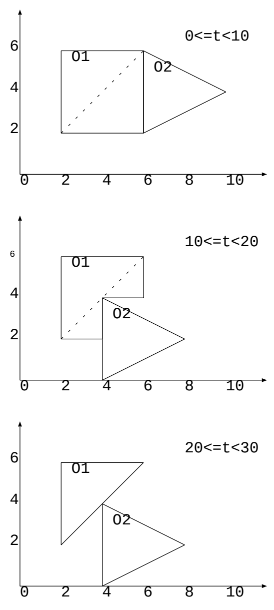

Fig. 1. Graphs representing spatio-temporal data

Figure 1 is an example of spatial objects changing with time. At time $t = 0$, there are two spatial objects in the graph, a square $O1$ and a triangle $O2$. At time $t = 10$, $O1$ changes its shape and $O2$ is moved to a new position. At time $t = 20$, $O1$ has some more changes in shape while $O2$ stays unchanged.

An internal representation of the spatio-temporal objects shown in Figure 1 could be as follows:

$(O1,\ [(2,6),(2,2),(6,2),(6,6)],[0,10))$
$(O2,\ [(6,6),(6,2),(10,4)],[0,10))$
$(O1,\ [(2,6),(2,2),(4,2),(4,4),(6,4),(6,6)],$
$[10,20))$
$(O2,\ [(4,4),(4,0),(8,2)],[10,20))$
$(O1,\ [(2,6),(2,2),(6,6)],[20,30))$
$(O2,\ [(4,4),(4,0),(8,2)],[20,30))$

Here, the regions are represented by a circular list of vertexes, and the time elements are stored as intervals. Mapping method between internal representation and the conceptual model will be discussed in Section 6.

Table 1 shows how the changes are recorded in the database at the conceptual level. From time $t = 0$ on, the square $O1$ is represented by two triangles and the triangle $O2$ is represented by one triangle. At time $t = 10$, changes of the shape of $O1$ and position of $O2$ occur and their representation also change accordingly. Now $O1$ is represented by three triangles while $O2$ is still represented by one triangle with changed coordinates of the vertexes. This representation is valid from time $t = 10$ till further change occurs. At time $t = 20$, the shape of $O1$ is changed further while $O2$ remains unchanged. Now $O1$ is represented by one triangle and $O2$ stays unchanged.

In Table 1, we also notice that the valid time of each fact is recorded as a time instant `VTime`.

ID	x_1	y_1	x_2	y_2	x_3	y_3	VTime
$O1$	6	2	2	2	6	6	0
...	...	...	...	...	...	...	...
$O1$	6	2	2	2	6	6	9
$O1$	6	6	2	2	2	6	0
...	...	...	...	...	...	...	...
$O1$	6	6	2	2	2	6	9
$O2$	6	6	6	2	10	4	0
...	...	...	...	...	...	...	...
$O2$	6	6	6	2	10	4	9
$O1$	6	2	2	2	6	6	10
...	...	...	...	...	...	...	...
$O1$	6	2	2	2	6	6	19
$O1$	6	6	4	4	6	4	10
...	...	...	...	...	...	...	...
$O1$	6	6	4	4	6	4	19
$O1$	4	4	2	2	4	2	10
...	...	...	...	...	...	...	...
$O1$	4	4	2	2	4	2	19
$O2$	4	4	4	0	8	2	10
...	...	...	...	...	...	...	...
$O2$	4	4	4	0	8	2	19
$O1$	6	2	2	2	6	6	20
...	...	...	...	...	...	...	...
$O1$	6	2	2	2	6	6	29
$O2$	4	4	4	0	8	2	20
...	...	...	...	...	...	...	...
$O2$	4	4	4	0	8	2	29

Table 1. Conceptual model of the spatio-temporal data shown in Figure 1

In our spatial model we use triangles to represent polygons; a similar approach was proposed in [11, 18]. A polygon having n vertexes can be decomposed into $n - 2$ triangles in $O(n \log n)$ TIME. Decomposing a polygon into a set of triangles makes determine spatial relationships between two polygons easy to do. We extended the triangulation method to use sets of directed triangles to represent polygons at the conceptual level. The three edges of a triangle are directed lines and form a counterclockwise circle. The directed triangulation method not only makes testing whether a point is inside a triangle need fewer calculations than the method proposed in [18] but also can handle holes in polygons.

We use user-defined aggregates [24] to support spatial operators such as area, inside, etc., temporal operators such as duration, overlap, etc., and spatio-temporal operators such as moving_distance, etc.

3 Temporal Operators

As has been discussed in [7, 15], an important requirement of all temporal languages is to support Allen's interval operators such as overlap, precede, contain, equal, meet, and intersect [1]. Figure 2 shows the meaning of these operators.

Temporal languages that are based on temporal intervals [17] use the overlap operator to express temporal joins. In a point-based temporal model, no explicit use of overlap is needed since two intervals overlap if and only if they share

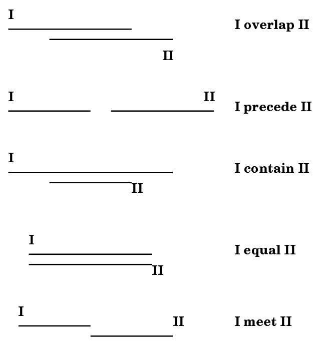

Fig. 2. Allen's interval operators

some common points [2, 22]. Moreover, since intervals are sets of contiguous points, set aggregates should be used to support interval-oriented reasoning [7]. For instance the duration operator that compute the total time span of a set of intervals is in fact equivalent to a count of time points (but a new name is used in SQL^T to improve its intuitive appeal and to allow its direct and more efficient implementation).

For example, consider the following SQL^T relation:

Example 1. Define the employ relation

```
CREATE TABLE employ
        (name CHAR(30), title CHAR(30), salary DECIMAL(2),
        mgrname CHAR(256), VTime DATE)
```

Then, a query such as: "find employees who for a time had the same manager as Melanie, and show their history during such time" can be expressed as follows:

Example 2. Employees who had the same manager as Melanie and their history during such time.

```
SELECT E2.name, E2.title, E2.VTime
FROM  employ AS E1 E2
WHERE E1.name = "Melanie"
    AND E1.mgrname = E2.mgrname AND E2.name <> "Melanie"
    AND E1.VTime = E2.VTime
```

This example illustrates that project-select-join queries can be expressed in a very natural fashion in SQL^T. The "same time" notion is simply expressed as the equality of the *points in time* in which E1 and E2 worked for the same manager. The same notion in an interval-oriented language would be expressed by a condition stating that the time intervals in which E1 and E2 worked for a same manager overlap. Supporters of interval-based approaches would hereby point that the notion of overlapping periods is quite intuitive; however, detractors would point out that the coalescing issue limits the appeal of this approach. For instance if we modify Example 2 by dropping E.title from the SELECT clause, then histories of employees who had a change of title while working under Melanie's manager must be consolidated by coalescing their intervals into larger ones. Example 3 below, also discusses this important issue.

The desire of avoid coalescing was a motivation in *implicit valid time* approach followed by TSQL2. In TSQL2, there would be no "**VTime** DATE" column in Example 1, and an annotation will be used instead to denote that this is a valid-time relation, rather than an ordinary non-temporal relation. Likewise, there is no valid time attribute in TSQL2 queries; in fact if we take Example 2 and drop E2.VTime from the SELECT clause, and E1.VTime = E2.VTime from the WHERE clause, we obtain a correct TSQL2 formulation for our query. Therefore, "at the same time" becomes the default interpretation for all queries in TSQL2, and the coalescing operations needed to support this interpretation are implicitly derived by the system.

Consider now a query such as: "Find all the positions held by Melanie for more than 90 days." This can be expressed in SQL^T as follows:

Example 3. Melanie's positions of more than 90 consecutive days.

```
SELECT title
FROM employ
WHERE name = "Melanie"
GROUP BY title
HAVING DURATION(VTime) > 90
```

The fact that the time span of each position must be computed irrespective of the change of manager is expressed naturally and explicitly by the group-by qualifications attached to the new SQL^T aggregate duration. In fact, all period oriented operators, such as contain and precede from Allen's interval algebra, can be expressed naturally by new temporal aggregates. Rather than relying on constructs such as group-bys and aggregates already in SQL, TSQL2 introduces two new constructs to express the same query: one is the "snapshot" annotation in the "select" clause, and the other is the restructuring construct in the "from" clause. Furthermore, TSQL2 constructs cannot be extended to languages such as QBE or Datalog that do not have select clauses and from clauses— a lack of robustness called "lack of universality" in [7]. More sample queries of temporal queries using user-defined aggregates can be found in [7, 15]

An efficient implementation for SQL^T called TEnORs (for Time Enhanced Object Relational system) is available for DB2 [15]. In TEnORs, the point-based representation for valid time is mapped into a modified internal model based on intervals that are segmented, indexed and allocated to temporal blocks to optimize temporal clustering and storage utilization. User defined aggregates [24] are then used to map the queries expressed on the point-based model into equivalent queries expressed on the segmented interval-based model. The design and implementation of SQLST discussed in this paper apply and extend to the spatio-temporal domain the lessons learned with TEnORs.

4 Spatial Operators

Our representation method of a spatial object is based on directed triangulation. The three basic spatial data types, i.e., points, lines (straight line segments), regions (polygons), are represented by triangles at the conceptual level as follows:

• point (x, y): is represented as $((x, y), (x, y), (x, y))$.

• line $((x_1, y_1), (x_2, y_2))$: is represented as $((x_1, y_1), (x_2, y_2), (\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}))$, where (x_1, y_1) is the start point and (x_2, y_2) is the end point of the line, and $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$ is the center of mass of the line.

• region $[(x_1, y_1), (x_2, y_2), (x_3, y_3) \ldots, (x_{n-1}, y_{n-1}), (x_n, y_n)]$: is represented as a set of directed triangles, i.e., $\{((x_1, y_1), (x_2, y_2), (x_3, y_3)), \ldots, ((x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n))\}$. The region is represented as a circular list at the physical level and the line with two ends as (x_i, y_i) and (x_{i+1}, y_{i+1}) is an edge of the region. The algorithm to decompose a region into a set of triangles will be discussed in Section 6. For each triangle the region is decomposed into, the three vertexes are ordered according to a counterclockwise orientation.

If a region has a hole, the vertexes of the hole also form a circular list but prefixed with a negative sign. And the vertexes of each triangle in the set that the hole is decomposed into are clockwisely orientated.

The commonly used spatial predicates [19, 5] are `equal`, `disjoint`, `overlap`, `meet`, `contain`, `adjacent` and `common_border`, etc. Spatial operations include `intersect`, `area`, `perimeter`, `distance`, etc.

4.1 Properties of Directed Triangles

First, we define the direction of border lines of a triangle as *counterclockwise.*

Definition 1: a triangle is a **counterclockwisely directed triangle** if its three vertexes, point1 (x_1, y_1), point2 (x_2, y_2), and point3 (x_3, y_3) are *counterclockwisely* orientated, i.e.,

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} > 0$$

Next, we define the following basic spatial predicates. According to [5], these are first-order queries with linear arithmetic constraints.

– **left(point, line):** a point (x_0, y_0) is on the *left* side of a line $((x_1, y_1), (x_2, y_2))$, which is an edge of a directed triangle, iff

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_0 & y_0 & 1 \end{vmatrix} \geq 0$$

A point is considered to be on the *left* side of a line if it is on the (extended) line, i.e., if the above determinant equals zero.

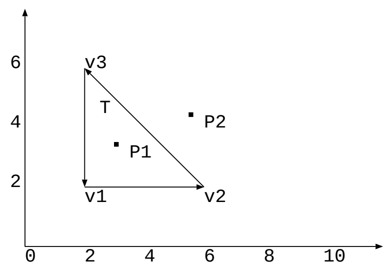

Fig. 3. An example of counterclockwisely directed triangle

Figure 3 is an example of a directed triangle and the meanings of `left` and `inside` operators are showed. The vertexes of the triangle T — $v1$, $v2$ and $v3$ are *counterclockwisely* orientated. The edges $v1 \rightarrow v2$, $v2 \rightarrow v3$ and $v3 \rightarrow v1$ are directed lines and form the counterclockwisely directed triangle T. For example, point $P1$ is on the *left* side of all three edges, so $P1$ is *inside* T. On the other hand, point $P2$ is on the *left* side of edges $v1 \rightarrow v2$ and $v3 \rightarrow v1$ but not on the *left* side of edge $v2 \rightarrow v3$, so $P2$ is not *inside* T.

- **inside(point, triangle):** a point is *inside* a triangle iff the point is on the *left* side of all three edges of the triangle.
- **vertex(point, triangle)** — iff the point is one of the vertexes of the triangle
- **on(point, line)** — iff $\begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0$ and $(min(x_1, x_2) \leq x_0 \leq max(x_1, x_2)$ or $min(y_1, y_2) \leq y_0 \leq max(y_1, y_2))$
- **boundary(point, triangle)** — iff the point is on an edge of the triangle
- **equal(line1, line2)** — iff the set of the end points of line1 is equal to the set of the end points of line2
- **overlaps(line1, line2)** — iff for two lines, line1 $((x_1, y_1), (x_2, y_2))$ and line2 $((x'_1, y'_1), (x'_2, y'_2))$, $\frac{y_2 - y_1}{x_2 - x_1} = \frac{y'_2 - y'_1}{x'_2 - x'_1}$ and (x_1, y_1) is on line2.
- **boundary(line, triangle)** — iff the line overlaps with an edge of the triangle
- **edge(line, triangle)** — iff the two end points of the line are two neighboring vertexes of the triangle.
- **crosspoint(line1, line2)** — the cross point of two lines, line1 $((x_1, y_1), (x_2, y_2))$ and line2 $((x'_1, y'_1), (x'_2, y'_2))$ is point0 (x, y) with the coordinates as:

$$x = \frac{(x_1 - x_2)\begin{vmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \end{vmatrix} - (x'_1 - x'_2)\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}}{(y_1 - y_2)(x'_1 - x'_2) - (y'_1 - y'_2)(x_1 - x_2)}$$
$$y = \frac{(y_1 - y_2)\begin{vmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \end{vmatrix} - (y'_1 - y'_2)\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}}{(y_1 - y_2)(x'_1 - x'_2) - (y'_1 - y'_2)(x_1 - x_2)}$$

  and $(min(x_1, x_2) \leq x \leq max(x_1, x_2)$ or $min(y_1, y_2) \leq y \leq max(y_1, y_2))$. When there is no solution to the above equations, we say point0 = *null*.
- **cross(line1, line2)** — iff the cross point of the two lines is not *null*

4.2 Spatial Relationships

Based on the basic operators discussed in the above section, we can compute relationships between triangles as follows:

- **equal(triangle1, triangle2)** — iff the set of the vertexes of triangle1 are equal to the set of the vertexes of triangle2
- **overlap(triangle1, triangle2)** — iff at least one vertex of triangle2 is *inside* triangle1
- **contain(triangle1, triangle2)** — iff the three vertexes of triangle2 are all *inside* triangle1
- **disjoint(triangle1, triangle2)** — iff non of the vertexes of triangle1 is *inside* triangle2 and vice versa; and non of the edges of triangle1 *crosses* with any edge of triangle2
- **adjacent(triangle1, triangle2)** — iff one edge of triangle1 *overlaps* with an edge of triangle2 and at least one vertex of triangle1 is not *inside* triangle2
- **commonborder(triangle1, triangle2)** — iff an edge of triangle1 is *equal* to an edge of triangle2 and one vertex of triangle1 is not *inside* triangle2
- **meet(triangle1, triangle2)** — iff one vertex of triangle1 is *on* an edge of triangle2 and two vertexes of triangle1 are not *inside* triangle2

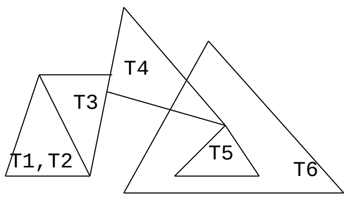

Fig. 4. Example of Relationships Between Triangles

Figure 4 illustrates the definition of these relationships between two triangles. For example, (i) $T1$ and $T2$ which have the same set of vertexes are *equal* to each other. (ii) $T1$ and $T3$ share a *commonborder*, so do $T2$ and $T3$; (iii) $T1$, $T2$, and $T3$ are *disjoint* with $T5$ and $T6$, also $T1$ and $T2$ are *disjoint* with $T4$; (iv) $T3$ and $T4$ are *adjacent* because an edge of $T3$ overlaps with an edge of $T4$ and two vertexes of $T3$ are not inside $T4$; (v) $T4$ *meets* $T5$ because one vertex of $T4$ is on an edge of $T5$ and the other two vertexes of $T4$ are not inside $T5$; (vi) $T6$ *overlaps* $T4$ because one vertex of $T4$ is inside $T6$; and (vii) $T6$ *contains* $T5$ because all three vertexes of $T5$ are inside $T6$.

Furthermore, the relationships between these spatial operators are:

$$\begin{array}{rcl} \mathsf{equal} & \Longrightarrow & \mathsf{contain} \\ \mathsf{equal} & \Longrightarrow & \mathsf{overlap} \\ \mathsf{contain} & \Longrightarrow & \mathsf{overlap} \\ \mathsf{adjacent} & \Longrightarrow & \mathsf{overlap} \\ \mathsf{commonborder} & \Longrightarrow & \mathsf{overlap} \\ \mathsf{meet} & \Longrightarrow & \mathsf{overlap} \\ \mathsf{commonborder} & \Longrightarrow & \mathsf{adjacent} \end{array}$$

For example, if equal(triangle1, triangles2) evaluates to TRUE, then it implies that contain(triangle1, triangle2) also evaluates to TRUE.

4.3 Spatial Operations

The main operations associated with spatial objects are as follows:

- **intersect(triangle1, triangle2, region)** — calculates the intersection formed by the cross points of the edges of the two triangles. The steps are showed in Algorithm 1 where $t1$ and $t2$ are the two input triangles and L is the circular list of the vertexes of the intersected region.

Algorithm 1 intersect two triangles into a region

```
Require: initially $t1$, $t2$, $L = \emptyset$.
1: for each edge $e1$ of $t1$ do
2:   for each edge $e2$ of $t2$ do
3:     $point0 = \mathsf{crosspoint}(e1, e2)$
4:     if $point0 \neq null$ then
5:       append $point0$ to $L$
6:     end if
7:   end for
8: end for
9: return $L$
```

A triangle $((x1, y1), (x2, y2), (x3, y3))$ has the following properties [3]:

- **area** $a = \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$
- **perimeter** $p = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} + \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}$
- **centerofmass** $x_c = \frac{x_1 + x_2 + x_3}{3}$, $y_c = \frac{y_1 + y_2 + y_3}{3}$

The distance between two triangles are defined as the distance between their centers of mass:

- **distance** $d = \sqrt{(x_{2c} - x_{1c})^2 + (y_{2c} - y_{1c})^2}$

5 Spatio-Temporal Queries

In this section, we express various queries in SQLST. We use examples taken from [10].

The database contains three relations: (i) forest relation in which the location and the development of forests changing over time are recorded; (ii) forest_fire relation in which the evolution of forest fires are recorded; and (iii) fire_fighter relation which records the motion of fire fighters.

First, we define the schema in SQLST as follows:

Example 4. Define the forest relation

```
CREATE TABLE forest (forestname CHAR(30), territory REGION, VTime DAY)
```

Example 5. Define the forest_fire relation

```
CREATE TABLE forest_fire (firename CHAR(30), extent REGION, VTime DAY)
```

Example 6. Define the fire_fighter relation

```
CREATE TABLE fire_fighter (fightername CHAR(30), location POINT, VTime DAY)
```

The columns *territory* and *extent* have a spatial data type as REGION and *location* has a type as POINT; temporal data column *VTime* has a granularity of DAY.

Example 7. When and where did the fire called "The Big Fire" reach what largest extent?

```
SELECT F1.VTime, F2.extent, AREA(F1.extent)
FROM forest_fire as F1 F2
WHERE F1.firename = "The Big Fire" AND F2.firename = "The Big Fire"
   AND F1.VTime = F2.VTime
GROUP BY F1.VTime
HAVING AREA(F1.extent) = (SELECT MAX(AREA(extent))
                          FROM forest_fire WHERE firename = "The Big Fire")
```

We only use an user-defined spatial aggregate `area` and a built-in aggregate `max` to express the query. We group-by F1.VTime to calculate the area of the fire extent at each time instant. On the contrary, [10] introduces several special constructs plus a new clause LET to accomplish the same query.

Example 8. When and where was the spread of fires larger than 500 km^2?

```
SELECT F1.VTime, F2.extent
FROM forest_fire as F1 F2
WHERE F1.VTime = F2.VTime AND F1.firename = F2.firename
GROUP BY F1.VTime, F2.extent, F1.firename
HAVING AREA(F1.extent) > 500
```

Example 9. How long was fire fighter Th. Miller enclosed by the fire called "The Big Fire" and which distance did he cover there?

```
SELECT DURATION(fire_fighter.VTime),
        MOVING_DISTANCE(fire_fighter.location, fire_fighter.VTime)
FROM forest_fire, fire_fighter
WHERE forest_fire.VTime = fire_fighter.VTime
    AND firename = "The Big Fire" AND fightername = "Th. Miller"
GROUP BY forest_fire.VTime
HAVING INSIDE(location, extent)
```

Example 10. Determine the times and locations when "The Big Fire" started.

```
SELECT VTime, extent
FROM forest_fire
WHERE firename = "The Big Fire"
    AND VTime = (SELECT MIN(VTime)
                    FROM forest_fire WHERE firename = "The Big Fire")
```

These examples suggest that SQLST (i) provides a simple and expressive formulation for complex spatio-temporal queries, and (ii) represents a minimalist's extensions for SQL that preserves the syntax and the flavor of SQL3, since only new functions and aggregates are required.

6 Implementation of SQLST

As we have discussed before, at the conceptual level and physical level, the spatio-temporal objects should have different data models. At conceptual level, we have used point-based time model and directed-triangulation-based spatial data model. At physical level, an interval-based time model is used, and the same approach should be taken for spatial data, i.e., a region should be stored by its vertexes. Mapping methods between the conceptual level and the physical level representation is thus needed.

6.1 Mapping Between Different Representations

Mapping to an interval-based time model at the physical level solves the space efficiency problem associated with the point-based time model used at the conceptual level. Then, tuples in our internal relations are timestamped with two time instants: one indicates the start-point and the other indicates the end-point of the interval.

Each temporal join is mapped into an intersect operation. A coales aggregation is required on the temporal argument when various columns have been projected. However, coalescing is not needed for those rules where (i) there is no temporal argument in the SELECT clause of a query, or (ii) all variables appearing in the WHERE clause of a query also appear in its SELECT clause. [15]

To map a set of triangles which have the same timestamp into one region, we use Algorithm 2. In Algorithm 2, the input T is a set of triangles and the output L is a circular list of the vertexes of the region that the triangles are merged into.

Algorithm 2 map a set of triangles into a region

```
Require: initially $T$, $L = \emptyset$.
 1: for each triangle $t1 = (v1, v2, v3) \in T$ do
 2:   for each triangle $t2 = (v4, v5, v6) \in T$ do
 3:     if commonborder($t1, t2$) then
 4:       $v = \{v1, v2, v3\} - \{v4, v5, v6\}$
 5:       $v' = \{v4, v5, v6\} - \{v1, v2, v3\}$
 6:       $v_s \in \{v1, v2, v3\} - \{v\}$
 7:       $v_e \in \{v1, v2, v3\} - \{v\} - \{v_s\}$
 8:       if left($v, (v_s, v_e)$) and $\neg$ left($v', (v_s, v_e)$) then
 9:         append $[v, v_s, v', v_e]$ to $L$
10:       end if
11:     end if
12:   end for
13: end for
14: return $L$
```

To decompose a region R into a set of directed-triangles T, we use Algorithm 3 where the input and output are just opposite to that of Algorithm 2.

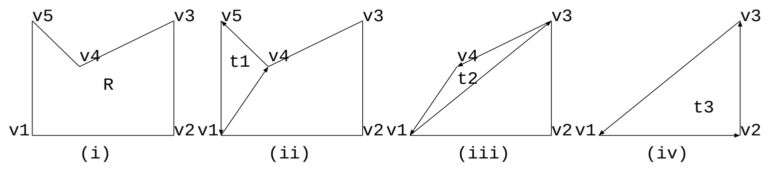

Fig. 5. Decomposing a region into triangles

Figure 5 is an example of decomposing a region R into set of triangles T. Step (i) showed the region with $v1, v2, v3, v4$ and $v5$ as its vertexes. So, initially, $L = [v1, v2, v3, v4, v5]$ and $T = \emptyset$.

In step (ii), we start with $v1$ which has the smallest x and y values, and get $v2 = next(v1)$ and $v5 = previous(v1)$. Since $v4$ is inside the triangle $(v1, v2, v5)$ so we move on to start with $v5$ and get $v1 = next(v5)$ and $v4 = previous(v5)$. Since no other vertex of R is inside the triangle $(v5, v1, v4)$ (namely $t1$) so we set $T = \{t1\}$, $L = [v1, v2, v3, v4]$.

In step (iii), we start with $v4$, and get a triangle $(v4, v1, v3)$. Since no other vertex of R is inside this triangle (namely $t2$), so we set $T = \{t1, t2\}$, $L = [v1, v2, v3]$.

Algorithm 3 decompose a region into a set of triangles

```
Require: initially $L$, $T = \emptyset$.
1: for each element $v \in L$ do
2:   if $x_v = \min(\text{all } x_{vi})$ and $y_v = \min(\text{all } y_{vi})$ then
3:     $v0 = v$
4:   end if
5: end for
6: $v1 = \text{next}(v0)$ and $v2 = \text{previous}(v0)$
7: if $\text{length}(L) > 3$ then
8:   for each element $v' \in L$ do
9:     if $\text{inside}(v', (v0, v1, v2))$ and $v' \notin \{v0, v1, v2\}$ then
10:      $v0 = v2$
11:      goto line 6
12:    end if
13:  end for
14:  insert triangle $(v0, v1, v2)$ into $T$
15:  remove $v0$ from $L$
16: end if
17: insert triangle $(v0, v1, v2)$ into $T$
18: return $T$
```

Lastly, we get the triangle $(v3, v1, v2)$ (namely $t3$), and set $T = \{t1, t2, t3\}$. The procedure stops.

6.2 Spatial Operators for Regions

The spatial operators defined for triangles in Section 4 can be used towards regions with little change.

Let S and S' denote sets of triangles that two regions $R1$ and $R2$ are decomposed into, and t and t' denote an element in S and S', respectively. The operators proposed in Section 4 can be used on two regions in the following ways:

1. $R1$ is `equal` to $R2$ if $S = S'$.
2. $R1$ `overlaps` $R2$ if $\exists t \in S$, $t' \in S'$, such that t *overlaps* t'.
3. $R1$ `contains` $R2$ if $\forall t' \in S'$, $\forall$vertexes v of t', $\exists t \in S$, such that v is *inside* t.
4. $R1$ is `disjoint` with $R2$ if $\forall t \in S$, $\neg \exists t' \in S'$, such that t *overlaps* t'.
5. $R1$ is `adjacent` with $R2$ if $\exists t \in S$, $\exists t' \in S'$ where t is *adjacent* with t' and the two ends of the adjacent edge are *neighbouring vertexes* of both $R1$ and $R2$.
6. $R1$ and $R2$ have a `commonborder` if $\exists t \in S$, $\exists t' \in S'$ where t and t' have a *common border* and the two ends of the common border are *neighboring vertexes* of both $R1$ and $R2$.
7. $R1$ `meets` $R2$ if $\exists t \in S$, $\exists t' \in S'$, such that t *meets* t' and $\forall t1 \in S$, $\forall t1' \in S'$, $t1 \neq t$, $t1$ does not *overlap* $t1'$.

Since a region with n vertexes can be decomposed into $n - 2$ triangles while the relationships between two triangles can be determined in constant time, so the comparison of two regions can be done in $O(n^2)$ TIME.

To find the intersected part of two regions, we only need to find the intersected parts of the triangles decomposed from them. Similarly, The area of a region is simply the sum of the areas of all the triangles it decomposed into. The perimeter of a region is the sum of the length of all its edges. The center of mass of a region can be calculated in a similar way to the calculation of the center of mass of a triangle. The distance between two regions is also defined as the distance between their centers of mass.

7 Conclusion

In this paper, we propose a spatio-temporal data model and query language — SQLST that satisfies Worboys' prescription for spatio-temporal model. A cornerstone to the simplicity and generality of this approach, is the use of a point-based representation of time and directed-triangulation-based representation of spatial objects at the conceptual level. Whereas query languages proposed in the past rely on the introduction of new spatio-temporal constructs, we have taken a minimalist's approach, and showed that the basic syntactic constructs and semantic notions provided by current query languages are sufficient if user-defined functions and user-defined aggregates are supported [24]. With these minimal extensions, SQLST can express queries as powerful as those expressible in other works [14, 10] in a simple and intuitive fashion.

Efficient support for spatio-temporal queries can be obtained by using internal representations that are different from the conceptual one, and then mapping conceptual queries into equivalent queries on the internal representations. This approach has already produced an efficient implementation for SQL^{T}. In this paper we have laid the foundations for efficient implementations of spatial structures by using directed triangular representations and defining equivalent operators on general polygons in terms of these. The subject of efficient implementation for objects whose shape or position continuously changes with time has been left for later research.

References

1. J.F. Allen. Maintaining knowledge about temporal intervals. In *Communications of the ACM*, Vol.26, No.11, pp.832-843, 1983
2. M.H. Bohlen, R. Busatto and C.S. Jensen. Point- Versus Interval-based Temporal Data Models. In *Proceedings of the 14th International Conference on Data Engineering*, pp.192-200, 1998
3. Y.S. Bugrov. *Fundamentals of linear algebra and analytical geometry*, Moscow : Mir Publishers, 1982
4. M. Cai, D. Keshwani, and P.Z. Revesz. Parametric Rectangles: A Model for Querying and Animation of Spatiotemporal Databases. In *Proceedings of the 7th International Conference on Extending Database Technology*, pp.430-444, 2000
5. J. Chomicki and P.Z. Revesz. Constraint-Based Interoperability of Spatiotemporal Databases. In *Advances in Spatial Databases*, LNCS 1262, pp.142-161, Springer, 1997

6. J. Chomicki and P.Z. Revesz. A Geometric Framework for Specifying Spatiotemporal Objects. In *Proceedings of the 6th International Workshop on Time Representation and Reasoning*, pp.1-46, 1999
7. C.X. Chen and C. Zaniolo. Universal Temporal Extensions for Data Languages. In *Proceedings of the 15th International Conference on Data Engineering*, pp.428-437, 1999
8. M.J. Egenhofer. Spatial SQL: A Query and Presentation Language. In *IEEE Transactions on Knowledge and Data Engineering* Vol.6. No.1. pp.86-95, 1994
9. R.H. Guting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. In *VLDB Journal*, Vol.4, No.2, pp.243-286, 1995
10. 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. To appear in *ACM Transactions on Database Systems*
11. M.R. Garey, D.S. Johnson, F.P. Preparata, and R.E. Tarjan. Triangulating a Simple Polygon. In *Information Processing Letters*, Vol.7, No.4, pp.175-179, 1978
12. S. Grumbach, P. Rigaux and L. Segoufin. Spatio-Temporal Data Handling with Constraints. In *Proceedings of ACM International Symposium on Geographic Information Systems*, pp.106-111, 1998
13. S. Grumbach, P. Rigaux and L. Segoufin. The DEDALE System for Complex Spatial Queries. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pp.213-224, 1998
14. C. S. Jensen and R. T Snodgrass. Temporal Data Management. In *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, No.1, pp.36-44, 1999
15. J. Kong, C.X. Chen, and C. Zaniolo. A Temporal Extension of SQL for Object Relational Databases. *submitted for publication*
16. P.C. Kanellakis, G. Kuper and P.Z. Revesz. Constraint Query Languages. In *Journal of Computer and System Sciences*, special issue edited by Y.Sagiv, Vol.51, No.1, pp.26-52, 1995
17. N.A. Lorentzos and Y.G. Mitsopoulos. SQL Extension for Interval Data. In *IEEE Transactions on Knowledge and Data Engineering, Vol.9, No.3*, pp.480-499, 1997
18. R. Laurini and D. Thompson. *Fundamentals of Spatial Information Systems.* Academic Press, 1992
19. M. Schneider. *Spatial Data Types for Database Systems*, LNCS 1288. Springer, 1997
20. R.T. Snodgrass. The Temporal Query Language TQuel. In *Proceedings of the 3rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* pp.204-213, 1984
21. R.T. Snodgrass, et al. *The TSQL2 Temporal Query Language*, Kluwer, 1995
22. D. Toman. Point vs. Interval-based Query Languages for Temporal Databases. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp.58-67, 1996
23. D. Toman. A Point-Based Temporal Extension of SQL. In *Proceedings of the 6th International Conference on Deductive and Object-Oriented Databases*, pp.103-121, 1997
24. H. Wang and C. Zaniolo. User Defined Aggregates in Object-Relational Systems. In *Proceedings of the 16th International Conference on Data Engineering*, pp.135-144, 2000
25. M.F. Worboys. A Unified Model for Spatial and Temporal Information. *Computer Journal*, Vol.37, No.1, pp.26-34, 1994
26. C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, and R. Zicari. *Advanced Database Systems*, Morgan Kaufmann, 1997