

A Framework for Representing Moving Objects

Ludger Becker, Henrik Blunck, Klaus Hinrichs, and Jan Vahrenhold

Westfälische Wilhelms-Universität Münster, Institut für Informatik, 48149 Münster,
Germany. EMail: {beckelu,blunck,khh,jan}@math.uni-muenster.de.

Abstract. We present a framework for representing the trajectories of moving objects and the time-varying results of operations on moving objects. This framework supports the realization of discrete data models of moving objects databases, which incorporate representations of moving objects based on non-linear approximation functions.

1 Introduction

The need for storing and processing continuously moving objects arises in a wide range of applications, including traffic control, physical simulation, or mobile communication systems. Most of the existing database systems, which assume that the data is constant unless it is explicitly modified, are not suitable for representing, storing, and querying continuously moving objects because either the database has to be continuously updated or a query output will be obsolete. A better approach would be to represent the position of a moving object as a function $f(t)$ of time, so that the position changes without any explicit change in the database system, and the database needs to be updated only when the function $f(t)$ changes. Recently, there has been a collection of results on extending existing database systems to handle moving-object databases (e.g., [5, 6, 8]).

In their approach to modeling moving objects, Erwig *et al.* [5] distinguish the abstract and the discrete data model. The abstract model defines the data model a user of the system sees. A crucial point in modeling moving objects is the kind of motion, e.g., linear or non-linear motion. While the abstract model allows for general “smooth curves” [8], their discrete model [5, 6] only considers piecewise linear motion. They approximate continuous motion of objects by assuming piecewise linear motion, interpolating an object’s position or shape between static snapshots [14]. Furthermore, most indexing methods for handling moving objects assume that objects have piecewise linear trajectories (e.g., [1, 7, 10, 13]). The discrepancy between smooth curves and their approximation by polygonal lines has been recognized, and several authors indicate that a discrete model ultimately should allow for representing non-linear motion [4, 12, 15].

In this paper, we develop a framework for representing moving objects that simultaneously addresses three important issues. First, the representation of moving objects’ trajectories should be as natural as possible. In moving object databases, not only the trajectory itself but also higher-order derivatives can be of interest, e.g., to obtain information about change of direction or about

acceleration. Hence, we require that these quantities should be continuous and non-trivial. The second issue is to be able to derive a discrete model closed under (concatenation of) all operators. For example, in a model based on linear motion, the distance between two moving objects is a quadratic function. Applying the distance operator to moving objects whose construction is based on the result of previous applications of the distance operator (we call this the concatenated distance operator) can result in functions of arbitrarily high complexity, which obviously are not captured by a model using linear motion. Because the second issue cannot be resolved using both constant-size information and exact representation, the final issue addressed is to develop a compact representation with bounded space requirements and best possible approximation of the trajectory in question. Objects modeled in specific applications will impose quite different requirements for how these issues may be addressed. Consequently, being “best” is not a global property of any fixed representation but heavily depends on the application at hand, and we present our approach in an extensible manner.

2 Representing Trajectories for Moving Objects

We will use the notation of the abstract model by Forlizzi *et al.* and Güting *et al.* [6, 8]. The abstract model provides basic constant types like `int`, `real`, `string`, and `bool`, spatial data types `point` $\langle d \rangle$, `line` $\langle d \rangle$, and `region` $\langle d \rangle$, where d is the number of dimensions of the underlying space, and timevariant types. In principle, each of the constant and spatial data types can be lifted to a timevariant type [8]. In most applications objects of type `line` $\langle d \rangle$ are polylines and extended geometric objects of type `region` $\langle d \rangle$ are polygonal regions which both can be described by a set of timevariant vertices. Therefore we concentrate on the representation of d -dimensional timevariant points denoted as `mpoint` $\langle d \rangle$ and on onedimensional timevariant datatypes, especially `mreal`. For all data types there are constructors, which can be used to initialize corresponding objects, e.g., there is a constructor which initializes an object of type `mpoint` $\langle d \rangle$ based upon d objects of type `mreal`. All operators defined on constant types can be lifted to timevariant types and thus be applied to timevariant objects. In addition, other operators for moving objects of type `mpoint` $\langle d \rangle$ can be defined, e.g., `trajectory`, `derivative`, `speed`, `velocity`, and `acceleration` [8].

2.1 Requirements for Moving Object Trajectories

The motion of real-world objects is usually recorded in discrete steps, e.g. by radar-, radio-, or GPS-based tracking devices. Hence any attempt to model the motion has to respect this discrete set of spatial information. A central concept will be what we call a *constraining fact*, a pair (t_i, f_i) , which indicates information (e.g., a spatial position) f_i valid for an object at a specific point in time t_i . The motion of an object can then be modeled by a discrete collection of constraining facts and an appropriate interpolation function which gives the trajectory of the object as the graph of the interpolation function in the $(d + 1)$ -dimensional

space (d spatial dimensions, one temporal dimension). The information f_i can consist of, e.g., the spatial position, speed, direction, acceleration, or higher-order derivatives at time t_i . Current models usually use the spatial position. A *concept of representation* is given by one or more kinds of trajectories (e.g., piecewise linear curves or splines), a construction algorithm for deriving an appropriate interpolation function, and an algorithm for evaluating this function.

There is an obvious trade-off between the size of the representation and its precision. A representation of arbitrary precision would not only incur increased overhead for evaluating interpolating functions but, more important, due to unpredictable differences in the objects' sizes, organizing data layout and data accesses becomes more expensive. Therefore Cao *et al.* [3] consider algorithms for compressing trajectories which select from the facts known about a moving point the most relevant ones which are then used for interpolation by a (compressed) trajectory. We abstract from this by using as defining facts those which have to be interpolated, but we require, however, that the amount of storage needed for the result of an operation on timevariant data is bounded as follows: **Bounded size requirement:** The size $|I(mp)|$ of the interpolating function's representation $I(mp)$ linearly depends on the number $F(mp)$ of constraining facts, that is $|I(mp)| \leq c \cdot F(mp)$, where the constant c is global for the concept of representation. When applying an operator, the maximum allowed size for the representation of the result is bounded by the maximum size for the representation of any of the operands, that is, if an operator op is applied to operands mp_1, \dots, mp_n , we have $|I(op(mp_1, \dots, mp_n))| \leq \max\{|I(mp_1)|, \dots, |I(mp_n)|\}$.

2.2 Trajectory Types for Specific Requirements

Different applications impose different requirements on the kind of interpolation function, and there are several classes of interpolation functions available.

If only discrete positions of a moving object are known, but not speed, moving direction, and acceleration, the trajectory can be represented by piecewise linear interpolation, i.e., by a polyline in $(d + 1)$ -dimensional space, which is given as a sequence of points, e.g., for 2-dimensional (x, y) -space by a sequence $(x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_n, y_n, t_n)$. This representation has the disadvantage that the first-order derivative of the trajectory is a non-continuous function, i.e., speed and direction may change only in discrete steps. Furthermore, acceleration which is represented by the second-order derivative is always zero, i.e., piecewise linear interpolation cannot capture non-zero acceleration.

If continuous first- and second-order derivatives are needed to enforce smooth changes of speed and direction, (higher-degree) splines can be used. For representing smooth curves, B-splines [11] form a well-investigated concept with numerous applications in numerical analysis, geometric modeling, computer graphics and other areas. Besides continuous first- and second-order derivatives cubic splines have a nice balancing property: they have a nearly optimal, minimal average quadratic curvature and therefore are inherently smoother and oscillate less than other interpolation functions like higher-order polynomials. In our context this means that changes of direction and speed are minimized and uniformly

distributed between constraining facts. The interpolating spline function of a trajectory does not deviate too much from the polygonal chain connecting the sample points, and its local behavior can be predicted [11].

Besides location information, speed and direction of movement may also be available, e.g., by GPS-based tracking devices. In the context of Hermite-interpolation [11] not only given time/space positions can be prescribed, but also the first-order derivatives of the resulting trajectory in these points, i.e., speed and direction. In certain applications, it may be useful to adapt the representation of moving objects to reflect additional properties of an object's motion. For example, an airplane performs turns only very close to sample points and otherwise moves almost linearly. This can be modeled by ν -splines [9] for which the amount of change in the second continuous derivative can be (even locally) prescribed. This amount is determined by the turning radius of the airplane. This exact prescription can be necessary, e.g., in modeling airport approach procedures where airplanes approach two parallel runways in a butterfly-shaped flight pattern.¹ Other applications may require mixed representations, i.e., the representation of a trajectory uses different types of interpolation functions for different time intervals, e.g., circular arcs and piecewise linear curves.

3 Representing Results of Operations on Moving Data

Our goal is to represent the timevariant results of operations on moving data, i.e., to represent results of operations on trajectories, such that the requirements for a representation are not violated. In terms of the abstract model described in Section 2, this demands the closure of the types `mpoint` $\langle d \rangle$ and `mreal` under operations, in particular under the concatenation of lifted operations. The following examples illustrate the major problems that can arise with respect to the closure under operations (mp_i denoting an instance of class `mpoint` $\langle d \rangle$):

Example 1: Compute the distance `mdistance`(mp_1, mp_2).

If mp_1 and mp_2 are represented by piecewise linear trajectories, the exact result is given piecewise by the root of a polynomial of degree two. Erwig *et al.* [5] observed this problem in the linear model, and Forlizzi *et al.* [6] proposed an extended model using piecewise linear interpolation. They propose to extend the datatype `mreal` for moving reals to also represent timevariant roots of polynomials of degree 2. However, this is not sufficient if the results obtained by applying the `mdistance` function are used in further calculations. The result in the next example cannot be represented in the extended model by an `mreal` since it cannot be described piecewise by the root of a polynomial of degree 2.

Example 2: Compute the concatenated distance `mdistance`(`mdistance`(mp_1, mp_2), `mdistance`(mp_3, mp_4)), the timevariant absolute value of a difference.

¹ The European Organization for Safety of Air Navigation maintains a database <http://www.eurocontrol.fr/projects/bada> of the inflight behavior, e.g., velocity or descent speed, of over 250 aircraft types to support exact modeling.

This computation is useful when analyzing the timevariant distances of pairs of planes flying in formation or the timevariant distances of particles to a nucleus. Other applications may use the results of the `mdistance` function to construct new instances of class `mpoint` $\langle d \rangle$. In Forlizzi *et al.*'s extended model these instances cannot be represented because they cannot be described piecewise by a line or a root of a polynomial. To make things worse, the `mdistance` function as well as other functions may even be applied to such constructed instances. This shows that the representation model should be closed under unlimited number of lifted operations. Applications using the results of the `mdistance` function to construct new instances of class `mpoint` $\langle d \rangle$ can be found, e.g., in object tracking. Distance-based tracking systems (e.g., GPS or sonar tracking) use distance measurements with respect to a fixed number of sites to reconstruct the movements of objects. If these timevariant distances are of interest in their own right, they should be appropriately interpolated over time (and thus be stored as `mreals`). If it cannot be guaranteed that the distances are measured synchronously at the same points-in-time it becomes necessary to interpolate the timevariant distances in order to reconstruct and store the recorded motion.

Example 3: Compute the timevariant center of gravity of a set $\{mp_1, \dots, mp_n\}$.

This operation requires the closure of the model under addition and scalar multiplication of the data type `mpoint` $\langle d \rangle$. This requirement is already fulfilled for a model based upon piecewise linear trajectories, but, in general, the bounded size requirement will be violated: To exactly represent the result, each constraining fact of the operands has to contribute to the representation of the result. More precisely, if for each mp_i the number of constraining facts $F(mp_i)$ is at most s , then the following holds for the result m_r : $s \leq F(m_r) \leq n \cdot s$.

The discussion shows that no concept of representation is known to exactly represent the results of relevant lifted geometric operations since the representation size for the exact results obtained by iterated applications of the `mdistance` operator cannot be bounded. Additionally, it is impossible to maintain the bounded size requirement for the application of non-unary operators while at the same time respecting all constraining facts of all operands. To resolve this conflict while maintaining the closure of the model, we propose that the construction algorithm included in the concept of representation must support a size reduction of the representation of the interpolation function, which preserves the main characteristics of the trajectory. We propose to classify and to process the constraining facts of the operands according to their relevance.

Application of Operators and Recruitment Strategies In the following, we describe how a construction algorithm can proceed to select constraining facts from the set of constraining facts given by all operands. Such a recruitment strategy first has to determine a bounded number of points in time for which either reliable spatial information is known, or which are necessary for preserving the main characteristics of the trajectory. If these points result in more spatial information than can be accommodated in the maximum allowed space (given

by the bounded size requirement), the strategy has to select the most relevant points, if there are less such points, the strategy may choose to include additional points of less reliability. This process is guided by assigning to each point-in-time t_i given by a constraining fact (t_i, f_i) a relevance score $v(t_i)$, and then selecting a bounded number of points-in-time according to these scores. The result of an operator is then computed using the following two steps: (i) For each chosen point-in-time, retrieve the spatial information of all operands valid at that point-in-time and apply the operator to this spatial information. (ii) Interpolate the results of Step (i) using the given interpolation method.

A *first-class point-in-time* has an associated spatial information that has to be accounted for in the representation of the trajectory, whereas a *second-class point-in-time* has an associated spatial information considered less relevant. We can restate the bounded size requirement in the context of an operator’s result:

Definition 1. *Let op be an n -ary operator and let $\{mp_1, \dots, mp_n\}$ be the set of its operands. The bounded size requirement for op is fulfilled if and only if the size of the representation for $\text{op}(mp_1, \dots, mp_n)$ is bounded by a (global) constant times the maximum number of first-class points-in-time for any of the operands.*

As long as we can bound the size of the representation by a (global) constant times the number of first-class points-in-time, this definition exactly matches our earlier requirement that the maximum allowed size for the representation of the result is bounded by the maximum allowed size for the representation of any of the operands, and we will demonstrate below how to obtain these bounds. This definition also subsumes the construction of an instance of class `mpoint` $\langle d \rangle$ from a set of m constraining facts if we consider the constructor as an operator that takes an argument of size m and if we consider each timestamp of a constraining fact as a first-class point-in-time. Second-class points-in-time will not occur in trajectories initially constructed from a set of constraining facts as all facts are exactly known and thus reliable. They may occur, however, in the representation of an operator’s result, e.g., if, at a given point-in-time t , reliable information is known for only one of the operands. Consequently, the spatial information of the operator’s result at point-in-time t may be less relevant for the (overall) representation than the spatial information at any point-in-time t' where reliable information is known for all operands. In the following description, we assume that all operands are defined on a (time) interval $[a, b]$.

We propose to implement a skeleton for a recruitment strategy as follows:

- Step 1:** Select $t = a$ and $t = b$ as first-class points-in-time. Assign these points-in-time a score of $v(t) = 5 \cdot n$ (always select interval end points).
- Step 2:** Collect all first- and second-class points-in-time given by the operands and label them as follows: Each point-in-time t is assigned a score of $v(t) = 3 \cdot fc(t) + sc(t)$ where $fc(t)$ denotes the number of operands considering t as a first-class point-in-time and where $sc(t)$ denotes the number of operands considering t as a second-class point-in-time. Consequently $v(t) \leq 3 \cdot n$.
- Step 3:** [Optional] Select points-in-time according to a strategy-specific heuristic and assign each such point-in-time t a score of $v(t) = 4 \cdot n$ (a recruiting

heuristic, e.g., uniform sampling, is allowed to override the operand-given classification), a score of $v(t) = 2 \cdot n$ (respect points that coincide with many first-class points), or a score of $v(t) = 0$ (supply additional points as needed).

Step 4: Let m be the maximum number of points-in-time that can be accounted for according to the bounded size requirement. Of the m recruited points-in-time with the highest assigned scores, the m' points-in-time with the highest assigned scores are labeled first-class points-in-time, where m' is the maximum number of first-class points-in-time of any single operand. The remaining $m - m'$ points-in-time are labeled second-class points-in-time.

It is easy to realize that this strategy does not violate the bounded size requirement. Note, that the global constant c allows for an application-specific trade-off between exactness and storage costs. The third step (selection according to a heuristic) can prove useful, e.g., when computing the distance. In this situation, the points-in-time corresponding to local extremal points of the distance function may be chosen to ensure that all such extremal points are accounted for in the representation of the operator’s result. In addition to the flexibility in handling operator-specific requirements, the third step of the recruitment strategy facilitates extending the framework: If a specific interpolation function is known to work best for, say, equidistant interpolation points, it can use this step to enforce a corresponding (over-)sampling of all operands. Neglecting a possible overriding due to the recruiting heuristics in Step 3, our concept creates a representation of an operator’s result that is derived from “relevant” (e.g. exactly known) information, i.e. the (important) constraining facts of the argument trajectories. The categorization of the result’s constraining facts according to their relevance guarantees this property also for results of operations applied to operation results—a main advantage over standard curve simplification algorithms.

4 Implementing the Framework

We are currently implementing a framework with the following properties:

- Encapsulation of various interpolation methods.
- Incorporation of construction/evaluation algorithms for each trajectory type.
- Closure with respect to an arbitrary concatenation of operators.
- Incorporation of different kinds of recruitment strategies.
- Capability of handling different kinds of constraining facts.

Our approach (Fig. 1) associates a moving object of class `mpoint<d>` with a representation of its interpolation function, which consists of d objects of class `Function`, each representing the motion in one spatial dimension. There are subclasses of class `Function` for each kind of trajectory. An `mpoint<d>` object also references the constraining facts interpolated by its trajectory. Each constraining fact is represented by an object of class `InTimeData`. This object at least has attributes for a timestamp and a classification, which in turn may be used to guide recruitment strategies. We may derive subclasses from `InTimeData` to represent

different kinds of constraining facts, e.g., the class `InTimePositionData` represents the spatial position corresponding to a point-in-time. Other subclasses may be derived to represent other kinds of information, e.g., speed, direction, or acceleration. Depending on the concrete subclass of `Function`, the information part f of a constraining fact (t, f) can be provided efficiently by the `Function` object. Then, constraining facts are represented by instances of subclasses of class `InTimeData`, and `getInTimeData()` uses the method `Function::evaluate()` to determine f .

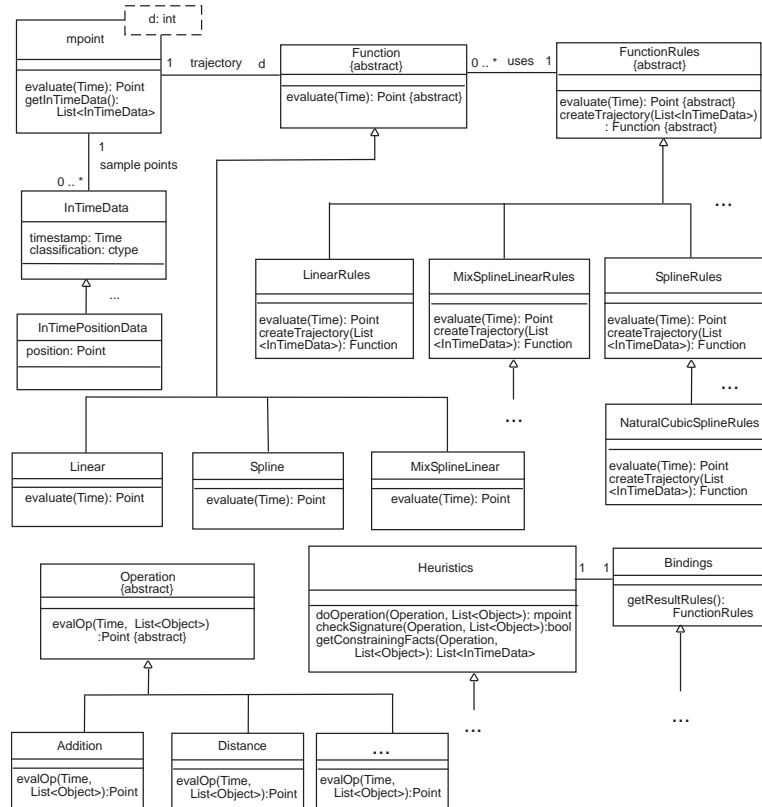


Fig. 1. Class diagram for representing and handling objects of class `mpoint(d)`.

The construction algorithm and the evaluation algorithm corresponding to a concept of representation are realized by a factory method `createTrajectory()` and by a method `evaluate()`, both operations of a subclass of `FunctionRules` to which construction and evaluation is delegated. Each `Function` object, which is responsible for the representation of the interpolation function associated with a moving object, has an associated `FunctionRules`-object. This design allows interpolation functions to be combined with different construction and evaluation algorithms in the form of subclasses of `FunctionRules`.

To construct moving objects resulting from the application of an operator, an object of class `Heuristics` and an object of class `Bindings` are required. The `Heuristics`-object provides the recruitment algorithm (see Section 3) and the `Bindings`-object selects the concept of representation (i.e., the subclass of class `FunctionRules`), which is used to represent the resulting object.

Operations are implemented as subclasses of class `Operation`. Each `Operation`-object has a signature, which is used to check the type of each operand. If an object of class `mpoint<d>` is constructed as the result of an operation, the method `doOperation()` of the responsible object of class `Heuristics` is activated by a message containing the responsible object of class `Operation` and the corresponding operands. The referenced `Bindings`-object determines for each dimension the kind of the resulting trajectory, i.e., a subclass of class `FunctionRules` is selected depending on the operation and the trajectory types of the arguments. The construction algorithm realized by that class finally constructs an object of the corresponding subclass of class `Function`.

5 Experimental Evaluation

We have implemented the class structure described above as an extension of the GIS-database-kernel GOODAC, developed at the University of Münster [2] on top of the object-oriented database system OBJECTSTORE. We have conducted a small set of experiments to examine the practical relevance of the proposed model. The focus of our interest was on the extra cost for using trajectories that allow for higher-order derivatives. We created piecewise linear trajectories and cubic B-spline trajectories for randomly generated constraining facts and compared construction and evaluation time for randomly selected points in time.

No. of Facts	Construction			Evaluation		
	Linear	Spline	Ratio	Linear	Spline	Ratio
40	0.314s	0.510s	1.61	0.793ms	0.933ms	1.33
160	0.693s	1.219s	1.76	0.892ms	1.174ms	1.32
320	1.461s	2.722s	1.86	0.964ms	1.307ms	1.36
640	5.783s	8.814s	1.52	1.014ms	1.518ms	1.50

The results given above have been obtained on a Sun Enterprise 250 (2 * 400 MHz, 1.5 GB RAM) and reflect the timings averaged over five runs. Evaluation timings additionally have been averaged over sequences of 10 and 1000 evaluations. A non-trivial amount of the running time, however, is spent on allocating or accessing (persistent) GOODAC-objects. Thus, relative performance may change if only transient objects are involved, e.g., in main-memory databases. The non-constant evaluation time for linear trajectories is due to having to locate the interval containing the elevation time. The construction time for the piecewise linear representation mainly consists of time spent within GOODAC for allocating objects representing the constraining facts: the difference in construction time is the time actually needed for computing the spline coefficients.

The important conclusion that can be drawn from this small set of experiments is that there is only a moderate loss in construction and evaluation performance when going from piecewise linear to spline interpolation. In our setting, evaluating the (more powerful) spline representation takes no more than 150% of the evaluation time for piecewise linear trajectories.

Conclusions. We have presented a framework that supports the realization of discrete data models of moving objects databases, which allow for representations of moving objects based on non-linear approximation functions and of results of time-variant operators. We also have stated general and strategy-dependent requirements for non-linear approximations. The framework is closed with respect to arbitrary concatenation of operators and can be extended easily. Preliminary experiments show that evaluation performance diminishes only moderately when switching from linear to a (more powerful and complex) spline representation.

References

1. P. K. Agarwal, L. A. Arge, J. Erickson. Indexing moving points. *Symp. Principles of Database Systems*, 175–186, 2000.
2. L. Becker, A. Voigtmann, K. H. Hinrichs. Developing applications with the object-oriented GIS-kernel GOODAC. *Symp. Spatial Data Handling*, 5A1–5A18, 1996.
3. H. Cao, O. Wolfson, G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *Symp. Mobile Ad Hoc Networking and Comp.*, 33–42, 2003.
4. J. Chomicki, P. Z. Revesz. A general framework for specifying spatiotemporal objects. *Workshop Temporal Representation and Reasoning*, 41–46, 1999.
5. M. Erwig, R. H. Güting, M. Schneider, M. Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica*, 3(3):269–296, 1999.
6. L. Forlizzi, R. H. Güting, E. Nardelli, M. Schneider. A data model and data structures for moving objects databases. *SIGMOD Int. Conf. Management of Data*, 319–330, 2000.
7. L. J. Guibas, J. S. B. Mitchell, T. Roos. Voronoi diagrams of moving points in the plane. *Workshop Graph-Theoretic Concepts in Computer Science, LNCS 570*, 113–125, 1992.
8. R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Systems*, 25(1):1–42, 2000.
9. G. D. Knott. *Interpolating Cubic Splines*. Birkhäuser, Boston, MA, 1999.
10. G. Kollios, D. Gunopulos, V. J. Tsotras. On indexing mobile objects. *Symp. Principles of Database Systems*, 261–272, 1999.
11. G. Nürnberger. *Approximation by Spline Functions*. Springer, New York, 1978.
12. D. Pfoser, C. S. Jensen. Querying the trajectories of on-line mobile objects. *Workshop Data Engineering for Wireless and Mobile Access*, 66–73, 2001.
13. S. Saltenis, C. S. Jensen, S. T. Leutenegger, M. A. López. Indexing the positions of continuously moving objects. *SIGMOD Conf. Management of Data*, 331–342, 2000.
14. E. Tøssebro, R. H. Güting. Creating representations for continuously moving regions from observations. *Symp. Spatial and Spatio-Temporal Databases, LNCS 2121*, 321–344, 2001.
15. T.-S. Yeh, B. De Cambray. Modeling highly variable spatio-temporal data. *Australian Database Conference*, 221–230, 1995.