Modeling and Engineering Algorithms for Mobile Data

Henrik Blunck, Klaus H. Hinrichs, Joëlle Sondern, Jan Vahrenhold

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

Abstract

In this paper, we present an object-oriented approach to modeling mobile data and algorithms operating on such data. Our model is general enough to capture any kind of continuous motion while at the same time allowing for encompassing algorithms optimized for specific types of motion. Such motion may be available in a specific form, e.g., described by polynomials or splines, or implicitly restricted using bounds for speed or acceleration given by the application context.

Key words: spatio-temporal data; object-oriented modeling; algorithm engineering

1 Introduction

Over the past years, mobile data, such as points, lines, and regions whose coordinates change according to some time-variant function, has been the subject of increasing interest in Geographical Information Science and closely related fields. The Spatio-Temporal Databases community has contributed to the modeling and representation of such data, whereas algorithmic aspects have mainly been investigated in the field of Computational Geometry. Two recent surveys [2, 32] not only list the main results but also identify a number of issues that arise when handling mobile data; these issues include the ques-

tion of numerical robustness, the incorporation of more realistic descriptions of motion, and the question of how to trade off realism and efficiency.

Efficient algorithms for processing and analyzing mobile data usually build on assumptions about the nature of the object's trajectories; commonly, these trajectories are assumed to have a representation based upon piecewise linear or (fixed-degree) polynomial curves – see, e.g., [1, 6, 18, 21, 25]. It has been noted, however, that such an assumption should be avoided wherever possible to allow for more realism [5, 13, 29, 37], especially since these assumptions are crucial only for the analysis of the running time but rather seldom for the correctness of the algorithm. In contrast, real-world scenarios may involve heterogeneous sets of objects whose motion should not be oversimplified by using a single type of motion description such as polynomials. Examples include (air) traffic control, monitoring meteorological phenomena, or tracking wildlife. Furthermore, a complete motion description of the objects may not be known at all or the objects may not be able (or willing) to publish this description, e.g., in the context of mobile services.

The most general (if not: minimalistic) interpretation of the trajectory of a mobile object considers it simply as a continuous function $f(t)$. Hence, a trajectory is modeled as a black box whose interface consists of a single method that allows to obtain the position of the object at any given point in time of its lifespan. Clearly, the trajectory of any "real-world" object can be modeled using this approach; this, however, raises a number of new issues such as the closure of the model and the efficiency of operations on trajectories.

Applications in which the analysis of mobile objects is required can be classified as either *real-time* or *retrospective*; in the first scenario, data can only be accessed at the current point in time (and thus only in increasing chronological order), whereas the second scenario allows random access to data at any point in the past. Application contexts may also provide additional information about mobile objects, e.g., bounds on speed, acceleration, or turning radius, and we need to be able to incorporate such information.

1.1 Related Work

Collision Detection and Motion Restrictions. A frequently performed task in managing mobile data is collision detection and collision warning, and Lin and Manocha [26] review the broad body of literature. Most practical methods are considered in a *real-time* setting, e.g., interaction in virtual-reality environments, and thus cannot make any assumption about the description of motion. To cope with this, most methods use hierarchical decompositions of the objects or temporal coherence for fast pruning of the search space.

A notable exception is the work by Hayward et al. [23] (and the conceptually similar model by Kahan [24]) that relies on a completely different concept: to quickly identify objects that are most likely (or most unlikely) to collide, it uses a restriction-based approach in which, for each mobile object, bounds on the maximal or minimal speed are known. Note, however, that if no such bounds are known (and exploited), the correctness of the approach cannot be guaranteed. Similar concepts have been used in the context of spatio-temporal indexing [31] and in the context of managing uncertainty in spatio-temporal databases [12, 28, 30, 36].

Kinetic Data Structures. A variety of algorithmic problems involving mobile objects has been addressed successfully in the context of *kinetic data structures* [5, 6]. A kinetic data structure (or: *KDS*) for a set $\mathcal{P}$ of mobile objects maintains a time-variant combinatorial description of some property of $\mathcal{P}$, e.g., its extent as given by the convex hull – see Guibas' survey [19] of recent KDS-related results. Even though the objects are moving according to some known, continuous "flight plan" (which may or may not be updated), the combinatorial structure maintained in the KDS will change only at some discrete points in time. To obtain these points in time, the KDS repeatedly identifies roots of so-called certificate functions that guarantee the validity of the combinatorial description.

The main requirement for a meaningful theoretical analysis of the efficiency of a KDS is that the motion, and thus the description of the certificates as well, is given as a polynomial function of time. A major benefit of working with polynomials is that there exists a variety of numerically robust methods for isolating roots – see, e.g., the survey by Schirra [33]. Several of such methods have been integrated in an upcoming extension package [20] for the Computational Geometry Algorithms Library CGAL [14] thus providing efficient support for working with KDSs. The question of whether or not polynomials provide a "good" level of realism for modeling the motion of real-world objects has been raised (and answered) by Basch who concludes that using polynomials may be "too restrictive to be of much use in applications, although it is perfectly adequate for theoretical purposes" [5, p 103].

Modeling Mobile Data. In the context of spatio-temporal databases, a number of approaches to modeling mobile data has been presented [7, 15, 22]. In this paper, we revisit a model that we have proposed earlier [7], and we refer the reader to our original paper for a discussion of related approaches.

Recently, Mount et al. [27] presented a framework that changes the concept of kinetic data structures to make them applicable to the *real-time* setting: In contrast to the original approach, this framework follows an earlier model of Kahan [24] and is based on incremental updates that involve small

time steps. Unlike a classical KDS framework, it does not require complete knowledge of the kind of motion but instead estimates future locations revisiting these estimates whenever safety constraints are violated. Again, the correctness depends on the presence of motion restrictions. Kahan's model has also been revisited in the context of the competitive analysis of on-line algorithms – see [11] and the references therein.

1.2 Our Results

The main purpose of this paper is to present a general framework for modeling and engineering algorithms for mobile data. This framework has been successfully implemented in the context of the GOODAC object-oriented geo-database kernel [8], and it makes a first attempt at addressing the issue of engineering robust algorithms for more realistic and possibly heterogeneous motion data. It builds upon a representation scheme we have proposed earlier [7], but whereas our earlier scheme focused on representing and storing moving objects in an object-oriented database management system, we now extend it to also support spatio-temporal (main-memory) algorithms. The main features of our approach are the use of a minimalistic interface (thus allowing for arbitrary continuous motion description) and a strict isolation of algorithmic primitives (thus allowing for better algorithm engineering); it can be seen as extending the concept of kinetic data structures to encompass more general motion description. We use the problem of collision detection for a heterogeneous set of objects as a running example and present corresponding primitives for both the *real-time* and the *retrospective* setting as well as an improved approach that exploits the properties of the *retrospective* setting.

2 Representing Mobile Data

Our approach to representing mobile data [7] is based upon two assumptions: (1) the trajectory of a moving point (and thus also of a segment's endpoint or a polygon's vertex) is a t-monotone, continuous curve $f(t)$, and (2) the representation $f(t)$ can be evaluated at any point in its domain. These two assumptions are the most basic assumptions that can be made about trajectories and do not imply any restrictions for the representation of the motion of real-world objects.

Almost all motion data obtained from real-world moving objects is either available in advance as a complete motion description or is given as a

collection of timestamped locations, e.g., obtained through the use of GPS-based devices. In order to convert the latter discrete points to a continuous function, interpolation and approximation techniques are applied. It has been noted frequently that there is no single interpolation technique, e.g., piecewise linear or polynomial, that is optimal over a wide range of scenarios; the trajectory of an airport's ramp-agent or of a plane being towed on ground level may be represented by a piecewise linear function, but the trajectory of an airborne plane with a limited possible turning radius can be represented in a much more realistic way using ν-splines; that is, even if we are working with a single class of objects, e.g., planes, their motion may have completely different characteristics.

These considerations result in a class design (see Fig. 1) whose core class `mpoint`$\langle d \rangle$ represents a time-variant point in d dimensions. Each instance of `mpoint`$\langle d \rangle$ stores a collection of timestamped location data, and the (continuous) representation of the motion restricted to each dimension can be (re-) constructed using a specialization of the `Function` interpolation class. More specifically, an instance of `mpoint`$\langle d \rangle$ aggregates d instances of specializations of `Function` and delegates the evaluation of the trajectory to them.

When working with this framework, a number of issues have to be taken into consideration, most notably the issue of maintaining the model closed under (concatenated) operations such as (time-variant) difference or distance computation. For a more detailed description of the framework and for a discussion of the practical efficiency of its implementation we refer the reader to our previous paper [7].

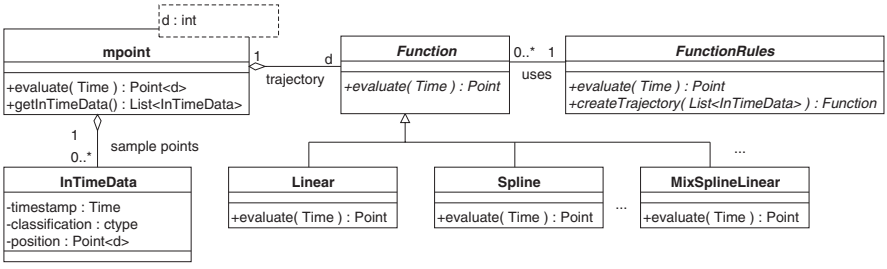

Fig. 1. Class diagram for representing mobile point data (see [7])

3 Modeling Algorithms for Heterogeneous Sets of Mobile Data

As we have mentioned in the introduction, polynomials are a very popular type of curves for modeling motion. A main reason for this is that there exists a variety of numerically robust methods for isolating the roots of a polynomial; this is exploited, e.g., in the context of (certificates for) kinetic data structures. However, in certain applications involving mobile objects (such as in the above-mentioned airport scenario), it is necessary to consider sets of objects that have different motion characteristics.

In this section, we extend the model sketched in the previous section to allow for modeling algorithms for heterogeneous sets of mobile objects. In Section 4, we discuss how to provide means for a more efficient treatment of special instances for which additional information about the objects involved is available.

Example. Our exposition proceeds using the following well-known problem setting as a running example: Given a set of mobile objects that move along the real axis, find all collisions between them. In the two-dimensional (t, y)-parameter space, this setting translates to the problem of finding all intersections induced by a set of t-monotone curves. At first, this seems identical to what templated algorithms for t-monotone curves can handle, e.g., the industrial-strength methods of CGAL's `Arrangement_2` class. A closer look, however, reveals that the implementation of such methods always assumes that the curves belong to the same class of functions, e.g., polynomials, and since we do not make any assumption about the nature of the objects involved, our setting is much more general and thus encompasses a wider range of scenarios. Any (two-dimensional) specialization of our `Function` class, on the other hand, would work fine in the context of the `Arrangement_2` class as long as all primitive operations required by the corresponding algorithm are realized. This issue is addressed as part of the following discussion.

3.1 Isolating Primitive Operations from Algorithms

Our design for modeling algorithms for heterogeneous sets of mobile objects follows a classic "black box"-based approach, that is, we isolate from the general algorithm all operations (*primitives*) that are dependent on the type of trajectory. For each kind of primitive, e.g., intersection-finding, we have a `Decider` class that encapsulates knowledge about how to handle different types of trajectories. Whenever the algorithm needs to process heterogeneous

motion descriptions, it polls a `Decider`-instance which, depending on the types of motion, selects an appropriate specialization of the primitive (see Fig. 2).

Fig. 2. Handling heterogeneous sets of mobile objects using a `Decider`-object

Example. In our intersection-finding example, the main operation that needs to be isolated is the test for whether or not two given trajectories intersect in some given (possibly unbounded) time interval $[begin, end]$; Boissonnat and Vigneron [10] declare this predicate as "mandatory". Assuming that we have to check two curves s and t, this test is implemented as follows:

```
if ( s.intersectsWithin(t, begin, end) )  /* ... */
```

In the above situation, the class of which s is an instance needs to provide a polymorphic version of the method `intersectsWithin` for each additional type of trajectory that is supported by the system.

Using a `Decider`-instance, we decouple the knowledge about other classes in the system from the class representing a certain trajectory type. This knowledge (and thus the main administrative burden) is encapsulated in the corresponding `Decider`-class, and the above code fragment then looks as follows:

```
IntersectionPred ip = myIntersectionDecider.poll(s, t);
if ( ip.eval(s, t, begin, end) == true )  /* ... */
```

The "Double Dispatch"-Problem. The problem we have addressed in this section is known as the *double dispatch* problem [16] where the (type of) result of an operation depends on the type of its operands. While some programming languages, e.g. Smalltalk, provide mechanisms to directly address this issue, the generic solution is to employ the so-called *Visitor* design pattern [16] which reduces the problem to type-dependent single-argument dispatching. As we show in Section 4.2, our algorithms not only depend on the *type* of objects but also on (a combination of) their properties. Thus, the *Visitor* pattern cannot be used, and we feel that our solution discussed above is the best-suited approach for the problem at hand.

3.2 Modeling Compound Functions

For our running example of intersection-finding, we observe that finding intersections between two curves s and t is equivalent to determining the zeros of $s - t$. At this point, the minimalistic interface provided by the class `mpoint`$\langle d \rangle$ (see Section 2) turns out to be a strong design advantage: the concatenation of continuous functions again is a continuous function (with the necessary care taken for the case of division). A modification of the base framework allows to represent compound functions (such as `Difference`) that are composed of other functions, and we have implemented the framework given in Figure 3.

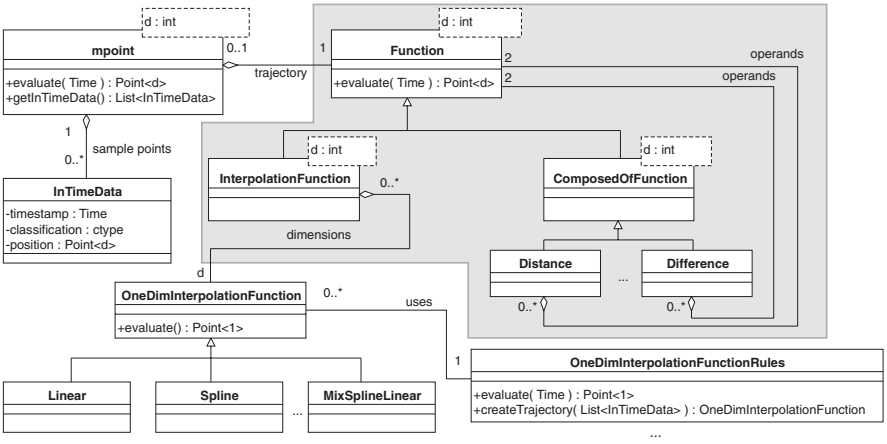

Fig. 3. Class diagram for representing mobile point data (extensions highlighted)

To allow for a nested composition of functions, we need to slightly modify the original framework (cf. Fig. 1): in our extended setting, each instance of class `mpoint`$\langle d \rangle$ now aggregates a single d-dimensional function instead of d one-dimensional functions.

Example. Assuming that we have a `Decider`-instance for selecting an appropriate specialization of the root-finding primitive, the code for intersection-finding can be rewritten as follows.

```
Difference diff = new Difference(s, t);
ZeroFinderPred zfp = myZeroFinderDecider.poll(diff);
if ( zfp.eval(diff, begin, end) == true )  /* ... */
```

4 Handling Motion with Known Restrictions

The example at the end of the previous section reduces the problem of intersection-finding to the problem of isolating roots, and there exists a number of numerical and algebraic methods for isolating roots of functions whose mathematical description is known. For example, if both trajectories are given by cubic polynomials, the difference between them is a cubic polynomial as well, and we may use an algebraic approach to implementing a root-finding primitive. If, on the other hand, one trajectory is approximated using a wavelet while the other is approximated using a ν-spline, we have to resort to iterative numerical methods. In Section 4.2, we demonstrate that an iterative algorithm can be guaranteed not to miss any root if we can exploit additional information such as upper bounds on the velocity of both objects. Figure 4 illustrates a simple iteration rule: If $f(t_i)$ and $g(t_i)$ are known, no root of $f - g$ can occur prior to time t_{i+1} which is determined by assuming that f and g move towards each other at maximum speed. The time t_{i+1} at which $f - g$ can have its "next" root is the time of the intersection of bounded-slope segments extending the trajectories of f and g from time t_i onwards—or, equivalently, the root of a bounded-slope linear function extending the trajectory of $f - g$. This method is referred to as L_{one} as it involves one linear function.

If no restrictions are known, the only feasible approach to root-finding is to employ "classical" numerical methods such as Newton's Method, the Secant Method, or Bisection. However, as these methods may fail to produce the roots in chronological order, skip roots, or even fail to converge [35], the correctness of algorithms using them as a primitive cannot be guaranteed.

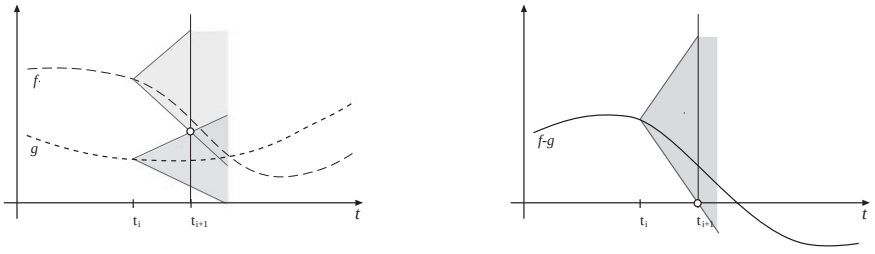

Fig. 4. Intersection-finding by exploiting an upper bound on the velocity

4.1 Modeling Algorithms: The Case of Known Restrictions

By design, our class model does not make any assumption about the type of trajectories (except for assuming continuity). To fully integrate known results for handling classes of trajectories for which additional information is available, we introduce the concept of *restrictions*.[1] A realization of the interface `MotionRestriction` models additional information about a trajectory (or a composition thereof); examples are "real-world" restrictions such as bounds on speed or acceleration given as part of the application context.[2] Such restrictions are defining features for applications involving mobile real-world data and distinguish our setting from related settings involving t-monotone curves.

Figure 5 displays the extension to our class diagram resulting from the inclusion of the concept of restrictions: In addition to the design discussed above, an instance of (a non-abstract specialization of) class `Function` can aggregate any number of instances of realizations of `MotionRestriction`,[3] and different realizations are distinguished by unique identifiers.

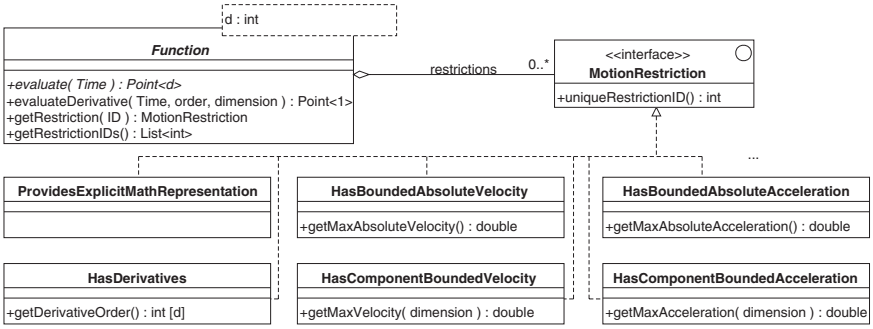

Fig. 5. Additions to the class diagram to model motion with known restrictions

[1] A preliminary version of the approach discussed in this subsection has been presented in our previous work [9].

[2] 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. For obtaining a correctness guarantee, it is sufficient to use conservative estimates for velocity or acceleration.

[3] The data type `double` is to be read as a placeholder for the actual numeric data type used in the application.

To allow for easy access to motion status data, e.g., speed and velocity, we enhance the interface of `Function`, such that derivatives at a given time can be evaluated. Whether or not such information is available, is modeled by class `HasDerivatives`, a realization of `MotionRestriction`.

We also use a realization of `MotionRestriction` to indicate whether or not we may explicitly access a mathematical representation such as the coefficients of a fixed-degree polynomial; this allows a `Decider`-instance to also consider (semi-)algebraic methods and thus to encompass the techniques discussed for the polynomial-based KDS-framework of Guibas et al. [20]. The representation can be accessed using the well-known *Factory* design pattern [16] or using a language-dependent construction, such as the `Java` *Reflection* API [4].

4.2 Designing Primitives: The Case of Known Restrictions

In this section, we continue to consider our running example and focus on primitives for root-finding. The iterative approach L_{one} uses a velocity-based restriction to determine a "next" iteration point t_{i+1} such that $[t_i, t_{i+1}]$ is guaranteed not to contain a root. We show that acceleration-based restrictions can be used to obtain a similar result; an important observation is that such a restriction can lead to more efficient algorithms in the *retrospective* setting than in the *real-time* setting.

As a proof-of-concept we present two restriction-based methods for our running example that we have implemented within our framework. Due to space constraints we omit proofs for their correctness and efficiency; the reader may find these proofs in the thesis of one of the authors [34].

Methods for Root-Finding Using Acceleration-Based Restrictions. Let us assume that the objects subject to collision detection have an upper bound b_{acc} on their acceleration. The earliest collision after time t_i can be computed by assuming that both objects move towards each other with maximum possible acceleration. The resulting "exclusion region" for occurrence of the next possible root is induced by a parabola (see Fig. 6 left), and t_{i+1} can be computed – assuming w.l.o.g. that $f(t_i) > 0$ – as follows [34, Sec. 4.3.1]:

$$t_{i+1} = t_i + \frac{1}{b_{\text{acc}}} \left(f'(t_i) + \sqrt{f'^2(t_i) + 2 \cdot b_{\text{acc}} \cdot f(t_i)} \right) \tag{1}$$

This approach, which we refer to as P_{one} since it involves one parabola, has also been used for collision detection. An alternative, more efficient approach, computes the earliest possible point in time t_{i+1} at which a *second* root may occur. The earliest such occurrence coincides with a double root

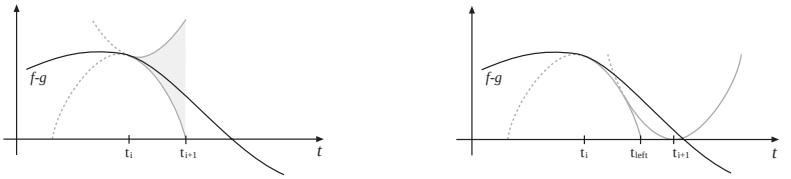

Fig. 6. Intersection-finding by exploiting an upper bound on the acceleration

of the distance function, i.e., the (real-world) objects touch each other. The continuity of motion and speed implies that this point in time t_{i+1} can be computed by first extending $f - g$ by a parabola P as in P_{one}, but then to model the deceleration by an inverted copy $\hat{P}$ of P that continuously extends P such that the vertex of $\hat{P}$ lies on the t-axis (thus inducing a double root) – see Fig. 6 (right). If even a maximal deceleration cannot avoid a collision, the extension of the distance function by $\hat{P}$ from t_i onwards intersects the t-axis, and we choose its first intersection point as t_{i+1}. In both cases, t_{i+1} can be computed in the following, surprisingly simple way [34, Sec. 4.3.1]:

$$t_{i+1} = t_i + \begin{cases} \frac{1}{b_{\text{acc}}} \left(\quad f'(t_i) + \sqrt{2} \cdot \sqrt{f'^2(t_i) + 2 \cdot b_{\text{acc}} \cdot f(t_i)} \right) & \text{if } f'(t_i) \geq \sqrt{2 \cdot b_{\text{acc}} \cdot f(t_i)} \\ \frac{1}{b_{\text{acc}}} \left(-f'(t_i) - \quad \sqrt{f'^2(t_i) - 2 \cdot b_{\text{acc}} \cdot f(t_i)} \right) & \text{else (collision unavoidable)} \end{cases} \tag{2}$$

If $f - g$ has the same sign at time t_i and at time t_{i+1}, no root lies within $[t_i, t_{i+1}]$, and we iterate. Otherwise we can employ Newton's Method to efficiently find the root inside $[t_{\text{left}}, t_{i+1}]$, where t_{left} is determined as in P_{one}.[4] Since this approach involves two parabolas, we refer to it as P_{two}.

Quality and Applicability of the Methods P_{one} and P_{two}. Equations 1 and 2 indicate that the cost of computing the next increment, i.e., the number of arithmetic operations, is almost identical for both methods; the cost is exactly the same if our cost measure is the number of invocations of `evaluate` and `evaluateDerivative`. We analyzed increment and order of convergence:

Fact 1 ([34, Sec. 4.1]) *The increment $\Delta_{P_{\text{two}}}$ is always larger than $\Delta_{P_{\text{one}}}$:*

[4] The correctness of Newton's Method is guaranteed since there is exactly one root inside $[t_{\text{left}}, t_{i+1}]$. Also, due to the bound on the acceleration, the method cannot leave $[t_{\text{left}}, t_{i+1}]$ – for a better understanding of this crucial property, see, e.g., [35].

$$\Delta_{P_{\text{one}}} < \Delta_{P_{\text{two}}} \leq (\sqrt{2}+1) \cdot \Delta_{P_{\text{one}}}.$$

For decelerating compound speed, e.g., when approaching a root, we have:

$$\sqrt{2} \cdot \Delta_{P_{\text{one}}} < \Delta_{P_{\text{two}}} \leq (\sqrt{2}+1) \cdot \Delta_{P_{\text{one}}}.$$

Fact 2 ([34, Sec. 4.2]) *P_{one} and P_{two} both converge quadratically while L_{one} converges linearly.*

However, as both methods use a global bound on the acceleration, their *actual* quality inside some time interval depends on how much the acceleration locally deviates from this global bound. Thus P_{two} is "better" than P_{one}, since it eventually switches to Newton's Method, which then is guaranteed to have quadratic convergence with a (globally) good asymptotic error constant [35]. We conclude that fully exploiting the power of the *retrospective* setting, i.e., being able to move forward and backward in time, can lead to more efficient algorithms than "simply" using known *real-time* algorithmic primitives.

4.3 Generalizations

Combination of Restrictions. We mention in passing that we can also engineer primitives for trajectories that underlie a combination of restrictions [34, Sec. 2.3]; one of these primitives, for example, combines L_{one} and P_{two}. All of these primitives can be implemented using the methods `evaluate` and `evaluateDerivative` provided by class `Function`. All such specializations of a primitive can be incorporated into the framework we have presented: the only necessary modification is the incorporation into the decision process represented by the `Decider`-class associated with the respective kind of primitive.

Applications to Kinetic Data Structures. Many problems in the context of KDSs can be reduced to tracing the relative position of objects and hyperplanes, and algorithms employ root-finding primitives for real-valued functions to check for changes. Our framework thus can be used to extend the concept of KDS to encompass more realistic motion descriptions. Moreover, we can transfer the idea underlying the method P_{two} to the above certificate functions: using a bound on the acceleration we can – in a *retrospective* setting – determine the earliest possible point in time at which the relative position of an object and a hyperplane can have changed twice.

Collision Warning in Multiple Dimensions. The problem of collision warning is to find the time intervals during which the distance between two objects is smaller than some threshold $\varepsilon > 0$. This problem can be solved, e.g., by solving a collision *detection* problem in which one of the objects is extended by a buffer of width ε, see, e.g., [36]. The boundary of this buffer can be seen as a replacement for the hyperplane used in the context of a KDS certificate, and – provided that the description of the buffer is not too complicated – we can again employ a *retrospective* P_{two}-like approach whose efficiency unfortunately diminishes for very small values of ε. A *real-time* P_{one}-like approach is discussed by Hayward et al. [23]. Note that the seemingly more fundamental problem of collision *detection* in a multidimensional spatio-temporal setting reduces to intersection finding of curves in more than two dimensions; for this problem, no (theoretically) efficient algorithms are known – even if the curves are straight lines. This is in contrast to the collision warning setting where (at least for the three-dimensional case) nontrivial algorithms are known [3].

5 Conclusions

We have presented an object-oriented approach to modeling mobile data and algorithms operating on such data. Our model is general enough to capture not only polynomial motion descriptions but also more general (and thus more realistic) descriptions of continuous motion, e.g., of motion restricted only by bounds for the absolute speed or acceleration. In addition to being able to encompass "classical" exact algorithms for polynomials, our approach addresses the problem of numerical robustness and efficiency by modeling and efficiently utilizing motion restrictions. Using algorithmic primitives for collision detection as a proof-of-concept, we have shown how to engineer and to implement efficient algorithmic primitives that exploit such restrictions. A beneficiary side effect of our approach is that these primitives also have a direct applicability in the context of kinetic data structures; thus they extend this concept to encompass more realistic motion descriptions.

References

1. Agarwal PK, Arge LA, Vahrenhold J (2001) Time responsive external data structures for moving points. In: Proc 7^{th} Int Workshop Algorithms and Data Structures (= LNCS 2125), pp 50–61

2. Agarwal PK et al. (2002) Algorithmic issues in modeling motion. ACM Comp Surveys 34(4):550–572
3. Agarwal PK, Sharir M (2000) Pipes, Cigars, and Kreplach: The Union of Minkowski Sums in Three Dimensions. Discrete & Computational Geometry 24(4):645–657
4. Arnold K, Gosling J, Holmes D (2006) The Java™ Programming Language, 4th ed. Addison-Wesley
5. Basch J (1999) Kinetic Data Structures. PhD Thesis, Dept of Computer Science, Stanford University
6. Basch J, Guibas LJ, Hershberger J (1999) Data structures for mobile data. J Algorithms 31(1):1–28
7. Becker L, Blunck H, Hinrichs HK, Vahrenhold J (2004) A framework for representing moving objects. In: Proc 15th Int Conf Database and Expert Systems Applications (= LNCS 3180), pp 854–863
8. Becker L, Voigtmann A, Hinrichs KH (1996) Developing Applications with the Object-Oriented GIS-Kernel GOODAC. In: Proc 7th Int Symp Spatial Data Handling vol I:5A1–5A18
9. Blunck H, Hinrichs KH, Puke I, Vahrenhold J (2004) Verarbeitung von Trajektorien mobiler Objekte (in German). In: Beiträge zu den Münsteraner GI-Tagen, pp 29–41
10. Boissonnat JD, Vigneron A (2002) An elementary algorithm for reporting intersections of red/blue curve segments. Computational Geometry: Theory and Applications 21(3):167–175
11. Bruce R, Hoffmann M, Krizanc D, Raman R (2005) Efficient Update Strategies for Geometric Computing with Uncertainty. Theory of Computing Systems 38:411-423
12. Cheng R, Kalashnikov DV, Prabhakar S (2004) Querying imprecise data in moving object environments. IEEE Trans Knowledge and Data Engineering 16(9):1112–1127
13. Chomicki J, Revesz PZ (1999) A general framework for specifying spatiotemporal objects. In: Proc 6th Int Workshop Temporal Representation and Reasoning, pp 41–46
14. Fabri A, Giezeman GJ, Kettner L, Schirra S, Schönherr S (2000) On the design of CGAL a computational geometry algorithms library. Software – Practice and Experience 30(11):1167–1202
15. Forlizzi L, Güting RH, Nardelli E, Schneider M (2000) A data model and data structures for moving objects databases. In: Proc ACM Int Conf Management of Data, pp 319–330
16. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley
17. Goodman JE, O'Rourke J (eds) (2004) Handbook of Discrete and Computational Geometry. Discrete Mathematics and its Applications, 2nd ed. CRC Press
18. Gudmundsson J, van Kreveld M, Speckmann B (2004) Efficient Detection of Motion Patterns in Spatio-Temporal Data Sets. In: Proc 12th Symp Geographic Information Systems, pp 250–257

19. Guibas LJ (2004) Modeling motion. In: Goodman JE, O'Rourke J (eds) Handbook of Discrete and Computational Geometry. Discrete Mathematics and its Applications, chapter 50, 2nd ed, pp 1117–1134
20. Guibas LJ, Karavelas MI, Russel D (2004) A computational framework for handling motion. In: Proc 6th Workshop Algorithm Engineering and Experiments, pp 129–141
21. Guibas LJ, Mitchell JSB, Roos T (1992) Voronoi diagrams of moving points in the plane. In: Proc 17th Int Workshop Graph-Theoretic Concepts in Computer Science (= LNCS 570), pp 113–125
22. Güting RH, Böhlen MH, Erwig M, Jensen CS, Lorentzos NA, Schneider M, Vazirgiannis M (2000) A foundation for representing and querying moving objects. ACM Trans Database Systems 25(1):1–42
23. Hayward V, Aubry S, Foisy A, Ghallab Y (1995) Efficient collision prediction among many moving objects. Int J Robotics Research 14(2):129–143
24. Kahan S (1991) A model for data in motion. In: Proc 23rd ACM Symp Theory of Comp, pp 267–277
25. Kollios G, Gunopulos D, Tsotras VJ (1999) On indexing mobile objects. In: Proc 18th ACM Symp Principles of Database Systems, pp 261–272
26. Lin MC, Manocha D (2004) Collision and proximity queries. In: Goodman JE, O'Rourke J (eds) Handbook of Discrete and Computational Geometry. Discrete Mathematics and its Applications, chapter 35, 2nd ed, pp 787–807
27. Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AJ (2004) A computational framework for incremental motion. In: Proc 20th ACM Symp Computational Geometry, pp 200–209
28. Pfoser D, Jensen CS (1999) Capturing the uncertainty of moving-object representations. In: Proc 6th Int Symp Spatial Databases, (= LNCS 1651), pp 111–132
29. Pfoser D, Jensen CS (2001) Querying the trajectories of on-line mobile objects. In: Proc 2nd Int ACM Workshop Data Engineering for Wireless and Mobile Access, pp 66–73
30. Pfoser D, Tryfona N (2001) Capturing fuzziness and uncertainty of spatiotemporal objects. In: Proc 5th East European Conf Advances in Databases and Information Systems (= LNCS 2151), pp 112–126
31. Prabhakar S, Xia Y, Kalashnikov DV, Aref WG, Hambrusch SE (2002) Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. IEEE Trans Computers 51(10):1124–1140
32. Roddick JF, Egenhofer MJ, Hoel E, Papadias D, Salzberg B (2004) Spatial, temporal and spatio-temporal databases. Hot issues and directions for PhD research. SIGMOD Record 33(2):126–131
33. Schirra S (2000) Robustness and precision issues in geometric computation. In: Sack JR, Urrutia J (eds) Handbook of Computational Geometry, chapter 14. Elsevier, pp 597–632
34. Sondern J (2005) Nutzung von Bewegungsrestriktionen für Algorithmik in Moving-Objects-Datenbanken. Master's Thesis, Department of Computer Science, University of Münster (in German)
35. Suli E, Mayers DF (2003) An Introduction to Numerical Analysis. Cambridge

36. Trajcevski G, Wolfson O, Hinrichs KH, Chamberlain S (2004) Managing uncertainty in moving objects databases. ACM Trans Database Systems 29(3):463–507
37. Yeh TS, De Cambray B (1995) Modeling highly variable spatio-temporal data. In: Proc 6$^{\text{th}}$ Australasian Database Conf, pp 221–230