Manipulating Interpolated Data is Easier than You Thought

Stéphane Grumbach — INRIA

Philippe Rigaux — CNAM

Luc Segoufin — INRIA

Abstract

Data defined by interpolation is frequently found in new applications involving geographical entities, moving objects, or spatio-temporal data. These data lead to potentially infinite collections of items, (e.g., the elevation of any point in a map), whose definitions are based on the association of a collection of samples with an interpolation function. The naive manipulation of the data through direct access to both the samples and the interpolation functions leads to cumbersome or inaccurate queries. It is desirable to hide the samples and the interpolation functions from the logical level, while their manipulation is performed automatically.

We propose to model such data using infinite relations (e.g., the map with elevation yields an infinite ternary relation) which can be manipulated through standard relational query languages (e.g., SQL), with no mention of the interpolated definition. The clear separation between logical and physical levels ensures the accuracy and the simplicity of data manipulation. Moreover, we show that the evaluation of queries, which includes the computation of the sampling collections and the interpolation functions of the output, can be done very efficiently. The algorithms operating at the physical level are described.

Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

1 Introduction

The extension of database technology to handle spatial applications raises complex problems related to the modeling of data. One of the difficulties pertains to the logical representation of pointsets allowing user-friendly queries while guaranteeing reasonable complexity for query processing.

In the context of spatial databases, most of the efforts in the last decade has been devoted to 2-dimensional data. The recent emergence of new applications, involving for instance *mobile objects*, *Digital Elevation Models* (DEM), or *spatio-temporal* information, requires the representation of data in a space of higher dimension, and constitutes a new challenge to spatial database modeling.

Mobile objects and DEMs constitute excellent examples of a specific type of spatial data defined by interpolation. Interpolated data are either stored explicitly in the database, or computed from the stored data by applying some interpolation function. The trajectory of objects whose position depends upon time is commonly represented by a sample of points with time and position. The full trajectory can be recovered from these points using linear interpolation. Similarly, DEMs provide, at each point (x, y) of the earth surface, the value of a variable h, typically the height above sea level. It can be represented as a finite set of points P along with their elevation. An interpolation based on some triangulation of P gives the value of h at any location.

Many applications require the manipulation of interpolated data: traffic monitoring, intelligent navigation, mobile communication, as well as earth sciences such as geography, meteorology and geology. Consider an application which tracks the motion of aircrafts in a given area. Typical queries are: "retrieve the aircrafts which are in this region", "show the aircrafts which will be in the same region at the same time", "give the altitude of each aircraft with respect to ground or sea level, as well as its position", etc. These queries address simultaneously several kinds of multidimensional information (space, time, altitude), and existing query languages such as SQL seem at first completely inadequate.

The main contribution of the paper is the design of a data model for interpolated data, based on infinite relations manipulated by standard relational means, which meets the classical requirements of logical database modeling: (i) abstract representation independent from the physical storage, (ii) user-friendly query language, and (iii) reasonable complexity of query evaluation. An important aspect of the model is that the queries (including the examples given above) can be expressed with SQL, and evaluated by an algorithm which relies on a few simple geometric primitives.

One of the main aims of the paper is to demonstrate that it is neither desirable nor necessary to give access to the samples and the interpolation functions to the user for data manipulation purposes. As we show this might indeed lead not only to cumbersome queries, where for instance new samples need to be defined, but more importantly, to inaccurate queries, producing wrong output due to wrong interpolations. There is a need for a clear separation between the logical and the physical levels, to which the samples and the interpolation should be relegated.

To satisfy this requirement, we suggest to extend the relational data model to *abstract* infinite relations, containing all the points in the extension of objects, whether stored explicitly as samples, or computed by interpolation. This framework offers sound modeling of interpolated data. For instance, at this abstract level, the interpolation can be formalized with classical dependencies. The base domain (e.g., space for DEMs, and time for moving objects) is the *key* of an interpolated relation, and there is a functional dependency from the key to the interpolated variables. Moreover, any relational query language can be used to query the database. This defines a very simple interface to the end-user, who does not need to worry about complex data structures or geometric algorithms. A simple data format is introduced to represent the abstract relations at the physical level.

The second fundamental issue is efficient query evaluation. It is well known that the complexity of operations on geometric objects increases dramatically with their dimension [6], and that dimension 2 is the most reasonable for practical purposes. Since interpolated data is described in a d-dimensional space, with an arbitrary d, this raises potentially the problem of expensive manipulation of d-dimensional objects.

Fortunately, interpolated data constitutes a very specific type of object. For instance, a trajectory can be viewed as a 1-dimensional pointset embedded in 4D-space, and a DEM as a 2D pointset in 3D space. More generally, if k is the dimension of the base domain (*space*, *time*, etc.), an interpolated pointset with $d - k$ interpolated attributes can be viewed as a k-dimensional object embedded in a d-dimensional space. The intuition of our technique is that one should be able to evaluate queries with geometric algorithms that operate in dimension k, and not in dimension d.

Our second contribution is to identify a class of queries which can be evaluated over interpolated data with a base of dimension at most 2, using efficient 2D geometric algorithms. This shows that it is possible to manipulate efficiently complex data of higher dimension, without specifying how it should be interpolated, and using standard techniques (geometric algorithms, relational optimization, etc.).

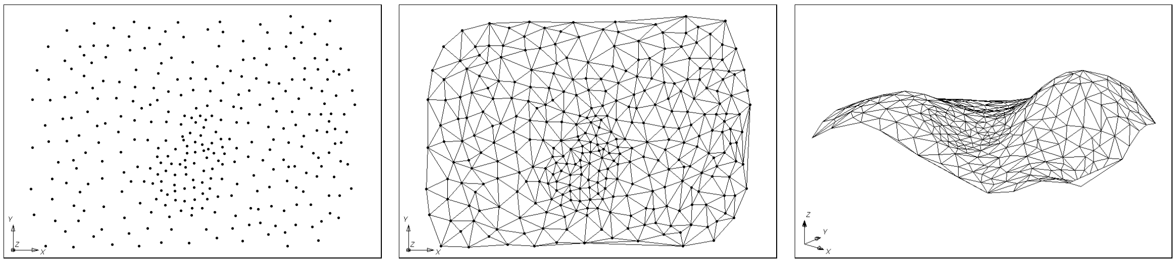
Related work

Interpolated data has been little studied in the database literature. In the area of data modeling, [12] proposes to integrate external interpolation procedures in the SQL language, and to introduce additional clauses to specify new samples. We shall review this approach in Section 2. To the best of our knowledge, no other work aims at integrating in a uniform data model all kinds of interpolated data. However the area of *DEM* is well-established in computational geometry and Geographic Information Systems (GIS) [18]. DEM are used to represent natural phenomena which are continuously variable (temperature, pressure, slope, etc).

Recently the management of mobile objects has received much attention (see for instance the European ChoroChronos Project [5]). The MOST (Moving Objects Spatio-Temporal) model, presented in [16], relies on the concept of *dynamic attributes* which encapsulate motion information (speed + direction) of a mobile object. A prototype, DOMINO [19], has been implemented to test the capabilities of MOST.

An alternative approach is to consider mobile objects as geometric objects embedded in a high-dimensional space, and to specify the appropriate data types and operations [2, 4]. This extends the traditional approach of adding new types to handle geo-referenced objects. In the area of constraint databases [8], some works address the modeling of spatio-temporal data [7, 1]. Relevant research concerns indexing of mobile objects [9, 17, 14] and uncertainty of spatiotemporal information [10, 13].

We investigated the functionalities of some commercial systems regarding the manipulation of interpolated data. Object-relational database systems such as Oracle8 [15] or PostgreSQL [11] provide spatial types (lines, polygons). This clearly relates to the *entity-oriented* modeling of space which focuses on separate entities (parcels, roads, etc), whereas we are rather interested in phenomena that are continuously variable. In geographic applications, this is referred to as *field-oriented* data [20]. To the best of our knowledge, there is currently no support for such data in object-



a. The point sample b. Triangulation c. The TIN

Figure 1: Triangulated Irregular Network

relational databases.

Regarding commercial GIS, there exists a module (ArcTin) dedicated to the management of DEM in the Arc/Info [3] software. It is possible, through a specific set of commands, to combine elevation data with classical maps. However the user needs to be aware of the physical representation in order to apply the proper operations. The Arc/View product proposes a simpler language, *Avenue*, to create scripts that call spatial operations provided by the core. As in Arc/Info, there is no high level query language.

In contrast, the present paper proposes a quite simple logical model which hides to the end user the internal representation of data and the operations which must be applied to this representation to evaluate a query. In addition, the data model integrates naturally mobile objects with spatial data, and constitutes an innovative approach to the modeling of multi-dimensional applications.

The remainder of the paper is organized as follows. Section 2 presents the general issues related to the modeling of interpolated data. We demonstrate the need for a logical level and describe a generalized relational framework. The algorithms are described in Section 3 together with some practically motivated restrictions of the model. Various extensions of this work are presented in the conclusion.

2 Interpolated data modeling

In this section, we consider different approaches to modeling interpolated data, and their impact on the way queries can be expressed.

2.1 Interpolated data

A *Digital Elevation Model* (DEM) is defined as a continuous function in two (argument) variables, that, for convenience, we shall denote x and y, while h denotes the result of the function. The most common example of a DEM is the elevation (i.e., the height above the sea level), but the model is relevant for any location-dependent attribute (e.g., precipitation, temperature, pollution, etc.).

DEM's are generally based on a finite collection of sample values, from which other values are obtained by interpolation. There are various ways to define the interpolation, which mostly depend on the sampling policy. The most interesting is obtained by *triangulated irregular networks*, TIN's.

A TIN is based on a triangular partition of the space with no assumption on the distribution and position of the vertices of the triangles. The elevation value is recorded at each vertex, and inferred at any point P by linear interpolation of the three vertices of the triangle that contains P (see Figure 1).

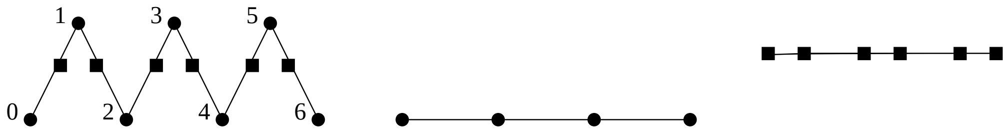
Another category of examples originates in spatio-temporal applications with phenomena depending upon time. The trajectory of a moving object shares some fundamental common properties with the previous example. A trajectory can be seen as a triple of functions (f_x, f_y, f_a) from the time t to x, y and a respectively, where a denotes the altitude. A finite representation can be supported by a set of positions $P_i(x, y, a, t)$. The position at any time t can then be approximated by interpolating the two nearest positions.

So the data we consider are represented by:

- A finite collection of sample points, with their associated value;
- An interpolation function.

The manipulation of such data is a priori not easy. Indeed, it imposes to reconstruct the input data needed by interpolation, but also and more importantly to generate the way the output is itself interpolated. Neugebauer proposes in [12] to integrate external interpolation procedures in the SQL language. The syntax of SQL is extended with (i) aggregate functions, and (ii) so-called "table functions", with the nesting of queries in the **from** clause. Interpolated and non-interpolated data are logically modeled differently, and the user has to manage the interpolation.

Consider the example of a cross-section of a TIN with, say, the plane defined by the equation $x = 50$. We assume that a TIN is represented by a relation *Elevation* which contains a collection of triplets (x, y, h), and an interpolation function *my-function*. The syntax of [12] leads to a query of the type:



a. The initial sample b. A sample on even points c. A sample on middle points

Figure 2: Erroneous samples and interpolation functions

select	y, h
from	Elevation **by method** my–function(h) **entry** (x=50, y= 200) **step** (0, 10)
where	y <= 500

The query corresponds to an operation which can be decomposed in two steps. First the user specifies an *output sample*, i.e., a collection of points where the elevation value is required. This is done with the clause **entry** which defines the starting point, the clause **step** which gives the distance for x and y between two consecutively generated points and the clause **where** which gives a bound for y. In summary, these clauses define a set S of 31 points along the segment $[(50, 200), (50, 500)]$.

In the second step a user-defined interpolation function is applied to the input sample in order to get the elevation value at each point of the output sample. This is done with the clause **by method** which specifies the interpolation function *my-function*.

The explicit specification of the interpolation mechanism in the query syntax raises technical problems. The most important one is the dichotomy between the sampling and the interpolation. First it is the responsibility of the user to associate an interpolation function to a sample stored in the database, and second the result consists of a user-defined sample of points, without any associated interpolation function. The distinct manipulation of a sample and its interpolation function might contradict the proper semantics of the data resulting from the integration of the two components.

This may lead to erroneous results and computation. Consider again the cross section of a TIN according to a new user-defined sample of points and the usual linear interpolation function f. The original data is shown in Figure 2.a. In Figure 2.b, the sampling defined by the user returns the even points of the input sample, using the functions f one obtains a flat line which ignores all the mountains. Again Figure 2.c illustrates a bad choice of output sample: by considering only the middle points in each slope, one obtains a result which is not representative either of the actual data.

Although this can be considered as extreme cases, these examples illustrate how a wrong specification of samples and interpolation is likely to result in an important loss of information. Moreover, the output sample must be guaranteed to be finite, and it is preferable not to leave this responsibility to the user. If we forget the **where** clause in the above SQL query, the execution will generate an infinite sample of points.

The specification of the samples and the choice of the interpolation functions should be made during the data acquisition phase. It has an impact on the precision of the data stored in the database. We show that for relational queries, the system can completely handle the samples and the interpolation, and it is preferable to prevent the user from having to take care of the *correctness* of the specifications.

An immediate consequence is that the system should be able to perform the necessary tasks by itself. In particular, it should generate the appropriate output sample supporting the correct representation of the result, and build this result by exploiting properly the interpolation over the input sample(s). This should be managed in a systematic way by maintaining a strong integration of the elements that constitute the semantics of an interpolated dataset: the sample and the interpolation function.

2.2 Abstract modeling

We show how data modeling can be achieved in respect of these principles with the *abstract data model*. It consists simply in the classical relational model with infinite relations instead of finite relations.

Let us give formal definitions. We consider the universe $\mathbb{R}$ of real numbers. A *(database) schema* s is a finite set of relation symbols together with their arity. An *instance* I of s is an interpretation of the relation symbols by relations of the corresponding arity over $\mathbb{R}$. A *query* is a mapping from instances to relations.

In the abstract model, a DEM will be modeled as an infinite ternary relation, with attributes x, y, h, in which for each tuple over x, y, there is a unique value for h. Similarly, a trajectory is modeled as an infinite 4-ary relation, with attributes t, x, y, a, in which for each value of t, there is a unique value for x, y, a. In both cases we have infinite relations satisfying a functional dependency. We will see the importance of these dependencies in Section 3.

Standard relational query languages can be used to query infinite relations, such as relational algebra or SQL. Assume now that the *Elevation* relation is an infinite set of ternary tuples. The cross-section query discussed above can be simply expressed as follows.

select y, h
from Elevation
where $200 \leqslant y \leqslant 500$
and x= 50

Note that neither the collection of samples nor the interpolation function are visible. They are hidden inside the data. This presents numerous advantages.

- **Correctness** The answer to the query is correct since the user cannot use inaccurate samples, or a wrong interpolation function.
- **User-friendliness** It allows the use of standard query languages, with the usual operations (join, intersection, selection, etc.) and a clear meaning. Moreover, no need to worry about samples and interpolation.
- **Safety** The answer to a query can be infinite. The fact that it can be represented by interpolation over a finite collection of samples is considered in the next section.
- **Uniformity** Relations are all dealt with uniformly, whether finite or infinite, interpolated or not.

We next present several examples of queries, expressed in SQL, which combine the various types of relations in the schema. It is not relevant whether a relation is finite or interpolated. The database can be queried in a purely declarative way. We consider a database schema with the following relations.

- A map $Map(x, y, name)$ which gives (attribute $name$) the ground occupancy ('forest', 'pasture', etc.) at location (x, y). The key is (x, y).
- The trajectory of an aircraft, $Traj(t, x, y, a)$, a being the altitude with respect to the sea level at each time t. The key is t.
- A TIN, $TIN(x, y, h)$ for the height above the sea level. The key is (x, y).

1. Give the altitude and location of the aircraft at time $t1$.

   select x, y, a **from** Traj **where** t = 't1'

2. Show the trajectory of the aircraft with the altitude above the ground at each point:

   select t, t1.x, t1.y, a - h
   from TIN t1, Traj t2
   where t1.x = t2.x **and** t1.y = t2.y

3. Show the forests between 1000 and 2000m.

   select t.x, t.y
   from TIN t, Map m
   where t.x = m.x **and** t.y = m.y
   and $1000 \leqslant$ h $\leqslant 2000$
   and name = 'forest'

4. Show the parts of the trajectory where the aircraft was above the sea, and higher than 10000m.

   select l.x, l.y, l.t
   from Traj l, Map m
   where m.x = l.x **and** m.y = l.y
   and name = 'sea'
   and a > 10000

These queries are very simple to express, in particular because there is no need to think about the internal machinery (interpolation functions or samples). Designing a query just involves viewing the data as infinite relations. This offers a good logical interface to the user's need.

During the evaluation phase, the system must translate automatically a query into operations on the physical storage. Infinite relations need to be encoded in some finite way to be effectively manipulated. The last aspect of the data model is thus a data format which supports the representation of interpolated relations.

2.3 Data format

We distinguish two levels of representation of the data. The *abstract level*, presented above, aims at keeping the essential features of the relational paradigm. The *physical level* supports the finite representation of interpolated relations, and specialized algorithms on this representation.

The finite representation is a compact encoding of all the information pertaining to an interpolated relation, namely the sample and the interpolation function. We first take the example of a TIN. Let $\{T^i \mid i \in [1, n]\}$ be a partition of the plane into finitely many triangles T^i. Inside each triangle T^i, the elevation h can be computed by a linear interpolation from the heights of the three vertices of T^i. In other words, for all points p in T^i we have $h = f_h^i(p)$ where f_h^i is a linear function depending only upon i. Let $t^i(x, y)$ be the Boolean predicate which returns true iff the point (x, y) belongs to T^i. The TIN is symbolically represented by the collection $\{t^i, f_h^i \mid i \in [1, n]\}$ such that:

$$TIN = \{(x, y, h) \mid t^i(x, y) \wedge h = f_h^i(x, y)\},\ i \in [1, n]\}$$

The trajectory of a moving object can be represented in a similar way. The position of the object is recorded at finitely many time points. This leads to finitely many time intervals T^i. The speed of the object is assumed to be constant in each interval. The coordinates of its position at time t are defined by $x = u^i t + x^i$, $y = v^i t + y^i$ and $a = w^i t + a^i$, where i is the index of the interval T^i which contains t, and u^i, v^i, w^i the respective speeds on the axis of the object during that interval. The trajectory $Traj(x, y, a, t)$ can thus be symbolically represented by the collection $\{t^i, f_x^i, f_y^i, f_a^i \mid i \in [1, n]\}$ such that:

$$Traj = \{(x, y, a, t) \mid t^i(t) \wedge x = f_x^i(t) \wedge y = f_y^i(t) \\ \wedge a = f_a^i(t),\ i \in [1, n]\}$$

where $t^i(t)$ is a predicate which is true iff t is in the time interval T^i and where f_x^i, f_y^i and f_a^i are respectively the linear functions $u^i t + x^i$, $v^i t + y^i$ and $w^i t + a^i$.

More generally, let $R(k_1, \ldots k_p, h_1, \ldots h_q)$ be an interpolated relation with key $\{k_1, \ldots k_p\}$ and interpolated attributes $\{h_1, \ldots h_q\}$. We use the following terminology: p is the *interpolation dimension*, $p + q$ the *global dimension* and the *key-space* is the p-dimensional space over the variables $\{k_1, \ldots k_p\}$.

Definition 1 *The finite description of R relies on the following components:*

1. *A partition $\{c^i, i \in [1, n]\}$ of $\pi_{k_1, \ldots k_p}(R)$ in* cells.
2. *A set $f_{h_j}^i, i \in [1, n], j \in [1, q]$ of linear functions. Each $f_{h_j}^i$ is defined only on the cell c^i and gives the value of h_j.*

R is represented as a collection of elements $\langle c^i; f_{h_1}^i, \ldots f_{h_q}^i \rangle$, denoted blocks.

The relations TIN and $Traj$ described above admit such a representation.

3 Query evaluation

This section is devoted to the evaluation of relational queries on interpolated relations. The main characteristics of the techniques, which allow to compute relational operators on the data format, relies on (complex) geometric computation applied to the convex *cells* of the interpolated relations, followed by (simple) propagation to the other attributes using the interpolation function.

As shown in [18], this principle is verified for many classical operations on interpolated data, such as intersection and difference of TIN's, cross-section of TIN's, window and point queries on TIN's, which require only algorithms for data of the interpolation dimension (generally 2). This suggests that the set of interpolated objects defines a subclass which does not need the full power of geometric computation in the global dimension.

The operations on TIN's mentioned above are easily expressed with a relational query on the abstract relations. If we consider these operations one by one, one can easily find an evaluation strategy which is optimal. We design a general strategy which guarantees that queries are evaluated with geometric operators in low-dimensional spaces.

In fact we need to ensure both reasonable evaluation complexity and query closure. The first requirement implies working on databases consisting of interpolated relations with the interpolation dimension bounded by 2. Indeed, the time complexity of manipulating objects in dimension 2 is low, and the algorithms are rather simple to implement. To this end we require that the keys k_i and k_j of two distinct relations are *orthogonal*, that is either $k_i \subseteq k_j$ or $k_i \cap k_j = \emptyset$. The second requirement of query closure means that queries should deliver interpolated relations. In order to achieve this, we need to restrict the relational queries considered. First we concentrate on conjunctive queries to avoid negation. Second we require that the key of the output relation is a key of some of the input relations. We call *key-restricted* such queries. This is fundamental for the computation which aims at preserving the functional dependencies, and needs to identify the key during the query evaluation process.

In summary, we study the evaluation of conjunctive queries mapping an interpolated database to an interpolated relation, such that the interpolation is always based on one of the well-identified key-spaces of the input schema. Typically these key-spaces will be *space*, *time*, etc. Consequently, the evaluation of queries can be reduced to some operation on the key attributes, followed by an easy propagation to the interpolated attributes via the proper linear interpolation function.

The algorithms relies on a small set of primitives which operate on the *cells* of the *key-space*, and the geometric part of query evaluation is reduced to these primitives. The list of primitives is given below.

1. **Projection**, $proj_v(c)$, projects the cell c on the variable v.
2. **Intersection**, $inter(c_1, c_2)$, computes the cell corresponding to the intersection of c_1 and c_2.
3. **Range**, $range(c, f)$, for any linear function f and convex cell c, computes the interval $[f_{min}, f_{max}]$, image of c by f.

In dimension 2, these primitives enjoy a very low time complexity (and can be found in most systems handling spatial data). For instance, it suffices to scan the n vertices of a convex polygon to compute *proj* and *range*. The intersection of convex sets in dimension 2 is an easy task, since several simple algorithms compute the intersection of convex polygons in $O(n)$ [6]. Now, the algorithms exploit the interpolated form of the data in order to organize the evaluation as a combination of these primitives.

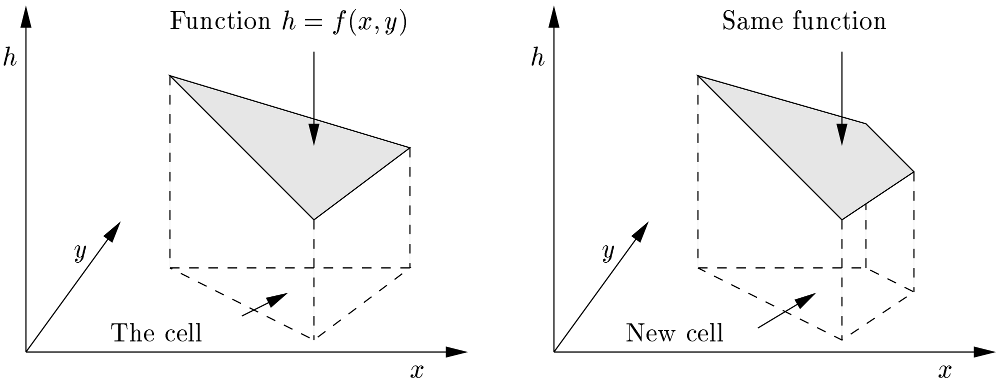


Figure 3: Evaluation of $q = \sigma_{\alpha_x x + \alpha_y y + \alpha_h h = \beta}(TIN)$

We illustrate this intuition with several examples. They are all based on a simple schema with two relations: $TIN(x, y, h)$ which gives the height above the sea, and $Traj(t, x, y, a)$ which describes the trajectory of an aircraft, x, y being the usual variables related to the 2D-space, and a the altitude (w.r.t the sea level). The keys are (x, y) for the relation TIN and t for the relation $Traj$, all the other attributes being interpolated from their respective key.

Selection

The following example illustrates how the selection can be done efficiently by substitution techniques.

Example 1 Consider the query q with the selection $q = \sigma_{\alpha_x x + \alpha_y y + \alpha_h h = \beta}(TIN)$. This computes the part of the TIN which is contained in a sub-space H defined by $\alpha_x x + \alpha_y y + \alpha_h h = \beta$, an operation which a priori involves a 3D object. Let s be one of the blocks of TIN, represented by

$$\langle c^s ; f_h^s \rangle$$

where c^s is the cell (triangle) in the key-space corresponding to the block s and f_h^s the linear function interpolating h on c^s.

Selecting the points of the object represented by s which are *also* in H can be achieved by considering those points in H such that $h = f_h^s(x, y)$. One can thus substitute h by $f_h^s(x, y)$ in the equation of H. This defines a new object H^s as the set of points (x, y) which satisfies $\alpha_x x + \alpha_y y + \alpha_h f_h^s(x, y) = \beta$: a convex object in the 2D key-space.

Finally, the selection is performed by an iteration on the set of blocks which performs at each step the operation $inter(c^s, H^s)$ and returns the block:

$$\langle inter(c^s, H^s); f_h^s \rangle$$

The interpolation function remains unchanged, while an intersection is computed upon the key attributes x and y with the *inter* primitive.

This is depicted in Figure 3. A block s can be seen as consisting of two parts: first a cell $c(x, y)$ in the 2D space defining a "cylinder" in the 3D space, second this cylinder is "cut" by a hyperplane (the function $h = f(x, y)$). Whenever a selection is performed, it suffices to evaluate the operation on the cell. For instance the selection $\sigma_{\alpha_x x + \alpha_y y + \alpha_h h = \beta}(s)$ defines a new object s' as a restriction of the initial convex polygon. This yields a new cylinder, cut by the *same* function f. □

We denote as σ^{subst} this operation. It illustrates a simple substitution technique, based on the interpolation function, which reduces the 3D operation of a selection into a primitive applied to the cells.

Join

We now examine the techniques for more involved queries featuring selection, join and projection. We start with an example of a join which can be processed using substitution techniques similar to the previous example.

Example 2 Consider a query giving the values of t, x and y when the aircraft was over a ground with altitude over 1000. The output key is t.

select	t, Traj.x, Traj.y
from	TIN, Traj
where	h $\geqslant$ 1000
and	TIN.x = Traj.x **and** TIN.y = Traj.y

The algebraic expression is

$$\pi_{t,x,y}\ (Traj \bowtie \sigma_{h \geqslant 1000}(TIN))$$

First the selection can be evaluated with σ^{subst}, as before. The query involves then a join which delivers pairs of blocks $[r, s]$, from TIN and $Traj$ respectively, of the following form:

$$(r) \qquad \langle c^r(x, y)\ ;\ h = f_h^r(x, y) \rangle$$

$$(s) \qquad \langle d^s(t)\ ;\ x = g_x^s(t)\ ,\ y = g_y^s(t)\ ,\ a = g_a^s(t) \rangle$$

where c^r and d^s are respectively a triangle in the space (x, y) and a time interval. It is possible to drop out the terms $h = f_h^r(x, y)$ and $a = g_a^s(t)$, which do not play a role in the query anymore. The join operation creates then structures, associating blocks, of the form:

$$[c^r(x, y)\ ;\ d^s(t)\ ;\ x = g_x^s(t)\ ,\ y = g_y^s(t)]$$

The join can be evaluated by first performing a general substitution that replaces the variables x and y according to their interpolated definition. One obtains structures of the form:

$$[d^s(t)\ ;\ c^r(g_x^s(t), g_y^s(t))\ ;\ x = g_x^s(t)\ ,\ y = g_y^s(t)]$$

By computing $d' = inter(d^s(t), c^r(g_x^s(t), g_y^s(t)))$, one gets the result of the join as a block in the appropriate format $\langle d'(t); x = g_x^s(t), y = g_y^s(t)\rangle$. Note that the interpolation functions for x and y remain unchanged. □

So far we have simply exploited the functional dependencies to perform substitutions. We develop a more involved example of application of the substitution mechanism within the next example.

Example 3 Consider the query computing the ground level under the trajectory of the aircraft. One obtains a *profile curvature* which gives the altitude of the ground at each point of the polyline on $[x, y]$. The query is:

select TIN.x, TIN.y, h
from TIN , Traj
and TIN.x= Traj.x **and** TIN.y = Traj.y

The algebraic expression is $\pi_{x,y,h}(TIN \bowtie Traj)$. The query is very similar to the previous one, but the final key is $[x, y]$. We therefore need to rewrite the intermediate structure associating blocks using the dependency functions in order to get the appropriate key. As in the previous example the intermediate structure has the following representation:

$$[c(x, y)\ ;\ h = f_h(x, y)\ ;\ d(t)\ ;\ x = g_x(t)\ ,\ y = g_y(t)]$$

All the information pertaining to t is an interval $d \equiv t_{min} \leqslant t \leqslant t_{max}$ and two functions $x = g_x(t)$ and $y = g_y(t)$. These functions[1] can be put in the form $t = g_x^{-1}(x)$ and $t = g_y^{-1}(y)$, and this allows to rewrite the system as follows:

$$[c(x, y)\ ;\ t_{min} \leqslant g_y^{-1}(y) \leqslant t_{max}\ ;$$
$$g_x^{-1}(x) = g_y^{-1}(y)\ ;\ h = f_h(x, y)\ ;\ t = g_y^{-1}(y)]$$

[1]Throughout this presentation, we assume that no degenerate case occurs. In particular, a linear function can be inverted, and two distinct functions are assumed to be linearly independent.

The representation $t_{min} \leqslant g_y^{-1}(y) \leqslant t_{max}\ ;\ g_x^{-1}(x) = g_y^{-1}(y)$ describes a segment seg in the 2D space. In order to get the correct result of the join in the right data format it suffices to compute $inter(c, seg)$. Finally one removes the last term ($t = g_y^{-1}(y)$) to complete the projection. □

We can now explain the general computation required for a join operation. It should be noted that *any* join involving two interpolated relations can be reduced to a join involving the key-spaces. Indeed, we can substitute, in each block, any interpolated variable by the proper function on the key. So, during a join $R_1 \bowtie R_2$, the computation is based on intermediate structures of the form:

$$[c^r(x, y)\ ;\ c^s(u, v)\ ;\ f_1, \cdots, f_n]$$

where $c^r(x, y)$ and $c^s(u, v)$ are cells from respectively R_1 and R_2, and the f_i's are linear functions over the variables (x, y, u, v). These functions define a *mapping*, M, which associates the key-space of R_1 to the key-space of R_2 (recall that the key-spaces are orthogonal).

We denote by A the subset of the points of c^r whose image by M intersects c^s, and by B the subset of the points of c^s which are image by M of some point of c^r (see Figure 4.a). Computing A and B (and keeping the functions) is sufficient to evaluate the join. Intuitively, the points in A and B are in relationship via the mapping M, and thus qualify to the join semantics.

Let us assume first that the mapping M contains at least two functions (one for each coordinate). In that case, the first two functions can be rewritten as :

$$m = \begin{cases} u = f(x, y) \\ v = g(x, y) \end{cases}$$

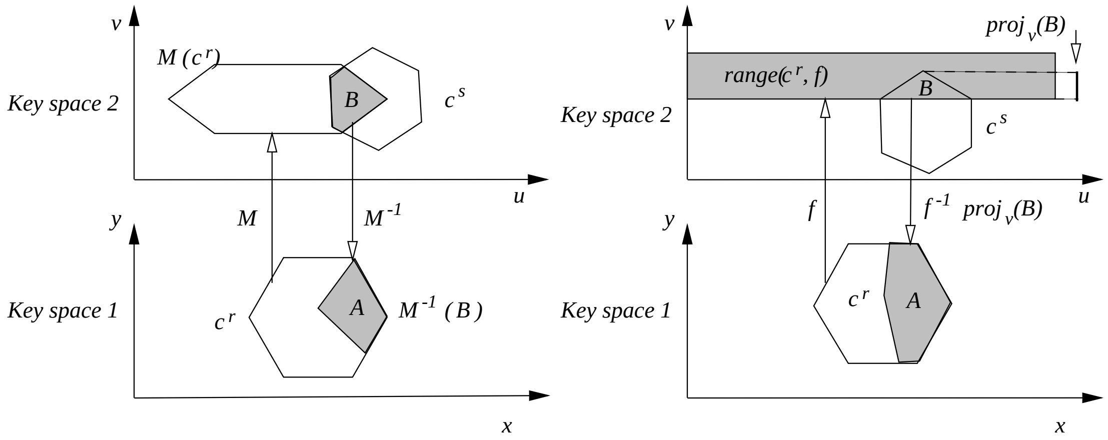
where m defines a one-to-one linear application between the two key-spaces. Now, in the remaining functions, u and v can be replaced by $f(x, y)$ and $g(x, y)$, and this yields a cell $c(x, y)$. The sets A and B can thus be easily obtained as follows:

(1) compute $o = inter(c^r, c)$,

(2) compute $B = inter(m(o), c^s)$,

(3) compute $A = m^{-1}(B)$.

We consider now the case where there is only one function that links two key-spaces of dimension 2: M defines in that case a "loose" connection between the two key-spaces. Imagine, for instance, that the association between the space (x, y) and the space (u, v) is reduced to the function $v = f_v(x, y)$. The computation relies on $range$ and $proj$: first one computes $range(c^r, f_v) = I = [v_{min}, v_{max}]$, and $B = inter(I, c^s)$. Then A is obtained as $inter(f^{-1}(proj_v(B)), c^r)$: see Figure 4.b.



a. The mapping between key-spaces

b. A mapping with a single function.

Figure 4: Evaluation of a join

We denote by JoinCells this operator. It takes as input two cells from two distinct key-spaces, a set of functions, and constructs the sets A and B. Essentially, the techniques consist of some simple manipulations from linear algebra to compute the image of a cell by a system of linear functions, and the primitives *inter*, *range* and *proj*. The join algorithm can be described as a simple nested loop which evaluates JoinCell at each step.

The following example gives a new, general strategy, based on the JoinCells algorithm, for evaluating the queries presented in the Examples 2 and 3.

Example 4 We focus on the computation of the join $TIN \bowtie Traj$. The algorithm JoinCells operates on a cell $c^r(x, y)$ from TIN, $d^s(t)$ from $Traj$, and the set F of functions $\{g_x, g_y\}$ which link the key-space *time(t)* to the key-space *space(x,y)*. In order to evaluate the join, we compute JoinCells(c^r, F, d^s), which yields $A = inter(d^s(t), c^r(g^s_x(t), g^s_y(t)))$ and $B = inter(c^r(x, y), seg(x, y))$ (seg is the segment defined by $t_{min} \leqslant g_y^{-1}(y) \leqslant t_{max} \ \wedge \ g_x^{-1}(x) = g_y^{-1}(y)$).

By replacing d^s with A, and c^r with B, one obtains the following result:

$$[A \ ; \ x = g_x(t) \ , \ y = g_y(t), a = g_a(t) \ , \ h = f_h(x, y) \ ; \ B]$$

So we computed the correct values on the cells, and kept the interpolation functions unchanged. From this result we can either project on (t, x, y), by removing B, g_a and f_h, or on (x, y, h) by removing A, g_x, g_y and g_a. We get the results of Examples 2 and 3. □

We can directly relate the geometric complexity of queries to the interpolated dimension of the relations involved in the query. The intersection of two trajectories for instance does not even require a computation in the 2D space: operations on time intervals suffice. With opportune restrictions preventing the creation of non-interpolated objects, it is possible to extend the set of geometric primitives (with 2D union and 2D difference for instance) in order to be able to evaluate more queries.

4 Conclusion

We investigated the modeling of several innovative applications involving for instance mobile objects or digital elevation maps. Our goal has been to develop user-friendly and efficient query languages, based on standard techniques. We demonstrated the need for a logical modeling clearly separated from the physical level. In particular, we proved that the collections of samples together with the interpolation functions should belong to the physical level, and be hidden from the user, therefore resulting in query languages without special primitives.

One of the contributions of the paper is to show that such an approach is possible with simple formalization of these data in the relational framework, together with a data format which provides a compact representation and supports the evaluation of relational queries. A striking aspect of the model is the ability to express SQL queries on geometric data without any geometric skill.

A challenging issue in this context is the complexity of query processing, which strongly depends upon the dimension. Hence, a crucial property of query languages for multidimensional data should be the independence of the complexity of query evaluation from the dimension of the embedding space. Fortunately, interpolated objects can be manipulated at a cost which depends essentially upon the dimension of the key-space, and not of their global dimension. We described a set of algorithms which are used for query evaluation.

The present approach offers a great potential for optimization. First the use of standard relational query languages allows to rely on existing techniques. More importantly, the physical data independence leaves full control to the system over the definition of the sample collections. In some cases it allows the use of indexes in the collection of samples that would not be possible if they were user-defined.

The current setting assumes *keys* of dimension at most 2. This restriction is motivated by complexity reasons. Indeed, objects of dimension 2 can be efficiently manipulated and the implementation of the operators is relatively simple. It is straightforward to generalize the ideas to keys of higher dimension $k > 2$. In this case, the evaluation is performed using operators in dimension k.

Acknowledgments. We are very grateful to Robert M. Wallace who provided the illustrations of Figure 1. We are also indebted to Marlon Dumas, Michel Scholl and Victor Vianu for their useful comments on earlier drafts of this paper.

References

[1] J. Chomicki and P. Revesz. A Geometric Framework for Specifying Spatio-Temporal Objects. In *Proc. Intl. Workshop on Time Representation and Reasoning*, 1999.

[2] M. Erwig, R.H. Güting, M. Schneider, and M. Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3):269–296, 1999.

[3] ESRI, editor. *Understanding GIS: the Arc/Info method*. ESRI Press, 1996.

[4] L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 2000.

[5] A.U. Frank, S. Grumbach, R. H. Güting, C.S. Jensen, M. Koubarakis, N.A. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, T.K. Sellis, B. Theodoulidis, and P. Widmayer. Chorochronos: A Research Network for Spatiotemporal Database Systems. *SIGMOD Record*, 28(3):12–21, 1999. http://www.dbnet.ece.ntua.gr/~choros/.

[6] Jacod E. Goodman and Joseph O'Rourke. *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[7] S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-Temporal Data Handling with Constraints. In *Proc. Intl. Symp. on Geographic Information Systems*, 1998.

[8] P. Kanellakis, G Kuper, and P. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995. A shorter version appeared in PODS'90.

[9] G. Kollios, D. Gunopolos, and V.J. Tsotras. On Indexing Mobile Objects. In *Proc. ACM Symp. on Principles of Database Systems*, pages 261–272, 1999.

[10] M. Koubarakis. The Complexity of Query Evaluation in Indefinite Temporal Constraint Databases. *Theoretical Computer Science*, 171(1/2), 1997.

[11] Bruce Momjian. *PostgreSQL, Introduction and Concepts*. Addison Wesley, 2000. To appear. See `http://postgresql.org`.

[12] L. Neugebauer. Optimization and Evaluation of Database Queries Including Embedded Interpolation Procedures. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 1991.

[13] D. Pfoser and C.S. Jensen. Capturing the Uncertainty of Moving-Object Representations. In *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, pages 111–132, 1999.

[14] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 2000.

[15] J. Sharma. Oracle8i Spatial: Experiences with Extensible Databases. An Oracle Technical White Paper, May 1999.

[16] A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 422–433, 1997.

[17] Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento. On the Generation of Spatiotemporal Datasets. In *Intl. Conf. on Large Spatial Databases (SSD'99)*, 1999.

[18] M. van Kreveld. Digital Elevation Models and TIN Algorithms. In *Algorithmic foundations of Geographic Information Systems*, number 1340 in LNCS, pages 37–78. Springer Verlag, 1997.

[19] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, and S. Chamberlain. DOMINO: Databases fOr MovINg Objects tracking. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 547–549, 1999. (Demo sessions).

[20] M. Worboys. *GIS: A Computing Perspective*. Taylor and Francis, London, 1995.