Building a Constraint-Based Spatial Database System: Model, Languages, and Implementation

Philippe Rigaux[a*] Michel Scholl[b,c] Luc Segoufin[c] Stéphane Grumbach[c]

[a] LRI, Bât. 490, Université Paris-Sud, 91400 Orsay, France, rigaux@lri.fr
[b] CNAM, 292 rue St Martin 75141 Paris Cedex 03, France, scholl@cnam.fr
[c] INRIA, BP 105, 78153 Le Chesnay Cedex, France, {luc.segoufin,stephane.grumbach}@inria.fr

Abstract

This paper presents DEDALE, a spatial database system which provides an abstract and non-specialized data model and query language for representating and manipulating geometric data in arbitrary dimension. DEDALE relies on a logical model based on linear constraints. The main features of the constraint model are (1) a uniform representation of all sorts of data, including geometric, spatio-temporal or elevation data, (2) an algebraic query language whose formal foundations constitute a basis for practical query optimization.

We show the practical relevance of the approach by describing an implementation which builds on standard technology for data storage, database indexing and on the parsing and optimization of SQL. DEDALE validates the linear constraint model over various applications, proposes a user query language based on SQL which allows to query the database in a purely declarative way, and gives some first steps towards query optimization.

We believe that this experience is a fruitful step toward sound and consistent database models which hide the complexity of arbitrary geometric data, while keeping manipulation languages intuitive and efficient.

1 Introduction

Spatial data has emerged as central to a large range of applications including geographic information systems (GIS), computer-aided design (CAD), robotics, etc. Off the shelf database management systems (DBMS) are not appropriate for spatial data handling, in particular because of the complex structure of geometric information and of the intricate topological relationships among sets of spatially related objects. Moreover, the costly operations involved in geometric object management seem, at first glance, to prevent a logical-level approach based on simple data structures (e.g., relations) manipulated through a limited set of simple operations (e.g., relational algebra). Therefore, in the past, the main trend for handling spatial data has been to introduce, as additional logical components of the database model, carefully designed geometric structures associated with computational geometry algorithms in order to provide an efficient evaluation of spatial operations [45].

Most of the existing Geographic Information Systems (GIS) rely on a strict distinction between classical alpha-numerical data and geometric data. For instance, Arc/INFO [33] extends an efficient module for manipulating geometric and topological information, called Arc, with a relational module, called INFO,

*This paper has been written while the author was affiliated to the CNAM institute.

for manipulating alphanumeric information associated with geometric objects. Several commercial DBMS such as Oracle and DB2 provide in their latest versions a separate module devoted to spatial data. Lack of uniformity, restrictions on data structures or on the dimension of the embedding space implying the addition of new types whenever new requirements arise, and finally complexity of the data model are some difficulties raised by this approach.

The constraint data model, first introduced by Kanellakis, Kuper and Revesz [26] offers a powerful paradigm for the representation and querying of geometric data. The basic idea is to see spatial objects as infinite sets of points, and to provide a finite representation of these infinite sets, with constraints over a dense domain. From the user's point of view, queries can be expressed as if the infinite extension was stored in the database. Considering a polygon *explicitly* as an infinite set of points rather than through its *implicit* definition by its sequence of vertices allows us to manipulate spatial objects through standard set operations (e.g. those of relational algebra).

A strong feature of the model is the ability to represent and manipulate data in arbitrary dimension within a uniform framework. This advantage is well illustrated by the new requirements of some emerging applications handling 3D or 4D data, such as *spatio-temporal objects* and *elevation data*. In the first case we are mostly interested in *moving objects*, i.e., objects whose position in the 3D space depends on time. A trajectory can be approximated by a sequence of connected segments in the 4-dimensional space whose axes are respectively the x and y coordinates of the earth's surface, the altitude and the time dimension t. In the second case, *elevation data* can be viewed as a function h of x and y where $h(x, y)$ is any position-dependent value, such as the average precipitation or the height above the sea level.

Among the difficulties that arise when trying to put into practice the constraint data model, the following are worth mentioning:

1. **Integration of alphanumerical and geometric data**. Although the constraints allow to handle uniformly finite and infinite instances, in practice we often have to associate atomic attribute values (e.g., the name, departure and arrival cities of an aircraft) with infinite sets of points (e.g., the trajectory of this aircraft). The model should therefore be able to treat such sets of points as first-class citizens.

2. **User query languages**. The model is founded on formal query languages which are not convenient for end user's querying. Designing a simple end-user language, encompassing alpha-numeric and spatial query capabilities, is not a trivial task.

3. **Implementation issues**. Finally the suitability of such a model and query language must be assessed by the design and implementation of a system.

This paper reports on the development of such a system, DEDALE, as a practical answer to the above questions.

First DEDALE relies on a non-first normal form (N1NF) model with one level of nesting.[1] Alphanumeric information is easily represented in unnested attributes whereas set nesting is used for spatial data. Second the designed query language benefits from the solid data model foundations, is not application specific and does not require expertise. It allows the user to express complex spatial queries, in a declarative manner, without having to bother with the algorithmic aspects of the queries.

Finally, a major part of the paper is devoted to the report of the implementation of DEDALE. It relies on the BASIS system, an open database engine initially designed for the experimentation of spatial indexing and query processing. The modules specific to DEDALE include a library of constraint operations that manipulate sets of points, a parser for the SQL-like query language, and an optimizer.

[1] A constraint-based N1NF model has also been independently proposed in [3].

Part of the material reported in this paper has appeared elsewhere in extended abstracts and conference proceedings papers that describe part of the design at earlier stages of the project. A first version of the implementation of the constraint engine was presented in [12]. The data model and a first version of the query language was proposed in [15]. A spatio-temporal application running on DEDALE has been described in [14] and the chapter [13] in [28] describes an O_2-based implementation. Optimization rules are from [15], further developed in [18]. The current paper reports on the full design and implementation of the system. Some features, such as the current SQL-like version of the query language and more importantly the current implementation using the BASIS kernel, are novel.

The paper is organized as follows. The motivation for a new data model encompassing large classes of geometric data is discussed in Section 2. We explore the requirements of new applications – in particular those handling spatio-temporal data – for which the current technology runs short. Section 3 introduces the DEDALE data model, together with an algebra. The user query language, along with some queries which illustrate the power of the language in the context of various applications, is described in Section 4. The architecture and implementation of DEDALE is presented in Section 5.

2 Handling geometric data

In this section we illustrate some fundamental issues related to spatial data management by investigating examples taken from the areas of geographic data management, field-based information management and moving points. In each case we successively consider the problems of data modeling, finite representation and we give examples of some useful operations.

2.1 Entity-based data

A common modeling of geographic objects – such as, for instance, a road network, or a set of land-use parcels – distinguishes two parts: (1) an alphanumerical *description* and (2) a *spatial component* which corresponds to the shape and location of the object in the embedding 2-dimensional space [31, 49, 38]. The *description* of a road includes for instance its name and its number of lanes, whereas its spatial component consists of the set of points describing the road's geometry. This description can be viewed as an identifier for the set of points of its geometry.

A practical rationale behind this distinction is that the management of geometric information goes beyond the capabilities of standard DBMS. More precisely, a naive representation in a relational DBMS of the set of points describing a geographic object would result in an infinite instance. Therefore the fundamental issue is *how* to represent in a finite concise way this infinite set of points, yet preserving a satisfactory integration with existing data structures (e.g., tables) and query languages (e.g., SQL).

A traditional answer to the problem is to enrich the relational data model with new abstract data types (ADT) encapsulating geometric data structures and operations. While the definition of "geometric data" as a set of points in the Euclidean space is quite general, this approach considers practical (one might say, application-driven...) needs and uses one of the following types. *Points* are used for representing the location of entities whose shape is not considered as useful (cities, crossings, churches). *Linear objects* are commonly used for representing networks (roads, hydrography). Finally regions or *surfacic objects* are mostly used for representing entities with large areas such as administrative units or parcels. There is no common agreement on the set of operations which must be provided in a spatial ADT based model, except perhaps for topological relationships [6]. The models vary depending on the targeted application [20, 21, 39, 22].

The ADT approach has been adopted in some relational DBMS, including ORACLE [42] or Postgres [32]. For concreteness, Table 1 shows the main types in the Postgres system (without the associated operations).

Geometric Type	Representation	Description
point	(x, y)	Point in space
line	$((x_1, y_1), (x_2, y_2))$	An infinite straight line defined by two points.
lseg	$((x_1, y_1), (x_2, y_2))$	A line segment defined by its two end-points.
box	$((x_1, y_1), (x_2, y_2))$	A rectangle (rectangular box).
path	$[(x_1, y_1), ...]$	A polyline (list of pairwise connected segments).
polygon	$((x_1, y_1), ...)$	Polygon.

Table 1: A subset of geometric types in Postgres

It is not straightforward to decide which operations must be associated with such a geometric type system. This raises several difficult issues such as *closure* (the intersection of two polygons can be a *set* of polygons or, more generally, a set of polygons, polylines and points), consistency (is there a sound definition of a consistent set of geometric operations), redundancy (can an operation be defined by means of other operations), etc. Here again the choice is mostly motivated by practical requirements. The set of chosen operations typically includes: test whether a point belongs to a polygon, test whether a rectangle intersects a polygon, compute the area of a polygon or the length of a polyline, etc.

Given these types and operations, a relational table can be created with geometric attributes using the standard SQL ANSI syntax. We shall consider the following tables to illustrate the querying features of this extended relational data model.

1. *Counties (idCounty: int, nameC: string, geomC: polygon)*. A table that stores information on the counties in the US, including the geometric description of their area.

2. *Roads (idRoad: int, nameR: string, geomR: path)*. A table that stores the description of roads in the US.

Spatial queries are expressed by introducing geometric operators in the **where** clause of SQL. For instance the following query selects all the roads that go through the county of Dubois.

select	R.*
from	Counties C, Roads R
where	geomC **overlaps** geomR
and	nameC = 'Dubois'

Although simple enough, this approach is somewhat limited by the restrictions of the data model. In the case of the Postgres type system for instance, it is not possible to *compute* the part of a road that lies inside the county, as this part potentially consists of several components (in other words, it might be a set of polylines), and the existing representation is not powerful enough. This shortcoming also restricts the possibility of *composing* geometric operations. For instance, asking for these roads whose portion inside the county has a length greater than 100 Kms is not possible — this would require applying the *length* operation on the intersection of the road and the county.

There exists data models more powerful than Postgres only used to illustrate the essential features of this approach:

- The model relies on somewhat arbitrary choices for abstract data-types and associated operations. These choices are made according to the type of applications the system is devoted to. New application types will call for new ADTs whose closure will be harder and harder to maintain. This approach is also difficult to extend to dimensions higher than 2. This is why, apart from a few quite recent proposals (which do not go beyond dimension 3), most of the ADT-based data models limit the dimension to 2, which is sufficient for geographic data, but not for 3D objects or spatio-temporal objects.

- The elaboration of a query depends on the data type. In the case of alphanumerical data, the database is viewed as a set of items. Elements of this set that satisfy some property are extracted from this set. In the case of spatial data on the other hand, the spatial data type is a black box supporting a given set of operations. Expressing a query means choosing, and sometimes compose the proper operation(s). This heterogeneity potentially leads to cumbersome, not completely declarative and not generic query languages.

In spite of their limits, the ADT models are appropriate in the context of geographic applications which served as their first motivation and thus strongly influenced their design. Beyond geographic data, there exists many areas where the above-mentioned features become a restriction, either because space is no longer interpreted as a finite set of spatial objects, or because the dimension is no longer restricted to 2, or both.

2.2 Field-based data

In many scientific applications, space is seen as a continuous field, and the information of interest – temperature, pressure, elevation – is obtained, at each point, as the result of a measure. A *Digital Elevation Model* (DEM) is defined as a continuous function h in two (argument) variables, that, for convenience, we shall denote x and y. 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.).

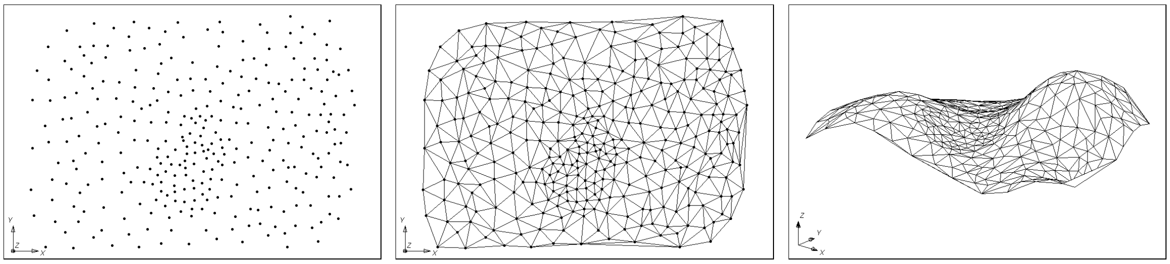
This view of space as a continuous field is in contrast with an entity-based interpretation which identifies a set of points as an object. The data of interest consists solely of an infinite set of geometric points, without any associated descriptive information. Finally observe that, because the description of a DEM has three variables, x, y and h which take their values in a dense domain, the dimension of the embedding space is 3.

The representation of DEMs, and their manipulation, is therefore quite different from that of geographic objects as described above. Let us first consider the usual finite representation. Instead of using a predefined set of geometric types (points, lines, polygons), DEMs 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 one is that of *triangulated irregular networks* (TIN). A TIN is based on a triangular partition of the space with no assumption on the distribution and location 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).

Typical operations on TINs [47] include *point location* ("give the elevation at this point"), *windowing* ("Give the part of the TIN which intersects a rectangle"). A more specific operation is the addition or the subtraction of TINs. Assume for instance that a TIN T_1 gives the height of the surface above the sea, while T_2 gives the height of some geological layer (say the groundwater) above the sea. By subtracting T_2 from T_1, one obtains the height of surface above the geological layer.

Field-based data is ignored in the current database technology. Specialized GIS softwares (Arc/Info for instance [7]) provide specific modules, including command languages to apply operations. In this setting



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

Figure 1: Triangulated Irregular Network

the user has to make explicit references to TIN operations while writing the query. The design is thus far from the declarativeness which is expected from high-level and general database systems. It is also far from the data independence principle of databases which says that the physical implementation (being a TIN for instance) should be hidden to the end-user.

2.3 Moving points

Spatio-temporal applications deal with spatial phenomena depending upon time. For instance, an object trajectory can be seen as a triple of functions (f_x, f_y, f_a) from time t to x, y and a respectively, where a denotes the altitude.

The trajectory of a moving object shares some common properties both with entity-based and field-based modeling. On the one hand, we have to deal with "objects". Each object is distinguished by its identity, and characterized both by alphanumerical attributes and by a geometric description of its trajectory. On the other hand this geometric information is particular: it is based on four variables (time t, position in x and y, altitude a), but three out of the four variables, namely (x, y, a) can be obtained as a continuous function of the fourth, t.

A finite representation of the trajectory 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 in time. This is illustrated in Fig. 2. For clarity, we restrict ourselves to a trajectory in the (x,y) plane: we omit the altitude a. We also restrict the moving object to be represented by a point (in contrast to a line or a polygon) whose location is a function of time. At time $t = 0$, the moving point is located at point P_0. It then moves from P_0 to P_3, which it reaches at time $t = 5$, via P_1 ($t = 1$) and P_2 ($t = 2$).

Basic operations on moving points are: computing the position of an object o at a given time t_1, searching for the time when o was at a given position P and computing the meeting point of two object trajectories o_1 and o_2.

There has been recently a growing interest in the database community for mobile objects. Among the pioneering proposals the MOST model [43] essentially consists in applying temporal operators to spatial predicates. More recent studies propose a more systematic investigation of spatio-temporal operators [23, 8] or spatio-temporal data modeling based on constraints [14, 5]. We will compare some of the most significant aspects of these proposals with the DEDALE approach.

2.4 Summary

The above examples show the diversity of applications that need to handle geometric information. The difference lies first in the modeling of data, or, more precisely, in the integration between alphanumeric and

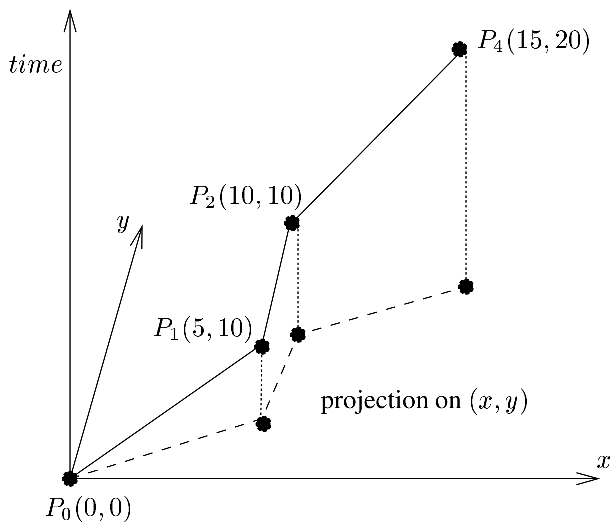


Figure 2: Trajectory of a moving object

geometric data. In some cases (geographic objects, moving points) a set of points is viewed as a part of an objects' description. In other cases (field-oriented data) pure geometric data is handled.

Common to all aforementioned examples is the necessity to provide a finite and concise representation of a potentially infinite set of points. An immediate answer is to build on the application's needs and to enrich the existing database system with new types. However, this type system has in turn to be extended when it comes short to fulfill new requirements. Given that points, lines and polygons in a 2-dimensional space have very slowly found their way in modern database systems, it is to be expected that the extension to spatio-temporal information or objects in higher dimension will be even more difficult. This extension is also likely to increase greatly both the complexity of the system and the heterogeneity of the data model.

Indeed, an important feature of this approach is the lack of uniformity. TINs are treated as TINs, moving objects as moving objects, 2D data as 2D data... each family of objects coming with its specific set of functions in order to manipulate them.

Finally the lack of declarativeness of the resulting query language is an essential issue. The query language is based on the set of operations provided with the data types and requires a great expertise from the end-user to be used appropriately.

The spatial data modeling approach of the current paper adopts a quite general and abstract point of view, where a set of points in dimension n is seen as an n-ary relation with an infinite extension. The idea is of course motivated by the constraint data model, proposed by Kanellakis, Kuper and Revesz [26] a decade ago. The model is an elegant solution to the problem of data heterogeneity, because it permits uniform reasoning about data as being sets, finite or infinite. It also features a general solution to the finite representation problem by storing relations as quantifier-free formulae over some arithmetical domain.

The approach is not application-driven: any geometric object, in any dimension, can be represented and queried with uniform structures and languages. Hence it captures, among others, geographic objects, field-oriented data and spatio-temporal data, and the characterization of these specific situations relies on a classical logical modeling process, and not on an heavy ad hoc extension of the underlying systems' technology. Data manipulation is set-based and a rather simple abstract view of data is provided. This abstraction is fundamental because it permits to hide the physical structure to the end-user (data independence principle) and it facilitates the design of high level declarative query languages. Moreover the model relies on a sound and simple first-order semantics which permits a sound specification.

In the following we describe the DEDALE system, designed as an effective application of the above ideas to spatial data handling, validating the model on real applications.

3 The Data Model

This section presents the data model of DEDALE. This model is "abstract" in the sense that it allows for several implementations which can be very different from what the user sees and thinks about (hence the model supports the "data-independence principle"). An algebraic query language is defined, which is the foundation for the user SQL-like query language presented in Section 4. A similar comment holds for the user query language regarding the algebraic definition: many other choices could have been done in the set of potential languages equivalent to our algebra. We shall also justify and comment the choices of DEDALE pertaining to this topics in Section 4 and 5.

We begin with a brief presentation of the constraint data model and then introduce the DEDALE extensions.

3.1 The constraint data model

A spatial set S with dimension d can be viewed as a subset of $\mathbb{R}^d$ and can be represented with a relation symbol R of arity d. For a given d-tuple $\bar{a}$, $R(\bar{a})$ holds iff $\bar{a}$ belongs to S. In this standard relational setting, spatial relations are manipulated using standard query languages, such as first-order (FO) or fix-point (FixPoint), over a relational signature [1].

Unfortunately this idealistic representation cannot be implemented without further restrictions. The main issue is that computers can only manipulate finite sets of objects. Thus it is necessary to have a finite representation of the possibly infinite subset of $\mathbb{R}^d$ manipulated. However providing a finite representation is not sufficient, because even if the input relations are finitely representable nothing guarantees that the output of the queries will.

One goal is to identify a set of infinite relations that can be finitely represented using an encoding denoted by *enc*, and a query language L which fulfills the following properties:

- At any time, the relations manipulated must be finitely representable and have a representation in *enc*.
- Closure: The output of any operation or query in L should be a relation representable in *enc*. In other words an operation or query takes as input(s) encodings of relations in *enc* and returns as an output an encoding of a relation in *enc*.
- Computability: One representation of the output has to be computable from any given representation of the input relations (There might be several different representations of the same subset of $\mathbb{R}^d$ in *enc*).
- Tractability vs. expressiveness: The complexity of evaluating queries in L should be maintained as low as possible whereas the expressive power of L should be as high as possible.

The constraint database framework meets these requirements. The infinite relations considered are those that correspond to the tuples satisfying a quantifier-free formula in a first-order language with arithmetic operations, interpreted in a dense domain such as $\mathbb{Q}$ or $\mathbb{R}$. The query language is the associated first-order language.

Several models have been proposed in the constraint database framework [28]. For the sake of clarity, we restrict the presentation to semi-linear encoding which is the one used in DEDALE.

Linear constraints

A subset S of $\mathbb{R}^d$ is said to be semi-linear if it satisfies a boolean combination of linear inequalities with integer coefficients. A linear inequality is a condition of the form: $a_1.x_1 + \cdots + a_d.x_d \geq a_0$, where the a_is are integers, and the x_is are variables. As any boolean combination of linear inequalities can be equivalently rewritten in disjunctive normal form (DNF), any semi-linear subset S can be represented by a formula of the form:

$$\bigvee_i \bigwedge_j \theta_{i,j}(\bar{x})$$

that is to say, $S = \{\bar{x} \mid \bigvee_i \bigwedge_j \theta_{i,j}(\bar{x})\}$. The set of formulas of this form is the desired encoding *enc*. Semi-linear constraint databases manipulate semi-linear relations through their DNF encoding. Although several forms of a boolean combination of linear inequalities can be equivalently used for encoding a semi-linear relation, DNF encoding is more appropriate for efficient query evaluation.

For instance a convex polygon is represented as a conjunction of linear inequalities defining half-planes. The polygon of Figure 3.a can be represented by the following formula, each conjunct delimiting a half plane:

$$y \leq x \wedge x \leq 7 \wedge y \geq 1 \wedge x + 3y - 16 \leq 0$$

The non-convex polygon of Fig. 3.b is defined as a union of two convex components (convex polygons) with constraint representation:

$$\begin{array}{c}(y \leq x \wedge y \geq 1 \wedge x + 3y - 16 \leq 0 \wedge 3x - 2y - 15 \leq 0) \\ \vee \\ (x \leq 9 \wedge y \geq 1 \wedge 3x - 2y - 15 \geq 0)\end{array}$$

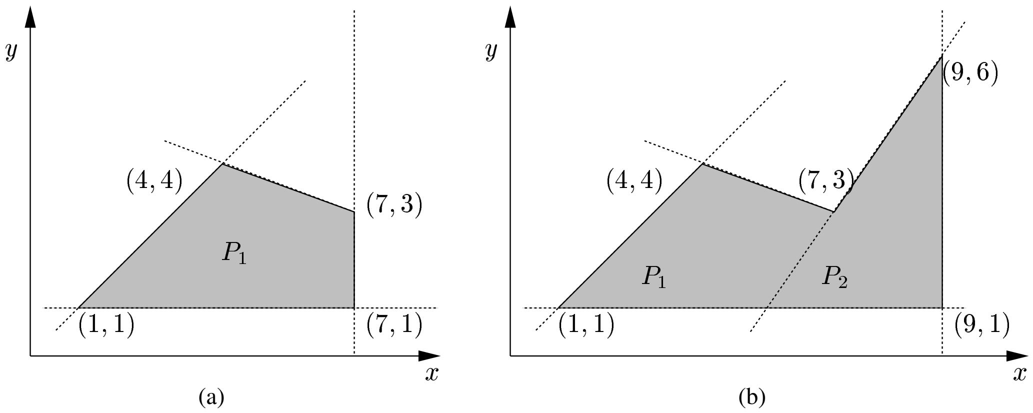


Figure 3: (a) a convex polygon (b) a non-convex polygon represented as a union of convex polygons

Figure 2 depicts an example of a trajectory of a moving point in the plane. It can be finitely represented as the following formula over the variables x, y and t:

$$\begin{array}{ll} x = 10t \ \wedge \ y = 5t \ \wedge \ 0 \leq t \leq 1 & \text{(a)} \\ \vee & \\ y = 5t \ \wedge \ x = 10 \ \wedge \ 5 \leq y \leq 10 & \text{(b)} \\ \vee & \\ 3y = 10t + 10 \ \wedge \ 3x = 5t + 20 \ \wedge 2 \leq t \leq 5 & \text{(c)} \end{array}$$

Each conjunct represents a segment in the trajectory: for instance, the object moves from point P_0 to point P_1 during the time interval [0,1]. The speed for a given segment is a constant value which can be easily computed.

To summarize we manipulate semi-linear relations via a representation using semi-linear inequalities in DNF. The choice of an appropriate "normal" form, like DNF, among the many equivalent representations of a given semi-linear set is implementation dependent and does not affect the model. Section 5 will present and discuss the choices made in DEDALE.

Algebra

A natural query language for manipulating semi-linear relations is first-order over the ordered reals with addition: FO$(0, 1, <, +, -)$(a precise definition of can be found in [28]). It extends the classical relational calculus with some limited arithmetics: manipulation of any particular integer, addition, subtraction and order comparison. This language has the following desired properties:

- It is closed. This is a consequence of the fact that FO$(0, 1, <, +, -)$ admits *quantifier elimination* [40]. Indeed after replacing each relation symbol by its semi-linear encoding, each formula of FO$(0, 1, <, +, -)$ has an equivalent quantifier free formula which is thus a semi-linear encoding of the output. It can afterwards be transformed into the desired normal form (in our case DNF).
- It is computable. Indeed this process of eliminating quantifiers can be done in time polynomial in the size of the semi-linear encoding of the inputs (see [28] for more details on the complexity issues). This guarantees a low data complexity. However the complexity is exponential in the dimension of the data [16]. As observed in [17], assumptions on the geometry of the data such as functional dependencies permit to really manipulate efficiently high dimensional data.
- It is equivalent to the algebra ALG$_{(<,+,-)}$ presented below which permits rewriting techniques and thus high level optimization.
- On top of this algebra it is easy to design user-friendly declarative query languages.

To illustrate the closure of FO$(0, 1, <, +, -)$ and the elimination of quantifiers, consider the following example.

Example 1 Assume that the schema consists of a unique binary relation R defined on x and y. Let q be the query computing the projection of R over the x axis. q can be expressed in FO$(0, 1, <, +, -)$ by:

$$\{x \mid \exists y R(x, y)\}$$

Let I be an instance for the relation R depicted in Figure 3.a. A semi-linear encoding of this instance is $y \leq x \wedge x \leq 7 \wedge y \geq 1 \wedge x + 3y - 16 \leq 0$. The *evaluation* of q over I corresponds to the formula:

$$\{x \mid \exists y\ y \leq x \wedge x \leq 7 \wedge y \geq 1 \wedge x + 3y - 16 \leq 0\}$$

After eliminating the quantifier over y the foregoing formula is equivalent to: $x \leq 7 \wedge x \geq 1$ which is a semi-linear representation of the output of q over I. This process is computable and its data complexity is PTIME (the size of the data being the size of its encoding). □

Several algebraic languages equivalent to FO$(0, 1, <, +, -)$ have been proposed. We present below the algebra ALG$_{(<,+,-)}$. It contains as a basis all semi-linear inequalities together with the following operators: intersection, $\cap$, projection, π, union, $\cup$, difference, $-$, and renaming, ρ.

Let R_1 and R_2 be two relations represented respectively by F_1 and F_2.

1. $R_1 \cap R_2$ is the semi-linear relation represented by $F_1 \wedge F_2$.

2. $\pi_{\bar{x}}\ R_1$ is the semi-linear relation represented by the quantifier-free formula equivalent to $\exists \bar{y} F_1$, where $\bar{y}$ are the free-variables of F_1 which are not in $\bar{x}$. This formula can be computed using for instance the Fourier algorithm [40].

   We allow the definition of new variables defined as terms of the form $v[t(\bar{x})]$, where v is the name of the new variable and t is a term constructed from the variables $\bar{x}$ and the function $+$. For instance $\pi_{x,y,z[a-h]}\ R(x, y, a, h))$ yields a relation over the variables x, y and z, the latter being $a - h$. Its interpretation is obvious, and its computation straightforward using $\cap$ and π.

3. $R_1 \cup R_2$ is the relation represented by $F_1 \vee F_2$.

4. $R_1 - R_2$ is the relation represented by $F_1 \wedge \neg F_2$.

5. $\rho_{x \to x'}\ R_1$ is R_1 but its representation is now $F_1[x, x']$ where variable x has been replaced by x'.

If $R(x_1, \cdots x_n)$ is a relation represented by the formula $F(x_1, \cdots x_n)$, the selection operation $\sigma_f(R)$ where f is a selection criteria of the form $a_1x_1 + \cdots a_nx_n \geq a_0$ can be expressed in the above algebra by $R \cap \{f\}$. If $R_1(x_1, \cdots x_n)$ and $R_2(x_1, \cdots x_n)$ are relations, $R_1 \times R_2$ can be expressed in the above algebra by $R_1 \cap \rho_{x_i \to y_i, i \in [1,n]}(R_2)$. Similarly the join, $\bowtie$, of two relations can be expressed using a suitable combination of $\times$ and ρ.

As can be seen from the above definitions, the algebraic operators applied to the constraints representation simulate standard relational operators applied to infinite relations, and therefore deliver a correct mathematical representation of the result that complies with the constraint representation. It is easy to see that FO$(0, 1, <, +, -)$ and ALG$_{(<,+,-)}$ have the same expressive power.

This setting can be extended in order to uniformly capture different types of data. Indeed a finite instance of a classical relation schema $People(name)$ can be represented as a union of tuples of the kind $name =' Alice' \vee name =' Bob' \vee ....$

Summary

In summary the constraint model offers a powerful and generic framework for reasoning about geometric data. The foregoing discussion presented the data requirement and a query language corresponding to the semi-linear constraint database framework.

Other encodings have been studied in the constraint database framework such as semi-algebraic relations [28]. Nevertheless semi-linear encoding offers a good trade-off between representational power and query complexity. Any semi-linear set can be encoded with linear constraints, and it turns out that this is sufficient for most applications handling spatial data [30].

All the operations of the foregoing algebra ALG$_{(<,+,-)}$ have a PTIME data complexity, the costly operations being the projection π and the difference $-$ which are inherently exponential in the dimension of their inputs [16].

3.2 The model of DEDALE

We introduce now the model of DEDALE which adapts the linear constraint model to practical requirements regarding representation and querying of spatial objects. It must be stressed that, although the examples are drawn for the applications discussed in Section 2, the model is general and can handle any application whose data satisfies the restrictions discussed thereafter. We successively introduce the DEDALE model and the DEDALE algebra.

3.2.1 Data model

DEDALE is based on the semi-linear framework described in the previous section. As a consequence, we cannot represent objects described by higher-order polynomials. Disks for instance, definable by constraints of the form $x^2 + y^2 \leq \alpha$, cannot be represented in DEDALE.

In order to associate alpha-numeric descriptive attributes with a set of points, data is stored in non-first normal form relations [46]. This allows the manipulation of infinite sets of points as first class citizens. One level of set nesting is sufficient to achieve this goal.

In the sequel, *nested relations* of arity d will correspond to – possibly infinite – semi-linear subsets of $\mathbb{R}^d$, whereas *ground relations* are *finite* sets of tuples over an infinite alphanumerical domain U. Obviously nested relations will be represented by their finite encoding.

We assume the existence of an infinite set of attribute names *Att*. If R is a relation name of arity $k = k_1 + k_2$ a schema for R is of the form $R(a_1, \cdots, a_{k_1}, b_1[x_1^1, \cdots, x_{j_1}^1], \cdots, b_{k_2}[x_1^{k_2}, \cdots, x_{j_{k_2}}^{k_2}])$, where $(a_i), (b_i), (x_i)$ are attribute names. Attributes b_i's denote nested relations representing a semi-linear subset of $\mathbb{R}^{j_i}$ where each axis is denoted by x_j^i, $1 \leq j \leq j_i$.

Example 2 The relation schema *Flight (fname, from, to, traj[t, x, y, a])* represents aircrafts by their code, the departure and arrival cities, and the trajectory including the altitude at each point.

In this example the first 3 attributes are elements of the alphanumerical domain U while the attribute *traj* is a nested relation which represents a subset of $\mathbb{R}^4$. Each attribute of the nested relation *traj* is named and can thus be accessed explicitly. For instance if u is a tuple of relation *Flight*, *u.traj.x* ranges over the second attribute of the nested relation *traj*. □

An instance of a tuple in relation *Flight* is shown in Figure 4. Attributes *name*, *from* and *to* are represented as regular relational atomic values, whereas the nested attribute *traj* is represented as a DNF formulae.

name	from	to	traj
DF234	Paris	Washington	$x = 230t + 354 \wedge y = 134t - 26 \wedge a = 260t + 10 \wedge 1 < t < 11$
			$x = 289t + 567 \wedge y = 190t + 2 \wedge a = 120t \wedge 11 < t < 20$
			$x = 340t + 5 \wedge y = 176t + 18 \wedge a = 2400 \wedge 20 < t < 30$
			$\cdots$

Figure 4: An object with nested attributes representing point sets.

The "non-ubiquity" property of an aircraft (at a given instant it has a single position) is expressed in the formulas by the functional dependency of variables x, y, and a on t. In other words, t is the key of this (infinite) relation. This is similar to the concept of key in the classical relational setting, except that the functional dependency is explicitly given as a function.

In practical situations, the trajectory of objects whose position depends on 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, and functions that give x, y, and a as a function of time express this interpolation. The foregoing *Flight* relation is therefore an example of an *interpolated relation*, more precisely defined as follows:

Definition 1 *$R \subseteq \mathbb{R}^d$ is an* interpolated relation *with key $\bar{x}$ if R is a linear constraint relation, and*

- *A subset $\bar{x}$ of the attributes (called the* key*) is used as a basis for computing the other attributes (called the* interpolated values*).*

- *each conjunct in the DNF representation consists of:*
  - *either equality or inequality constraints restricted to the attributes of the key;*
  - *or constraints on the interpolated attributes defined as linear functions of the key.*

The relation *interpolation dimension* is $|\bar{k}|$, whereas d is its global dimension. Therefore interpolated relations in a d-dimensional space can be symbolically represented using a special normal form, called *Interpolated Normal Form* (INF), which captures all the aforementioned examples of linear interpolation.

Definition 2 *A quantifier free formula $\varphi \in$ FO$(0, 1, <, +, -)$ with free variables $x_1 \cdots x_k$, and $h_{k+1}, \cdots, h_d$ is in* interpolated normal form (INF) *with* key $\bar{x} = x_1 \cdots x_k$ *if it is under the form:*

$$\bigvee_i \; c_i(\bar{x}) \wedge h_{k+1} = f_i^{k+1}(\bar{x}) \wedge \cdots \wedge h_d = f_i^d(\bar{x})$$

where each $c_i(\bar{x})$ is a conjunction of linear constraints over $\bar{x}$, and each f_i^j is a linear function over $\bar{x}$.

For example the instance of the relation *Flight* depicted in Figure 4 contains a flight which trajectory is an interpolated relation of global dimension 4, interpolated dimension 1 with key t. Notice also that its representation is in INF.

Introducing functional dependencies in spatial data modeling brings several advantages. First some application-dependent restrictions such as "a moving object has a single position at a given time instant" or "there is one elevation value for a given position" are elegantly expressed. Second, a relation with interpolated dimension l corresponds to a l-dimensional object embedded in a d-dimensional space, and, for most queries, the evaluation process will depend only on l and not in d. For instance, viewing a trajectory as a set of points whose values depend solely on one of the variables, t, nicely expresses that the point set is actually a 1D object, embedded in a higher-dimensional space. The same comment holds for elevation data which is essentially a 2D object embedded in a 3D space.

As we mentioned earlier the complexity of some operators of the algebra is exponential in the dimension of a relation. In most cases it should be possible to compute operations on interpolated relations at the cost of their interpolation dimension (1 or 2) and not at the general cost of 3D or 4D geometric computation. Indeed, as shown in [17], it is possible to define classes of queries for which the evaluation complexity has an exponential exponent independent from the total number of variables (i.e., it depends only on the size of the key). This is particularly important to avoid the overhead of applying geometric algorithms in high-dimensional spaces. Although we do not explore this aspect in this paper, Section 5 illustrates on an example how we can take advantage of the specific form of interpolated relations to optimize query evaluation.

In order to insure tractability the current implementation of DEDALE is restricted to semi-linear sets that are interpolated relations with interpolated dimension 2. This imposes a limitation only on the interpolated dimension, not on the global dimension of the data. In particular this allows us to model TINs and trajectories, 4 dimensional objects, in a unified framework, which current systems can hardly handle. Besides, this representation is sufficient to encode most natural applications and in particular all those described in Section 2.

In summary the data model of DEDALE is a semi-linear constraints model, with one level of nesting in order to handle spatial relations as first-class citizens, and such that the nested relations are interpolated with an interpolated dimension 2.

In the rest of the paper we shall use a sample database schema with the following relations:

- $LandUse(lname, geom[x, y])$ represents the ground occupancy or landuse (attribute $lname$) such as 'forest' or 'pasture' at location(s) (x, y).
- $Flight(fname, traj[t, x, y, a])$ represents the trajectory of a set of aircrafts, identified by their name (attribute $fname$). Variables (x, y) denote the location, a the altitude and t the time. Each trajectory is an interpolated relation with key t.
- $Country(cname, geom[x, y, h])$ is a map of countries identified by their name (attribute $cname$). The height above the sea level h is interpolated from the location in the plane: the key is (x, y).

3.2.2 Algebra

We now introduce the DEDALE algebra. It extends the relational algebra and contains a *Tuple* constructor $[o_1, \ldots, o_k]$ that builds tuples from objects $o_1, \ldots, o_k$ of atomic or spatial types, i.e., defined on the alphanumeric domain U or on a spatial domain.

We distinguish between *valid* and *invalid* tuples. A tuple is *valid* if its nested attributes are non empty sets, and invalid otherwise. Invalid tuples are discarded. The rationale behind this choice is that an object with empty space (empty nested relation) is considered as void. A different choice is adopted in [29] which permits the manipulation of objects with empty nested relations.

Attribute projection $t.i$ where i is a natural number, extracts the i^{th} attribute of a tuple, t, if t is valid, and is undefined otherwise. For example, if $t = [3, 0, \{1, 2\}]$, and $t' = [3, 0, \{\}]$, then $t.1 = 3$ while $t'.1$ is undefined. Projection relies on an empty-set test and testing the emptiness of a set is a costly operation (it relates actually to *normalization* of tuples, see Section 5.) The empty tuple (i.e., tuple of arity zero) is denoted by [ ]. Nested relations are constructed using the *set* operator, $\{\bar{x}|\varphi(\bar{x})\}$, where $\bar{x}$ is a tuple of rational variables, and $\varphi(\bar{x})$ is a formula. Sets can equivalently be constructed algebraically from $\mathbb{R}^k$.

Let $E(X)$ be any algebraic expression, with a variable X. Then $\lambda X.E(X)$ denotes a function. For instance, let X be a variable of type $[U, \{\mathbb{R}^2\}]$, then

$$\lambda X.[X.1, \{[x, y]|[x - 1, y - 1] \in X.2\}]$$

denotes a function which performs a translation of the geometry.

The language contains the relational algebra (see previous section) and the following operations.

- *Restructuring*, MAP, applies to inputs of *relation* type. If $E(X)$ is an algebraic expression and R is a relation then $MAP_{\lambda X.E(X)}(R)$ defines the relation in which each tuple t of R has been replaced by $E(t)$.
- *unionest*, applies to relations with a nested attribute i. $unionest_i(R)$ replaces the tuples of R having the same values for all attributes other than i, by a single tuple having the same value for all attributes but i and for i the union of all the corresponding values of the i attribute.

A query has two levels, one at the root level which corresponds to classical relational computation, and one at the nested level which corresponds to spatial computation. The expressive power of the algebra described above corresponds to the relational calculus at the relational level and to FO$(0, 1, <, +, -)$ at the nested level.

The similarity of the languages at the ground and at the nested levels permits algebraic optimization, by transforming expressions over nested relations into simplified expressions at the ground level. We shall explore optimization issues in Section 5.

To illustrate the algebra, we provide a few examples of algebraic queries on our sample schema.

1. Which parcel(s) in *LandUse* shares at least one point with rectangle $Rect(x_{min}, x_{max}, y_{min}, y_{max})$?

$$\pi_{lname,geom}(MAP_{\lambda X[X.lname, X.geom, \sigma_{x_{min}<x<x_{max} \wedge y_{min}<y<y_{max}}(X.geom)]}(\mathit{LandUse}))$$

   The selection expression σ takes advantage of the removal of invalid tuples from the result. If σ returns an empty set, then the corresponding tuple is invalid.

   Note that expressing other spatial selections such as *point query* is similar. Actually we avoid all the typing problems raised by the usual way of defining spatial operations. We do not care whether the spatial criteria is a point, a line, a rectangle or any other object, nor do we care about the type of the nested relation.

2. Give the part of the parcels in *LandUse* contained in a given rectangle (clipping)

   The algebraic expression is similar to the previous one (and it would remain the same for any type of spatial object). The only difference is that we keep in the result the intersection of the rectangle and the geometry.

$$MAP_{\lambda X[X.lname, X.geom \cap Rect]}(\mathit{LandUse})$$

3. Which location were the airplanes flying over at time t_1?

   Time, altitude and location coordinates are treated uniformly by considering each of them as an axis in a multidimensional space. The query is expressed as the selection $t = t_1$ on each aircraft trajectory, followed by a projection on (x, y).

$$MAP_{\lambda X[X.fname, \pi_{x,y}(\sigma_{t=t_1}(X.traj))]}(\mathit{Flight})$$

4. Which position and when were the airplanes A1 and A2 flying over at the same time?

   This can be expressed as a join on x, y and t. Hopefully the airplanes do not share the same altitude, so there is no join on a.

$$MAP_{\lambda X[X.traj \bowtie_{x,y,t} X.traj2]}(\sigma_{fname='A1'}(\mathit{Flight}) \times \sigma_{fname='A2'}(\rho_{traj \rightarrow traj2}(\mathit{Flight})))$$

5. Give the trajectory of the airplanes and their height above the ground when they were over the French territory.

   Here we perform a join on (x, y) in order to obtain, for each point of the trajectory, the elevation h of the ground above sea. By subtracting h from the altitude of the aircraft (which is recorded with respect to the sea level), one gets the required information.

$$MAP_{\lambda X[X.fname, \pi_{t,x,y,z[a-h]}(X.traj \bowtie_{x,y} X.geom)]}(Flight \times \sigma_{cname='France'}(\mathit{Country}))$$

   Note that we exploit in this query the ability to create a new variable, z, defined by the term $a - h$.

In the next section we propose a user-friendly language based on SQL, and show how the above examples, as well as several others, can be simply expressed.

4 Querying

A query language for spatial data should be declarative, user-friendly and independent of the applications. Its expressive power should be kept under control in order to maintain a low complexity query evaluation process. Moreover it should hide as much as possible the underlying complexity of the data model.

The query language proposed in this section meets those requirements. It is based on a slight extension of SQL and manipulates spatial and non-spatial data in a uniform way. Thus it is fairly easy to understand and use for those users already having SQL knowledge. Actually, as end users, we need not worry about constraints, first-oder formulas, query evaluation techniques, or even the dimension of spatial data. In other words, the database can be queried in a strictly declarative way. We can simply consider the relations in the schema as storing finite or infinite extensions, and use a classical relational query. This language, used in DEDALE, represents one of the main contributions of the paper. Section 5 describes it evaluation by the system.

4.1 The user query language

The SQL language uses variables representing tuples ranging over a relation. Each tuple T in the result is obtained by instantiating each of these variables by a tuple in the base relations, checking that the **where** clause is satisfied, and then projecting on one or several attributes.

Our query language simply consists of extending this feature by allowing variables to range over infinite relations, in the **from** clause. The result of a query is a nested relation, with one or several nested point sets, constructed as follows:

1. if $v_1, \ldots v_n$ are tuple variables ranging over *finite* relations $R_1(a_1^1, \ldots, a_1^{p_1}), \ldots R_n(a_n^1, \ldots, a_n^{p_n})$, then any expression over the $v_i.a_i^j, i \in \{1, \ldots, n\}, j \in \{1, \ldots, p_i\}$ yields an attribute in the result relation.

2. if $v_1, \ldots v_n$ are tuple variables ranging over *infinite* relations $G_1(x_1^1, \ldots, x_1^{p_1}), \ldots G_n(x_n^1, \ldots, x_n^{p_n})$, then $Gname[S]$, where S is a subset of the $\{v_i.x_i^j, i \in \{1, \ldots, n\}, j \in \{1, \ldots, p_i\}\}$ denotes a nested relation with name $Gname$.

As a first example to illustrate the language, consider the *LandUse(lname, geom[x,y])* relation, and express the *clipping* query which keeps the part of the parcels contained within a rectangle $Rect(xmin, ymin, xmax, ymax)$. The algebraic expression has been given in the previous section.

select	l.name, clip[p.x, p.y]
from	LandUse **as** l, l.geom **as** p
where	p.x **between** xmin **and** xmax
and	p.y **between** ymin **and** ymax

The result of this query is a nested relation with schema $[name, clip]$ where $clip$ is a nested point set in the 2D space. We could simplify the notation to $clip[x, y]$ in the **select** clause, since there is no ambiguity.

At the abstract level the query can be interpreted as follows:

1. Unnest relation *LandUse*. One obtains conceptually an infinite instance which associates to each point the descriptive attributes;

2. Evaluate the **where** clause on the unnested relation. One obtains a set of tuples that satisfy the selection criteria;

3. Nest the result obtained at the previous step on the descriptive attributes: this yields a finite relation with nested pointset.

Obviously, the query will not be evaluated this way! There is a straightforward translation into the algebra presented in Section 3. The query can then be optimized using rewriting rules in order to derive the query evaluation plan in the physical algebra. We will see in Section 5 how DEDALE evaluates queries.

The previous query computes the intersection of geometries of two objects. More generally any query computing the points that satisfy a (first-order) property can be expressed. Another useful and common class of spatial queries considers the pointset describing the geometry of an object as a whole, and keeps the entire pointset if *at least* one of its points satisfies a property. An example is given by the *windowing* query: take the parcels which intersect a rectangle $Rect$. It can be expressed as follows.

select	lname, p1
from	LandUse **as** l, l.geom **as** p1, l.geom **as** p2
where	p2.x **between** xmin **and** xmax
and	p2.y **between** ymin **and** ymax

The interpretation of the query is similar to that of the previous query: unnest the relation(s), compute the join, then nest again. If both objects share at least one point, then $geom \cap Rect$ is non empty and the whole geometric extension of the *LandUse* will be included in the result.

4.2 Sample queries

The following queries manipulate relations satisfying our sample schema mixing several kinds of geometric objects. Relation *LandUse (lname, geom[x, y])* gives the ground occupancy or landuse (attribute *lname*) at location (x, y). We also represent airplanes as moving points in the relation *Flight (fname, traj[t, x, y, a])* which gives the trajectory of a set of planes. Variables (x, y) denote the location, a the altitude and t the time. Finally *Country (cname, geom[x, y, h])* is a map of countries, featuring the height above the sea level.

1. Which location were the airplanes flying over at time t_1?

select	fname, pos[p.x,p.y]
from	Flight **as** f, f.traj **as** p
where	p.t = t_1

2. Which position and when were the airplanes A1 and A2 flying over the same point at the same time?

select	pos[p1.t, p1.x, p1.y]
from	Flight **as** f1, Flight **as** f2, f1.traj **as** p1, f2.traj **as** p2
where	p1.x = p2.x **and** p1.y = p2.y **and** p1.t = p2.t
and	f1.fname= 'A1'
and	f2.fname= 'A2'

3. Give the trajectory of the planes and their height above the ground when they were over the French territory.

select	fname, FranceTraj[FT.t, FT.x, FT.y, z[FT.a-MG.h]]
from	Flight **as** f, f.traj **as** FT, Country **as** c, c.geom **as** MG
where	FT.x = MG.x **and** FT.y = MG.y
and	c.cname = 'France'

4. Give the contour line on the ground for h=1000 (isoline with altitude 1000) in Italy.

select	contour[x, y]
from	Country **as** c, c.geom **as** MG
where	MG.h = 1000
and	c.cname = 'Italy'

5. Show the 2D map of the countries with a summit higher than 4000m.

select	cname, MG
from	Country **as** c, c.geom **as** MG
where	MG.h > 4000

6. Show the trajectory of airplane A1 with the elevation h of each point the airplane is flying over.

select	ElevTraj[FG.x, FG.y, FG.t, MG.h]
from	Country **as** c, c.geom **as** MG, Flight **as** f, f.traj **as** FG
where	MG.x = FG.x **and** MG.y = FG.y
and	f.fname = 'A1'

7. Show the forests whose altitude ranges between 1000 and 2000 meters.

select	HighForests[MG.x, MG.y]
from	Country **as** c, c.geom **as** MG, LandUse **as** l, l.geom **as** LG
where	MG.x = LG.x **and** MG.y = LG.y
and	MG.h **between** 1000 **and** 2000
and	l.lname = 'forest'

The queries above are simple to express, mainly because the internal mechanisms are not of our concern. *Point location* and *windowing* are other standard operations that are simple to express (selection) because the internal mechanisms are not of our concern. All queries correspond to an algebraic expression, many of them were already given in Section 3.

4.3 Discussion

In order to illustrate the benefits of the DEDALE approach, we compare it with three among the most significant proposals in the area of spatio-temporal models. The first one, called MOST, extends existing DBMS to track moving objects. The second one uses a more systematic set of spatio-temporal datatypes. By integrating interpolation procedures into SQL, the third allows the querying of DEMs.

The MOST model was first proposed in 1997 [43], and later implemented in the DOMINO prototype [48]. The model is intended to manage the current and future positions of moving objects, and is thus mainly oriented towards traffic monitoring applications.

Data representation relies on the concept of *dynamic attributes*. Intuitively, the value of such an attribute changes over time according to a given function with no need for an explicit update. The motion information (speed + direction) of a mobile object is represented by dynamic attributes, one for each coordinate of the space embedding the trajectory of this object. Dynamic attributes need to be updated only when the motion information changes (for instance when a taxi takes a turn). Here is the constraint representation of a dynamic attribute:

$$x = \alpha_x t + \beta_x \ \wedge \ y = \alpha_y t + \beta_y \ \wedge t_{update} \leq t$$

This illustrates the differences between the two approaches. First a dynamic attribute consists of only *one* symbolic tuple, while our data model handles a disjunction (a set) of symbolic tuples. In other words, dynamic attributes do not store the history of updates, and queries about the past are not possible, while the constraint data model permits to represent the whole trajectory.

The FTL (Future Temporal Logic) is proposed to extend a spatial query language with temporal operators.

- f **Until** g is true if g is true at some instant t in the future, and f is true until t.
- **Nexttime** f is true if f is true at the next instant.

>From these two basic operators, new temporal predicates can be derived: **Eventually**, **Always**, **Eventually_within_**c, **Eventually_after_**c, etc. For instance the following FTL query extracts the aircrafts that enter polygon P in the next 3 time units:

select fname
from Flight
and **Eventually_within_3** INSIDE(geom, P)

The language assumes the existence of a spatial operator *INSIDE(obj, area)*. The same query is expressed in our language by the following query.

select fname
from Flight **as** f, f.traj **as** FG, P
where FG.t $>$ now + 3
and f.x = P.x **and** f.y = P.y

In summary MOST is a pragmatic solution to introduce in an existing DBMS some functionalities to track moving objects. It lacks however of generality and its expressive power is somewhat unclear.

A more systematic definition of spatio-temporal data types is used in [23, 8]. Following a well-established trend in spatial data modeling, the design of spatio-temporal data types is presented in [23], and a type system is described in [8]. The model relies on two abstraction levels: the abstract level, described in [23], allows to view data as infinite point-sets, whereas the discrete level [8] provides a finite representation suitable for an implementation.

Given a set of classical types, alpha-numerical or spatial, whose instances are static, the *moving* constructor builds a type whose values are functions of time. For instance the *point* type has an associated time-dependent type *moving(point)*, or *mpoint* in short. Here is the schema of the *Flight* relation.

Flight (fname : string, traj : mpoint)

Operations on spatial types can then be "lifted" to operations on spatio-temporal data types. For instance the *distance* operation that takes two points and yields a real, can be generalized to take as inputs two moving points, and returns a "moving real", i.e., a real function of time. For instance the following query gives the flights whose length is greater than 5000 Kms.

select fname
from Flight
where length (trajectory (traj)) $>$ 5000

Here, *trajectory* gives the 2D projection of the flight's trajectory. The equivalent DEDALE query would be, assuming a *length*[2] function:

[2] Note that the current version of DEDALE does not contains the *length* function because it raises some problems in the closure of the linear constraint model [28].

```
select   fname
from     Flight as f, f.traj as FG
where    length (Route[FG.x, FG.y]) > 5000
```

The data type approach leads to powerful languages, with the counterpart (non declarativity) that the user must learn a lot of operations, and must be aware of the potential type errors when applying or composing operations.

Let us finally mention one of the few studies addressing the modeling of DEMs. Neugebauer proposes in [34] 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 explicitly manage interpolation.

Consider the example of a cross-section of the French DEM with, say, the plane defined by the equation $x = 5000$. In the model of [34], the TIN would be represented by a collection of triplets (x, y, h), and an interpolation function *my-function*. The syntax of [34] leads to a query of the type:

```
select  y, h
from    Country by method my–function(h)
                 entry (x=5000, y= 200)
                 step (0, 10)
where   y <= 500
and     cname = 'France'
```

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 $[(5000, 200), (5000, 500)]$.

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 ones are: (1) 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 separate manipulation of a sample and its interpolation function may lead to erroneous results and computation. (2) it breaks the "data independence principle". Indeed the user has to know that she is manipulating a TIN in order to query it.

In contrast, viewing the DEM as an infinite set of points does not require to worry about the proper combination of functions and samples. The DEDALE query would be:

```
select   CrossSection[y, h]
from     Country as c, c.geom as CG
where    cname = 'France'
and      CG.x = 5000
```

4.4 Summary

As illustrated in the foregoing discussion, the model completely hides the underlying structure. The end user does not have to know that the relation *Country* is actually stored, say, as a TIN, and that relation *Flight* is stored as a set of finitely many sample points for each flight, the complete trajectory being computed when needed using interpolation. This layered point of view meets one of the most important design requirements of database systems, as it leaves to the DBMS the freedom of choosing an appropriate storage of data and an efficient evaluation of queries.

The language fulfills the other requirements given at the beginning of this section, namely it is declarative, user-friendly and independent of the application. It can be considered as SQL made applicable to any kind of applications. Among other advantages, this permits to cover with a single, generic approach, a wide range of applications handling geometric information. Several studies address application-oriented data, but we are not aware of any attempt to integrate in a data model the functionalities required by the manipulation of maps, moving objects, TINs or more ambitiously arbitrary d-dimensional objects.

After translating the query into its algebraic form, the optimizer will derive a query evaluation plan which can take into account the internal/physical implementation of each relation and plug the appropriate physical operators in order to compute the required operation. We develop these implementation aspects in the next section.

5 Implementation

The implementation relies on two components: (1) a standard DBMS architecture providing all the modules related to query evaluation, from the SQL interface to disk storage, and (2) a dedicated constraint database engine.

With respect to the general model presented in the previous sections, the implementation addresses a restricted class of objects, namely those for which the interpolation dimension is less or equal 2. Recall (Section 3) that the interpolated relations have a key $\bar{k}$ and (possibly several) interpolated attributes whose value is obtained as a linear function defined over the key-space. The interpolated dimension is the cardinality of the key: by restricting it to 2, we capture maps, trajectories and elevation models, which is already more than existing systems, yet keeping a reasonably simple implementation and efficient query evaluation complexity.

We first outline the architecture of the system. The rest of the discussion successively covers the data storage, the query evaluation module, and the algorithms for implementing the algebraic operators. We end up the presentation by illustrating a typical query evaluation.

5.1 Architecture overview

In earlier versions, DEDALE was implemented on top of the O_2 DBMS. We used the O_2 high-level modules to implement constraint representation and operations, and querying was expressed by means of the OQL language using classes for constraint manipulations. Although this permitted an easy implementation, in this setting we had no access to low level facilities and the query evaluation mechanism was not aware of the specific semantics of constraint-based evaluation. In the current version, the prototype relies on the BASIS System [19] which proposes an extensible C++ implementation of the classical storage management functionalities, as well as an open query processor.

This implementation aims at showing that the proposed data model can be integrated in a relational DBMS with minimal effort. This is natural if we view geometric data as relational data, as advocated in the previous sections. From a practical point of view, we build our implementation on the standard technology for data storage, database indexing and on the parsing and optimization of the SQL query language. Since, however, the storage provides a finite representation of conceptually infinitely many points, the query evaluation process features a dedicated storage and specific algorithms.

Figure 5 depicts the architecture of the prototype. It consists of the following components:

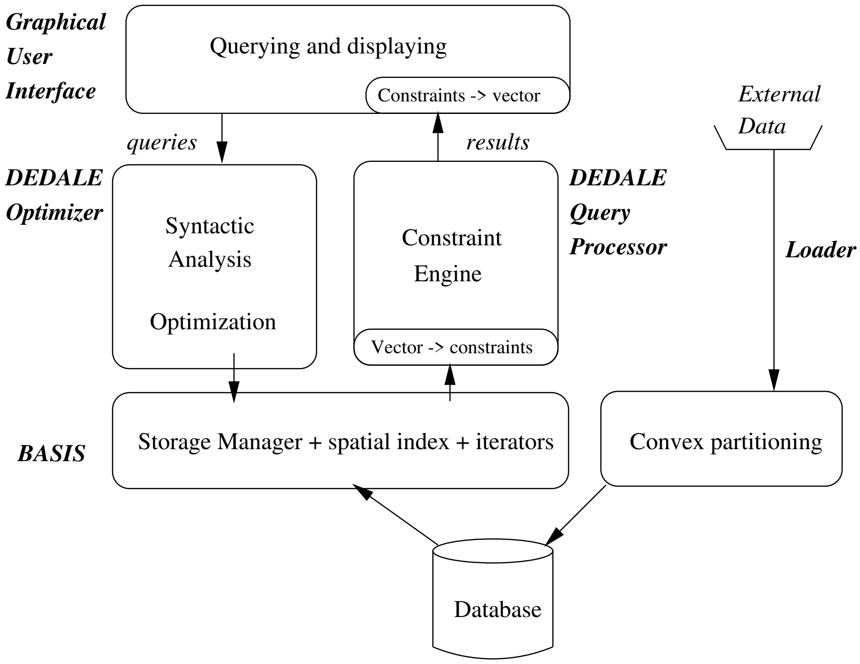


Figure 5: The prototype architecture

1. A Graphical User Interface (GUI), written in Java, allows to query the database with parameters such as points or rectangles, and to display query results including maps and other thematic data. Fig. 6 shows a screen copy of DEDALE while working on the application given by the French laboratory LAMA, Grenoble.

2. The DEDALE optimizer composed of the following modules: (i) the lexical and syntactical analyzer of a query and (ii) the semantic analyzer which performs the optimization phase according to rewriting rules and access path selection.

3. The query processor relies partly on functionalities provided by BASIS, such as collection scans, index traversals, construction of query plans, and partly on modules specific to multi-dimensional data management, namely a constraint engine used to process the part of queries on multi-dimensional data represented by constraints.

4. As the lowest layer, the BASIS kernel provides data storage, access paths and schema information.

A last component, external to the query evaluation process, is the *loader* which allows to import in the database external data usually provided with a vector-based geometric representation. Data is further decomposed into convex components. This will be explained in the following.

5.2 Data storage

This level is handled by BASIS. It provides a classical physical storage of relations in binary files containing records. A record is the physical representation of a relational tuple, and consists of fields, each being the physical representation of one of the tuple's attributes. This is a straightforward and classical organization.

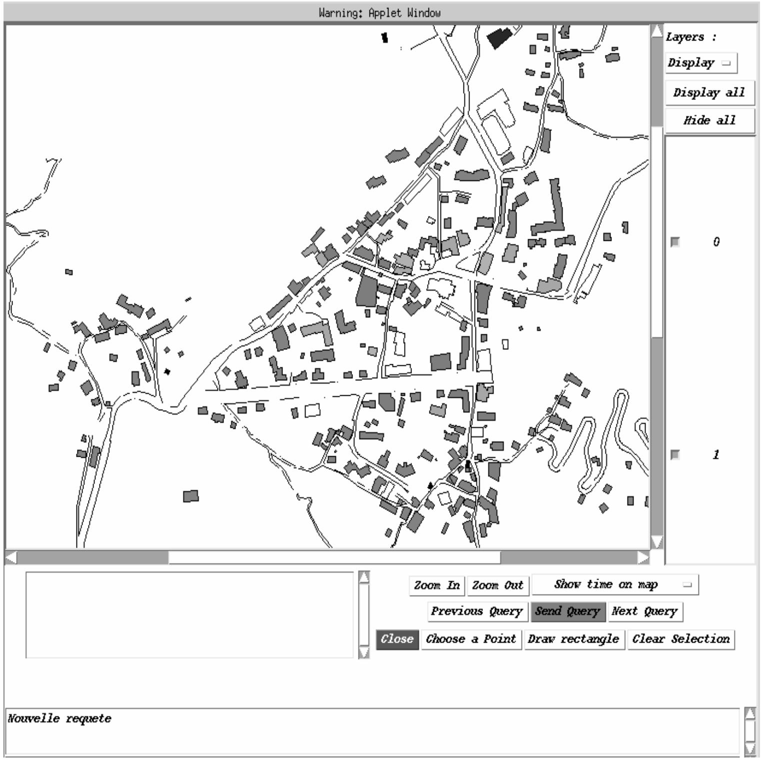


Figure 6: The application running on DEDALE

Storage of nested relations

We decided to store a constraint representation of the nested relation. This constraint representation is assumed to be in a normal form described in the following. Each constraint is represented as a compact binary object and the constraints representing a nested relation are clustered in an array of bytes.

Since the size of a nested relation can be arbitrarily large, it is stored separately from the main record and linked to it via a pointer, denoted *Geometric reference* (*GeomRef* for short) (see Fig. 7). In order to speed up the evaluation process, a *GeomRef* consists of two components:

1. the *minimal bounding box* (mbb) of the nested relation, i.e., the tuple ((min$_1$, min$_2$,..., min$_n$), (max$_1$, max$_2$, ..., max$_n$)) of points in the n-dimensional space with minimal (maximal) coordinates satisfying the constraints.

2. the physical address of the nested relation.

Using mbb's in query evaluation allows for a first fast filtering step. In most cases we just need to access the record to know whether it is worth fetching the nested relation. This generates a two-step evaluation process to be described in the following.

The normal form for the constraint representation satisfies the following requirements:

1. The representation is in disjunctive normal form (DNF). Geometrically this means that the relation has been partitioned into convex components.

2. There are no redundant constraints in a conjunct. For instance, $x < 5$ and $x < 1$ cannot belong to the same normalized conjunct.

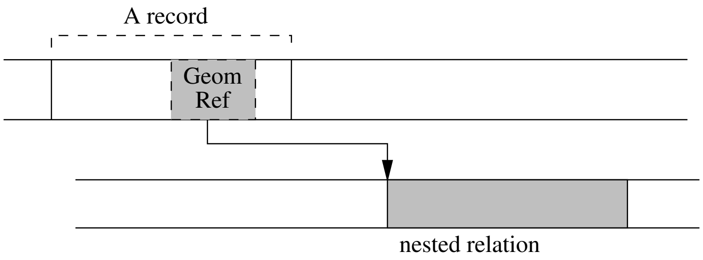


Figure 7: Data storage

3. There are no inconsistent constraints in a conjunct. For instance, $x > 5$ and $x < 1$ cannot belong to the same conjunct. It follows that each conjunct of a normalized relation is always satisfiable.

4. Although not central to normalization, the constraints in a normalized tuple are stored in a specific order which facilitates a possible display later on. This is mostly important for polygons: see the following discussion on vector/constraint conversion.

5. If the relation is *interpolated*, the material above defines an encoding of a convex element in its key-space (a time interval for trajectories, a triangle for DEMs ...). Then each linear function defined over this convex for variables such as position (trajectories) or elevation (DEM) is stored as a vector of coefficients.

Again, the normal form is mostly used for optimization purposes. It does not restrict the set of objects manipulated. The first condition can be viewed as a preprocessing of the data in order to speed up the query evaluation later on. Each object is partitioned into simpler (convex) objects. Efficient computational geometry algorithms exist for most operations on convex objects. One may wonder whether objects are not partitioned into even simpler components, such as triangles in dimension 2 and simplices in higher dimensions. The drawback with this method is that the size of the storage drastically increases. We believe that data partitioning into convex achieves a good trade-off between efficient evaluation and storage size.

Data loading and conversion

We now explain how 2 dimensional data are loaded in DEDALE and converted into normal form. We assume the data to be vector-based as it was the case for all data we obtained from different sources. In particular, non-convex objects must be convexified prior to the storage. Since we restricted the interpolation dimension to 2, the convexification only applies to polylines or polygons in the 2D space.

A *polyline* is an ordered collection of connected segments whose point-based (vector-based) representation is a list of points $< P_1, P_2, \ldots, P_n >$. In DEDALE, a segment P_iP_{i+1} is represented as a tuple featuring a conjunction of 5 constraints: one for the equation of the line collinear to the segment, and 4 for the mbb (we use 4 constraints for the mbb in order to take into account the special cases of vertical/horizontal segments).

The usual vector-based representation of a *polygon* is also the list of points $< P_1, P_2, \ldots, P_n >$ describing its boundary as a closed polyline. Many algorithms have been designed for polygon partition, mostly for polygon triangulation [10, 9, 44, 4, 41]. However, triangulation results in a large, possibly prohibitive, number of convex components in the partition. Given a polygon with n vertices, the number of triangles in the partition is $n-2$. Thus, we would like to minimize the number of convex elements in the partition. There exists an algorithm proposed in [27] which gives the optimal number of convex components. Unfortunately,

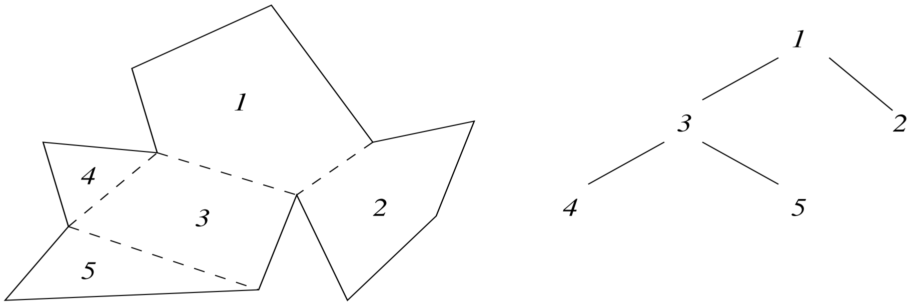


Figure 8: A simple polygon, its convex partition and its adjacency graph

in addition to being rather complex, the algorithm time complexity is large: $O(r^2 n \log n)$ where r is the number of "notches".[3]

Hence, we preferred to implement an algorithm suboptimal in terms of the number of components, but simple to implement and running in quadratic time. It works in two steps: first the non-convex polygon is triangulated into $n-2$ triangles, second the number of convex components is reduced by merging adjacent triangles whose union is still convex.

The first step of the algorithm computes a triangulation of the polygon. The triangulation algorithm we implemented is a variant of the "greedy" $O(n^2)$ algorithm described in [35]. There exists faster (linear) algorithm that computes triangulation but they are difficult to implement and the complexity is not a critical issue here because the module is executed only once, when the data is loaded into the database. The greedy algorithm not only triangulates the polygon but also provides an adjacency tree of the triangles [4].

In the next step [25] the adjacency tree is scanned bottom up: one first tests whether the union of triangle t_1 with another adjacent triangle t_2 is convex. If yes one keeps scanning the tree up and merging a convex polygon with adjacent triangles until the object is no longer convex. The process is then repeated with another leaf if any. Figure 8 illustrates the partitioning of a polygon into convex components and the resulting adjacent tree.

The second step is linear in the number of vertices of the initial polygon. As a result, we get a set of convex polygons whose number of elements, as shown in [25], is never worse than four times the optimal.

A conjunct is created for each resulting convex polygon: the boundary is scanned in counterclockwise order, and for each oriented edge *e* the constraint $a_1x + a_2y \leq b$ representing the half-plane on the left side of *e* is appended to the conjunct. Each conjunct thereby inserted in the database is normalized: there are no redundant nor inconsistent constraints. Notice that the storage order of the constraints in each conjunct is also the order of the segments along the boundary of the convex polygon. Thanks to this order the vector representation can be recomputed by a simple scan.

Table 1 gives some statistics on the convex partitioning of polygons from a SEQUOIA dataset.

Type	# Polys	Avg # pts	% of notches	Tuples/notch	# constr.	# Tuples
Cropland	4 324	58	34.17	1.006	4.36	23
Forests	3 404	133	34.44	0.949	4.44	53
Resid.	7 899	27	25.10	1.20	4.33	10

Table 1: Convex partitioning statistics

If r is the number of notches, the optimal number of components is in $[\lceil \frac{r}{2} \rceil + 1, r+1]$. Table 1 shows

[3] A notch is a vertex such that the internal angle is larger than π.

[4] The nodes of the tree correspond to the triangles. There is an edge between two nodes if the corresponding triangles are adjacent.

that the number of convex components is almost r, the number of notches (0.95) in spite of the algorithm sub-optimality ($\#constr.$ denotes the average number of constraints per tuple). This shows the efficiency of our implementation, but the large ratio (approximately 30) between the number of convex polygons and the number of simple polygons illustrates the need for an efficient representation of nested relations.

For higher dimension data, the key space, whose dimension is not higher than 2, is partitioned as needed (triangulated for TINs, separated into segments for trajectories) and each component is stored together with the corresponding interpolation functions.

Notice that data is loaded in the database in DNF where the disjuncts (convex components) form a partition of the data (no overlap). With the exception of union ($\cup$) and projection (π), each operation in the physical algebra described below preserves this property. In most cases the output of a query will contains only non-overlapping convex components, and this makes easier the construction of the corresponding polygon.

5.3 Constraint manipulation and physical algebra

In this section, the physical algebra of DEDALE is described. This algebra obviously includes the standard physical operations $\sigma, \pi, \times, \bowtie$... for manipulating relational tuples.

Strictly speaking, at the physical level there should be one operation (denoted for example π_{geom}) which performs a geometrical projection and one operation π_{rel} which performs a standard relational projection. Each corresponds to different physical operations implemented separately. For simplification we will use the π notation because it is always clear from the context which physical operation this refers to. In the following we will only present the operators added in order to encapsulate geometrical operations.

As shown in [16] the query evaluation is basically exponential in the dimension of geometric data. In order to master this complexity, the evaluation strategy aims at reducing the geometric computation to computation on 2D data. To this end we use the algorithms from [17] which take advantage of the specific shape of the data in order to evaluate higher dimensional data as if it was 2D data. For instance in the case of TIN, the interpolating functions can be used to transfer 3D geometrical operations into 2D operations over the key. We first illustrate this process with an example, and then develop the algorithms for 2D data.

Interpolation operators

Let us consider the DEM of our sample schema, with name *Country*, and the query language presented in the previous section. A user might express for instance a relational selection of the form

$$q = \text{MAP}_{\lambda X.[\sigma_{\alpha_x x+\alpha_y y+\alpha_h h=\beta}(X.\mathit{geom})]}(\mathit{Country})$$

This computes the part of the country 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.

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

Let s be one of the conjuncts of $X.\mathit{geom}$, represented by $\langle c^s; f^s_h\rangle$ where c^s is the triangle in the key-space (x, y) and f^s_h 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 $h = f^s_h(x, y)$. One can thus substitute h by $f^s_h(x, y)$ in the equation of H. This defines a new object H^s as the set of points (x, y) which satisfies $\alpha_x x + \alpha_y y + \alpha_h f^s_h(x, y) = \beta$: a convex object in the 2D key-space.

Finally, the selection is done by an iteration on the set of conjuncts which performs at each step the intersection of c^s and H^s and keeps the interpolation function unchanged. In other words the computation consists of a computational geometry algorithm in the 2D space.

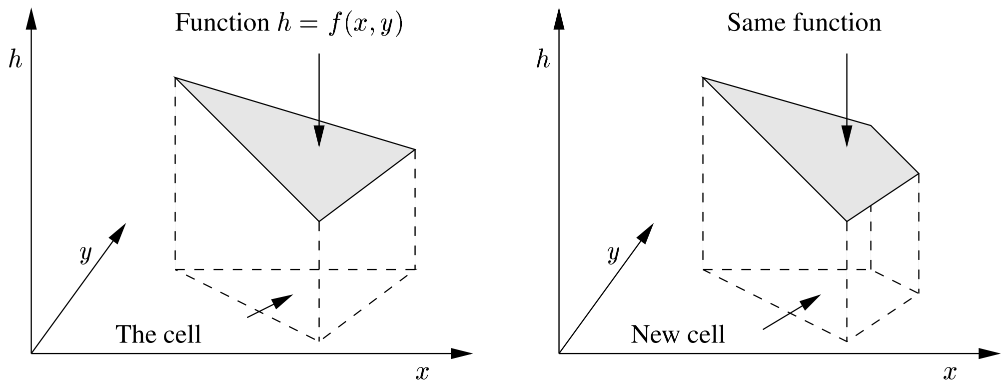


Figure 9: Evaluation of $q = \sigma_{\alpha_x x + \alpha_y y + \alpha_h h = \beta}(s)$

This is depicted in Figure 9. A conjunct s can be seen as consisting of two parts: first a "cell" $c(x, y)$ in the 2D space defines 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.

For most queries, such substitutions are sufficient in order to reduce the manipulation to 2D geometrical computation. The algebra thus contains operators σ^{subst}, π^{subst}, $\bowtie^{subst}$ which perform automatically the necessary substitutions before applying the 2D algorithms given in the following. We refer the reader to [17] for a detailed description of the corresponding operators.

The rest of the physical algebra focuses on 2D operations. All operators will assume that (1) their input is given in a DNF format, and thus consists of a union of convex components, and (2) the result is a relation which is also in DNF.

In order to enforce the normal form above we first need a *Normalization* operator which simplifies each conjunct by eliminating redundancies and unsatisfiable constraints.

Normalization

Conjunct *normalization* is central to the implementation of the algebra. Normalization eliminates redundancy and checks for satisfiability: if the resulting set is empty, then the conjunct is unsatisfiable. The normalization algorithm over 2-dimensional point sets works as follows: first the constraints are scanned in order to count the number of constraints with operator *equal*. Except for degenerated cases, this number e indicates the type of the result: if e is larger than 2, the conjunct is non satisfiable; if $e = 2$, the constraint represents a point, if $e = 1$, a linear object, and if $e = 0$, then every constraint defines a half-plane.

In the latter case, a *divide-and-conquer* algorithm from computational geometry is used for normalizing a conjunct, denoted as *half-plane intersection (HPI)* in the following: (i) the set of half-planes in the conjunct is recursively split until one obtains singletons; this yields a binary tree; (ii) the half-plane in each leaf singleton is intersected with a very large rectangle: this yields a convex polygon in each leaf. The tree is then processed bottom up: at each node of the binary tree, one computes the intersection of two convex polygons. The final convex polygon P eventually delivered by the algorithm is the intersection (if any) of all the half-planes: it is enough then to convert P towards a constraint representation as explained above. The core of the algorithm, excluding the final conversion, is given below.

Function *HPI (s : set of half-planes): a convex polygon*

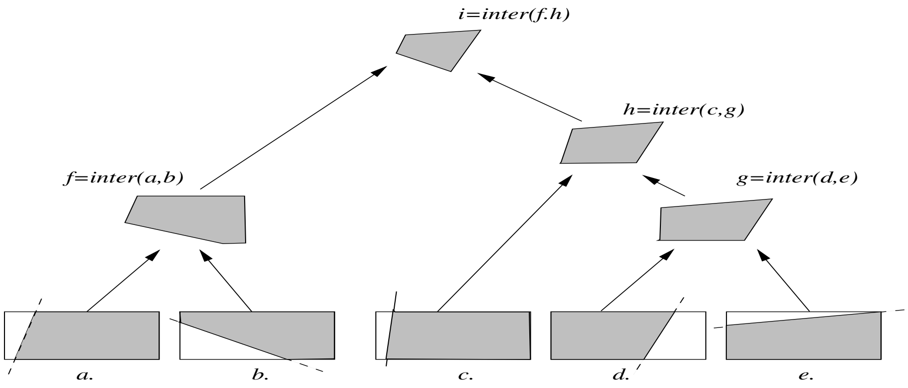


Figure 10: The Half-Plane Intersection Algorithm

begin
  if *Card(s)* ≥ 2 **then**
    return *HPI (s/2)* $\cap$ *HPI (s-s/2)*
  else
    return $s \cap$ *a sufficiently large rectangle*
end

We use as a shorthand notation $s/2$ (resp. $s - s/2$) to denote the first half of the set of half-planes s (resp. the second half). Figure 10 illustrates the process: starting from 5 constraints, one obtains a convex polygon i with four edges. Meanwhile, the constraint c which is redundant with a has been eliminated. Since the intersection of convex polygons is linear in the number of vertices, the complexity of HPI is in $O(n \log n)$ where n is the initial number of constraints.

The *NORMALIZE* algorithm is given in the following[5]: $t.constraints$ denotes the set of constraints in t; CEQ denotes the subset of $t.constraints$ such that *op* is '=' (straight lines), CNE is $t.constraints - CEQ$ (half-planes). The MBB function computes the bounding box of any point-based geometric object. $ToConstraints$ converts a polygon into its constraints representation.

Function *NORMALIZE(conjunct t): a normalized conjunct t'*
begin
  $t'.constraints := \emptyset$; $t'.type := Unknown$; $t'.mbb := \emptyset$; $t'.normal := true$
  if $Card(CEQ) > 2$ **return** t' *// Unsatisfiable*
  else if $Card(CEQ) = 2$ **then** *// A point pt*
  begin
    Check that each constraint in CNE contains pt
    if yes **then**
      t'.constraints $\leftarrow (x = pt.x \wedge y = pt.y)$
      t'.mbb := MBB (pt)
    else *return* t'
  end
  else if $Card(CEQ) = 1$ **then** *// A segment*
  begin
    t'.mbb := MBB(HPI(CNE))
    t'.constraints $\leftarrow CEQ + ToConstraints(MBB(HPI(CNE)))$

[5] To simplify the description, we assume that the input does not contain any degenerated case such as two constraints representing parallel straight lines, or unbounded objects.

```
    end
    else if Card(CEQ) = 0 then // A convex polygon
    begin
        t'.mbb := MBB(HPI(CNE))
        t'.constraint := ToConstraint(HPI (t.constraints))
    end
return t'
end
```

The complexity of *NORMALIZE* is bounded by that of *HPI* and is therefore $O(n \log n)$ as well.

At the relation level, two functions are useful for constraint solving. The first simply applies the *NORMALIZE* function to each of its conjuncts (recall that all relations are assumed to be in DNF). The second, *SATISFY*, Boolean, is set to true if the relation is non empty. *SATISFY* simply scans the conjuncts of the relation, normalizes it and returns *true* as soon as it finds a satisfiable one.

Overlap

OVERLAP takes as inputs two relations, R_1 and R_2, and performs a nested loop over both collections of conjuncts, performing the conjunction of each pair of conjuncts. A simple optimization is possible in order to eliminate some empty conjuncts: two tuples are concatenated only if the intersection of their respective bounding boxes is non empty. The complexity of the algorithm is $O(n_1.n_2)$.

Note that the resulting relation is in DNF but the conjuncts may not be normalized: some constraints in a tuple might be redundant or inconsistent. We don't enforce normalization at this stage because it is a costly operation.

Projection, π

The projection of a tuple in the general case can be computed with the Fourier-Motzkin algorithm. In arbitrary dimension this is a costly operation, basically exponential in the dimension of the data. However, when restricted to dimension 2 there is an easier and faster way to perform projection. First each conjunct is normalized; during normalization, the corresponding *mbb* is computed; then it suffices to project the *mbb* on the correct axis to get the result. In dimension 2, the complexity of the algorithm is $O(n \log n)$.

Join, $\bowtie$, and intersection, $\cap$

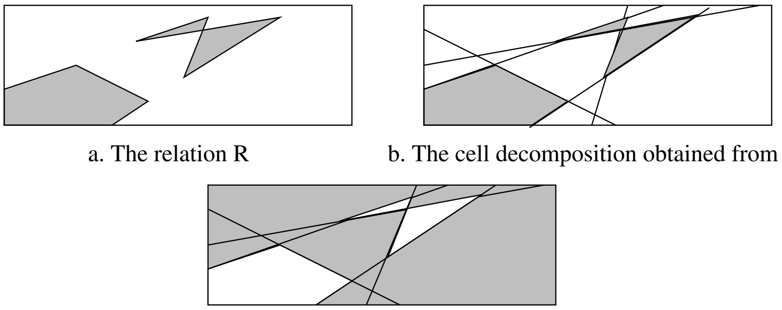
The join is implemented by a direct call to the $OVERLAP$ function. The resulting set is *not* normalized.

Union, $\cup$

$r = r_1 \cup r_2$ is defined as the set of conjuncts which belong either to r_1 or to r_2. It is implemented as the concatenation of the r_1 and r_2 conjuncts lists. The operation has complexity $O(1)$.

Selection, σ

$\sigma_F(R)$ is implemented as *OVERLAP (R, F)*. The result is not normalized: it might contain empty conjuncts. It has for a worst case complexity $O(n)$.



a. The relation R

b. The cell decomposition obtained from R

c. The negation of R

Figure 11: Difference

Difference, $-$

$r = r_1 - r_2$ gives the set of points that are in r_1 but not in r_2. Since $r_2 = \bigvee \bigwedge c_{ij}^2$, its negation is $\neg r_2 = \bigwedge \bigvee \neg c_{ij}^2$ which is a conjunctive normal form (CNF) formula. Transforming a CNF formula into a DNF one is exponential in general, but for dimension 2 it is possible to compute negation in time $O(n^3)$. There exist several algorithms for computing the negation of sets, all are intricate and costly. We implemented the following which has reasonable constants. The idea is roughly to compute a *cell decomposition* of the plane using the constraints of r_1 and r_2. r_1 and r_2 are unions of such *cells* and therefore r is the union of the *cells* that are part of r_1 but not part of r_2 (see Figure 11). The computation of the cells, together with the constraints defining them, can be done in time $O(n^2 \log n)$ where n is the number of constraints in r_1 and r_2. Then we just have to check whether a *cell* c belongs or not to a relation r_i. This can be done in time $O(n)$ since it is enough to check whether a point of c is in r_i. As there might be n^2 such cells, the overall computation time is $O(n^3)$.

5.4 Query processing

In this section we present the most important components of the DEDALE query processor, see Figure 5.

Overview

The design is classical and is illustrated in Figure 12. First a user query is translated into an equivalent algebraic expression by the *parser*. The *type checker* controls the existence of relation names, attribute names and attribute types using schema information. The third step is the *optimization* of the query according to rewriting rules and access paths selection. The optimizer delivers a Query Execution Plan (QEP) which is evaluated. The operators in a QEP will be described in the following.

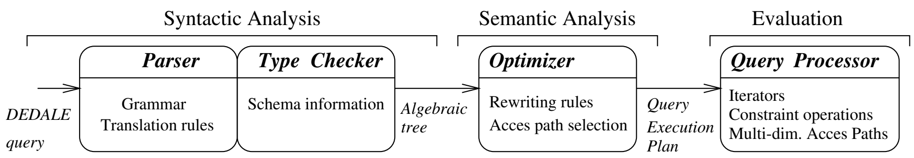


Figure 12: Evaluation of queries

The query engine of BASIS is based on the classical pipelined query execution model described, for instance, in [11]. In this model, query execution is a demand-driven process with iterator functions. Each operator in the query engine is an iterator, and the iterators are connected to each other to form execution plans. This allows for a pipelined execution of multiple operations: a tuple goes from one node to another, from the disk up to the root of the query tree, and is never stored in intermediate structures.

The operators involved in an execution plan are specialized into two forms, depending on the kind of "records" they manipulate.

1. A first class of operators, provided by BASIS, perform a relational-like evaluation involving only the "relational" tuples of a ground relation. Alphanumerical or spatial selections and joins are carried out using some commonly used strategies, and rely on the bounding box of nested relations for the spatial part of the query.

   Indexing is on R*Tree [24, 2], the main class of spatial indices supported by BASIS. A study of the query evaluation strategies for selection and joins with R*trees can be found in [37].

   The part of query evaluation supported by these operators delivers a superset of the result, and corresponds to the so-called "filter step".

2. When it turns out that accessing the nested relations is necessary to refine the result of the filtering step, a nested relation is fetched and a secondary execution is performed using a second form of operators dealing with a constraint representation.

The basic motivation underlying this two-step process, as in more traditional spatial database designs, is that the manipulation of spatial objects is costly because of the intricacy of the geometry. Therefore it pays off to first evaluate a simplified query on a conservative approximation of the geometry, namely the bounding-box. Several studies on spatial databases have investigated queries with complex spatial conditions on bounding-boxes, see for example [36]. The first step allows to detect, at a very low cost, the constraints which are trivially not satisfiable and to remove them from further consideration (*filter step*). For the remaining ones, the exact constraints need to be processed (*refinement step*). In practice, drastic optimization results from the filtering of a large part of the input.

What renders the DEDALE approach particularly suitable to describing this process is the similarity of the languages involved in each step. Indeed, expressions in FO$(0, 1, <, +, -)$ evaluated over systems of constraints represented in the same language are transformed in simplified expressions in the language $FO(<)$, evaluated over "box" objects described with constraints of the form $\alpha < x < \beta$.

We present in the following section a set of rewriting rules for DEDALE algebraic queries which introduces a preliminary evaluation step on box constraints.

Rewriting rules

Consider the operations of intersection, $\cap$, difference, $-$, projection, π, selection, σ, and join, $\bowtie$. For each operation, rewriting rules are given. For the presentation, two sample relations $R(A_1, A_2, \ldots A_n, geo)$ and $R'(A'_1, A'_2, \ldots A'_m, geo')$ are used. We assume two additional attributes box, and box' which represent the bounding box information available at the ground relation level. The rules consist in rewriting a query on attribute geo into an equivalent query featuring a filtering step on box.

Intersection Operation: $\cap$

Intersection between a fixed rectangle $rect$ and the geometric information geo of R

$$MAP_{\lambda X.[X.geo \cap rect]}(R) \equiv MAP_{\lambda X.[X.geo \cap rect]}(\sigma_{box \cap rect \neq \emptyset}(R))$$

Intersection between the geometric parts of two relations R and R'

$$MAP_{\lambda X.[X.geo \cap X.geo']}(R \times R') \equiv$$
$$MAP_{\lambda X.[X.geo \cap X.geo']}\, \sigma_{box \cap box' \neq \emptyset}(R \times R')$$

Difference: $-$

Difference between the geometric part of R and a fixed rectangle $rect$

$$MAP_{\lambda X.[X.geo-rect]}(R) \equiv MAP_{\lambda X.[X.geo-rect]}\,\sigma_{box \cap rect \neq \emptyset}(R) \;\cup\; MAP_{\lambda X.[X.geo]}\,\sigma_{box \cap rect = \emptyset}(R)$$

Difference between the geometric part of two relations R, R'

$$MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}(R \times R') \equiv$$
$$MAP_{\lambda X.[X.geo_R - X.geo_{R'}]}\,\sigma_{box_R \cap box_{R'} \neq \emptyset}(R \times R') \cup MAP_{\lambda X.[X.geo_R]}\,\sigma_{box_R \cap box_{R'} = \emptyset}(R \times R')$$

Projection: π

The projection on one dimension, x_i, of the geometric part geo of R.

$$MAP_{\lambda X.[\pi_{x_i} X.geo]}\, R \equiv \pi_{x_i}(R.box)$$

Selection: σ

Select the geometric objects that contain a given point $P(a_1, .., a_k)$

$$MAP_{\lambda X.[\sigma_{x_1=a_1 \wedge .. \wedge x_k=a_k} X.geo]}(R) \equiv MAP_{\lambda X.[\sigma_{x_1=a_1 \wedge .. \wedge x_k=a_k} X.geo]}\,\sigma_{P \in box}(R)$$

The selection condition $P \in box$ holds if the coordinates of P which correspond to attributes of box satisfy the constraint of box. The rule can be generalized to arbitrary selection criteria.

$$MAP_{\lambda X.[\sigma_F X.geo]}(R) \equiv MAP_{\lambda X.[\sigma_F X.geo]}(\sigma_{F \wedge box \neq \emptyset}(R))$$

Join: $\bowtie$

The spatial join between relations R and R' constructs the pairs of tuples from $R \times R'$ whose spatial attributes satisfy a spatial predicate. There are several types of spatial predicates: *intersect, contains*, and other *topological* predicates, *distance, directional,* etc. We consider only the *intersect joins* also referred to as *overlap* joins. The following rule expresses the classical sequence filter step/refinement step in spatial databases.$\bowtie_{box}$ denotes an overlap join on two collections of bounding boxes.

$$MAP_{\lambda X.[X.geo \bowtie X.geo']}(R \times R') \equiv MAP_{\lambda X.[X.geo \bowtie X.geo']}(R \bowtie_{box} R')$$

Example

The query evaluation is illustrated on the following example:

Give the trajectory of the planes and their height above the ground
when they were over the French territory at an altitude lower than 1000 m.

The DEDALE query is:

select fname, Traj[t, FT.x, FT.y, z[FT.a-TG.h]]
from Flight, Flight.traj **as** FT, Country **as** c, c.geom **as** TG
where FT.x = TG.x **and** FT.y = TG.y
and cname = 'France'
and FT.a $<$ 1000

This query is first translated into the following algebraic expression:

$$MAP_{\lambda X.[fname, \pi_{t,x,y,z[a-h]}(X.geom \bowtie_{x,y} \sigma_{a<1000}(X.traj))]}(\sigma_{cname='France'}(Country) \times Flight)$$

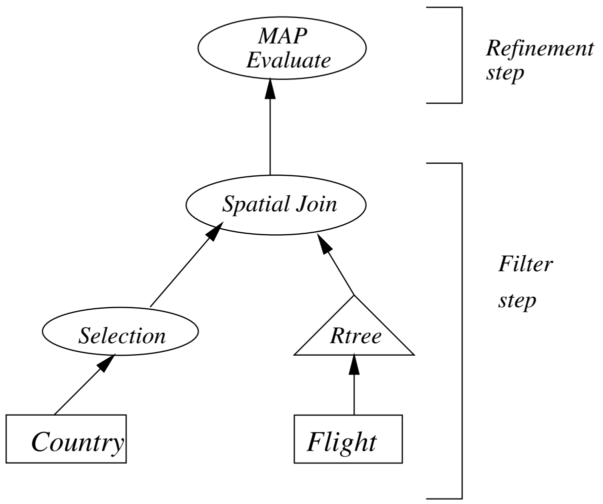


Figure 13: The QEP of the sample query

Using the foregoing classical algebraic equivalences, the above expression is translated into:

$$MAP_{\lambda X.[fname, \pi_{t,x,y,z[a-h]}(X.geom \bowtie_{x,y} \sigma_{a<1000}(X.traj))]}$$
$$(\sigma_{cname='France'}(Country) \bowtie_{box(x,y)} \sigma_{a<1000 \wedge box \neq \emptyset}(Flight))$$

Then the evaluation can be carried out by alternating (1) a relational-like query execution plan (here over the relations *Flight* and *Country*) including an *approximate* evaluation on bounding boxes with (2) an algebraic expression over nested generalized relations with *exact* geometric evaluation. Since the spatial join addresses the (x, y) subspace, we must, at the ground level, join the relations on the projection of their bounding boxes on (x, y) (hence the $\bowtie_{box(x,y)}$ notation).

The QEP for this query is shown on Figure 13. It first scans the *Country* relation and keeps the tuple related to France. The mbb of the French map (which is available at the relational tuple's level) is then used to prune the index on the $Traj$ collection (spatial join node) to retrieve a subset of the planes.

So far the execution only deals with relational tuples. At some point of the evaluation, it is necessary to fetch the nested relations and to perform the algebraic expression over constraints (*refinement step*).

An important issue is to decide at which place this geometric evaluation must take place. In the current implementation of DEDALE, it is always performed as the last operation, as illustrated in Figure 13, because it is likely that the number of relational tuples to consider is minimal at this point.

The nested relations store respectively a list of points describing (in the 3D space) a trajectory, and a list of triangles describing (also in the 3D space) the country.

The *MAP* node gets the geometric data from the disk (see the section devoted to constraint operators in Section 5.3) and computes the result using a nested loop over the sets of symbolic tuples. One obtains, at each step, a pair having the form shown in Figure 14, on the left.

So x, y and a are linear functions of t, and h is a linear function of (x, y). The symbolic operation is a conjunction of all these constraints. Now, after an easy substitution of variables, x and y are replaced by the proper function of t in all the inequality constraints. As a result one obtains x, y, a and h as functions of t (which is the *key* of the result), and a more complex conjunction of constraints over the single variable t (see Figure 14, on the right).

The process ends up with the normalization of the result: the system of equations on t is solved, keeping unchanged the other constraints. The result is depicted in Figure 15. In that case the result is trivially

From *Traj*	From *Country*
$0 \leq t \leq 2$	$-4x + y + 1 \leq 0$
$x = 2t + 1$	$y - 3 \leq 0$
$y = t + 2$	$3x + y - 18 \leq 0$
$a = 5t - 23$	$h = x + y + 3$

The new symbolic tuple
$0 \leq t \leq 2$
$-4(2t + 1) + (t + 2) + 1 \leq 0$
$(t + 2) - 3 \leq 0$
$3(2t + 1) + (t + 2) - 18 \leq 0$
$x = 2t + 1$
$y = t + 2$
$h = (2t + 1) + (t + 2) + 3$
$a = 5t - 23$

Figure 14: Joining two symbolic tuples

obtained in linear time in the size of this system. The constraint solving algorithm delivers an interval and linear functions for x, y, a and h. Finally a new attribute name z is generated for the newly created variable containing the result of the term $a - h$.

The normalized result
$0 \leq t \leq 1$
$x = 2t + 1$
$y = t + 2$
$h = 3t + 6$
$a = 5t - 23$

The final result after projection
$0 \leq t \leq 1$
$x = 2t + 1$
$y = t + 2$
$z = 2t - 29$

Figure 15: Final result

In summary, if the system is based on a commercial DBMS, the only add-on is an implementation of our algorithms to deal with spatial data viewed as infinite relations. If, on the other hand, the system is based on a specialized software such as Arc/Info, our data model defines a logical level enabling the user to access and manipulate spatial data with a standard and easy-to-use declarative language.

5.5 Graphical Interface

The prototype DEDALE provides a Graphical User Interface (GUI), written in Java. It permits to query the database with parameters such as points or rectangles, and to display query results including maps and other thematic data (see Fig. 6).

The GUI transfers the query to the system as a string and obtains as a result a string representing the output of the query in DNF. The main difficulty of the GUI is thus to convert data from their constraint representation to their vector representation, which is finally used to display the result. Two techniques are used:

- If the data comes directly from the database, for example, if it results from a *window* query, then it uses the adjacency graph stored in the database. From this graph the polygons can be easily reconstructed in time linear in the number of constraints;
- Otherwise, the data has been modified by a query, and the adjacency graph has not been maintained, for example after a *clipping* query. Maintaining the adjacency graph was rejected in DEDALE because it is complicated and expensive. DEDALE uses instead a function that, given a set of polygons, returns

a set of non-intersecting, non-adjacent polygons. This can be achieved in time $O(n \log n)$ where n is the number of constraints involved. We implemented a *sweep-line* algorithm as described in [35].

The current implementation of the interface can only display 2D data plus, up to some extend, some 2D-temporal data.

6 Conclusion

This paper presented DEDALE, a database system for representing and manipulating geometric data in arbitrary dimension. DEDALE relies on a logical model based on linear constraints. The constraint model allows for a uniform representation whether the information is alphanumeric 1-dimensional data or multi-dimensional data such as geometric data, spatio-temporal data or elevation data. DEDALE provides an abstract and non specialized data model and a user declarative query language based on SQL.

The power and practical relevance of the approach was validated by a prototype. This implementation aimed at illustrating that the proposed data model can be integrated in a relational DBMS, or any post-relational DBMS, such as object-oriented DBMS or object-relational DBMS, with minimal effort.

The DEDALE prototype was demonstrated on several geographical applications, among which an elegant high level modeling of DEMs is noteworthy. Its current implementation contains a database kernel, BASIS, with storage capabilities, index functionalities and a standard set of relational algebraic operators. On top of it the new 2D algebraic operators described in this paper have been added and a graphical user interface allows the user to ask queries and display their output. The language described in Section 4 is implemented with a transformation to the algebra $\text{ALG}_{(<,+,-)}$. The optimization however is so far limited to the rewriting using bounding boxes.

The constraint model provides an algebraic query language whose formal foundations constitute a basis for query optimization. We gave a set of rewriting rules as a first step, but optimization specific to the abstract representation of geometric data remains a challenging issue which must be further explored in future work.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*tree : An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, pages 322–331, 1990.

[3] A. Belussi, E. Bertino, and B. Catania. Manipulating Spatial Data in Constraint Databases. In *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, pages 115–141. Springer Verlag, LNCS 1262, 1997.

[4] B. Chazelle. Triangulating a Simple Polygon in Linear Time. *Discrete and Computational Geometry*, 6:485–524, 1991.

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

[6] M. J. Egenhofer and R. Franzosa. Point-Set Topological Spatial Relations. *Int. J. Geo. Info. Sys.*, 5(2), 1991.

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

[8] 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.

[9] A. Fournier and D.Y. Montuno. Triangulating Simple Polygons and Equivalent Problems. *ACM Transactions on Graphics*, 3:153–174, 1984.

[10] M.R. Garey, D.S. Johnson, F.P. Preparata, and R.E. Tarjan. Triangulating a Simple Polygon. *Information Processing Letter*, 7(4):175–180, 1978.

[11] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[12] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. DEDALE: A Spatial Constraint Database. In *Proc. Intl. Workshop on Database Programming Languages*, pages 38–59, 1997.

[13] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. The DEDALE Prototype. In G. Kuper, B. Libkin, and J. Paradaens, editors, *Constraint Database Systems*. Springer Verlag, 2000.

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

[15] S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE System for Complex Spatial Queries. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 213–224, 1998.

[16] S. Grumbach, P. Rigaux, and L. Segoufin. On the Orthographic Dimension of Constraint Databases. In *Proc. Intl. Conf. on Database Theory*, pages 199–216, 1999.

[17] S. Grumbach, P. Rigaux, and L. Segoufin. Manipulating Interpolated Data is Easier than you Thought. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 2000.

[18] S. Grumbach, P. Rigaux, and P. Veltri. Hierarchical Optimization of Linear Constraint Processing. In *Convegno Nazionale su Sistemi Evoluti per Basi di Dati (SEBD'2000)*, 2000.

[19] C. Gurret, Y. Manolopoulos, A. Papadopoulos, and P. Rigaux. The BASIS System: a Benchmarking Approach for Spatial Index Structures. In *Intl. Workshop on Spatio-Temporal Database Management (STDBM'99)*, LNCS 1678, 1999.

[20] R. H. Güting. Geo-Relational Algebra : A Model and Query Language for Geometric Database Systems. In *Proc. Intl. Conf. on Extending Data Base Technology (EDBT)*, pages 506–527, 1988.

[21] R.H. Güting. Gral: An Extensible Relational Database System for Geometric Applications. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 1989.

[22] R.H. Güting. An Introduction to Spatial Database Systems. *The VLDB Journal*, 3(4):357–399, 1994.

[23] R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems (TODS)*, 25(1):1–42, 2000.

[24] A. Guttman. R-trees : A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, pages 45–57, 1984.

[25] S. Hertel and K. Mehlhorn. Fast Triangulation of Simple Polygons. In *Proc. Intl. Conf. on Foundations of Computer Theory*, LNCS No. 158, pages 207–218. Springer-Verlag, 1983.

[26] 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.

[27] J.M. Keil. Decomposing a Polygon into Simpler Components. *SIAM Journal of Computing*, 14(4):799–817, 1985.

[28] G. Kuper, L. Libkin, and J. Paradaens, editors. *Constraint Databases*. Springer Verlag, 2000.

[29] G. Kuper, S. Ramaswamy, K. Shim, and J. Su. A Constraint-Based Spatial Extension to SQL. In *Proc. Intl. Symp. on Geographic Information Systems*, 1998.

[30] G. Kuper and M. Scholl. Geographic Information Systems. In G. Kuper, B. Libkin, and J. Paradaens, editors, *Constraint Database Systems*. Springer Verlag, 2000.

[31] R. Laurini and D. Thompson. *Fundamentals of Spatial Information Systems*. Number 37 in The A.P.I.C. Series. Academic Press, New York, 1992.

[32] Bruce Momjian. *PostgreSQL, Introduction and Concepts*. Addison Wesley, 2000.

[33] S. Morehouse. The Architecture of ARC/INFO. In *Proc. Intl. Symp. on Computer-Assisted Cartography (Auto-Carto 9)*, pages 266–277, 1989.

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

[35] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[36] Dimitris Papadias, Yannis Theodoridis, Timos K. Sellis, and Max J. Egenhofer. Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 92–103. ACM Press, 1995.

[37] A. Papadopoulos, P. Rigaux, and M. Scholl. A Performance Evaluation of Spatial Join Processing Strategies. In *Proc. Intl. Conf. on Large Spatial Databases (SSD)*, 1999.

[38] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases*. Morgan Kaufmann, 2001.

[39] M. Scholl and A. Voisard. Thematic Map Modeling. In *Proc. Intl. Symp. on Large Spatial Databases (SSD)*, LNCS No. 409, pages 167–192. Springer-Verlag, 1989.

[40] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[41] R. Seidel. A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons. *Computational Geometry*, 1(1), 1991.

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

[43] 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.

[44] R.E. Tarjan and C.J. van Wyk. An $O(n \log \log n)$ Time Algorithm for Triangulating a Simple Polygon. *SIAM Journal of Computing*, 17(1):143–178, 1988.

[45] C. D. Tomlin. *Geographic Information Systems and Cartographic Modeling*. Prentice-Hall, 1990.

[46] J.D. Ullman. *Database and Knowledge Base Systems*. Computer Science Press, 1988.

[47] 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.

[48] 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).

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