Dual Grid: A New Approach for Robust Spatial Algebra Implementation*

Jose Antonio Cotelo Lema and Ralf Hartmut Güting
Praktische Informatik IV, Fernuniversität Hagen
D-58084 Hagen, Germany
{Jose-Antonio.Cotelo-Lema, gueting}@fernuni-hagen.de

8th May 2000

Abstract

Systems of spatial data types and operations, or *spatial algebras*, are fundamental for the implementation of spatial database systems. Several designs of such algebras have been proposed in the last decade, and recently commercial DBMS offer such algebras in the form of extension packages (e.g. "data blades"). However, actual implementations are generally severely restricted when compared to designs in the literature. A main reason is that at the implementation level one cannot further ignore the problems of robustness and topological correctness arising from the discrete number representations used in computers. Therefore, implemented packages either avoid "problematic" operations, or accept inconsistencies and topological errors in the answers of queries due to rounding effects.

The ROSE algebra [GS95], proposed and implemented earlier, goes a long way towards avoiding such problems, since it was defined from scratch with robustness problems in mind. It is founded on a discrete geometric basis called a *realm*. The ROSE algebra guarantees a correct behavior of operations and has an entirely robust implementation. Unfortunately, the realm concept and its interaction with DBMS are difficult to implement, and certain kinds of topological problems still remain.

In this paper we introduce the concept of *dual grid*, which provides a new approach for the representation of spatial information that avoids any robustness and topological correctness problem and allows a less restrictive implementation of spatial algebras. As an example, we show how can it be used for implementing the ROSE algebra *without realms*, and show that such an implementation does not suffer from the side effects and disadvantages of the original realm-based one.

1 Introduction

In the past, a big research effort has been spent in the development of database systems able to deal with spatial information. With the appearance of the extensible database technology some of this effort has moved to the development of modules to extend the actual database models (e.g. the relational model) and implementations with support for spatial data, allowing an integrated management of both spatial and non-spatial information.

*This work was partially supported by the CHOROCHRONOS project, funded by the EU under the Training and Mobility of Resarchers Programme, Contract No. ERB FMRX-CT96-0056.

Within this effort, of fundamental importance has been the development of generic spatial algebras, as for example [Güt88, SH91, Ege94, GS95]. Such algebras pursue two basic goals. On the one hand, they provide a set of spatial data types able to suitably and efficiently represent the kind of spatial data managed in a wide set of application environments. On the other hand, they offer a suitable set of operations over these types that fulfils the requirements of those applications.

The main problem in translating the expressiveness of these generic spatial algebras to the real world (that is, commercial databases) is that they are usually defined over a continuous domain, but for their implementation we need to use discrete representations for the spatial types. For example, a value of type *point* is represented as a pair of coordinates, each of them represented as a 32 bit signed integer number, each of the segments of a *line* is represented by its two end points and each of the polygons of a *region* value is represented as a sequence of points (the vertices).

The problems arising from such a restriction are far from trivial. For example, the representation of values is not closed any more under the set operations (e.g. forming the union or intersection of regions). The straightforward solution to such problems (as for example, to approximate the non-representable points by the closest representable ones) makes the operations closed,[1] but violates the basic properties of set theory. For example, for regions A and B, the laws $A \subseteq (A \cup B)$, $(A \cap B) \subseteq A$ or $(A \setminus B) \cap B = \emptyset$ do not hold any more, generating inconsistencies between the answers. Whereas in certain domains (e.g. graphical user interfaces) such rounding errors may be acceptable, they cannot be tolerated in query evaluation, since they may lead to wrong answers. For example, the intersection of a river and a highway may be found to lie neither on the river nor on the highway.

Actual spatial extensions for commercial databases can usually be classified in one of the following two groups, depending on their approach to this problem:

1. They do not provide any operation that is not closed under the selected discrete representation. That usually means that they provide only predicates over spatial data types and operations for composing and decomposing such objects (e.g. operations for constructing the segment between two given points).

2. They provide the whole set of operations, but for those operations that are not closed over the selected discrete representation they return an approximation of the result.

Examples of the first group of spatial implementations are *Illustra 2D Spatial Datablade* [Ill94] and earlier versions of Oracle's spatial extension (*Oracle8 Spatial Cartridge*) [Ora97]. *Illustra's* extension provides data types *Point*, *Line* (in the mathematical sense), *Segment*, *Path* (a polyline), *Polygon* and *Polygon Set* (a region as a set of polygons with holes), apart from some other data types as *Circle*, *Ellipse* or *Box*. The operations that it provides to the user are basically predicates and operations for decomposing/constructing a value (for example, retrieving the n-th *Line Segment* of a *Path* or constructing the *Line Segment* having as end points two given *Point* values).[2] Any operations that become non-closed over the discrete representation used, as the set operations, are not provided.

[1] Artificially closed, because now we are not implementing the *intersection* operation (for instance), but an *approximated intersection* operation.

[2] The only exception to this are operations that are already expected to return approximated values, as getting the aproximation of an *Ellipse* value as a *Polygon* or getting the bounding box of any spatial value.

DB2's Spatial Extender [Dav98] (by *ESRI*, available also under *Informix*) and recent versions of Oracle's extension (*Oracle8i Spatial Cartridge*) [Ora99] are examples of implementations belonging to the second group. DB2's extension provides a wide set of data types, as *point*, *points*, *polylines* and *polygons*, as well as sets of them. It provides also a wide set of operations, including the set operations, but, as mentioned, returns only approximate answers.

A proposal in the area of spatial databases that addresses the robustness problem already at the conceptual level is the ROSE algebra [GS95]. The ROSE algebra uses an underlying discrete geometric basis called a *realm* [GS93]. Intuitively, a realm contains a consistent representation of all geometric data of an application. All numerical problems are treated at the realm level. Values of spatial data types are defined on top of realms which ensures that all operations (including the set operations) return consistent answers (for example, all the laws $A \subseteq (A \cup B)$, $(A \cap B) \subseteq A$ or $(A \setminus B) \cap B = \emptyset$ hold in the implementation). However, some side problems are introduced by the use of *realms* that make the implementation of such an algebra in existing commercial database systems difficult. We will discuss these problems in more detail below.

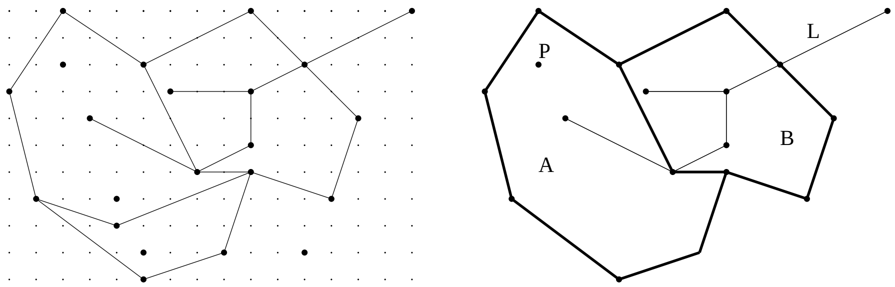
Our goal in the rest of the paper is to propose a solution for solving the closure and consistency problems for the spatial operations proposed in the ROSE algebra, without adding the side effects introduced by the *realms* approach. The basic idea is to control the resolutions for the representations of points and of line equations and so to guarantee that all intersections can be represented without error.

The problem of robustness and topological correctness of geometric computation has of course also been addressed in the computational geometry literature [DS90, For85]. One can distinguish perturbation-free approaches (e.g. [KM83, OTU87]) where the idea is to perform geometric computations with sufficiently high precision such that no errors occur; the task is to determine how many digits are needed to represent the result of the used geometric primitives exactly. Perturbation approaches (e.g. [DS90, GM95, Mil89, Sch94]) allow one to slightly change the input data or the results of computations in order to be able to represent data at a fixed level of precision. In this context, the realm-based ROSE algebra can be viewed as an application of the perturbation approach within spatial databases, and the dual grid based ROSE algebra proposed in this paper uses instead a perturbation-free technique. Compared to the computational geometry setting in general, in this paper we study a special situation arising from the fact that the set of operations of the ROSE algebra has been designed specifically to be closed over a realm basis. This means that no new geometries are created by ROSE operations (only existing points and line segments are rearranged) which is also crucial for the dual grid approach presented here.

In Section 2 we review the ROSE algebra and the *realm*-based approach proposed in it. In Section 3 we introduce the concept of *dual grid* and show its suitability for solving the closure and consistency problems. In Section 4 we show how the original implementation of the ROSE algebra [GdRS95] can be adapted to use the *dual grid* approach, inheriting all its advantages and solving the open problems of the original implementation based on *realms*. Finally, Section 5 concludes the paper and points to future research directions.

2 Review: The Realm-Based Approach

In [GS95] the ROSE algebra is proposed. It offers the spatial data types *points* (a set of isolated points in the plane), *lines* (a set of line segments) and *regions* (a set of non overlapping polygons



(a) Elements in a realm (b) Some objects over the realm

Figure 1: Example of objects over a *realm*

with holes). The set of operations provided includes a wide set of spatial predicates as well as binary operations over spatial types, including the set operations. In contrast to other proposals, the definition of the ROSE algebra takes into account the fact that values need to be finitely represented, and introduces an underlying discrete geometric basis called a *realm* for ensuring the consistency of answers returned by the DBMS.

A *realm* is basically a finite set of points and non-intersecting line segments over a discrete domain (a *grid*). Spatial values are represented in the database in terms of points and segments present in the realm. For example, given the realm in Figure 1, A and B would be valid <u>*regions*</u> values, L would be a valid <u>*lines*</u> value and P would be a valid <u>*points*</u> value.

Whenever a new spatial object (represented at the lowest level in terms of points and line segments) is inserted into the database, its points and segments are inserted into the *realm* and the spatial object is rewritten in terms of elements of the new *realm*.

For each point or segment that is inserted in the *realm*, some extra steps must be followed to ensure that the resulting *realm* is a valid one:

1. When a new point P is inserted, any segment S_r in the *realm* containing P in its *interior* (i.e. P lies on S_r but is not an end point of S_r) is split into two new segments connecting the original end points with P. The original segment is replaced by the new ones in the *realm*, and any spatial object using it is modified to reflect this change.

2. When a new segment S is inserted into the *realm*, its end points are also inserted. If S contains a realm point P_r in its interior, it is split at P_r. Furthermore, all intersections of S with segments already present in the realm are determined. For each true intersection point P_i of S with some segment S_i a closest grid point $\overline{P_i}$ is determined. Then, a *redrawing* of S is performed which transforms S into a polyline (a sequence of line segments) passing through all the grid points $\overline{P_i}$. (This process of redrawing is explained in more detail below.) Each segment S_i is redrawn as well to pass through $\overline{P_i}$, its grid intersection point with S. Finally, all changes in segments are propagated to the database to modify the spatial objects using these segments.

To explain the redrawing (originally proposed in [GY86]), the concept of an *envelope* is needed. For a segment S, its envelope consists (roughly) of the grid points immediately below or above it. Figure 2 shows the grid points forming the envelope of S.

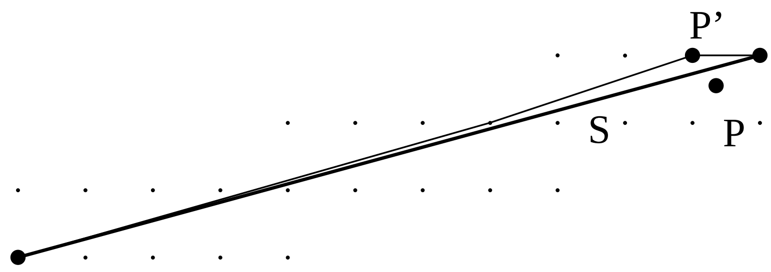


Figure 2: Redrawing of a segment S

Intuitively, the process of redrawing can be easily understood if we view the segment S as a rubber band and the points on the envelope of S as nails on a board. When a second segment S_r is found that intersects it in a point P, we grab the rubber band at point P and pull it around the closest nail P'. As a result, the rubber band will touch one or more nails of the board (of the envelope). The resulting list of segments is the redrawing of S. This is also illustrated in Figure 2. This approach guarantees that the polyline into which the segment is decomposed always remains within the envelope of the original segment. For more details, see [GS93].

The use of realms as the basis for the representation of spatial objects has the following advantages:

- It enforces *geometric consistency* of spatial objects. For example, the common part of the boundary of two adjacent regions will belong completely to both regions and be exactly the same in both of them.

- It guarantees *closure properties* for the spatial objects. For example, the intersection of two regions A and B can always be represented, because it will be just a new combination of segments of the *realm* belonging to regions A and B.

- Simplicity and efficiency of spatial operations. Given that no intersection points between segments need to be detected, the algorithms for implementing spatial operations are more simple and more efficient.

- It ensures *numerical correctness* and *robustness* of geometric computations. Those problems arise basically in the computation of intersection points of line segments, which in general will not lie on the grid. As there is no pair of segments in the *realm* that intersects, this problem does not arise when performing geometrical computations. All these problems are solved at the realm level, whenever a new value is inserted into the database.

However, the realm approach has also some disadvantages, some of which make the use of a *realm*-based representation in commercial databases (e.g. offering the ROSE algebra as an extension module) difficult:

- Relationships between points and segments can change after rewriting. The rewriting of segments can cause points of the *realm* that previously did not lie on the segment to now lie on it. This can change topological relationships between point objects and objects of type *lines* and *regions*. This problem could be solved (as pointed out in [GS93]) by adding the restriction that points of the *realm* cannot lie on the envelope of segments already in the *realm*.

- Complexity of data structures. The original implementation proposed in [GS95], with a layer representing the realm and the representation of spatial objects referencing the elements of the *realm* of which they are composed, results in a rather complex representation. The use of *virtual realms*, as proposed in [MPF+96, FDP+99], simplifies it. The spatial objects are directly represented in the database in a *realmized* form (making intersection points explicit), but without further references to *realm* elements. Instead of storing the corresponding *realm* explicitly, the portion of it that is needed is dynamically computed whenever a new insertion is performed, reducing in that way the size of the data stored in the database and the time needed to recover the spatial objects from it.

- Space overhead. Due to the rewriting process, a segment of a spatial object can be decomposed into multiple subsegments in the *realm*. If the overlapping of spatial objects in one area of the plane is high, the overhead produced can lead to a considerable increase in the space needed to store the objects of such an area. Indeed, the fact of representing the object with a higher number of segments will negatively affect the efficiency of the spatial operations.

- Complexity and efficiency of updates. Given that the update of spatial objects in the database can lead to a cascade of rewritings, the efficiency of update operations can be seriously affected, mainly if the spatial object is inserted into an area of the plane with a high concentration of objects.

- Values can change without directly modifying them. A region A stored in the database can slightly change its value just because the insertion of a second region B has caused the rewriting of some of the segments in the *realm* that compose it. This will be true even if the user that inserted B has no access rights for modifying A.

- Consistency over time: although consistency between answers is guaranteed as long as no new values are introduced in the database, it is not guaranteed between operations performed with different contents of the database. After introducing a new value in the database, the answers of the algebra operations will be consistent again, but perhaps not with respect to the answers given before inserting the new value. For example, the answer to $(A \setminus B) \cup (A \cap B)$ can be different from A if a third object C has been inserted into the database between the computation of $(A \setminus B)$ and $(A \cap B)$. This might complicate transaction management. Moreover, it is a problem for applications in which the time dimension (transaction time) is relevant and hence the consistency of answers over time is also a requirement.

3 The Dual Grid Approach

The reason why the use of *realms* solves most of the problems when implementing the ROSE algebra is that the set of operations of this algebra has a nice property: given the set of segments and points defining the boundary of the arguments of the operation, the result (if it is a spatial value) can be constructed using only segments and points already present in the input arguments, or subsegments or points resulting from the intersection of input segments.

As the redrawing process performed in the insertion of new segments in the *realm* decomposes them into subsegments, making the intersection points explicit, the operations of the ROSE algebra only need to recombine (in the appropriate way) the elements of the input values for building the result.

However, the fact that in general the intersection points do not lie on the underlying grid causes the rewriting process to modify the exact value represented, and the insertion of new objects can again modify the value of objects previously stored. This is the main reason of the unsolved problems of the *realm* approach, as for example the inconsistencies between answers over time.

A solution for solving those remaining problems would be to find a discrete representation for spatial data types that ensures that any intersection point found and used in the redrawing process can be exactly represented, avoiding in that way to modify the spatial value to be stored in the database or any other value already stored. Instead, only their representations would change.

In a rough description, what we need is a domain set G_p for points and a domain set G_s for segments fulfilling the following three properties:

1. The end points of any segment of G_s are points of G_p.
2. For any two non-collinear segments $S_1, S_2 \in G_s$, their intersection point $P = S_1 \cap S_2$, if it exists, belongs to G_p.
3. For any segment $S = (P_a, P_b)$, $S \in G_s$ and any $P \in G_p$, if $P \in S$ and $P \notin \{P_a, P_b\}$, then the subsegments $S_b = (P_a, P)$ and $S_b = (P, P_b)$ belong to G_s.

Such a pair of domains G_p and G_s we will call a *dual grid*. If valid spatial values are restricted to those whose boundary can be represented using only points and segments belonging to G_p and G_s, the insertion of a new spatial object will only lead (in the worst case) to a change in the representation of some of the spatial objects in the database (due to the rewriting process), but their values will remain unmodified, and hence the consistency between answers over time will also be guaranteed.

The requirements above mean, first of all, that the domain $Coord$ over which the coordinates of points in G_p are defined has to be closed under the arithmetic operations $+$, $-$, $*$, and $/$. This requirement is needed to be able to represent the intersection point of two segments. A subset of the real numbers that fulfils such a requirement are the rational numbers ($\mathbb{Q}$). Given the obvious need of a finite representation, the coordinate domain must be restricted to a subset of the rational numbers.

Definition 3.1. The domain of point coordinates $Coord(n, m)$ is defined as the set of rational numbers whose numerator and denominator can be represented with integer numbers of n and m bits, respectively,[3] that is, $Coord(n, m) = \{x \in \mathbb{Q} \mid x = \frac{num}{den},\ num, den \in \mathbb{Z},\ |num| < 2^n,\ 0 < den < 2^m,\ n, m \in \mathbb{N}\}$.

We will see later on that the use of a $Coord(n, m)$ domain for the coordinates will be enough for defining a closed representation.

[3] We are intentionally ignoring the bit required for representing the sign of these numbers. To take it into account would only make these explanations more complex. Just keep in mind that whenever we speak about a numeric value the extra bit required for the sign is implicit.

Definition 3.2. A grid of points $G_p(n, m)$ is defined as the set of points that can be represented with coordinates in $Coord(n, m)$, that is, $G_p(n, m) = \{(x, y) \mid x, y \in Coord(n, m)\}$.

Proposition 3.1. $G_p(n, m) \subseteq G_p(n + i, m + j)$, $\forall i, j \geq 0$.

The proof is really obvious and hence we will omit it.

Once the set of valid points G_p has been defined, the set of valid segments G_s must be defined in such a way that the intersection of two segments (when the result is a point) belongs to G_p.

Definition 3.3. A grid of segments $G_s(n, m)$ is defined as the set of segments $S = (P_a, P_b)$, such that $P_a, P_b \in G_p(n, m)$ and their supporting lines can be represented with equations $A \cdot x + B \cdot y = C$ having integer coefficients, with $|A|$ and $|B|$ smaller than $\sqrt{2^{m-1}}$ and $|C| < \frac{2^{n-1}}{\sqrt{2^{m-1}}}$.

The bounds on the values of the coefficients A, B and C ensure that the intersection point of any pair of segments $S_1, S_2 \in G_s(n, m)$ belongs to $G_p(n, m)$.

Theorem 3.1. The intersection of any two segments of $G_s(n, m)$ (if not empty) is either a segment of $G_s(n, m)$ or a point of $G_p(n, m)$. Formally, $\forall S_1, S_2 \in G_s(n, m)$, $(S_1 \cap S_2 = \perp) \vee (S_1 \cap S_2 \in G_s(n, m)) \vee (S_1 \cap S_2 \in G_p(n, m))$.

Proof. If $S_1 \cap S_2$ is a segment, it must belong to $G_s(n, m)$, because the resulting segment must have the same supporting line as S_1 and S_2 (and hence fulfil the requirements for the coefficients) and each of its end points is an end point of either S_1 or S_2, and therefore belongs to $G_p(n, m)$.

If $S_1 \cap S_2$ is a point, either $S_1 \cap S_2$ is an end point of both S_1 and S_2 (because they meet at their end point) or, given $r: A_r x + B_r y = C_r$ and $s: A_s x + B_s y = C_s$ the supporting lines of S_1 and S_2 respectively,

$$S_1 \cap S_2 = P = \left(\frac{\begin{vmatrix} C_r & B_r \\ C_s & B_s \end{vmatrix}}{\begin{vmatrix} A_r & B_r \\ A_s & B_s \end{vmatrix}}, \frac{\begin{vmatrix} A_r & C_r \\ A_s & C_s \end{vmatrix}}{\begin{vmatrix} A_r & B_r \\ A_s & B_s \end{vmatrix}} \right) = \left(\frac{num_x}{den}, \frac{num_y}{den} \right)$$

Let us denote by $\|A\|$, $\|B\|$ and $\|C\|$ the maximum values that $|A|$, $|B|$ and $|C|$ can have, respectively. Then,

$$|num_x| = |B_s C_r - B_r C_s| \leq 2 \cdot \|B\| \cdot \|C\| < 2 \cdot \sqrt{2^{m-1}} \cdot \frac{2^{n-1}}{\sqrt{2^{m-1}}} = 2^n$$

In the same way,

$$|num_y| = |A_r C_s - A_s C_r| \leq 2 \cdot \|A\| \cdot \|C\| < 2 \cdot \sqrt{2^{m-1}} \cdot \frac{2^{n-1}}{\sqrt{2^{m-1}}} = 2^n$$

$$|den| = |A_r B_s - A_s B_r| \leq 2 \cdot \|A\| \cdot \|B\| < 2 \cdot \sqrt{2^{m-1}} \cdot \sqrt{2^{m-1}} = 2^m$$

Therefore, given that $|num_x| < 2^n$, $|num_y| < 2^n$ and $|den| < 2^m$, it is proved that $S_1 \cap S_2 \in G_p(n, m)$. $\square$

Proposition 3.2. $\forall i, j \geq 0,\ j \leq 2 \cdot i : G_s(n, m) \subseteq G_s(n+i, m+j)$.

Proof. We have already seen that $G_p(n, m) \subseteq G_p(n+i, m+j)$. Hence, all what we have to prove is that for any segment S belonging to $G_s(n, m)$ the coefficients of its supporting line also fulfil the requirements for $G_s(n+i, m+j)$. For coefficient A, it is clear that if $A < \sqrt{2^{m-1}}$ then also $A < \sqrt{2^{(m+j)-1}}$. In the same way, this is shown for coefficient B. For coefficient C, we can see that if $C < \frac{2^{n-1}}{\sqrt{2^{m-1}}}$ then also $C < \frac{2^{(n+i)-1}}{\sqrt{2^{(m+j)-1}}}$, because

$$\frac{2^{(n+i)-1}}{\sqrt{2^{(m+j)-1}}} = \frac{2^{n-1}}{\sqrt{2^{m-1}}} \cdot \frac{2^i}{\sqrt{2^j}}$$

and

$$j \leq 2i \Rightarrow \sqrt{2^j} \leq 2^i \Rightarrow \frac{2^i}{\sqrt{2^j}} \geq 1$$

Hence, $G_p(n, m) \subseteq G_p(n+i, m+j)$. □

Definition 3.4. A *dual grid* $DG(n, m)$ is defined as a pair of grids $G_p(n, m)$ and $G_s(n, m)$, where $G_p(n, m)$ is a grid of points and $G_s(n, m)$ is a grid of segments. Formally speaking, $DG(n, m) = G_p(n, m) \cup G_s(n, m)$.

The *dual grid* defined above fulfils the three requirements identified previously, and hence its use as the base for the representation of spatial objects ensures that the insertion of new elements into a *realm* does not modify the value of any spatial object in it. As a result, it solves the consistency problems over time, as well as most of the remaining problems of the *realms*-based approach.

Corollary 3.1. $\forall i, j \geq 0,\ j \leq 2 \cdot i$, any element of a *dual grid* $DG(n, m)$ is also an element of a *dual grid* $DG(n+i, m+j)$.

Proof. Obvious given that $G_p(n, m) \subseteq G_p(n+i, m+j)$ and $G_s(n, m) \subseteq G_s(n+i, m+j)$. □

This last property defines the restrictions that have to be fulfilled to ensure that data provided in one *dual grid* resolution can be imported into a table where a different resolution is used (for example, when importing data from another database). It also provides a way of improving the resolution of the *dual grid* used by a set of spatial data without altering at all such data.

For the successful use of the *dual grid* approach as the base for the representation of spatial objects, however, it would be interesting to provide some extra properties to allow a better interaction between database and external applications:

- Data in the database can be exported using a resolution B for points (including end points of segments) and B' for segment slopes.
- Data using a resolution B for points and a resolution B' for segment slopes can be loaded into the database.
- Data using a resolution A for points and end points of segments, with no restrictions for segment slopes, can be loaded into the database.

The first and second point ensure that data can be exported and re-imported into the database without loss of information. The third point ensures that data in more usual formats (where no restrictions for segment slopes exist) can be loaded into the database. Such properties are interesting to ensure that user applications can effectively interact with the spatial database, retrieve and modify spatial objects and store them back in the database, as well as for being able to import currently existing data. Whereas the first and second points are already fulfilled by any *dual grid*, we need to study some further properties of *dual grids* for selecting the subset that most suitably fulfils the third one.

Proposition 3.3. Let $G_p(n,1)$ be a grid of points with integer coordinates (that is, the only valid value for the denominator is 1). Then, $\forall S = (P_a, P_b),\ P_a, P_b \in G_p(n,1) : S \in G_s(3n+3,\ 2n+3)$.

Proof. To make the proof more readable, let $n' = 3n+3$ and $m' = 2n+3$. What we have to prove is that $\forall S = (P_a, P_b),\ P_a, P_b \in G_p(n,1) : S \in G_s(n',\ m')$.

It is clear that $G_p(n,1) \subset G_p(n',\ m')$ and hence $P_a,\ P_b \in G_p(n',\ m')$. Therefore, the only thing that needs to be proved is that $\forall P_a,\ P_b \in G_p(n,1)$, the supporting line $r:\ A{\cdot}x + B{\cdot}y = C$ of segment $S = (P_a,\ P_b)$ has $|A|$ and $|B|$ smaller than $\sqrt{2^{m'-1}} = \sqrt{2^{(2n+3)-1}} = 2^{n+1}$ and $|C| < \frac{2^{n'-1}}{\sqrt{2^{m'-1}}} = \frac{2^{(3n+3)-1}}{\sqrt{2^{(2n+3)-1}}} = 2^{2n+1}$.

Given $P_a = (X_a,\ Y_a)$ and $P_b = (X_b,\ Y_b)$, the coefficients of the supporting line are $A = Y_a - Y_b$, $B = X_b - X_a$ and $C = X_bY_a - X_aY_b$. It is easy to see that:

$$|A| = |Y_a - Y_b| < 2^n + 2^n = 2^{n+1}$$

$$|B| = |X_b - X_a| < 2^n + 2^n = 2^{n+1}$$

$$|C| = |X_bY_a - X_aY_b| < 2^n \cdot 2^n + 2^n \cdot 2^n = 2^{2n+1} \leq 2^{2n+1}$$

□

Proposition 3.4. Let $G_p(n,m)$ be a grid of points. Let $P_a = (\frac{num_{ax}}{den_a},\ \frac{num_{ay}}{den_a})$ and $P_b = (\frac{num_{bx}}{den_b},\ \frac{num_{by}}{den_b})$ be two points with homogeneous[4] coordinates such that $P_a,\ P_b \in G_p(n,m)$. Let $S = (P_a,\ P_b)$. Then, $S \in G_s(3n+m+3,\ 2n+2m+3)$.[5]

Proof. To make the proof more readable let $n' = 3n+m+3$ and $m' = 2n+2m+3$, that is, we have to prove that $S \in G_s(n',\ m')$.

In the previous proof for points with integer coordinates we have already seen that the supporting line of S is $r:\ A{\cdot}x + B{\cdot}y = C$, with $A = Y_a - Y_b$, $B = X_b - X_a$ and $C = X_bY_a - X_aY_b$. In this case,

$$A = \frac{num_{ay}}{den_a} - \frac{num_{by}}{den_b} = \frac{den_b \cdot num_{ay} - den_a \cdot num_{by}}{den_a \cdot den_b}$$

[4] Two coordinates are called *homogeneous* if they have the same denominator.

[5] Note that we give an upper bound for the values of n and m in the required segment grid $G_s(n,m)$, and do not define the minimum for them. In specific cases (as in the one with integer coordinates shown above) a segment grid with slightly smaller sizes for numerator and denominator could be used.

$$B = \frac{num_{bx}}{den_b} - \frac{num_{ax}}{den_a} = \frac{den_a \cdot num_{bx} - den_b \cdot num_{ax}}{den_a \cdot den_b}$$

$$C = \frac{num_{bx}}{den_b} \cdot \frac{num_{ay}}{den_a} - \frac{num_{ax}}{den_a} \cdot \frac{num_{by}}{den_b} = \frac{num_{bx} \cdot num_{ay} - num_{ax} \cdot num_{by}}{den_a \cdot den_b}$$

Such a line is equivalent to the line $r' : \; A' \cdot x + B' \cdot y = C'$, where

$$A' = A \cdot den_a \cdot den_b = den_b \cdot num_{ay} - den_a \cdot num_{by}$$

$$B' = B \cdot den_a \cdot den_b = den_a \cdot num_{bx} - den_b \cdot num_{ax}$$

$$C' = C \cdot den_a \cdot den_b = num_{bx} \cdot num_{ay} - num_{ax} \cdot num_{by}$$

Hence,

$$|A'| = |den_b \cdot num_{ay} - den_a \cdot num_{by}| < 2^{n+m+1}$$

$$|B'| = |den_a \cdot num_{bx} - den_b \cdot num_{ax}| < 2^{n+m+1}$$

$$|C'| = |num_{bx} \cdot num_{ay} - num_{ax} \cdot num_{by}| < 2^{2n+1}$$

It is easy to see that:

$$\sqrt{2^{m'-1}} = \sqrt{2^{(2n+2m+3)-1}} = 2^{n+m+1} > |A'|$$

$$\sqrt{2^{m'-1}} = \sqrt{2^{(2n+2m+3)-1}} = 2^{n+m+1} > |B'|$$

$$\frac{2^{n'-1}}{\sqrt{2^{m'-1}}} = \frac{2^{(3n+m+3)-1}}{\sqrt{2^{(2n+2m+3)-1}}} = \frac{2^{3n+m+2}}{2^{n+m+1}} = 2^{2n+1} > |C'|$$

Therefore, $S \in G_s(3n+m+3,\, 2n+2m+3)$. □

As a result of the previous analysis, we can reach the following conclusions. If we intend to import external data, then we have two basic possibilities. One is to use an optimized *dual grid* to be able to load data where the point coordinates are integer numbers of n bits (this would be the resolution A mentioned above). In this case, (as we have seen in the previous analysis) the best option is to use a *dual grid* $DG(3n+3,\, 2n+3)$. The other option is to optimize the *dual grid* to accept external data whose points are represented with homogeneous coordinates (the same denominator for both coordinates), where n bits are used for the numerator and m for the denominator (this would be another example of a resolution A). In this case, the optimal *dual grid* would be $DG(3n+m+3,\, 2n+2m+3)$. Such a *dual grid* would also accept data with integer coordinates of $n + \frac{m}{3}$ bits. Note that one special case of data provided in such an homogeneous format is the case when point coordinates are represented as fixed point decimal numbers. If i bits are used for the integer part and m bits are used for the decimal part, then we can represent those points with homogeneous coordinates with $i+m$ bits for the numerator and m bits for the denominator.

4 Adapting the ROSE Algebra

In the original implementation proposed for the ROSE algebra [GdRS95, GS95] all the spatial data types are defined over a *realm*, in which all the intersection points between the boundaries of the spatial objects are made explicit at insertion time, and their values are approximated, if needed, to the closest representable point. The points of the *realm* are points with unsigned integer coordinates of a given size n (in bits).

It is important to note that the algorithms implementing operations of the ROSE algebra rely on the fact that all intersection points are explicit. They are generally more simple and more efficient because of this. On the other hand, these algorithms will crash or yield wrong results if they receive arguments with intersecting segments.

The basic idea for implementing a *realm-less* version of the ROSE algebra (which can also be viewed as using an *implicit realm)* is to represent data over a dual grid such that all intersection points are representable exactly. In addition, we need to make sure that the algorithms receive only arguments that do not have proper segment intersections. In the sequel we first explain how a realm can be represented over a dual grid, and in the next step, how the (explicit) realm can be omitted altogether.

For representing a *realm* over a *dual grid* $DG(n, m)$, the domain of valid points of the *realm* must be $G_p(n, m)$, and the set of valid segments must be $G_s(n, m)$. Formally, a *realm* over a *dual grid* $DG(n, m)$ is a subset $R = P \cup S$ of $DG(n, m)$ such that:

1. $P \subseteq G_p(n, m)$, $S \subseteq G_s(n, m)$
2. $\forall p \in P \, \forall s \in S : \neg(p \,\mathrm{in}\, s)$
3. $\forall s, t \in S$, $s \neq t : \neg(s \text{ intersects } t) \wedge \neg(s \text{ overlaps } t)$

The predicates *in*, *intersects*, and *overlaps* are defined precisely in [GS93]. Informally, they mean that a point does not lie on a segment except possibly on an end point, that two segments do not intersect except in their end points, and that two collinear segments can only touch in an end point.

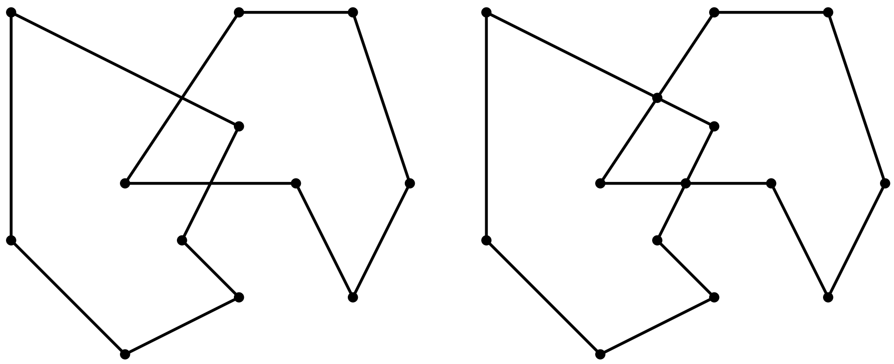
Now operations provided for importing data into the database[6] should check that the input data conform to the *dual grid* requirements. If the data to be imported is represented with a resolution of type A (e.g. integer coordinates of size i such that $DG(3i+3, 2i+3) \subseteq DG(n, m)$), then no check needs to be performed. This will be the normal case when importing data from external sources.

Otherwise, the import algorithm must check that the type of the input coordinates is compatible with the one used in the *dual grid* (the resolution B for end points) and the supporting lines of all segments of the input values fulfill the restrictions given for segments of a *dual grid* (the resolution B' for segment slopes). This case will normally only occur when data have been exported previously from a dual grid database of the same resolution.

As a result, it is guaranteed, that intersection points can be represented as realm points. One can now proceed as before inserting data into a realm (splitting segments at intersection points), but with the following advantages:

- Relationships between points and segments are not altered by the redrawing process.

[6] Although they are not part of the ROSE algebra, they should be provided when implementing an spatial extension for a database.



(a) Non realm-based arguments. (b) Realm-based arguments.

Figure 3: Construction of a realm with the arguments of a ROSE operation

- Insertion of new data does not modify other data in the database. The new inserted data are also inserted without modifying their values.
- Consistency of answers over time is also guaranteed.

Yet, two problems of the *realm*-based approach remain in this case:

- Complexity of updates. Although the cascade of rewritings in the *realm* is reduced when using *dual grid* (because now when a segment in the *realm* is decomposed, the resulting segments still contain the same set of points, and hence the new segments will not intersect extra points or segments of the *realm*), the rewriting process can still produce a considerable overhead when the new data are located in areas of the plane with a high density of objects.
- Space overhead. Although some decompositions of segments are avoided because no modifications of the values represented in the *realm* are produced, one should still expect a considerable overhead for data located in high density areas of the plane.

However, there exists a better solution: Given that now the use of a *realm* does not modify the data stored in the database, the ROSE operations will return the same value whether the *realm* is constructed with all the objects of the database or it is constructed only with the arguments of the operation. Hence, instead of storing in the database the *realm*-based representation of the data, we can simply store the data as they are inserted into the database. Whenever a pair of spatial objects is used as argument of an operation, a preprocessing step (*realmizator*) is added that computes a *realm*-based version of the two arguments, where only the intersection points between them are made explicit (any other objects in the database are ignored). This is illustrated in Figure 3. This *realm*-based version is then passed to the ROSE algebra algorithm, which computes the result. Such a preprocessing algorithm can, for example, be implemented as a variant of the well-known plane-sweep algorithm by Bentley and Ottmann for finding intersections of line segments [BO79] (see also [PS85]) which requires $O(n \log n + k)$ time where n is the total number of line segments and k the number of intersections. Note

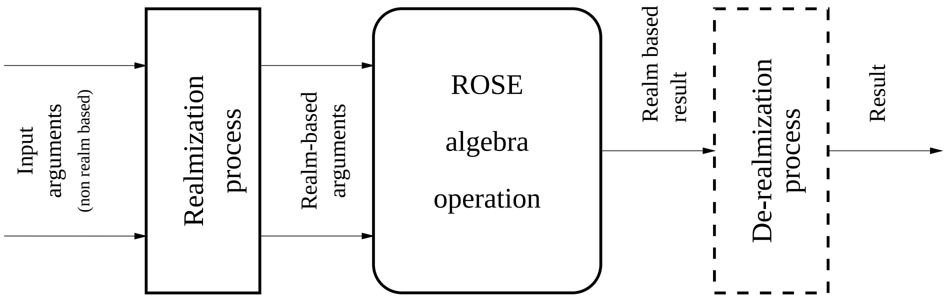


Figure 4: Replacement of *realms* by a preprocessing step before applying an operation

that one can implement this algorithm once and for all and then apply all ROSE operations unchanged, instead of modifying all the ROSE algorithms to detect intersections.[7]

Optionally, a post-processing step (*derealmizator*) can be added to normalize the result (e.g. to melt into one those contiguous segments of the returned object that are collinear). Both pre- and post-processing steps are illustrated in Figure 4. Currently, an open question is whether using the “realmization” preprocessing step causes a significant overhead with respect to the time requirements for executing a ROSE operation, or whether that overhead can be neglected. In many cases, the ROSE operations are implemented by means of plane sweeps themselves and have a similar time complexity as the preprocessing algorithm mentioned above [GdRS95].

Clearly, there is a trade-off involved here: Storing all objects in a realm-based fashion with explicit intersections increases space requirements. It also increases processing time for operations due to the larger representation of objects. On the other hand, the preprocessing algorithm is not needed which saves some time. The other (realm-less) option needs less space and manipulates smaller object representations, but needs the realmization algorithm. We plan in the near future an experimental evaluation to investigate the effects of this trade-off.

5 Conclusions

In this paper we have presented the *dual grid*, a new approach in the representation of spatial objects that solves the robustness problems in the computation of spatial operations and ensures the consistency between answers. This has been done by selecting an appropriate discrete representation for the spatial values that is completely closed under the set of target operations, in this case the operations of the ROSE algebra. We have also shown how the use of a *dual grid* as the domain set of *realms* not only solves the open problems of the original implementation of the ROSE algebra, but also simplifies the insertion of elements into the database and allows a very efficient representation of spatial values.

The dual grid approach provides closure and consistency between answers for an important set of operations, including the set operations. Although some other interesting operations are left[8] (e.g. the *convex-hull* operation or the computation of *Voronoi* diagrams), we expect the consistency between answers for such operations not to be so important as for the set

[7] Of course, adapting all the ROSE algorithms separately is also possible, and might lead to a slightly more efficient execution, but involves a quite large amount of work, complicates algorithms, and may introduce new errors into an already well-tested software package.

[8] Such operations are not part of the ROSE algebra because they would not be closed over the realm basis.

operations. For example, an implementation of the *convex-hull* operation that returns an approximate result but ensures that all the generator points are contained in it is not likely to produce consistency problems between answers, because the more important properties of the result as well as its more important relationships with the arguments are fulfilled.

Further work in this area is:

- Implementation of the ROSE algebra over a *dual grid*, using the preprocessing algorithm instead of *realms*. Such an implementation, as an extension package for *Secondo* and *Informix Illustra*, is already in development and almost finished.
- Comparison of the efficiency of the implementation of the ROSE algebra over a *dual grid* using a stored realm-based representation (à la *virtual realms*) and the same implementation without realm, using the preprocessing algorithm instead.
- Studying extensions of the ROSE algebra by operations creating new geometries (which were not possible in the realm-based approach) in such a way that closure over the dual grid is maintained. For example, a convex hull operator applied to a *points* value might return the smallest convex polygon enclosing the points with boundary segments contained in the dual grid.

Finally, let us remark that although the proposal of this paper is relatively simple and easily implementable, it solves an important problem in offering the only spatial database algebra with guaranteed robustness and correctness properties besides the (original, realm-based) ROSE algebra. Compared to the original ROSE algebra, the relatively complex realm machinery (especially the redrawing) and its interaction with a DBMS need not be implemented any more; yet, the correctness properties of the dual grid approach are even better than those of the realm-based ROSE algebra.

References

[BO79] J.L. Bentley and T. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers*, C-28:643–647, 1979.

[Dav98] J.R. Davis. IBM's DB2 Spatial Extender: Managing Geo-Spatial Information Within The DBMS. Technical report, IBM Corporation, May 1998.

[DS90] D. Dobkin and D. Silver. Applied Computational Geometry: Towards Robust Solutions of Basic Problems. *Journal of Computer and System Sciences*, 40:70–87, 1990.

[Ege94] M. Egenhofer. Spatial SQL: A Query and Presentation Language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, 1994.

[FDP+99] A.A.A. Fernandes, A. Dinn, N.W. Paton, M.H. Williams, and O. Liew. Extending a Deductive Object-Oriented Database System with Spatial Data Handling Facilities. *Information and Software Technology*, 41(1):483–497, 1999.

[For85] A.R. Forrest. Computational Geometry in Practice. In *Fundamental Algorithms for Computer Graphics*, pages 707–723. Springer-Verlag, 1985.

[GdRS95] R.H. Güting, T. de Ridder, and M. Schneider. Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types. In *Proc. of the 4th Intl. Symposium on Large Spatial Databases*, pages 216–239, Portland, Maine, August 1995.

[GM95] L.J. Guibas and D.H. Marimont. Rounding Arrangements Dynamically. In *11th Annual Symposium on Computational Geometry*, pages 190–199, 1995.

[GS93] R.H. Güting and M. Schneider. Realms: A Foundation for Spatial Data Types in Database Systems. In *Proc. of the 3rd. Intl. Symposium on Large Spatial Databases*, pages 14–35, Singapore, June 1993.

[GS95] R.H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4(2):100–143, 1995.

[Güt88] R.H. Güting. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In *Proc. of the Intl. Conf. on Extending Database Technology*, pages 506–527, Venice, Italy, 1988.

[GY86] D. Greene and F. Yao. Finite-Resolution Computational Geometry. In *Proc. 27th IEEE Symposium on Foundations of Computer Science*, pages 143–152, 1986.

[Ill94] *Illustra 2D Spatial Datablade (Release 1.3) Guide*, October 194.

[KM83] P. Kornerup and D.W. Matula. Finite Precision Rational Arithmetic: An Arithmetic Unit. *IEEE Transactions on Computers*, C-28:378–388, 1983.

[Mil89] V. Milenkovic. Double Precision Geometry: A General Technique for Calculating Line and Segment Intersections Using Rounded Arithmetic. In *30th Annual Symposium on Foundations of Computer Science*, pages 500–505, 1989.

[MPF+96] V. Muller, N.W. Paton, A.A.A. Fernandes, A. Dinn, and M.H. Williams. Virtual Realms: An Efficient Implementation Strategy for Finite Resolution Spatial Data Types. In *Proc. of the 7th Intl. Symposium on Spatial Data Handling - SDH'96*, volume 2, pages 11B.1–11B.13, Delft, The Netherlands, 1996.

[Ora97] *Oracle8i Spatial Cartridge*, June 1997.

[Ora99] *Oracle8i Spatial: Features Overview*, February 1999.

[OTU87] T. Ottmann, G. Thiemt, and C. Ullrich. Numerical Stability of Geometric Algorithms. In *3rd ACM Symposium on Computational Geometry*, pages 119–125, 1987.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry.* Springer-Verlag, Berlin, Heidelberg, New York, 1985.

[Sch94] P. Schorn. Degeneracy in Geometric Computation and the Perturbation Approach. *The Computer Journal*, 37(1), 1994.

[SH91] P. Svensson and Z. Huang. Geo-Sal: A Query Language for Spatial Data Analysis. In *Proceedings of the 2nd Intl. Symposium on Large Spatial Databases*, pages 119–140, Zürich, Switzerland, 1991.