# Virtual Realms: An Efficient Implementation Strategy for Finite Resolution Spatial Data Types

Volker Muller*, Norman W. Paton†, Alvaro A.A. Fernandes††,
Andrew Dinn and M. Howard Williams

Department of Computing and Electrical Engineering,
Heriot-Watt University, Riccarton,
Edinburgh, UK.
email: [andrew,howard]@cee.hw.ac.uk

Department of Computer Science†,
University of Manchester, Oxford Road,
Manchester, UK.
e-mail: norm@cs.man.ac.uk

Department of Mathematical Sciences††,
Goldsmiths College, New Cross,
London SE14 6NW, UK.
email: a.fernandes@gold.ac.uk

**Abstract**

A realm is a planar graph over a finite resolution grid that has been proposed as a means of overcoming problems of numerical robustness and topological correctness in spatial database systems. While the realm structure and the spatial algebra that is associated with it provide a range of desirable facilities for modelling spatial information in database systems, widespread exploitation will only be practical if efficient implementation strategies are identified. This paper shows how data types can be supported efficiently over *virtual realms*, where the finite resolution grid is not stored explicitly, but is generated only partially and as needed. This approach avoids the considerable storage space overheads associated with the original proposal for the implementation of realms, and provides overall runtime performance that often improves upon that of less space efficient implementation strategies.

**Keywords:** Spatial databases, spatial data types, finite resolution, object management.

## 1 Introduction

The notion of *realm* is presented in [6, 7], along with an associated collection of spatial data types and operations, known as the ROSE algebra. The ROSE algebra supports a wide range of operations over data types representing *points*, *lines* and *regions*, and has been designed in such a way that all operations are closed (i.e. yield results that can be operated on by the algebra), and can be efficiently implemented, as described in [5]. However, the overall efficiency of a spatial database based upon the ROSE algebra depends not only upon the efficiency of the operations of the algebra, but also on the way in which the underlying realm has been constructed.

*This work was done when the first author was visiting, Heriot-Watt University from the University of Karlsruhe, supported by an Erasmus scheme coordinated by the EU HC&M Network ACT-NET.

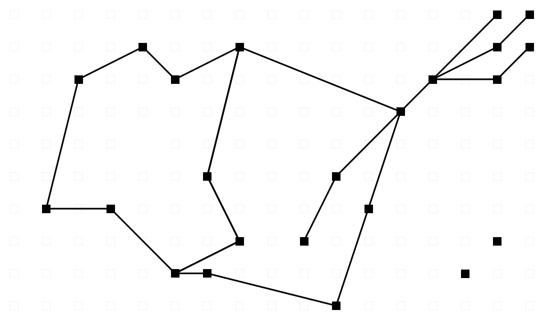

Figure 1: Example of a realm

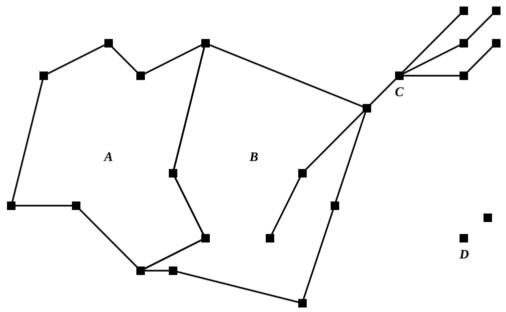


Figure 2: Spatial values which can be built from the realm

A comprehensive description is given in [5] of how the ROSE algebra operations can be implemented, and data structures and algorithms for implementing realms are described in [6]. In that proposal, the realm is stored explicitly, and data in the structures that represent the realm is replicated in data structures that support the operations of the ROSE algebra. The approach presented in this paper can be used directly with the implementations of the ROSE algebra operations in [5]; the focus is thus on improving upon the implementation of the underlying realm presented in [6]. The principal characteristic of the approach presented in this paper is that, unlike that in [6], the realm is not explicitly stored (i.e. it is *virtual* – fragments of realm are only generated temporarily, and as needed by operations that update the database). As the implementation in [6] is expensive in terms of space used, considerable overall savings in storage space for ROSE algebra objects are obtained. The question that suggests itself, given the reduced space overhead of virtual realms, is what impact this has on the runtime performance of systems built on top of realms. It is shown in this paper that, although the two approaches have different performance profiles, the virtual realm supports runtime performance that is at least comparable with that offered by stored realms.

This paper is structured as follows. Section 2 describes realms and their relationship to spatial data types in more detail; section 3 summarises the implementation used by [6] and presents the virtual realm approach; section 4 compares the performance of the two approaches; and section 5 presents some conclusions.

## 2 Context: Realms and the ROSE Algebra

A realm, an example of which is depicted in figure 1, is a set of points and non-intersecting line segments over a finite discrete grid. All the spatial data values of an application have realm points and line segments as their components, so the realm contains the complete underlying geometry of an application. Figure 2 shows two *regions* objects $(A, B)$, a *lines* object $(C)$ and a points object $(D)$ which can be constructed from the points and line segments in the realm depicted in figure 1.

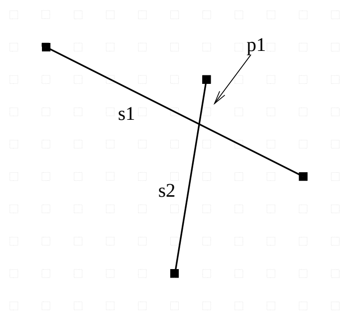


Figure 3: Two segments in a realm before changes are computed

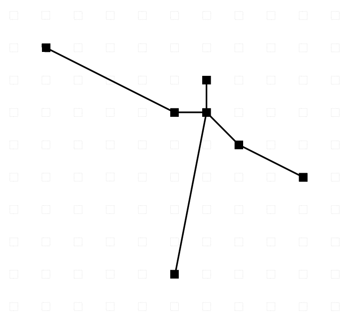

Figure 4: Two segments in a realm after changes are computed

To represent a realm in the computer, integer representations and error-free integer arithmetic are used. This is made practical by exploiting algorithms that use finite-resolution computational geometry to characterise the inevitable distortion of the geometry when contiguous space is mapped onto discrete space [4]. The essence of this approach is that the ambiguities associated with inexact representation of spatial concepts are resolved when information is inserted into the database, which allows all future processing to assume that information is represented precisely with respect to the finite resolution grid, thereby simplifying processing and avoiding error propagation.

The insertion-time resolution of ambiguities involves the following steps:

- All points from the application are mapped to points in the realm; if two points are close together, it is resolved at insertion time whether or not they are to be represented by the same realm point.
- All line segments from the application are mapped onto a sequence of one or more intersection-free line segments in the realm. In the process, lines may be split and new realm points introduced to represent intersections involving the inserted line segments. Thus all intersections between points and line segments and between groups of line segments are identified and resolved at insertion time. The size of the error introduced when resolving intersections is bounded by the *envelope* of the line segment, which is, informally, the set of grid points adjacent to the line segment. As an example, figure 3 shows two segments, *s1* and *s2*, one of which has just been inserted into the realm. As the realm records intersection-free line segments at finite resolution, it is necessary to identify a realm point to act as the intersection of the two segments, and to redraw the component segments so that they pass through this point. An example of such a redrawing is given in figure 4, where the segments *s1* and *s2* from figure 3 are redrawn so that they intersect at realm point *p1*. A formal description of the redrawing process is given by [4].

A realm forms part of a layered architecture which supports the data types and operations of the ROSE algebra. The ROSE algebra supports the types *points*, *lines* and *regions*, which in turn

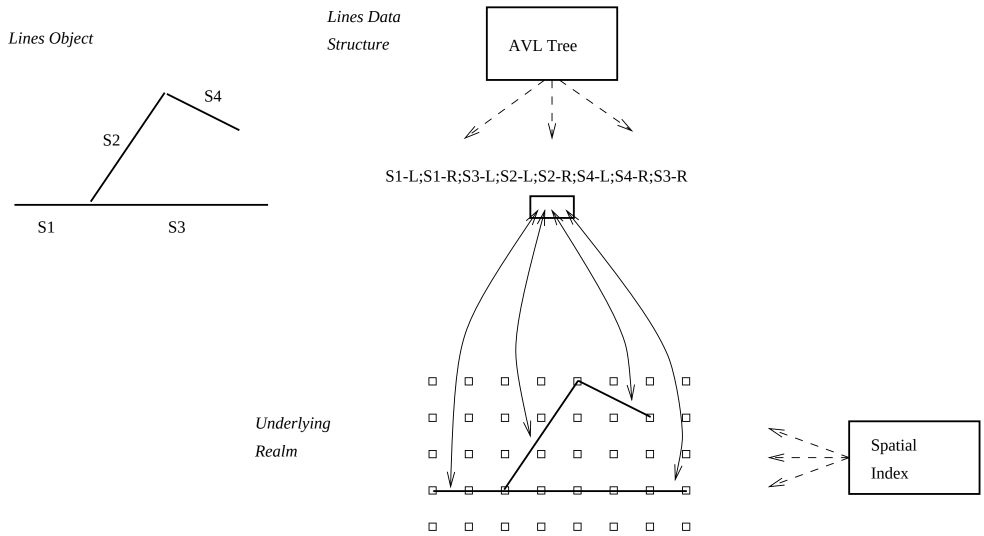


Figure 5: A *lines* data object and its stored representation.

are associated with a comprehensive, if conventional, collection of spatial operations defined in [7]. Algorithms for the implementation of the ROSE algebra operations are detailed in [5], along with the data structures used to represent the associated types.

## 2.1 Stored Realms

This subsection gives an overview of the implementation strategy proposed for the ROSE algebra and the underlying realm in [6, 5]

Figure 5 shows the relationship between the data types of the ROSE algebra and the underlying realm for a *lines* spatial object. The *lines* object consisting of the segments *S1*, *S2*, *S3* and *S4* [1] is represented by an ordered sequence of half segments, as this supports efficient implementation of spatial operations using plane sweep algorithms, etc [5]. The half segments are named in the figure with the suffix $L$ representing the leftmost end of the segment and $R$ representing the rightmost. This representation is scanned from left to right by algorithms to identify, for example, where two *lines* objects intersect. To support efficient updating of the ordered sequence of half segments, the halfsegments are themselves indexed using an AVL tree (a balanced tree searching structure) to support rapid search and update. The representation of the spatial data types described so far is independent of the use of a stored or a virtual realm.

In the representation used in [6], the realm is stored explicitly, and every realm object (*point* or *segment*) both references and is referenced by every spatial object (*points, lines* or *regions*) of which it is part. However, the geometry (i.e. coordinates describing the absolute locations of points and line segments) is stored redundantly in the realm and in the ordered collection of half segments, so that it is not necessary to access the realm during the execution of ROSE algebra operations. As a result of this redundant storage of the geometry, the realm only needs to be accessed when operations take place that change ROSE algebra objects. It is not made clear in [6] exactly what data structures are used to implement the realm, but information in the realm does have to be indexed to support efficient update of the realm.

[1] Note that in the original data, *S1* and *S3* could have been joined as a single segment; if so, the partitioning would have taken place when the intersection with *S2* was resolved on insertion into the realm

## 2.2 Virtual Realms

The implementation proposed here uses the same data structures and algorithms as [5] to implement ROSE algebra operations, but the realm is not stored explicitly. Instead, all that is held in the database is the ordered list of half-segments described above, and fragments of realm are constructed only temporarily, to identify and resolve intersections, when ROSE algebra objects are updated. Essentially, whenever a ROSE algebra object $R$ is updated, all objects that intersect the minimum bounding rectangle of $R$ are retrieved, and the section of realm relating to these objects is constructed temporarily to resolve interactions involving these objects and $R$. When such interactions have been resolved, the changes made to $R$ and nearby ROSE algebra objects are stored, and the recently constructed fragment of realm is discarded. The consequences for implementors of the choice of virtual or stored realms are considered more fully in section 3.

# 3 Implementations: Stored and Virtual Realms

As the role of the realm is to provide a context for the resolution of ambiguities when mapping continuous space onto discrete space, and the underlying realm is not used directly when implementing ROSE algebra operations, the principal operations that are supported on the realm involve the insertion of points and segments. This section outlines how operations for inserting segments are implemented in both stored and virtual realms, as insertion of a segment is similar to, if somewhat more complex than, insertion of a point. The relative performance of the approaches is considered in section 4.

## 3.1 Segment Insertion in the Stored Realm

The algorithm presented in figure 6 for inserting a segment into a realm is essentially that given in [6]; differences are in presentation rather than substance.

*InsertSegmentS* takes two input parameters, a segment $s$, and the realm $R$ into which $s$ is to be inserted. The outputs are an updated realm $R'$ and a set of ROSE algebra objects $RO$ that have to be updated to reflect the changes to the realm. This propagation of updates is required to ensure that the information stored in the spatial data types is consistent with that in the realm, using the structures in figure 5.

The algorithm performs the following steps:

**Step 1:** Initialise the set of ROSE objects that have to be revised to the empty set.

**Step 2:** Check to see if the segment $s$ is already in the realm; if it is, then the realm need not be changed. If the segment is not already in the realm, it has to be inserted; when doing this, it is necessary to identify which points and segments in the realm are in close proximity with the new segment.

Lines (16-20) identify the consequences of having a point in the envelope of the new segment. The first consequence is that the segment must be split into two, as it is considered to go through any point within its envelope. The fact that the segment is to be split is marked by inserting a hook [4] (a short directed line segment) from the new segment to the grid point. The second consequence is that any further segments in the realm that intersect the hook must also be redrawn.

Lines (21-27) identify the consequences of having a segment in the realm intersect the new segment. Where this is the case, both the intersecting segments have to be redrawn through the realm point nearest to the intersection; this need for redrawing is again marked using hooks. Any further segments intersecting the hook are identified in lines (25-27) and are themselves marked for redrawing.

```
algorithm InsertSegmentS
inputs:     s: Segment
            R: Realm
outputs:    R': Realm
            RO: set of RoseObjects

Step 1: Initialisation
      RO := ∅

Step 2: Find nearby realm objects
      if s already in R then
            R' := R
            return
      else
            SR := ∅
            for each point p in the envelope of s do
                  create a hook h from s to p
                  for each segment v intersecting h do
                        create a hook from the intersection of h and v to p
                        SR := SR ∪ {v}
            for each segment t intersecting s do
                  create a hook h from the intersection
                  of s and t to the nearest grid point
                  SR := SR ∪ {t}
                  for each segment v intersecting h do
                        create a hook from the intersection of h and v to p
                        SR := SR ∪ {v}
      end-if

Step 3: Redraw hooked lines
      RS := ∅
      for each segment t in {s} ∪ SR do
            RS := RS ∪ segments created when redrawing
                  t using the procedure of [4]

Step 4: Update realm
      R' := R minus all segments in {s} ∪ SR
      R' := R' plus all segments in RS
      RO := spatial objects of redrawn segments
```

Figure 6: Segment insertion algorithm for the stored realm.

**Step 3:** All segments that need to be redrawn have been identified by Step 2. These segments are then redrawn using the algorithm of [4], which is not stated here to save space, and because it is applied similarly with both the stored and the virtual realm.

**Step 4:** The changes identified are applied to the realm, and the ROSE algebra objects that have to be updated are recorded in $RO$.

### 3.2 Segment Insertion in the Virtual Realm

The algorithm presented in figure 7 indicates the steps that are involved in constructing a portion of a virtual realm in response to the insertion of a segment. In this algorithm, as with the algorithm in figure 6, it is assumed that the end points of the segment have already been inserted.

*InsertSegmentV* takes one input parameter, the segment $s$ that is to be inserted. The output is the set of ROSE algebra objects $RO$ that have to be updated to reflect the changes to the realm.

The algorithm performs the following steps:

**Step 1:** Initialise the set of ROSE algebra objects $RO$ that are affected by the insertion, and the virtual realm $VR$.

**Step 2:** The spatial values that have components that are in close proximity to the inserted segment have to be identified. It is assumed that all *points*, *lines* and *regions* objects are stored using a spatial index (in the implementation, an R-tree is used), and that fast access is thus supported to all spatial objects that overlap a given minimum bounding rectangle (MBR).

Lines (12–23) identify the *lines* and *regions* values that have components (segments) that either need to be redrawn as a result of the insertion of $s$, or that force $s$ to be redrawn. If it transpires that any existing spatial object contains a segment identical to $s$ (line 14), then there is no need to proceed further with the construction of the virtual realm. The interactions that are important are: intersection of $s$ with existing segments (line 17) and the discovery of endpoints of existing segments in the envelope of $s$ (lines 19 and 21). Any such interactions are recorded in the virtual realm $VR$.

Lines (24–29) identify the *points* values that have components that are within the envelope of $s$, and store them in $VR$.

**Step 3:** The nature of the interactions between the objects in the virtual realm and $s$ have to be worked out, and such redrawing as is necessary performed. For each point in $VR$, this involves creating a hook to $s$ and identifying all segments in $VR$ that intersect the hook (lines 32–35). For each segment in $VR$, this involves creating a hook from the intersection with $s$ to the nearest grid point, and also identifying all segments in $VR$ that intersect the hook (lines 36–40).

Lines (42–45) then perform redrawing of the portion of the realm affected by the insertion of $s$ using the algorithm of [4], and record the ROSE algebra objects affected by the changes in $RO$.

**Step 4:** As the realm is only created partially and temporarily, $VR$ is then deleted.

## 4 Performance: Comparing Stored and Virtual Realms

This section presents an informal comparison of the performance of stored and virtual realms for the insertion of a segment, as this operation is both common and representative of realm based activity. Significant to a broader consideration of the performance of a spatial database based upon the ROSE algebra, however, is the fact that most ROSE algebra operations never access structures at the realm level, and thus will perform in an identical manner for both stored and virtual realms.

```
algorithm InsertSegmentV
inputs:     s: Segment
outputs:    RO: set of RoseObjects

Step 1: Initialisation
        RO := ∅
        VR := ∅

Step 2: Identify nearby objects for insertion into realm
        SR := ∅
        SLR := lines and regions objects with MBR that overlaps s
        for each spatial value sv in SLR do
                for each segment t in sv do
                        if t = s then
                                delete VR
                                return
                        elseif intersect(t,s) then
                                VR := insertRsegment(t,sv,VR)
                        elseif t.p1 in envelope of s then
                                VR := insertRpoint(t.p1,VR)
                        elseif t.p2 in envelope of s then
                                VR := insertRpoint(t.p2,VR)
                        end-if
        SP := points spatial values with MBR that overlaps s
        for each ps in SP do
                for each point pr in ps do
                        if pr in envelope of s then
                                VR := insertRpoint(pr,VR)
                        end-if

Step 3: Perform redrawing of objects in virtual realm
        for each point rp in VR do
                create a hook h from s to rp
                for each segment t in VR intersecting h do
                        create a hook h from the intersection of h and t to rp
        for each segment rs in VR do
                create a hook h from the intersection of
                s and rs to the nearest grid point rp
                for each segment t in VR intersecting h
                        create a hook h from the intersection of h and t to rp
        RS := ∅
        for each segment t in VR do
                RS := RS ∪ segments created when redrawing
                        t using the procedure of [4]
        RO := spatial objects of redrawn segments in RS

Step 4: Tidy up
        delete VR
```

Figure 7: Segment insertion algorithm for the virtual realm.

| Separate Realm | Virtual Realm | Difference |
|---|---|---|
| Retrieve required nodes of spatial index (many entries) | Retrieve required nodes of spatial index (fewer entries) | –I/O |
| Retrieve segments and points from MBR of inserted segment (possibly many) | Retrieve points, lines and regions spatial values from MBR of inserted segment (possibly many, possibly large) | +I/O |
| – | Scan all points, lines and regions spatial values, found above, to construct the virtual realm | +CPU |
| Compute changes in the realm | Compute changes in the virtual realm | – |
| Delete changed segments from disk | – | –I/O |
| Write new segments to disk | – | –I/O |
| Compute changes to index (possibly several deletions) (possibly many insertions) | Compute changes to index (max. 1 deletion) (max. 1 insertion) | –CPU |
| Write changed index to disk | Write changed index onto disk | –I/O |
| Retrieve the spatial values of which the changed segments are elements (possibly many) | – | –I/O |
| Replace changed segments in the spatial values of which they are elements | Replace changed segments in the spatial values of which they are elements | – |
| Write changed spatial values to disk | Write changed spatial values to disk | – |
| – | Delete virtual realm | +CPU |

Figure 8: Performance comparison of stored and virtual realm for insertion of a segment.

The precise storage saving that derives from exploiting virtual realms is not straightforward to state, depending as it does upon the data structures used to implement both approaches and the patterns of data stored. We note, however, that the overall space saving is likely to be significant, as space is saved not only in the realm itself, but also in linking the realm to the structures used to support the spatial data types. We estimate that the storage space occupied by spatial information described using the types of the ROSE algebra will be reduced by at least 30% when virtual realms are used in preference to stored realms.

In considering the relative performance of stored and virtual realms, it is assumed the realm resides on secondary storage, as will certainly be the case in any realistic applications. Both approaches require the use of a spatial index – in the case of the stored realm, this is assumed to be an index on the realm, in the case of the virtual realm, this is assumed to be an index on the spatial types built using the realm.

Figure 8 summarises the tasks performed when inserting a segment using the stored and the virtual realms. The *Difference* column indicates whether the virtual realm increases (+) or decreases (–) the cost of performing input-output (I/O) or processing (CPU) tasks.

The results presented in the table can be summarised with respect to the effect on I/O costs and CPU time:

**I/O:** Considerably more I/O activity is required with the stored realm than with the virtual realm. This is to be expected, in that realm information must be read from and written to disk when it is stored, whereas this is not required when the realm is virtual. The only entry which shows the virtual realm imposing an increased I/O load is when spatial values in close proximity to the inserted segment are read from disk. This shortcoming of the virtual realm is partly compensated for later, as many of these spatial values must be read in when propagating updates from the realm to associated points, lines and regions values.

**CPU:** The virtual realm is more expensive than the stored realm in terms of CPU time. This is also to be expected, as additional processing is required to build and destroy the virtual realm, which is a main memory data structure.

Overall, it can be argued that the virtual realm is likely to provide faster processing than the stored realm, as the performance improvement resulting from the reduction in disk activity is likely to be more than enough to compensate for the increased CPU activity associated with the building of the virtual realm.

# 5 Conclusions

This work is being carried out in the context of a project which is seeking to extend the kernel of the deductive object-oriented database system ROCK & ROLL [2] with facilities to support efficient handling of spatial concepts [1, 8]. Such research requires efficient support for a coherent collection of spatial concepts that can be implemented efficiently. The ROSE algebra seemed to us to provide a range of facilities that made it highly suitable for incorporation into the kernel of a database system. However, the space overhead associated with explicit storage of the realms was a source of considerable concern. The notion of a virtual realm was introduced to overcome the space problems associated with stored realms, while retaining the advantages of the realm in terms of numerical robustness and topological correctness. As demonstrated in section 4, this space saving has not been acquired at the cost of slower runtime performance, as reduced overall I/O activity means that systems based on the virtual realm will often perform more quickly than those based upon stored realms.

At time of writing, the virtual realm has been implemented in C++ for use with the secondary storage management facilities of the EXODUS extensible database system [3]. Work is underway on the integration of the virtual realms and the associated ROSE algebra operations with the ROCK & ROLL deductive object-oriented database.

**Acknowledgements:** We are pleased to acknowledge the support of the EPSRC initiative on Architectures for Integrated Knowledge Manipulation Systems (grant GR/J99360) for funding this work, and of Neil Smith and Glen Hart of Ordnance Survey as the Industrial Partners.

**System Availability:** The ROCK & ROLL system is freely available over the Internet. It is anticipated that the first public release of the spatially extended ROCK & ROLL system will be in March 1997. For details, see WWW page *http://www.cee.hw.ac.uk/Databases/dood.html.*

# References

[1] A.I. Abdelmoty, N.W. Paton, M.H. Williams, A.A.A. Fernandes, M.L. Barja, and A. Dinn. Geographic Data Handling in a Deductive Object-Oriented Database. In D. Karagiannis, editor, *Proc. 5th Int. Conf. on Databases and Expert Systems Applications (DEXA)*, pages 445–454. Springer-Verlag, 1994.

[2] M.L. Barja, A.A.A. Fernandes, N.W. Paton, M.H. Williams, A. Dinn, and A.I. Abdelmoty. Design and Implementation of ROCK & ROLL: A Deductive Object-Oriented Database System. *Information Systems*, 20:185–211, 1995.

[3] M. Carey, D. DeWitt, G. Graefe, D. Haight, J. Richardson, D. Schuh, E. Shekita, and S. Vandenberg. The EXODUS Extensible DBMS Project: An Overview. In S. Zdonik and D. Maier, editors, *Readings in Object-Oriented Databases*, CA 94303-9953, 1990. Morgan Kaufman Publishers, Inc.

[4] D. Greene and F. Yao. Finite-Resolution Computational Geometry. In *IEEE Symp. on Foundations of Computer Science*, pages 143–152, 1986.

[5] R.H. Guting, T. de Ridder, and M. Schneider. Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types. In *Proc. 4th Int. Symposium on Large Spatial Databases (SSD)*. Springer-Verlag, 1995.

[6] R.H. Guting and M. Schneider. Realms: A Foundation for Spatial Data Types in Database Systems. In D. Abel and B.C. Ooi, editors, *Proc. 3rd Int. Conf. on Large Spatial Databases (SSD)*, Lecture Notes in Computer Science, pages 14–35. Spriger Verlag, 1993.

[7] R.H. Guting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB J.*, 4(2):243–286, 1995.

[8] N.W. Paton, A.I. Abdelmoty, and M.H. Williams. Programming Spatial Databases: A Deductive Object-Oriented Approach. In D. Parker, editor, *Innovations 3 (Proc. GIS Research UK (GISRUK))*. Taylor and Francis, 1996.