SPATIAL INDEXING IN ICAD SYSTEMS
Feliz RIBEIRO GOUVEIA, Jean-Paul A. BARTHES

CNRS URA 817 HEUDIASYC
Université de Technologie de Compiègne
BP 649  60206 Compiègne
FRANCE
email: barthes@frutc51.bitnet

This paper presents an overview of spatial indexing mechanisms and their implementation in a more general framework, as the one provided by Computer-aided Design systems. It is expected that such mechanisms play an important role in the context of so-called ICAD systems, providing a more rational and efficient management of geometric data. Benefitting from the object-oriented programming concepts of encapsulation, inheritance, and data abstraction, it will be possible to add spatial reasoning capabilities (restricted in our case to indexing problems) to already existing environments with minor changes from an organizational point of view.

1. Introduction

It has been widely recognized that the object paradigm is an important one for developing Intelligent Computer Aided Design (ICAD) systems, although some object-oriented languages commercially available do exhibit severe limitations [Tomiyama 89]. We also indicated previously that several models exist for representing objects [Barthès 89]. For large industrial applications such models must be complemented with persistency and sharing, leading to the notion of object environment including in particular object-oriented databases [Barthès et al 89]. Within object environments there is definitely a need for accessing the information efficiently. Today object systems allow at most two kind of accesses: a navigational access, allowing to move from object to object or to sub-parts following semantic links (real or virtual); a query style access, using indexes, allowing to access objects symbolically. However, in industrial applications involving several hundreds of thousands of objects (like designing a plane), the location of objects in space can be of importance and must be handled efficiently. Although such spatial indexing was done efficiently in the early drafting systems, it is not usually found in today object systems. This paper reviews
some of the indexing techniques that should be added to the next generation of ICAD systems, when spatial indexing is a requirement.

A spatial layout, be it a technical drawing or a power plant design, contains information which is hardly represented by existing data-oriented database systems. Besides the geometric information concerning the objects themselves, two other kinds of geometric information are implicitly present: topological information, e.g. inclusion, intersection, and metric information, e.g. proximity, parallelism. Although transparent to a user, his interaction with a CAD system involves often manipulation of the above mentioned information, as in the case of interference detection and computation when he moves (parts of) drawings from one location to another, or in the case of containment queries resulting from placing windows over regions of interest in the drawing. This function (i.e., geometric data processing) is delegated to a component of the CAD system which will accept input directly from the user but most frequently as a result of the user's actions. Hereafter will be referring to the collection of information such component should be able to manage as the geometric, or spatial, database.

Main concern in spatial databases is given to retrieval operations, where the time spent on searching data items must be minimized. Attributes being inherently interval valued, e.g. the sides of a square span several values over the coordinate axis, traditional database techniques become useless, for they are devised to work only with point data. Other characteristics of CAD systems which are of interest to us include:

• **locality of reference** - geometric data requested during consecutive operations is likely to be confined to a small neighborhood in space.

• **volatility of the data** - modifications to a drawing entail suppression and/or insertion of geometric data. This can happen quite often, at least in an early stage of the conception process.

• **uniformity of distribution of data in space** - an uniform distribution can be assumed, as long as the entire drawing can be subdivided in regions with uniform data distribution.

• **resolution, and different levels of space granularity** - sets the smallest unit of space that can be accessed, and which can contain a variable number of data items. Capability of viewing the data at different levels of resolution (zooming in and out) is also important.

All these considerations affect the design of spatial indexing mechanisms, as they constrain the choice of the underlying representational structures. In the context of large
collections of objects the representations must be suited to work with peripheral storage media, either directly or indirectly as we shall see. This is a main objective in our work.

2. Spatial databases

Manipulation of data in more than one dimension has been studied extensively, from an algorithmic point of view, in the computational geometry field [Dobkin 87, Preparata and Shamos 85] covering topics such as range searching, planar subdivision searching, and intersection problems. Computer science, on the other hand, views multidimensional data handling as an active area of storage organization design research, aiming at efficient access times of retrieval operations for most frequent queries (see for example [Vaydia 89]). Two problems arise, however, when dealing with geometric data which prevents it from taking full advantage of the above mentioned approaches: algorithmic solutions, such as the plane sweep paradigm, are not practicable when in the presence of large amounts of data, and besides no natural ordering exists for multidimensional data; the other problem stems from the fact that geometric data has implicit in it a notion of extent, as it is associated with physical world entities, which is difficult to represent in bitmap-like databases [Nievergelt and Hinterberger 84].

In what follows we will restrict our presentation to the two dimensional case, generalizations of the algorithms and the techniques to k dimensions (k > 2) being straightforward, with only minor changes in the time complexity and in storage space requirements. Geometric database organization is first confronted with the problems of representing the space occupied by the objects and then the entire space, that is, of how to represent individual objects and how to represent the entire collection of objects. We discuss the various approaches in the following sections.

2.1 Geometric data representation

Geometric data has been represented, for spatial indexing purposes (see [Samet 88] for a discussion on the subject), by point-based methods, such as characteristic points, and area-based methods, such as polygonal approximations and array structures. Such representations are derived from the CAD representation of the objects and they contain only the minimal description of the object necessary to locate it in space. Using directly the CAD representations of the entities being manipulated is possible in some applications, such as VLSI design where we deal with rectilinear polygons; an example
is given in [Nandy and Patnaik 87]. In engineering computer-aided design objects are often more complex and are represented by more powerful techniques (e.g. volumetric or surfacic models), which makes that spatial indexing uses yet another simplified representation of the objects, the one we will be talking about.

The characteristic point approach maps the area occupied by an object to a point in an usually higher dimensional space where it can be dealt with multi-attribute point data techniques. There are several choices for a characteristic point: a centroid, one corner, a point with coordinates the length of the sides of a bounding box. Polygonal approximation represents the object either by its contour or by its bounding primitive, which can be a rectangular polygon, or a circle, for example. Array structures, on the other hand, are mainly used because of the ease with which each of its locations can be indexed and traversed in a random or sequential manner, and also because of its easy construction process. We will not discuss them here because, solely as indexing structures, they have little practical interest as individual representations of objects.

Though it seems attractive, a major disadvantage of the characteristic point approach is that topological information is lost in the mapping process. Two objects that are close together may have their characteristic points well far apart in the new coordinate system, and reciprocally. Also, operations such as intersection are less obvious and cannot be performed directly over the data. On the other hand, it allows for simpler storage structures than their region-based counterparts as only the coordinates of a point need to be stored. Figure 1 shows a simple mapping from a one dimensional space to a two dimensional space. It is readily seen from the figure that overlapping segments can map to distant points in the two dimensional space. Samet [Samet 88] illustrates more examples of the characteristic point representation, and shows how some queries can be answered using this approach.

![Figure 1: Representing a segment by a characteristic point.](image)

Region-based representations, being easy to work with (in the sense that set-theoretic operations are easily performed), present however some disadvantages when we get to organize the entire database; while $k$-dimensional point data can be ordered over $k$...
attributes (for example the domination relation defines a partial ordering), \( k \)-dimensional line or region data has not an implicit order on it.

2.2 Geometric data organization

Organization of geometric data, represented by point-based or area-based methods, can be done with the aid of tree-like and cell-based techniques which fall in two categories: comparative search techniques and address computation techniques, respectively. The former organize the data by comparing each attribute in turn and deciding, at each step, the branch of the tree to follow. The latter use a correspondence function between the position of the data in space and the cells resulting from the space partition. As we shall see, both establish correspondences between regions in space and the data within those regions. When data is represented by the area they occupy it will be difficult to have it belonging to a single region, specially in the case of elongated objects. One possible solution consists of keeping track of all the regions the object intersects; others divide the object by the regions it belongs to. We have made experiments with redundant organizations, which we will discuss latter.

2.2.1 Tree organizations

Comparative search techniques, such as binary trees [Bentley 75], quadtrees and their multidimensional generalizations, e.g. the k-d tree and the oct-tree [Jackins and Tanimoto 83], are based on the recursive decomposition of space to yield a hierarchy of variable resolution regions. Such structures organize the data to be stored by partitioning the attribute space at the places where an uniform data distribution among the resulting regions is obtained, and this principle is applied recursively to the new regions until a stop condition (based on occupancy of regions or maximum depth in the tree) is met. They rely somehow on the divide-and-conquer principle of decomposing a \( k \)-dimensional problem into \( k \) one-dimensional problems.

Arborescent methods are efficient for updating operations such as insertion and deletion of data, for they simply alter boundaries between adjacent regions as to reflect the new data distribution. As hierarchical structures they allow for focalized search into geographical areas of interest. A comprehensive survey on hierarchical tree structures can be found in the literature [Samet 84]. They have been used extensively in various computing domains and many variants on recursive space decomposition have been proposed.
In spite of their variety, the tree structures differ basically by the symmetry of the space partitioning criteria. Quadtrees are symmetric structures because partitions are always placed at fixed points in space regardless of the data distribution, whereas non-symmetric structures, like binary trees and k-d trees, for example, can be easily adapted to different data distributions since partitions are drawn at arbitrary points in space.

![Symmetric structure](image1.png) ![Non-symmetric structure](image2.png)

Fig. 2: Symmetric vs non-symmetric space partitions

A major advantage of symmetric trees comes from the possibility of replacing nodes in the tree by numerical keys, eliminating pointers (that is, the tree structure) and consequently saving storage space. Coding schemes known as linear key have been proposed in [Abel and Smith 83]; other encoding of quadtrees is used in [Samet et al 84] to represent and store geographic data. Some interesting properties of the linear keys are discussed by Li [Li and Loew 87].

When data is to be kept in peripheral storage media, as it is the case in large applications, this coding allows to implement efficient storage and paging techniques, namely B-trees as in [Abel 84], or hashing methods. On the other hand no known coding scheme exists for non-symmetric trees, and external storage indexing is made through a smaller enough (higher-resolution) version of the original tree to be kept always in core; Matsuyama implemented such an approach with k-d-trees in a geographic information system [Matsuyama et al. 84].

Symmetric trees have also some other interesting properties which make them attractive in geometric applications where data is dynamic. Merging of old and new information is facilitated by the regular space subdivisions as they allow for simple set-theoretic operations to be performed. Although square or rectangular divisions are most common, several space tesselation patterns can be selected to conform to different data geometry.
2.2.2 Grid organizations

Grid (or cellular) structures belong to the second category of geometric data organization techniques, those which organize the attribute space from which data are drawn. As opposed to the tree organizations discussed before, space partitions are placed always at fixed points in space independently of the data distribution. This distinction between the two types of organization is not however totally exclusive: Taminen considers the quadtree as a cell implementation scheme having slightly different space division criteria [Taminen 83]. We will present grid methods as non-hierarchical, thus excluding quadtree-like structures.

Cellular methods are based on the gridding of the space to provide direct access to each cell by coordinate-based address computation. Cells have simple geometric forms (e.g. rectangles, squares) forming a directory where each entry contains pointers to all the objects that are within the corresponding cell. These pointers are secondary memory addresses, as grid schemes were primarily intended to work in two levels of memory, i.e., to provide efficient multi-attribute key to address transformations. Wu and Burkhard discuss grid files as a category of dynamic hashing schemes with directory that efficiently process multi-attribute range queries, together with interpolation hashing methods [Wu and Burkhard 87].

Depending on the geometry of the cells, or equivalently, on the gridding process, we find regular cell grids as discussed by Taminen [Taminen 83] and irregular cell grids as proposed by Nievergelt and Hinterberger [Nievergelt and Hinterberger 84]. Both methods can be adapted to different data distributions by allowing adjacent cells to merge and to split in order to keep an acceptable filling ratio (remember each cell contains pointers to all objects within it, whose number must not exceed certain limits to prevent overflow and underflow of the cell's capacity). This flexibility is important when data are non-uniformly distributed in space, although regions can no longer in this case be directly indexed in disk, making use of directories to map regions to cells.

Figure 3 shows a regular space partition over attributes A and B, and the resulting directory (which is not really necessary in this case since partitions are equal sized, but we show it for clarity). Data, or equivalently a region, with attributes \( \{a_i, b_j\} \) will be stored in cell \( C_k \).
3. Current work

Experiments have been carried out with a collection of a few hundred objects, represented by their bounding rectangle in a two dimensional space. Since we were mainly concerned with retrieval of objects contained in a search window (i.e., answering queries of the form $x_i < k_1 < x_j, y_i < k_2 < y_j$, where $k_1, k_2$ is a key), the fact of representing explicitly the area occupied by the objects facilitated the computation of intersection, containment or enclosure conditions. It was also assumed that complex objects could be decomposed into simpler ones, each resulting object having its own (rectilinearly oriented) bounding box, disjoint from the others. This allowed for a certain uniformity of the size of the polygons we were to organize.

In the first experiments, a three-level decomposition symmetric tree structure (a quadtree) was chosen. We have one region, i.e. the entire space, at level 0, 4 regions at level 1, 16 regions at level 2, and 64 regions at level 3, which gives a total of 85 regions. It is expected that objects could be distributed, as uniformly as possible, among these regions. An object is said to belong to a region if this one is the smallest region that totally encloses it. Figure 4 shows a two level quadtree, and two objects A and B. Object A crosses the border between two adjacent level 1 regions, so it is only totally enclosed by the region at level 0. Object B is totally enclosed by a level 2 region.
In practice, objects that are enclosed by regions much larger than their size (the case of object A in the figure above) should be decomposed among neighbor regions. This will increase search accuracy, as objects are confined to smaller regions.

Next the regions in the tree are coded with a linear key [Abel and Smith 83], and the keys subsequently organized in a binary tree for search efficiency. With this organization and for orthogonal range queries (in our case, we used rectangular windows the size ten to twenty times the average size of the objects), the number of rectangles inspected in the database is $O(\sqrt{n})$, where $n$ is the number of rectangles. Another parameter of interest is the ratio between the number of objects actually contained in the search window and the number of retrieved objects; depending on the size of the window, we found it nevertheless not greater than 0.6, which shows that the locality of reference of the quadtree has its limits.

Another set of experiments, using the same collection of objects, involved the gridding of space and the distribution of the objects among the resulting cells. Partitions of the space were made at the points where the number of objects across adjacent cells was minimized, resulting in an irregular tessellation of space. Objects belonging to more than one cell were placed in another grid using the same principle as for the main grid, but this time objects in more than one cell were decomposed into objects each belonging to only one cell; the choice of two grids (why not three or four ?) is in fact arbitrary, and was motivated by the fact that the number of objects we had to place in the secondary grid was small enough to avoid another gridding of space.

Measuring the parameters we had already used for the quadtree, we found better performances in what concerns the accuracy of search: 70% to 80% of the retrieved objects for a search window were in fact intersected/included by the window region. This results from the fact that partitions were drawn at arbitrary places in space, whereas in the quadtree partitions resulted always in regions of equal size, independently of the position of the objects in space. The double grid presented also...
some advantages over a single grid. Although we had two directories, one for each grid, they were smaller in size than the directory of the single grid, i.e., they had a smaller number of entries. This is explained by the fact that a smaller number of objects had to be created as a result of cell’s crossing. In what concerns the number of retrieved objects, it was only slightly increased with the double grid, as expected, since now the search was performed over a larger number of cells.

As stated in the beginning, an important characteristic of CAD systems is that data is dynamic, and the spatial database must behave well under a changing collection of data. While a grid is a highly dynamic data store, the size of the quadtree must be chosen in advance which precludes a continuous decomposition of space; when choosing the size of the tree, one must thus have a fairly idea of the characteristics (in terms of both the size and the quantity of the objects) of the CAD database.

4. Integration in an existing environment

We have implemented our algorithms in an object-oriented environment described elsewhere [Barthès et al 79], which we could tailor to our needs. The programs were actually developed without concern for the CAD models of the objects we were intended to be dealing with, considering just that we will be given some means of extracting the primitives we needed from these models.

Figure 5 shows the organization we were developing a sub-system for. The spatial database (at right in the figure) consisted of the bounding boxes of the objects in the model database.

![Fig. 5: The spatial indexing sub-system](image)

The spatial database receives requests concerning the location of objects in space and can produce various outputs: there are cases where a simple boolean answer is sufficient.
(e.g. when answering *Is the object colliding with someone?*), and there are cases where it must report all the objects satisfying the query (e.g. for visualization purposes); in this case the identifiers of the objects are passed to the model database which will output the desired descriptions of the objects.

As we have explained before, a region representation was selected because it greatly simplifies intersection computation. Furthermore, storage space requirements are low: a bounding box is identified by the coordinates of one of its corners and the length of the diagonal.

5. **Conclusions**

Compared with coordinate-based address computation techniques, data organization techniques overcome the difficulty of mapping areas representing object's contours or bounding primitives to points in a higher dimension space. It is often the case that topological information (adjacency, crossing, etc) is lost in the mapping process. On the other hand, data organization, to be effective, must be hierarchical and this precludes direct access to data items; that is, searching objects within a region of space entails also a search in the regions which contain it (the upper levels in the hierarchy) and in the regions which are contained in it (the lower levels in the hierarchy).

The tradeoff between direct access (direct means a couple of disk accesses) and the explicitness of some relations, such as neighborhood, must be weighted taking into account the type of queries the system should be able to answer to. Random-like accesses little or nothing care about previous answers, whereas focalized search should emphasize proximity relations and the means of moving from one area to adjacent areas. In a large database, it seems that both approaches we have discussed are really needed; grids will serve as a low-level resolution data store, possibly indexing high-level resolution trees, where needed, or directly the objects themselves.

We were also interested in the integration of these indexing facilities in a larger database, possibly dealing with different kinds of information (textual, symbolic, geometric,...) The object-oriented paradigm proved to be a valuable testbed for testing our ideas, by allowing the smooth development of different, yet related, techniques.

**References**
[Abel 84]
D. J. ABEL
A B+ tree for large quadtrees
Computer Vision Graphics and Image Processing, 27, 1, 1984

[Abel and Smith 83]
D. J. ABEL, J.L. SMITH
A data structure and algorithm based on a linear key for the rectangle retrieval problem
Computer Vision Graphics and Image Processing, 24, 1, 1983

[Barthès 89]
Jean-Paul A. BARTHES
A review of some object representations for intelligent CAD environments
Third IFIP WG 5.2 Workshop on ICAD, Osaka, Japan, November 89

[Barthès et al 79]
Jean-Paul A. BARTHES, M. VAYSSADE, M. MIACZYNSKA-ZNAMIEROWSKA
Property Driven Data Bases
IJCAI, Tokyo, Japan, 1979

[Barthès et al 89]
Jean-Paul A. BARTHES, Patrice ANOTA, Kamal EL DAHSHAN
An experience in adding persistency to intelligent CAD environments
Third Eurographics workshop on ICAD, Texel, The Netherlands, April 89

[Bentley 75]
Jon L. BENTLEY
Multidimensional binary search trees used for associative searching
Comm. ACM, 18, 9, 1975

[Dobkin 87]
David P. DOBKin
Computational geometry: Then and Now
Springer Verlag 1987

[Jackins and Tanimoto 83]
C. L. JACKINS, Steven L. TANIMOTO
Quad-trees, oct-trees and k-trees: a generalized approach to recursive decomposition of euclidean space
Computer Vision Graphics and Image Processing, 5, 5, 1983

[Li and Loew 87]
Shu-xiang LI, Murray H. LOEW
The quadcode and its arithmetic
Comm. ACM, 30, 7, 1987
[Matsuyama et al. 84]
T. MATSUYAMA, Le Viet HAO, Makoto NAGAO
*A file organization for geographic information systems based on spatial proximity*
Computer Vision Graphics and Image Processing, 26, 3, 1984

[Nandy and Patnaik 87]
S.K. NANDY, L.M. PATNAIK
*Algorithm for incremental compactation of geometrical layouts*
Computer Aided Design, 19, 5, 1987

[Nievergelt and Hinterberger 84]
J. NIEVERGELT, H. HINTERBERGER
*The Grid-file: an adaptable, symmetric, multikey file structure*
ACM trans. on Database Systems, 9, 1, 1984

[Preparata and Shamos 85]
F.P. PREPARATA, M.I. SHAMOS
*Computational Geometry*
Springer Verlag 1985

[Samet 84]
Hanan SAMET
*The quadtree and related hierarchical structures*
ACM Computing Surveys, 16, 2, 1984

[Samet 88]
Hanan SAMET
*Hierarchical representations of collections of small rectangles*
ACM Computing Surveys, 20, 4, 1988

[Samet et al. 84]
Hanan SAMET, A. ROSENFELD, C.A. SHAFFER, R.E. WEBBER
*Processing geographic data with quadtrees*
Seventh Int'l Conference on Pattern Recognition, Montreal, Canada, 1984

[Taminen 83]
Markku TAMINEN
*Performance analysis of cell based geometric file organizations*
Computer Vision Graphics and Image Processing, 24, 2, 1984

[Tomiyama 89]
Tetsuo TOMIYAMA
*Object-oriented programming paradigm for intelligent CAD systems*
pp 3-31, in Intelligent CAD systems II: implementation issues, Ackman, ten Hagen, Verkamp, Eds, Springer Verlag 1989

[Vaydia 89]
Pravin M. VAYDIA
*Space-time trade-offs for orthogonal range queries*
SIAM Journal on Computing, 18, 4, 1989
[Wu and Burkhard 87]
C.T. WU, W.A. BURKHARD
*Associative searching in multiple storage units*
ACM trans. on Database Systems, 12, 1, 1987