LA-UR 95-910

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE: **THE RDB - A PARALLEL SPATIAL DATABASE FOR THE IES/BTI SYSTEM**

AUTHOR(S): K. Carlson, L. Winters

SUBMITTED TO: The 4th International Symposium on Large Spatial Databases SSD '95

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive royalty-free license to publish or reproduce the published form of this contribution or to allow others to do so, for U.S. Government purposes.

The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED DT

Los Alamos

Los Alamos National Laboratory
Los Alamos New Mexico 87545

MASTER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

The RDB - a Parallel, Spatial Database for the IES/BTI System

Kristi Carlson, Larry Winter
Computer Research and Applications Group
Los Alamos National Laboratory
Los Alamos, New Mexico

1.0 Introduction

The manipulation and representation of spatial data on computers is an important issue in many computer applications (computer graphics, computer vision, database management systems, geographic information systems). Spatial data, which consists of points, lines, and regions in 2-dimensions, can be difficult to manage efficiently because it is often quite voluminous. For instance, the number of picture elements in even a small digital image is on the order of a million, while the number of locations stored in a terrain database can easily include billions of points. Furthermore, the kinds of operations performed on spatial data -- set operations, insertion, deletion, searches such as "near" -- are compute intensive, and hence slow unless the data is structured to reflect its underlying topology. Hence a conventional database which is organized on search keys is often not adequate for handling spatial data.

In order to provide efficient manipulation of spatial data, both efficient data structures and parallel computing can be employed. The data structures may be organized to provide efficient spatial operations, and parallel computing allows us to operate on large subsets of data in parallel.

The Image Exploitation System, which is an automated image analysis system, has a great need for efficient storage and manipulation of spatial data. The Image Exploitation System is part of the Advanced Research Project Agency's Balanced Technology Initiative and is abbreviated IES/BTI. IES/BTI must process tens to hundreds of megabytes of imagery in a few minutes, and is composed of many independent components which need to access and share spatial data. The system needed an efficient parallel spatial database, hence the motiviation for our work on the Region Database, or RDB.

The RDB is our attempt to meet the needs of the IES/BTI Cycle 2 system. The RDB provides for storage and retrieval of both raster and vector based spatial data as well as attribute-based retrievals. It also provides facilities for conversion between the two representations of spatial data (raster and vector) and for efficient, parallel boolean operations on vector data. In this paper we discuss the reserach and development performed to design and implement the RDB. [1]

2.0 The IES/BTI System

The IES/BTI system attempts to speed up the analysis of images obtained from synthetic aperture radar sensors by identifying where enemy forces are likely to be. It must process many types of data in order to perform its inference: terrain data, vehicle classifications, force structure, and signal intelligence information. It is intended to be used by image analysts to help them to quickly focus on the images most likely to contain information about enemy forces. As mentioned, IES/BTI must process large amounts of data in a short amount of time; specifically it is required that IES/BTI must process 128 megabytes of imagery every 5 minutes. This led to a system design which was based on high-speed parallel processing. Specifically, the Cycle 2 version is implemented in the C* data parallel programming language on a 32 node Connection Machine 5. [WS95]

The IES/BTI system architecture consists of a set of components and data flows through them, and the RDB is simply one more C* component in the system. However, unlike the other components in the system it is implemented as a library of data-parallel C* functions and is never called as a stand-alone process. The RDB receives data both prior to an IES/BTI run, mostly a priori terrain data (Interim Terrain Data and Digital Terrain Elevation Data from the Defense Mapping Agency) that will not change from run to run. It also receives data from many sources during an IES/BTI run which includes individual military force detections (represented as points), and military forces which are clustered into larger forces such as battalions (represented as regions). As an example of how IES components may use the RDB, consider the following: in a typical IES/BTI run, it is possible for the Detection component to process an image and identify possible military forces which it stores in the RDB, the Hospitability component may retrieve both detections and terrain information from the RDB and weed out detections that are impossible because of the underlying terrain and store the results back to the RDB, the Cluster component retrieves these pruned detections and clusters them into possible higher level forces and so on. Clearly, IES/BTI needs an efficient database to process this amount of information in a timely manner.

3.0 Rdb Design

The most important RDB requirement is to provide for fast storage and spatial searches. The need for efficient spatial queries drove the design of our data organization. It was necessary to implement an efficient parallel spatial index as well as provide fast access to secondary storage. It was also necessary that we support retrieval by search key (or object attribute), however it was of secondary importance. Other requirements included providing translation between raster and vector representations of spatial data as well as providing efficient manipulation of vector-based polygonal data. This entailed implementation of data parallel raster-to-vector conversion functions as well as data parallel implementations of boolean operations on vector data (UNION, INTERSECTION, etc.).

1. This research and development was performed in part using the resources located at the Advanced Computing Laboratory of Los Alamos National Laboratory.

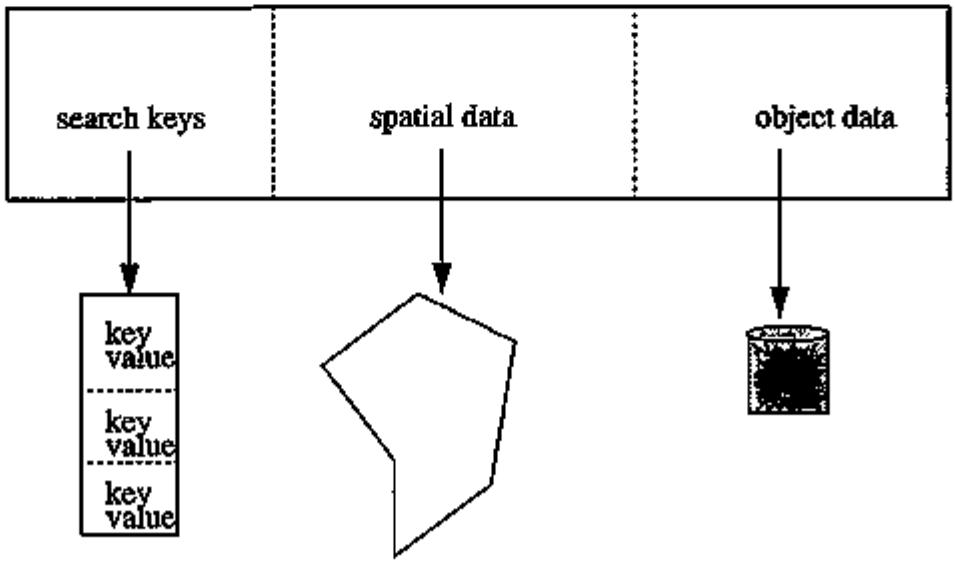
3.1 Data Organization and Representations

The IES system has two basic classes of data that are stored in the RDB: overlay data and object data. Each type of data has a different internal representation and different spatial index in the RDB.

Overlay data, mostly a priori terrain data such as elevation, consists of two-dimensional grids of numbers. This data type is used to calculate and store hospitability data and the like. Given the structure of the data and how it is used in IES/BTI, using a raster representation is a sound and straightforward approach. However, most terrain data is too large to simply store as a single grid, hence some kind of indexing was needed to divide the terrain data into sub-gridsthat could be loaded into memory at once. Also, the area of operation for an IES/BTI run is typically too large to represent as a single grid on the CM-5.

Objects are intended for use in representing every kind of IES/BTI data other than overlay data. Hence, a generic, flexible data type was needed. The data type designed for this use, the RdbObject, consists of 3 main fields: spatial data (either a point, line segment or polygon which defines where an object is located as well as what its spatial extent is), attribute information (actually search keys that can be used to query for objects) and object data (see figure 1).The only fields used in queries are the spatial data and the search keys; the object data is simply a pointer to data that the RDB stores and retrieve without examination or interpretation. This provided a flexible design which allows the RDB to remain independent of the specific data that is stored by IES/BTI components, as well as providing flexibility to users of the RDB (who need not define their data types until run-time) [CW94b].

Figure 1: Data Abstraction for RdbObject

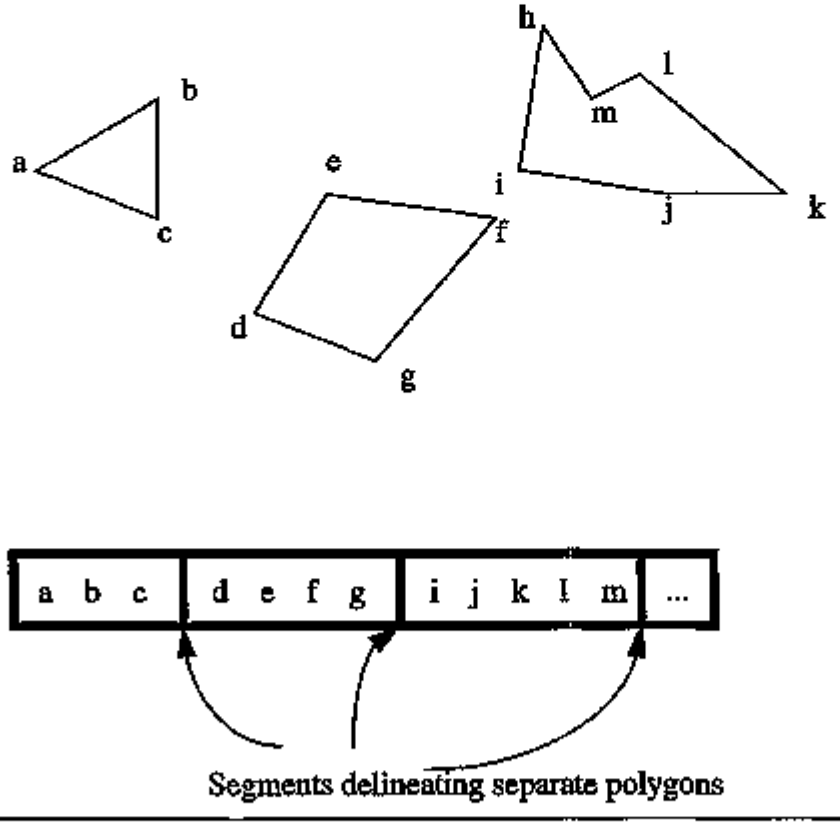


3.1.1 Representation of Spatial Data

RdbObjects can have the spatial type of point, line segment, polygon or none. For point and line segment data, we use the obvious representation of storing the single point or the pair of points for the object. For polygonal data, we performed experiments on a 32-node partition of the CM-5 to compare the relative efficiencies of a raster or full-resolution grid representation and a vector or vertex-list representation[CWK94]. In these experiments, efficiency was measured both in terms of the amount of space required to store polygonal objects and in the amount of time it took to perform typical operations on polygonal objects and databases of objects. We used the same spatial index for both representations.

For a subset of space in our spatial index, each vertex list consisted of a segmented parallel variable of vertices, each segment corresponding to an individual polygon (see figure 2). Thus, operations on the list of polygons could be done in parallel, even operations on individual vertices could be done in parallel.

Figure 2: a collection of polygons in a segmented vertex list



We tested operations such as searching for all polygons within a window which operated on a collection of polygons as well as lower level operations like intersection of two polygons. As would be expected on a data-parallel machine such as the CM-5, the lower level boolean operations were very efficient (they should take constant time for a parallel raster representation on a data-parallel machine). For all other operations, however, we found that in both storage requirements and in average time it took to perform various operations, the vector representation was superior. This was not unexpected since the vector format's storage requirements are based on the complexity of the polygon border, not the area of the polygon and that resolution does not effect the amount of storage required. Because of the reduced size of the vector data, it was possible to operate on far more polygons at once than was possible with the full-resolution grid representation. Hence we chose to store polygonal objects in vector format and had the new task of developing efficient, parallel boolean operation on vector-based polygons.

3.2 Spatial Indexing

3.2.1 The Parallel Index Grid

One possibility for spatial indexing in a data-parallel environment is to use a uniform index grid. One advantage of this method is its simplicity both conceptually and in implementation. It is also potentiall more efficient than hierarchical indices for operations which must examine all items in the data set. We chose this method for indexing overlay data in the RDB, implementing a parallel, 2-D index grid which subdivided the image space into "tiles", or sub-grids. Implicit in the location in the index grid parallel variable (pvar) of each grid element is the location and area that each grid-element indexes. When a spatial retrieval such as a window operation is done, the query is first mapped into the index grid to determine which pieces of overlay data need to be brought into memory, then the operation is completed on those pieces of data (see figure 3). The mapping into the grid is simply a rectangle intersection function performed in parallel on all grid elements at once.

There are potential drawbacks to this method of indexing that arise when the database is sparsely populated or when the indexed area is large but the data is clustered in only a few grid elements (hence many elements in the index grid do not have any data associated with them). It also is not appropriate for very large data sets that either may result in an index grid that is too large for the available memory or in "tiles" that are too large to operate on efficiently. An alternative approach would be to implement a pyramidal index structure such as a hierarchical index grid. However, for the purposes of the RDB this index grid is sufficient. The overlay data is all a priori terrain data and we knew that it was easily indexed by such a data structure. For RdbObject data, however, we could not guarantee that the amount of data would be well-confined. Hence we chose a different spatial index for that data type.

Figure 3: Parallel Index Grid and the Window Operation

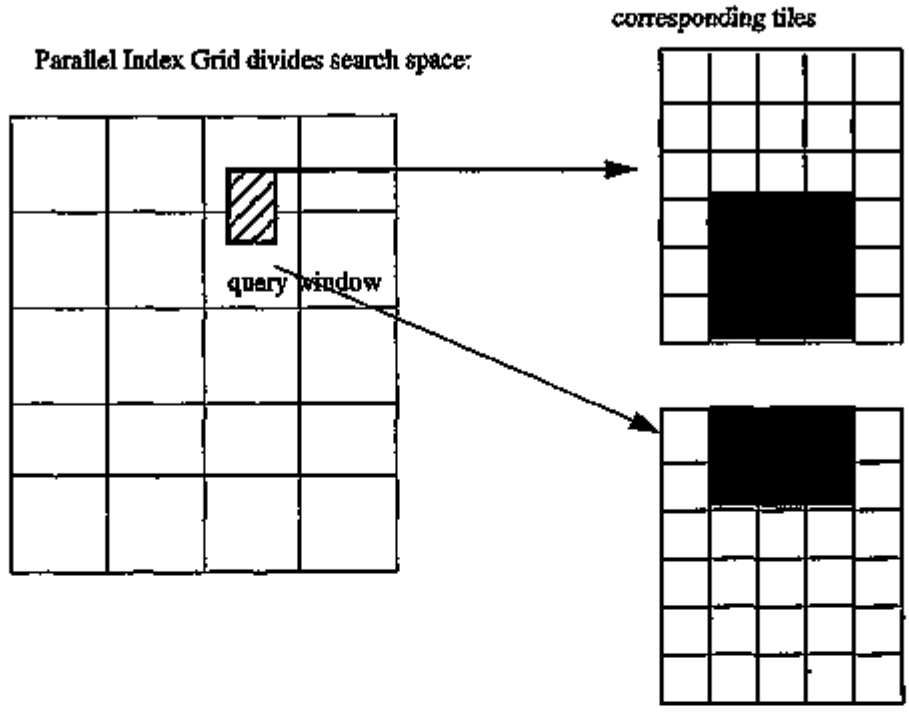


Figure 1

3.2.2 Hierarchical Spatial Indices

Tree-based data structures are an efficient way to store spatial data because they naturally support important topological relations like region inclusion. Also, their low space requirements and the *log(N)* height of these trees allows for fast searching of the data structure. A number of tree representations have been proposed for structuring spatial data. Among the most successful have been variations on the quadtree data structure.Quadtrees are a class of hierarchical data structures that recursively decompose two-dimensional space into quadrants. Each node of a quadtree spans a portion of space and its child nodes divide up that portion into four (equal or unequal) parts. Quadtrees vary in the type of data they are used to represent (point, line or region), the decomposition rule and the resolution (variable or not). In this context, resolution indicates the number of times a decomposition process is applied and may be fixed or may be governed by properties of the input data [Sam90].

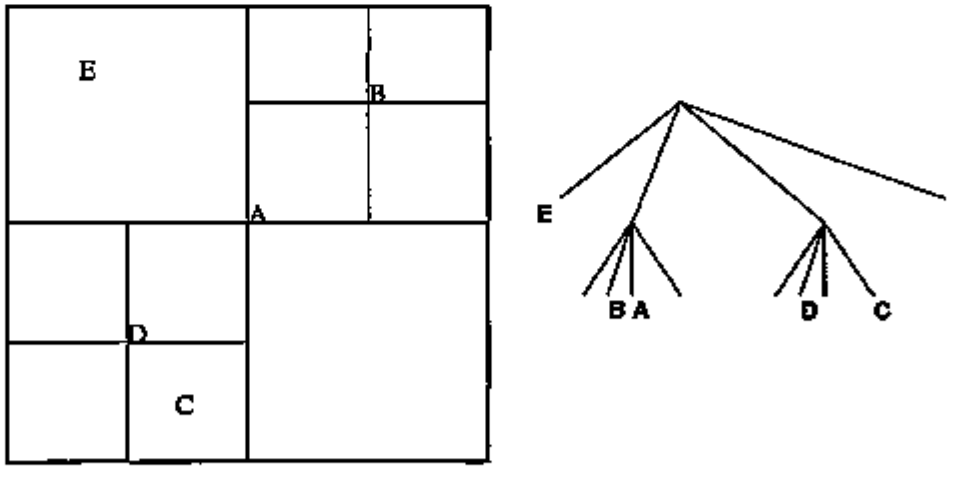
Sequential Indices. Early in our RDB design we performed experiments to determine the best indices for a sequential version of our index. These experiments were done to support development of an RDB for an earlier IES/BTI system which was to reside on an Encore Multi-max, a shared-memory parallel computer. We chose to look at quadtrees because the data for IES/BTI would be sparse and cover a large area, conditions the quadtree is well-suited for. IES/BTI would also require fast spatial lookup.

The experiment compared different quadtree index structures for point and rectangular data and measured their efficiency in performing various operations we thought would be important to the IES system. Rectangular data was used rather than polygonal data because it was anticipated that polygonal data would be stored by its minimum bounding rectangle. Queries would first be performed on the minimum bounding rectangles, then the operation would be performed on the actual polygons of any matching objects. Our experiments measured the effect of data density and clustering on the performance of the various quadtrees, and determined that for IES' purposes the PR quadtree was the best choice for point data and the MX-CIF quadtree was the best choice for rectangular [CW94a].

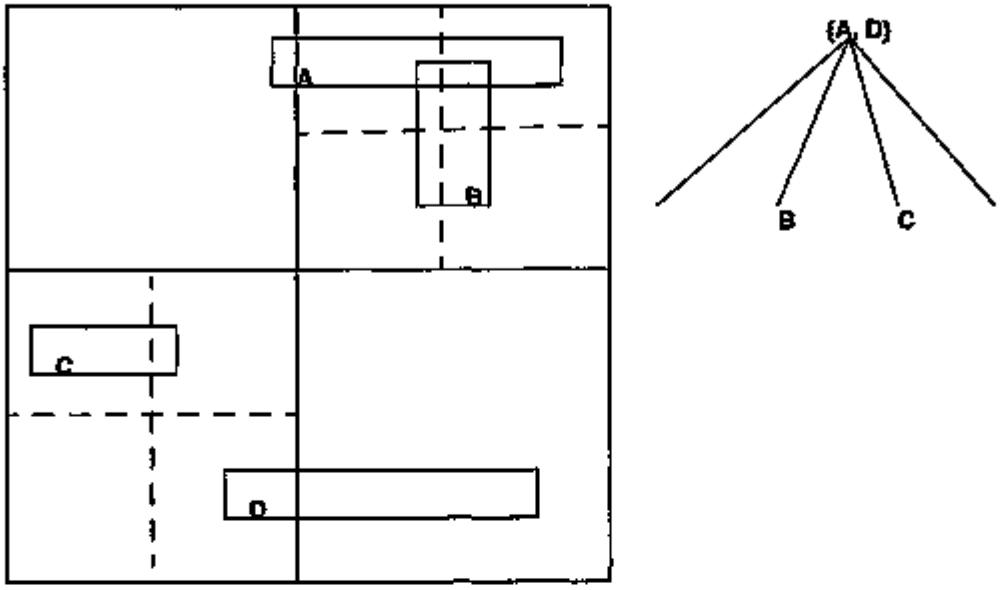
For a PR Quadtree, quadrants are subdivided into equally sized sub-quadrants of size 2^m x 2^m. A point datum is added to a PR Quadtree by finding the current quadrant into which the datum fits and inserting the datum if the quadrant is unoccupied. If the quadrant is already occupied, the quadrant is split until and empty sub-quadrant is available See Figure 4.

In an MX_CIF tree, each rectangle, R, is associated with the quadtree node corresponding to the smallest quadrant that contains R entirely. Hence data is stored at all levels of the tree, not just at the leaves. And since many rectangles can be associated with an individual node, an auxiliary data structure is needed to keep the list of rectangles associated with each quadrant. See Figure 5. For a more detailed description of these data structures, see [Sam90].

Figure 4: the PR-Quadtree



Figrure 5: The MX-CIF Quadtree



Not only were these two quadtrees the most efficient, but they also share the advantage that both use a regular decomposition rule. In other words, each quadrant subdivision divides the quadrant into four equal sub-quadrants. Thus quadrants at the same level and location in two different quadtrees that both use regular decomposition will be identical in the space they span. This makes it possible to perform operations across very different types of data (points and polygons) more efficiently. Also, of the trees tested the PR and MX-CIF quadtrees provided the most compact representation (i.e. fewer levels in the tree) of the quadtrees tested. This turned out to be useful for our parallel implementation.

Parallel Hierarchical Structures. Sequential hierarchical structures are not ideal for data-parallel architectures such as the CM-5 because they do not take advantage of its parallelism. Yet, because our search space was potentially large and the data potentially sparse we wanted to use a hierarchical, spatial index vs. another approach such as a parallel index grid. Furthermore, it was unlikely that IES/BTI would generate enough data so that an actual parallel version of the tree as discussed in [Bes90] or [Kuf95] was warranted. It was necessary to implement a structure that took advantage of parallelism and had a small, efficient tree structure. Ideally, we wanted a hierarchical, spatial index that allowed for the maximum number of objects to be operated on in parallel without over-

loading the available resources (memory)m effectively resulting in a flatter index that the sequential quatrees.

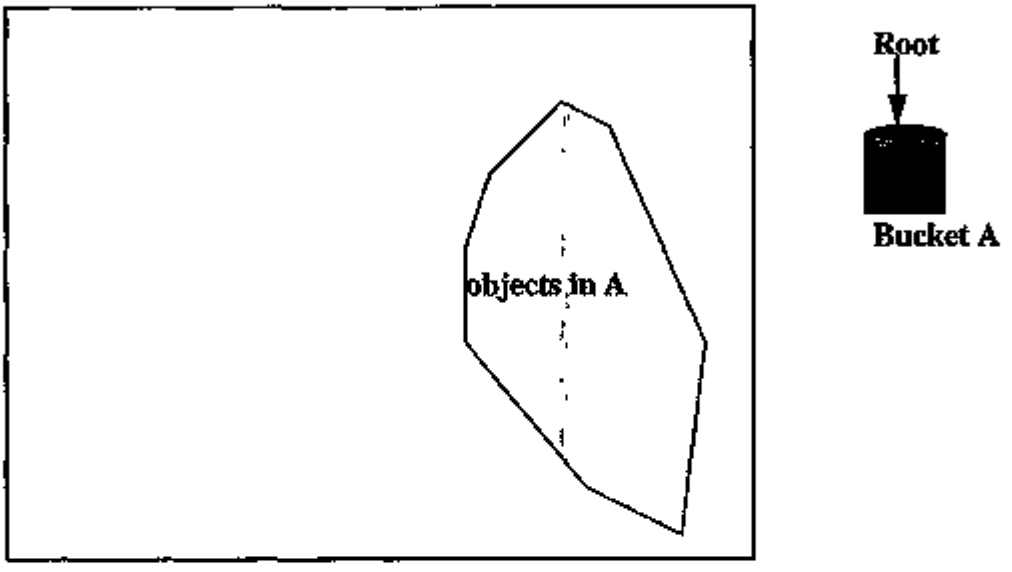
Parallel approaches are still a topic of research. Hoel and Samet used parallelism to build R-trees quickly and to build the results of queries quickly. [HS93]. Kuijt proposes in his thesis proposal to use a data-parallel index compression method that takes a hierarchical tree structure and scales it based on the available number of processors. Kuijt calls this p-ary tree (where p is the number of processors) the Par-tree. [Kui95]. Kuijt's approach could prove very useful for very large databases.

For the RDB, we chose to use bucket versions of our sequential quadtrees. Bucket methods simply store a collection of objects at a node rather than a single object, so an insertion would insert an object into an already occupied node if the bucket threshold had not yet been reached. The result is a smaller tree with fewer "structure nodes". We mapped this into our data parallel domain by representing each bucket as a parallel variable forming a parallel bucket quadtree. The bucket threshold is scaled to the maximum available memory so the maximum number of objects can be loaded into memory and operated on at a time. When the amount of data is large, this also allows us to keep only the structure nodes of the index in memory at a time, loading a bucket into memory as is needed to answer queries.

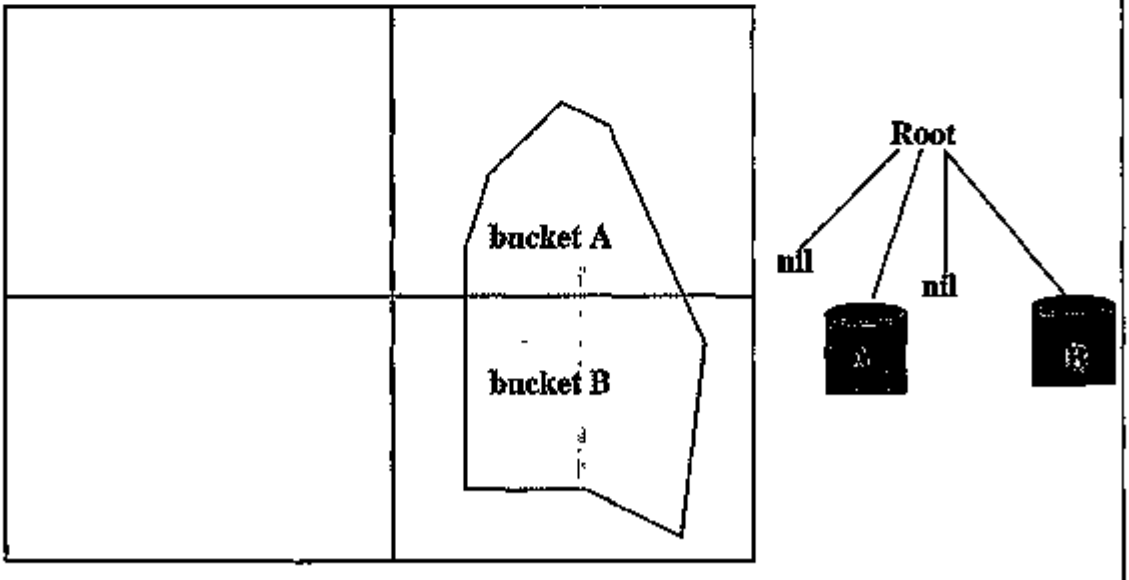
In the case of a small number of objects (e.g. 1000), ideally we would like for all to be loaded into a parallel variable at once so that operations can happen to all objects in parallel. However, if there are more objects it may be necessary to divide them into two separate buckets in order to operate on a collection of them at once. The bucket quadtrees accomplish all of this. If there are only 1000 objects in the database and the bucket threshold is larger than 1000, the quadtree will only have one node, its root, and all objects will be stored in that node. In this case, our datastructure has degenerated into a parallel array. Once the number of objects becomes greater than the bucket threshold, the root is split into 4 sub-quadrants and the objects are divided between them based on their spatial information (See figure 6). This subdivision of objects can be done with parallel SCAN primitives on the CM-5 quite efficiently.

Figure 6: Parallel Bucket PR-Quadtree

Parallel Bucket PR tree - with number of objects less than bucket capacity (objects enclosed in shaded area)



After addition of objects to bring total number of objects above bucket capacity:



3.3 Secondary Storage

Due to the RDB's need for fast response to queries, we needed to use the most efficient means of secondary storage. Thus we chose to use the CM-5's Scalable Disk Array,or SDA, for files containing large amounts of parallel data. Parallel files are formatted to take advantage of the CM-5's massive parallelism, each CM processor having its own data stream to and from the SDA. The data rates for a 32 node partition writing to the SDA is about 17 megabytes per second based on benchmarks performed at the Advanced Computing Laboratory of Los Alamos National Laboratory. Our own benchmarks indicated that we could read in 20 megabytes of data (ten 512x512 images) in about 1 second. However, for small amounts of data (less than a megabyte per node) the performance of the SDA may be even slower than for regular file IO. Hence, small files (initialization files, etc.) are stored as non-parallel files.

3.4 Queries

3.4.1 Spatial Queries

As was noted in the Spatial Index section, spatial queries (storage and retrieval by location) are first mapped onto the spatial index to prune the search to a smaller amount of data. For spatial operations on overlays (mostly *window* operations), this entails mapping the query onto the index grid, then retrieving the tiles that correspond to the selected index grid elements and completing the query. For RdbObjects, the queries are first mapped onto the parallel bucket quadtree, then the corresponding bucket or buckets are loaded into parallel variables for completion of the operation.

3.4.2 Attribute Queries

Retrieval of objects by attribute (e.g. retrieving all detections that are of type *tank*) is provided. However, it is a brute-force operation entailing retrieval of each bucket in turn and searching for objects that match the query. Each bucket can be searched in a data-parallel fashion, and we keep a separate table that indexes all of the buckets so we need not visit the spatial index to access them. However, buckets are organized by spatial information, hence the data is not organized for optimal attribute-based queries.

It is also worth noting that, though explicit temporal information is not maintained by the RDB, time can be defined as an object attribute or search key so temporal queries can be mapped into attribute queries.

3.4.3 Raster to Vector Conversion Functions

As has been noted, the RDB was required to provide conversion between raster and vector representations of spatial data. This was mainly to facilitate applying terrain data which is

stored in raster format to object data which is stored in vector format (e.g. find all tanks that are located in bodies of water).

We developed conversion algorithms for raster and vector images with block-like connected components. We based these algorithms on [Ble90] using scan primitives and pointer jumping, no step of which took greater than $O(\log n)$ time (where n is the image size and the number of processors). See [ZCW94] for more details.

3.4.4 Boolean operations

The RDB needs to be able to dtermine questions such as "which polygons overlap a search window" quickly for fast retrievals. Thus it was necessary to implement a data-parallel version of polygon intersection functions that operated on vector data and returned a yes or no result, in parallel, for all objects being tested. The RDB was also required to perform lower level operations on spatial data, specifically boolean functions (intersection, union, complement) on polygons that would return the actual resulting polygon(s) of the operation. For instance, the RDB needed an intersection function that would take two polygons and return the polygon or polygons which was the result of the intersection operation.

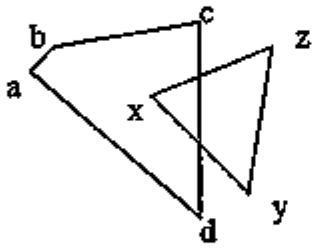
We based our implementation of the boolean function on work done by David Kuijt at the University of Maryland. Kuijt developed efficient sequential algorithms for boolean operations on vector format data.

For boolean operations on two polygons, Kuijt's algorithms all begin by classifying the vertices as inside or outside the other polygon, expanding the polygons to include the intersection points between the two polygons' edges, and classifying the edge fragments as being inside or outside the other polygon. Then these must be organized into new polygons depending on the operation (intersection,union, etc) [Kui95].

We attempted to implement a data-parallel verion of Kuijt's algorithms with mixed success. First, we took the two polygons and translated them into matrices that would allow us to perform intersect operations in data-parallel (see figure 6). We copied the first polygon's edges into the first *column* of its corresponding matrix, then broadcast them along the x-axis using a parallel *scan* operation. The second polygon's edges were copied into the first *row* of its matrix, then broadcase along the y-axis. Edge intersection of all of the edges of the first polygon with every edge in the other polygon was determined in parallel by operating on these matrices. Expanding both polygons was accomplished with the parallel *send* operation. However, calculating whether or not each edge of the polygon was inside or outside the other polygon required one sequential step that traced the boundary of one of the polygons, though the remainder of the operation used the parallel *scan* operation.

Ours was a fairly straighforward adaptation of Kuijt's sequential algorithm to the CM-5. It is possible that the boolean operations can be done much more efficiently as proposed by Kuijt in [Kui95].

Figure 6: boolean operations on polygons - data structures



matrices for polygons

ab ab ab	xy yz zx
bc bc bc	xy yz xz
cd cd cd	xy yz zx
da da da	xy yz zx

The edges of the first polygon are broadcast along the x-axis of the matrix, the edges of the second polygon are broadcast along the y-axis. Now, intersections can be calculated among all segments in parallel, and can be counted using scan operations.

4.0 Futures

There are many enhancements possible for the next version of the RDB. We would like to investigate data-parallel hierarchical grid methods and pyramidal indexing structures which may prove to be more efficient for indexing raster data (and possibly even object data). This may also prove to be useful in integrating spatial and attribute retrievals without sacrificing the performance of the attribute searches. Efficient indexing of attrivure data is another area we need to investigate. We would also like to perform experiements on our data-parallel bucket quadtrees to determine their efficiency in relation to data density, clustering and bucket size.

5.0 References.

[Bes92] T. Bestul. Parallel paradigms and practices for spatial data. PhD thesis, Univeristy of Maryland, College Park, MD, April 1992.

[Ble90] G.E.Clelloch. Vector Models for Data-Parallel Computing. The MIT PRess, Cambride, MAssachusetts, 1990.

[CW94a] K.M. Carlson and C.L. Winter. Experiements with Quadtree Data Structures, Los Alamos Unclassified Report number LA-UR-94-0359, Los Alamos Nationa Laboratory, 1994.

[CW94b] K. M. Carlson and C.L. Winter. Region Database Component Design Document. Report prepared for US Army Topographic Engineering Center, Fort Belvoir, Virginia, Los Alamos National Laboratory, 1994.

[CWK94] K.M. Carlson, C.L. Winter and David Kuijt. Data Structures for the RDB: Preliminary Experiments on Polygonal Data, a report prepared for the Topographic Engineering Cetner, Los Alamost National Laboratyr, 1994.

[HS93] E.G. Hoel and H. Samet. Data-parallel R-tree altorithms. In S. Haririr and P.B. Berra, editors, Proceedings of the 1993 International Conference on Parallel Processing, pages 47-50, St. Charles, Il., August 1993.

[Kui95] D. Kuijt. Data-Parallel Polygon Operations and Indexing: A Thesis Proposal, unpublished, University of Maryland, 1995.

[Sam90] H. Samet. The Design and Analysis of Spatial Data Structures. Addison-Wesley, Reading, MA, 1990.

[WS95] C.L. Winter and M.C. Stein. IES/BTI System Overview. Los Alamos Technical Report, February 1995.

[ZCW94] D. Zhang, K.M. Carlson, C.L Winter. Efficient Parallel Raster/Vector Image Conversion Algorithms for Spatial Databases, unpublished, Technical Report. Los Alamost National Laboratory, 1994.