DATA-PARALLEL R-TREE ALGORITHMS*

Erik G. Hoel†
Geography Division
Bureau of the Census
Washington, D.C. 20233

Hanan Samet
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Sciences
University of Maryland
College Park, Maryland 20742

Abstract – *Data-parallel algorithms for R-trees, a common spatial data structure are presented, in the domain of planar line segment data (e.g., Bureau of the Census TIGER/Line files). Parallel algorithms for both building the data-parallel R-tree, as well as determining the closed polygons formed by the line segments, are described and implemented using the SAM (Scan-And-Monotonic-mapping) model of parallel computation on the hypercube architecture of the Connection Machine.*

INTRODUCTION

The SAM (Scan-And-Monotonic-mapping) model of parallel computation [1] may be defined by one or more linearly ordered sets of processors which allow element-wise and scan-wise operations to be performed. A *scan* operation [2] takes an associative operator $\bigoplus$, a vector $[a_0, a_1, \ldots, a_{n-1}]$, and returns the vector $[a_0, (a_0 \bigoplus a_1), \ldots, (a_0 \bigoplus a_1 \bigoplus \cdots \bigoplus a_{n-1})]$. Both within and between each linearly ordered set of processors, monotonic mappings may also be performed. Being a subset of the scan-model [2], the SAM model considers scan operations as taking unit time, thus allowing sorting operations to be performed in $O(\log n)$.

The R-tree [3] is a data structure for representing spatial data based upon spatial occupancy. Such methods decompose the space from which the data is drawn into regions called *buckets*. The R-tree buckets the data on the basis of minimal bounding rectangles (in the 2-d case). Objects are then grouped into hierarchies, and then stored in another structure such as a B-tree.

The R-tree's drawback is that it does not result in a disjoint decomposition of space. An object is associated with a single bounding rectangle, but the area spanned by the rectangle may be included in several other bounding rectangles. To determine which object is associated with a particular point in a 2-d space, we may have to search the entire database in the degenerate case. Alternatives such as the R+-tree and the PMR quadtree [4] which decompose the space into disjoint cells which are then mapped into buckets are not described here.

This paper is organized as follows; First, we present the data-parallel R-tree and give an R-tree building algorithm. Next, we describe the parallel implementation of a polygonization.

*This work was supported in part by the National Science Foundation under Grant IRI-90-17393.

†Also with the Center for Automation Research at the University of Maryland.

PARALLEL R-TREES

We limit ourselves to objects which are line segments although our techniques are applicable to other objects as well. Standard sequential R-trees are constructed in a manner whereby all leaf nodes appear at the same level in the tree. Each entry in a leaf node is a 2-tuple of the form (R, O) so that R is the smallest rectangle enclosing line segment O (where O points to the actual line segment). Each entry in a directory (non-leaf) node is a 2-tuple of the form (R, P), where R is the minimal rectangle enclosing the rectangles in the child node pointed at by P. An R-tree of order (m, M) means that each node in the tree, with the exception of the root, contains between $m \leq \lceil M/2 \rceil$ and M entries. The root node has at least two entries unless it itself is a leaf node.

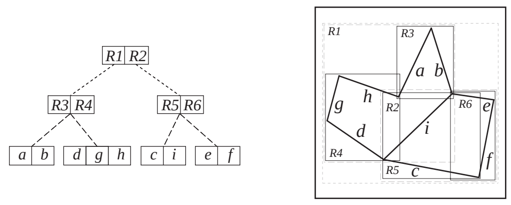

Figure 1: Example order (1,3) R-tree.

R-trees are built in the same way as B-trees. Line segments are inserted into leaf nodes. The appropriate leaf node is determined in a top-down fashion by traversing the R-tree starting at its root and at each step choosing the subtree whose corresponding bounding rectangle will have to be enlarged the least by the addition of line segment x. Once a leaf node is determined, a check is made to see if the insertion of x will cause the leaf node to overflow. If the leaf node is overflowing, it is then split and the $M + 1$ 2-tuples are redistributed among the two resulting nodes. Splits may propagate up the R-tree. Note that the tree's final shape depends on the insertion order of the line segments. In the data parallel environment, all line segments are inserted at the same time, and thus the final shape of the parallel R-tree will likely not be the same as its sequential one.

The parallel R-tree building algorithm proceeds as follows. Initially, one processor is assigned to each line of the data set, and one processor to the resultant R-tree (e.g., Fig. 2). Assume an order $(1, 3)$ R-tree. In the figure, label A denotes the line processor set, C_0 denotes the R-tree node processor set, with the associated square region containing the line processor identifier for the corresponding line segment group (i.e., the line processor segment group starting at processor 0 in A), Within set A, the square regions contain the line identifiers and indicate which processor is associated with which line. Each of the line processors is associated with the single R-tree node processor. A downward scan operation (i.e., a scan which returns the vector $[(a_0 \bigoplus a_1 \bigoplus \ldots \bigoplus a_{n-1}), (a_1 \bigoplus a_2 \bigoplus \ldots \bigoplus a_{n-1}), \ldots, a_{n-1}]$) is performed on the line processor set to determine the number of lines associated with the single R-tree processor (shown in Fig. 2 as the `count` field beneath the line processor set). The number of lines in the *segment* (a collection of line processors associated with a single R-tree node processor) is then passed to the single R-tree node processor. If the number of lines in the segment exceeds M, then split the R-tree root node into two leaf nodes and a root node (as is also done for the sequential R-tree). The two new leaf nodes are inserted into the R-tree processor set, with the former root node/processor updated to reflect the two new children.

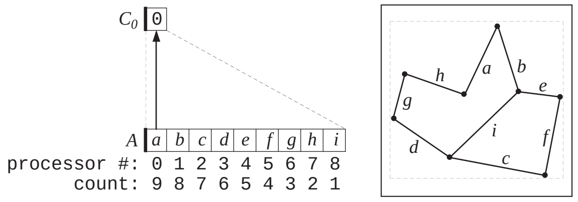

Figure 2: Initial processor assignments.

We use a node splitting algorithm that first sorts all lines in the segment according to the left edge of their bounding boxes. A parallel upward scan operation (i.e., a scan which returns the vector $[a_0, (a_0 \bigoplus a_1), \ldots, (a_0 \bigoplus a_1 \bigoplus \ldots \bigoplus a_{n-1})]$), is used to determine the extents of the bounding box formed by all lines preceding a line in the sorted segment. Similarly, a downward scan determines the bounding box for all following lines in the segment. For all legal splits (i.e., where each of the two resulting nodes receives at least m/M of the lines being redistributed), the amount of bounding box overlap is calculated, with the split corresponding to the minimal amount of overlap being selected as the x-axis candidate. An analogous procedure is employed for the y-axis in obtaining the y-axis candidate split coordinate value. Next, we choose the candidate split coordinate value corresponding to the minimal bounding box overlap. In case of a tie, use another metric such as the split with the minimal bounding box perimeter lengths. The complexity is $O(\log n)$ at each stage of the building operation as we employ two $O(\log n)$ sorts and a constant number of scan operations.

Once the splitting axis and the coordinate value are chosen, an *un-shuffle* operation [1] (where two intermixed types are rearranged into two disjoint groups via two monotonic mappings) is used to concentrate those line processors together into two new segments, each of which corresponds to one of the two R-tree leaf node processors (Fig. 3). For example, all lines with a midpoint less than the split coordinate value are monotonically shifted toward the left, while those which are

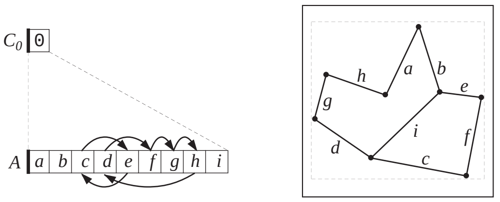

Figure 3: Un-shuffle operation.

greater than the split coordinate value are monotonically shifted toward the right among the line processors. Note that the root node of the R-tree is now associated with two segments in the line processor set A (i.e., (a, b, e, h) and (c, d, f, g, i)), and must itself be subdivided in an analogous manner. Fig. 4 shows the result which consists of two segments in the line processor set A, and two different R-tree processor sets C_0 and C_1 (each set corresponding to a node at a different height in the R-tree).

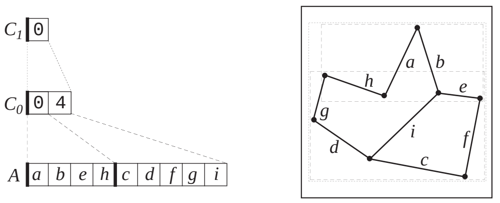

Figure 4: Completion of root node split operation.

The insertion algorithm proceeds iteratively as before, with each segment determining the number of lines it contains, and transmitting the count to the associated R-tree processor. If the number of lines in the segment is greater than M, then the segment (and corresponding R-tree node processor) are forced to subdivide (i.e., split). Note that this subdivision process may result in processors that correspond to internal nodes in the R-tree splitting themselves (with these splits possibly propagating up the R-tree). The building process terminates when all nodes in the R-tree processor set have at most M child processors. The R-tree root node corresponds to the single processor in set C_2, the leaf nodes are contained in processor set C_0, and all lines area grouped in segments of length ≤ 3 in processor set A (recall that our R-tree is order $(1, 3)$).

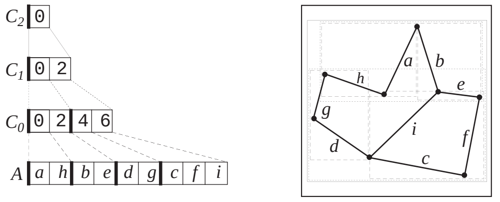

Figure 5: Completion of the R-tree building operation.

The data-parallel R-tree building operation is

$O(\log^2 n)$. Each of the $O(\log n)$ stages is $O(\log n)$ (a constant number of scans and two bounding box sorts).

POLYGONIZATION

Polygonization is the process of determining all closed polygons formed by a collection of planar line segments. We identify each polygon uniquely by the bordering line with the lexicographically minimum identifier (i.e., line number) and the side on which the polygon borders the line. Polygonization can be done without a data-parallel R-tree. Basically, the lines could be sorted based upon their identifier in $O(\log n)$ time, then each line in sorted sequence would transmit its endpoint coordinates, line identifier, and current left and right polygon identifiers to all following lines via a sequence of $O(n)$ scan operations. Each line can independently determine the identifiers of the left and right polygons. The drawback is that it is an $O(n)$ operation with a large amount of scans. The R-tree decomposition can reduce the amount of global scan operations (i.e., a scan across the entire processor set) by instead relying upon segmented scans executed in parallel, thus speeding the computation.

Given a data-parallel R-tree, start by constructing a partial winged-edge representation (see, e.g., [4]) (an association between the incident line segments forming the minimal and maximal angles at each endpoint of each segment). This representation enables us to determine all edges that comprise a face (i.e., polygon) and all edges that meet at a vertex in time proportional to the number of edges. It consists of face, vertex, and edge tables. The face table has an entry for each face which is one of the face's constituent edges. The vertex table has an entry for each vertex which is one of the edges that meets at the vertex. The edge table has an entry for each edge which consists of the two vertices defining the edge, the two adjacent faces, and the preceding and following edges for each of these faces.

We implicitly construct the entire data structure although our example only illustrates how we determine the adjacent faces of each edge. We proceed by broadcasting the endpoints of each line in a segment group to all other lines in the segment group through a series of scans. By *broadcast* we mean the process of transmitting a constant value from a single processor to all other processors in the same segment group via a scan operation (i.e., the vector $[a_0, a_0, \ldots, a_0]$). Locally, each line processor maintains the minimal and maximal angles formed at each endpoint as well as the identities of the corresponding lines. Once the broadcasts are done, each line processor locally assigns an initial polygon identifier for the bordering polygon on the left and right side (moving from source to destination endpoint).

In Fig. 6, the left polygon identifier for line segment z is selected from the minimum identifiers of the source endpoint minimal angle (w_R, where w is the line identifier and R denotes the right side of w), the destination endpoint maximal angle (y_R), and the line identifier itself (z_L). Similarly, for the right side polygon identifier, the minimum identifier among the source endpoint maximum angle (x_R), the destination endpoint minimal angle (v_R), and the line identifier (z_R) is selected.

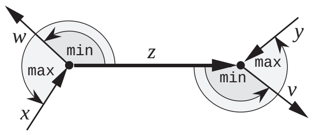

Figure 6: Selecting the initial polygon identifiers.

In Fig. 6, line z is assigned w_R as the initial left polygon identifier, and v_R as the right polygon identifier. Fig. 7 shows the initial polygon assignment for our example where the left and right polygon identifiers are contained in processor sets L_{ID} and R_{ID}, respectively. Since we are restricted to considering only lines that share the same R-tree node (e.g., in Fig. 5 in line processor set A, lines a, h compose the first segment group; lines b, e the second segment group; lines d, g the third segment group; and lines c, f, i the final segment group) when constructing the initial winged-edge representation, line i in Fig. 7 is assigned identifiers c_L and c_R rather than b_R and c_R as would be the case had line b also shared the same R-tree node.

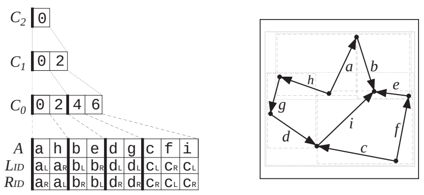
C2 0
C1 0 2
C0 0 2 4 6
A a h b e d g c f i
LID aL aR bL bR dL dL cL cR cL
RID aR aL bR bL dR dR cR cL cR
a
b
h
e
g
i
f
d
c

Figure 7: Initial polygon assignments.

Next, starting at the leaf level, merge all sibling lines together into the parent nodes. Mark all lines that intersect any of the overlapping regions formed by the bounding boxes of the nodes that have been merged for rebroadcasting among the lines in the merged nodes. This enables propagating the equivalence between the different identifiers in the merged nodes representing the same polygon (e.g., Fig. 8a where two R-tree nodes A and B are to be merged). In Fig. 8, A contains lines (a, c, g, h), and B contains lines (b, d, e, f). Lines (a, b, d) must be rebroadcast to the merged set of lines (i.e., lines (a, b, c, d, e, f, g, h)) as they intersect the overlapping region formed by the bounding boxes of A and B. This operation updates the winged-edge representations of any necessary lines (i.e., lines a and b in Fig. 8a). During the update, we note any polygon identifiers that must also be updated (e.g., line b has both its left and right polygon identifiers updated; b_L in Fig. 8a becomes a_L in Fig. 8b, and similarly, b_R becomes a_R). Neither of line a's polygon identifiers are updated because they are lexicographically minimal.

Broadcast the updates to all other lines in the merged node via scan operations (e.g., b_L to a_L and b_R to a_R in Fig. 8). Locally, if the transmitted polygon update matches either the left or right polygon identifiers of the local line, the local polygon identifier is updated to reflect the polygon identifiers that have been broadcast. In Fig. 8a, the right polygon identifier of line e is updated to show that polygon identifier b_R becomes a_R. Similarly, the left side polygon identifiers of lines d and f are updated to show that polygon identifiers b_L becomes a_L. Fig. 8b shows the resulting polygon identifiers and merged nodes. Continue the process up the entire R-tree until all lines are contained in a single node and all necessary broadcasts have been made. Fig. 9 is the final configuration of our example. The identifiers assigned to the three polygons are circled in the figure.

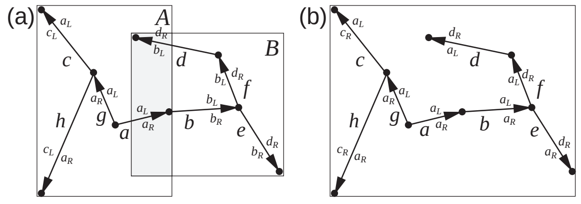

Figure 8: (a) Example of two nodes merging; and (b), the result of the merge operation.

The R-tree's spatial sort greatly limits the amount of inter-segment communication necessary as compared with a non-spatially sorted dataset where all lines would have to communicate their endpoints and polygon identifiers to all others. However, the non-disjoint decomposition of the R-tree causes more work in the local broadcasting phase of the sibling merge operation in comparison to an analogous disjoint decomposition spatial data structure such as the PMR quadtree and the R+-tree [4] because often many lines fall in the intersecting areas when the R-tree nodes are merged. Representations based on a disjoint decomposition of space mean that only those lines intersecting the splitting lines would need to be locally broadcast during the sibling merge operation.

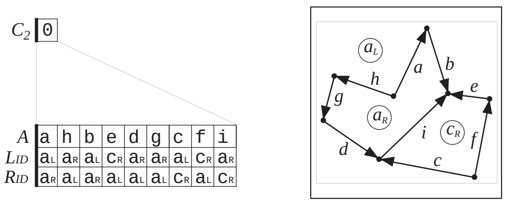
C2 0
A a h b e d g c f i
LID aL aR aL cR aR aR aL cR aR
RID aR aL aR aL aL aL cR aL cR
aL
aR
cR
a
b
c
d
e
f
g
h
i

Figure 9: Completion of the polygonization operation.

Now, let us estimate the number of broadcasts necessary during the polygon identification process due to the lines intersecting overlapping regions. Assume that each R-tree node has an average fanout of M. Let c (where $0 \leq c \leq 1$) be the fraction of the lines in each node intersecting one or more of the overlapping regions formed by bounding boxes of nodes that have been merged. Let h be the height of the R-tree (without loss of generality, $h = \log_M n$, n = number of lines in the tree). As $M^h = n$, it can be shown that the number of local broadcasts B that must be made during merging phases due to the intersection of lines with overlapping regions is $B = \sum_{i=2}^{h} cM^i \leq n(M/(M-1))$, which is $O(n)$.

bucket	build		polygonization	
size	time	scans	time	scans
25	105.6	1607	1042.1	92795
35	86.6	1315	997.7	81124
45	76.0	1147	1047.5	92496
55	73.0	1107	1046.6	92149
65	65.5	980	1741.8	115152
75	62.9	940	1695.9	111443
85	60.1	898	1479.4	104366
95	59.9	898	1625.6	108281

Table 1: Performance statistics for Montgomery Co., MD.

However, on the average it is expected to be lower. In particular, the average complexity of the line broadcasting step depends on the ability of the node splitting algorithm to partition the buckets as much as possible (thereby lowering the fraction c of lines intersecting overlapping regions).

CONCLUDING REMARKS

The data-parallel R-tree construction algorithm has been implemented on a Thinking Machines CM-2 with 16K processors and benchmarked against a sequential implementation on a Sun SPARCStation 1+. Data-parallel R-trees were observed to be approximately $10\times$ faster during the build operation than their sequential counterparts for large datasets (i.e., 250K line segments). Table 1 shows some of the results for the Montgomery County, MD dataset (90K line segments). Note that build times decrease with increasing bucket sizes; this is due to larger buckets requiring fewer subdivisions, and consequently fewer scan and sorting operations. Polygonization times behave differently, with larger bucket sizes implying that more buckets are being merged at each level. This results in more lines present in the intersecting bucket regions, thereby causing increased numbers of line rebroadcasts and winged-edge updates.

REFERENCES

[1] T. Bestul, *Parallel Paradigms and Practices for Spatial Data*, Comp. Sci. Dept., Univ. of Maryland, Ph.D. diss., CS-TR-2897, College Park, MD, 1992.

[2] G. E. Blelloch and J. J. Little, "Parallel Solutions to Geometric Problems on the Scan Model of Computation", *Proc. of the 17th Intl. Conf. on Parallel Processing*, 1988, St. Charles, IL, 218–222.

[3] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching", *Proceedings of the SIGMOD Conference*, June 1984, San Diego, 47–57.

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