A LEVEL LINKED R* TREE STRUCTURE WITH AN APPLICATION USING X-WINDOW GRAPHICAL INTERFACE

By

V C S REDDY KUMMETHA

Bachelor of Engineering

Osmania University

Hyderabad, India

1991

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 1993

OKLAHOMA STATE UNIVERSITY

A LEVEL LINKED R^* TREE STRUCTURE WITH

AN APPLICATION USING X-WINDOW

GRAPHICAL INTERFACE

Thesis Approved:

[illegible]

Thesis Adviser

[illegible]

Jacques LaFrance

[illegible]

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation and thanks to Dr. Huizhu Lu for her encouragement and advice throughout my thesis and course work. I would also like to thank Dr. Jacques Lafrance and Dr. Paul Benzamin for their valuable suggestions and for serving in my committee.

I wouldn't have been what I am today but for the love, guidance and support of my parents Mr. K.Ramakrishna Reddy and Mrs. Sita.

My appreciations to my sister Sujatha for her patience and guidance.

Heartfelt thanks to my friend Siva for being so supportive throughout my course work.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

II. LITERATURE REVIEW 3

R Tree 6
Insertion Algorithm 7
Pick Seeds Algorithm 8
Distribute Algorithm 9
Pick Next Algorithm 9
R^* Tree 10
Algorithm Choose Subtree 11
Algorithm Split 12
Algorithm Choose Split Axis 12
Algorithm Choose Split Index 13
X-Windows 14

III. R TREE BASED STRUCTURES: DRAWBACKS AND SOLUTIONS . . . 16

Inherent Drawback 16
Possible Solution 17
Implementation of Level Linked R^* Tree Structure 17
Leaf Level Linked R^* Tree 18
Algorithm Split 19
Costs involved in Leaf Level Linked R^* Tree 21
Application of Leaf Level Linked R^* Tree 22
Fully Level Linked R^* Tree 22
Application of Fully Level Linked R^* Tree 24
Advantages of Level Linked R^* Tree 24

IV. APPLICATION OF LEAF LEVEL LINKED R^* TREE
USING X-WINDOWS 26

Description of Data 27
Analysis of Data 27

Chapter | Page

X-Windows 30
Xlib 31
XtIntrinsic 32
OSF/MOTIF 32
Interface and Application 32

V. CONCLUSIONS 47

REFERENCES 48

LIST OF FIGURES

Figure Page

1. Example Structure of R* Tree 16

2. Level Linked R* Tree Structure 17

3. Root Node before Split in R* Tree 19

4. Leaf Level Linked R* Tree after Root Split 20

5. Leaf Level Linked R* Tree before Leaf Split 20

6. Leaf Level Linked R* Tree after Leaf Split 21

7. Fully Level Linked R* Tree before Leaf Split 23

8. Fully Level Linked R* Tree after Node Split 23

9. Hierarchy of X-Windows 31

10. Main Window of Interface 33

11. List of Counties when Load Button is Selected 35

12. Display when Loading Data 36

13. Display when Data is Already Loaded 37

14. Display of Scrolled List of Soils 38

15. Distribution of Selected Attribute in Caddo County in Shrink Scale 40

16. Distribution of Selected Attribute in Caddo County in Enlarge Scale 41

17. Obtaining Coordinates of Selected Area When Attribute is Displayed 42

Chapter Page

18. Querying a Selected Area in Caddo County 44

19. Display When Quit is Selected 45

LIST OF TABLES

Table		Page
1.	Development in Spatial Structure	5
2.	Summary of Soils in Caddo County	28

CHAPTER I

INTRODUCTION

Space is thought of as a collection of an infinite number of dimensionless points which form a continuum. Each point has a set of attributes that describes its properties. The collection of attribute values of spatial points is called spatial data.

Here we discuss two-dimensional spatial data, specifically land. Every spatial point has a location that is fixed with respect to the earth and addressed by a coordinate system. For example, a point can be addressed by its distances on North, South, East, and West directions from a fixed origin. Each point on the land has attribute values. These attributes include mineral contents, type of soil, chemical compositions, etc . . . By 'spatial locality', adjacent spatial objects tend to have nearly same attribute values. In other words, although they distribute irregularly, an attribute value spreads for a certain contiguous region. Thus space (land in this case) can be partitioned into small rectangular regions and attributes pertaining to these regions can be closely represented.

A set of spatial data is huge. Research is being done to develop data structures that can efficiently store and access spatial data. Currently, data structures like R-trees, R^+ trees and R^* trees are efficient for handling spatial data.

Unlike the B trees [Knu73] that store an alphabetic key or a numeric value in its node, R^* trees hold coordinates of spatial object and their attributes. R* tree's ability to handle spatial data is remarkable in that it provides very efficient access time when compared with other data structures.

Apart from storing and accessing, the other major factor is pictorial representation of this huge spatial data. Pictorial representation of spatial data gives a clearer image than just looking at the numbers. We see the way we imagine, therefore spatial data is more meaningful if shown on screen, in pictorial form, opposed to a printout of numbers.

X-Windows are flexible and have powerful graphical capabilities. A blend of X-Windows and R* tree provides a powerful interface for the spatial data.

CHAPTER II

LITERATURE REVIEW

Currently, 80% of information held by business and government is geographically referenced[Carl92]. Examples include land use information, mailing addresses, facility layouts, information about networks like water, cable, gas, electric, transportation etc . . The conventional method to preserve geographically referenced information is to draw maps on paper. Paper maps have several draw backs. They are difficult to update, manipulate, and combine with other data. Computerization of maps overcome these draw backs and supports many other features.

Currently available database systems are suitable for business applications like planning and accounting. In database applications involving geographical data, geographical objects, data structures that are capable of handling multi-dimensional data objects are required. Traditional data structures that support one dimensional data are not suitable for multi-dimensional data objects, for example, B+ trees. B+ tree stores one dimensional data keys like numbers (ex. age), or alphabetic keys (ex. name) or alphanumeric keys (ex. addresses). These keys have a fixed relation between them, i.e. we can say that a particular key is greater than, equal to, or less than another key. But spatial data key, in the simplest form, a point, has at least two coordinates (x, y), and in complex form, a polygon in multiple dimensions, has many more coordinates to represent it. These keys don't have a fixed relation between them, in the sense that we cannot say a particular spatial data key is less than, equal to, or greater than another key. Therefore new data structures are proposed to handle spatial data.

Early spatial data structures that are proposed are Quad tree [FB74] and K-D tree [Bent75] in 1974 and 1975. The basic structure of these data structures is similar to the binary tree. Developments over these data structures are Cell method and K-D-B tree [Robi81]. These spatial data structures are similar to traditional structures but modifications are made to handle multi dimensional data and proper algorithms are developed for insert and search. These data structures are not general enough and cannot handle both point and range query efficiently. These are expensive and are not appropriate for large databases.

Later developments are based on B+ tree and hashing principle (bucket method). In 1984, R-tree [Gutt84] and Grid file [NH84] structures are proposed. Grid file is based on bucket method, and R-trees are based on B+ tree structure. Grid file is good for handling point data. It can handle non point data by mapping to a higher level. R-trees are popular methods for accessing rectangles. This can also handle point data since a point is nothing but a rectangle having zero area.

After 1984, developments made, were, either to improve Grid file or R-tree. Bang file [FS87], GGF (generalized grid file) are examples of the development of grid files. R+ trees [SRF87], Greene's variation of R-tree and R* tree [NHRB90] are improvements done on R-tree.

The development of spatial data structures can be summarized as follows:

A tabular column is drawn (TABLE 1) that shows the year in which the structure is published, the name of the structure, and the traditional structure on which it is based on.

TABLE 1

DEVELOPMENT IN SPATIAL STRUCTURES

YEAR	SPATIAL STRUCTURE	TRADITIONAL STRUCTURE
1974	QUAD TREE	BINARY TREE
1975	K-D TREE	"
1979	CELL METHOD	BUCKET METHOD
1981	K-D-B TREE	B TREE
1984	R-TREE	B+ TREE
1984	GRID FILE	BUCKET METHOD
1987	R+ TREE	B+ TREE
1987	BANG FILE	BUCKET METHOD
1990	R* TREE	B+ TREE
1990	GGF	BUCKET METHOD

Note: The details in the tabular column are based on our literature survey. The spatial structures shown are the prominent structures that are developed during the period 1975 - 1990. This is not a complete listing.

Spatial data structures can be broadly divided into two groups based on the structures that efficiently handle

1) point data, and

2) range data.

Grid files can efficiently handle point query where as R-tree variants can handle range query efficiently. Point and range query are defined as follows:

1. A point query queries the database and gives the attributes associated with the given point.
2. A range query queries the database and gives all the attributes that are associated with the points within the given range.

In applications involving geological sciences, agriculture etc . . ., the attributes follow spatial locality. For example a particular soil in land may spread for a particular region and then fade away, from where on another soil may start. So land, in this case, can be divided into small rectangular regions consisting of a uniform attribute. For this type of applications R-tree variants are best suited. R-tree [Gutt84] is explained in detail.

R TREE

An R tree is a B+ tree like structure that stores multidimensional objects (rectangles). A non leaf node contains minimum bounding rectangles and pointers to its child nodes. A minimum bounding rectangle of a node is one that has minimum area and includes all rectangles that are the entries in its child node. Leaf node contains rectangles and the data object corresponding to that rectangle.

If 'M' is the maximum number of rectangles that fit in a node and 'm' is the minimum number of rectangles ($2 <= m <= M$) then R tree has the following properties:

1. The root has at least two children unless it is a leaf.

2. Every non-leaf node has between m and M children unless it is a root.
3. Every leaf node contains between m and M entries unless it is the root.
4. All leaves appear on the same level.

An R tree is completely dynamic. Insertions and deletions can be intermixed with queries and no periodic global reorganization is required. Since R tree is a dynamic structure, all the approaches of optimizing the retrieval performance must be applied during the insertion of new data rectangle. The insertion algorithm calls two more algorithms in which crucial decisions for good retrieval performance are made. To insert a rectangle, the leaf node 'N' into which it is to be inserted must be determined. This is done by the following algorithm.

Insertion Algorithm [Gutt84]:

```
CS1  Set N to be the root
CS2  If N is leaf
               return leaf
     Else
               Choose the entry in N whose rectangle needs least area
               enlargement to include the new data. Resolve ties by choosing
               entry with rectangle of smallest area.
     End
```

CS3 Set N to be the child node pointed by the child pointer of the chosen entry and repeat from cs2.

The insert function then determines if the leaf node can accommodate a new rectangle. If the node already has M rectangles, then it calls split function. Split algorithm in turn calls two more algorithms

1) pick seeds: determines the first two rectangles that go into each group.

2) distribute: this distributes the remaining entries into the two groups.

The algorithm for splitting M+1 entries into two groups is as follows [Gutt84]:

```
S1    Invoke pick seeds to choose two entries to be the first entries of the
      groups.
S2    Repeat
      Distribute entry
   Until
      all entries are distributed or
      one of the two groups has M-m+1 entries.
S3    If entries remain, assign them to the other group such that it has minimum
      number m.
```

Pick seeds Algorithm [Gutt84]:

```
PS1   For each pair of entries E1 and E2 compose a rectangle R including
      E1.Rectangle and E2.Rectangle.
      Calculate d = area (R) - area (E1.Rectangle) - area (E2.Rectangle)
```

PS2 Choose the pair with largest d.

Distribution Algorithm [Gutt84]:

This algorithm invokes another algorithm pick next, to determine the next entry considered for distribution. Then it selects the group where the entry is placed.

DE1 Invoke pick next to chose next entry to be assigned.

DE2 Add it to the group whose covering rectangle will have to be enlarged least to accommodate it. Resolve ties by adding the entry to the group with smallest area, then to the one with few entries then to either.

Pick Next Algorithm [Gutt84]:

PN1 For each entry E not yet in a group, calculate d1, the area increase required in the covering rectangle of the group1 to include the E.Rectangle.Similarly calculate d2 for group2.

PN2 Choose the entry with maximum difference between d1 and d2.

The method of optimization is to minimize the area covered by a directory rectangle. This reduces overlap of rectangles and good retrieval performance is obtained. The algorithm pick seeds finds two rectangles that waste the largest area put in one group. The two selected rectangles will be the distant ones. 'Distribute entries' algorithm assigns the remaining entries based on the minimum area criteria. Pick next chooses an entry with the best goodness value in every situation. In R-trees, bounding boxes are formed from arbitrary set of rectangles in a way that arbitrary retrieval operations with query rectangles of arbitrary size are handled efficiently. The known parameters of retrieval performance affect each other in a very complex manner such that it is impossible to optimize one without influencing the other. This can cause deterioration in the overall performance. Since the data rectangles may have different size and shape and directory rectangles may grow and shrink dynamically, the success of methods that optimize one parameter is very uncertain. In R^* trees [NHRB90] a heuristic approach is applied taking various parameters into consideration.

R* TREE

The parameters that are taken into consideration by the *R* tree* are[NHRB90]:

1. Minimize the area covered by a directory rectangle. The dead space in the directory rectangle not covered by any of its child rectangles is minimized.
2. Minimize the overlap between directory rectangles that decreases the number of paths to be traversed.
3. Minimize the margin of the directory rectangle. Margin is the sum of the lengths of the edges of a rectangle. For a fixed area square has the minimum margin. Thus minimizing the margin yields more quadratically shaped directory rectangles. This results in more packed directory rectangles. Queries with large query rectangles profit this.
4. Optimize storage utilization. The higher the storage utilization, the lower the tree height and the better the querying.

Unlike R tree that take only area parameter into consideration, *R^* tree* takes area, margin and overlap. The overlap of an entry is defined as [NHRB90]:

Let E1, . . ., Ep be the entries in the current node. Then

Overlap (Ek) = Σ area (Ek.Rectangle $\cap$ Ei.Rectangle), $1 \leq k \leq p$.

To insert a new rectangle *choose subtree* function is invoked to find an appropriate node.

Algorithm Choose Subtree [NHRB90]:

```
CS1   Set N to be the root
CS2   If N is a leaf, Return N
      Else If the child pointers in N point to leaves [determine the minimum
            overlap cost], choose the entry in N whose rectangle needs least
            overlap enlargement to include the new data rectangle. Resolve
            ties by choosing the entry whose rectangle needs least area
            enlargement then the rectangle with smallest area.
      End
CS3   Set N to be the child node pointed to by the child pointer of the chosen
            entry and repeat from CS2.
```

From the above algorithm, the subtree is chosen and the node is selected in which the new entry is to be inserted. If the node has less than M entries, the new entry is inserted in that node. If it has M entries algorithm *split* is invoked. Algorithm *split* in turn calls two more algorithms:

1. Choose split axis: This chooses the axes along which the split has to be performed by computing various goodness values.
2. Choose split index: This selects the distribution of entries into two groups.

Along each axis the entries are first sorted by the lower value, then by the upper value of the rectangles. For each sort M - 2m + 2 distributions of the M + 1 entries into two groups are determined where the k^{th} distribution [k = 1, . . ., (M-2m+2)] has first [m-1+k] entries in the first group and the remaining in the second group. For each distribution the following goodness values are determined:

(1)	area-value:	area[bb(first group)] + area[bb(second group)]
(2)	margin-value:	margin[bb(first group)] + margin[bb(second group)]
(3)	overlap-value:	area[bb(first group)] $\cap$ area[bb(second group)]

Where bb represents the bounding rectangle.

Algorithm Split [NHRB90]:

S1 Invoke choose split axis to determine the axis, perpendicular to which the split is performed.

S2 Invoke choose split index to determine the best distribution into two groups along that axis.

S3 Distribute the entries into two groups.

Algorithm Choose Split Axis [NHRB90]:

CSA1 For each axis sort the entries by the lower and by the upper values of their rectangles and determine all distributions as described above. Compute S, the sum of all margin values of the different distributions.

End

CSA2 Choose the axis with the minimum S as split axis.

Algorithm Choose Split Index [NHRB90]:

CSI1 Along the chosen split axis choose the distribution with the minimum overlap value. Resolve ties by choosing the distribution with minimum area value.

Once the split is performed the tree structure has to be updated along the insertion path. All the covering rectangles have to be adjusted such that they are the minimum bounding boxes enclosing their children.

Though the insertion is costly, it provides an order in the structure that contributes for fast accessing.

Experiments conducted found that R^* tree outperforms the R tree variants in all experiments[NHRB90]. The conclusions of the experiments are [NHRB90]:

1. The R^* tree is the most robust method, underligned by the fact that for every query less accesses are required than by any other variants.

2. The gain in efficiency of the R^* tree for small query rectangles is higher than for large rectangles because storage utilization gets more important for large query rectangles. This emphasizes the goodness of the order preservation of the R* tree.

3. The maximum performance gain of the R^* tree taken over all query and data files is in comparison to the linear R tree about 400% and quadratic R tree is 180%.

4. R^* tree has the best storage utilization.

X-Windows

To represent these data structures graphically, we need a sound graphical interface and user friendly environment. X-Windows(X), which runs under UNIX environment, provides an excellent interface as it does not restrict the window to any pre-defined interface like most window systems do. We can define and design the window as we desire.

X-Windows provide mechanisms to support many interface styles rather than enforcing one particular policy, i.e. it does not provide any scroll bars, button boxes, menu, etc . . ., by default. All it provides is a rectangular section of screen. Applications can create their own decorations like title bar, scroll bar, menus, etc . . ., in the required fashion and style. X has a variety of resources like windows, bitmaps, fonts, colors and data structures used by applications.

X has a standard toolkit known as X toolkit. The X toolkit has two parts: Xt Intrinsic and widget. Widget sets provide user interface components like windows, scroll bars, title bars, dialog boxes etc . . . Xt Intrinsic supports many widget sets. The popular widget set is Open Software Foundation (OSF) Motif Widget set. Both widget and Xt Intrinsic are built on top of Xlib. Xlib the basic low level X library provides with various functions and capabilities. Xlib also provides complete access and control over the display, windows and input devices.

Programmers can directly program in Xlib, however it requires lot of redundant code to maintain the conventions. The work done by 100 lines Xlib code can efficiently be done by five to ten lines of code using widget. Efficient X programs are written with a combination of Xlib, Xt Intrinsic, and a widget set.

CHAPTER III

R TREE BASED STRUCTURES: DRAWBACKS AND SOLUTIONS

INHERENT DRAWBACK

R-tree structure is analogous to B+ tree structure. In R-tree structure index rectangles are stored in non leaf nodes and actual rectangles are stored in leaf nodes. The structure of R-tree is shown below:

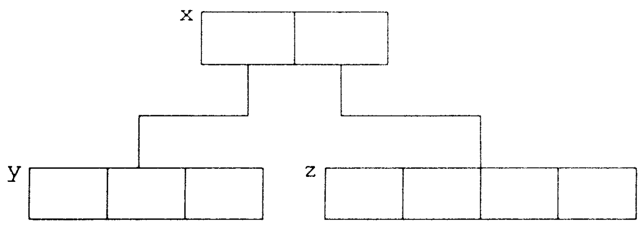


Figure 1. Example Structure of R^* Tree

This structure (R-tree based spatial structure) is efficient for point and range querying, but does not provide any means to access the actual data directly. If an application requires access to actual data sequentially, then no matter what, the whole tree has to be traversed to achieve this. This is costly and time consuming. In figure 1 from node x we need to go to node y access all data then go back to node x and come down to node z to access other data. This is very costly and time consuming.

POSSIBLE SOLUTION

The problem encountered in R-tree based structures can be solved by making it leaf level linked structure. The leaf level linked R* tree structure is as show below:

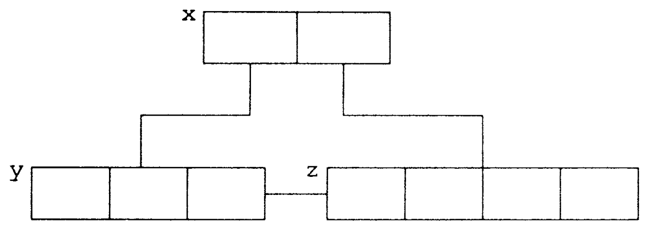


Figure 2. Level Linked R^* Tree Structure

The level linked R* data structure has all the properties of a R* data structure. In addition it has all the nodes in the same level linked. Querying, either point or range is done similar to that of the R* tree. Sequential access to actual data is done by utilizing the level linked list of the leaf level. The level linking of the non leaf nodes can be utilized to implement hotspots principle that helps in improving the query algorithm.

Implementation of level linked R^* structure

Implementation of the level linked R* tree depends on the requirements of the application. Some applications may require only level linking of the leaf level while others may require level linking of the whole tree. Though both types are discussed, since the application developed in this thesis requires leaf level linked R* tree structure, it is discussed in more detail.

Leaf level linked R* trees:

This type of implementation is particularly needed for applications involving graphical interface for spatial data. These applications can access the actual data sequentially, using the leaf level, simultaneously exploit the efficient query capabilities of R* tree.

In R* tree new nodes are created only in two cases:

1. When a node is over filled, i.e if the node already contains M (maximum number) elements and a new element has to be inserted into that node, 'split' occurs. Whenever split occurs, a new node is created and is linked to its parent. This created node forms a new sibling to the old node. The newly created node can be in the leaf level or non leaf level.

2. When ever the root node splits two new nodes are created, one forming the new root and the other forming a new sibling for the old root that split.

Always there is only one node in the root level. So when a new root is created we need not worry about linking it to any other siblings in the root level. For leaf level linked R* trees the only case that has to be considered is, to link the newly created leaf node to the list of the leaf level. R* trees are similar to B+ tree structure and have all the leaf nodes in the same level. So the newly created leaf node should be in the same level as that of the leaf node that has been split. Whenever a leaf node splits, by assigning the new node's sibling pointer to the old node's sibling pointer and the old node's sibling pointer to the new node, level linking of the leaf level can be achieved. Thus, only the split algorithm in the insertion algorithm of R* trees has to be modified. The modified split algorithm for leaf level linked R* trees is as follows:

Algorithm Split:

```
S1    Invoke choose split axis to determine the axis, perpendicular to which
      the split is performed.
S2    Invoke choose split index to determine the best distribution into two
      groups along that axis.
S3    Distribute the entries into two groups.
S4    If the split node is a leaf node update the links of the nodes. Set new
      nodes NEXT (pointer to next node in the same level) to old nodes
      NEXT and old nodes NEXT to new node.
```

The working of the leaf level linked algorithm is shown in the following examples:

An external pointer is set to the first node created, since this is a leaf node itself and is first leaf node created. As the tree grows the leaf level linked list is created and can be traversed using this pointer.

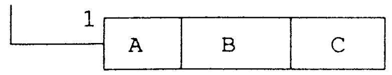


Figure 3. Root Node Before Split in R* Tree

Let M (maximum elements in a node) = 3

Let m (minimum elements in a node) = 2

Initially the root node1 shown in figure 3 is full. If a new element D has to be inserted the node 1 should split. Since the root node is being split, two new nodes are created one forming new root and the other forming new sibling as shown in figure 4.

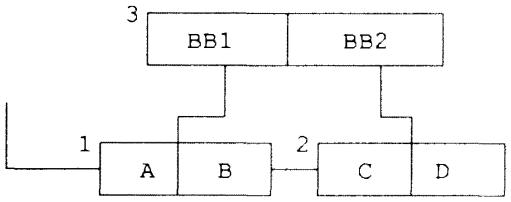


Figure 4. Leaf Level Linked R* Tree After Root Split

The new nodes 2, 3 are created, 3 forming the new root and 2 forming the new sibling. 1, 2 are linked. BB1 and BB2 are the bounding boxes of the nodes 1, 2.

Another scenario is depicted below:

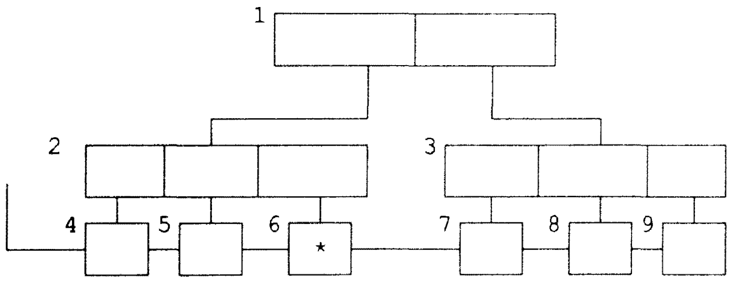


Figure 5. Leaf Level Linked R* Tree before Leaf Split

In the figure 5 if node 6 is overfilled and a new element is to be placed into it, split occurs and propagates as shown in figure 6. Node 6 is split and node 10 is created. According to the algorithm, sibling pointer of node 10 is set to the node pointed by sibling pointer of node6, i.e. node7, and sibling pointer of node6 is set to new node (node10). The level linking of the leaf level is perfectly maintained and is shown below:

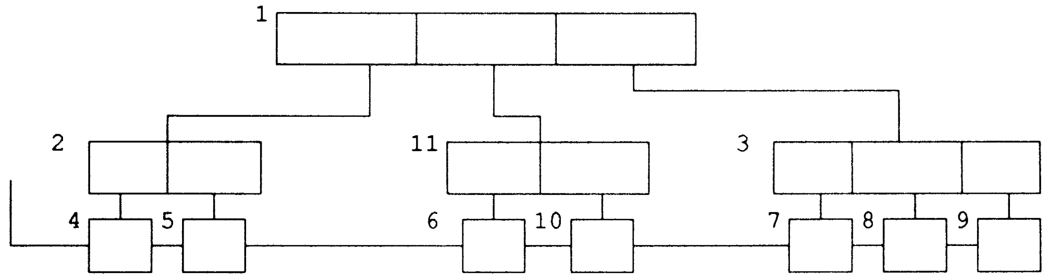


Figure 6. Leaf Level Linked R* Tree After Leaf Split

The non leaf node 2 also splits and new node 11 is formed, propagating a new bounding box to node 1.

Costs involved in implementing leaf level linked R trees:*

1. The access time to access the actual data is greatly reduced. In R* trees the whole tree has to be traversed to achieve this. In leaf level linked R* trees only the leaf level is accessed to achieve this. The improvement in the access time of the actual data is clear and is discussed below.

Let there be N > 1 nodes in the tree.

In R* tree, to access the actual data sequentially, we need to traverse all the N nodes. If it takes one time unit to traverse from one node to other, then it requires a value greater than N to access all the data in leaf nodes. In leaf level linked trees we traverse only the leaf nodes which are a subset of total nodes N. So time taken to access the actual data by leaf level linked trees is less than N units. Therefore time taken to access the actual data is clearly better in leaf level linked trees.

2. In leaf level linked R* trees querying is done using the basic R* tree structure, so there is no additional cost for querying.

3. One additional pointer is needed for leaf level linked R* tree structure. So additional memory requirement is size of a pointer per node.

4. Whenever a split occurs only two pointer assignments are done for level linking. This is small when compared with the time complexity for R* tree insertion and can be neglected. So we can say that there is no or negligent increase in the time complexity for insertion in leaf level linked R* tree.

Application of leaf level linked R trees:*

Applications that require graphical representation of the spatial data as well as querying can use a leaf level linked R* tree structure. The practical application of this is done in this thesis for soils in Caddo county, Oklahoma, explained in chapter IV.

Fully level linked R* trees:

These structures are similar to leaf level linked structures. A brief outline of these structures is discussed here.

As explained in the previous section new nodes are created only when split occurs, two nodes are created when root node splits, and we need not worry about the newly formed root node. The only other case to consider is, to link the newly created node into the level, of the split node. So whenever a node splits, by assigning the new node's sibling pointer to the old node's sibling pointer and the old node's sibling pointer to the new node level linking level can be achieved.

The working of the fully level linked split algorithm is similar to that of the leaf level linked split algorithm and is shown in the following examples:

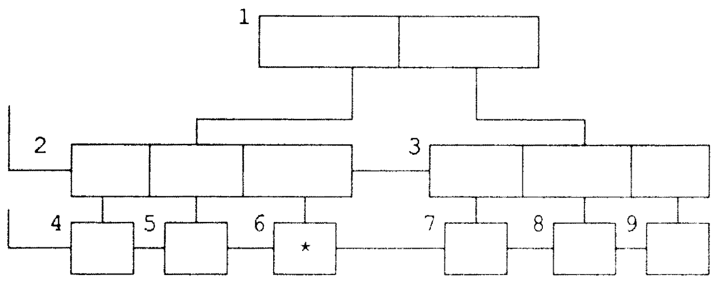


Figure 7. Fully Level Linked R* Tree before Leaf Split

In the above figure if node 6 is overfilled and a new element is to be placed into it, split occurs and propagates. Node 6 is split and node 10 is created as shown in figure 8. According to the algorithm, sibling pointer of node10 is set to node pointed by sibling pointer of node6, i.e. node7, and sibling pointer of node6 is set to new node (node10). The level linking of the leaf level is perfectly maintained. Now the bounding rectangle is propagated to parent (node 2) in the immediate upper level. Since node 2 is also over filled, it also splits creating a new node 11. Again links are manipulated as explained above and the resulting tree is shown below:

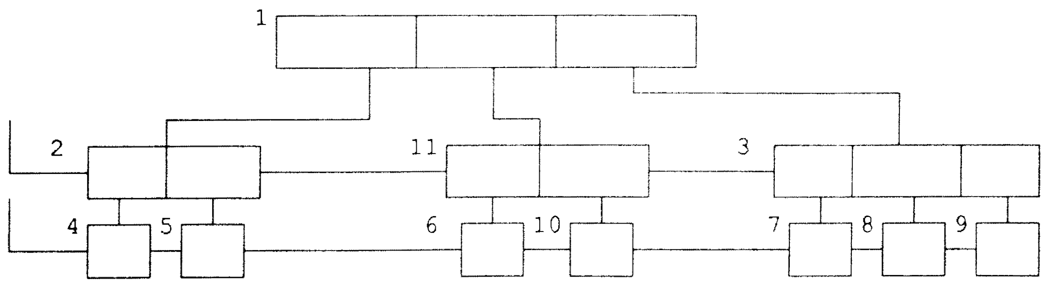


Figure 8. Fully Level Linked R* Tree after Node Split

Links are properly maintained and tree is completly linked. By having an external pointer to each level we can traverse each level individually if needed. Cost involed is similar to that of the leaf level linked tree.

Application of fully level linked trees:

Fully level linked trees can be used to implement the concept of hotspots. If an element is accessed, the chances for accessing neighboring elements are more. By having a pointer to the current node we can traverse neighbournig nodes and access data quickly. This algorithm depends on the data and type of query done. To implement this we need to have links to all neighboring nodes. R* tree structure provides links to children and parent. In addition to this we need to have two sibling pointers, to communicate and possibly traverse to sibling nodes. Also the links in the level linked list should be adjusted corresponding to the positions of the modified elements in parent node. This may lead to additional costs.

Advantages of Level Linked R* tree

1. Provides an easier way to access the actual data sequentially.
2. Allows implementation of the hotspot concept to achieve better accessing.
3. Can be implemented with almost same cost as of R* trees.
4. Supports both point and range query efficiently.

The explanation for the above points is given below.

1. Accessing the actual data: In R* trees actual data is stored in the leaf nodes.

So to achieve easy accessing of the actual data only level linking of the leaf level is enough. By having a pointer to the first node in the leaf level we can traverse the whole list and access all the actual data.

2. Provision to implement hotspots: In addition to child pointers and parent pointer, level linked R* tree has pointers to siblings. This means a node can communicate with its neighboring node easily. In other words we can move to any neighboring node with just one pointer movement. This property can be exploited to implement hotspots.

3. Low cost: To implement level linked R* trees, only two pointer assignments are required that results in negligible additional cost over the insertion of R* trees.

4. Point and Range Query: Since the basic structure is similar to R* tree, the efficient accessing algorithms of R* trees can be used without any modifications. Therefore point and range query can be handled efficiently.

CHAPTER IV

APPLICATION OF LEAF LEVEL LINKED R* TREE USING X-WINDOW INTERFACE

The main advantage of these structures is to access the actual data with minimum cost. Also their structure supports the efficient querying provided by the R* tree. An application that needs this kind of structure is the one that requires graphical representation of the spatial data as well as querying. The practical application that we are going to deal with, is the graphical representation and querying of soils in Caddo county, Oklahoma.

For graphical representation of this spatial data we need to access the leaf nodes because the actual data is in leaf nodes. Leaf level linked R* tree is used because this application efficiently exploits both the leaf level linking for graphical representation and basic R* tree structure for querying.

The application can be divided into two parts.

1. Implement the leaf level linked R* tree
2. Provide user friendly interface to represent the spatial data graphically and to allow querying.

The leaf level linked structure is implemented as explained in the CHAPTER III. To represent the data graphically, we need some sort of interface. For interface X-

Windows are used because they are flexible, have strong graphical capabilities and provide user friendly components like push buttons, menus etc . . . This interface sits on the leaf level linked R* tree. Though query request may be done through interface the actual querying is done by the level linked R* tree.

Description of data:

Caddo County is one of the counties present in the state of Oklahoma. It spreads to an area of four hectors. This area is divided into small rectangles, each of 200 x 200 meters, and the most prominent attribute pertaining to this area is determined. This data is obtained from Dr. Mark Gregory, Department of Agriculture.

Analysis of the data:

The obtained data is analyzed and then reformed into two files. The first file has the coordinates of the rectangular area with an attribute code. The format of this file is as follows:

X1 Y1 X2 Y2 ATT

Where X1, Y1 are the lower left and X2, Y2 are upper right corners of the rectangle. ATT denotes the attribute code that corresponds to this piece of area. For each attribute code there is a description in the second file. There are about 99 attribute codes. These codes and their description are listed in TABLE 2. The data is also analyzed based on the amount of area they spread. In the tabular column shown in TABLE 2, column 1 represents code, column 2 shows the attribute, column 3 shows the

TABLE 2

SUMMARY OF SOILS IN CADDO COUNTY

Code	Attribute	Area
0	no data	10377
1	Acme-Gypsum outcrop complex, 2 to 8 percent slopes	439
2	Breaks	904
3	Cobb fine sandy loam, 1 to 3 percent slopes	438
4	Cobb fine sandy loam, 3 to 5 percent slopes	2226
5	Cobb fine sandy loam, 5 to 8 percent slopes	420
6	Cobb fine sandy loam, 3 to 8 percent slopes, eroded	1846
7	Cobb and Grant soils, 3 to 8 percent slopes, severely eroded	1664
8	Cyril fine sandy loam	276
9	Cyril fine sandy loam, noncalcareous variant	62
10	Darnell soils, 3 to 12 percent slopes, severely eroded	392
11	Darnell-Noble association, rolling	6766
12	Darnell-Noble association, hilly	1780
13	Dougherty loamy fine sand, 1 to 3 percent slopes	1176
14	Dougherty and Eufaula loamy fine sands, 3 to 8 % slopes	3087
15	Eufaula fine sand, rolling	1020
16	Eufaula loamy fine sand, 1 to 3 percent slopes	102
17	Eufaula loamy fine sand, hummocky	98
18	Foard silt loam, 0 to 1 percent slopes	717
19	Gracemont soils	1538
20	Grant loam, 1 to 3 percent slopes	1604
21	Grant loam, 3 to 5 percent slopes	2084
22	Grant loam, 3 to 6 percent slopes, eroded	566
23	Grant loam, 5 to 8 percent slopes	293
24	Grant-Wing complex, 1 to 5 percent slopes	263
25	Hollister silt loam, 0 to 1 percent slopes	746
26	Konawa loamy fine sand, 1 to 5 percent slopes, eroded	640
27	Konawa soils, 2 to 8 percent slopes, severely eroded	152
28	Limestone cobbly land	546
29	Lucien-Dill fine sandy loams, 3 to 12 percent slopes	3524
30	Lucien-Dill fine sandy loams, 12 to 30 percent slopes	379
31	McLain silty clay loam	497

TABLE 2 (CONTINUED)

32	Miller silty clay loam	143
33	Minco very fine sandy loam, 3 to 8 percent slopes	2891
34	Minco very fine sandy loam, steep	437
35	Minco silt loam, 3 to 5 percent slopes	1073
36	Noble fine sandy loam, 1 to 3 percent slopes	1335
37	Noble fine sandy loam, 3 to 8 percent slopes	7136
38	Norge silt loam, 1 to 3 percent slopes	1342
39	Norge silt loam, 3 to 5 percent slopes	1311
40	Pond Creek fine sandy loam, 0 to 1 percent slopes	963
41	Pond Creek fine sandy loam, 1 to 3 percent slopes	5186
42	Pond Creek silt loam, 0 to 1 percent slopes	1730
43	Pond Creek silt loam, 1 to 3 percent slopes	4239
44	Pond Creek silt loam, 1 to 3 percent slopes, eroded	180
45	Port silt loam	2715
46	Port and Pulaski soils, channeled	260
47	Pulaski soils	1392
48	Quinlan-Woodward complex, 5 to 12 percent slopes	2475
49	Reinach silt loam, upland, 0 to 1 percent slopes	88
50	Reinach silt loam, upland, 1 to 3 percent slopes	1824
51	Reinach silt loam, 0 to 1 percent slopes	1291
52	Rough broken land	976
53	Shellabarger fine sandy loam, 1 to 3 percent slopes	407
54	Shellabarger fine sandy loam, 3 to 5 percent slopes	268
55	Talpa-Rock outcrop complex, 5 to 30 percent slopes	1056
56	Tillman silty clay loam, 1 to 3 percent slopes	2440
57	Tillman silty clay loam, 3 to 5 percent slopes	935
58	Tillman silty clay loam, 2 to 5 percent slopes, eroded	199
59	Vernon soils, 5 to 12 percent slopes	118
60	Woodward-Quinlan complex, 3 to 5 percent slopes	972
61	Yahola soils	975
81	Fill	0
82	Borrow Pits	0
83	Gravel Pits	0
84	Mine Pits and Dumps	0

TABLE 2 (CONTINUED)

85	Oil-Waste Land	0
86	Pits	0
87	Pits, Quarries	0
88	Quarries	28
89	Slickspot	89
96	Water, Sand Channel	57
98	Water	712
99	Border	0

number of 200 x 200 meters rectangles each attribute is spread. This gives us an idea about the attributes.

X-Windows are used as an interface for this application. They are discussed in detail in the following section.

X-Windows:

X-Windows provide an efficient interface. An application can open as many windows as needed. An X-Window is a plain rectangular shaped window on the screen. There are different types of interface objects like scroll bars, push buttons etc . . . that are to be added to the window as desired. To create a window and add different decorations we need to program using some X-tools. The basic programming is done using X-lib. There are other tools like Xt intrinsic that are built on top of X-lib. OSF/Motif widget set is built on top of the Xt Intrinsic. These tools have functions and macro that does most of the work and are convenient to use. The hierarchy is shown below [Hel92]:

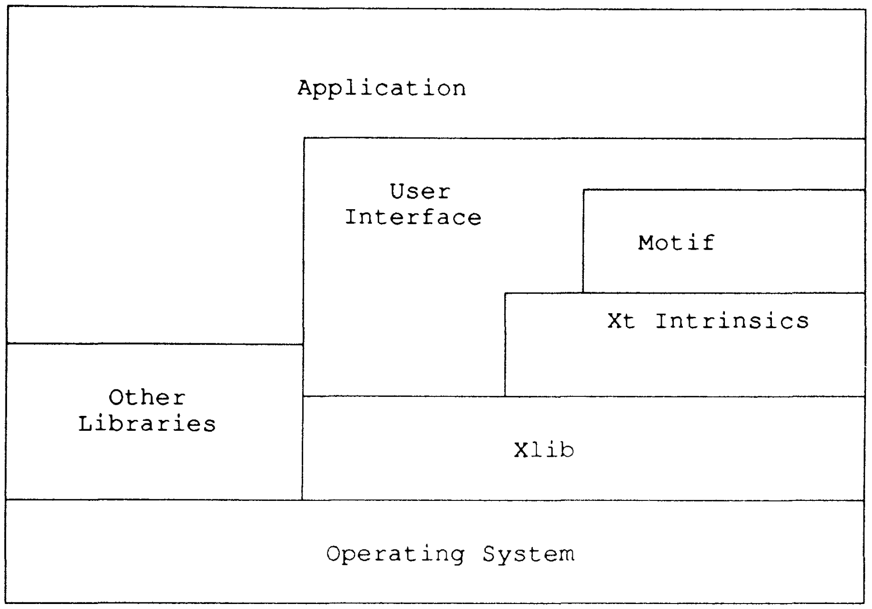


Figure 9. Hierarchy of X-Windows [Hel92]

Xlib:

Xlib is the X library and is the lowest level of programming interface to X. Xlib is a programming interface that has subroutine package written in C and is provided with the X Window system. Xlib is powerful enough to write effective applications without additional programming tools and is necessary for certain tasks even in applications written in higher level. The disadvantage of using Xlib is that it makes programming difficult. For example to create a new window, about 60 lines of similar code is written every time a new window is created. Also the all the conditions are written in one big main loop with case statements. This complicates programming a complex interface. This is explained in greater detail in [Oli89].

Xt Intrinsics:

Xt Intrinsic are built on top of Xlib. The purpose of Xt is to provide an object oriented layer that supports the user interface abstraction called a widget. A widget is a reusable, configurable piece of code that operates independently of the application except through prearranged interaction. Xt defines certain base classes of widget, whose behavior can be inherited and augmented or modified by other widget classes or its sub classes. For programming in Xt intrinsics [AT92] is referred.

OSF/Motif widget set (Xm):

Xt provides an object oriented framework for creating reusable, configurable user interface widget. Motif provides widget for such common user-interface elements as labels, pushbuttons, menus, scrollbars etc . . . In addition it provides manager widget that control the layout of the other widget. Xt provides functions for creating and setting resources on widget. Xm provides the widgets themselves plus an array of utility and convenience functions for creating groups of widget that are used collectively as a single type of user interface element. For programming using Motif [Hel92] is referred.

INTERFACE AND APPLICATION

The X-Window interface sits on the R* tree. The interface provides access to load data, display soils (attributes), and query the attributes. The main window has four push buttons, one for load, display soils, query soils, and quit. This is shown in the figure 10. When the LOAD button is pushed the names of the counties to be selected is

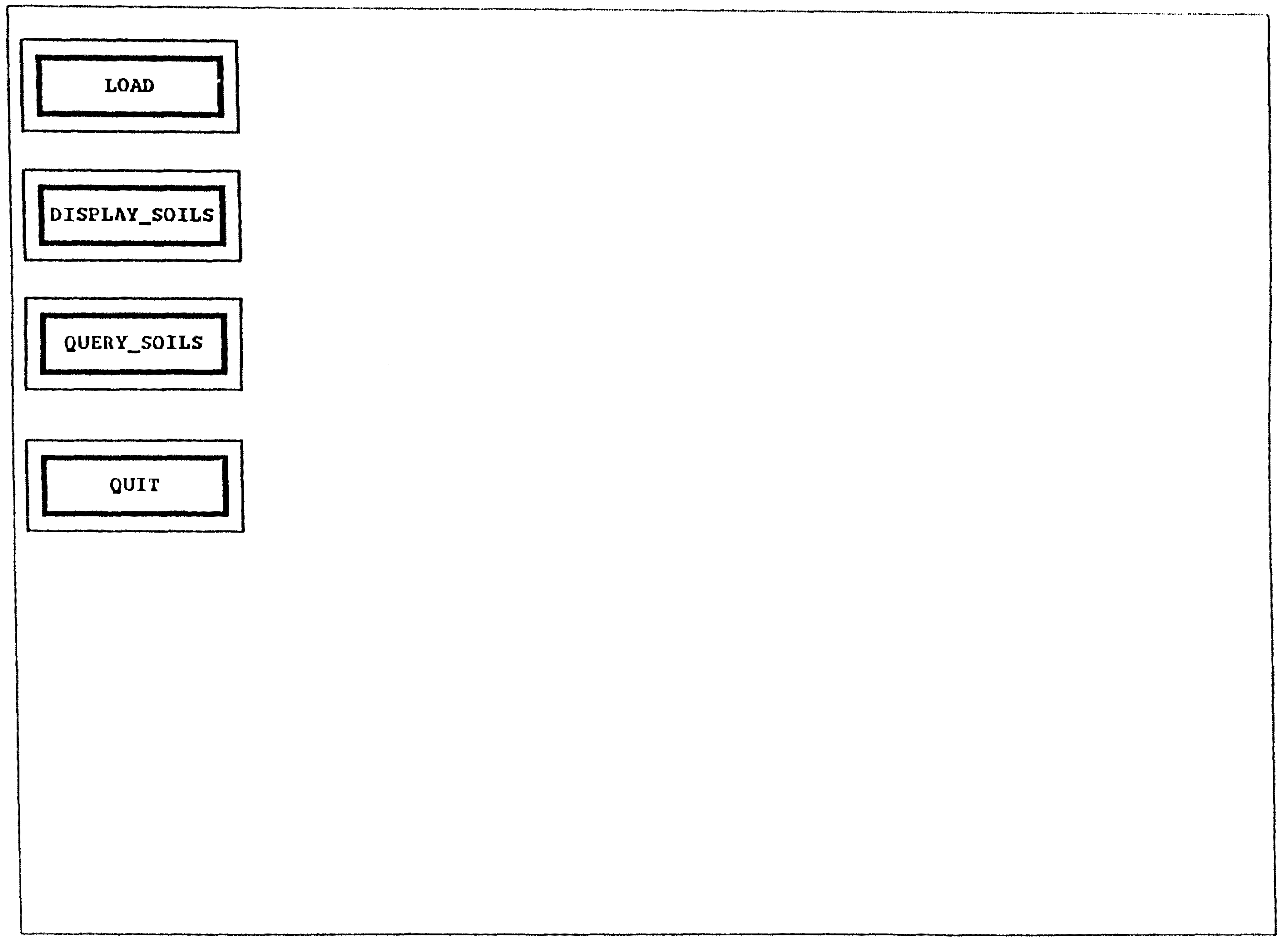


Figure 10. Main Window of Interface

displayed. In our case, since data of only one county is available the name of the Caddo county is displayed. This is shown in the figure 11. When this button (Caddo) is pushed it pops up a dialog box with a message informing whether the data will be loaded or data has already been loaded. This is shown in figures 12 and 13. Both of the dialog boxes are opened in SYSTEM_MODAL. This means that when they are in action they have to be responded. The mouse button will not work for other options until they are responded. Also the dialog boxes display different symbols, one with sand timer to represent it is busy loading and the other with information symbol. When the Load is selected by responding to 'OK' in the dialog box of figure 12, loading starts, with the scale widget displaying the amount of data displayed. Internally, the file containing the data of the Caddo county is opened and the insertion function is called to build the R* tree with the data. Since the data is huge (about 90,000 rectangles) it takes about 8 to 10 minutes to load the data of Caddo county. Once the data is loaded, the scaled widget, dialogue box, and Caddo button will disappear. Attempt to load the data again results in the second dialogue box (figure 13) that informs that data has already been loaded. Once loading is completed display of attributes and query of attributes can be selected.

If DISPLAY SOILS is selected a scrolled list appears as shown in figure 14. The scrolled list displays all the attributes. Since there are 99 attributes it is inconvenient and not possible to display all of them in a single window. So a scrolled window is used to display these attributes. The scrolled list shows 15 items at a time. By pressing the arrow buttons (up and down) provided on the right side of the list, the list can be

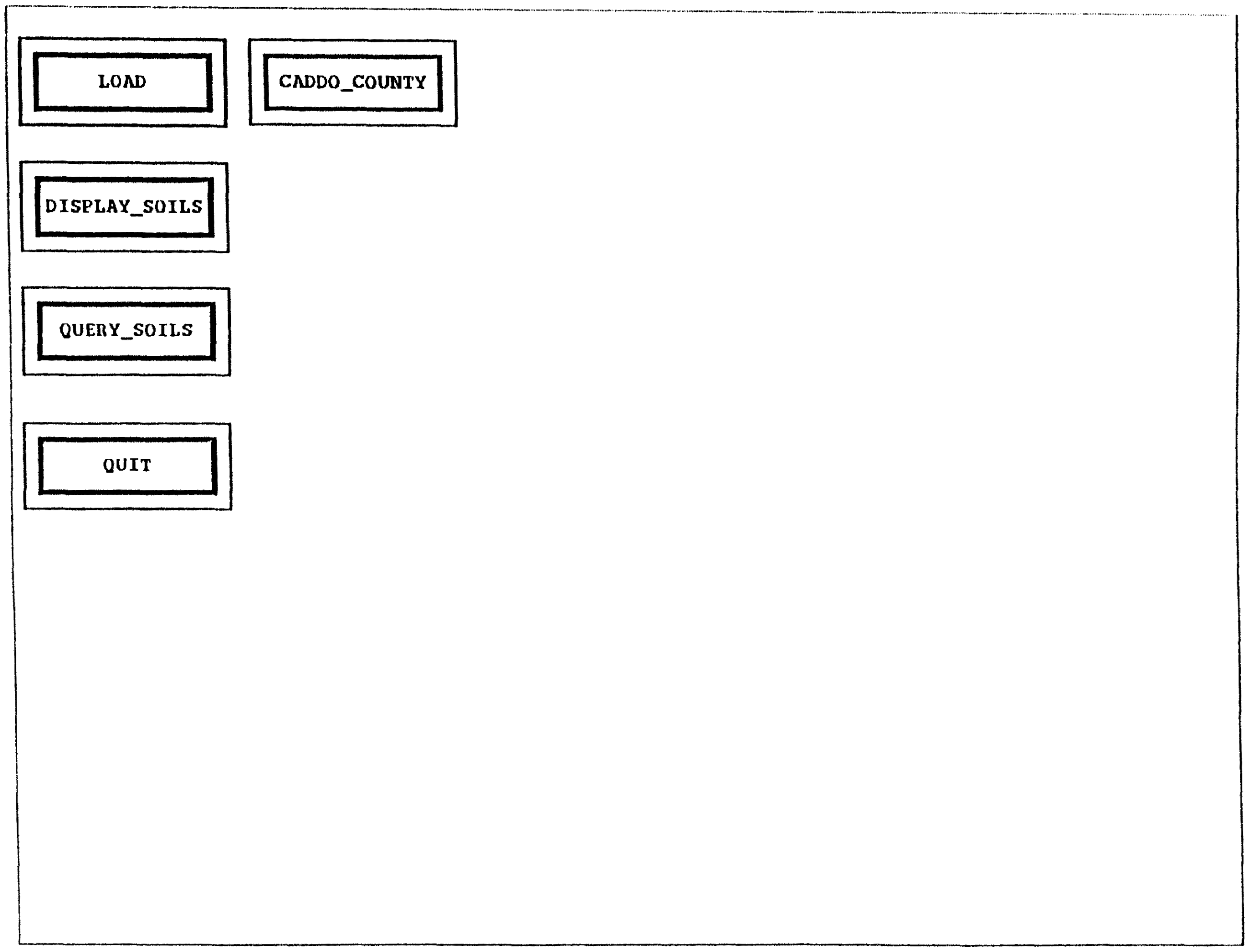
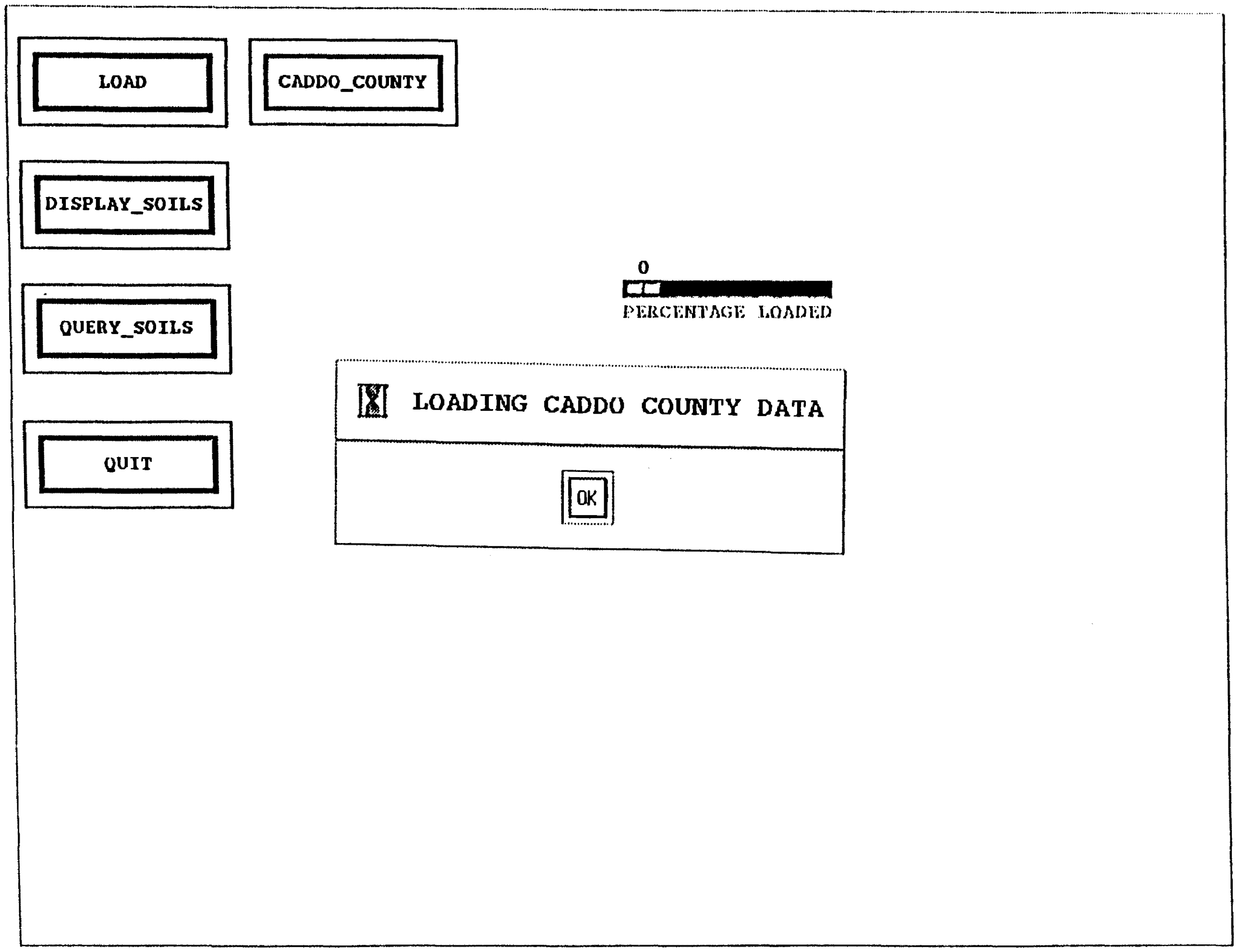
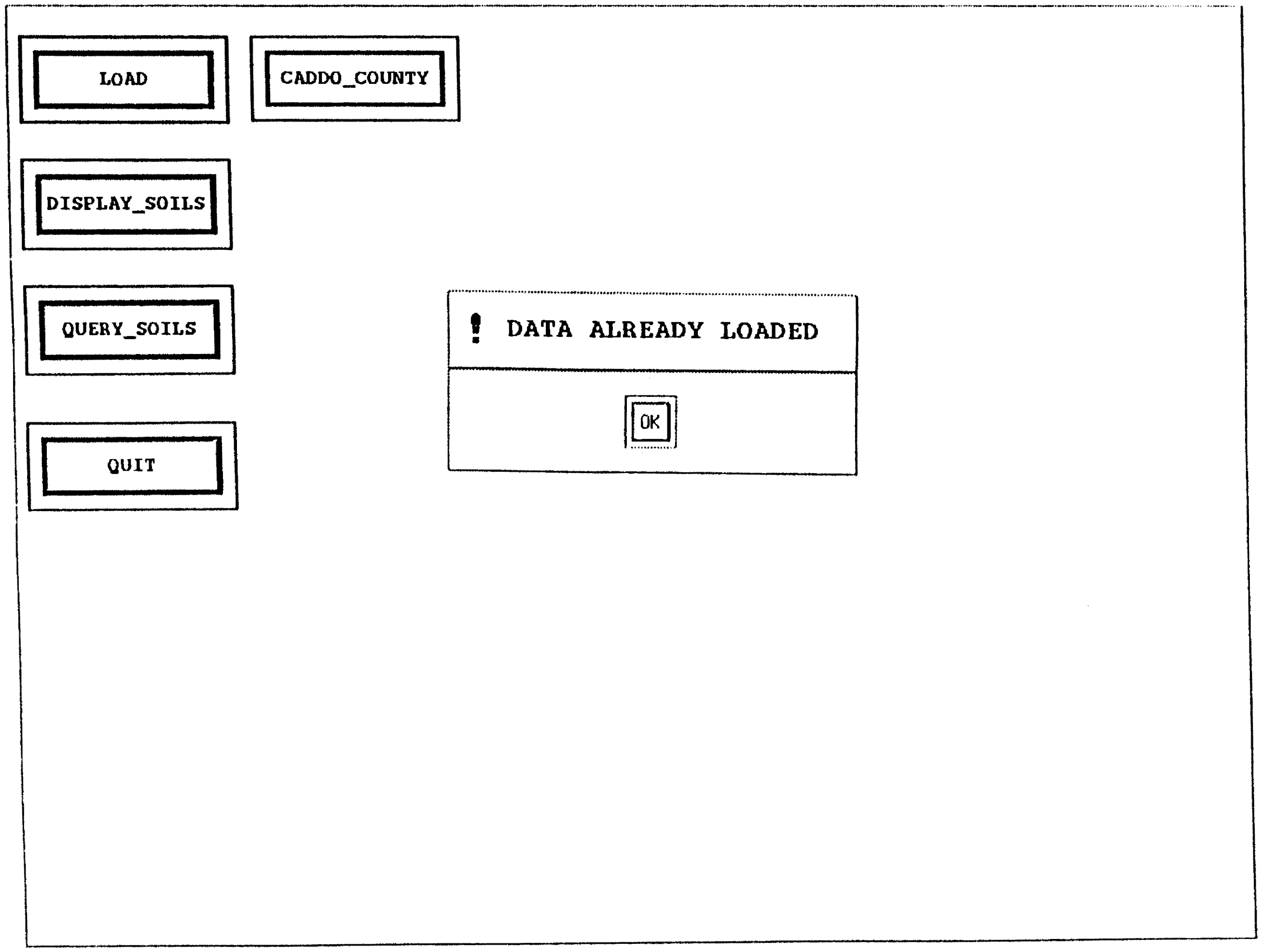


Figure 11. List of Counties when Load Button is Selected



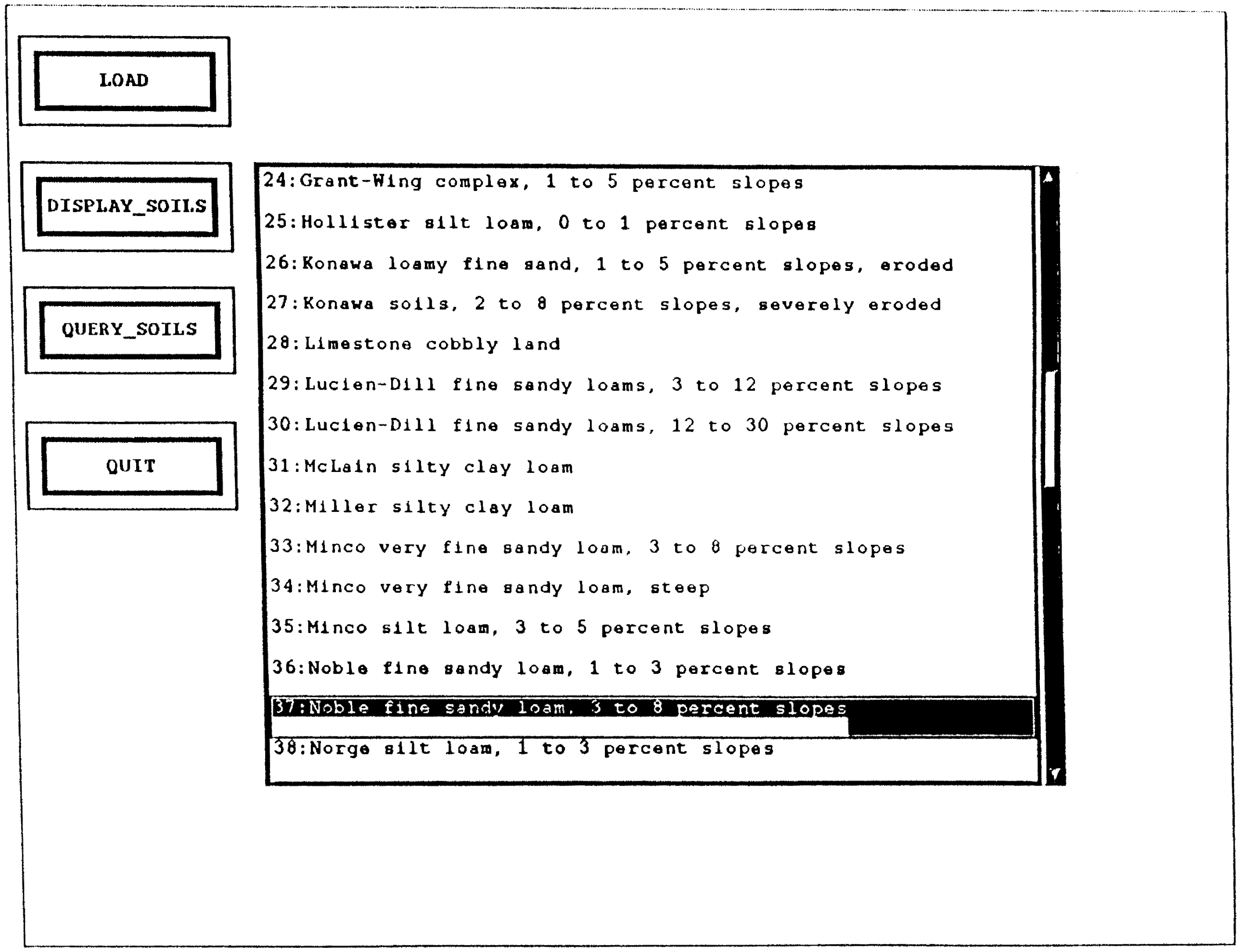
LOADING CADDO COUNTY DATA

Figure 12. Display when Loading Data



! DATA ALREADY LOADED

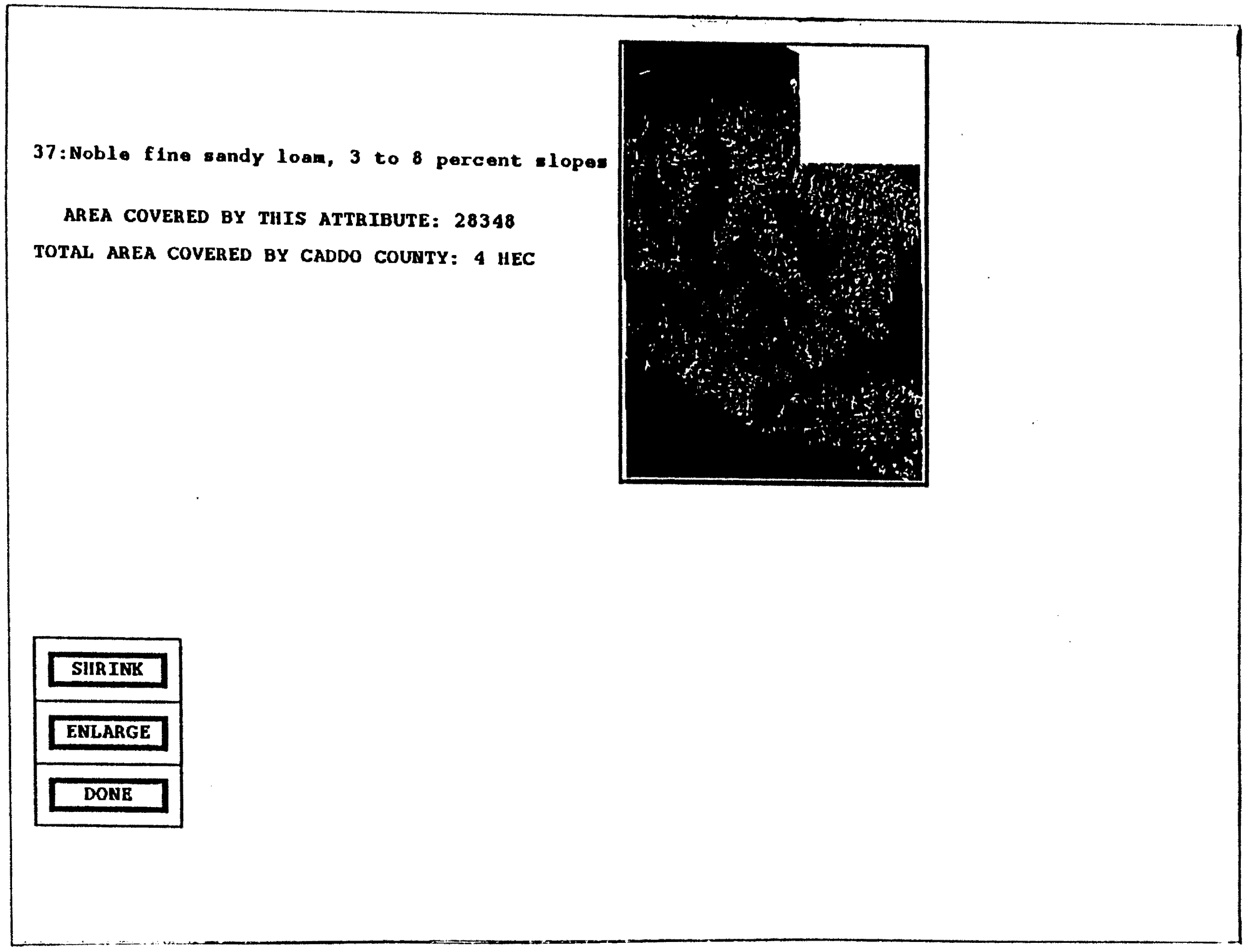
Figure 13. Display when Data is Already Loaded



```
24:Grant-Wing complex, 1 to 5 percent slopes
25:Hollister silt loam, 0 to 1 percent slopes
26:Konawa loamy fine sand, 1 to 5 percent slopes, eroded
27:Konawa soils, 2 to 8 percent slopes, severely eroded
28:Limestone cobbly land
29:Lucien-Dill fine sandy loams, 3 to 12 percent slopes
30:Lucien-Dill fine sandy loams, 12 to 30 percent slopes
31:McLain silty clay loam
32:Miller silty clay loam
33:Minco very fine sandy loam, 3 to 8 percent slopes
34:Minco very fine sandy loam, steep
35:Minco silt loam, 3 to 5 percent slopes
36:Noble fine sandy loam, 1 to 3 percent slopes
37:Noble fine sandy loam, 3 to 8 percent slopes
38:Norge silt loam, 1 to 3 percent slopes
```

Figure 14. Display of Scrolled List of Soils

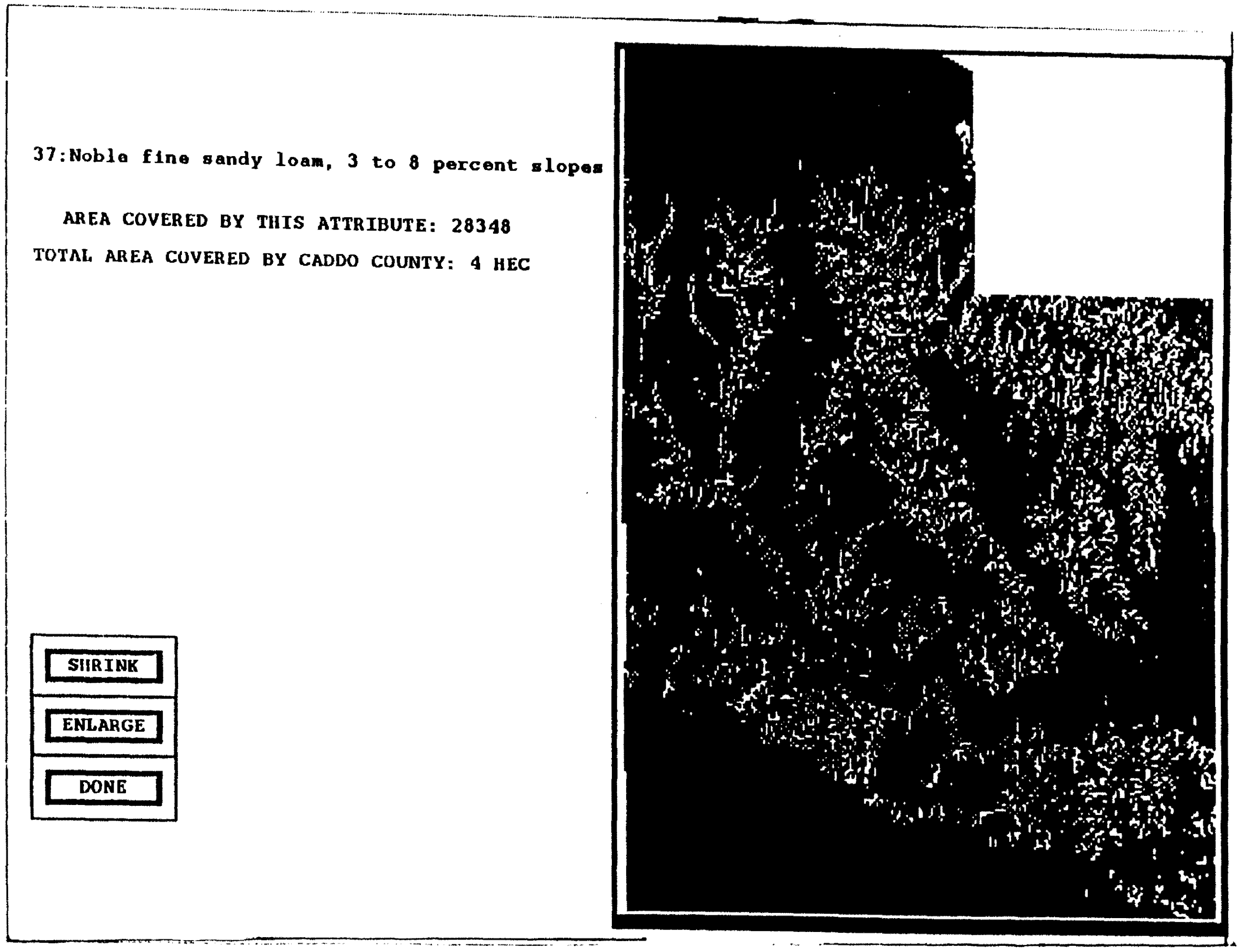
traversed up or down. To select an attribute, mouse button has to be clicked twice on the desired item. This opens up another window (figure 15) to display the selected attribute. This window displays the outline of Caddo county with the distribution of the selected attribute in the county. The name of the selected attribute and the area covered by this attribute are also displayed. Internally the leaf level linked list of the R* tree is traversed and the rectangles containing the selected attribute are displayed on the screen, at the selected portion with respect to the scale desired. Three options are provided, either to shrink the map, to enlarge the map or to quit when done. Only two scales are provided (SHRINK and ENLARGE). The first scale (SHRINK) draws every rectangle with an area of 1 x 1 pixel area and the other (ENLARGE) with 2 x 2 pixel area. These two are shown in figures 15 and 16. If 3 x 3 or more than that is selected the map grows out of proportions, so these are not provided. Though not provided in this interface, with little change in the code, larger areas can be handled by having scroll bars for the drawing area. The coordinates of any particular piece of land can be obtained by pressing the mouse button at the desired location, dragging it to cover the desired area and releasing the button. When the button is pressed and dragged, a rectangle is displayed showing the area selected. When the button is released the coordinates of the selected area are shown. This is shown in the figure 17. When the mouse button is pressed in the map draw area, the mouse button is grabbed in map window not allowing the mouse to move out of the window. Since monochrome monitor is used, only one attribute is displayed at a time. Background is displayed in black and foreground (distribution of attribute) is


37:Noble fine sandy loam, 3 to 8 percent slopes

AREA COVERED BY THIS ATTRIBUTE: 28348

TOTAL AREA COVERED BY CADDO COUNTY: 4 HEC

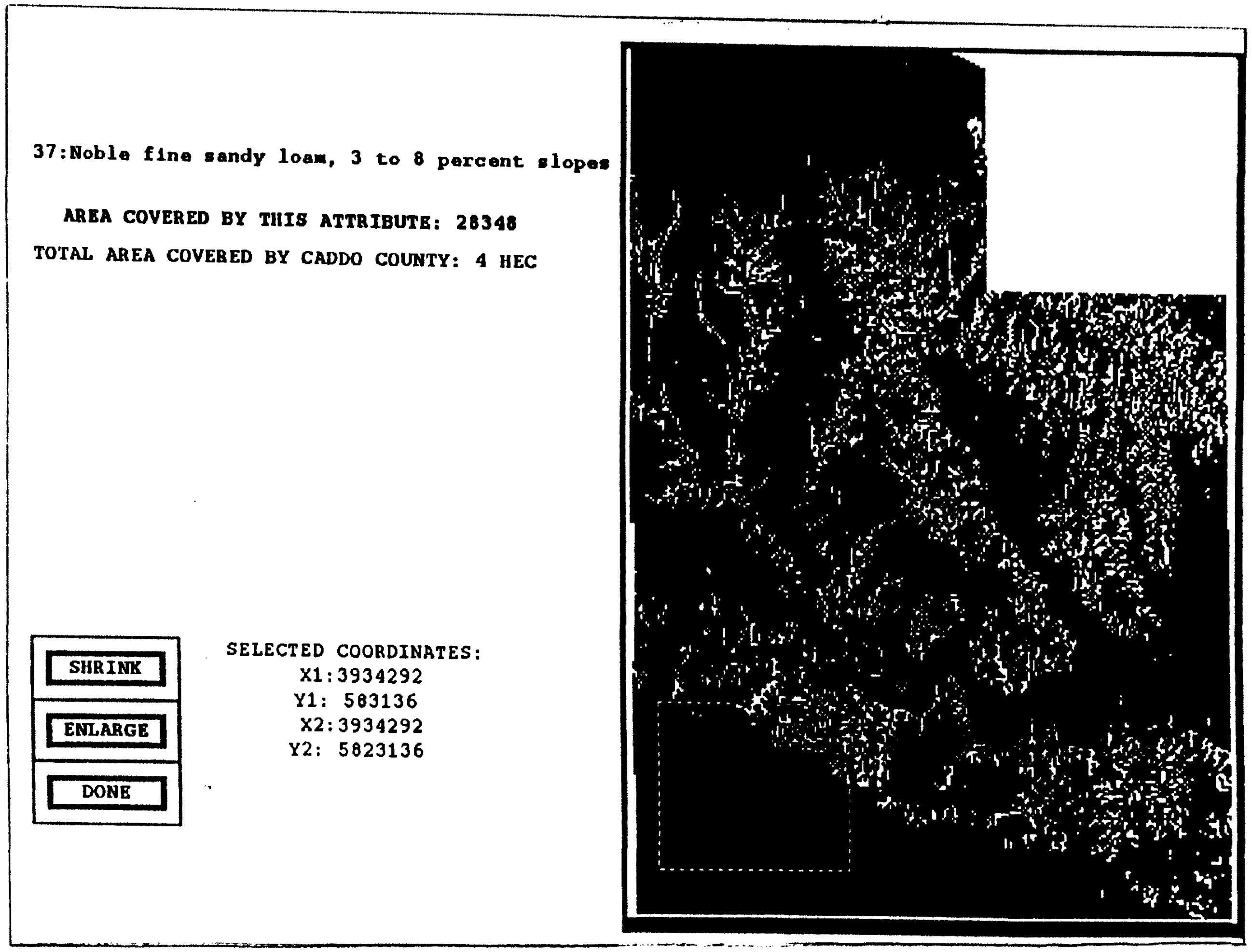
Figure 15. Distribution of Selected Attribute in Caddo County in Shrink Scale



```
37:Noble fine sandy loam, 3 to 8 percent slopes

   AREA COVERED BY THIS ATTRIBUTE: 28348
TOTAL AREA COVERED BY CADDO COUNTY: 4 HEC
```

Figure 16. Distribution of Selected Attribute in Caddo County in Enlarge Scale



37:Noble fine sandy loam, 3 to 8 percent slopes

AREA COVERED BY THIS ATTRIBUTE: 28348

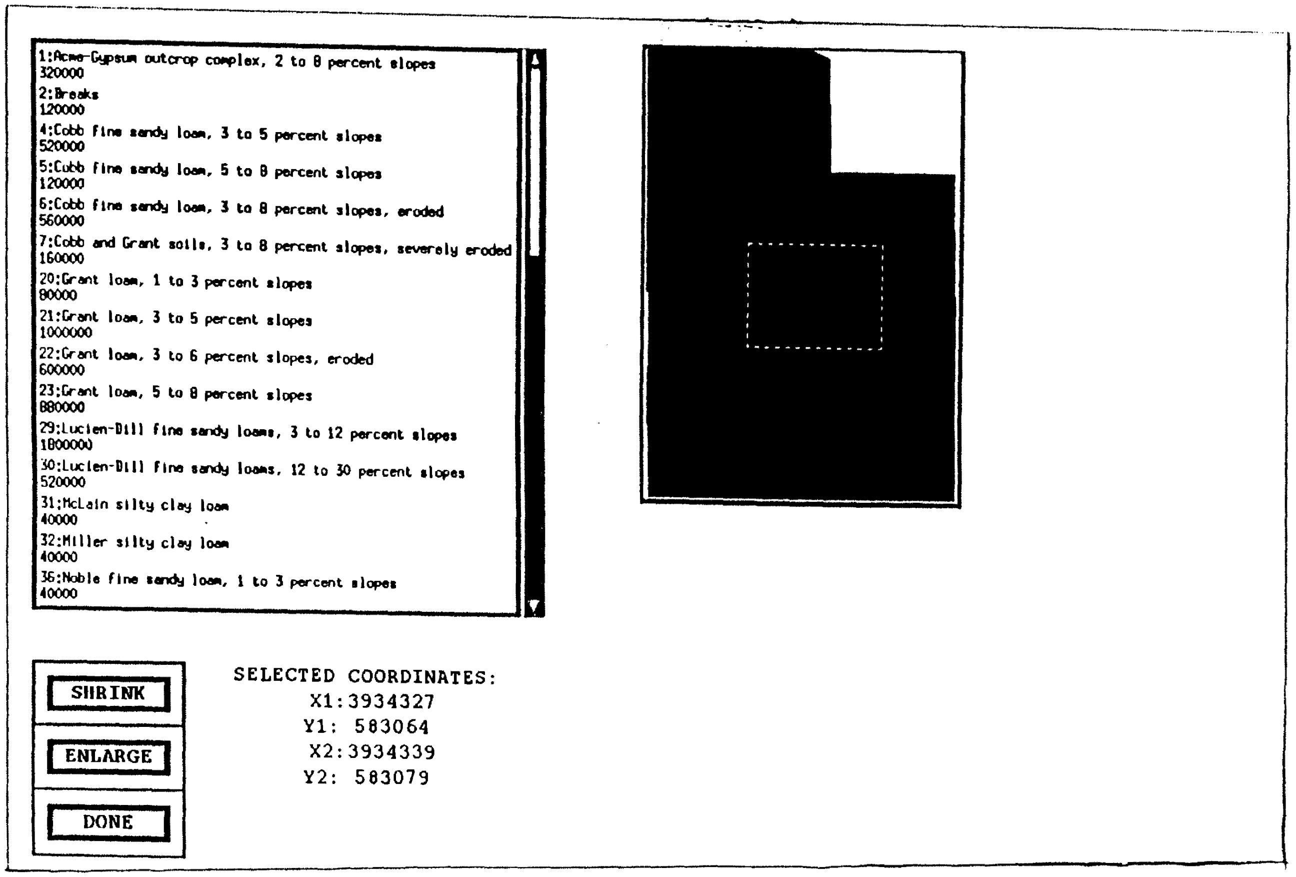
TOTAL AREA COVERED BY CADDO COUNTY: 4 HEC

SELECTED COORDINATES:
X1:3934292
Y1: 583136
X2:3934292
Y2: 5823136

Figure 17. Obtaining Coordinates of Selected Area when Attribute is Displayed

shown in white. Since we cannot get very many shades from these two colors only one attribute at a time is displayed. Code can be modified to display multiple attributes if working on a color monitor. By selecting the DONE button the control is passed back to the main window.

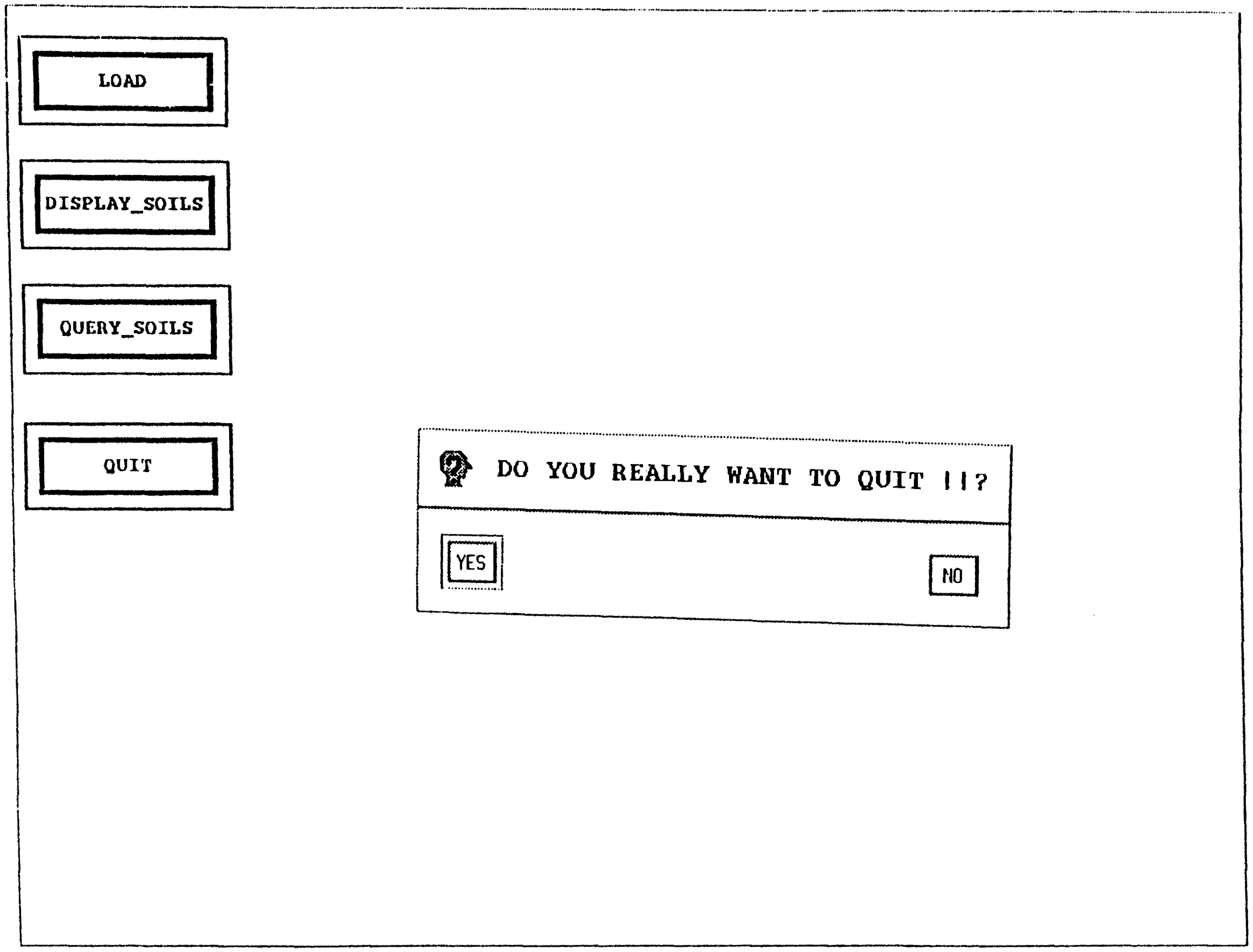
Query of soils can be done by pressing the QUERY_SOILS button. Once this is selected a new window is opened similar to that of the display soils window. But this window shows only the outline of the Caddo county. Both point and range queries can be preformed. To perform a point query, press and release the mouse button at the desired location on the map showing caddo county. A window opens showing the attribute and its area. To perform range query press and drag the mouse button till the required area is selected. When it is pressed and dragged, a rectangle is shown displaying the area being selected. When the button is released a scrolled window displays the results. The window contains all the attributes and their amount of distribution in the area selected. A scrolled window is provided to handle a query with larger number of attributes. Also the coordinates of the selected area are displayed. This is shown in the figure 18. Internally, the coordinates of the selected area are scaled and passed to the query function of the R* tree. This function searches the R* tree for the embedded and the overlapping rectangles of the given search rectangle. The results are displayed using the interface. Similar to that of the display soils, two sizes are provided for the querying and these can be selected by pressing the SHRINK or ENLARGE button. Pressing the DONE button passes the control to main window and it is displayed waiting for further responses from the user.



```
1:Acme-Gypsum outcrop complex, 2 to 8 percent slopes
320000
2:Breaks
120000
4:Cobb fine sandy loam, 3 to 5 percent slopes
520000
5:Cobb fine sandy loam, 5 to 8 percent slopes
120000
6:Cobb fine sandy loam, 3 to 8 percent slopes, eroded
560000
7:Cobb and Grant soils, 3 to 8 percent slopes, severely eroded
160000
20:Grant loam, 1 to 3 percent slopes
80000
21:Grant loam, 3 to 5 percent slopes
1000000
22:Grant loam, 3 to 6 percent slopes, eroded
600000
23:Grant loam, 5 to 8 percent slopes
880000
29:Lucien-Dill fine sandy loams, 3 to 12 percent slopes
1800000
30:Lucien-Dill fine sandy loams, 12 to 30 percent slopes
520000
31:McLain silty clay loam
40000
32:Miller silty clay loam
40000
36:Noble fine sandy loam, 1 to 3 percent slopes
40000
```

```
SELECTED COORDINATES:
      X1:3934327
     Y1:  583064
      X2:3934339
     Y2:  583079
```

Figure 18. Querying a Selected Area in Caddo County



DO YOU REALLY WANT TO QUIT !!?

Figure 19. Display when QUIT is Selected

Display of soils and query of soils can be selected as many times as desired. When QUIT button is pressed, a popup dialogue box opens as shown in figure 19. If YES is selected the program terminates otherwise the control passes back to the main window.

The insertion function of the R* tree is used for loading the data, and for the querying, search function of the R* tree is used. To display the selected attribute the leaf level linked list inbuilt in the R* structure is utilized.

CHAPTER V

CONCLUSIONS

The R* tree provides efficient point and range querying capability. But its structure as such, is not suitable for graphical interface of the data. By making it leaf level linked structure the actual data can be accessed by traversing the leaf level. This reduces the time for accessing the actual data sequentially in the R* tree. Also the leaf level linked list is implemented with negligible increase in the cost for insertion than in R* trees. Querying can be done with the same performance as that of R* tree since level linked structure preserves the basic R* tree structure. Thus the level linked structure proposed in this thesis performs both querying and graphical representation of spatial data. Existing structures can perform operations like insert, delete, and access. But the proposed structure, in addition to all above operations can perform graphical representation of the spatial data.

Fully level linked R* trees are useful for implementing hotspot concept. This linking of the levels helps to implement many more applications that are to be explored. X Windows provide excellent user interface and supports good graphics.

REFERENCES

[AT92] Adrain, N., Tim, O'reilly.: "X Toolkit Intrinsics Programming Manual", O'Reilly & associates, Inc., California, 1992.

[Carl92] Carl, F.: "An Introduction to Geographic Information Systems: Linking Maps to Databases", Database, April 1992 pp 12-21

[Frank92] Frank, A.U.: "Spatial Concepts, Geometric Data Models, and Geometric Data Structures", Computers and Geosciences Vol.18, No.4, 1992 pp.409-417.

[Hel92] Heller, D.: "Motif Programming Manual", O'Reilly & Associates, Inc, California, 1992.

[HT90] Henskes, D.Th., Tolmie, J.C.: "Prototyping and Visualization in Interface Design", Electrical Communication, Vol.64,No.4, 1990, pp. 321-326.

[NHRB90] Nobert, B., Hans, P.K., Ralf, S., Bernhard, S.: "The R* -tree: An Efficient and Robust Access Method for Points and Rectangles", ACM SIGMOD, NewYork, Vol 19, No.2, 1990 pp.322- 331.

[OS90] Ohsawa, Y., Sakauchi, M.: "A New Tree Type Data Structure With Homogeneous Node Suitable For A Very Large Spatial Database", Proc. of the IEEE Sixth Int. Conf. on Data Engg. 1990 pp.296-303.

[Oli89] Oliver, J.: "Introduction to the Window X System", Prentice Hall, NewJersy, 1989.

[Peter89] Peter. H.L.: "When Maps Are Tied To DataBases", Newyork Times, Sunday, May 28 1989, pp 10.

[FS87] Freestone, M.: "The BANG File: A New Grid File", Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1987, pp.260-269.

[SRF87] Sellis, T., Roussopoulos, N., Faloutsos, C.: "The R+ -tree: a Dynamic Index Structure For Multi-dimensional objects", Computer Science TR-1795, University of Maryland,College Park, MD, February 1987.

[Gutt84] Guttman, A.: "R-Trees: A Dynamic Index Structure for Spatial Searching", Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1984, pp.47-57.

[NH84] Nievergelt,J., Hinterberger, H.: "The Grid File: An Adoptable Symmetric Multikey File Structure", ACM Transactions on Data Base Systems, Vol.9,No.1, pp 38-71.

[Rob81] Robison, J.T.: "The K-D-B Tree: A Search Structure for Larger Multi Dimensional Dynamic Indexes", Proc. of the ACM SIGMOD Conference, 1981, pp.10-18.

[BF77] Bentley, J.L., Friedman, J.H.: "Data Structures for Range Searching", ACM Computing Surveys, Vol.11,No.4 , 1979, pp.397-409.

[BSW77] Bentley, J.L., Stanat, D.F., Williams, Jr., E.H.: "The Complexity of Fixed Radius Near Neighbor Searching", Inf. Proc. Lett. Conference, Vol.6, No.6, 1977, pp.209-212.

[Bent75] Bentley, J.L.: "Multi-dimensional Binary Search Trees Used for Associated Searching", Communications of the ACM, Vol.18, No.9, 1975, pp.509-517.

[FB74] Fink, R.A., Bentley, J.L.: "Quad Trees: A Data Structure for Retrieval on Composite Keys", Acta Informatica, Vol.4, No.1, 1974, pp.1-9.

[Knu73] Knuth, D.E.: "The Art of Computer Programming", Vol.1, Fundamental Algorithms,Second Edition, Addisson-Wesley, MA, 1973.

VITA

VCS REDDY KUMMETHA

Candidate for the Degree of

Master of Science

Thesis: A LEVEL LINKED R* TREE STRUCTURE WITH AN APPLICATION USING X-WINDOW GRAPHICAL INTERFACE

Major Field: Computer Science

Biographical:

Personal Data: Born in Anantapur, Andhra Pradesh, India, December 12, 1969, son of Rama Krishna Reddy K., and Sita K.

Education: Graduated from LRG High School, Anantapur, Andhra Pradesh, India, in May, 1985; received Bachelor of Engineering in Mechanical Engineering from Osmania University in May, 1991; completed requirements for the Master of Science Degree in Computer Science at Oklahoma State University in December, 1993.

Professional Experience: Graduate Research Assistant, University Computer Center, Oklahoma State University, July, 1992 to present.