# MULTI-DIMENSIONAL DATA STRUCTURES FOR MARINE SEISMIC DATA

by

**Peter Anthony Judd**

**TR95-097, September 1995**

This is an unaltered version of the author's
M.Sc.(CS) Thesis

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566

# MULTI-DIMENSIONAL DATA STRUCTURES FOR MARINE SEISMIC DATA

*by*

*Peter Anthony Judd*

B.Sc. Eng. University of New Brunswick, Canada, 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in the
Faculty of Computer Science

This thesis is accepted

.....

Dean of Graduate Studies
THE UNIVERSITY OF NEW BRUNSWICK

July 1995

# ABSTRACT

This thesis contains the results of an investigation into the design and implementation of spatial data structures for marine seismic data. An index structure, for marine seismic data in SEG-Y format, was designed using a modified PR quadtree data structure and implemented in the C++ computer language.

To test the software and to evaluate the implementation of the structures, experiments were carried out with data from a seismic survey in the Mid Pacific Ocean. The section of the survey used spanned 6 days and consisted of 8 seismic lines. The total amount of data is 335.8 Mbytes with a total of 39752 shot points.

It was shown that the modified PR quadtree structure was superior to the standard PR quadtree with an increase in 2D X/Y range searching speed varying from approximately 9% for the third 10 percent cover test to 67% for the 100 percent cover test. The modified PR quadtree structure groups the seismic shots into objects thus allowing pruning operations to be performed on the structure during a search. This pruning, in some cases, can result in large sections of the tree being accepted sooner during the search.

# ACKNOWLEDGEMENTS

I wish to thank my supervisor, Dr. Bradford G. Nickerson, for his suggestion of this subject and consistent guidance through each phase of this research. His valuable advice and insightful thoughts have greatly contributed in shaping this research work. Thanks also to Kirby Ward and Mike MacDonald for the technical help that they provided during my thesis work.

Great appreciation is also extended to the Ocean Mapping Group for helping to fund my research work and to Steve Bloomer for his assistance in acquiring the seismic data. Thanks also go to Dr. Larry Mayer for his comments and suggestions.

# TABLE OF CONTENTS

Page

Page

# LIST OF FIGURES

Page

Page

# LIST OF TABLES

Page

Page

# CHAPTER 1
# INTRODUCTION

## 1.1 Background

Methods for capturing multi-dimensional data have changed dramatically in recent years. This has resulted in a substantial increase in the volume of multi-dimensional data that is captured. Every time the resolution doubles, the data volume, on average, will increase by a factor of four. In response to this increase, new data structures to organize, store, retrieve, generate and distribute spatial data efficiently continue to be developed. This thesis looks at data structures for handling queries for marine seismic data.

Seismic data are used extensively in oil exploration and related geophysical applications to determine the earth's subsurface structure. This information is then used to determine the location and extent of ore bodies or petroleum deposits. For engineering applications, the information is used to determine where infrastructure such as bridge supports, pipelines, communication lines and roads are to be located. The data that were studied for this thesis were from marine sources but the work is also applicable to terrestrial applications.

Modern swath sounding instruments can produce bathymetric data sets covering the entire seabed. These data are normally processed and represented as an evenly spaced grid of depth values [Ware et al, 1992].

## 1.2 Seismic Exploration

Exploration seismology is based on earthquake seismology. With earthquakes, when the earth fractures, large amounts of energy are released. This energy produces seismic waves which travel outward from the epicenter of the quake. The waves are recorded at remote sites and the information is used to deduce the nature of the portion of the crust through which the seismic waves traveled [Telford, 1985].

Exploration seismology uses the same principles except that the location, the number of sources and the size of the energy emitted by the sources are carefully regulated. Much of the seismic data consists of continuous coverage of the survey area; i.e. successive portions of the crust are sampled along specific survey lines at distinct intervals.

Marine seismic data are normally obtained from a specially equipped geophysical survey vessel. The data collected include the seismic data and core samples which are used to verify the seismic results. The energy sources are devices such as air guns, imploders, sleeve exploder, Aquapulsers or chirp sonars. These devices create an energy pulse called a "shot" which is emitted at discrete time intervals. The *shot* interval depends on the depth of the water and the instrumentation being used.

The seismic wave travels down through the seafloor subsurface structure. At each interface boundary between the various seafloor sediment layers, the seismic energy wave is split into two components. The first component is reflected back to the surface while the second component is transmitted through to the next layer (see figure 1.1). The arrival of the reflected wave fronts is then detected at the sea surface by detectors called hydrophones. These

hydrophones (also called marine pressure geophones), are piezoelectric devices which depend on the fact that application of pressure to certain substances produces an electric potential difference between the two surfaces of the material. Hydrophones are often arranged in pairs so that their output is canceled for translation accelerations (due to sensor motion), but added for pressure pulses.

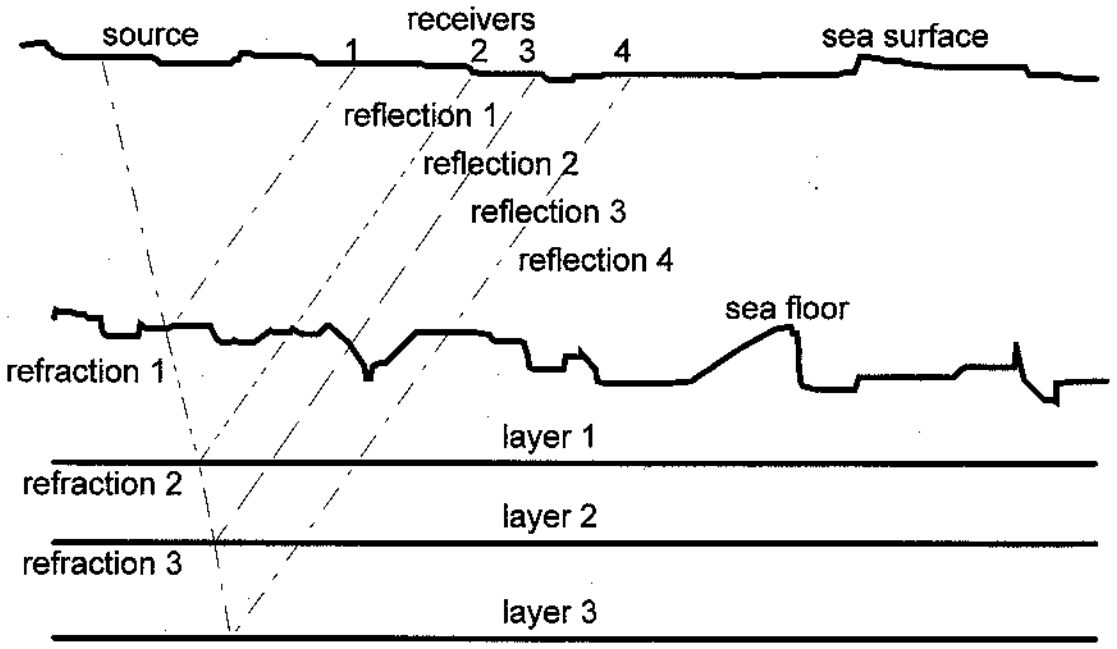



Figure 1-1. Refraction and reflection of seismic waves.

The hydrophones are mounted in long streamers towed behind the seismic ship at a depth often between 10 and 20 metres. The first group of hydrophones is usually spaced at least 100 metres behind the vessel and trailing the last group is a buoy which is used to determine the position of the individual hydrophones (see figure 1.2).

The information that is recorded for each shot include:

1. the travel time for the seismic wave
2. the position of the vessel
3. the position of the individual hydrophones
4. the time delay from the instant that the shot was emitted until the recording of the data started
5. the intensity of the reflected waves.

The shot information from the individual hydrophones can then be used to reduce the raw data and, if required, collapse the trace (see section 2.1), down to one trace per shot. The final data are then stored on tape using one of the standard formats for seismic data. A typical seismic tape can store about 30 MBytes of raw data. One of the standard tape formats for seismic data is called the SEG-Y tape format. The SEG-Y format is generally accepted as a common format for both marine and land seismic data.

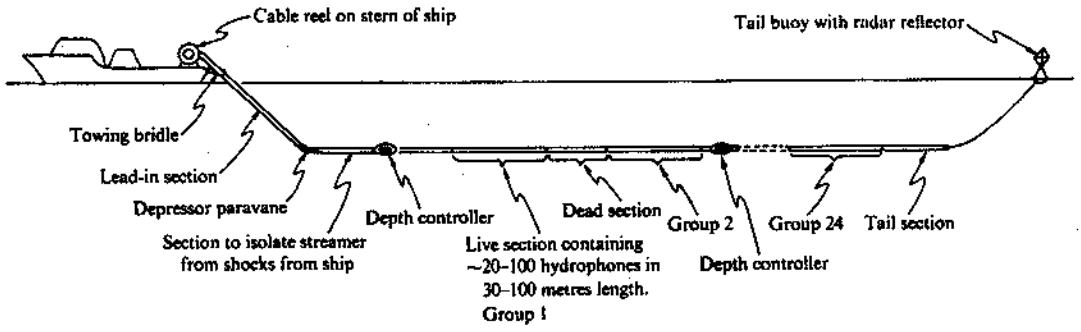



Figure 1-2. Marine seismic survey, from Sheriff [1973].

## 1.3 Literature Review

Spatial data structures have been studied for many years and numerous data structures have been developed to store point, line, polygon, surface and volume primitives. In some cases complex objects can be decomposed by mathematical methods into points or approximated by point primitives. In these cases point spatial operations can then be used on the decomposed representation of the object.

In order to speed up operations such as queries on GIS's, advanced indexing mechanisms have been developed using various spatial data structures such as grid methods, tree structures and linear encoding. Sections 1.3.1 to 1.3.4 contain discussions of several of the hierarchical tree structures such as the quadtree.

### 1.3.1 AVL Binary Search Tree

A tree is a collection of ***n*** nodes (one of which is called the root node), and ***n -1*** edges. In a binary tree, a node can have 0, 1 or 2 children, and every node except the root node has one parent ( see figure 1.3). In a binary tree the average depth is **O(logn )** but in the worst case the depth can be $\theta$**(n)** (vine tree)[Weiss, 1992]. A binary search tree has the property that the data values stored at all nodes in the right subtree of a node ***P*** are greater than the data value stored at node ***P***. Similarly all data values in this left subtree of ***P*** are less than the data value stored at ***P***. An AVL [Adelson-Velski and Landis, 1962] tree is a binary search tree with a balance condition which ensures that the depth of the tree is **O(log n).** The balance condition requires that for every node in the tree the height of the left and right subtrees can differ by at most 1. This enables

tree operations such as insert, delete and find to be performed in $\theta(\log n)$ time.

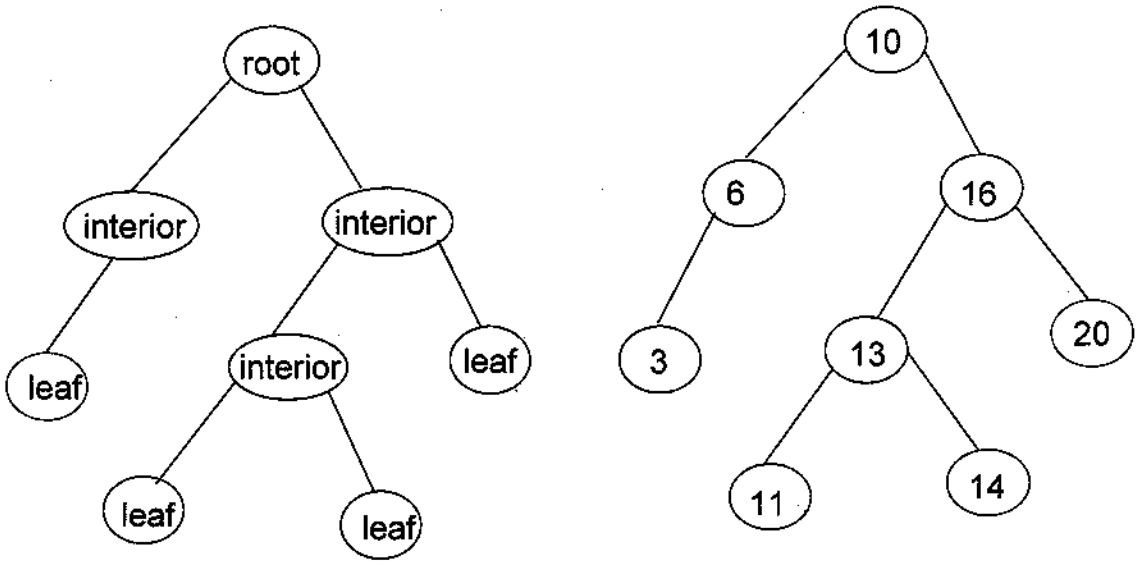



Figure 1-3. An example of an AVL binary search tree.

### 1.3.2 Quadtrees

The term quadtree is used to describe the class of structures that is based on the principle of recursive decomposition of space into four quadrants [Samet, 1990]. The decomposition may be regular (i.e. the quadrants at each level are of equal size), or irregular. In the irregular type, the order and the type of data govern the nature of the decomposition. Quadtrees can be used to store point, area, and curvilinear data. Another factor which distinguishes the various types of quadtrees is whether the resolution is fixed or variable. Some of the different types of quadtrees are:

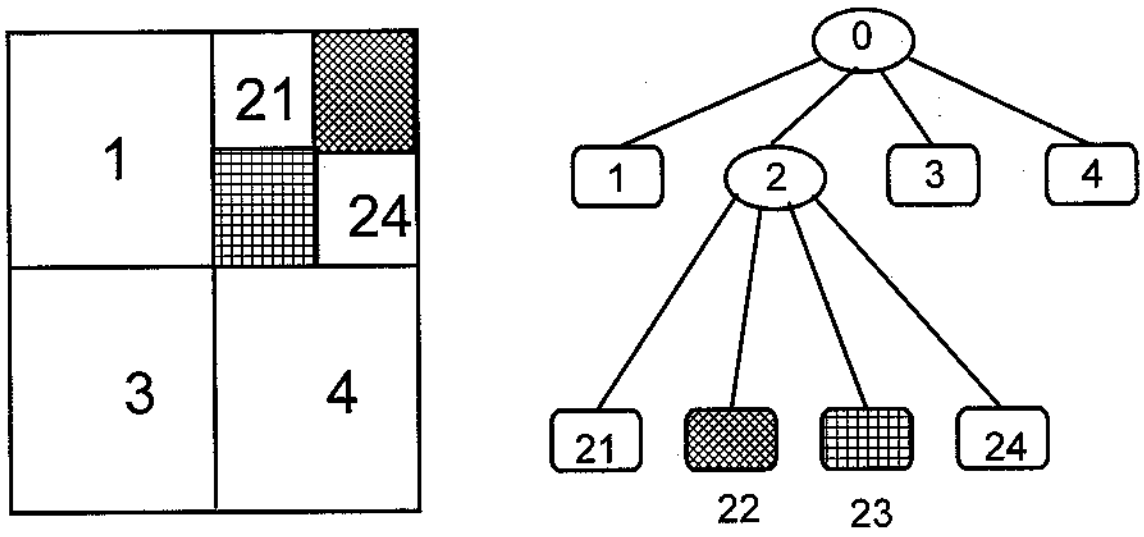
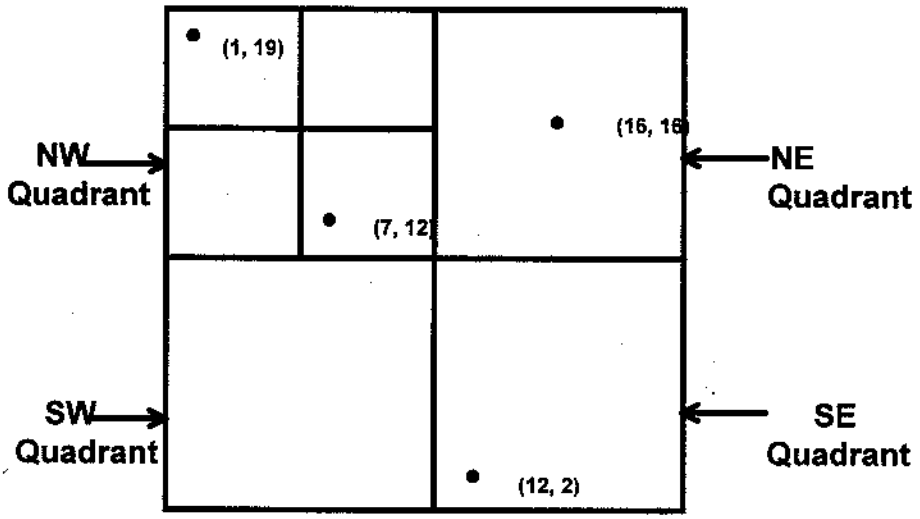



Figure 1-4. Region quadtree.

1. region quadtree: a bounded image array is subdivided into four equal sized quadrants.
2. point quadtree: points divide the space into quadrants as they are inserted. The dividing lines go through the points.
3. MX quadtree (Matrix quadtree): data points are treated as nonzero elements in a square matrix.
4. PR quadtree: an adaptation of the region quadtree for points [Samet, 1990]. As each point is inserted into the quadtree the quadrant into which the point falls is determined. If the target quadrant is already occupied then the quadrant is recursively subdivided until each quadrant contains only one point (See figure 1.5). The quadrant labeling that will be used throughout this thesis is NW (north west), NE

(north east), SW (south west) and SE (south east). The quadrants are labeled in a Z pattern starting at the NW quadrant.



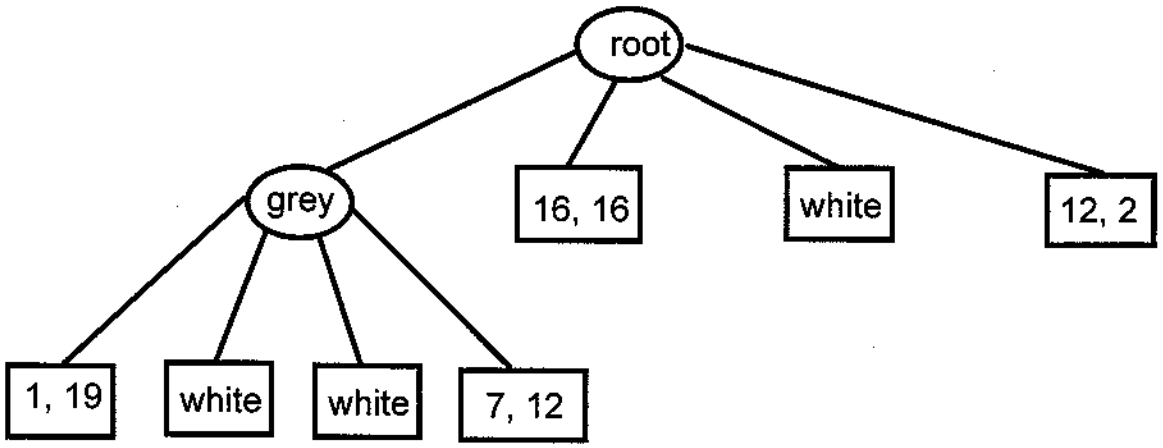




Figure 1-5. An example of a PR quadtree.

### 1.3.3 Range Trees

A range tree is a structure developed by Bently and Maurer [1975] that is designed to retrieve all points that fall within a given range. This structure is asymptotically faster than the point quadtree and the **k-d** tree but has an $\theta$**( n log n )** storage requirement due to its structure [Samet, 1990].

A two dimensional range tree is constructed by first sorting all the points in one direction, say **x** and storing the sorted points in the leaf nodes of a balanced binary search tree ***T.*** Attached at each non-leaf node ***P*** is another balanced binary search tree ***T'*** with its external nodes housing all the points in the subtree rooted at ***P.*** In the tree ***T'***, the points are ordered by the second dimension **y** (see figure 1.6).

| NAME | X | Y |
|---|---|---|
| CHICAGO | **35** | **40** |
| MOBILE | **50** | **10** |
| TORONTO | **60** | **75** |
| BUFFALO | **80** | **65** |
| DENVER | **5** | **45** |
| OMAHA | **25** | **35** |
| ATLANTA | **85** | **15** |
| MIAMI | **90** | **5** |

Figure 1-6. Sample list of cities with their x and y coordinate values, from Samet [1990].

For a given query ( [Xsw, Xne], [Ysw, Yne]), the search is started by searching for Xsw and Xne. The path to Xsw is ascended from the closest common non-leaf node ***Q*** of the two paths. If a non-leaf node ***P's*** left child is also in the path, search its right child's binary tree ***T'*** sorted on y. On the other path to Xne from ***Q***, if ***P's*** right child is also in the path, search its left child's binary tree

binary tree ***T'***. A two-dimensional range tree of n points requires **O(n $\log_2$ n )** storage and a range search returning **m** points takes **O ( $\log^2_2$ n + m )** time [Samet, 1990].

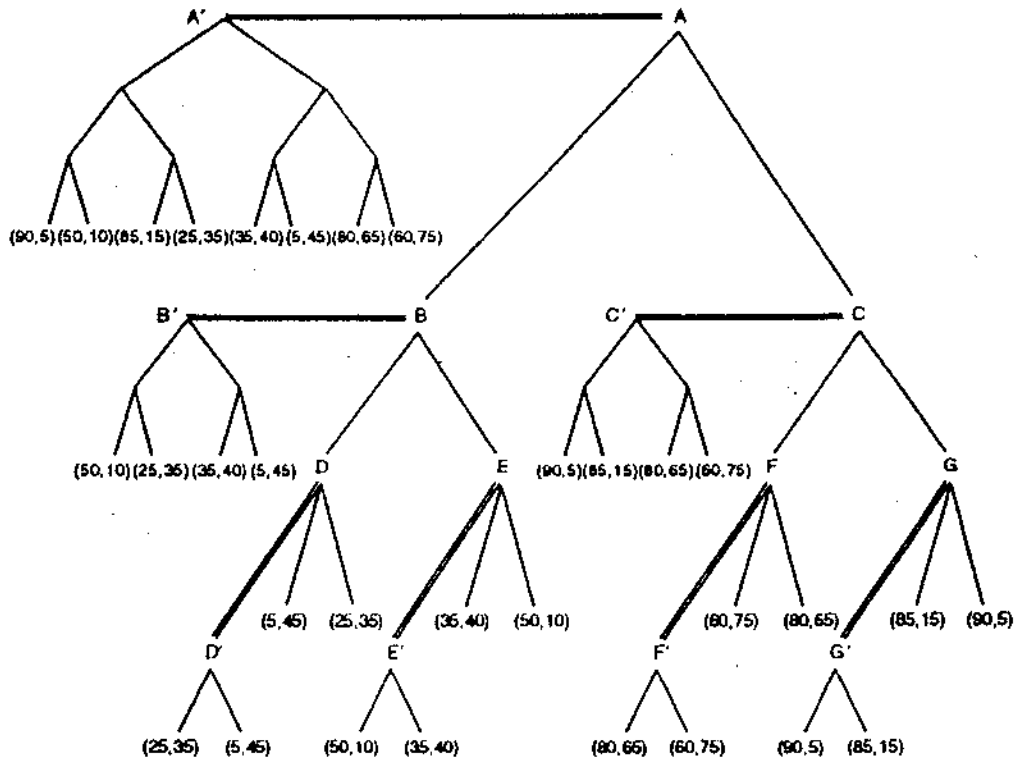



Figure 1-7. Example of a 2D range tree for the data of Figure 1-6, from Samet [1993].

### 1.3.4 Octrees

An octree is a three-dimensional extension of the region quadtree [Samet ,1990]. The octree is based on the successive decomposition of a 2n by 2n by 2n object array into octants and suboctants until each cube consists of a single data element. The decomposition process is represented by a tree of degree 8 in

to those cubes that contain only one data element. Leaf nodes are said to be black if they contain a data object and white if they do not. Interior nodes (non-leaf nodes), are termed grey (see figure 1.8). The octree can contain region, point, edge, surface and volume data. Work by Navazo and Brunet [1990] and others have developed the **extended octree** which is capable of storing boundary information.

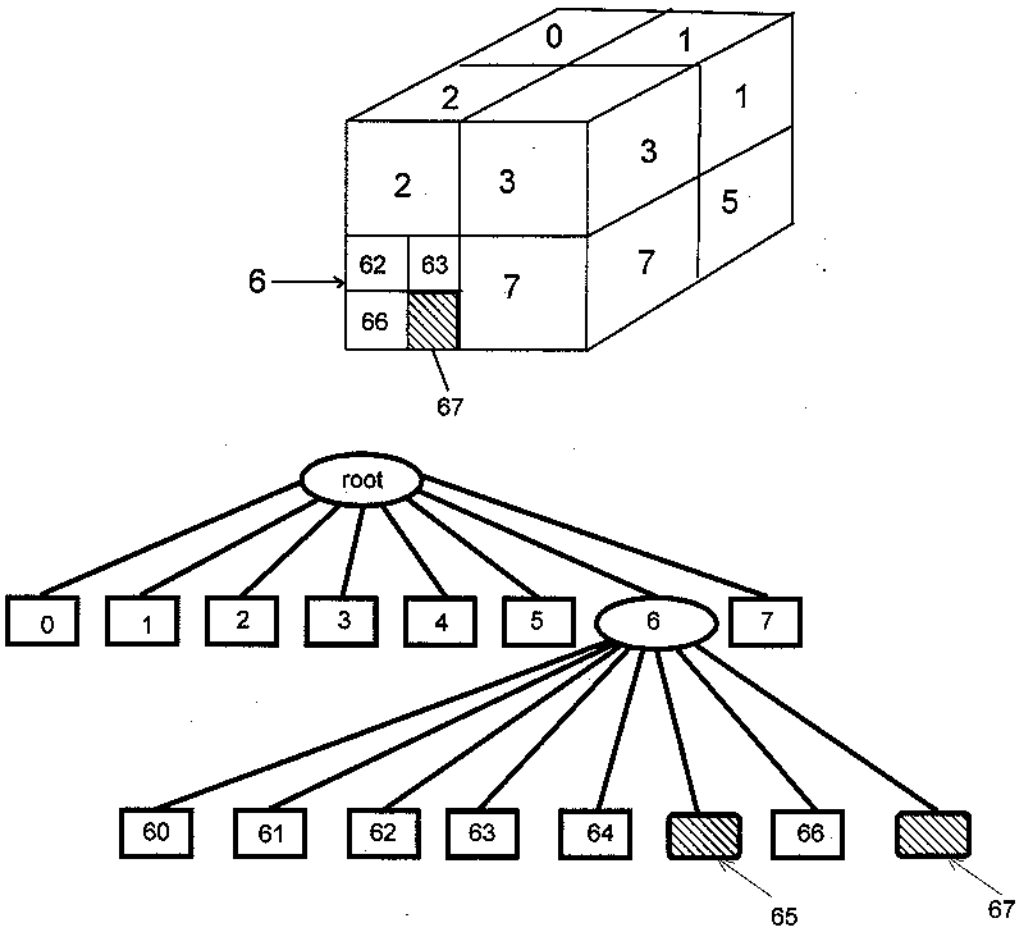



Figure 1-8. Representation of a region octree.

## 1.4 Thesis Objectives and Outline

This thesis investigates multi-dimensional data structures for efficient storage and retrieval of marine seismic data. The objectives of the thesis are:

1. To develop a data structure and algorithms for the storage and retrieval of seismic data.
2. To implement and test the data structures and algorithms on actual seismic data.
3. To compare the new data structures with the original data structures used for the seismic data.

The thesis is divided into 7 parts. Chapter 2 describes the data, methods of obtaining the data and the format that is used to store the data. Chapter 3 describes the seismic index objects. Chapter 4 describes the data structures used for the thesis. Chapter 5 outlines the modified PR quadtree algorithm developed here. Chapter 6 describes the implementation of the seismic index data structures. Chapter 7 describes the experimental testing that was done, and compares range search execution times of the modified and standard data structures. Chapter 8 contains the conclusions and areas where future research can be performed.

# CHAPTER 2

# SEISMIC DATA

## 2.1 Geophysical Data

The experimental data used in this thesis dealt only with marine seismic data but the data structures and algorithms developed here can also be used for terrestrial seismic data. The data structures can also be used to index other geophysical data such as gravity data and magnetic data that are point based and that can be grouped as objects.

As outlined in section 1.2, seismic data are obtained from a seismic survey. The energy pulse from a "shot" is received at the hydrophones in the form of a harmonic wave which can be expressed using sine and cosine expressions [Telford, 1985]. These harmonic waves can be recorded and stored using either analogue or digital recording. Analogue devices represent the signal by a voltage which varies continuously over time. Analogue recording produces two sources of data. The first is the signal from the hydrophones. The signal after it has been processed by appropriate filters and amplifiers is then stored on magnetic tapes or disk. At the same time, the processed signal is sent to either a photographic camera or a camera based on electrostatic printing, to produce a graph on photographic paper. The graph is used for monitoring and

for interpretation. Each individual graph represents the motion of a hydrophone ( or the average of a group of hydrophones) and is defined as a **trace.**

Analogue recording has been superseded by digital recording. This method represents the signal by a series of numbers which denote the output of the hydrophones at discrete intervals (see figures 2-1 to 2-3). The sampling interval, for the test data, ranged from 2 to 4 msec. Digital recording allows higher fidelity and permits numerical processing of the data without adding appreciably to the distortion. Figure 2-4 shows an example of a seismic line from tape 32 of the seismic test data discussed in chapter 7.

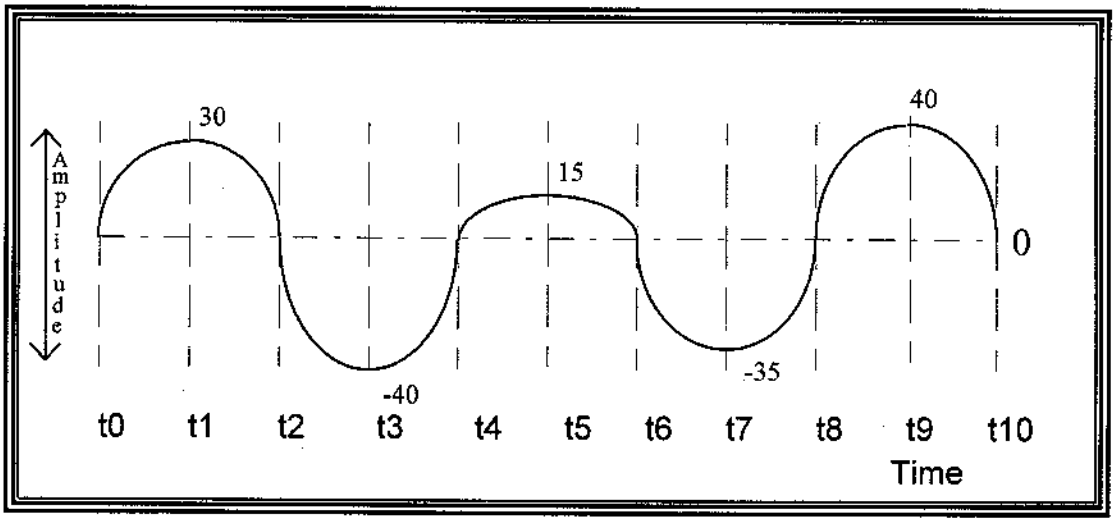



Figure 2-1. Example of a seismic trace from Landmark [1992].

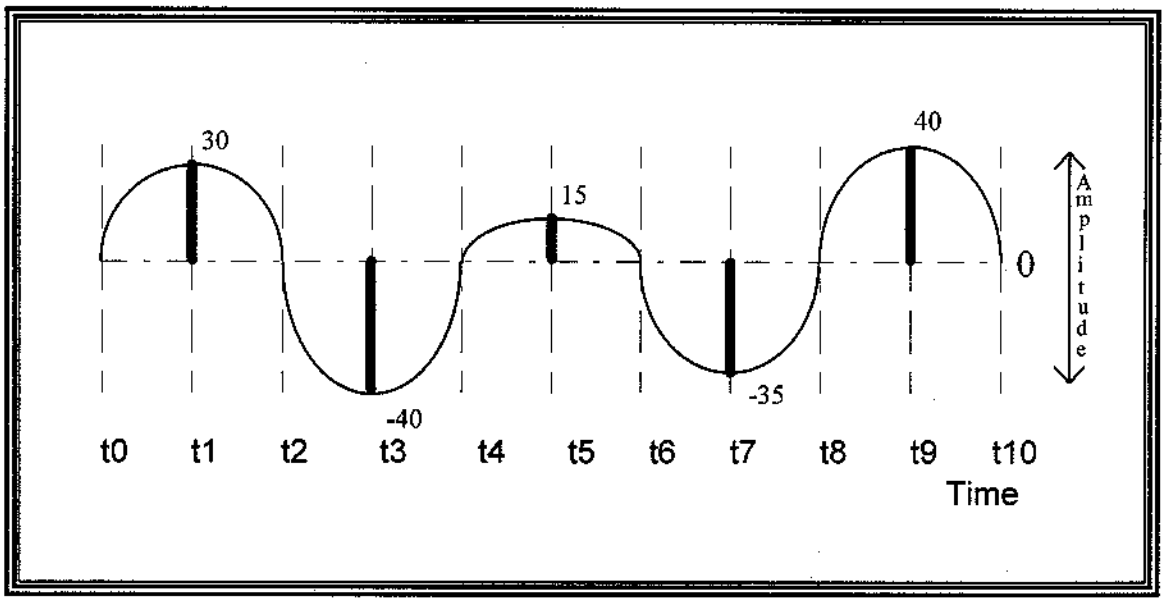



Figure 2-2. Example sampling of a seismic trace from Landmark [1992].

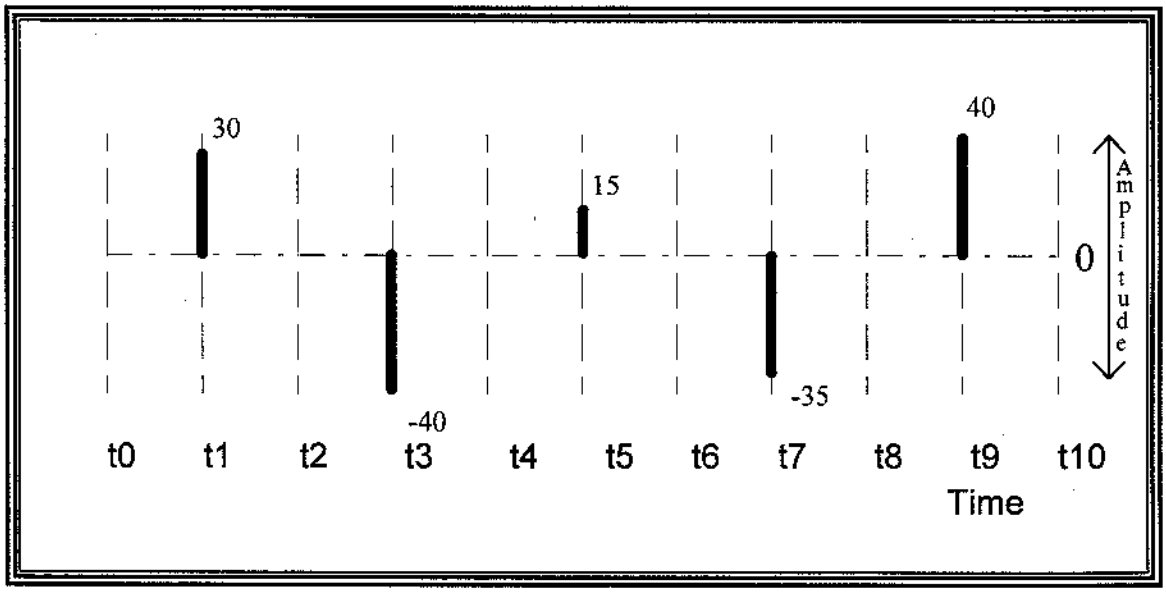



Figure 2-3. Digital samples of a seismic trace from Landmark [1992].

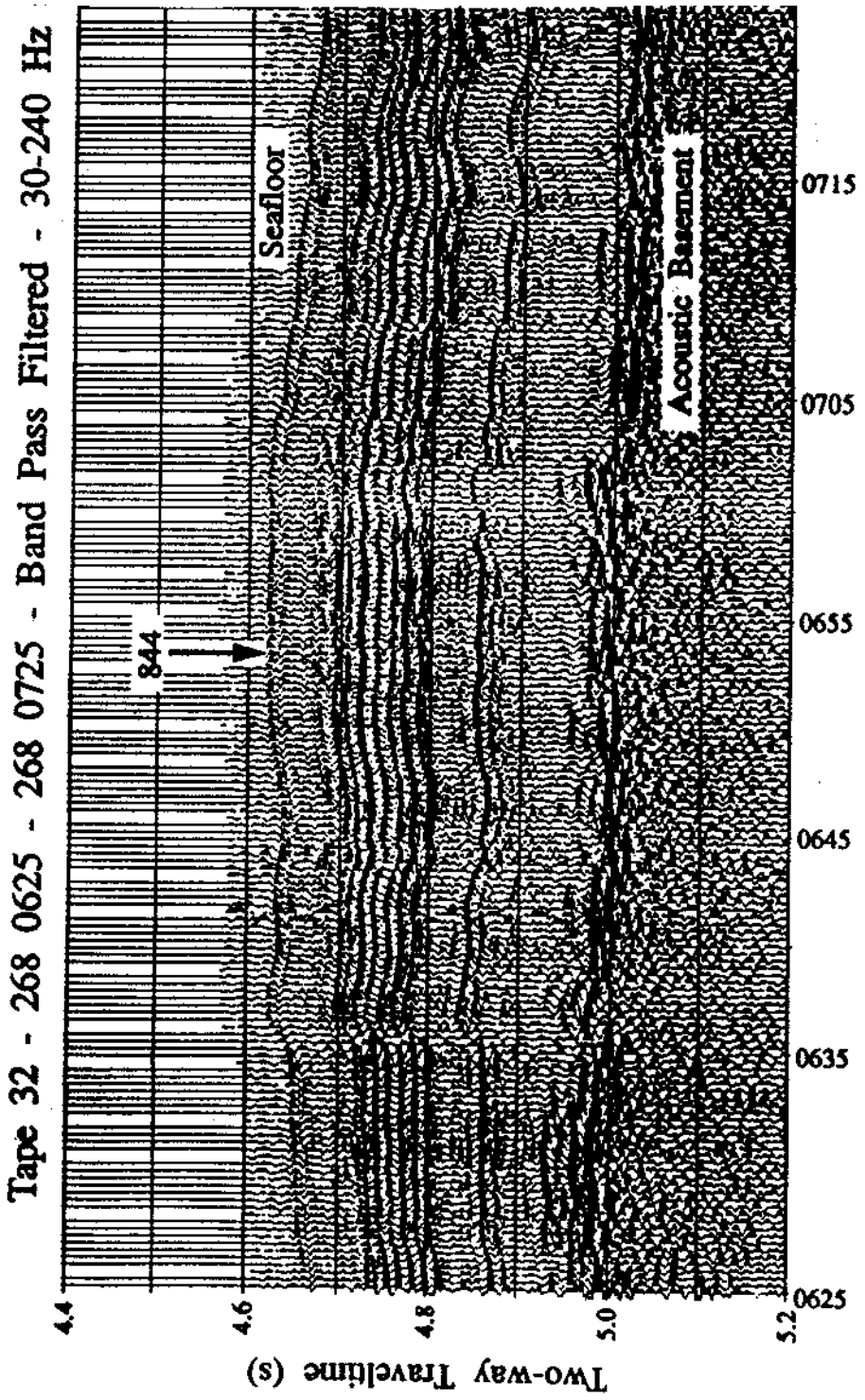



Figure 2-4. Example seismic line from tape 32 of the seismic test data [Bloomer, 1992].

## 2.2 SEG-Y Seismic Data Format

The SEG-Y tape format was developed between 1973 and 1975 and was established as one of the tape standards for capturing seismic data [Barry et al, 1975]. It was based on prior work and on the SEG Exchange Tape Format [Northwood et al, 1967]. The tape format was developed for application to computer field equipment and in the then present data processing centres with flexibility for expansion as new ideas were introduced. The SEG-Y format breaks the tape into 8 separate regions, with several of the regions being repeated as required (see figure 2-5).

| | | Header records | | Seismic trace records (repeated for each trace) | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 5 | 6 | 7 | 8 |
| 12 feet empty tape | BOT mark | 3200 byte EBCDIC header record | 400 byte Binary header record | 240 bytes seismic trace header | seismic amplitude samples | 240 bytes seismic trace header | seismic amplitude samples | EOF marks | |

Figure 2-5. Parts of an SEG-Y formatted tape from Landmark [1992].

The first 12 feet of the tape are empty. The **BOT** (beginning of tape) marker is a silver stripe on the tape. This is read by the tape drive as the

beginning of the tape write area. For the SEG-Y file stored on disk these two fields are irrelevant.

The third field is the **EBCDIC header record.** This is a text header that contains information such as the shooting parameters and processing history. This is an optional field and may be empty.

The fourth field is the **binary reel identification header record.** This record contains information about how the seismic data are stored on the tape or disk and includes the following information:

- the sample rate of the data
- the sample format of the data (floating point, 32 bit integer, 16 bit integer or 8 bit integer)
- the length of the seismic trace.

Appendix A gives a detailed description of the fields in the header record. In the following lists of the fields in the SEG-Y records, from which data were retrieved for this thesis, the symbol "[R]" indicates that it is strongly recommended that the fields contain the required information, while an "[O]" means the field is optional and may not contain valid information. For the header record, the required fields that were accessed are:

1. bytes 21 - 22 [R] : number of samples per data trace
2. bytes 25 - 26 [R] : data sample format code
3. bytes 55 - 56 [R] : measurement units (metres or feet).

The fifth and sixth fields, the **trace header** and the **seismic amplitude samples,** make up the seismic trace records. These fields are repeated for each

trace in the seismic line. The **trace header** contains the trace number, possibly the line number, the reference position for the trace, the coordinate units (whether meters, feet or seconds of arc) and various other information. Appendix B gives a complete description of the fields in the **trace identification header.** The required data fields for this research are:

4. bytes 1 - 4 [R] : trace sequence number within the line
5. bytes 65 - 68 [O] : water depth at group
6. bytes 69 -70 [O] : scale to be applied to the depth
7. bytes 71 - 72 [O] : scale to be applied to the coordinates to give the real values (one of 1, 10, 100, 1000 or 10000)
8. bytes 81 - 84 [O] : x coordinate for the group (see note below)
9. bytes 85 - 88 [O] : y coordinate for the group (see note below)
   (note: the coordinates are stored as a 4 byte signed integer)
10. bytes 89 - 90 [O] : coordinate units (1= length (metres or feet)
    2 = seconds of arc)
11. bytes 115 - 116 [R] : number of samples in the trace
12. bytes 157 - 158 [O] : year data were recorded
13. bytes 159 - 160 [O] : day of year
14. bytes 161 - 162 [O] : hour of day ( 24-hour clock)
15. bytes 163 - 164 [O] : minute of hour
16. bytes 165 - 166 [O] : seconds of minute.

The **seismic amplitude sample** fields were not directly accessed as this information was not stored in the index data structures. The index data structure does, however, contain a pointer to the trace header record for the seismic

amplitude data. These pointers are used for retrieving the amplitude data for viewing the selected seismic lines (see section 6.2).

The last two sections are the end of tape marks **(EOF).** One **EOF** marker indicates the end of a seismic line (there may be part of, one, or more than one seismic line on a tape). Two **EOF** markers indicate the end of volume. For lines stored on disk these sections do not exist.

## 2.3 Constraints

The **SEG-Y** format specifies whether the fields in the various sections of the data are required, recommended or are optional. This means that the SEG-Y files have to be preprocessed in order to make sure that all the fields accessed for building the spatial index are populated with valid data. This preprocessing includes:

1. Storing the reference coordinates for the traces in the trace headers. The navigation data are normally collected and stored in a separate file in latitude and longitude format.
2. The method for storing the reference positions of the coordinates in arc seconds did not allow the navigation coordinates, which were recorded in decimal degrees and with 6 digits of precision after the decimal point, to be stored so that the full precision of the original navigation was retained. For this thesis the coordinates, in decimal degrees, were transformed so that the maximum latitude (+/-90 degrees), and the maximum longitude (+/- 180 degrees) could be

stored in a 32 bit signed integer. The coordinates were transformed as follows:

**scaled_longitude =(long int) ( longitude * largest_32_bit_integer )/ 180.0**

**scaled_latitude = (long int) ( latitude * largest_32_bit_integer ) / 90.0**

and the results stored as a 32 bit signed integer. This allowed the full accuracy of the original navigation data to be maintained.

3. For the experimental data used here, none of the seismic lines had the line number field set. Within the index data structure, each seismic line is identified by the address of the structure containing the data for the line (see section 3.2).

This preprocessing ensures that the recommended and optional fields in SEG-Y files are set to valid values. This reduces the chance of having problems when the data are loaded into the index data structures.

# CHAPTER 3

# SEISMIC INDEX DATA OBJECTS

## 3.1 Information Used in the Spatial Index

The index structures were designed to allow searching both in time and 2D space. In order to allow for these searches, the time that the trace was captured and the position of the trace were extracted from the SEG-Y file. Other extracted information includes the data necessary to determine the trace positions in the file and the format of the trace data.

## 3.2 Index Objects and Functional Diagrams

The SEISMIC INDEX data structure was designed and implemented using object oriented methods. A list of the operations that would be required to interact with the data was developed. The operations that were identified are as follows:

1) Initialize: set the data structure parameters to the initial settings.

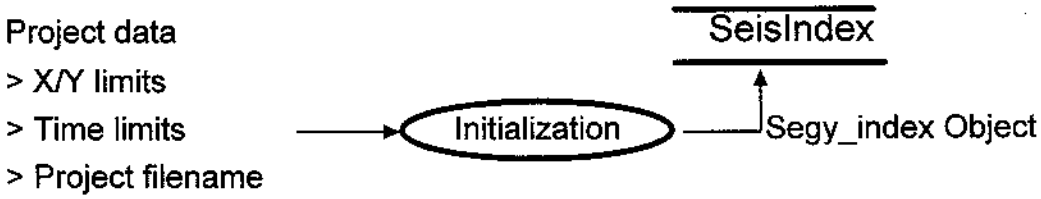



Figure 3-1. Function diagram for initialization of structure.

2) Insert seismic line: add a seismic line to the current structure.

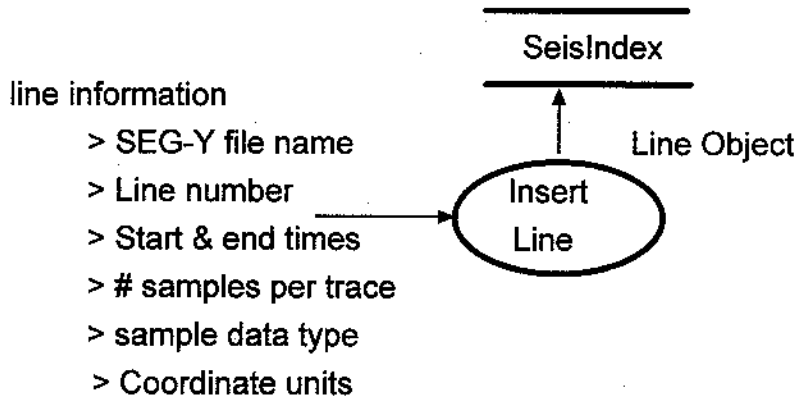



Figure 3-2. Insert line functional diagram.

3) Insert trace: add one trace of a seismic line to the current structure.

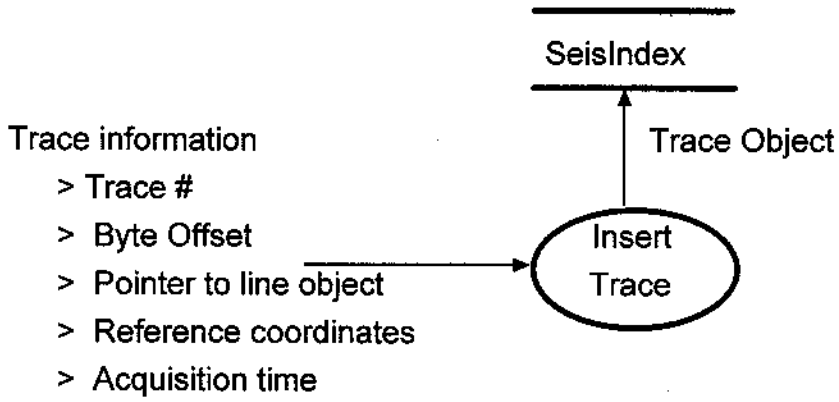



Figure 3-3. Insert trace functional diagram.

4) Delete: remove the specified seismic line from the data structure.

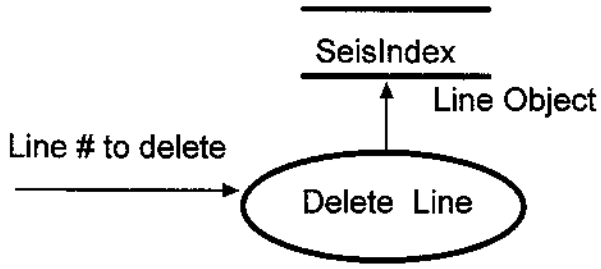



Figure 3-4. Delete line functional diagram.

5) 2D_Range_Search: return the lines that cross the specified region.

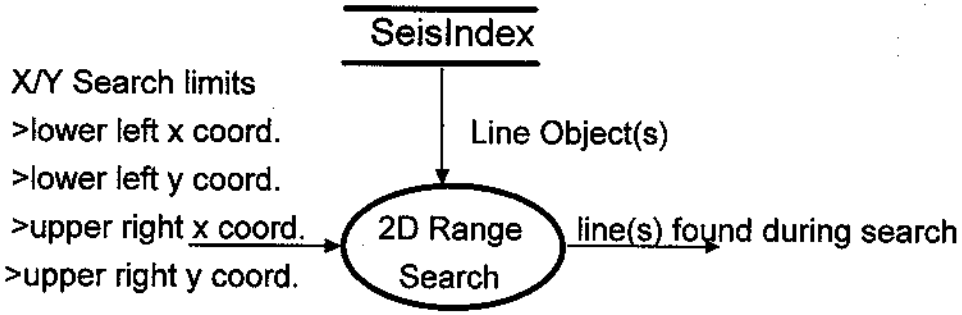



Figure 3-5. 2D range search functional diagram.

6) Time_Search: return the lines that were surveyed during the specified time period.

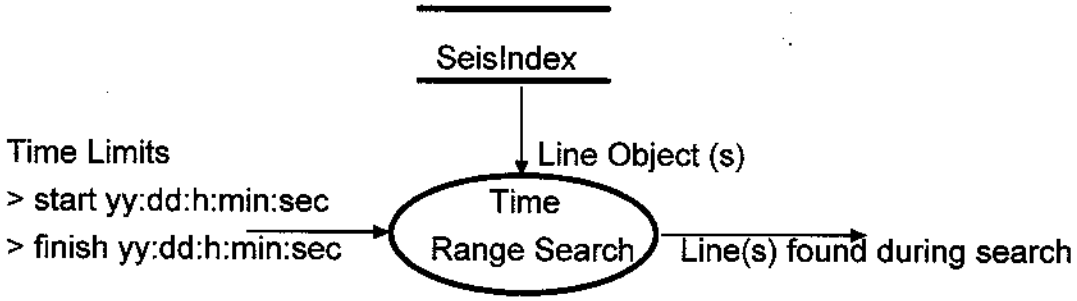



Figure 3-6. Time range search functional diagram.

7) Display Lines: display the specified line position data in a separate window from the trace data.

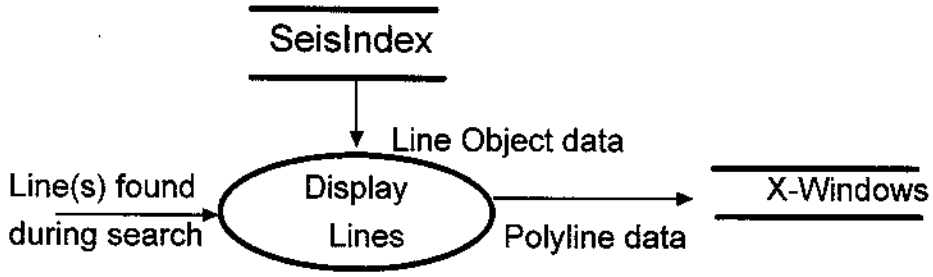



Figure 3-7. Display line objects functional diagram.

8) Display Traces: outputs the traces for the specified line to a TDR type file so the selected data can be displayed using the *FLEDERMAUS* 3D viewer [Paton, 1995].

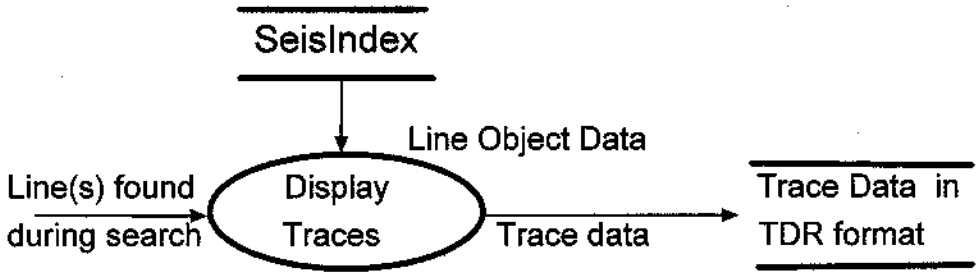



Figure 3-8. Display trace objects functional diagram.

Figure 3-9 shows the functional diagram for all the operations. The next sections define the objects within the "seismic index" data structure. The objects that will be covered in this chapter include the *Seismic Line* object, the *Seismic Trace* object, and the *Seismic Index* object. Sections 3.2.1 to 3.2.3 define these objects.

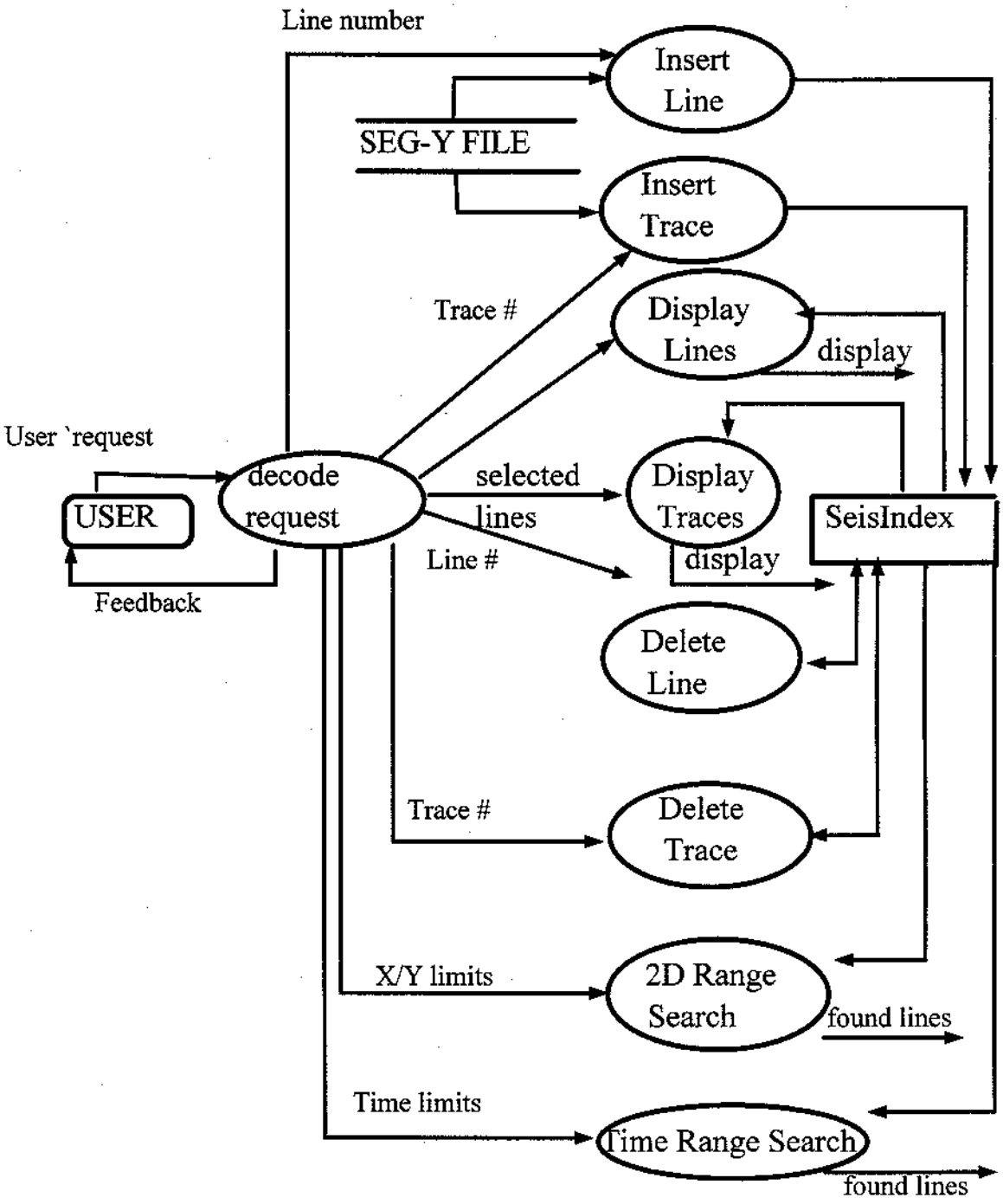



Figure 3-9. Functional diagram for the seismic index operations.

### 3.2.1 Seismic Line Object

The seismic line object contains general attributes and methods to store and manipulate the data for an instance of a line. As each line is loaded into the "seismic index" data structure the line object is created and the object attributes are updated until all the traces for the line have been loaded. The attributes that the object contains are as follows:

1. the seismic line filename complete with its path
2. an internally generated line number for the seismic line
3. the format for the trace amplitude data
4. the coordinate units
5. the time for the first trace in the line
6. the time for the last trace in the line.

The methods associated with the line object are:

1. Methods to store and return the filename for the seismic line
2. Methods to store and return the time for the first trace in the line
3. Methods to store and return the time for the last trace in the line
4. Methods to store and retrieve the data units
5. Methods to store and retrieve the coordinate type.

Figure 3-10 shows the object model for the seismic line class. Appendix C shows the complete C++ class definition.

| SEISMIC LINE | |
|---|---|
| Filename | : char * |
| Firstrace | : Seis_trace * |
| Line_number | : long |
| Data_type | : long |
| Units | : long |
| First_time | : unsigned long |
| First_year | : short |
| Last_time | : unsigned long |
| Last_year | : short |
| Xsouthwest | : double |
| Ysouthwest | : double |
| Xnortheast | : double |
| Ynortheast | : double |
| Add_fname | // add complete UNIX path name. |
| Get_fname | // return the complete UNIX path name. |
| Get_linum | // return the line number assigned to the line object. |
| Set_linum | // add the line number. |
| Get_units | // return the coordinate units. |
| Set_units | // add the coordinate units. |
| Get_Dtype | // return the trace amplitude data type. |
| Set_Dtype | // add the trace amplitude data type. |
| Add_Ftime | // add time for the first trace in yy:dd:hh:min:sec format. |
| Get_Ftime | // return time for the first trace in yy:dd:hh:min:sec format. |
| Add_Ltime | // add time for the last trace in yy:dd:hh:min:sec format. |
| Get_Ltime | // return time for the last trace in yy:dd:hh:min:sec format. |
| Drawline | // draw the seismic lines polyline in an X window. |
| Set_cover | // add the trace coordinates and update the line's cover |
| Get_cover | // return the seismic line's trace data cover. |

Figure 3-10. Seismic line object class model.

### 3.2.2 Seismic Trace Object

The seismic trace object contains specific attributes and methods to store and manipulate the data for an instance of a trace. As each trace is loaded into the

seismic index data structure, the trace object is created and the object attribute values are set. The attributes that the object contains are as follows:

1. a pointer to the seismic line object with which this trace is associated
2. the trace number that the trace was assigned within the SEG-Y file
3. the trace number that the trace was assigned within the "seismic index" structure
4. the byte offset from the beginning of the SEG-Y file to the beginning of the trace header record
5. the trace capture time expressed in seconds from 1900
6. the reference coordinates for the trace.

The methods associated with the line object are:

1. Methods to store and return the seismic line object pointer
2. Methods to store and return the trace's SEG-Y trace number
3. Methods to store and return the trace's "seismic index" trace number
4. Methods to store and retrieve the byte offset
5. Methods to store and retrieve the trace's coordinates
6. Methods to store and retrieve the trace capture time.

Figure 3-11 shows the object class model for the seismic trace class and Appendix D shows the complete C++ class definition.

| Seismic Trace Class | |
|---|---|
| line_obj | : Seis_line * |
| Trace_number | : long |
| traceId | : unsigned long |
| Byteoffset | : streampos |
| year | : int |
| Trace_time | : unsigned long |
| Xcord | : double |
| Ycord | : double |
| Zcord | : double |
| next | : Seis_trac * |
| Add_line_num | // add the pointer to the trace's seismic line object. |
| Add_trace_num | // add the trace's number within the seismic line. |
| Add_byte_pos | // add the byte offset to the trace header in the SEG-Y file. |
| Add_xyz | // add the trace's reference coordinates. |
| Add_time | // add the trace's time in yy:dd:hh:min:sec format. |
| SettraceCounter | // add the trace's seismic index trace number. |
| Get_line_num | // return the line pointer to the trace's seismic line object. |
| Get_trace_num | // return the trace's number within the seismic line. |
| Get_byte_pos | // return the byte offset to the trace header in the SEG-Y file |
| Get_xyz | // return the trace's reference coordinates. |
| Get_time | // return the trace's time in yy:dd:hh:min:sec format. |
| GettraceCounter | // return the trace's seismic index trace number. |

Figure 3-11. Seismic trace object class model.

### 3.2.3 Seismic Index Object

The seismic index object is the main class for the seismic index data structure. This object contains, as children, all the classes necessary to load the required SEG-Y files and perform the query operations to return the selected

lines. As the data are loaded into the seismic index data structure, the line and the trace objects are created and the data are inserted into the structures. At the same time, the seismic index attribute values are updated. The attributes that the object contains are as follows:

1. The file name of the Project Index file
2. The cover for the seismic index. This is the extent used by the PR quadtree (see section 4.3)
3. The tight cover for the lines that have been loaded from the project files
4. The maximum and minimum time for the data contained in the seismic index
5. Pointers to the AVL Time Tree (see section 4.1)
6. Pointer to the PR Quadtree (see sections 4.2 and 4.3)
7. Pointer to the Project file list class
8. An instance of the SGX class.

The Project and SGX classes are described in chapter 6. The methods associated with the seismic index object are:

1. Methods to insert and delete a seismic line from the structure
2. Method to perform the initialization of the class
3. Method to set and return the cover
4. Method to set, update and return the tight cover for the inserted seismic lines
5. Methods to set, update and return the time for the inserted seismic lines
6. Methods to display a seismic line or the results of a query
7. Method to perform a 2D range search or a time range search

8. Method to load a project or to create a project file

9. Method to save or read the seismic index structure to/ from disk.

Figures 3-12 and 3-13 show the object class model for the seismic index class and Appendix E shows the complete C++ class definitions.

| Seismic Index Class | |
|---|---|
| Filename | : char * |
| XswIndex | : double |
| XneIndex | : double |
| YswIndex | : double |
| YneIndex | : double |
| Depthmin | : double |
| Depthmax | : double |
| LineXsw | : double |
| LineXne | : double |
| LineYsw | : double |
| LineYne | : double |
| Timestart | : unsigned long |
| Timeend | : unsigned long |
| Number_of_lines | : int |
| Projectnames | : Index_Project_List |
| PruneTree | : int |
| Quadtree | : PR_Quadtree * |
| SearchTree | : Time_tree |
| SGX_IndexFile | : SGX_DiskIO |

Figure 3-12. Object model for the seismic index class.

| Seismic Index Class (cont.) | |
|---|---|
| Add_min_time | // add the minimum time for all the inserted data. |
| Add_max_time | // add the maximum time for all the inserted data. |
| Add_line_cover | // // add the cover for a seismic line and update the cover. |
| UpdateLineCover | // add and update the cover for the seismic line data. |
| UpdateTimeCover | // update the time data cover for the inserted data. |
| Set_proj_cov | // set the cover for seismic index. |
| Set_proj_tim | // set the minimum and maximum project times. |
| Incrm_num_line | // increment the number of lines that have been inserted. |
| UpdateIndexCover | // update the cover for the seismic index. |
| RtnIndexCover | // return the current seismic index data cover. |
| Get_proj_cov | // return the current project data cover. |
| Get_proj_tim | // return the minimum and maximum project times. |
| DisplayLINES | // display in an X window the selected line(s). |
| DisplaySHOTS | // write the selected trace data to a TDR file. |
| LoadProject | // load the information specified in a project file. |
| LoadSeismicIndex | // load and build the seismic index from the SGX file. |
| SaveProject | // create a seismic index project file. |
| SaveSeismicIndex | // create and save the seismic index in an SGX file. |
| Initialize | // initialize the seismic index environment. |
| Insert | // insert a seismic index line. |
| Delete | // delete a seismic index line. |
| Range_Search_2D | // perform a 2D range search on the PR quadtree. |
| Time_Search | // perform a time range search on the AVL time tree. |

Figure 3-13. Object model for the seismic index class (cont.).

### 3.2.4 Object Hierarchy

Figure 3-14 is an OMT [Rumbaugh et al, 1991] class diagram which shows the relationships among the seismic index objects.

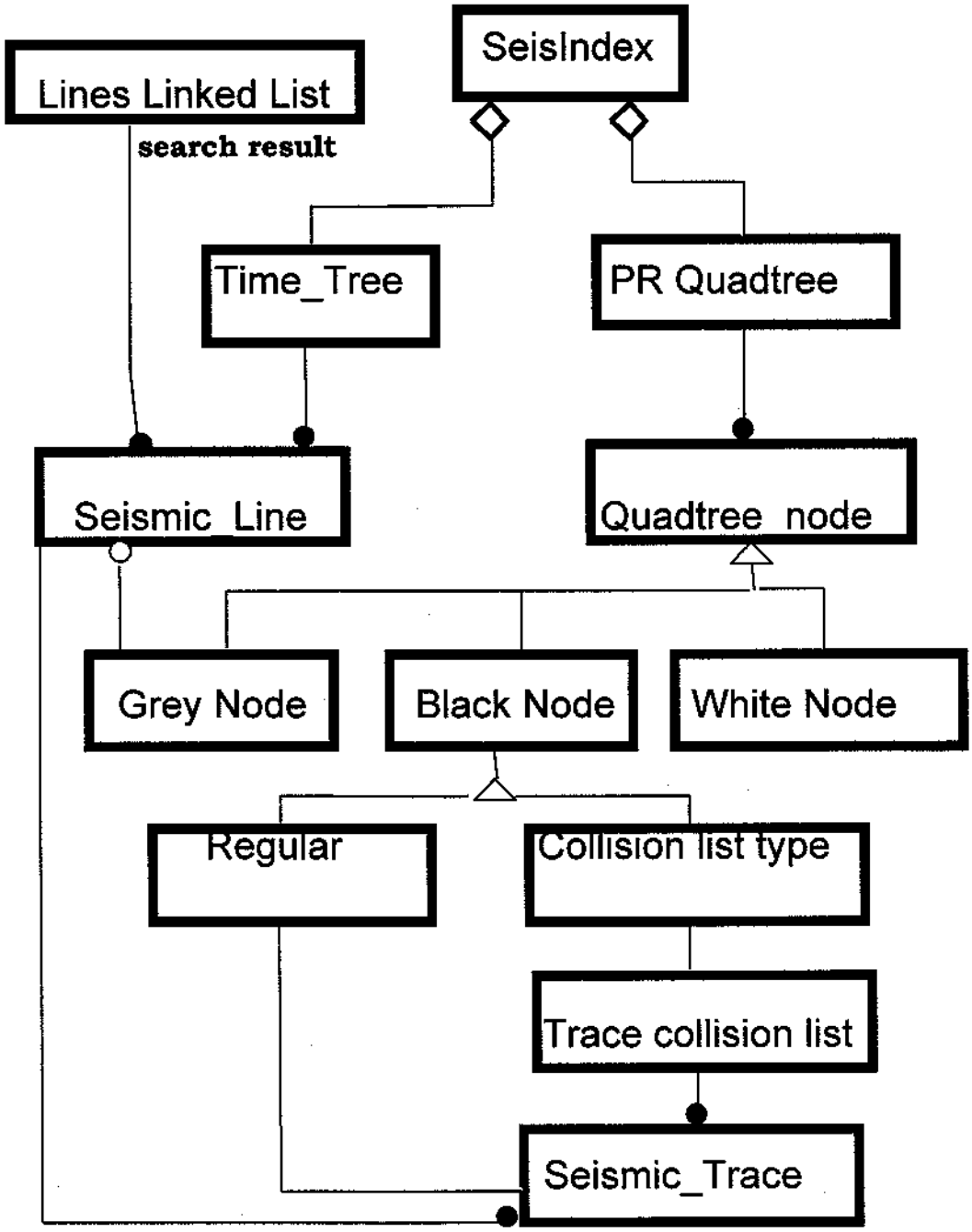



Figure 3-14. Object model for the SEG-Y seismic index object.

# CHAPTER 4

# SEISMIC INDEX DATA STRUCTURES

## 4.1 Seismic Index Data Structures

In chapter 3 the object classes that are used to index the raw SEG-Y seismic files were described. The data structures, which are used to index these objects so that time and 2D range queries can be performed, are described in this chapter. There are four types of structures used. The first is the pointer array which is used to store the address of the objects as they are created in the process of building the index from an SGX file (see section 6.4).

The second type is the linked list. These are used in several places. The first is to link all the traces together and bind them to the seismic line to which they belong. The second is in the quadtree structure which is used to store the traces. As duplicate coordinates are possible for the trace data at each node, a collision list is maintained of traces which fall within a cell. Section 4.3 describes this structure in more detail.

The other two basic types of structures used are the AVL binary search tree and the PR quadtree [Samet, 1990]. Section 4.2 describes the exact

structure of the AVL tree that was used. Section 4.3 describes the standard PR quadtree and section 4.4 describes the modified PR quadtree.

## 4.2 AVL Time Tree

The AVL time tree class was designed from an AVL binary search tree. This structure is used to organize the seismic line object class so that time range queries can be performed. The time of the first trace, in seconds from 1900, was used as the discriminator at a node to determine into which subtree the seismic line object should be placed. As there was a possibility that line objects could have duplicate first times, a policy that the duplicate time should be placed in the left subtree was adopted. The two classes which comprise the AVL Time Tree structure are described in sections 4.2.1 and 4.2.2.

### 4.2.1 Time Tree Object Class

The first object class is the container class, called the Time Tree class, through which requests to insert, modify or retrieve data are sent. This class only has one attribute, this being a pointer to the root of the AVL Time Tree. This class supports the following operations:

1. insert and remove a seismic line object from the tree
2. perform time range searches
3. read and write the structure to an SGX disk file (see section 6.4).

Figure 4-1 shows the object model for the Time Tree class and Appendix F shows the complete C++ class definition.

| Time Tree Object Class | |
|---|---|
| tree_node | Avl_node * |
| Insert | // insert a seismic line object into the AVL Time Tree. |
| RemoveAVL | // delete a seismic line object into the AVL Time Tree. |
| Time_search | // search the Time Tree using time for valid seismic line objects. |
| Remove | // delete the entire AVL Time Tree. |
| Range_search_2D | // perform a 2D range search on all the line objects in the tree. |
| Writefilenames | // write the seismic line UNIX path names to a project file. |
| ReDrawLines | // update an X windows display from all the seismic line objects. |
| WriteSeismicIndex | // write the AVL Time Tree to a seismic index file (SGX). |
| ReadSeismicIndex | // build the AVL Time Tree from a seismic index file (SGX). |

Figure 4-1. Object model for the Time Tree object class.

### 4.2.2 AVL Node Object Class

The second object class of the time tree is the AVL node object class. This class contains the line object and forms the nodes within the AVL Time Tree. The class contains the following attributes:

1. a pointer to the left and right children
2. a pointer to the seismic line object.

The class supports the following operations:

1. 2D range search
2. read from and write to an SGX file (see section 6.4)

3. write data to a project file
4. return line object address, start and end times and line number.

Figure 4-2 shows the object model for this class. Appendix F shows the complete C++ class definition for these objects.

| AVL Time Tree Node Object Class | |
|---|---|
| Left | Avl_node * |
| Right | Avl_node * |
| Height | int |
| lineobj | Seis_line * |
| Avl_node | // create a Time Tree Node. |
| Get_linum | // return the seismic line object's line number. |
| Get_stime | // return the minimum time in seconds for the line object. |
| Get_etime | // return the maximum time in seconds for the line object. |
| Range_search_2D | // perform a 2D range search on the seismic line object. |
| Writefilenames | // write the node information to the project file. |
| ReDrawLines | // redraw the seismic line object in an X window. |
| WriteSeismicIndex | // write the node information to an SGX file. |
| ReadSeismicIndex | // create a node from the data read from an SGX file. |

Figure 4-2. Object model for the AVL Time Tree node object class.

## 4.3 Standard PR Quadtree

The third data structure is the standard PR quadtree. This data structure is used to store the data for each instance of a trace. The reference coordinates for the traces, in double format, are used to determine into which subquadrant a

trace is to be inserted. The extent for the quadtree is set when an instance of the object is created. Once set, the extent cannot be changed without rebuilding the quadtree.

### 4.3.1 PR_Quadtree Object Class

The PR quadtree is built from two object classes. The first is the ***PR_Quadtree*** object class. This class is a container class through which all requests to insert, modify or retrieve data are sent. The attributes that this class supports are:

1. centre and the extent of the PR quadtree
2. bounding coordinates for the PR quadtree's cover. This is used to determine if a point falls outside the limits of the quadtree and to prevent infinite recursive splitting of the quadrants during the insertion of points that are outside the bounds of the quadtree.
3. pointer to the root of the PR quadtree.

This class supports the following operations:

1. insert and delete a trace
2. perform a 2D range search
3. read from and write the PR quadtree structure from an SGX file (see section 6.4).

Figure 4-3 shows the object model for this class and Appendix G shows the complete C++ class definition.

### 4.3.2 PR_Quadtree Node Object Class

The second class is the ***PR_QTnode*** class. This class forms the nodes within the PR quadtree. The nodes contain a pointer to the trace or traces which have been inserted into a quadrant. With seismic data there is a possibility that there may be duplicate coordinates due to the nature by which the seismic data are collected. To accommodate duplicate coordinates, a collision list is maintained at each leaf node into which multiple trace objects are inserted. This list is built from the ***NodePoint*** class.

| **PR_Quadtree Object class** | |
|---|---|
| Rootnode | : PR_QTnode * |
| Xcentre | : double |
| Ycentre | : double |
| Xrange | : double |
| Yrange | : double |
| southWestX | : double |
| southWestY | : double |
| northEastX | : double |
| northEastY | : double |
| PR_Quadtree | // initialize the PR quadtree. |
| PR_QTcompare | // compare the inserted trace to the current cell. |
| PR_QTinsert | // insert a trace into the quadtree. |
| PR_QTdelete | // delete a trace from the quadtree. |
| PRQTsearch | // search the quadtree for lines satisfying the query. |
| Remove | // remove the contents of the entire quadtree. |
| CQUAD | // determine the next quadrant in a clockwise direction. |
| CCQUAD | // determine the next quadrant in a counter clockwise direction. |
| WriteSeismicIndex | // write the quadtree to an SGX file. |
| ReadSeismicIndex | // create the quadtree from the SGX file data. |

Figure 4-3. Object model for the PR_Quadtree object class.

The ***NodePoint*** class contains two fields. The first is a pointer to an instance of a trace object. The second is a pointer which points to the next

instance of a ***NodePoint*** object or NULL if there are no more traces with identical coordinates in the collision list.

The PR quadtree node class contains the following attributes:

1. pointers to the four children
2. the node type (grey or black; white is implied by nil pointers in a grey node)
3. pointer to the ***NodePoint*** class object.

The operations that this class support are:

1. 2D range search
2. functions to determine the pointer to a child given the quadrant
3. return the pointer to the seismic line object
4. return the coordinates of the trace
5. add a trace's line to the list of valid lines for the 2D range search
6. write to and read the node information from an SGX file.

Figure 4-4 shows the object model for the PR quadtree node class and Appendix H shows the complete C++ class definition.

## 4.4 Seismic Index PR Quadtree

The seismic index PR quadtree, also known as the modified PR quadtree, is derived from the standard PR quadtree. Within this quadtree a mechanism has been added to keep track of which line a trace or group of traces are associated with. By using this association, pruning can be performed on portions of the tree within which all traces belong to the same seismic line. At the interior (grey) nodes, a field is set depending on whether the node subtrees contain traces that belong to one or more seismic lines. If the traces in the subtrees

belong to one and only one line, then the unique line number is inserted into this field. If the traces belong to more than one line, then an invalid value is inserted into this field so that pruning will not be performed.

| PR Quadtree Node Class | |
|---|---|
| SWnode | : PR_QTnode * |
| NWnode | : PR_QTnode * |
| NEnode | : PR_QTnode * |
| SEnode | : PR_QTnode * |
| Nodetype | : short |
| Xcoord | :double |
| Ycoord | : double |
| DataPTR | : PtrToData |
| AddAllchildren | // add all the line objects in the current subtree to the search line list. |
| Addchild | // add the trace's line object to the search line list. |
| Addnode | // add a new node at the specified quadrant. |
| ADJQUAD | // determine the adjacent quadrant from the current quadrant. |
| CQUAD | // determine the quadrant clockwise from the current quadrant. |
| CCQUAD | // determine the quadrant counter clockwise from the current quadrant. |
| ChildType | // determine node type for the specified child. |
| CreateNode | // create a new node of the specified type. |
| RtnChild | // return a pointer to the specified child. |
| RtnType | // returns the nodes type (grey or black). |
| RTNxcord | // returns the nodes X coordinate. |
| RTNycord | // returns the nodes Y coordinate. |
| RtnSeisLine | // returns a pointer to the seismic line object. |
| SetChild | // inserts a new node in the specified quadrant. |
| RangeSearch | // recursively perform a 2D range search. |
| WriteSeismicIndex | // write the quadtree node's data to an SGX file. |
| ReadSeismicIndex | // read the nodes data from an SGX file. |

Figure 4-4. Object model for the PR quadtree node object class.

In the modified PR quadtree class, no additional attributes have been added. The existing data pointer field is used to store the additional information. This additional information is a pointer to the address of the trace's line object. The class requires several new methods to set and update the line object pointer field for the interior nodes during insertion. The pruning operations are handled in the PR quadtree node object class. Figure 4-5 shows the object model for the modified PR quadtree class. The modified PR quadtree node class is the same as the PR quadtree node class, except that the AddAllChildren method has been changed so that pruning is performed.

| **Modified PR_Quadtree Object class** | |
|---|---|
| Rootnode | : PR_QTnode * |
| Xcentre | : double |
| Ycentre | : double |
| Xrange | : double |
| Yrange | : double |
| southWestX | : double |
| southWestY | : double |
| northEastX | : double |
| northEastY | : double |
| PR_QTcompare | // compare the trace being inserted with the current node. |
| PR_QTinsert | // insert a trace into the PR quadtree. |
| PRQTsearch | // perform a 2D range search. |
| Remove | // delete the entire PR quadtree. |
| EqualKey | // determine if two coordinates are the same. |
| CQUAD | // determine the adjacent quadrant in a clockwise direction. |
| CCQUAD | // determine the adjacent quadrant in a counter clockwise direction. |
| WriteSeismicIndex | // write the PR quadtree out to an SGX file. |
| ReadSeismicIndex | // build the PR quadtree from the data in an SGX file. |
| **CheckLineID** | **// determine if the current node and the new trace belong to the // same seismic line.** |
| **VerifyLineID** | **// update the seismic line pointer field for the current node.** |

Figure 4-5. Object model for the modified PR_Quadtree object class.

Figure 4-6 shows an example of three seismic lines. Figure 4-7 shows the modified PR quadtree for figure 4-6. In figure 4-7, the subtrees of the northeast quadrant all contain traces that belong to line A, so the interior node line object field has been set to line A. In the southwest quadrant the subtree contains traces that belong to lines B and C. In this case, the interior nodes in this portion of the quadtree are all set to null.

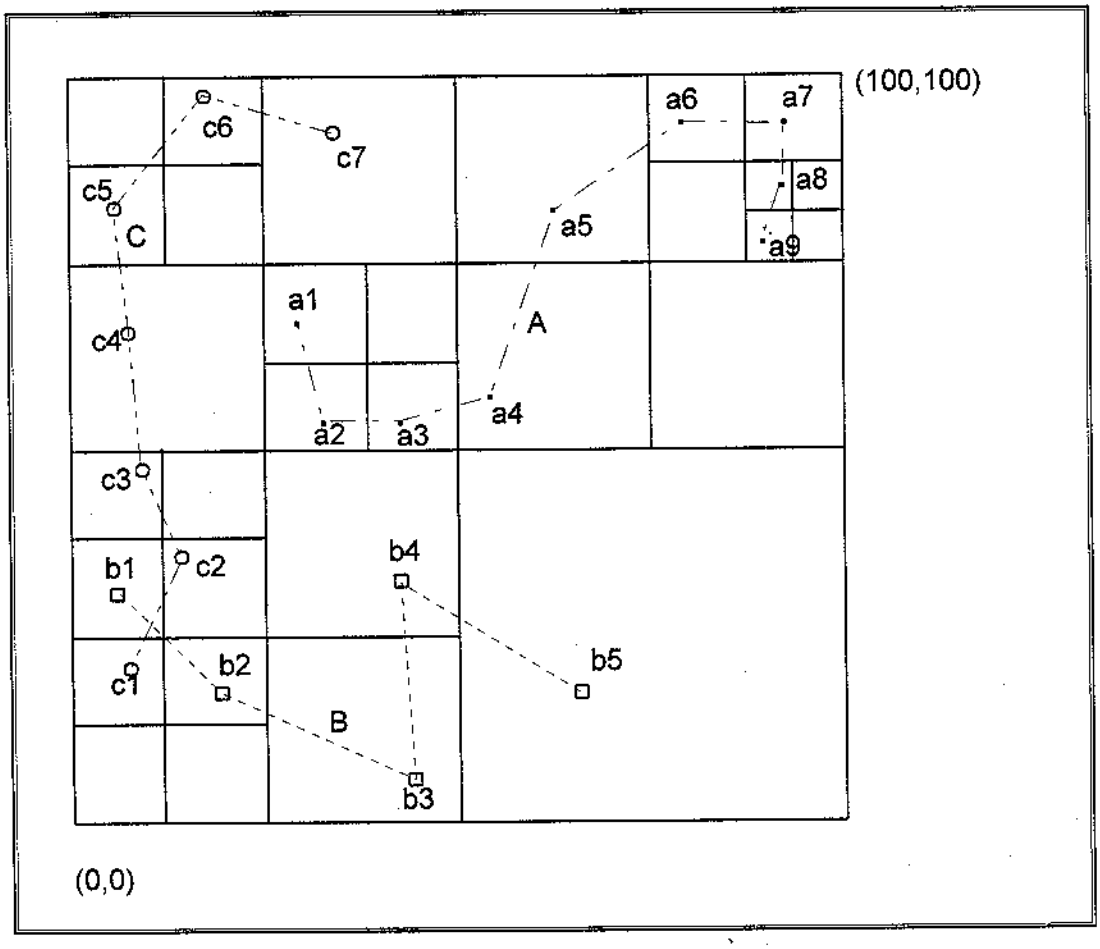



Figure 4-6. Example of 3 seismic lines.

The use of the modified PR quadtree object class allows, in some cases, extensive pruning of large sections of the quadtree thus improving the search

times. Chapter 5 describes the insert and search algorithms that use the modified PR quadtree while chapter 7 shows tests using the standard and the modified PR quadtree structures.

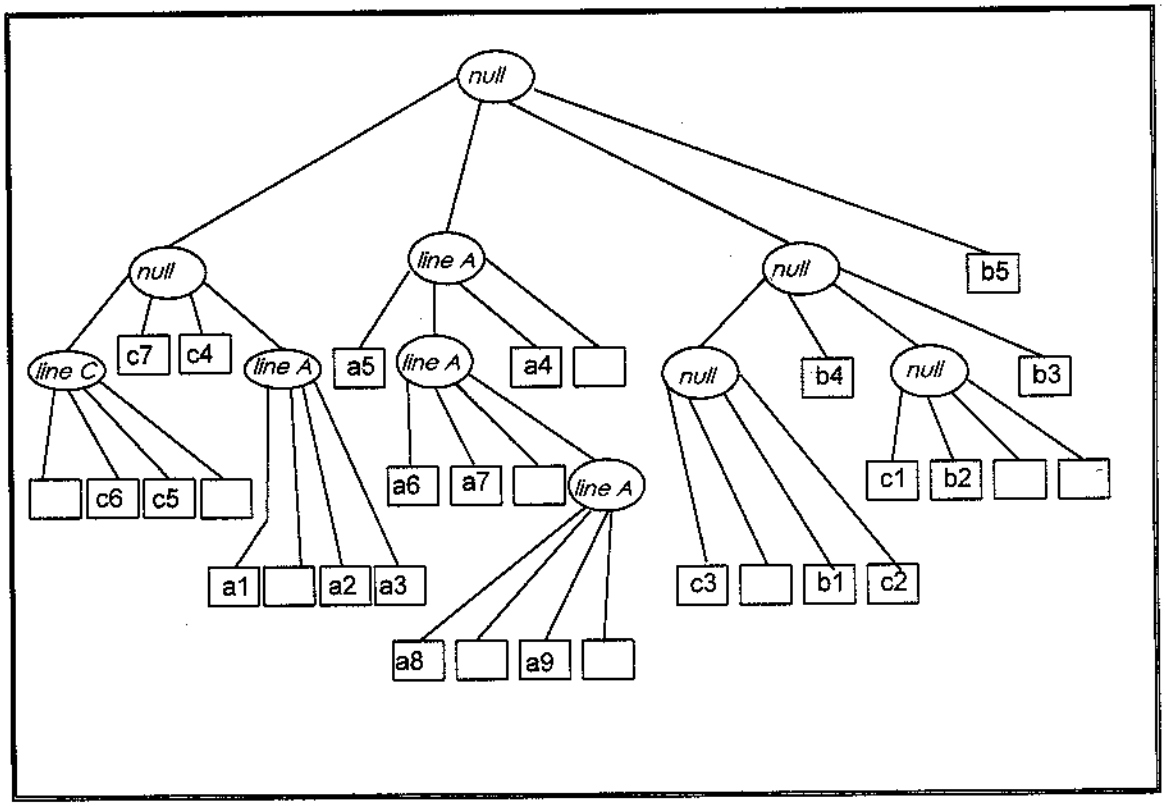



Figure 4-7. Example of the modified PR quadtree.

# CHAPTER 5

# SEISMIC INDEX PR QUADTREE ALGORITHMS

## 5.1 Insertion

The insert algorithm for the modified seismic index PR quadtree keeps track of when and if the seismic line object pointer field in the grey nodes of the quadtree needs to be set or modified. This algorithm is very similar to the standard PR quadtree insertion [Samet, 1990], except for keeping track of the line pointers. Figure 5-1 is a top level overview of the insertion algorithm. The input argument for PR_QTinsert is a trace object as discussed in section 3.2. The variables Xcentre, Ycentre, Xrange, Yrange, southWestX, southWestY, northEastX and northEastY apply to the entire PR quadtree. They are defined when an instance of the modified quadtree object class is created (see figure 4-5). In figures 5-1 to 5-4, the pseudocode follows the style used by Samet [1990] which is a variant of ALGOL [Naur, 1960].

There are three cases that can occur. In the base case, where the quadtree is empty, a black leaf node is created and the address of the trace object is placed in the data pointer field. In figure 5-2, lines 43 to 46 show the pseudocode for the base case.

```
1   preload real array XF ['NW', 'NE', 'SW', 'SE'] with -0.25, 0.25, -0.25,  0.25 ;
2   preload real array YF ['NW', 'NE', 'SW', 'SE'] with  0.25, 0.25, -0.25, -0.25 ;
3   procedure PR_QTinsert(Seis_trac * newtrace      )
4   // PR quadtree insertion algorithm.
5   // Xcent and Ycent are the centroid of the current cell in the quadtree
6   // LengthX and LengthY are the length of the sides of the current quadtree node.
7   // TmpParent is the pointer to the current node in the quadtree that is being evaluated.
8   // childnode is a pointer to the next node in the tree that will be traversed.
9  // Unode is the temporary node used in splitting a quadrant during the insertion.
10  begin
12   .   value double Xcent, Ycent, LengthX, LengthY;
13   .   value pointer PR_QTnode TmpParent;
14   .   value boolean rtncode ;
15   .  // initialize the quadrant centre and quadrant length to the PR quadtree class values.
16   .   Xcent ← Xcentre;
17   .   Ycent ←Ycentre;
18   .   LengthX ← Xrange;
19   .   LengthY ← Yrange;
20   .   // test to see if the trace falls outside the PR quadtrees extent
21   .   if XCOORD(newtrace) < southWestX  or YCOORD(newtrace) < southWestY or
22   .     XCOORD(newtrace) > northEastX  or YCOORD(newtrace) > northEastY
23   .    then return;

24   .   // call the base case method if the tree is empty or only one leaf node in the tree.
25   .   if null (Rootnode)  or  not (GREY (Rootnode) ) then
26   .      rtncode ← RootNodeBaseCase(Rootnode , newtrace);
27   .      if rtncode = FALSE then return ;
28   .   end if

29   .  // copy the root node and pass this variable to the function that walks the tree.
30   .   TmpParent ← Rootnode ;                    // copy the current parent node

31   .   // call the method to walk the tree and determine the quadrant should be placed.
32   .   // The method  will split nodes if required.
33   .  PR_QTWalk(TmpParent, Xcent, Ycent, LengthX, LengthY );

34  end
```

Figure 5-1. Top level overview of the insert algorithm.

```
35  boolean procedure RootNodeBaseCase (Rootnode * PR_QTnode , newtrace *Seis_trace) ;
36  // this method handles the base case where the tree is initially empty and the case where the
37  // tree contains only one leaf node. The input arguments are pointers to the rootnode which is
38  // modified as required and the trace to be inserted into the tree..
39    begin
40    .  value integer quad ;
41    .  value pointer Seis_line LineID;
42    .  value pointer PR_QTnode Unode;
43    .  if null (Rootnode) // if the tree is initially empty the create the root node.
44    .     Rootnode ←PR_QTnode(Black, newtrace); // call the node constructor.
45    .     return FALSE;
46    .
47        else if not (GREY (Rootnode) ) then // if the tree contains one node leaf node
48    .  begin
49    .  .  if EqualKey( (XCOORD(newtrace), YCOORD(newtrace), XCOORD(Rootnode),
50    .           YCOORD(Rootnode) )then // if this is a duplicate point
51    .  .  begin
52    .  .     Rootnode ← Addnode(newtrace);  // Add the node to the list of traces at this
node.
53    .  .     return FALSE;
54    .  .  end
55    .     else // else process the input node
56    .  .  begin
57    .  .  .  // Call the function to determine if the traces are from the same line.
58    .  .  .  LineID ← CheckLineID(Rootnode, newnode);
59    .  .  .  Unode  ← Rootnode;
      .  .  .
60    .  .  .  if not null (LineID) then // line ID's are the same add the line ID to the data
field.
61    .  .  .  .  Rootnode ← PR_QTnode(Grey, Xcentre, Ycentre, LineID);
62    .  .  .  else          // create the new root with line ID field set to null.
63    .  .  .  .  Rootnode ← PR_QTnode(Grey, Xcentre, Ycentre);
64    .  .  .  end if
      .  .  .
65    .  .  .  // determine the quadrant that the original parent falls in
66    .  .  .  quad ← PR_QTcompare(Unode, Xcent, Ycent );
      .  .  .
67    .  .  .  // Assign the original parent to the correct child of the root based on the
quadrant.
68    .  .  .  Rootnode ← SetChild(quad, Unode);
69    .  .  end if
70    .  end if
71    .  return TRUE ;
72    end
```

Figure 5-2. RootNodeBaseCase procedure for the insertion algorithm.

```
    procedure PR_QTWalk(TmpParent, Xcent, Ycent, LengthX, LengthY );
74  // this procedure accepts a pointer to the root node and the centre and extent of the entire
75  // modified PR quadtree. The tree will then be walked untill the quadrant that the new trace
76  //should be inserted is determined. The trace is then inserted into the tree. As the tree is
77  // traversed the line pointer field at the current grey node is set to NULL if the new trace an
78  // the  trace at the current node belong to different lines,
79  begin
80  . value pointer Seis_line LineID;
81  . value pointer PR_QTnode Cntnode, childnode
82  . value integer quad, quadUnode;

83  . // if the new trace and the trace at the current node are from different lines then set the
84  . VerifyLineID(CntNode, newnode);  // current nodes line pointer feild to null.
85  . quad ← PR_QTcompare(newnode, Xcent, Ycent) : // determine the newnode's quadrant
86  . // In a loop walk the tree until the correct place for the node is determined.
87  . childnode ←CntNode.RtnChild(quad,);
88  . if not null(childnode) Type ← childnode.RtnType(); // determine the node type  for the
child.

89  . // while the child node is not null and it is an interior node walk the tree.
90  . while not null(childnode ) and GREY(childnode) do // locate the new traces quadrant.
91  . begin
92  . .   CntNode ← CntNode.RtnChild(quad); // Step to the next node in the quadtree.

93  . .   // Update the value of the centre and the extent for the new current node.
94  . .   Xcent ← Xcent + XF[quad] * LengthX;          LengthX ← LengthX / 2.0;
95  . .   Ycent ← Ycent + YF[quad] * LengthY;          LengthY ← LengthY / 2.0 ;

96  . .   quad = PR_QTcompare(newnode, Xcent, Ycent ); // get next the quadrant for the trace.
97  . .   childnode ← CntNode,RtnChild(quad);          // Update the child pointer and the type
98  . .   if not null(childnode) Type ← childnode.RtnType();
99  . .   VerifyLineID(TmpParent, newnode);  // Update the line object pointer field.
100 . end while

101 . // if the child pointer is null insert the node in the current node's specified child.
102 . if null (childnode ) then
103 .    newnode ←PR_QTnode(Black, newtrace);      // call the node constructor.
104 .    CntNode ←SetChild(quad, newnode);
105 . // else if the node is already occupied with a trace with duplicate coordinates then add the
106 . else if XCOORD(newtrace) = XCOORD(childnode) and // trace the collision list.
107 .         YCOORD(newtrace) = XCOORD(childnode) then
108 .    childnode ← Addnode(newtrace);
109 .    return ;
```

Figure 5-3. Procedure to handle case 3 of the modified PR quadtree insertion algorithm.

```
110 . else // the node is already occupied so subdivide it until the node can be placed in the tree.
111 . begin
112 .    CntNode,RtnChild(quad, Unode );
113 .    LineID ← CheckLineID(Unode, newnode); // determine the LineID variable.
114 .    newnode ←PR_QTnode(Black, newtrace);      // call the node constructor.

115 .    do // repeatedly subdivide quadrant until Unode and newnode are in different
quadrants.
116 .    begin
117 .    .      if null ( LineID) then // create a node with line object field set to null.
118 .    .         Cntnode ← CreateNode(quad, Grey, Xcent + XF[quad] * LengthX,
119 .    .                        Ycent + YF[quad] * LengthY);
120 .    .  else
121 .    .      .  Cntnode ← CreateNode(quad, Grey, Xcent + XF[quad] * LengthX,
122 .    .      .                              Ycent + YF[quad] * LengthY, LineID);
123 .    .      end if

124 .    .      CntNode ← RtnChild(quad, CntNode);// update the parent pointer.
125 .    .      Xcent ← Xcent + XF[quad] * LengthX; LengthX ←LengthX / 2.0;
126 .    .      Ycent ←Ycent + YF[quad] * LengthY;  LengthY ← LengthY / 2.0;

127 . .      // determine the quadrant for the new node and the existing tree node.
128 .    .      quad ← PR_QTcompare(newnode, Xcent, Ycent ); determine next quadrant.
129 .    .      quadUnode ←PR_QTcompare(Unode, Xcent, Ycent );

130 .    until not (quad = quadUnode )

131 .    // reinsert the existing node and the new node.
132 .    CntNode ← SetChild(quad, newnode); // insert the children in the correct positions
133 .    CntNode ← SetChild(quadUnode, Unode);
134 . end if
135 end PR_QTinsertLine
```

Figure 5-4. Procedure to handle case 3 of the modified PR quadtree insertion algorithm (cont.).

In the second case only the root node is occupied. If the traces have duplicate reference positions, then the new trace is inserted in the collision list for the root node. Otherwise, a grey node is created and this becomes the root node. If the trace in the previous root and the new trace belong to the same

seismic line, the seismic line pointer is set to the address of the line object. The quadrant into which the previous root node should be placed is determined, and the node is assigned to the appropriate child in the new root. The trace to be inserted is then processed using the third case of the insertion algorithm. In figure 5-2, lines 47 to 72 show the pseudocode for case two of the insertion algorithm.

In the third case the tree has to be walked to determine which cell within the PR quadtree the trace is to be inserted. As the path from the root to the target cell is traversed, each grey node for which the seismic line pointer field has been defined is checked to determine if the seismic line objects from the current node and the trace being inserted are the same. If they are the same, no action is taken. If they are different, the data pointer field is set to its invalid state.

Once the target cell has been reached, the trace is inserted. If the new trace has duplicate reference coordinates, then the trace is placed in the collision list. If the target quadrant is vacant, then the trace is inserted into the new child. If the target quadrant is occupied, then the quadrant is recursively split until the new trace can be placed in a separate cell. As the splitting is being performed, the split node's seismic line object pointer field is set as required. Figure 5-3 shows the pseudocode for case 3 of the insertion algorithm. Appendix I shows the complete C++ implementation of the insertion algorithm.

## 5.2 Search

The search algorithm is expected to be faster because in some cases it allows pruning of large sections of the PR quadtree. A rectangular query window defines the search region. The quadtree is then traversed and at each node in

the search path, the four children are tested to determine if all or part of the child falls within the query window. If a child does not overlap the search window, the search of that path is terminated.

In the case where a child intersects or is enclosed by the query window, the seismic line pointer field is checked. If the field is undefined, then searching continues down the child's subtree. If the seismic line pointer field is set, then the seismic line is added to the list of lines that satisfy the query. At this point, searching of this node's children is terminated and the search is continued at the last previous level in the quadtree. Another feature of the search algorithm is that it terminates the search testing when a cell is completely within the search window, and picks up the seismic lines that are contained in the current subtree. The search algorithm for searching the modified PR quadtree is shown in figure 5-5 and the C++ implementation is shown in Appendix J.

```
procedure PRQTsearch (line_list *line, XYZpnt SWcorner, XYZpnt NEcorner)
begin
   if not null(Rootnode)then
      Root.RangeSearch(line, SWcorner, NEcorner, Xcentre, Ycentre, Xrange, Yrange );
   end if
   return;
end
procedure RangeSearch(line, SWcorner, NEcorner, Xcent, Ycent, Xlen, Ylen); // method to
   // recursively perform the 2D range search on the node. Inputs are the query window corners
   // and the centroid and length of the current node.
value XYZpnt SWcorner, NEcorner;
value double Xcent, Ycent, Xlen, Ylen;
begin
   value integer code;
   value double xmin, xmax, ymin, ymax, Qxsw, Qysw, Qxne, Qyne, Z;
   value pointer child;
   if BLACK(Nodetype) // if this is a leaf node then add the line to the list
   begin
   .  if Xcoord >= XCOORD(SWcorner) and Xcoord <= XCOORD(NEcorner) and
   .  .  Ycoord >= YCOORD(SWcorner) and Ycoord <= YCOORD(NEcorner) then
   .  .  Addchild(line); // add the line to the list of selected lines.
   .  .  return
   .  else
   .  .  return    // point falls outside the query window so return.
   .  end if
   end
   xmin ← Xcent - Xlen / 2.0;      ymin ← Ycent - Ylen / 2.0; // determine the extent of
   xmax ← Xcent + Xlen / 2.0;      ymax ←Ycent + Ylen / 2.0; // the quadrant;
   for child ← (NW, NE, SW and SE )   // for the four children
   .  if not null (child)
   .  .  // if the query window overlaps the quadrant then recursively check the children
   .  .  code ← child.checkquad(xmin, Ycent, Xcent, ymax, SWcorner, NEcorner);
   .  .  // if the quadrant is inside the query window, then return all children
   .  .  if (code = INSIDE ) then
   .  .  .  child.AddAllchildren(SWcorner, NEcorner, line);
   .  .  else if (code = OVERLAP) then // if the query window overlaps then walk the subtree.
   .  .  .  child.RangeSearch(line, SWcorner,  Necorner,  Xcent + XF[child] * Xlen,
   .  .                          Ycent + YF[child] * Ylen,  Xlen / 2.0,  Ylen / 2.0);
   .  .  end if
   .  end if
   end for
   return
end RangeSearch
```

Figure 5-5. Modified PR quadtree search algorithm.

Figure 5-6 shows an example of three seismic lines each with several traces. Figure 5-7 shows the modified seismic index PR quadtree for Figure 5-6. In this example, the grey nodes in the north east quadrant and its subtree have the seismic line pointer fields pointing to line ***A***. If the query window encompassed this quadrant then only the first level would have to be checked. The remaining subtree would be eliminated from the search by the pruning operation. For example, if a search was performed with a query window ([50,50], [100,100]), the pruning would result in the search being terminated at level 1 with line ***A*** being returned. If a search was performed with a query window of ([5,5], [45,45]) pruning would not occur at all and the search would continue to the leaves.

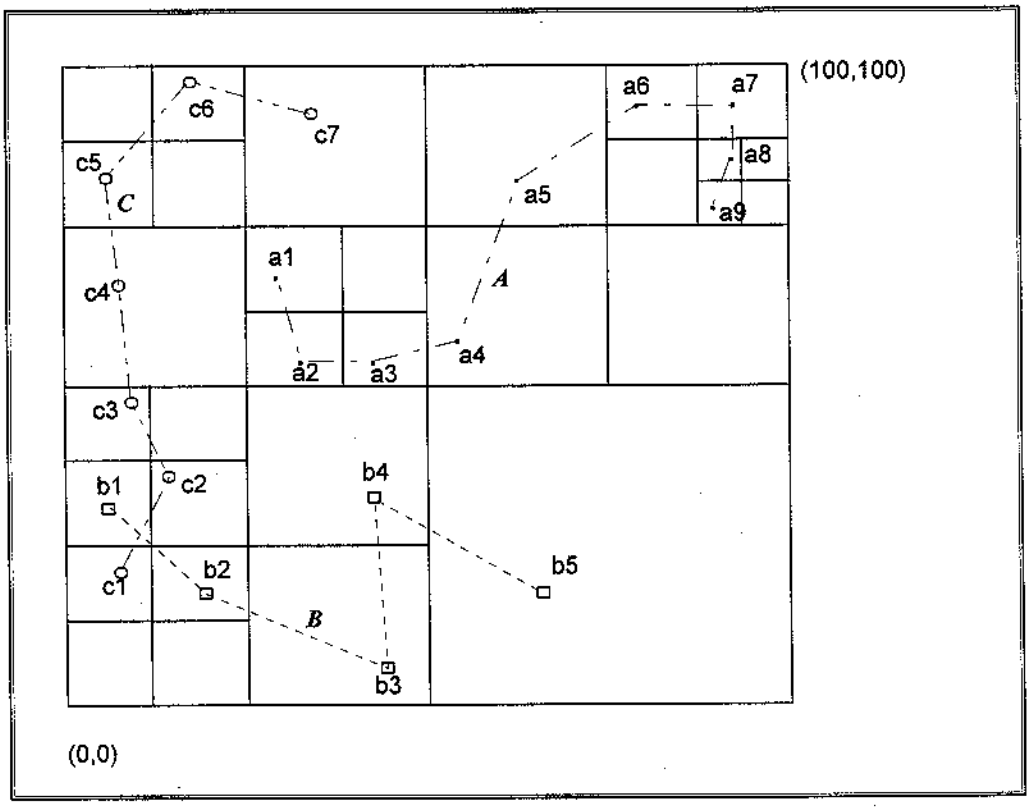



Figure 5-6. Example of seismic lines.

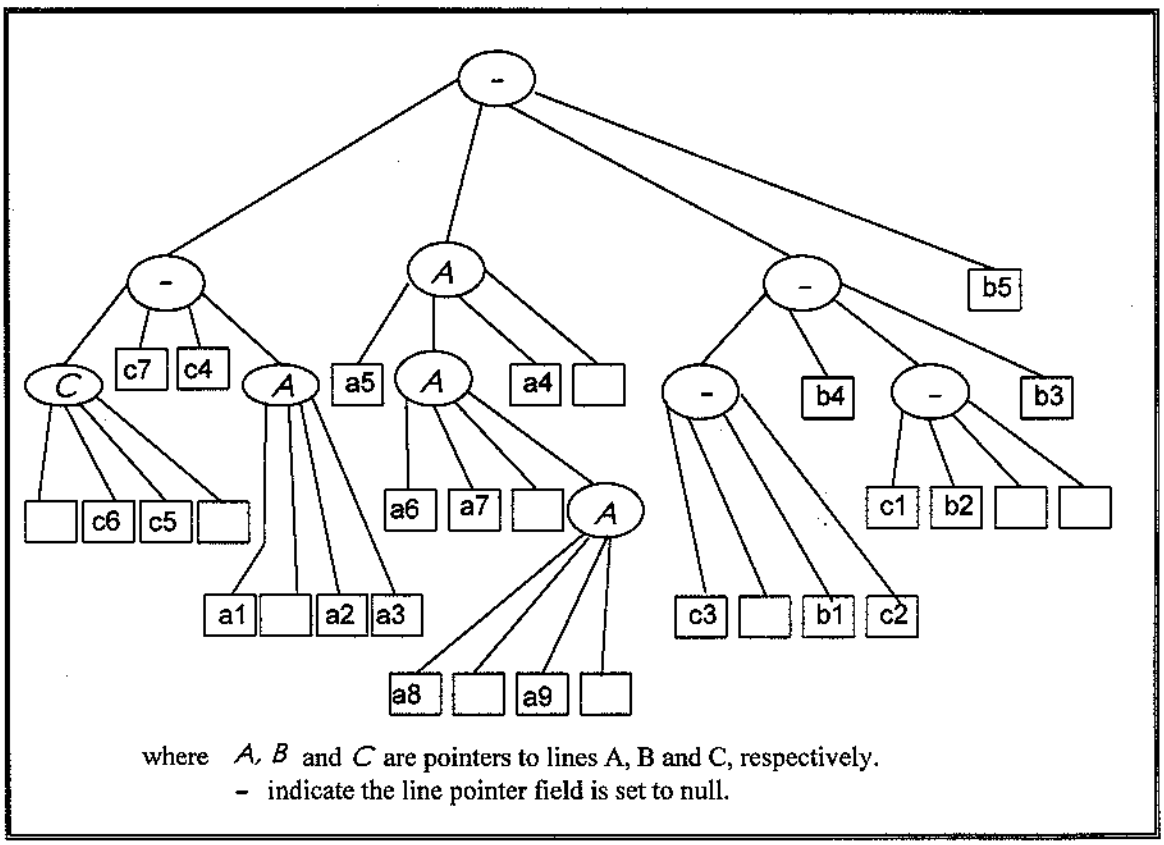



Figure 5-7. Modified seismic index PR quadtree for lines in figure 5-6.

## 5.3 Persistence

During the development of the seismic index structure, the requirement that a format be developed for storing in-memory data structures on disk was identified. The format adopted was to use keyed records in which each record within the stored data structure is delimited by a unique integer key. The

***SGX_MANAGER*** object was created to read and write the current structure to and from disk. The current implementation uses an ASCII format for storing the data on disk.

### 5.3.1 SGX File Format

The structure on disk is broken up into 5 major blocks as shown in figure 5-8. Each of the blocks and sub-sections within the blocks are identified using special key tags. Table 5.1 shows the list of key tags which are used to identify each block and sub-block. The following sections describe the various blocks and records within the SGX file format.

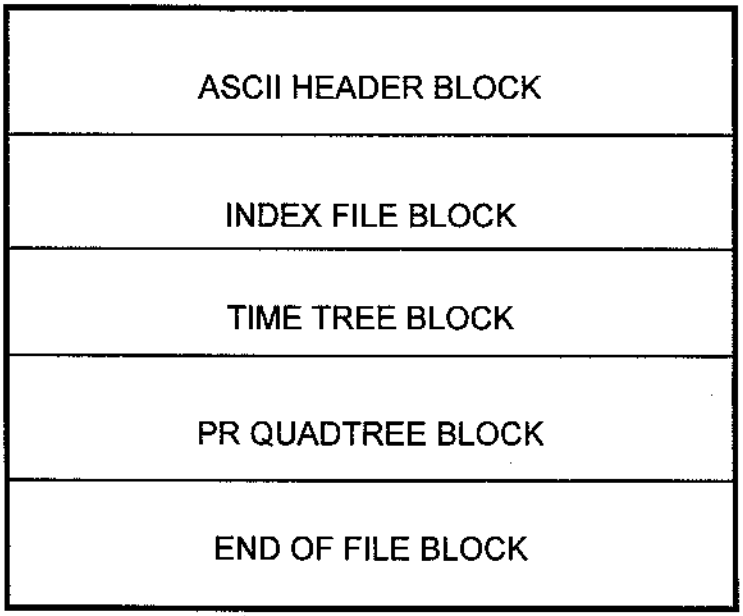



Figure 5-8. The basic format of the SGX file.

Table 5-1. Index for the SGX file major blocks.

| Block Name | Index Key |
|---|---|
| HEADER BLOCK | %%SGX |
| PROJECT INDEX BLOCK | 1010 |
| PROJECT FILE RECORD | 1020 |
| AVL TIME TREE BLOCK | 1030 |
| AVL NODE RECORD | 1040 |
| TRACE RECORD | 1045 |
| PR QUADTREE BLOCK | 1050 |
| GREY QUADTREE NODE RECORD | 1060 |
| BLACK QUADTREE NODE RECORD | 1070 |
| WHITE QUADTREE NODE RECORD | 1080 |
| SGX END OF FILE BLOCK | 999999999 |

### 5.3.2 SGX ASCII Header

The first block in the SGX file is the header block. The header is used to verify that the file is a valid SGX file and that the file version is compatible with the current version of the software. The header section contains three sections with the second section being repeated **n** times. Figure 5-9 shows the layout of the header block and figure 5-10 shows an example. The first line of this section contains the information that identifies the file as an SGX file. This line contains 4 fields and these are:

1. "**%%SGX**": the SGX file identifier

2. the format version number
3. the format for the data in the main body of the **SGX** file (this field is either '**ASCII**' or '**BINARY**')
4. the last field is a count n of the number of comment lines in section two of the header block.

The second section contains comment information about the data contained in the file. This information is not used and is ignored by the application. Comments are ASCII text that end with a carriage return at the end of each line.

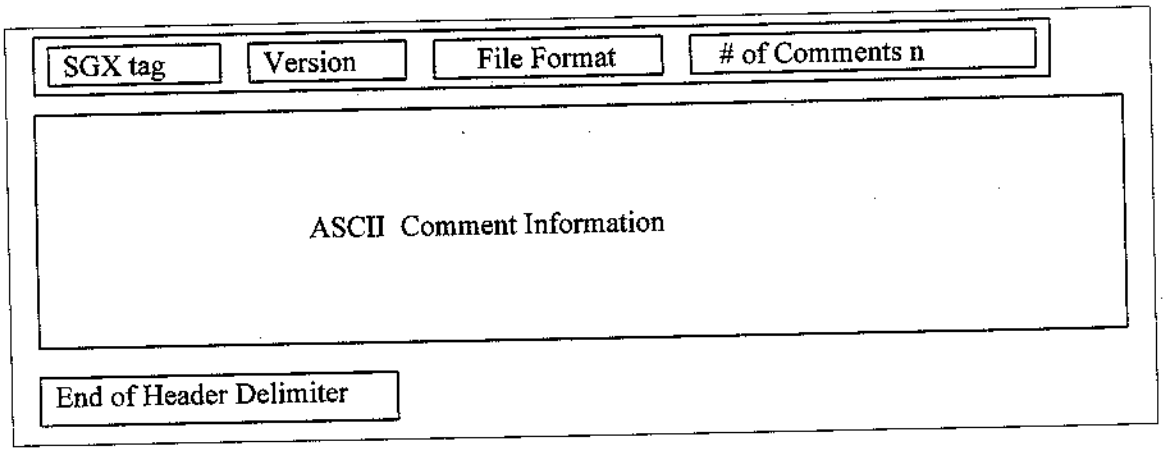



Figure 5-9. Format of the ASCII header block.

```
%%SGX 1.0 ASCII 3
Tue May 12 15:20:10 1995
Seismic index file for the marine seismic survey lines
recorded on leg138 at site 846 on the 10 June 1989.
%%^L
```

Figure 5-10. Example header block for an **SGX** file.

The last line in the header block contains the end of header delimiter. It consists of the three characters '%%^L'. The control-L character represents a page break. This prevents garbage from being printed on the screen when UNIX utilities such as 'more' are used to view the SGX file when the main part of the file is in binary format.

### 5.3.3 Project Index Block

The second block in the SGX file is a project index block. This section of the file lists the project index file and the list of project files that are contained in the seismic index. Figure 5-11 shows the format of this block and figure 5-12 shows an example. The first line of this block contains the project index file key. On the second line the first field contains the number of project files. The second field contains the number of characters in the project index file name. The third field is the project index filename. The next two lines are repeated for every project that the seismic index contains. The first line in the sub-section is the project file key. The second line contains 2 fields. The first is the length of the project filename and the second field contains the name of the project file.

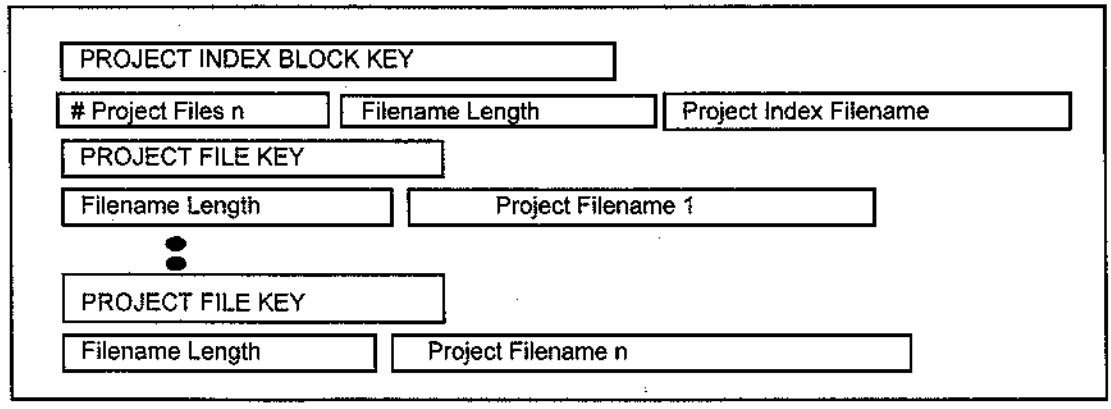



Figure 5-11. Format of the project index block in an SGX file.

```
1010
2  26  /home/segyfiles/leg138.idx
1020
12 leg138_A.prj
1020
 12 leg138_B.prj
```

Figure 5-12. Example of the project index block in an SGX file.

### 5.3.4 AVL Time Tree Block

The third block in the ***SGX*** file contains several sub-sections of information. Within this block there are several sections. These include:

- a section for the structure of the AVL time tree
- a section for each seismic line object's data
- a section for each seismic trace object's data for a seismic line.

Figure 5-13 shows the format of the AVL time tree block and figure 5-14 shows an example. A pre-order traversal of the AVL time tree is performed during the processing of the tree. For each node visited, the seismic line object's data and the associated trace data are output to the SGX file. The first line of the block contains the SGX_AVL_TREE key **'1030'.** The second line contains the number of nodes in the time tree written to the SGX file.

The third line starts the sub-block which contains the information for the root node. This sub-block is repeated for every node in the tree.

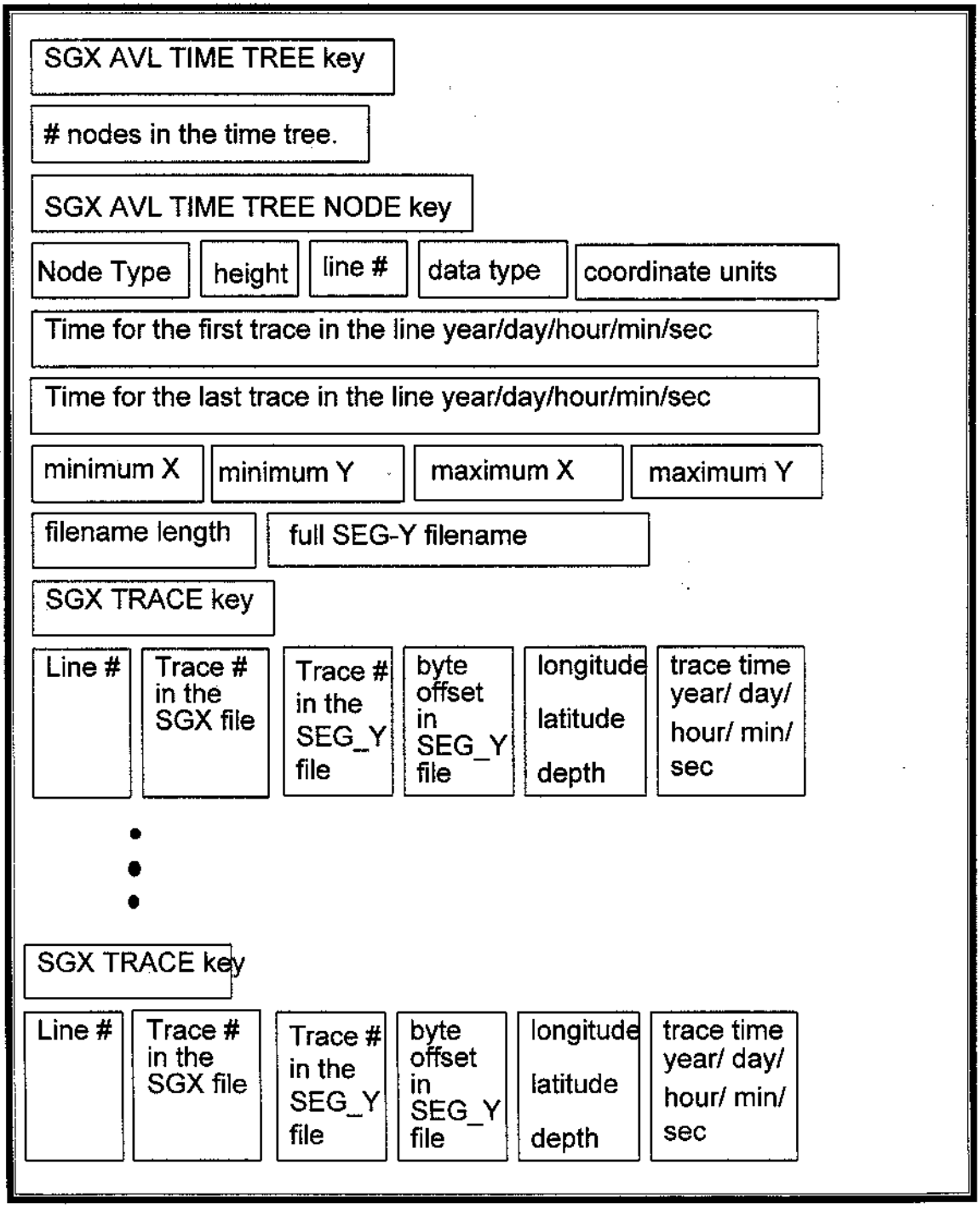



Figure 5-13. Format of the AVL time tree block in an SGX file.

```
1030
 1
1040
100  0 1 2 1
1989 268 23 25 43
1989 268 23 46 23
-91.1799573 7.4448751 -91.1377834 7.4688359
52 /home/atlantic/pjudd/segy_data/processed/tw32_23.sgy
1045
1 1 1879 3600 -91.1377452 7.4688327 3586 1989 268 23 25 43
1045
1 2 1880 11840 -91.1381735 7.4686457  3586 1989 268 23 25 53
1045
1 3 1881 20080 -91.1384341 7.4684461 3586 1989 268 23 26 3
1045
1 4 1882 28320 -91.1388126 7.4682582 3586 1989 268 23 26 13
1045
1 5 1883 36560 -91.1391677 7.4680561 3584 1989 268 23 26 23
1045
1 6 1884 44800 -91.1394953 7.4678628 3584 1989 268 23 26 33
1045
1 7 1885 53040 -91.1398499 7.4676738 3584 1989 268 23 26 43
```

Figure 5-14. Example of the AVL time tree block in an SGX file.

#### 5.3.4.1 AVL Time Tree Node Sub-Section

This sub-section begins with the key '**1040**' on the first line. The second line contains 5 fields. These are:

1. node type: this field indicates whether the node is an internal node or a leaf node. The valid values for this field are ROOT (100), LEAF_LEFT (12), LEAF_RIGHT (14), GREY_LEFT (22) and GREY_RIGHT (24).
2. height: this indicates the height of the node in the tree
3. line number: the line number assigned to the SEG-Y line
4. data type: the format for the trace data
5. coordinate units.

The third line contains the time for the first trace in the seismic line. The format is year, days, hours (24 hour clock), minutes and seconds. The fourth line contains the time for the last trace in the seismic line. The format is the same as for the first trace. The fifth line has 4 fields and contains the covering rectangle for the traces contained in the seismic line. The format is minimum X/ longitude, minimum Y/ latitude, maximum X/ longitude and maximum Y/ latitude. For latitude and longitude coordinates, the format is degrees and decimal degrees. The sixth line contains the SEG-Y filename information. There are two fields. The first contains the length of the SEG-Y filename and the second is the filename.

#### 5.3.4.2 SGX Trace Records

The next section, which contains two lines, has the information for each individual trace within the SEG-Y line. The section is repeated once for each trace. The first line contains the SGX TRACE key. The second contains the trace information and 12 fields. These are:

- field 1, **SEG-Y line number**.
- field 2, the **SGX trace number:** As each trace is written out to the SGX file it is assigned a unique sequential number. The number is used to reduce duplication of data and for linking the trace object to the correct cell within the PR quadtree when rebuilding the seismic index structure from the SGX file.
- field 3, **SEG-Y file trace number**: This is the trace's number within the SEG-Y file. Within the SEG-Y files these numbers are unique but may be duplicated in other SEG-Y files.

- field 4, **byte offset**: This gives the position of a trace's record within an SEG-Y file.
- fields 5 and 6, **trace's reference position:** These fields contain the coordinates for the trace.
- field 7, **water depth:** This field contains the water depth for the trace.
- fields 8 to 12, **shot time:** These fields contain the time that the trace was recorded. The time is in year, days, hours, minutes and seconds format.

### 5.3.5 PR Quadtree Block

The fourth block in the SGX file contains the data for the PR quadtree. This block contains two sections, the first containing the header information for the quadtree and the second contains the data for each node in the quadtree (see figures 5-15 and 5-16). For the leaf nodes, only the trace number (that was generated when the trace was written out as part of the AVL time tree block) is output.

The first line of this block contains the SGX_QUAD_TREE key '**1050**'. The second line contains 5 fields. These are:

- field 1, **# nodes:** The total number of nodes in the PR quadtree.
- fields 2 to 5, **quadtree limits:** These fields contain the bounding coordinates for the quadtree. The format is minimum X/ longitude, minimum Y/ latitude, maximum X/ longitude, maximum Y/ latitude.

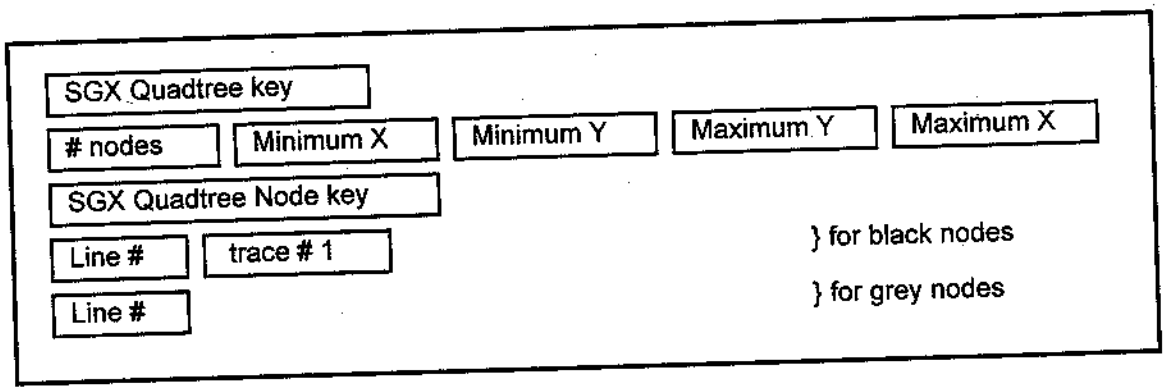



Figure 5-15. Format of the SGX PR quadtree block in the SGX file.

5.3.5.1 SGX Quadtree Node Section

The second section in this block contains the quadtree node records. Three types of nodes are output using a preorder traversal. These are **grey** or interior nodes, **black** leaf nodes and **white** leaf nodes. For the black nodes there are three different types which are output to the SGX file. The first is a black node that contains only one trace. The second type indicates that there is a collision list associated with the node and that at least the following traces belong to the current node. The third type indicates that the trace belongs in the collision list for the current black node. If there is a collision list at a node then the first trace will be output as a type two black node. The remaining traces in the collision list will be output as type three black nodes. For each type, different information is recorded. Each node record begins on the first line of the record with the key. The keys are as follows:

- GREY NODE **1060**
- BLACK NODE **1070**
- BLACK NODE with collision list **1072**
- continuation BLACK NODE **1074**

- WHITE NODE **1080**

For the grey node record, the second line contains 4 fields. The first field contains line object pointer field information. If valid, this field contains a line number or -1 if it is invalid. The second line for all the black node records contains two fields. The first field contains the line number of the seismic line object that the trace(s) belong to, while the second field contains the SGX trace number. These trace numbers are used to determine the memory location of the trace object that is to be inserted into the quadtree node. A dynamically generated list is created when the time tree is loaded that links the SGX trace number to the memory which contains the addresses of the trace objects. The white node record only contains the key field and is used only for reconstructing the quadtree.

```
1050
521 -91.4448624 7.2911465 -90.4079751 8.000000
1060
1
 .
 .
1080
1060
1
1060
1
1070
1 2
1070
1 3
1070
1 1
1060
1
1080
.
```

Figure 5-16. Example of the PR quadtree block in an SGX file.

### 5.3.6 SGX EOF Block

This is the last block in the file. It is used to indicate that there are no more valid data in the file. The SGX key for this block is '**999999999**'.

### 5.3.7 Loading an SGX File

During the loading of the SGX file, the building of the structures is performed in linear time with each object being created as it is read from the SGX file. The fields in the objects are set once and do not need to be updated during the recreation of the seismic index, AVL time tree and the PR quadtree data structures. When the data are read in from the AVL time tree block, the seismic line and seismic trace objects are created and addresses for each object are stored in an array indexed by the SGX seismic line number and SGX trace number which were assigned to the objects when they were written out to the SGX file. When rebuilding the PR quadtree, which is built in linear time, the address for the trace and line objects are obtained from the address arrays in constant time. The loading of the index is therefore expected to require O(n) time, for n = number of traces in the data.

# CHAPTER 6

# SEISMIC INDEX IMPLEMENTATION

## 6.1 Implementation

The data structures that were discussed in the preceding chapters were implemented using the C++ language on a Silicon Graphics workstation which runs the IRIX 5.2 operating system. The previously defined objects became the C++ classes upon which the application was based. The application was designed to run in three modes. The first is batch mode where all the commands are input from a file and the results are written out to a log file. The second is nongraphics/ text mode, where text based prompts are used and the output of the queries is a list of selected SEG-Y files. The third mode is graphics mode. This mode uses a graphical user interface built using the Motif X-Windows environment. The commands are selected by activating buttons and entering the query information into a dialog box. The results of the searches can be displayed graphically or written out to a file.

### 6.1.1 Project Index Support Files

The ***segyindex*** application is built around two groups of files. The first is the seismic line file group of files. These are organized in a three-level hierarchy of files. These files progressively contain more detail about the seismic data as you move level 1 to level 3. Figure 6-1 shows an example of organization of these

files which are described in sections 6.1.2 to 6.1.4. The other group of files are the application files and these are discussed in section 6.1.5.

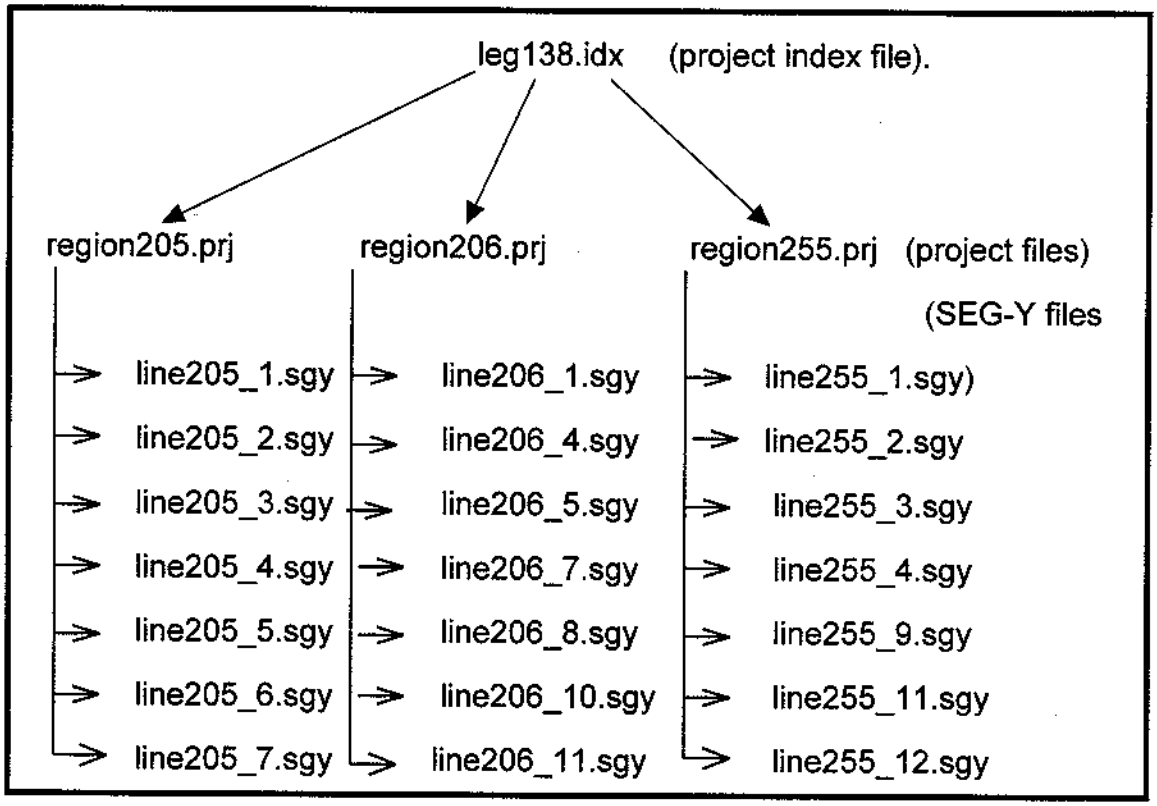



Figure 6-1. Hierarchy for the SEG-Y seismic data file.

### 6.1.2 Seismic Index Project File

The root of the hierarchy is the seismic index project file. This file contains general information about all the projects that are to be loaded into the seismic index data structure. This includes the index cover which is used to define the PR quadtree structure. This cover must encompass all the traces that are to be loaded into the seismic index; otherwise traces which fall outside this cover will fail to be inserted into the PR quadtree and thus will not be picked up during a

range search. This file also contains a list of the project files that will be used to build the search structure. The data records in this file begin with one of two character keys. These are:

- **%INDEX_COVER** minimum X / Y, maximum X / Y in (degrees) and minimum and maximum depth (in metres).
- **%PROJECT_FILE** the name of the project file.

Lines beginning with the character "**!**" are considered comments and are ignored. Figure 6-2 shows an example of a project index file which points to four project files.

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!   PROJECT INDEX FILE for leg138 survey
! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!      bounding coordinates
%INDEX_COVER -96.5000   -3.5000   -89.7500    0.5000   2953 3774
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!      Project list file
%PROJECT_FILE line17.prj
%PROJECT_FILE line18.prj
%PROJECT_FILE line19.prj
%PROJECT_FILE line20.prj
```

Figure 6-2. Example of a project index file.

### 6.1.3 Project Files

The project files form the second level in the hierarchy. These ASCII files point to a group of SEG-Y files which have been grouped by vessel, location, acquisition time or some another attribute. The project file contains the cover for

the files, the minimum and maximum time and the list of SEG-Y files. The data records begin with one of four character keys. These are:

- **%PROJECT_COVER** minimum X / Y, maximum X / Y and minimum and maximum depth
- **%MINIMUM_TIME** the minimum time in year/ day/ hour/ minute/second format
- **%MAXIMUM_TIME** the maximum time in year/ day/ hour/ minute/second format
- **%SEGY_FILE** the name of the SEG-Y file.

Lines in the project file beginning with the character "!" are considered comments and are ignored. Figure 6-3 shows an example of a *project file* which points to 3 SEG-Y lines.

```
!-------------------------------------------------------------
!     Example PROJECT FILE
!     File contains the following information.
!     a) the coordinates of the bounding rectangle for the project.
%PROJECT_COVER -96.1945 -0.4699 -95.2328 0.2428 3060 3369
!     b) the upper and lower time limits for the project.
!     start time
%MINIMUM_TIME 1989 258 20 20 16
!     end time
%MAXIMUM_TIME 1989 259 8 26 4
!     c) the filenames of the SEG-Y files
%SEGY_FILE line17A.sgy
%SEGY_FILE line17B.sgy
%SEGY_FILE line17C.sgy
```

Figure 6-3. Example of a project file.

### 6.1.4 SEG-Y Files

The seismic data files form the third level in the hierarchy. The structure of these files and the data which are extracted from these files are described in chapter 2.

### 6.1.5 Application Files

There are two files that directly affect the graphical component of the application and the initial values for the dialogue boxes and prompts for various input. The first is the ***segyindex*** X-Windows resource file. This file contains the information for defining the colours, positions, sizes and text for the various widgets within the interface.

The second file, the ***segyindex initialization file,*** contains initial values for prompts and default sizes for graphic objects used to display the results. The data records begin with one of seven character keys. These are:

- **X_LABEL_OFFSET** selected line label's X offset in pixels
- **Y_LABEL_OFFSET** selected line label's Y offset in pixels
- **LABEL_MARKER_SIZE** size of a selected line's label in pixels
- **DRAWING_LINE_WIDTH** line width in pixels for drawing lines
- **START_QUERY_TIMES** the start query time in year/ day/ hour/ minute/second format
- **END_QUERY_TIMES** the end query time in year/ day/ hour/ minute/second format
- **RANGE_SEARCH2D_XY** 2D range search coordinate cover.

Figure 6-4 shows an example of the ***segyindex initialization*** file.

```
!SEISMIC INDEX PROGRAM GRAPHICS AND VARIABLE INITIALIZATION FILE. !
The file contains the definitions for the current session of the SEISMIC index
!  program. When the program is started the values from the previous session will be
!  loaded and used as initial values in the prompts and for displaying the values.
!
!  Format : All lines beginning with a '!' are treated as comments. Keywords must begin in
!  the left most column, and there must be at least one blank space separating the keyword
!  from the value. If no  value is present the default value will be used.
X_LABEL_OFFSET 5
Y_LABEL_OFFSET -5
LABEL_MARKER_SIZE 10
DRAWING_LINE_WIDTH 2
START_QUERY_TIMES 1989 262 16 0 0
END_QUERY_TIMES 1989 262 22 0 0
RANGE_SEARCH2D_XY -95.1749 -2.30084 -94.4361 -1.87465
```

Figure 6-4. Example of the segyindex applications initialization file.

## 6.2 Running the ***segyindex*** Application

The application is started by typing ***segyindex*** on the command line along with any of the command line arguments (see figure 6-5). In graphics mode, the GUI in figure 6-6 will be displayed on the screen while in text mode the menu in figure 6-7 will be displayed. The next step is to select the INITIALIZATION command and to select the ***project index*** file to load. The INITIALIZATION menu (see figure 6-8 ) allows projects to be added or removed from the list which will be used to build the index, to create the SEISMIC INDEX, to create a project index file from the listed projects or build the index for an ***SGX*** file. Once the index structures have been built, the time and range searches can be performed.

| Command | Description | Example |
|---|---|---|
| -ba | batch file mode | -ba leg138_test.sh |
| -gr | graphic mode | -gr |
| -ti | used for timing operations | -ti |

Figure 6-5. Segyindex command line arguments.

```
********* MAIN SELECTION MENU ****************
-1) :: EXIT
 0) :: INITIALIZE THE DATA STRUCTURE
 2) :: INSERT A LINE INTO THE DATA STRUCTURE
 3) :: INSERT A TRACE INTO THE DATA STRUCTURE
 4) :: DELETE A LINE FROM THE DATA STRUCTURE
 5) :: 2D RANGE SEARCH FOR LINES IN THE DATA STRUCTURE
 6) :: TIME RANGE SEARCH FOR LINES IN THE DATA STRUCTURE
 7) :: DISPLAY SELECTED LINES
 8) :: DISPLAY THE SHOT INFORMATION FOR THE SELECTION
 9) :: DISPLAY THE LINE TIME TREE
10) :: ACTIVATE/ DEACTIVATE THE SETTINGS SWITCHES
12) :: LOAD A PROJECT FILE
13) :: SAVE THE CURRENT PROJECT TO DISK
14) :: SAVE ALL THE SEISMIC INDEX DATA STRUCTURES ON DISK
```

Figure 6-6. Segyindex main menu for nongraphics mode.

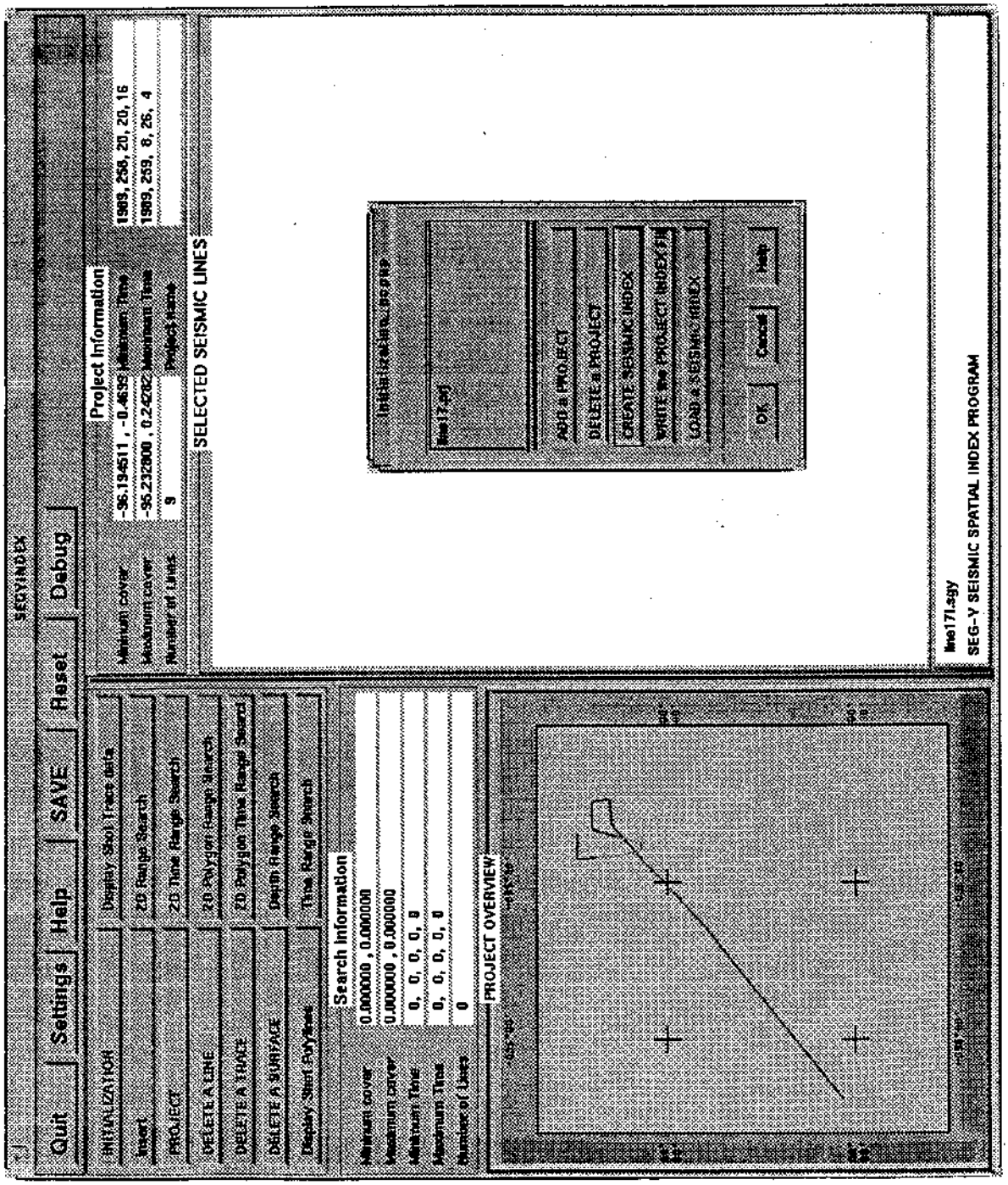



Figure 6-7. Graphical user interface for the SEGYINDEX application.

```
(-1) EXIT
( 1) Add another project to the list
( 2) Delete a project from the list
( 3) Create the Seismic index from the project files
( 4) Write the project to the Project index file
( 5) Load the Seismic index from the Index file
Enter the option number :>
```

Figure 6-8. Segyindex initialization menu for nongraphics mode.

## 6.3 Output and Display of Query Data

Once the SEG-Y data have been loaded into the data structures, two types of queries can be performed. The first are queries based on a time interval. The second are queries based on a user defined query rectangle. The results of the search can be listed, displayed graphically or written to a TDR file so that the results of the query can be viewed in 3D using the FLEDERMAUS application [Paton, 1995] on an SGI workstation. The following sections describe the two types of searches and the types of outputs.

### 6.3.1 Time Range Searches

The data structures allow searching to be performed on the trace's time component. From the main menu, the Time Range Search command is selected. The minimum and maximum times for the search interval are then entered. Using these values, a search is then performed on the AVL time tree. Once the

search has been completed, the number of lines and the covering rectangle for all the data contained in the selected lines will be displayed in the Search Information panel or on the screen.

The results of the query can then be viewed by selecting the Display Shot Polylines command. For nongraphics mode, the list of selected lines are displayed on the screen. In graphics mode the selected lines are displayed in the Selected SEISMIC LINES window (see figure 6-9). When the selected lines are displayed, the data are transformed so that the seismic data covers the entire drawing area.

The selected lines are identified in two ways. The first is that the lines are drawn using the same colour as was used to display it in the overview window. Secondly, the beginning of each line is identified by a large dot and a label. The label is a real number which represents the Julian date, from 1900, for the first trace in the line.

There are several drawing parameters that can be changed to vary the appearance of the displayed lines. These include:

- the width of the line which represents the path of the seismic line
- the size of the dot which marks the beginning of the line
- the offset of the text from the beginning of line marker
- whether the seismic line is represented by a polyline or whether the traces are drawn as dots.

The second method for outputting the results of the query is to create a TDR file. The TDR file contains all the data from the SEG-Y data files. The TDR file can then be loaded into the FLEDERMAUS program and the seismic profiles viewed in 3D.

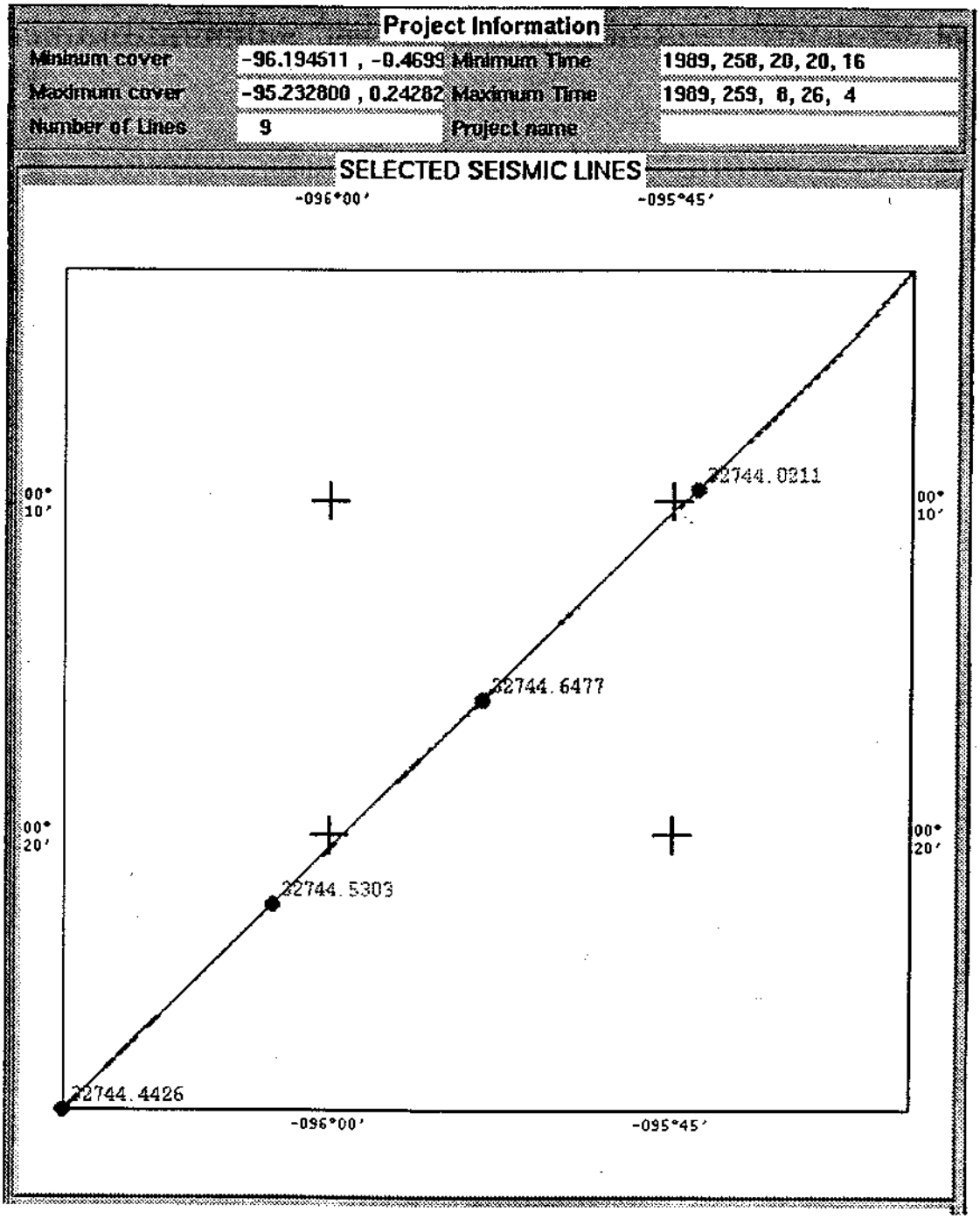



Figure 6-9. Graphical interface for displaying the selected lines.

### 6.3.2 2D Range Searches

2D range searching is the second supported searching method. The 2D range search command is selected from the main menu. The limits for the query can then be entered. In graphics mode, the limits can be input via a dialog box or by defining the search area using the mouse in the project overview area. The PR quadtree is then searched to determine which, if any, lines satisfy the query. The results of the query can then be displayed by selecting the Display Shot Polylines command. For nongraphics mode, the list of selected lines are displayed on the screen. In graphics mode the selected lines are displayed in the Selected SEISMIC LINES window (see figure 6-9). The results of the query can also be output to a TDR file. The TDR file contains all the data from the SEG-Y data files. The TDR file can then be loaded into the FLEDERMAUS program and the seismic profiles viewed in 3D.

# CHAPTER 7

# RESULTS

## 7.1 Test Data

The data used for testing the data structures developed for this thesis were obtained from a geological survey that was performed in the eastern equatorial Pacific Ocean in September 1989 (see figure 7-1). The 8800 km of single channel digital seismic data were collected during a site survey on the *Thomas Washington* from August to October 1989 [Bloomer, 1992]. The seismic source consisted of two synchronized water guns. The receiver consisted of a Teledyne streamer having 48 acceleration-cancelling hydrophones in a linear array. Both the source and streamer were towed at a depth of approximately 5 metres. The data were recorded on nine-track magnetic tape at a 1-ms sampling interval for the eastern transect and 2-ms for the western transect [Bloomer, 1992]. The resultant shots contain 2000 samples with each line containing approximately 4000 traces.

The navigation information was collected using GPS at intervals of 1 and 5 minutes. These data were combined with the course-change and speed-change information from the shipboard bridge log to give the reference position for the shots in latitude and longitude format.

Seismic lines from tapes 17 to 24 were used for the testing. Table 7.1 shows the positional information for the lines and figures 7-2 and 7-3 show the track information for the lines. The test was restricted to these eight lines due to

disk space limitation and the limited coverage of the test data. In order to increase the number of lines for the tests, each line was broken up so that each partitioned **SEG-Y** line contained approximately 500 traces. This partitioning increased the number of lines to 68. Several additional lines were added by modifying the reference positions for the traces in lines 23 and 24. This was done to increase the number of lines to 85 and to increase the density of the lines in the western section of the survey area. Appendix K lists the coordinate coverage and times for lines 17 to 26; these are summarized in Table 7-1. The data require 335.8 MBytes of disk storage and contains a total of 39752 shot points.

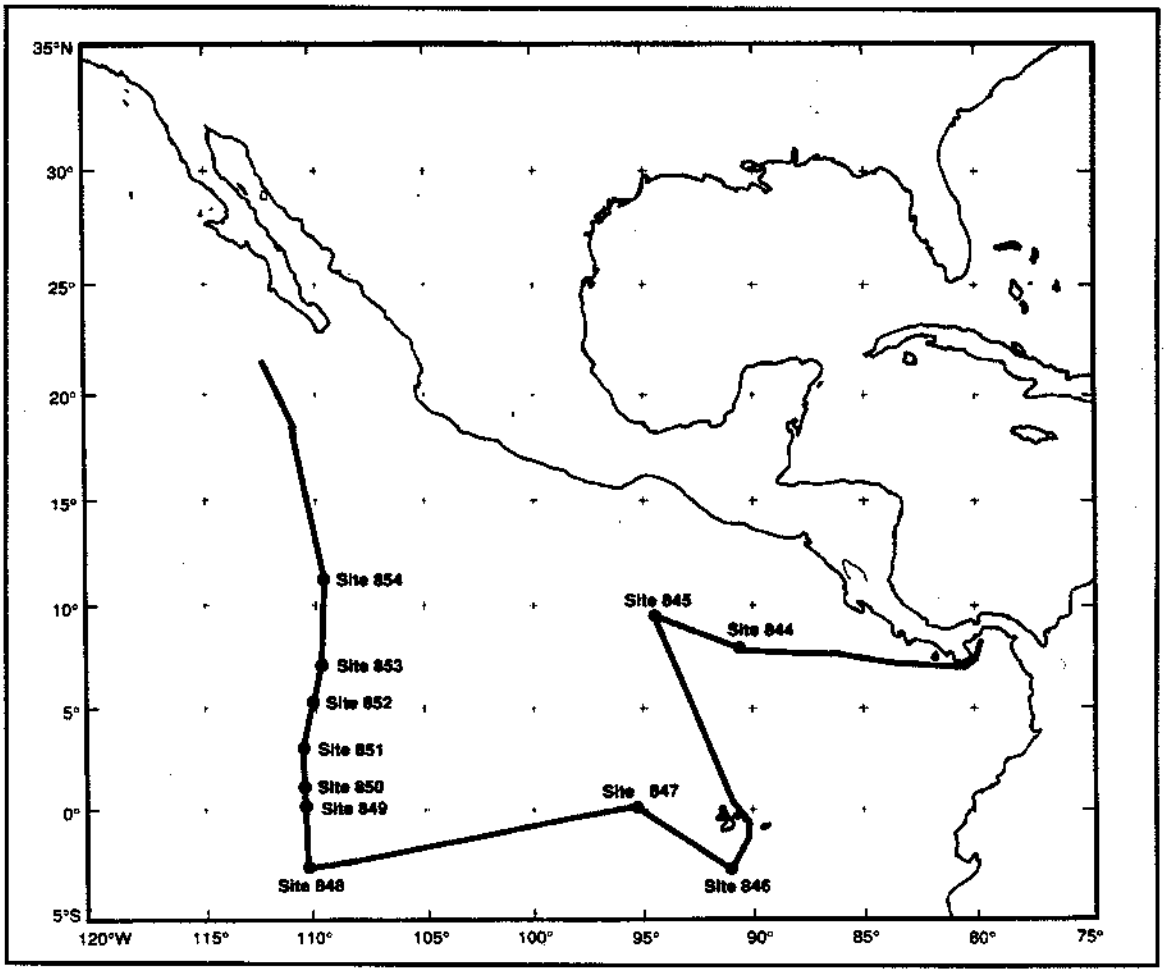



Figure 7-1. Leg 138 track line in the eastern equatorial Pacific Ocean.

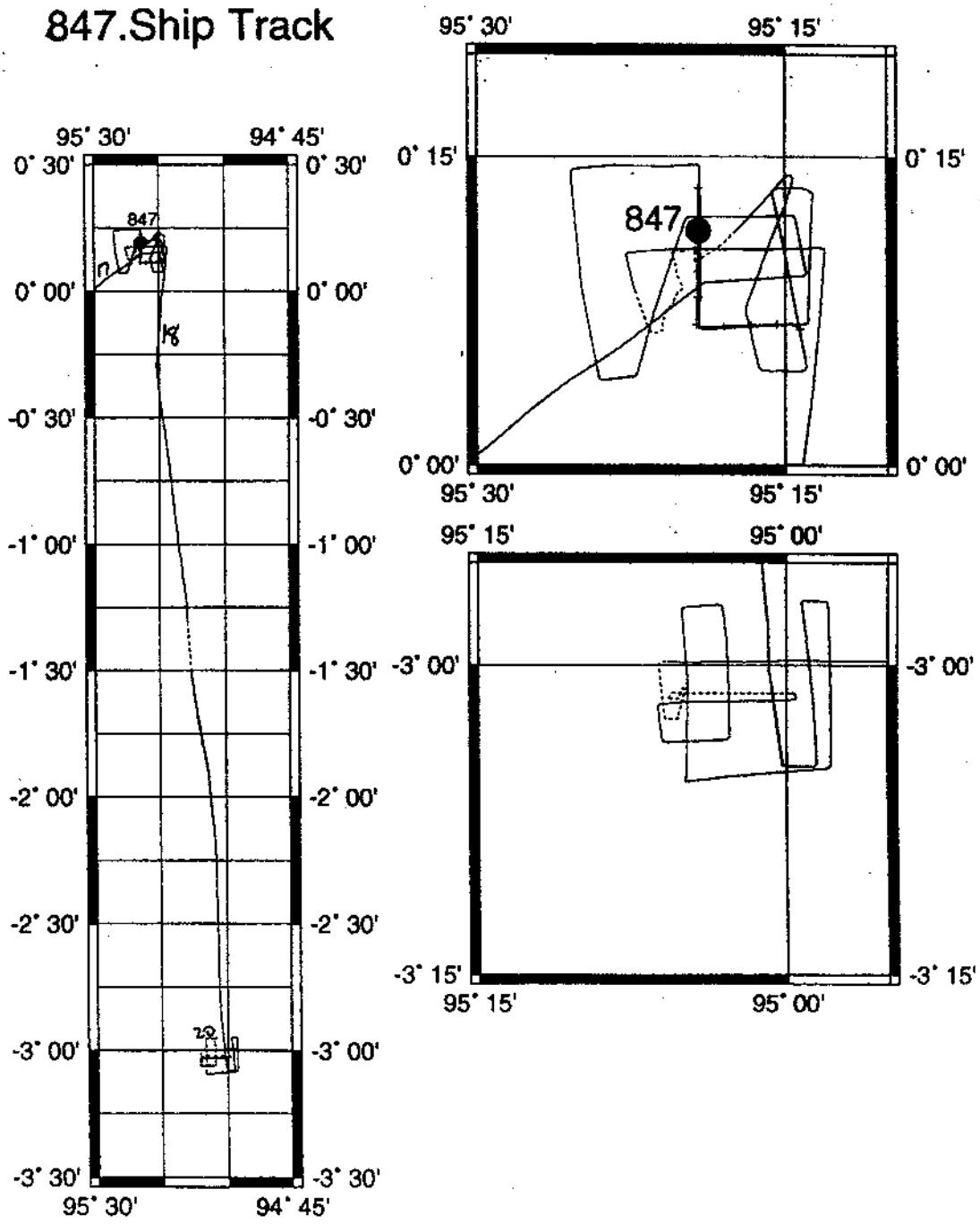



Figure 7-2. 847 ship track [Bloomer, 1992].

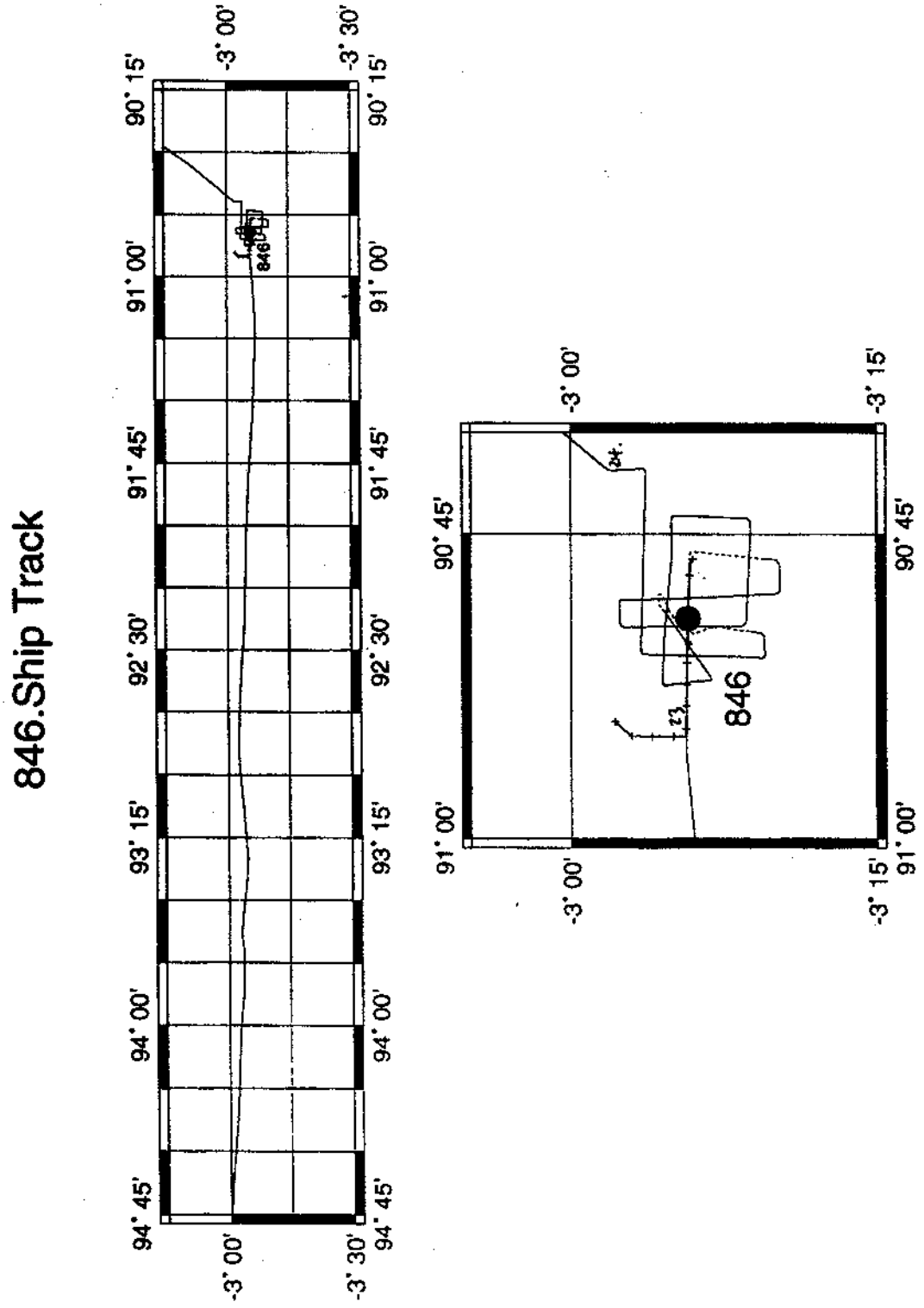



Figure 7-3. 846 ship track [Bloomer, 1992].

Table 7-1. Position and time specifications for lines 17 to 24.

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 17 | -96.194511 | -0.469996 | -95.232864 | 0.242827 | 3060 | 3369 | 1989 | 258 | 20 | 20 | 16 |
| | | | | | | | 1989 | 259 | 8 | 26 | 4 |
| 18 | -95.378540 | -0.555635 | -95.217369 | 0.243596 | 3226 | 3418 | 1989 | 259 | 8 | 26 | 14 |
| | | | | | | | 1989 | 260 | 10 | 2 | 51 |
| 19 | -95.227921 | -2.496941 | -95.034996 | -0.556040 | 3186 | 3580 | 1989 | 260 | 10 | 3 | 1 |
| | | | | | | | 1989 | 260 | 23 | 0 | 7 |
| 20 | -95.104584 | -3.094664 | -94.954750 | -2.497341 | 3152 | 3662 | 1989 | 260 | 23 | 0 | 17 |
| | | | | | | | 1989 | 261 | 17 | 6 | 26 |
| 21 | -94.954369 | -3.075835 | -93.354362 | -2.996772 | 2989 | 3774 | 1989 | 261 | 17 | 6 | 36 |
| | | | | | | | 1989 | 262 | 4 | 16 | 34 |
| 22 | -93.353981 | -3.096644 | -91.635811 | -3.041333 | 2953 | 3684 | 1989 | 262 | 4 | 16 | 44 |
| | | | | | | | 1989 | 262 | 15 | 59 | 52 |
| 23 | -91.635422 | -3.171968 | -90.735168 | -3.039950 | 3267 | 3636 | 1989 | 262 | 16 | 0 | 2 |
| | | | | | | | 1989 | 263 | 3 | 28 | 40 |
| 24 | -90.852531 | -3.159670 | -89.971893 | -1.836223 | 3131 | 3379 | 1989 | 263 | 9 | 15 | 53 |
| | | | | | | | 1989 | 263 | 21 | 8 | 40 |

## 7.2 Experiment Description

The tests were performed with all the 85 lines loaded into the *SEISMIC INDEX STRUCTURE*. Each of the tests was run 10 times using the standard PR Quadtree structure and the modified *SEISMIC INDEX PR QUADTREE*. The tests entailed building the index structures which took 34.6 seconds for the standard structure and 34.8 seconds for the modified structure. A series of searches were then performed using several different-sized query windows which ranged in size from 10 percent of the area covered by the data up to 90 percent. For each case a randomly generated query window was generated which covered 10%, 25%, 50%, 75% and 90% of the range of the data. The query windows were restricted so that they always fell within the limits of the data and did not fall outside the data limits. For example, in case one (10% coverage), the origin for the query window was bounded by the lower limit of the seismic data and 90% of the seismic data range in both latitude and longitude. Tables L-1 to L-5 show the query windows that were used for the testing. Each of the tests was run 10 times and the average times and their standard deviations were calculated (see tables M-1 to M-11).

The modified structure storage requirements are as follows:

- the PR quadtree class requires 88 bytes.
- the PR quadtree node class requires 40 bytes. The quadtree built for the experiment contained 157761 nodes and thus required 6,310,528 bytes to store the seismic line information.

Figure 7-4 shows the algorithm that was used to generate the random query windows for the case where the query area is 10% of the total coverage area for the seismic data.

```
set query_longitude_window size to 10% of longitude_range
set query_latitude_window size to 10% of latitude_range
for I equal to 1 up to the number of trials
        p_longitude = random ( )  [value between 0 and 0.9]
        p_latitude  = random ( ) [value between 0 and 0.9]
        longitude_query_origin = longitude origin  +
                                        p_longitude  * longitude_range
        latitude_query_origin = latitude origin  +
                                        p_latitude  * latitude_range
        longitude_query_top = longitude_query_origin +
                                query_longitude_window_size
              latitude_query_top = latitude_query_origin +
                                query_latitude_window_size
        end of for loop
```

Figure 7-4. Pseudocode for generating test query windows for case where query area is 10%.

## 7.3 Range Query Search Tests

The tests were run a total of ten times on May 18, 1995 between 4 am and 10 am on the Ocean Mapping Group's Silicon Graphics workstation Aegean. This workstation has the following specifications:

1) 33 MHz IP12 processor

2) operating system: IRIX 5.2

3) main memory: 48 MBytes

4) CPU: MIPS R2000A/ R3000 processor chip

5) FPU : MIPS R2010A/ R3010 VLSI floating point chip.

Each search in each test was performed a total of 250 times so that non-zero times could be obtained. The times were recorded in microseconds using the built in system time function *"getrusage"* which returns the total amount of time that was used. The times are returned in seconds and microseconds from the time that the program began execution. The average for each query window was calculated in microseconds and the results are shown in tables M-1 to M-11. Figures 7-5 to 7-10 show the difference in the search times for the different query windows.

## 7.4 Comparison of Query Search Times

The following graphs (figures 7-5 to 7-10) show the comparison of the search times for both the existing and the modified data structures. Tables 7-2 to 7-4 and figures 7-11 to 7-13 show the relative difference in the query search speeds between the standard and modified data structures. The difference is expressed as follows:

$$\%\ difference = \frac{(speed\ for\ existing\ structure - speed\ for\ modified\ structure) * 100}{speed\ for\ modified\ structure}$$

The test numbers in Tables 7-3 to 7-5 correspond to the following query windows:

Table 7-2. Test number to query window mapping.

| Test Numbers | Test Query Window |
|---|---|
| 1 - 10 | 10% , 1 - 10 |
| 11 - 20 | 25%, 1 - 10 |
| 21 - 30 | 50%, 1 - 10 |
| 31 - 40 | 75%, 1 - 10 |
| 41 - 50 | 90%, 1 - 10 |
| 51 | 100% |

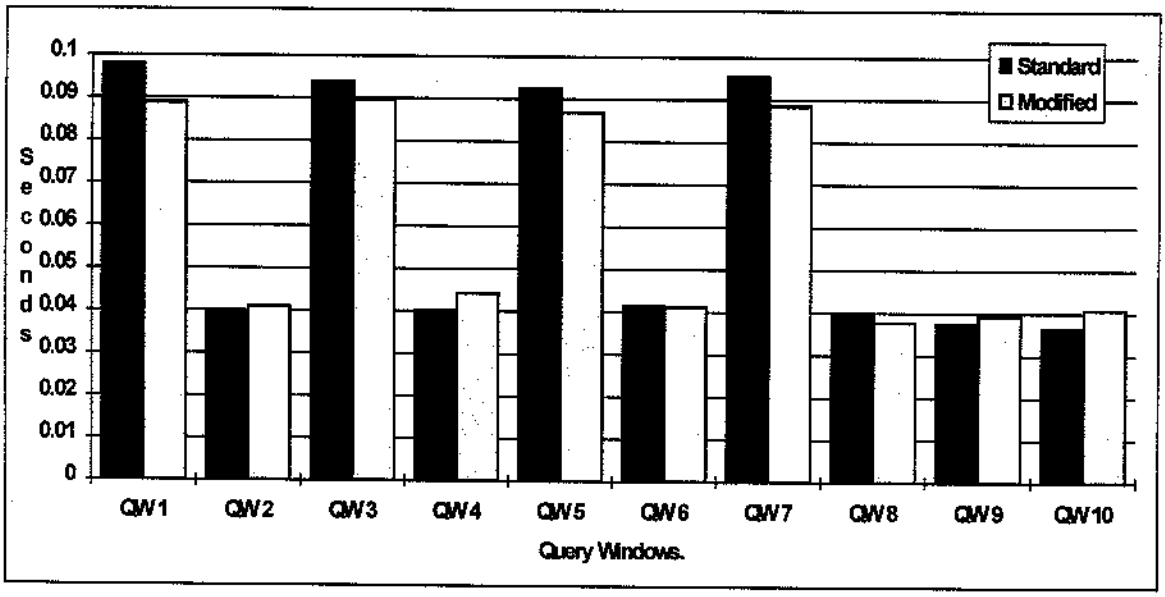



Figure 7-5. Comparison of search times for 10% coverage tests.

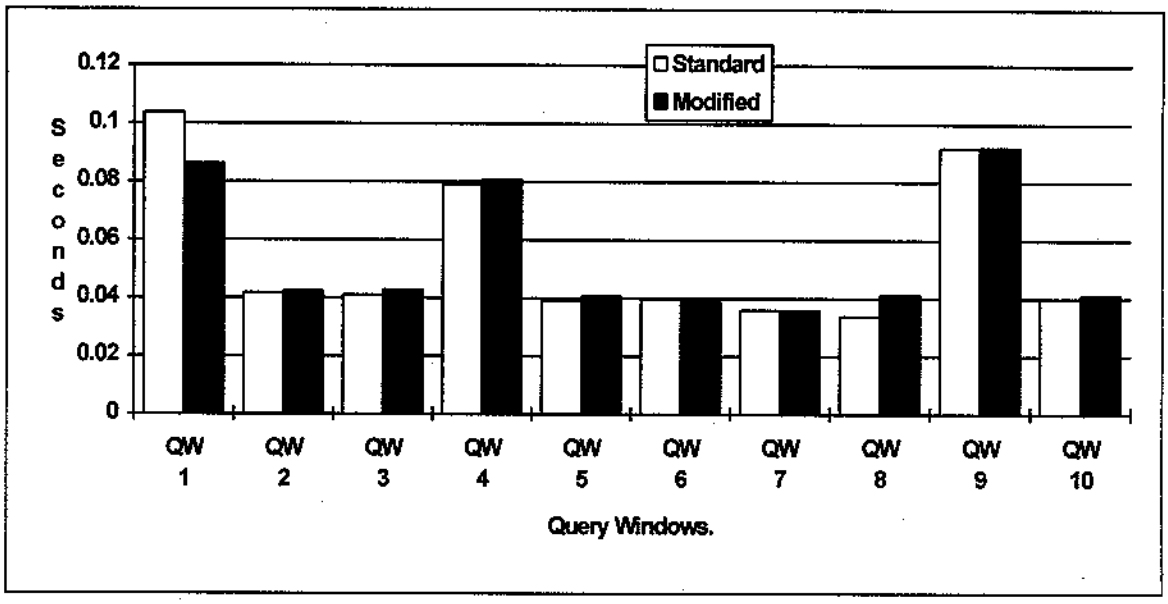



Figure 7-6. Comparison of search times for 25% coverage tests.

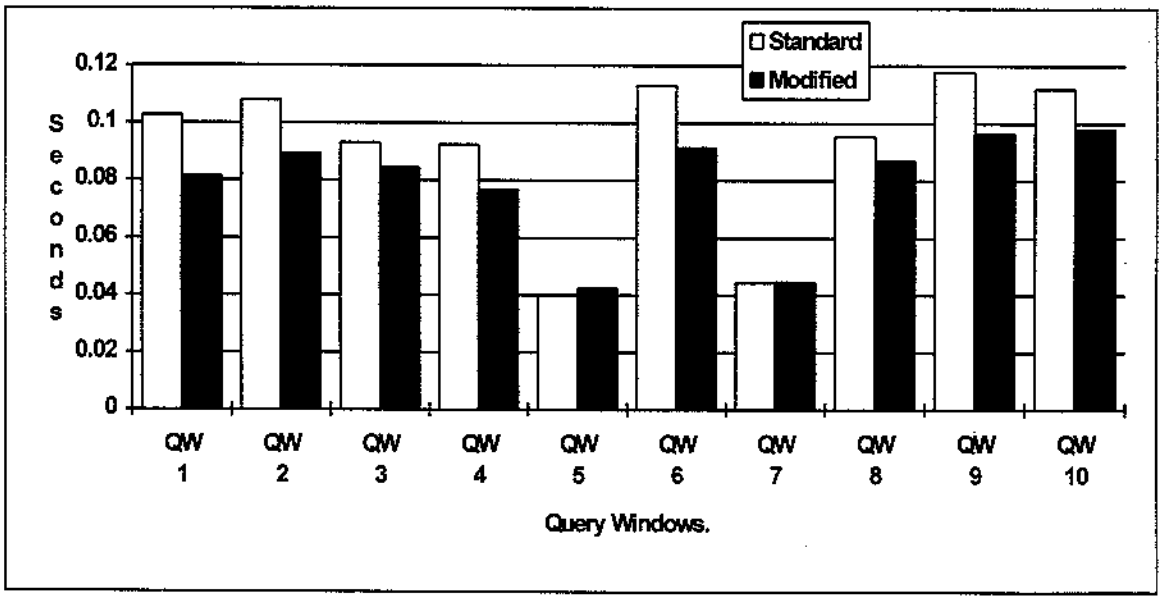



Figure 7-7. Comparison of search times for 50% coverage tests.

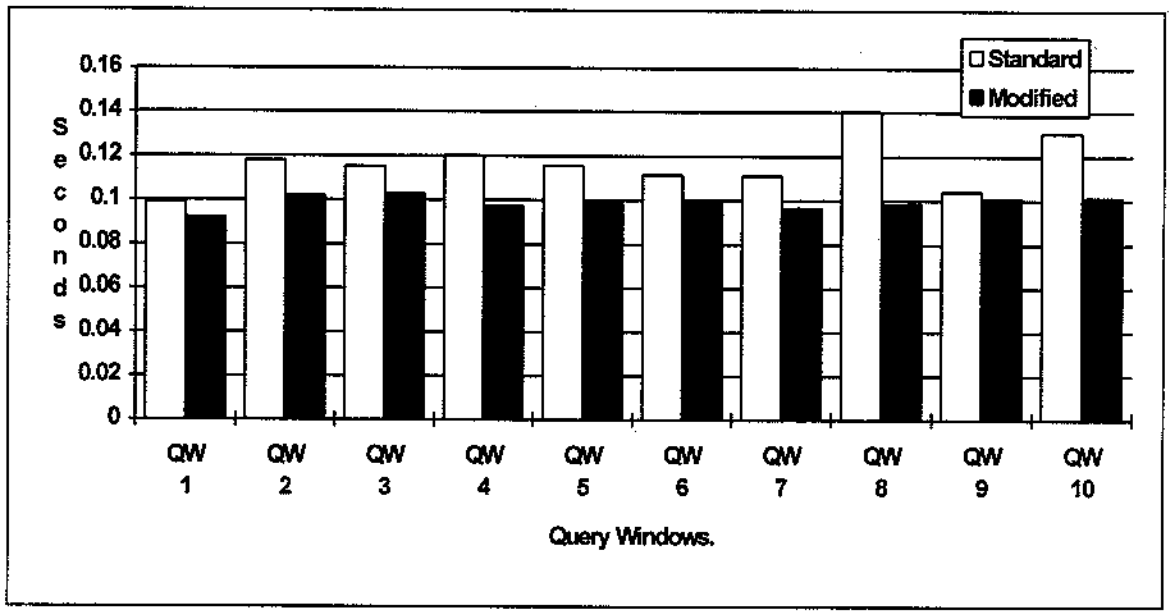



Figure 7-8. Comparison of search times for 75% coverage tests.

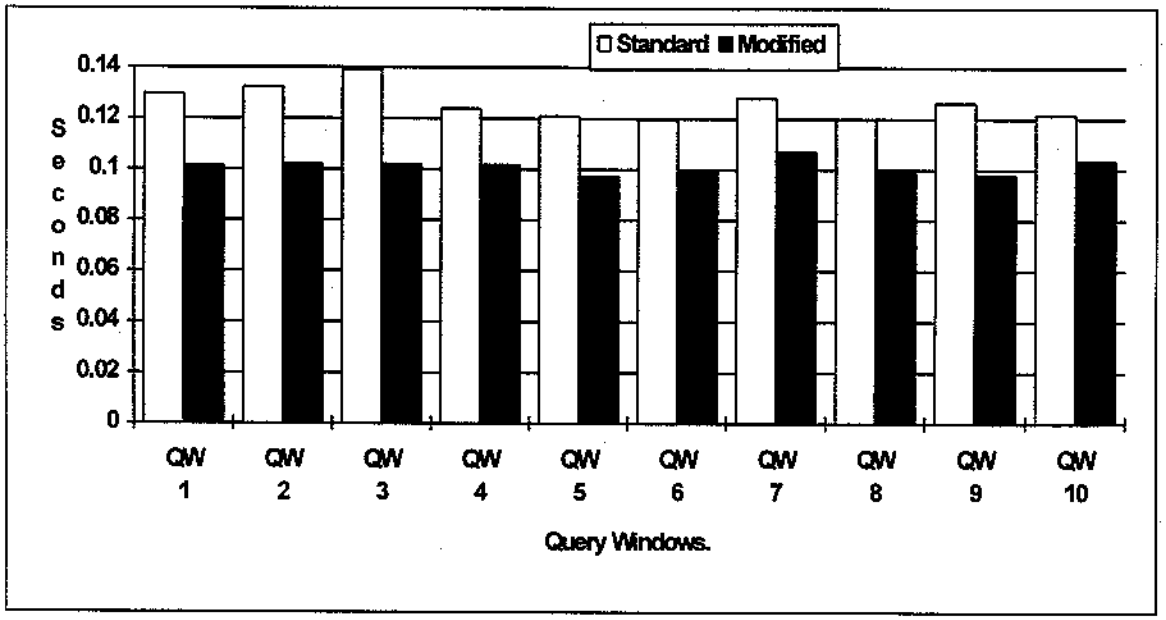



Figure 7-9. Comparison of search times for 90% coverage tests.

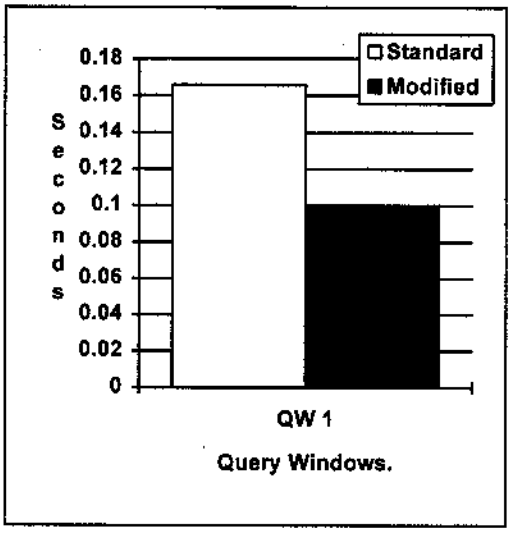



Figure 7-10. Comparison of search times for 100% coverage tests.

Table 7-3. Percentage difference (standard - modified) in search speed for 10% and 25% coverage tests.

| Test | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | -3 | 5 | -9 | 6 | 1 | 8 | 6 | -5 | -11 | 20 | -1 | -4 | -2 | -4 | 2 | 1 | -6 | 0 | -3 |

Table 7-4. Percentage difference (standard - modified )in search speed for 50% and 75% coverage tests.

| Test | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 26 | 21 | 10 | 20 | -6 | 24 | -1 | 10 | 17 | 14 | 8 | 16 | 12 | 23 | 17 | 12 | 15 | 43 | 40 | 29 |

Table 7-5. Percentage difference (standard - modified) in search speed for 90% and 100% coverage tests.

| Test # | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 28 | 30 | 36 | 22 | 25 | 20 | 20 | 21 | 29 | 17 | 67 |

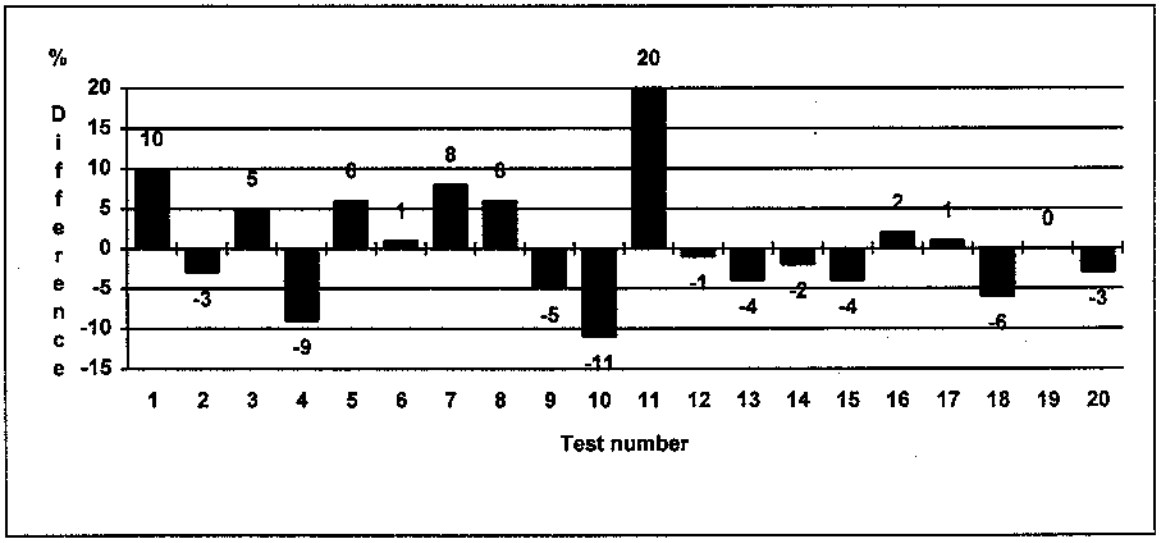



Figure 7-11. Percentage difference (standard - modified) in search speed for 10% and 25% coverage tests.

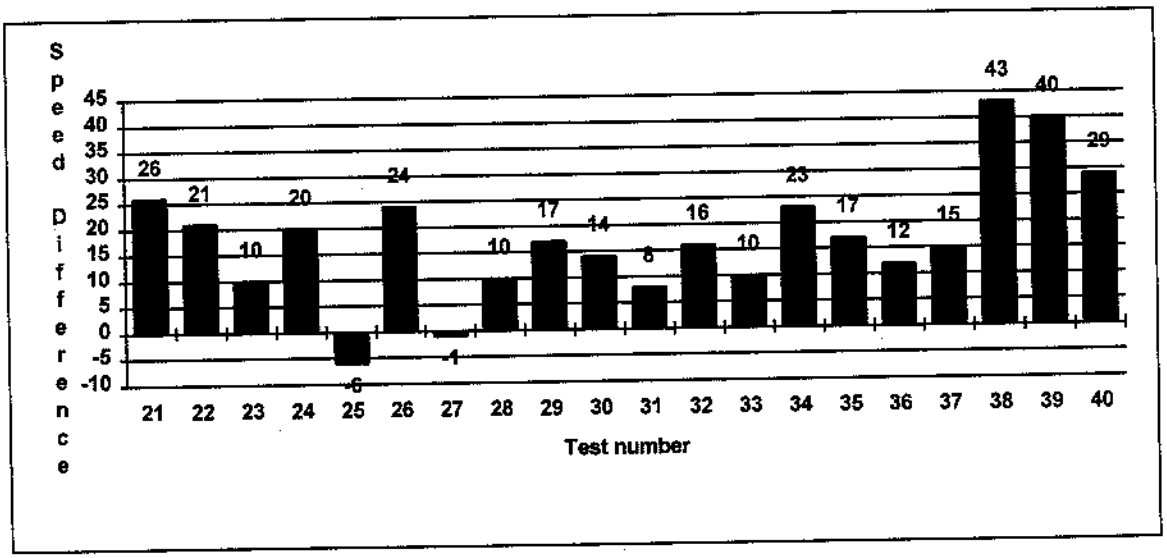



Figure 7-12. Percentage difference (standard - modified) in search speed for 50% and 75% coverage tests.

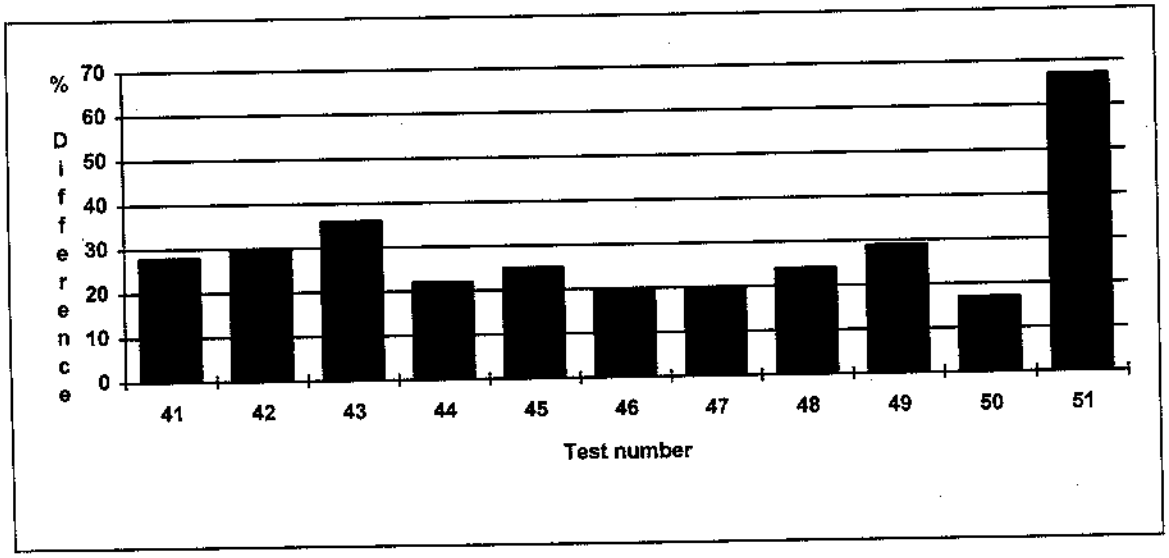



Figure 7-13. Percentage difference (standard - modified) in search speed for 90% and 100% coverage tests.

## 7.5 Remarks on the Differences in the Search Query Times

The query windows that encompassed no seismic lines showed no difference in search speed as expected. In the other cases the difference in search speed varied from approximately 9% for the third 10 percent cover test in which only two lines were returned and 67% for the 100 percent cover test in which all the lines were returned. The greatest improvement in the search times was obtained for the larger query windows within which most of the lines fell. For the smaller query windows the search speed improvement averaged about 10%.

If the number of traces was increased up to the 4000 in the original lines, then the search times would probably decrease in the modified data structure as the pruning of the sub-quadrants will occur closer to the root in the PR quadtree.

# CHAPTER 8

# SUMMARY/ CONCLUSION

Spatial indexing is an important and active area of research in spatial data structures. Spatial indexes for marine geophysical data are also areas of interest. This thesis investigated spatial index data structures for marine seismic data in SEG-Y format. Object modeling and functional modeling methods were used to design the seismic index prototype and the prototype was implemented and run on an SGI workstation. The data structures developed allow for the creation of a spatial index for marine seismic data and also allow for 2D range searches and 1D time searches to be performed.

## 8.1 Summary

The three objectives of this thesis will now be examined to determine what was accomplished by this research and what contribution this work has made to spatially indexing marine seismic data.

The first objective was: "To develop a data structure and algorithms for the storage and retrieval of seismic data." This objective was the main focal point of the thesis. The standard PR quadtree was modified so that a hierarchical structure could be maintained within the quadtree. This hierarchy allowed the

data within a subtree to be grouped together as objects. Thus, in some cases during a search, pruning of large sections of the quadtree can be performed which reduces the number of paths that have to be searched in order to determine all the valid seismic lines.

The second objective was: "To implement and test the data structures and algorithms on actual seismic data." Two data sets from the South Pacific survey [Bloomer, 1992] were used to test the structures and algorithms. The data had to be preprocessed to ensure that the fields that were accessed contained valid data. Also, the reference coordinates for the traces had to be added to the SEG-Y files from the navigation files. The data structures and algorithms were implemented using object oriented C++ on an SGI workstation. The implementation allowed the track lines of the loaded seismic data and the selected seismic data to be viewed graphically.

The final objective was: "To compare the new data structures with the original data structures used for the seismic data." Timing tests were performed for building the data structures and performing 2D range searches on the data structures. The build time for the modified PR quadtree data structure took about 10% more time. The 2D range search tests entailed using different query windows which ranged in size from 10% up to 100% of the area covered by the seismic data. The modified structure generally showed faster times which ranged from 10% for the smallest query windows up to 67% for query windows encompassing all the data.

## 8.2 Conclusion

The data structures developed do not require any additional storage space when compared to the standard PR quadtree. The build times are slightly slower by about 10% and the speed improvement for the search becomes more noticeable with the increase in the number of nodes that have to be checked.

## 8.3 Future Work and Open Questions

Some possible future work that would be beneficial to research includes the following:

1. allow the seismic index data structure to be saved in binary format thus reducing the size of the *SGX* file.
2. investigate and develop a deletion algorithm for the modified PR quadtree.
3. investigate and develop methods which will allow the use of polygonal query windows.
4. enhance the graphical interface to allow the information for individual traces to be obtained by pointing at a segment of a seismic line.
5. enhance the 2D range search so that a second 2D range search can be performed on the resultant seismic lines to return the list of traces within the seismic lines that satisfy the second query.
6. design and develop methods which will allow the structure to support depth range searches on the seismic data.

# CHAPTER 9

# REFERENCES


Adelson-Velskii, G. M. and Landis, E. M. (1962). "An Algorithm for the Organization of Information". *Soviet Math.* Doklady 3, pp. 1259-1263.

Barry, K. M., Cavers, D. A. and Kneale, C. W. (1975). "Special Report. Recommended standards for digital tape formats". *Geophysics*, Vol. 40, pp. 344-352.

Bentley, J. L. and Maurer, H.A. (1975). "Multidimensional binary search trees used for associative searching". *Communications of the ACM*, Vol. 18, pp. 509-517.

Bloomer, S. F. (1992). "Underway Geophysics". *Proceedings of the Ocean Drilling Program, Initial Reports,* College Station, TX( Ocean Drilling Program), Vol. 138.

Bloomer, S. F., Mayer, L. A. and Moore, T. C. Jr. (1994). "Seismic Stratigraphy of the Eastern Equatorial Pacific Ocean: Paleoceanographic Implications". *Proceedings of the Ocean Drilling Program, Scientific Results,* College Station, TX(Ocean Drilling Program), Vol. 138.

Gao, F. (1993). "Spatial Indexing of Large Volume Bathymetric Data Sets". Faculty of Computer Science Technical Report, TR 93-081, University of New Brunswick, Canada.

Landmark/LGC. (September 1992). "Learning Seismic Data Management: Training Manual", Houston TX, USA, Part Number 46-555-00 Rev. C.

Naur, P. (1960). "Revised report on the algorithmic language ALGOL 60". *Communications of the ACM*, Vol. 3, pp. 299-314.

Northwood, E.J., Wisinger, R.C. and Bradley, J. J. (1967). "Recommended standards for digital tape formats". *Geophysics*, Vol. 32, pp. 1073-1084.

Navazo, I. and Brunet, P. (1990). "Solid representation and operation using extended octrees". *ACM Transactions on Graphics,* Vol. 9, no 2, pp. 170-197.

Paton, M. (1995). "An Object Oriented Framework for Interactive 3D Scientific Visualization". Masters Thesis, Faculty of Computer Science, University of New Brunswick, Canada.

Samet, H. (1990). *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, Massachusetts.

Samet, H. (1993). "Notes on Data Structures". University of Maryland, Computer Science Department, Course Notes for CMSC 420, Aug. 24.

Sheriff, R. E. (1973). *Encyclopedic Dictionary of Exploration Geophysics.* Tulsa, Soc. of Exploration Geophysicists.

Telford, W. M., Geldart, L. P., Sheriff, R. E. and Keys, D. A.(1985). *Applied Geophysics,* Cambridge University Press, New York.

Ware, C., Slipp, L., Wing Wong, K., Nickerson, B., Wells, D., Lee, Y.C., Dodd, D. and Costello, G. (1992). "A System for Cleaning High Volume Bathymetry". *International Hydrographic Review*, Vol. 69, no 2, pp. 77-94.

Weiss, M. A. (1992). *Data Structures and Algorithm Analysis.* The Benjamin/Cummings Publishing Company, Inc., New York.

# Appendix A

# SEG-Y BINARY HEADER RECORD

# TECHNICAL

# SPECIFICATIONS

## *400 Byte Binary Reel Identification Header (Record 2, the Binary Coded Block) Binary Code: Right Justified*

| Byte Numbers | Should Record | Description |
|---|---|---|
| 3201 - 3204 | | Job identification number |
| 3205 - 3208 | | Line number (only one line per reel) |
| 3209 - 3212 | | Reel number |
| 3213 - 3214 | | Number of data traces per record (includes dummy and zero traces inserted to fill out the record or common depth point) |
| 3215 - 3216 | | Number of auxiliary traces per record (includes sweep, timing, gain, sync, and all other nondata traces) |
| 3217 - 3218 | | Sample interval in microseconds (for this reel of data) |
| 3219 - 3220 | | Sample interval in microseconds for original field recording (Intervals are designated in microseconds to accommodate intervals less than one millisecond.) |
| 3221 - 3222 | | Number of samples per data trace (for this reel of data) |
| 3223 - 3224 | | Number of samples per data trace (for original field recording) |
| 3225 - 3226 | | Data sample format code:<br>1 = floating point 4 bytes<br>2 = fixed point 4 bytes<br>3 = fixed point 2 bytes<br>4 = fixed point with gain code 4 bytes<br>Auxiliary traces use the same number of bytes per sample. |
| 3227 - 3228 | | CDP fold (expected number of data traces per CDP ensemble) |
| 3229 - 3230 | | Trace sorting code:<br>1 = as recorded (no sorting)<br>2 = CDP ensemble<br>3 = single fold continuous profile<br>4 = horizontally stacked |
| 3231 - 3232 | | Vertical sum code:<br>1 = no sum<br>2 = two sum<br>N = N sum ( N < 32767 ) |
| 3233 - 3234 | | Sweep frequency at start |
| 3235 - 3236 | | Sweep frequency at end |
| 3237 - 3238 | | Sweep length (milliseconds) |
| | | |

| Byte Numbers | Should Record | Description |
|---|---|---|
| 3239 - 3240 | | Sweep type code:<br>1 = linear, 2 = parabolic<br>3 = exponential, 4 = other |
| 3241 - 3242 | | Trace number of sweep channel |
| 3243 - 3244 | | Sweep trace taper length in milliseconds at start if tapered (the tape starts at zero time and is effective for this length) |
| 3245 - 3246 | | Sweep trace taper length in milliseconds at end (the ending taper starts at sweep length minus the taper length at end) |
| 3247 - 3248 | | Taper type:<br>1 = linear 2 = cos2 3 = other |
| 3249 - 3250 | | Correlated data traces:<br>1 = no 2 = yes |
| 3251 - 3252 | | Binary gain recovered<br>1 = yes 2 = no |
| 3253 - 3254 | | Amplitude recovery method:<br>1 = none<br>2 = spherical divergence<br>3 = AGC<br>4 = other |
| 3255 - 3256 | | Measurement system<br>1 = metres 2 = feet |
| 3257- 3258 | | Impulse signal<br>1 = Increase in pressure or upward geophone case movement gives negative number on tape Polarity.<br>2 = Increase in pressure or upward geophone movement gives positive number on tape. |
| 3259 - 3260 | | Vibratory polarity code; seismic signal lags pilot signal by:<br>1 = 337.5 to 22.5<br>2 = 22.5 to 67.5<br>3 = 67.5 to 112.5<br>4 = 112.5 to 157.5<br>5 = 157.5 to 202.5<br>6 = 202.5 to 247.5<br>7 = 247.5 to 292.5<br>8 = 292.5 to 337.5 |
| 3261 - 3600 | | Unassigned; for optional information |

# Appendix B

# SEG-Y TRACE HEADER

# TECHNICAL

# SPECIFICATIONS

## *Trace Identification Header*

| Byte Numbers | Should Record | Field Description |
|---|---|---|
| 1 - 4 | | Trace sequence number within line; numbers continue to increase if additional reels are required on same line |
| 5 - 8 | | Trace sequence number within reel; each reel starts with trace number one |
| 9 - 12 | | Original field record number |
| 13 - 16 | | Trace number within the original field record |
| 17 - 20 | | Energy source point number; used when more than one record occurs at the same effective surface location |
| 21 - 24 | | CDP ensemble number |
| 25 - 28 | | Trace number |
| 29 - 30 | | Trace identification code: 1 = seismic data<br>2 = dead<br>3 = dummy<br>4 = time break<br>5 = uphole<br>6 = sweep<br>7 = timing<br>8 = water break<br>9 - N = optional use (N<32767) |
| 31 - 32 | | Number of vertically summed traces yielding this trace (1 is one trace, 2 is 2 summed traces, etc.) |
| 33 - 34 | | Number of horizontally stacked traces yielding this trace (1 is one trace, 2 is two stacked traces, etc. ) |

| Byte Numbers | Should Record | Field Description |
|---|---|---|
| 35 - 36 | | Data use: 1 = production 2 = test |
| 37 - 40 | | Distance from source point to receiver group (negative if opposite to direction in which line is shot) |
| 41 - 44 | | Receiver group elevation; all elevations above sea level are positive and below sea level are negative |
| 45 - 48 | | Surface elevation at source |
| 49 - 52 | | Source depth below surface (a positive number) |
| 53 - 56 | | Datum elevation at receiver group |
| 57 - 60 | | Datum elevation at source |
| 61 - 64 | | Water depth at source |
| 65 - 68 | | Water depth at group |
| 69 - 70 | | Scaler to be applied to all elevations and depths specified in bytes 41-68 to give the real value Scaler = 1, +10, +100, +1000, or +10,000. If positive, scaler is used as a multiplier; if negative. scaler is used as a divisor. |
| 71 - 72 | | Scaler to be applied to all coordinates specified in bytes 73-88 to give the real value Scaler = 1, +10, +100, +1000, or +10,000 If positive, scaler is used as a multiplier; if negative, scaler is used as divisor. |
| 73 - 76 | | Source coordinate X; see Note below |
| 77 - 80 | | Source coordinate Y; see Note below |
| 81 - 84 | | Group coordinate X; see Note below |

| Byte Numbers | Should Record | Field Description |
|---|---|---|
| 85 - 88 | | Group coordinate Y; see Note below<br><br>Note: If the coordinate units are in seconds of arc, the X values represent longitude and the Y values latitude. A positive value designates the number of seconds east of Greenwich Meridian or north of the equator and a negative value designates the number of seconds west or south. |
| 89 - 90 | | Coordinate units: 1 = length (meters or feet)<br>2 = seconds of arc |
| 91 - 92 | | Weathering velocity |
| 93 - 94 | | Subweathering velocity |
| 95 - 96 | | Uphole time at source |
| 97 - 98 | | Uphole time at group |
| 99 - 100 | | Source static correction |
| 101 - 102 | | Group static correction |
| 103 - 104 | | Total static applied (zero if no static has been applied) |
| 105 - 106 | | Lag time A. Tune in_ms between end of 240-byte trace identification header and time break. Positive if time break occurs after end of header, negative if time break occurs before end of header. Time break is defined as the initiation pulse which may be recorded on an auxiliary trace or as otherwise specified by the recording system. |

| Byte Numbers | Should Record | Field Description |
|---|---|---|
| 107-108 | | Lag time B. Tune in ms between time break and the initiation time of the energy source. May be positive or negative. |
| 109 - 110 | | Delay recording time. Tune in ms between initiation time of energy source and time when recording of data samples begins (for deep water work if data recording does not start at zero time). |
| 111 - 112 | | Mute time: start |
| 113 - 114 | | Mute time: end |
| 115 - 116 | *R* | Number of samples in this trace |
| 117 - 118 | *R* | Sample interval in ms for this trace |
| 119 - 120 | | Gain type of field instruments: 1 = fixed<br>2 = binary<br>3 = floating point<br>4 - N = optional use |
| 121 - 122 | | Instrument gain |
| 123 - 124 | | Instrument gain constant |
| 123 - 124 | | Instrument early or initial gain (dB) |
| 125 - 126 | | Correlated 1 = yes<br>2 = no |
| 127 - 128 | | Sweep frequency at start |
| 129 - 130 | | Sweep frequency at end |
| 131 - 132 | | Sweep length in ms |

| Byte Numbers | Should Record | Field Description |
|---|---|---|
| 133 - 134 | | Sweep type: 1 = linear<br>2 = parabolic<br>3 = exponential<br>4 = other |
| 135 - 136 | | Sweep trace taper length at start in ms |
| 137 - 138 | | Sweep trace taper length at end in ms |
| 139 - 140 | | Taper type: 1 = linear<br>2 = cos2<br>3 = other |
| 141 - 142 | | Alias filter frequency. if used |
| 143 - 144 | | Alias filter slope |
| 149 - 150 | | Low cut frequency. if used |
| 151 - 152 | | High cut frequency, if used |
| 153 - 154 | | Low cut slope |
| 155 - 156 | | High cut slope |
| 157 - 158 | | Year data recorded |
| 159 - 160 | | Day of year |
| 161 - 162 | | Hour of day (24-hour clock) |
| 163 - 164 | | Minute of hour |
| 165 - 166 | | Second of minute |
| 167 - 168 | | Time basis code: 1 = local 2 = GMT 3 = other |
| 169 - 170 | | Trace weighting factor; defines as 2-N volts for the least significant bit (N = 0. 1 .... 32. 767) |
| 171 - 172 | | Geophone group number of roll switch position one |

| Byte Numbers | Should Record | Field Description |
|---|---|---|
| 173 - 174 | | Geophone group number of trace number one within Original field record |
| 175 - 176 | | Geophone group number of last trace within original field record |
| 177 - 178 | | Gap size (total number of groups dropped) |
| 179 - 180 | | Overtravel associated with taper at beginning or end of line: 1 = down (or behind)<br>2 = up (or ahead) |
| 181 - 240 | | Unassigned; for optional information |

# Appendix C

SEISMIC LINE OBJECT

C++ CLASS

DEFINITION

```
#ifndef SEIS_LINE
#include <fstream.h>
#include <Xm/Xm.h>
class Seis_trac;                    // forward declaration for the seismic trace object.
class Seis_line
{
    friend class Line_list;
    public:
      ifstream        Linefile;     // input stream class object associated with the line file.
      char    *Filename;    // pointer to the line file name
       Seis_trac      *Firstrace;   // pointer to the first trace.

       // Function definitions.
       Seis_line ( );                          // constructor
       ~Seis_line ( );                         // destructor
       void    Add_fname(char *fname);   // set the seismic line file name.
       void    Get_fname(char *fname);   // return the seismic line file name.
       // set the seismic line number.
       void    Get_linum(long & linenumb){linenumb = Line_number;}
       void    Set_linum(long linum ) {Line_number = linum;}
       int     RtnLinNum( ) {return( Line_number); }
       void    Get_units (long &Units) {Units = units ; }          // set the unit type.
       void    Set_units (long unt) {units = unt; }                // get the unit type.
       void    Get_Dtype (long &Dtype) {Dtype = Data_type ; }  //set the data type
       void    Set_Dtype(long data_type){Data_type = data_type;}//get the data type
         // time cover methods for the seismic method.
       void    Add_Ftime(short Year,short Day,short Hour,short Min,short Sec);
       void Get_Ftime(short &Year, short &Day, short &Hour, short &Min, short &Sec);
       void    Set_Ftime_sec(unsigned long Sec) { First_time = Sec;}
       void    Get_Ftime_sec(unsigned long &secs) {secs = First_time;}
       void    Add_Ltime(short Year,short Day, short Hour,short Min,short Sec);
       void Get_Ltime(short &Year, short &Day, short &Hour, short &Min, short &Sec);
       void    Set_Ltime_sec(unsigned long Sec) { Last_time = Sec ;}
       void    Get_Ltime_sec(unsigned long &secs) {secs = Last_time ;}
       void    Drawline( Widget LineWindow, double xmin, double ymin,
                                 GC gc, double scalex, double scaley, int label_line);
       void    Get_colour(long &color) {color = Colour; }
       void    Set_colour(long color ) { Colour = color; }
       int     RtnColour( ) {return( Colour); }
       void    Set_cover(double Xcoord, double Ycoord);
       void    Get_cover(double &minx,double &miny,double &maxx,double &maxy);
```

```
        protected:      //          variable definitions.
          long          Line_number; // the line number from the binary header.
          long          Colour;      // colour used to display the seismic line
          long          Data_type;   // the trace data type. 1 = floating point 4 byte
                                     // 2 = fixed point 4 bytes, 3 = fixed point 2 bytes
                                     // 4 = fixed point with gain control 4 bytes
          long          units;       // the type of coordinates used. 2 = feet; 1 = metres
          unsigned long First_time; // the time that the first trace was record.
          short         First_year;
          unsigned long Last_time ; // the time that the last trace was recorded.
          short         Last_year ;
          double        Xmin, Ymin;  // the SW corner of the cover.
          double        Xmax, Ymax;  // the NE corner of the cover.
        };
#define SEIS_LINE
#endif
```

# Appendix D

# SEISMIC TRACE OBJECT

# C++ CLASS

# DEFINITION

```
#include <fstream.h>                    //      INCLUDE Files:
#ifndef SEIS_TRAC
#define SEIS_TRAC
class Seis_line;                                // forward declaration for the Seismic line
object.
class Seis_trac                         //      Class definition
{
   public:
        Seis_trac ( );                          // constructor.
        Seis_trac(Seis_line *lptr);                     // constructor.
        Seis_trac(Seis_line *lptr, long Trace_num);  // constructor.
        void            Add_line_num(Seis_line *lptr);
        void            Add_trace_num(int trace_num);
        void            Add_byte_pos(streampos bytepos);
        void            Add_xyz(double xpos, double ypos, double zpos);
        void            Add_time(int year, int day, int hour, int min, int sec);
        void            SettraceCounter(unsigned long Tracecnt) {traceId = Tracecnt;}
        void            Get_line_num(Seis_line *line);
        Seis_line *     RtnLine_ptr() {return(line_obj); }
        long    Get_trace_num();
        streampos       Get_byte_pos();
        void            Get_xyz(double &xpos, double &ypos, double &zpos);
        void            Get_time(int &year, int &day, int &hour, int &min, int &sec);
        unsigned long GetUnsignedLongTime(){return(Trace_time); }
        void            GettraceCounter(unsigned long &Tracecnt){ Tracecnt = traceId;}
        void            Display();

        friend int      Cmptraces(Seis_trac *traceRec1, Seis_trac *traceRec2);
        Seis_trac *next;

   protected:               // variable definitions.
        Seis_line       *line_obj;      // pointer to the line object.
        long            Trace_number; // the traces number.
        unsigned  long traceId;         // seismic index trace ID
        streampos       Byteoff;        // the position of the start of the trace in SEGY file
        int             Tyear;          // the year that the trace was recorded.
        unsigned long  Trace_time ;  // the time that the last trace was record  seconds
        double          Xcord, Ycord, Zcord; // trace's reference coordinates.

};
#endif
```

# Appendix E

# SEISMIC INDEX OBJECT

# C++ CLASS

# DEFINITION

```
#include <Xm/Xm.h>          // Widget definition.
#include "coords.h"         // XYpnt and Zpnt objects.
#include "DAGraticule.h"    // graticule grid image.
#include "Project.h"        // The project that are contained in the current INDEX.
#include "PRquadtree.h"     // The PR quadtree class definition.
#include "Seis_line.h"      // pointer to the line object.
#include "Seis_trac.h"      // pointer to the trace object.
#include "SGX_Manager.h"    // the Seismic index file IO class
#include "Time_tree.h"      // pointer to the time range search structure.
#include "Time_obj.h"               // object to store the input time.
//   Class definition
class Seisindex
{
   public:
        ifstream Projfile;
        Seisindex();                    // constructor.
        Seisindex( char* Fname);        // constructor.
        int Graphics;                   // true if graphics are to be used.
        int inputmode;                  // defined the input mode for data.
        GC    QueryGC;                  // graphics content for the lines drawing.
        GC    OverviewGC ;              // graphics content for the lines drawing.
        Widget shell;                   // the main widget for the graphics application.
        Widget Querycanvas;             // drawing area for the selected lines.
        Widget Overviewcanvas;          // drawing area for the inserted seismic lines.
        Widget message;                 // message area for the application.
        Pixmap QueryPixmap;
        DAGraticule  ProjectGrid, Grid;
        Index_Project_List Projectnames;

        int             Add_min_time(int year, int day, int hour, int min, int sec);
        int             Add_max_time(int year, int day, int hour, int min, int sec);
        int             Add_line_cover(double Xmin, double Ymin, double Zmin,
                            double Xmax, double Ymax, double Zmax);
        int             UpdateLineCover(int Mode, double &Xmin, double &Ymin,
                            double &Xmax, double &Ymax);
        int             UpdateTimeCover(int Mode, short &Syear, short &Sday,
                                    short &Shour, short &Smin, short &Ssec,
                                    short &Eyear, short &Eday, short &Ehour,
                                    short &Emin, short &Esec);
        int             UpdateNumLines();
        void    Set_proj_cov(double &xmin, double &ymin, double &zmin,
```

```
                double &xmax, double &ymax, double &zmax);
void          Set_proj_tim(Time_obj Stime, Time_obj Etime);
int           Incrm_num_line(int mode);
int           Display_query_boundary(XYZpnt MinPT, XYZpnt MaxPT);
int           UpdateIndexCover(double swX, double swY, double neX,
                        double neY, double Mindepth, double Maxdepth);
int           RtnIndexCover(double &swX, double &swY, double &neX,
                        double &neY, double &Mindepth, double &Maxdepth);
void    Get_proj_cov(double &xmin, double &ymin, double &zmin,
                        double &xmax, double &ymax, double &zmax);
void          Get_proj_tim(Time_obj &Stime, Time_obj &Etime);
int           DisplayLINES(line_list lineptrs);
int           DisplayLINES(Seis_line * SeisLineRecord);
int           DisplaySHOTS(line_list lineptrs);
void          DisplayQTOAQ(){Quadtree->DisplayOAQ( ); }
void          DisplayQT(){Quadtree->Display( ); }
void          DisplayTtree() {SearchTree.Display (Tree_node ); }
int           LoadProject(char * filename);
int           LoadSeismicIndex();
int           RedrawOverview ( );
int           SaveProject( );
int           SaveSeismicIndex( );
int           Settings();
int           SetQuadtree(void);

static void     DefineBoundarydialogCB( Widget filewidget, XtPointer
                        clientdata, XtPointer calldata);

void          SetWaitcursor();
void          UnSetWaitcursor();
int           Initialize();
int           Insert(Seis_line *line);
int           Insert(Seis_trac *trace);
int           Delete(Seis_line *line);
int           Delete(Seis_trac *trace);
int     Range_Search_2D(XYZpnt minxy, XYZpnt maxxy, line_list *lineptrs);
int     Time_Search(Time_obj mintime, Time_obj maxtime, line_list *Qlines );
int           SOgetline(char *bufrec);
int           Getcolourmap ( char * promptstg, char * colourMapName );
int           Getfilename  ( char * promptstg, char * FileName     );
int           Getnumber ( int & number );
int           Getboundary(XYZpnt & minxy, XYZpnt & maxxy, int Update);
```

```
        int             GetTimes  (Time_obj &mintime, Time_obj &maxtime);
        int             createcolourmap();
        int             Load_list(char * buf);
        int             Reset();
    private:
        char            * Filename;
        double          XswIndex;     // the index's X south west corner
        double          YswIndex;     // the index's Y south west corner
        double          Depthmin;     // the index's Z minimum depth
        double          XneIndex;     // the index's X south west corner
        double          YneIndex;     // the index's Y south west corner
        double          Depthmax;     // the index's Z minimum depth

        // Current tight Index line cover.
        double LineXsw, LineYsw, LineXne, LineYne; // tight line data cover
        unsigned long  LineStime, LineEtime;
        unsigned long   Timestart;   // the start time for the project in seconds; for
                                     // day + hour + min + sec
        unsigned long   Timeend;     // the start time for the project in seconds; for
                                     // day + hour + min + sec
        int             Num_lines;   // the number of seismic lines that have been loaded.
        int             PruneTree;   // flag to indicate if the line ID's should be stored in
                                     // the tree nodes.
        Avl_node        *Tree_node; // the root node for the tree.
        PR_Quadtree     *Quadtree;   // Pointer to the PR quadtree.
        Time_tree       SearchTree;  // pointer to the time search tree.
        SGX_DiskIO      IndexFile;   // seismic index disk IO class.
};
```

# Appendix F

# AVL TIME TREE

# C++ CLASS

# DEFINITION

```
#ifndef TIME_TREE
#include <fstream.h>
#include "Time_obj.h"
#include <stdio.h>
#include "coords.h"
#include "Line_list.h"
#include "Seis_line.h"
#include "SGX_Manager.h"
class Avl_node
{
    friend void SRleft(Avl_node *& K2);   // Function to perform a single left rotation.
    friend void DRleft(Avl_node *& K3);   // Function to perform a double left rotation.
    friend void SRright(Avl_node *& K2);  // Function to perform a single right rotation.
    friend void DRright(Avl_node *& K3); // Function to perform a double right rotation.
        friend  long MaxOf(long height1, long height2);    // function to determine the
maximum                                                    // of two numbers.

    friend  long Get_linum(void);
    friend  unsigned long Get_Ftime_sec(void);
    friend  unsigned long GetLtime_sec(void);
    public:
        Avl_node *Left;       // pointer to the left child.
        Avl_node *Right;      // pointer to the right child.
        int    Height;        // the height of the node.
        Seis_line * lineobj;  // pointer to the line object.

        Avl_node( Seis_line *& Seismic_line );     // constructors for the node.
        Avl_node(void);                            // constructors for the node.
        unsigned long Get_stime(void);     // return start time in seconds
        unsigned long Get_etime(void);     // return end time in seconds
        void     Range_search_2D(Line Querywindow, line_list * line);
        void     Writefilenames(ofstream & PrjFile); // write the filenames to a project file.
        void     ReDrawLines( );    // redraw the line in an X window.
        void     WalkAVLTree(long &numberNodes ); // count the # nodes in the tree
        void     Display(int child, int height);  // display the contents of the node
        void     Remove( );         // delete the tree node.
        int      WriteSeismicIndex(int Nodepath, SGX_DiskIO &IndexFile );
        int      ReadSeismicIndex(SGX_DiskIO &IndexFile );
```

```
};

class Time_tree : public Avl_node
{
    friend long MaxOf(long height1, long height2);
    friend Avl_node * Find_Min (Avl_node * Treenode);

    public:
        void Insert(Seis_line *& LineData, Avl_node *& Tnode );
        int  RemoveAVL(Time_obj Start_time, Seis_line *& Sline, Avl_node *&
                                    Treenode, int CpLine);
        void    Time_search(Time_obj Stime, Time_obj Ftime, Avl_node *&
Treenode,
                            line_list * lines);
        void    Display(Avl_node * Tnode);
        int     Remove (Avl_node * & Tnode);
                Time_tree(void){tree_node = NULL;} // constructor
        void    Range_search_2D(Line Querywindow,Avl_node *& Tnode,
                                    line_list * line);
        void        Writefilenames(Avl_node *& Tree_node, ofstream & PrjFile);
        void        ReDrawLines( Avl_node *& Tree_node);
        int     WriteSeismicIndex(Avl_node *& AvlTree_node, SGX_DiskIO
                                    &IndexFile );
        int     ReadSeismicIndex(Avl_node *& AvlTree_node, SGX_DiskIO
                                    &IndexFile,  long &ID_key);
    protected:
        Avl_node * tree_node;
};
#define TIME_TREE
#endif
```

# Appendix G

# PR QUADTREE

# C++ CLASS

# DEFINITION

```
#ifndef PR_QUADTREE
#define PR_QUADTREE
#include <stdio.h>
#include "coords.h"
#include "Line_list.h"
#include "QuadNode.h"
#include "Quadtree.h"
#include "Seis_trac.h"
#include "SGX_Manager.h"
class PR_Quadtree
{
    public:
        PR_Quadtree( );                 // default constructor.
        PR_Quadtree(double CentreX, double CentreY, double Xdist, double Ydist );
        int     PR_QTcompare(PR_QTnode * quadnode, double X, double Y);
        int     PR_QTcompare(double X, double Y, double Xcentre, double Ycentre);
        void    PR_QTinsert(double Xpnt, double Ypnt);
        void    PR_QTinsert(Seis_trac * traceobject);
        void        PR_QTinsertLine(Seis_trac * newtrace);
        void    PR_QTdelete(Seis_trac * traceobject);
        void    PR_QTdelete(PR_QTnode newnode, PR_QTnode parent, double
                    Xcentre, double Ycentre, double LengthX, double LengthY);
        int     PRQTsearch (line_list * line, XYZpnt SWcorner, XYZpnt NEcorner);
        int     RemoveNode(PR_QTnode * &Cntnode, Seis_trac * del_trace);
        int     RemoveNode(PR_QTnode * &Cntnode);
        void        Remove( );
        int     EqualKey(double X1, double Y1, double X2, double Y2) ;
        int     CQUAD(int quadrant); // clockwise quadrant.
        int     CCQUAD(int quadrant);  // counter clockwise quadrant.
        Seis_line * CheckLineID(PR_QTnode * Rootnode, PR_QTnode * newnode);
        void    VerifyLineID(PR_QTnode * &CurrentNode, PR_QTnode * newnode);
        void    DisplayOAQ( );
        int     WriteSeismicIndex(SGX_DiskIO  &IndexFile);
        int     ReadSeismicIndex(SGX_DiskIO  &IndexFile);
    private:
        PR_QTnode *Rootnode;
        double Xcentre, Ycentre, Xrange, Yrange;  // quadtree centroid and lengths
        double QTolerence;
        double southWestX, southWestY, northEastX, northEastY;
};
#endif
```

# Appendix H

# PR QUADTREE NODE

# C++ CLASS

# DEFINITION

```
#ifndef QUADNODE
#define QUADNODE
#include <stdio.h>
#include "coords.h"
#include "Line_list.h"
#include "Quadtree.h"
#include "Seis_trac.h"
#include "Seis_line.h"
#include "SGX_Manager.h"
class NodePoint;
union PtrToData
{
        NodePoint   * Pointrecord;     // record for each trace.
        Seis_line   * SeisLineClass;   // pointer to the seismic line.
};
class PR_QTnode
{
    public:
        PR_QTnode(int NodeType);      // Constructors
        PR_QTnode(int NodeType, double Xpnt, double Ypnt);
        PR_QTnode(int NodeType,double Xpoint,double Ypoint,Seis_line * SeisLine);
        PR_QTnode(int NodeType, Seis_trac * newtrace);
        PR_QTnode(int NodeType, Seis_trac * newtrace, Seis_line * LinePtr);

        // interface functions.
        int             AddAllchildren(XYZpnt SWcorner, XYZpnt NEcorner, line_list *line
);
        int             Addchild(line_list *line );
        int             Addnode(Seis_trac * trace);
        int             ADJQUAD(int quadrant);
        int             CQUAD(int quadrant);
        int             CCQUAD(int quadrant);
        int             Checkchildren(int Quadrant);
        int             checkquad(double xsw, double ysw, double xne,
                                double yne,  XYZpnt SWcorner, XYZpnt NEcorner);
        int             ChildType(int quadrant);
        void            CreateNode(int quadrant,int NodeType, double Xpnt, double Ypnt);
        void            CreateNode(int quadrant, int nodetype, double Xpnt,
                                double Ypnt, Seis_line * lineID);
        void            Display(int level );
        void            Display_OAQ(FILE * filename, int &colcnt );
```

```
        void        RtnChild(int quadrant, PR_QTnode * &childnode);
        int         RtnType( ) { return( (int) Nodetype) ; }
        double      RTNxcord( ) { return (Xcoord); }
        double      RTNycord( ) { return (Ycoord); }
        Seis_line * RtnSeisLine();
        void        SetChild(int quadrant, PR_QTnode * newchild);
        int         RangeSearch(line_list *line, XYZpnt SWcorner, XYZpnt
                    Necorner,  double Xcent, double Ycent, double Xlen, double Ylen);
        void        Walktree(int cntlevel, int &numlevels, int &numnodes,
                            int &numblack, int &numwhite, int &numgrey);
        int         RemoveNode ( );
        int         WriteSeismicIndex (SGX_DiskIO &SeismicIndex);
        int         ReadSeismicIndex (SGX_DiskIO &SeismicIndex);


    // Public variables.
        PtrToData  DataPTR;

    private :
        // Define the pointers to the children.
        PR_QTnode * SWnode;
        PR_QTnode * NWnode;
        PR_QTnode * NEnode;
        PR_QTnode * SEnode;
        short       Nodetype;       // The node type. Black, White or Grey.
        double      Xcoord;         // The X coordinate of the data point.
        double      Ycoord;         // The Y coordinate of the data point.
};
#endif
```

# Appendix I

# C++ CODE
# FOR
# PR QUADTREE
# INSERTION ALGORITHM

/*  FUNCTION NAME :PRQTinsertLine.c++

This function performs the insertion into the PR quadtree for the shot points. The function also keeps track of the lines that the shots in the nodes subtree belong to. i.e. If a nodes subtree contains only one line then the node will point to the line that the shot in the subtree belong. If the nodes subtree contain shot that belong to more than one line then the line pointer is set to NULL. The line information will be used in the search procedure to prune children from the search and thus improve the search time. The improvement in time in the base case would be O(1) for the case where query area overlaps the quadtree extent and the quadtree contains only one line. In the worst case the search would still have to visit all node's which fall in the query rectangle as the tree is traversed. The cases that have to be considered during the insertion are:

1) BASE case 1: the tree is empty; In this case the new node becomes the root node and the data pointer field points to the linked list of the shot points.

2) Case only one node in the tree. The line identifier of the current node is checked against the new node;
   a) if they are the same then the line identifier is inserted in the newly created root node,
   b) if they are different then the root node data identifier field is set to NULL and the new shot is placed in the appropriate quadrant.

3) Shot is being inserted in a tree with multiple nodes. In this case there are several situations that can occur.
   a) The line identifier of the nodes that are walked are the same as the shot point that is to be inserted. In this situation if new grey nodes are created as the node is split then the data field will have to be set to point to the common line identifier.
   b) As the tree is traversed a node is encountered for which the data identifier field points to a different line. In this case the data identifier at the current node would be set to NULL and the traversal would continue repeating the step as required. The data identifier of the children should already be set properly from a previous insertion.

```
#include "PRquadtree.h"
#include "Quadtree.h"
#include "Seis_trac.h"
#include "Writemsg.h"
#include <iostream.h>                    // cout, cerr I/O streams.
#include <stdlib.h>
extern  int DEBUG;
void PR_Quadtree::PR_QTinsertLine(Seis_trac * newtrace      )
{
```

```
int  quad;          // The pointer to the quadrant that a child is to go
int  QuadUnode; // The pointer to the quadrant that a displaced child is to go.
int  Type;          // The node type GREY or BLACK.
double Xnew, Ynew, Xparent, Yparent, Ztemp; // cell and trace reference positions.
PR_QTnode *  TmpParent;
PR_QTnode *  Unode;
PR_QTnode *  childnode;
PR_QTnode *  newnode;
double Xcent, Ycent, LengthX, LengthY;          // centroid and length of a quadrant.
Seis_line * LineID;         // pointer to a seismic line object.
int Splitnode = 1;
// Set the initial root quadrant variables.
Xcent = Xcentre; Ycent = Ycentre; LengthX = Xrange; LengthY = Yrange;
LineID = NULL;

// if the tree is empty then create the root node and insert the trace.
if (NULL == Rootnode )
{
    Rootnode = new PR_QTnode(Black, newtrace  );
     return ;
}

newtrace->Get_xyz(Xnew, Ynew, Ztemp);// get the traces coordinates.
if( (Xnew < southWestX ) || (Ynew < southWestY ) || // check to make sure that
(Xnew > northEastX ) || (Ynew > northEastY ) )     // the trace in inside the quadtree
{  return;  }

Xparent = Rootnode->RTNxcord( );       // get the centroid for the root node.
Yparent = Rootnode->RTNycord( );
newnode = new PR_QTnode(Black, newtrace  ); // create a new node.

//if the tree contains one node (BLACK) then create a new root node.
if(( Type = Rootnode->RtnType()) != Grey) {
     // if this is a duplicate point then add the point to the collision list.
     if( EqualKey(Xnew, Ynew, Xparent, Yparent )      )
     {
          Rootnode->Addnode(newtrace);// Add the node to the collision list.
          return;
     }

     else // else create a new root node. Copy the current root and create a new root.
```

```
    {
        // Call the function to determine if the shots are from the same line.
        LineID = CheckLineID(Rootnode, newnode);
        Unode  = Rootnode;              // copy the address of the current root.

        if(LineID != NULL )// line ID's are the same; create the new node with the
        {                           // line pointer field set.
            Rootnode = new PR_QTnode(Grey, Xcentre, Ycentre, LineID);
        }
        else // traces belong to different lines so do not set the line pointer field.
        {
            Rootnode = new PR_QTnode(Grey, Xcentre, Ycentre);
        }

        // determine the quadrant that the original parent falls in
        quad = PR_QTcompare(Unode, Xcent, Ycent );
        Rootnode->SetChild(quad, Unode);// Insert old root into a quadrant.
    }
} // end of if only root node

TmpParent = Rootnode ;    // copy the current parent node
VerifyLineID(TmpParent, newnode);// Update the line object pointer field.

// determine the quadrant into which the newnode should be placed.
quad = PR_QTcompare(newnode, Xcent, Ycent );
TmpParent->RtnChild(quad, childnode);// get the address of the child and node type.
if(childnode != NULL) Type = childnode->RtnType();

// In a loop walk the tree until the correct place for the node is determined.
// While the child is not NULL and the child is an interior node traverse the tree.
while(childnode != NULL && Type == Grey )
{
    TmpParent->RtnChild(quad, TmpParent);// traverse the tree to the next level.
    Xcent = Xcent + XF[quad] * LengthX; // Update the X value of the centre.
    Ycent = Ycent + YF[quad] * LengthY; // Update the Y value of the centre.
    LengthX = LengthX / 2.0; LengthY =LengthY / 2.0 ;

    // Determine the next quadrant on the insertion path.
    quad = PR_QTcompare(newnode, Xcent, Ycent );
    TmpParent->RtnChild(quad, childnode);// Update the child pointer and the type
    if(childnode != NULL) Type = childnode->RtnType();
```

```
    VerifyLineID(TmpParent, newnode);// Update the line pointer field.

} // End of while loop.
// If the child of the parent is NULL then assign the new node as the new child.
if( NULL == childnode )          { TmpParent->SetChild(quad, newnode); }

// Else if the node is already in the tree
else if( EqualKey(Xnew,Ynew,childnode->RTNxcord( ),childnode->RTNycord( ) ) )
{
    childnode->Addnode(newtrace);
    return ;
}
// node is already occupied so subdivide it until the node can be placed in the tree.
else
{
    TmpParent->RtnChild(quad, Unode );// Copy the address of node to split.

    // determine if the line pointer field should be set in the split nodes.
    LineID = CheckLineID(Unode, newnode);

    while(Splitnode)      // split the nodes until the new node and the current child no
    {                     // longer occupy the same quadrant.
        if(LineID == NULL)// create a new node with line pointer field set.
        {
            TmpParent->CreateNode(quad, Grey, Xcent + XF[quad] * LengthX,
                Ycent + YF[quad] * LengthY);
        }
        else
        {
            TmpParent->CreateNode(quad, Grey, Xcent + XF[quad] * LengthX,
                Ycent + YF[quad] * LengthY, LineID);
        }
        TmpParent->RtnChild(quad, TmpParent);// traverse to the next level .

        Xcent = Xcent + XF[quad] * LengthX;   // Update the centre value and the
        LengthX = LengthX / 2.0;              // length of the quadrants sides.
        Ycent = Ycent + YF[quad] * LengthY;
        LengthY = LengthY / 2.0;
        quad = PR_QTcompare(newnode, Xcent, Ycent ); // get the next quadrant.
        quadUnode = PR_QTcompare(Unode, Xcent, Ycent );
```

```
            if (quad != quadUnode )break;// If the quadrants are not the same the exit.
        } // end of split loop.

        // insert the children in the correct positions.
        TmpParent->SetChild(quad, newnode);      // insert the new node.
        TmpParent->SetChild(quadUnode, Unode); // reinsert the existing node.
    } // end of inserting the new node.
}

//  Function to compare the line ID fields for two nodes.
Seis_line * PR_Quadtree::CheckLineID(PR_QTnode * Rootnode, PR_QTnode * newnode)
{
    Seis_line * LineID1, * LineID2; // pointers to the two seismic lines.

    LineID1 = Rootnode->RtnSeisLine() ;   // retrieve the Pointers to the two seismic
    LineID2 = newnode->RtnSeisLine() ;    // line objects from the nodes.

    if (LineID1 == LineID2 ) return (LineID1 );// compare the two line pointer
                                               // identifiers.
    else return (NULL);                        // line ID's are different; return NULL.
}

//  Function to determine if the current node's line ID is the same as the new nodes line
//  ID. If they are different the current nodes Line ID field is set to NULL. If the
//  current node is not grey then no change is made as the DataPTR value contains the
//  trace record link list class.
void PR_Quadtree::VerifyLineID(PR_QTnode * &CurrentNode, PR_QTnode * newnode)
{
    int  NodeType;
    Seis_line * LineID1, * LineID2; // pointers to the two seismic lines.

    // if the current node is not grey then change nothing and return.
    NodeType = CurrentNode->RtnType() ;
    if (NodeType != Grey ) return;

    // If the data pointer (SeisLineClass) field is NULL then return.
    if (CurrentNode->DataPTR.SeisLineClass == NULL) return;

    // retrieve the Pointers to the two seismic line objects from the nodes.
    LineID1 = CurrentNode->DataPTR.SeisLineClass;
```

```
    LineID2 = newnode->RtnSeisLine() ;

    if (LineID1 == LineID2 ) return; // if the same don't change anything.

    // they are different so set the field to NULL and return.
    CurrentNode->DataPTR.SeisLineClass = NULL;
}
```

# Appendix J

C++ CODE

FOR

PR QUADTREE

2D RANGE SEARCH ALGORITHM

```
//        FUNCTION NAME : PRQTsearch.c++
//        DESCRIPTION   :Modified PR quadtree search method.
#include "PRquadtree.h"
#include "coords.h"
#include "Line_list.h"
#include "segylib.h"
#include "QuadNode.h"
#include "Writemsg.h"
int PR_Quadtree::PRQTsearch (line_list *line, XYZpnt SWcorner, XYZpnt
                             NEcorner)
{
     if (Rootnode != NULL)// if the root is null then return after warning the user
    {
        // Resursively search the tree to find the target points.
        line->Reset();
        Rootnode->RangeSearch(line, SWcorner, NEcorner, Xcentre, Ycentre,
                                        Xrange, Yrange );
        return(TRUE);// return to the calling routine.
    }
}

//  FUNCTION NAME : QTRangeSearch.c++
//  DESCRIPTION   : This module contains the methods for the PR_QTnode class
#include <iostream.h>       //        INCLUDE FILES.
#include <stdio.h>
#include <stdlib.h>
#include "Writemsg.h"
#include "QuadNode.h"
#include "Quadtree.h"
extern DEBUG;
int PR_QTnode::RangeSearch(line_list *line, XYZpnt SWcorner, XYZpnt
                    NEcorner,double Xcent, double Ycent, double Xlen, double Ylen)
{
    // Local variables
    int code;
   double xmin, xmax, ymin, ymax, Qxsw, Qysw, Qxne, Qyne, Z;
   // if this is a leaf node then add the line to the list
   if (Nodetype == Black)
   {
      SWcorner.Get_XYZ(Qxsw, Qysw, Z);
      NEcorner.Get_XYZ(Qxne, Qyne, Z);
      if( Xcoord >= Qxsw && Xcoord <= Qxne && Ycoord >= Qysw &&  Ycoord <=
              Qyne)
```

```
    {
        Addchild(line);

        // return to the calling routine.
        return (TRUE);
    }
    else  return (FALSE);
}
xmin = Xcent - Xlen / 2.0;  ymin = Ycent - Ylen / 2.0;
xmax = Xcent + Xlen / 2.0; ymax = Ycent + Ylen / 2.0;

// If the North west child is not NULL then check to see if it falls in the query window.
if (NWnode !=NULL)
{     // if the query window overlaps the quadrant then recursively check the children
      code = NWnode->checkquad(xmin, Ycent, Xcent, ymax, SWcorner,
                                        NEcorner);

      // if the quadrant is inside the query window the return all children
      if (code == INSIDE)NWnode->AddAllchildren(SWcorner, NEcorner, line);

      // if the window overlaps the walk the subtree.
      if (code == OVERLAP) NWnode->RangeSearch(line, SWcorner,
            Necorner, Xcent+ XF[NWquad] * Xlen, Ycent + YF[NWquad] *
            Ylen, Xlen/2.0, Ylen/2.0);
}

// If the North east child is not NULL then check to see if it falls in the
// query window.
if (NEnode !=NULL)
{  // if the query window overlaps the quadrant then recursively check the children
    code = NEnode->checkquad(Xcent, Ycent, xmax, ymax, SWcorner,
                                        NEcorner);

    // if the quadrant is inside the query window the return all children
    if (code == INSIDE) NEnode->AddAllchildren(SWcorner, NEcorner, line);

    // if the window overlaps the walk the subtree.
    if (code == OVERLAP) NEnode->RangeSearch(line, SWcorner, NEcorner,
            Xcent+ XF[NEquad] * Xlen, Ycent + YF[NEquad] * Ylen, Xlen/2.0,
            Ylen/2.0);
}

// If the South west child is not NULL then check to see if it falls in the
```

```
    if (SWnode !=NULL)  // query window.
    {   // if the query window overlaps the quadrant then recursively check the children
        code = SWnode->checkquad(xmin, ymin, Xcent, Ycent, SWcorner, NEcorner);

        // if the quadrant is inside the query window the return all children
        if (code == INSIDE)  SWnode->AddAllchildren(SWcorner, NEcorner, line);

        // if the window overlaps the walk the subtree.
        if (code == OVERLAP) SWnode->RangeSearch(line, SWcorner, NEcorner,
                Xcent+ XF[SWquad] * Xlen, Ycent + YF[SWquad] * Ylen, Xlen/2.0,
                 Ylen/2.0);
    }

    // If the South east child is not NULL then check to see if it falls in the
    if (SEnode !=NULL)// query window.
    {   // if the query window overlaps the quadrant then recursively check the children
        code = SEnode->checkquad(Xcent, ymin, xmax, Ycent, SWcorner,
                                        NEcorner);

        // if the quadrant is inside the query window the return all children
        if (code == INSIDE) SEnode->AddAllchildren(SWcorner, NEcorner, line);

        // if the window overlaps the walk the subtree.
        if (code == OVERLAP) SEnode->RangeSearch(line, SWcorner, NEcorner,
                 Xcent+ XF[SEquad] * Xlen, Ycent + YF[SEquad] * Ylen, Xlen/2.0,
                Ylen/2.0);
     }

    // return to the calling routine.
    return(TRUE);
}
```

# Appendix K

# TIME AND COORDINATE COVERS

# FOR

# PARTITIONED SEISMIC LINES

# 17A TO 26H.

*Table K-1. Position and time specifications for lines 17A to 17I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 17 A | -96.194511 | -0.469996 | -95.232864 | 0.242827 | 3060 | 3369 | 1989 | 258 | 20 | 20 | 16 |
| | | | | | | | 1989 | 258 | 21 | 43 | 26 |
| B | -96.042252 | -0.367381 | -95.889526 | -0.266126 | 3060 | 3346 | 1989 | 258 | 21 | 43 | 36 |
| | | | | | | | 1989 | 258 | 23 | 6 | 46 |
| C | -95.889221 | -0.265921 | -95.732178 | -0.160849 | 3206 | 3350 | 1989 | 258 | 23 | 6 | 56 |
| | | | | | | | 1989 | 259 | 0 | 30 | 15 |
| D | -95.731865 | -0.160642 | -95.576820 | -0.051351 | 3232 | 3333 | 1989 | 259 | 0 | 30 | 25 |
| | | | | | | | 1989 | 259 | 1 | 53 | 45 |
| E | -95.576515 | -0.051116 | -95.425735 | 0.068115 | 3213 | 3353 | 1989 | 259 | 1 | 53 | 55 |
| | | | | | | | 1989 | 259 | 3 | 17 | 5 |
| F | -95.425400 | 0.068361 | -95.258446 | 0.151678 | 3227 | 3369 | 1989 | 259 | 3 | 17 | 15 |
| | | | | | | | 1989 | 259 | 4 | 40 | 35 |
| G | -95.352470 | 0.135529 | -95.232864 | 0.202454 | 3258 | 3362 | 1989 | 259 | 4 | 40 | 45 |
| | | | | | | | 1989 | 259 | 6 | 3 | 54 |
| H | -95.419487 | 0.069030 | -95.352623 | 0.189586 | 3236 | 3360 | 1989 | 259 | 6 | 4 | 4 |
| | | | | | | | 1989 | 259 | 7 | 27 | 24 |
| I | -95.423645 | 0.190064 | -95.343452 | 0.242827 | 3211 | 3351 | 1989 | 259 | 7 | 27 | 34 |
| | | | | | | | 1989 | 259 | 8 | 26 | 4 |

*Table K-2. Position and time specifications for lines 18A to 18I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 18 A | -95.343094 | 0.110067 | -95.284363 | 0.243596 | 3276 | 3371 | 1989 | 259 | 8 | 26 | 14 |
| | | | | | | | 1989 | 259 | 9 | 49 | 24 |
| B | -95.284035 | 0.112002 | -95.227203 | 0.225143 | 3247 | 3378 | 1989 | 259 | 9 | 49 | 34 |
| | | | | | | | 1989 | 259 | 11 | 12 | 43 |
| C | -95.283058 | 0.075924 | -95.232628 | 0.191600 | 3246 | 3376 | 1989 | 259 | 11 | 12 | 53 |
| | | | | | | | 1989 | 259 | 12 | 36 | 3 |
| D | -95.378540 | 0.138597 | -95.243729 | 0.235190 | 3331 | 3373 | 1989 | 259 | 12 | 36 | 13 |
| | | | | | | | 1989 | 260 | 3 | 49 | 12 |
| E | -95.373863 | 0.148126 | -95.217369 | 0.176359 | 3285 | 3373 | 1989 | 260 | 3 | 49 | 22 |
| | | | | | | | 1989 | 260 | 5 | 12 | 52 |
| F | -95.240471 | -0.050436 | -95.219429 | 0.147732 | 3226 | 3390 | 1989 | 260 | 5 | 13 | 2 |
| | | | | | | | 1989 | 260 | 6 | 36 | 21 |
| G | -95.257095 | -0.256927 | -95.240494 | -0.050829 | 3269 | 3361 | 1989 | 260 | 6 | 36 | 31 |
| | | | | | | | 1989 | 260 | 7 | 59 | 41 |
| H | -95.260239 | -0.458550 | -95.242004 | -0.257339 | 3291 | 3409 | 1989 | 260 | 7 | 59 | 51 |
| | | | | | | | 1989 | 260 | 9 | 23 | 11 |
| I | -95.241951 | -0.555635 | -95.227974 | -0.458950 | 3345 | 3418 | 1989 | 260 | 9 | 23 | 11 |
| | | | | | | | 1989 | 260 | 10 | 2 | 51 |

*Table K-3. Position and time specifications for lines 19A to 19I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 A | -95.227921 | -0.764625 | -95.202705 | -0.556040 | 3344 | 3414 | 1989 | 260 | 10 | 3 | 1 |
| | | | | | | | 1989 | 260 | 11 | 26 | 20 |
| B | -95.202660 | -0.973728 | -95.178383 | -0.765043 | 3266 | 3390 | 1989 | 260 | 11 | 26 | 30 |
| | | | | | | | 1989 | 260 | 12 | 49 | 50 |
| C | -95.178337 | -1.185097 | -95.151276 | -0.974160 | 3286 | 3368 | 1989 | 260 | 12 | 50 | 0 |
| | | | | | | | 1989 | 260 | 14 | 13 | 10 |
| D | -95.151215 | -1.586781 | -95.118874 | -1.185513 | 3253 | 3361 | 1989 | 260 | 14 | 13 | 20 |
| | | | | | | | 1989 | 260 | 16 | 52 | 8 |
| E | -95.118813 | -1.792574 | -95.086716 | -1.587181 | 3301 | 3366 | 1989 | 260 | 16 | 52 | 18 |
| | | | | | | | 1989 | 260 | 18 | 15 | 38 |
| F | -95.086639 | -2.003391 | -95.059341 | -1.792977 | 3336 | 3429 | 1989 | 260 | 18 | 15 | 48 |
| | | | | | | | 1989 | 260 | 19 | 39 | 07 |
| G | -95.059280 | -2.209985 | -95.041344 | -2.003819 | 3334 | 3414 | 1989 | 260 | 19 | 39 | 17 |
| | | | | | | | 1989 | 260 | 21 | 2 | 27 |
| H | -95.041321 | -2.416991 | -95.037155 | -2.210390 | 3376 | 3544 | 1989 | 260 | 21 | 2 | 37 |
| | | | | | | | 1989 | 260 | 22 | 26 | 47 |
| I | -95.037140 | -2.496941 | -95.034996 | -2.417396 | 3186 | 3580 | 1989 | 260 | 22 | 26 | 57 |
| | | | | | | | 1989 | 260 | 23 | 0 | 7 |

*Table K-4. Position and time specifications for lines 20A to 20I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 A | -95.034988 | -2.701429 | -95.031296 | -2.497341 | 3152 | 3605 | 1989 | 260 | 23 | 0 | 17 |
| | | | | | | | 1989 | 261 | 0 | 23 | 46 |
| B | -95.031281 | -2.910158 | -95.021149 | -2.701831 | 3565 | 3640 | 1989 | 261 | 0 | 23 | 56 |
| | | | | | | | 1989 | 261 | 1 | 47 | 26 |
| C | -95.021111 | -3.082007 | -94.982368 | -2.910580 | 3492 | 3615 | 1989 | 261 | 1 | 47 | 36 |
| | | | | | | | 1989 | 261 | 3 | 10 | 56 |
| D | -94.987434 | -3.082097 | -94.964401 | -2.947708 | 3485 | 3659 | 1989 | 261 | 3 | 11 | 6 |
| | | | | | | | 1989 | 261 | 4 | 34 | 16 |
| E | -95.079239 | -3.094397 | -94.963203 | -2.990831 | 3491 | 3662 | 1989 | 261 | 4 | 34 | 26 |
| | | | | | | | 1989 | 261 | 5 | 58 | 5 |
| F | -95.084633 | -3.094664 | -95.050217 | -2.951183 | 3492 | 3596 | 1989 | 261 | 5 | 58 | 15 |
| | | | | | | | 1989 | 261 | 7 | 21 | 35 |
| G | -95.104584 | -3.062332 | -95.047287 | -2.973177 | 3492 | 3626 | 1989 | 261 | 7 | 21 | 45 |
| | | | | | | | 1989 | 261 | 8 | 44 | 55 |
| H | -95.103500 | -3.030202 | -94.992157 | -2.997164 | 3494 | 3593 | 1989 | 261 | 8 | 45 | 5 |
| | | | | | | | 1989 | 261 | 16 | 30 | 26 |
| I | -95.034065 | -2.997606 | -94.954750 | -2.995995 | 3495 | 3595 | 1989 | 261 | 16 | 30 | 36 |
| | | | | | | | 1989 | 261 | 17 | 6 | 26 |

*TableK-5. Position and time specifications for lines 21A to 21I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 A | -94.954369 | -3.002132 | -94.756721 | -2.996772 | 2989 | 3579 | 1989 | 261 | 17 | 6 | 36 |
| | | | | | | | 1989 | 261 | 18 | 29 | 46 |
| B | -94.756317 | -3.015772 | -94.555496 | -3.002152 | 3087 | 3665 | 1989 | 261 | 18 | 29 | 56 |
| | | | | | | | 1989 | 261 | 19 | 53 | 16 |
| C | -94.555099 | -3.031727 | -94.355911 | -3.015810 | 3054 | 3689 | 1989 | 261 | 19 | 53 | 26 |
| | | | | | | | 1989 | 261 | 21 | 16 | 46 |
| D | -94.355507 | -3.039669 | -94.155472 | -3.031751 | 3417 | 3774 | 1989 | 261 | 21 | 16 | 56 |
| | | | | | | | 1989 | 261 | 22 | 40 | 5 |
| E | -94.155075 | -3.050785 | -93.958473 | -3.039688 | 3088 | 3625 | 1989 | 261 | 22 | 40 | 15 |
| | | | | | | | 1989 | 262 | 0 | 3 | 25 |
| F | -93.958084 | -3.060127 | -93.759377 | -3.050815 | 3220 | 3666 | 1989 | 262 | 0 | 3 | 35 |
| | | | | | | | 1989 | 262 | 1 | 26 | 45 |
| G | -93.758987 | -3.059989 | -93.563400 | -3.051002 | 3286 | 3696 | 1989 | 262 | 1 | 26 | 55 |
| | | | | | | | 1989 | 262 | 2 | 50 | 4 |
| H | -93.562981 | -3.075334 | -93.361649 | -3.057273 | 3236 | 3701 | 1989 | 262 | 2 | 50 | 14 |
| | | | | | | | 1989 | 262 | 4 | 13 | 24 |
| I | -93.361267 | -3.075835 | -93.354362 | -3.075361 | 3550 | 3556 | 1989 | 262 | 4 | 13 | 34 |
| | | | | | | | 1989 | 262 | 4 | 16 | 34 |

*Table K-6. Position and time specifications for lines 22A to 22I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 22 A | -93.353981 | -3.077318 | -93.160950 | -3.062970 | 35290 | 3589 | 1989 | 262 | 4 | 16 | 44 |
| | | | | | | | 1989 | 262 | 5 | 40 | 4 |
| B | -93.160568 | -3.062945 | -92.965042 | -3.045351 | 3509 | 3597 | 1989 | 262 | 5 | 40 | 14 |
| | | | | | | | 1989 | 262 | 7 | 3 | 34 |
| C | -92.964615 | -3.045328 | -92.754105 | -3.041333 | 3298 | 3639 | 1989 | 262 | 7 | 3 | 44 |
| | | | | | | | 1989 | 262 | 8 | 26 | 54 |
| D | -92.753662 | -3.055994 | -92.544678 | -3.044838 | 3232 | 3649 | 1989 | 262 | 8 | 27 | 4 |
| | | | | | | | 989 | 262 | 9 | 50 | 23 |
| E | -92.544273 | -3.067753 | -92.340111 | -3.056017 | 2953 | 3684 | 1989 | 262 | 9 | 50 | 33 |
| | | | | | | | 1989 | 262 | 9 | 50 | 33 |
| F | -92.339691 | -3.072990 | -92.127853 | -3.067763 | 3022 | 3587 | 1989 | 62 | 11 | 13 | 53 |
| | | | | | | | 1989 | 262 | 12 | 37 | 3 |
| G | -92.127426 | -3.079864 | -91.922760 | -3.072990 | 3535 | 3612 | 1989 | 262 | 12 | 37 | 13 |
| | | | | | | | 1989 | 262 | 14 | 0 | 22 |
| H | -91.922348 | -3.091457 | -91.721397 | -3.079884 | 3532 | 3632 | 1989 | 262 | 14 | 0 | 32 |
| | | | | | | | 1989 | 262 | 15 | 23 | 42 |
| I | -91.721008 | -3.096644 | -91.635811 | -3.091492 | 3521 | 3620 | 1989 | 262 | 15 | 23 | 52 |
| | | | | | | | 1989 | 262 | 15 | 59 | 52 |

*Table K-7. Position and time specifications for lines 23A to 23I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 23 A | -91.635422 | -3.107574 | -91.436317 | -3.096665 | 3522 | 3636 | 1989 | 262 | 16 | 0 | 2 |
| | | | | | | | 1989 | 262 | 17 | 23 | 32 |
| B | -91.435913 | -3.114819 | -91.233070 | -3.107594 | 3473 | 3593 | 1989 | 262 | 17 | 23 | 42 |
| | | | | | | | 1989 | 262 | 18 | 46 | 51 |
| C | -91.232674 | -3.114869 | -91.031059 | -3.103955 | 3394 | 3584 | 1989 | 262 | 18 | 47 | 1 |
| | | | | | | | 1989 | 262 | 20 | 10 | 11 |
| D | -91.030647 | -3.103924 | -90.831322 | -3.094961 | 3272 | 3483 | 1989 | 262 | 20 | 10 | 21 |
| | | | | | | | 1989 | 262 | 21 | 33 | 31 |
| E | -90.830910 | -3.171968 | -90.764824 | -3.072899 | 3287.00 | 3365.00 | 1989 | 262 | 21 | 33 | 41 |
| | | | | | | | 1989 | 262 | 23 | 24 | 41 |
| F | -90.826767 | -3.145480 | -90.797867 | -3.039950 | 3278 | 3355.00 | 1989 | 262 | 23 | 24 | 51 |
| | | | | | | | 1989 | 263 | 0 | 48 | 0 |
| G | -90.797516 | -3.147900 | -90.735168 | -3.081095 | 3291 | 3348.00 | 1989 | 263 | 0 | 48 | 10 |
| | | | | | | | 1989 | 263 | 2 | 11 | 50 |
| H | -90.875008 | -3.115520 | -90.795464 | -3.075990 | 3267 | 3342.00 | 1989 | 263 | 2 | 12 | 0 |
| | | | | | | | 1989 | 263 | 3 | 28 | 40 |

**Table K-8.** ***Position and time specifications for lines 24A to 24I.***

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 24 A | -90.852531 | -3.159670 | -90.785156 | -3.058192 | 3286 | 3379 | 1989 | 263 | 9 | 15 | 53 |
| | | | | | | | 1989 | 263 | 10 | 39 | 12 |
| B | -90.784767 | -3.062333 | -90.643700 | -2.967100 | 3277 | 3336 | 1989 | 263 | 10 | 39 | 22 |
| | | | | | | | 1989 | 263 | 12 | 2 | 32 |
| C | -90.643425 | -2.966789 | -90.511284 | -2.798116 | 3279 | 3279 | 1989 | 263 | 12 | 2 | 42 |
| | | | | | | | 1989 | 263 | 13 | 25 | 52 |
| D | -90.511047 | -2.797775 | -90.386169 | -2.629336 | 3135 | 3279 | 1989 | 263 | 13 | 26 | 2 |
| | | | | | | | 1989 | 263 | 14 | 49 | 21 |
| E | -90.385910 | -2.628991 | -90.262398 | -2.458957 | 3131 | 3233 | 1989 | 263 | 14 | 49 | 31 |
| | | | | | | | 1989 | 263 | 16 | 12 | 41 |
| F | -90.144981 | -2.286548 | -90.024200 | -2.107699 | 3149 | 3265 | 1989 | 263 | 17 | 36 | 21 |
| | | | | | | | 1989 | 263 | 18 | 59 | 31 |
| G | -90.091347 | -2.107341 | -90.021118 | -1.909005 | 3183 | 3255 | 1989 | 263 | 18 | 59 | 41 |
| | | | | | | | 1989 | 263 | 20 | 22 | 50 |
| H | -90.058899 | -1.908740 | -89.971893 | -1.836223 | 3185 | 3251 | 1989 | 263 | 20 | 23 | 0 |
| | | | | | | | 1989 | 263 | 21 | 8 | 40 |

*Table K-9. Position and time specifications for lines 25A to 25I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | -95.635422 | -2.107574 | -95.436317 | -2.096665 | 3522 | 3636 | 1989 | 262 | 16 | 0 | 2 |
| | | | | | | | 1989 | 262 | 17 | 23 | 32 |
| B | -95.435913 | -2.114819 | -95.233070 | -2.107594 | 3473 | 3593 | 1989 | 262 | 17 | 23 | 42 |
| | | | | | | | 1989 | 262 | 18 | 46 | 51 |
| C | -95.232674 | -2.114869 | -95.031059 | -2.103955 | 3394 | 3584 | 1989 | 262 | 18 | 47 | 1 |
| | | | | | | | 1989 | 262 | 20 | 10 | 11 |
| D | -95.030647 | -2.103924 | -94.831322 | -2.094961 | 3272 | 3483 | 1989 | 262 | 20 | 10 | 21 |
| | | | | | | | 1989 | 262 | 21 | 33 | 31 |
| E | -94.830910 | -2.171968 | -94.764824 | -2.072899 | 3287 | 3365 | 1989 | 262 | 21 | 33 | 41 |
| | | | | | | | 1989 | 262 | 23 | 24 | 41 |
| F | -94.826767 | -2.145480 | -94.797867 | -2.039950 | 3278 | 3355 | 1989 | 262 | 23 | 24 | 51 |
| | | | | | | | 1989 | 263 | 0 | 48 | 0 |
| G | -95.559959 | -2.109979 | -95.357254 | -2.102100 | 3291 | 3348 | 1989 | 263 | 0 | 48 | 10 |
| | | | | | | | 1989 | 263 | 2 | 11 | 50 |
| H | -95.356857 | -2.171968 | -94.770638 | -2.109993 | 3267 | 3342 | 1989 | 263 | 2 | 12 | 0 |
| | | | | | | | 1989 | 263 | 3 | 28 | 40 |

*Table K-10. Position and time specifications for lines 26A to 26I.*

| Line # | Minimum Longitude | Minimum Latitude | Maximum Longitude | Maximum Latitude | Min Depth | Max Depth | Year | Day | Hour | Minute | Second |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | -94.852531 | -3.159670 | -94.785156 | -3.058192 | 3286 | 3379 | 1989 | 263 | 9 | 15 | 53 |
| | | | | | | | 1989 | 263 | 10 | 39 | 12 |
| B | -94.784767 | -3.062333 | -94.643700 | -2.967100 | 3277 | 3336 | 1989 | 263 | 10 | 39 | 22 |
| | | | | | | | 1989 | 263 | 12 | 2 | 32 |
| C | -94.643425 | -2.966789 | -94.511284 | -2.798116 | 3279 | 3279 | 1989 | 263 | 12 | 2 | 42 |
| | | | | | | | 1989 | 263 | 13 | 25 | 52 |
| D | -94.511047 | -2.797775 | -94.386169 | -2.629336 | 3135 | 3279 | 1989 | 263 | 13 | 26 | 2 |
| | | | | | | | 1989 | 263 | 14 | 49 | 21 |
| E | -94.385910 | -2.628991 | -94.262398 | -2.458957 | 3131 | 3233 | 1989 | 263 | 14 | 49 | 31 |
| | | | | | | | 1989 | 263 | 16 | 12 | 41 |
| F | -94.144981 | -2.286548 | -94.024200 | -2.107699 | 3149 | 3265 | 1989 | 263 | 17 | 36 | 21 |
| | | | | | | | 1989 | 263 | 18 | 59 | 31 |
| G | -94.091347 | -2.107341 | -94.021118 | -1.909005 | 3183 | 3255 | 1989 | 263 | 18 | 59 | 41 |
| | | | | | | | 1989 | 263 | 20 | 22 | 50 |
| H | -94.058899 | -1.908740 | -93.971893 | -1.836223 | 3185 | 3251 | 1989 | 263 | 20 | 23 | 0 |
| | | | | | | | 1989 | 263 | 21 | 8 | 40 |

# Appendix L

# QUERY WINDOWS

# FOR

# THE EXPERIMENT

All query window bounding coordinates
are given in degrees and decimal degrees.

Table L-1. Experiment query windows for 10 percent coverage.

| Min. Longitude | Min. Latitude | Max. Longitude | Max. Latitude | # Lines |
|---|---|---|---|---|
| -95.210385 | -1.5923257 | -94.588123 | -1.2507693 | 2 |
| -93.200943 | -2.2232257 | -92.578681 | -1.8816693 | 0 |
| -95.232775 | -0.2589454 | -94.610513 | 0.0826109 | 2 |
| -94.926496 | -1.0133049 | -94.304234 | -0.6717484 | 0 |
| -95.496154 | -1.6510533 | -94.873892 | -1.3094969 | 2 |
| -94.012445 | -2.9140730 | -93.390183 | -2.5725166 | 0 |
| -94.133282 | -2.3197604 | -93.511020 | -1.9782040 | 2 |
| -93.196158 | -0.1488076 | -92.573896 | 0.1927487 | 0 |
| -92.574029 | -0.8182652 | -91.951767 | -0.4767088 | 0 |
| -91.824910 | -0.8137622 | -91.202648 | -0.47220581 | 0 |

Table L-2. Experiment query windows for 25 percent coverage.

| Min. Longitude | Min. Latitude | Max. Longitude | Max. Latitude | # Lines |
|---|---|---|---|---|
| -95.485501 | -1.0638081 | -93.929846 | -0.20991714 | 6 |
| -94.725928 | -1.5697008 | -93.170273 | -0.71580984 | 0 |
| -91.913957 | -2.2833137 | -90.358302 | -1.4294227 | 0 |
| -94.322285 | -1.8405110 | -92.766630 | -0.9866199 | 1 |
| -92.528964 | -1.6176242 | -90.973310 | -0.7637332 | 0 |
| -92.134579 | -0.7856492 | -90.578924 | 0.0682417 | 0 |
| -93.046550 | -0.9521693 | -91.490896 | -0.0982783 | 0 |
| -93.478822 | -1.2291555 | -91.923167 | -0.3752645 | 0 |
| -94.534789 | -2.1748802 | -92.979134 | -1.3209892 | 3 |
| -92.335261 | -2.6590392 | -90.779607 | -1.8051482 | 0 |

Table L-3. Experiment query windows for 50 percent coverage.

| Min. Longitude | Min. Latitude | Max. Longitude | Max. Latitude | # Lines |
|---|---|---|---|---|
| -94.752372 | -2.4616908 | -91.641063 | -0.7539088 | 4 |
| -95.801123 | -1.4997311 | -92.689814 | 0.2080508 | 20 |
| -93.212432 | -2.8088552 | -90.101123 | -1.1010732 | 4 |
| -94.921862 | -1.9125133 | -91.810553 | -0.2047313 | 2 |
| -93.836460 | -1.8397031 | -90.725151 | -0.1319211 | 0 |
| -96.107155 | -1.5378822 | -92.995846 | 0.1698997 | 21 |
| -93.839404 | -2.6276376 | -90.728095 | -0.9198555 | 0 |
| -94.360218 | -2.7569968 | -91.248909 | -1.0492148 | 4 |
| -93.219934 | -3.0978548 | -90.108625 | -1.3900728 | 21 |
| -96.009828 | -2.6269600 | -92.898519 | -0.9191780 | 21 |

Table L-4. Experiment query windows for 75 percent coverage.

| Min. Longitude | Min. Latitude | Max. Longitude | Max. Latitude | # Lines |
|---|---|---|---|---|
| -94.771030 | -2.7946528 | -90.104067 | -0.23297978 | 10 |
| -96.009638 | -2.6833267 | -91.342675 | -0.12165368 | 29 |
| -95.802405 | -2.6854375 | -91.135441 | -0.12376450 | 28 |
| -95.826238 | -2.7485536 | -91.159274 | -0.18688056 | 28 |
| -95.562792 | -2.7646844 | -90.895828 | -0.20301138 | 27 |
| -95.530318 | -2.4265105 | -90.863355 | 0.13516250 | 30 |
| -95.600298 | -2.8660818 | -90.933335 | -0.30440882 | 27 |
| -95.944691 | -3.1351199 | -91.277727 | -0.57344689 | 54 |
| -95.110865 | -2.7259339 | -90.443902 | -0.16426093 | 18 |
| -95.570958 | -3.0890728 | -90.903994 | -0.52739979 | 52 |

Table L-5. Experiment query windows for 90 percent coverage.

| Min. Longitude | Min. Latitude | Max. Longitude | Max. Latitude | # Lines |
|---|---|---|---|---|
| -96.042169 | -2.9077978 | -90.441813 | 0.16620977 | 43 |
| -96.051399 | -3.0548775 | -90.451042 | 0.01913005 | 56 |
| -96.005005 | -3.0702318 | -90.404649 | 0.00377581 | 61 |
| -96.171703 | -2.8689379 | -90.571347 | 0.20506966 | 43 |
| -95.946476 | -2.9495660 | -90.346120 | 0.12444163 | 43 |
| -95.738644 | -2.9409768 | -90.138288 | 0.13303084 | 42 |
| -96.049348 | -2.8516553 | -90.448991 | 0.22235231 | 45 |
| -95.592645 | -2.8855743 | -89.992289 | 0.18843329 | 45 |
| -95.926004 | -2.9060154 | -90.325648 | 0.16799223 | 44 |
| -95.690864 | -2.9417273 | -90.090508 | 0.13228032 | 42 |

# Appendix M

# SEARCH TIMES

# FOR

# THE EXPERIMENT

All search times are given in microseconds.

Table M-1. Search times for 10% coverage using modified data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 83000 | 43000 | 86000 | 48000 | 89000 | 47000 | 90000 | 40000 | 39000 | 54000 |
| 2 | 95000 | 40000 | 95000 | 44000 | 89000 | 41000 | 88000 | 37000 | 43000 | 32000 |
| 3 | 87000 | 39000 | 73000 | 43000 | 94000 | 43000 | 92000 | 39000 | 38000 | 50000 |
| 4 | 90000 | 44000 | 96000 | 42000 | 70000 | 39000 | 90000 | 34000 | 37000 | 37000 |
| 5 | 84000 | 39000 | 86000 | 39000 | 92000 | 40000 | 83000 | 33000 | 30000 | 36000 |
| 6 | 85000 | 37000 | 101000 | 48000 | 94000 | 45000 | 83000 | 38000 | 43000 | 33000 |
| 7 | 79000 | 39000 | 99000 | 45000 | 78000 | 38000 | 90000 | 42000 | 36000 | 45000 |
| 8 | 98000 | 42000 | 81000 | 40000 | 84000 | 41000 | 100000 | 41000 | 49000 | 42000 |
| 9 | 85000 | 43000 | 94000 | 47000 | 96000 | 42000 | 84000 | 40000 | 43000 | 34000 |
| 10 | 103000 | 44000 | 86000 | 46000 | 83000 | 36000 | 86000 | 34000 | 36000 | 45000 |
| **AVG** | **88900** | **41000** | **89700** | **44200** | **86900** | **41200** | **88600** | **37800** | **39400** | **40800** |
| **σ** | **7148** | **2366** | **8343** | **3027** | **7790** | **3092** | **4883** | **3027** | **5004** | **6860** |

Table M-2. Search times for 10% coverage using standard data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 101000 | 43000 | 89000 | 44000 | 97000 | 43000 | 96000 | 41000 | 39000 | 35000 |
| 2 | 87000 | 35000 | 93000 | 46000 | 98000 | 43000 | 109000 | 40000 | 39000 | 36000 |
| 3 | 102000 | 38000 | 90000 | 43000 | 87000 | 38000 | 100000 | 43000 | 34000 | 29000 |
| 4 | 96000 | 42000 | 100000 | 36000 | 95000 | 52000 | 86000 | 40000 | 42000 | 45000 |
| 5 | 97000 | 41000 | 93000 | 40000 | 88000 | 41000 | 95000 | 35000 | 37000 | 47000 |
| 6 | 98000 | 42000 | 82000 | 36000 | 88000 | 40000 | 90000 | 39000 | 45000 | 42000 |
| 7 | 97000 | 40000 | 93000 | 38000 | 88000 | 38000 | 86000 | 41000 | 39000 | 25000 |
| 8 | 97000 | 34000 | 102000 | 35000 | 88000 | 44000 | 101000 | 41000 | 30000 | 37000 |
| 9 | 105000 | 42000 | 101000 | 42000 | 96000 | 41000 | 101000 | 42000 | 35000 | 37000 |
| 10 | 99000 | 39000 | 95000 | 41000 | 99000 | 35000 | 91000 | 38000 | 34000 | 32000 |
| **AVG** | **97900** | **39600** | **93800** | **40100** | **92400** | **41500** | **95500** | **40000** | **37400** | **36500** |
| **σ** | **4504** | **2939** | **5810** | **3562** | **4716** | **4365** | **7032** | **2145** | **4128** | **6515** |

Table M-3. Search times for 25% coverage using modified data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 80000 | 43000 | 50000 | 87000 | 39000 | 50000 | 30000 | 39000 | 90000 | 43000 |
| 2 | 85000 | 40000 | 39000 | 86000 | 36000 | 36000 | 42000 | 46000 | 96000 | 37000 |
| 3 | 85000 | 47000 | 43000 | 83000 | 44000 | 36000 | 38000 | 42000 | 89000 | 53000 |
| 4 | 92000 | 34000 | 44000 | 75000 | 49000 | 37000 | 33000 | 42000 | 90000 | 40000 |
| 5 | 94000 | 39000 | 35000 | 81000 | 42000 | 31000 | 41000 | 38000 | 96000 | 39000 |
| 6 | 88000 | 42000 | 45000 | 70000 | 40000 | 44000 | 36000 | 42000 | 88000 | 39000 |
| 7 | 71000 | 44000 | 42000 | 77000 | 37000 | 37000 | 33000 | 36000 | 96000 | 39000 |
| 8 | 94000 | 42000 | 44000 | 76000 | 26000 | 38000 | 27000 | 43000 | 86000 | 38000 |
| 9 | 89000 | 43000 | 43000 | 89000 | 43000 | 44000 | 44000 | 43000 | 92000 | 41000 |
| 10 | 84000 | 48000 | 42000 | 82000 | 51000 | 34000 | 33000 | 42000 | 97000 | 41000 |
| **AVG** | **86200** | **42200** | **42700** | **80600** | **40700** | **38700** | **35700** | **41300** | **92000** | **41000** |
| **σ** | **6660** | **3789** | **3689** | **5713** | **6694** | **5349** | **5216** | **2722** | **3768** | **4313** |

Table M-4. Search times for 25% coverage using standard data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 102000 | 45000 | 46000 | 83000 | 35000 | 42000 | 37000 | 39000 | 92000 | 41000 |
| 2 | 104000 | 37000 | 39000 | 77000 | 41000 | 34000 | 46000 | 37000 | 89000 | 38000 |
| 3 | 97000 | 41000 | 42000 | 86000 | 37000 | 47000 | 27000 | 36000 | 102000 | 34000 |
| 4 | 99000 | 41000 | 44000 | 91000 | 41000 | 34000 | 43000 | 42000 | 80000 | 35000 |
| 5 | 118000 | 39000 | 40000 | 70000 | 39000 | 45000 | 31000 | 44000 | 93000 | 37000 |
| 6 | 93000 | 39000 | 41000 | 74000 | 33000 | 32000 | 31000 | 39000 | 96000 | 39000 |
| 7 | 101000 | 42000 | 38000 | 82000 | 41000 | 47000 | 42000 | 37000 | 97000 | 43000 |
| 8 | 105000 | 41000 | 44000 | 77000 | 43000 | 40000 | 17000 | 39000 | 91000 | 45000 |
| 9 | 116000 | 44000 | 40000 | 80000 | 40000 | 40000 | 42000 | 35000 | 91000 | 43000 |
| 10 | 103000 | 47000 | 35000 | 70000 | 41000 | 35000 | 43000 | 40000 | 87000 | 43000 |
| **AVG** | **103800** | **41600** | **40900** | **79000** | **39100** | **39600** | **35900** | **38800** | **91800** | **39800** |
| **σ** | **7414** | **2871** | **3081** | **6434** | **2982** | **5352** | **8734** | **2600** | **5671** | **3572** |

Table M-5. Search times for 50% coverage using modified data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 76000 | 73000 | 76000 | 77000 | 41000 | 84000 | 38000 | 78000 | 95000 | 99000 |
| 2 | 88000 | 111000 | 96000 | 79000 | 44000 | 97000 | 46000 | 93000 | 93000 | 101000 |
| 3 | 89000 | 97000 | 93000 | 82000 | 46000 | 92000 | 39000 | 79000 | 94000 | 86000 |
| 4 | 72000 | 98000 | 86000 | 84000 | 47000 | 104000 | 51000 | 89000 | 105000 | 103000 |
| 5 | 81000 | 86000 | 92000 | 52000 | 41000 | 92000 | 36000 | 96000 | 87000 | 95000 |
| 6 | 77000 | 81000 | 76000 | 57000 | 40000 | 80000 | 49000 | 95000 | 95000 | 106000 |
| 7 | 91000 | 89000 | 83000 | 92000 | 40000 | 96000 | 47000 | 84000 | 96000 | 94000 |
| 8 | 77000 | 78000 | 77000 | 81000 | 45000 | 94000 | 44000 | 89000 | 110000 | 107000 |
| 9 | 83000 | 87000 | 87000 | 89000 | 43000 | 81000 | 50000 | 86000 | 97000 | 94000 |
| 10 | 81000 | 93000 | 78000 | 74000 | 37000 | 95000 | 46000 | 82000 | 92000 | 95000 |
| **AVG** | **81500** | **89300** | **84400** | **76700** | **42400** | **91500** | **44600** | **87100** | **96400** | **98000** |
| **σ** | **5937** | **10479** | **7172** | **12232** | **2973** | **7242** | **4984** | **6074** | **6232** | **6116** |

Table M-6. Search times for 50% coverage using standard data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 99000 | 114000 | 92000 | 91000 | 38000 | 106000 | 54000 | 104000 | 119000 | 100000 |
| 2 | 108000 | 99000 | 89000 | 90000 | 36000 | 119000 | 50000 | 107000 | 107000 | 116000 |
| 3 | 98000 | 109000 | 96000 | 96000 | 38000 | 105000 | 53000 | 82000 | 97000 | 114000 |
| 4 | 103000 | 111000 | 94000 | 110000 | 43000 | 128000 | 51000 | 87000 | 124000 | 109000 |
| 5 | 100000 | 110000 | 97000 | 82000 | 36000 | 110000 | 53000 | 92000 | 124000 | 126000 |
| 6 | 108000 | 115000 | 104000 | 74000 | 45000 | 109000 | 46000 | 99000 | 98000 | 98000 |
| 7 | 104000 | 98000 | 84000 | 93000 | 40000 | 113000 | 26000 | 98000 | 119000 | 100000 |
| 8 | 107000 | 111000 | 77000 | 100000 | 39000 | 112000 | 31000 | 90000 | 114000 | 120000 |
| 9 | 103000 | 108000 | 106000 | 95000 | 40000 | 120000 | 50000 | 100000 | 123000 | 117000 |
| 10 | 98000 | 105000 | 92000 | 93000 | 44000 | 110000 | 29000 | 95000 | 103000 | 121000 |
| **AVG** | **102800** | **108000** | **93100** | **92400** | **39900** | **113200** | **44300** | **95400** | **112800** | **112100** |
| **σ** | **3763** | **5459** | **8191** | **9178** | **3015** | **6765** | **10508** | **7351** | **10157** | **9375** |

Table M-7. Search times for 75% coverage using modified data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 90000 | 103000 | 102000 | 96000 | 104000 | 95000 | 89000 | 101000 | 113000 | 107000 |
| 2 | 96000 | 91000 | 106000 | 100000 | 101000 | 86000 | 99000 | 94000 | 107000 | 107000 |
| 3 | 99000 | 91000 | 103000 | 111000 | 97000 | 87000 | 108000 | 106000 | 109000 | 97000 |
| 4 | 89000 | 110000 | 101000 | 101000 | 98000 | 107000 | 95000 | 93000 | 92000 | 116000 |
| 5 | 85000 | 115000 | 112000 | 83000 | 111000 | 99000 | 95000 | 94000 | 113000 | 100000 |
| 6 | 96000 | 111000 | 99000 | 99000 | 104000 | 104000 | 98000 | 104000 | 83000 | 99000 |
| 7 | 90000 | 91000 | 98000 | 89000 | 82000 | 102000 | 96000 | 92000 | 99000 | 107000 |
| 8 | 86000 | 97000 | 103000 | 88000 | 88000 | 108000 | 100000 | 106000 | 96000 | 91000 |
| 9 | 92000 | 113000 | 102000 | 104000 | 97000 | 99000 | 98000 | 99000 | 102000 | 97000 |
| 10 | 97000 | 98000 | 105000 | 102000 | 104000 | 107000 | 86000 | 96000 | 95000 | 92000 |
| **AVG** | **92000** | **102000** | **103100** | **97300** | **98600** | **99400** | **96400** | **98500** | **100900** | **101300** |
| **σ** | **4561** | **9165** | **3754** | **8001** | **8002** | **7552** | **5713** | **5182** | **9268** | **7417** |

Table M-8. Search times for 75% coverage using standard data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 97000 | 119000 | 131000 | 120000 | 118000 | 113000 | 130000 | 145000 | 109000 | 130000 |
| 2 | 102000 | 123000 | 94000 | 127000 | 137000 | 113000 | 110000 | 137000 | 96000 | 127000 |
| 3 | 84000 | 117000 | 109000 | 114000 | 115000 | 115000 | 118000 | 137000 | 91000 | 138000 |
| 4 | 97000 | 129000 | 118000 | 121000 | 114000 | 121000 | 100000 | 139000 | 102000 | 129000 |
| 5 | 102000 | 124000 | 106000 | 120000 | 94000 | 113000 | 101000 | 148000 | 101000 | 126000 |
| 6 | 101000 | 112000 | 126000 | 105000 | 133000 | 102000 | 106000 | 135000 | 109000 | 128000 |
| 7 | 107000 | 114000 | 116000 | 118000 | 106000 | 115000 | 100000 | 142000 | 107000 | 128000 |
| 8 | 97000 | 119000 | 115000 | 116000 | 113000 | 92000 | 119000 | 146000 | 107000 | 122000 |
| 9 | 100000 | 114000 | 118000 | 133000 | 111000 | 109000 | 118000 | 137000 | 121000 | 144000 |
| 10 | 103000 | 109000 | 118000 | 127000 | 115000 | 123000 | 109000 | 138000 | 98000 | 135000 |
| **AVG** | **99000** | **118000** | **115100** | **120100** | **115600** | **111600** | **111100** | **140400** | **104100** | **130700** |
| **σ** | **5831** | **5779** | **9813** | **7409** | **11646** | **8546** | **9460** | **4294** | **7993** | **6149** |

Table M-9. Search times for 90% coverage using modified data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 102000 | 100000 | 91000 | 100000 | 94000 | 95000 | 101000 | 93000 | 104000 | 96000 |
| 2 | 95000 | 102000 | 110000 | 100000 | 86000 | 107000 | 97000 | 114000 | 107000 | 100000 |
| 3 | 106000 | 114000 | 85000 | 99000 | 90000 | 81000 | 111000 | 94000 | 105000 | 113000 |
| 4 | 104000 | 95000 | 105000 | 92000 | 98000 | 100000 | 98000 | 87000 | 95000 | 109000 |
| 5 | 92000 | 92000 | 101000 | 123000 | 111000 | 109000 | 106000 | 117000 | 93000 | 105000 |
| 6 | 113000 | 103000 | 93000 | 96000 | 104000 | 101000 | 112000 | 106000 | 99000 | 101000 |
| 7 | 99000 | 98000 | 105000 | 98000 | 106000 | 87000 | 105000 | 92000 | 91000 | 95000 |
| 8 | 113000 | 107000 | 121000 | 94000 | 92000 | 104000 | 115000 | 94000 | 106000 | 98000 |
| 9 | 92000 | 100000 | 99000 | 104000 | 93000 | 99000 | 116000 | 107000 | 87000 | 105000 |
| 10 | 100000 | 108000 | 110000 | 111000 | 98000 | 115000 | 109000 | 86000 | 93000 | 114000 |
| **AVG** | **101600** | **101900** | **102000** | **101700** | **97200** | **99800** | **107000** | **99000** | **98000** | **103600** |
| **σ** | **7228** | **6155** | **10040** | **8707** | **7400** | **9631** | **6419** | **10536** | **6782** | **6422** |

Table M-10. Search times for 90% coverage using standard data structures.

| | QW 1 | QW 2 | QW 3 | QW 4 | QW 5 | QW 6 | QW 7 | QW 8 | QW 9 | QW 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 138000 | 139000 | 123000 | 131000 | 111000 | 127000 | 123000 | 125000 | 133000 | 123000 |
| 2 | 138000 | 122000 | 146000 | 120000 | 137000 | 128000 | 130000 | 121000 | 139000 | 113000 |
| 3 | 136000 | 129000 | 128000 | 133000 | 114000 | 110000 | 122000 | 116000 | 134000 | 117000 |
| 4 | 125000 | 126000 | 131000 | 116000 | 117000 | 112000 | 134000 | 122000 | 117000 | 141000 |
| 5 | 137000 | 134000 | 147000 | 118000 | 131000 | 127000 | 123000 | 123000 | 128000 | 127000 |
| 6 | 115000 | 126000 | 164000 | 135000 | 118000 | 133000 | 137000 | 110000 | 128000 | 112000 |
| 7 | 116000 | 139000 | 126000 | 126000 | 120000 | 115000 | 118000 | 119000 | 124000 | 115000 |
| 8 | 130000 | 139000 | 137000 | 103000 | 114000 | 119000 | 138000 | 123000 | 129000 | 116000 |
| 9 | 132000 | 137000 | 147000 | 134000 | 124000 | 113000 | 134000 | 110000 | 115000 | 133000 |
| 10 | 131000 | 132000 | 142000 | 123000 | 125000 | 113000 | 123000 | 129000 | 116000 | 119000 |
| **AVG** | **129800** | **132300** | **139100** | **123900** | **121100** | **119700** | **128200** | **119800** | **126300** | **121600** |
| **σ** | **8146** | **5967** | **11937** | **9555** | **7778** | **7862** | **6838** | **5879** | **7772** | **8980** |

Table M-11. Range search times for 100% coverage.

| | QW for modified | QW for standard |
|---|---|---|
| 1 | 94000 | 168000 |
| 2 | 93000 | 156000 |
| 3 | 108000 | 145000 |
| 4 | 110000 | 173000 |
| 5 | 88000 | 173000 |
| 6 | 94000 | 161000 |
| 7 | 95000 | 172000 |
| 8 | 97000 | 174000 |
| 9 | 112000 | 177000 |
| 10 | 102000 | 161000 |
| **AVG** | **99300** | **166000** |
| **σ** | **7785** | **9560** |

# VITA

| | |
|---|---|
| Candidate's Full Name: | **Peter Anthony Judd** |
| Place and Date of Birth: | **Dee Why (Sydney), New South Wales, Australia**<br>**November 26 , 1957** |
| Permanent Address: | **36 Marigold Street**<br>**Fredericton, New Brunswick**<br>**E3B 7E1** |
| School Attended: | **Moss Vale High School**<br>**Moss Vale, NSW, Australia, 1971-1975**<br>**High School Certificate** |
| Technical College Attended: | **Wollongong College of Advanced Education**<br>**Wollongong, NSW, Australia, 1977-1980**<br>**Structural Engineering Certificate** |
| University Attended: | **University of New Brunswick**<br>**Fredericton, NB, Canada, 1984-1989**<br>**B.Sc. Surveying Engineering Degree** |