DATA STRUCTURES AND KNOWLEDGE REPRESENTATION FOR AUTOMATED NAME PLACEMENT

by

Lisa Mullin

TR93-076 April 1993

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

Faculty of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, N.B.
Canada E3B 5A3

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

DATA STRUCTURES AND KNOWLEDGE REPRESENTATION FOR AUTOMATED NAME PLACEMENT

by

Lisa Mullin

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE

Master of Science in Computer Science

in the Faculty

of

Computer Science

This thesis is accepted.

.....

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

March, 1993

ABSTRACT

This thesis contains the result of an investigation into the design and implementation of a computer-assisted system to deal with the particular problem of point name placement on EMR Canadian topographic maps. Software was written to correlate the names database, National Geographic Names Data Base, to the features database, National Topographic Data Base. This software was tested on the names and features of one 1:250,000 map sheet, NTS21G. This test showed that of 309 point names in the map sheet area, 169 (55%) were matched to identifiable features. Of these 169 matches, only 20 were found to be placed on the map sheet. Sixteen names of villages were found on the map, but no matching feature was found in the features database. A novel object based quadtree spatial data structure was designed, tested and implemented. A prototype name placement system called NAPLES was designed, and a portion of it was implemented. This prototype makes use of the quadtree structure to detect interference and overlap of names with features, grid lines, and other names.

ACKNOWLEDGMENTS

I would like to thank Dr. Brad Nickerson for suggesting the topic of name placement and supervising my thesis. Thanks also to Kirby Ward for the technical help he provided during my thesis work.

This work was supported in part by a grant from the EMR/NSERC Research Agreement Program, agreement number 288 5 92.

This thesis is dedicated to the memory of Barbara Wunch. She was a special friend to all of her peers while being both an employee and student at UNB. Barb was very interested in my graduate studies progress and her enthusiasm is greatly missed.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1.

INTRODUCTION

1.1 Background

The task of name placement on topographic maps is a delicate process which continues to be a time consuming effort for cartographers. The aesthetic aspect of the task plays an important role in determining the correct placement of each name. An example of name placement on Canadian map sheet NTS21G is shown below in Figure 1. Unfortunately, there are no algorithms which work consistently; experienced cartographers use heuristic rules of thumb with many exceptions in placing names on maps [Imhof, 1975]. These considerations have hampered cartographers from using computers to automate this process.

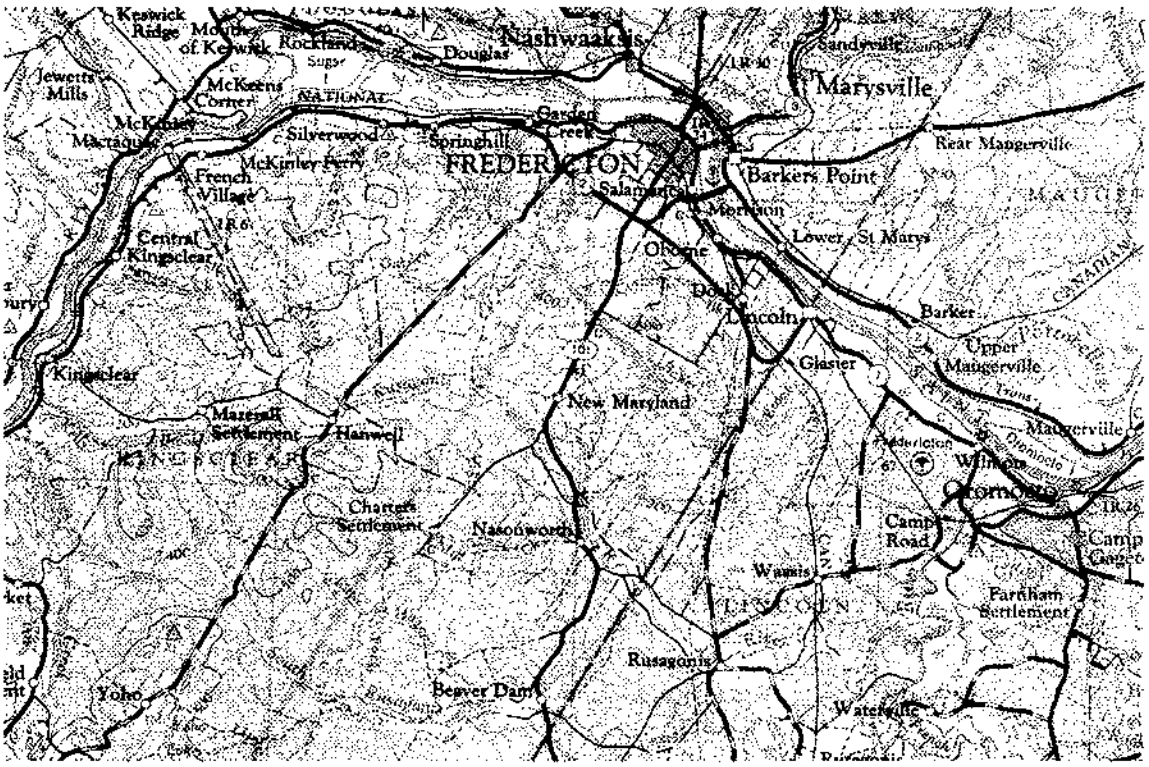


Figure 1. A sample of the EMR Canada NTS21G 1:250 000 map sheet.

This thesis explores the possibility of automated name placement for production of topographic maps. This automation will involve an expert system for implementing the cartographic rules and a spatial data structure for determining the name placement. This research was initiated with a literature search followed by the design work. The design was performed in phases, some of which was done concurrently because of the inter-dependencies of the various aspects of the problem.

1.2 Literature Review

The name placement problem has been studied for many years. Our research effort began with a literature search of automated name placement. This search produced many interesting papers, but only those which provided us with an approach useful for our research efforts are included here.

Yoeli discussed the ideas of automated name placement and the associated problems over twenty years ago [Yoeli, 1972]. He broke the problem down into three classifications; points and small areas, lines or bands, and areas. He also proposed a grid structure for name placement, which had ten possible placements and associated priorities for each.

Another significant contribution to the area of name placement research was from Imhof three years later [Imhof, 1975]. He was concerned with the rules and conventions for name placement and he gave some examples of good and poor name placements. One example of a rule Imhof proposed was "While type lines should not cling to their objects, they should not be too far from them, nor should the names cross their objects."[Imhof,

1975]. The illustrations which accompanied this rule for names near linear features is show in Figure 2.

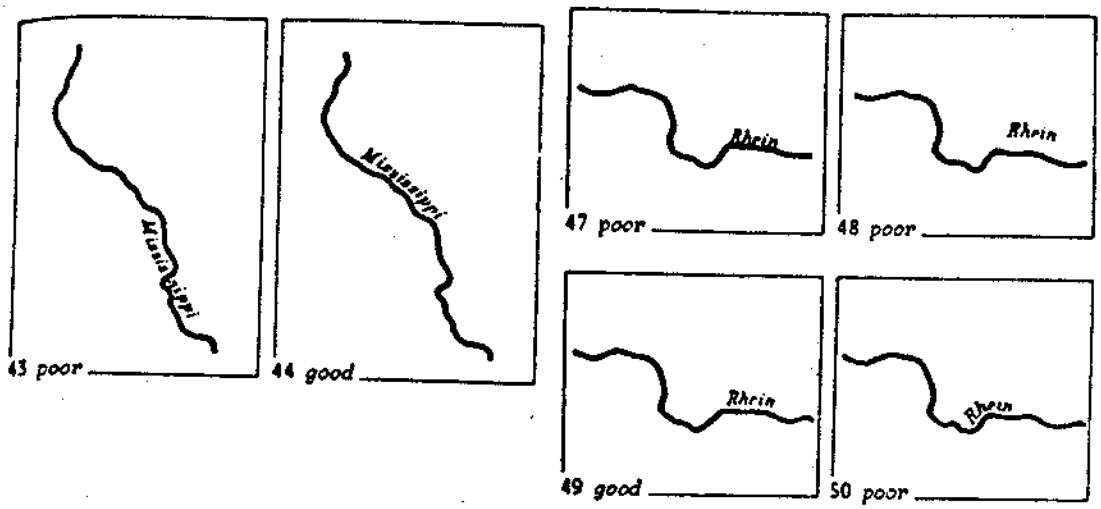


Figure 2. Examples of names placed "near" linear features [from Imhof, 1975].

Since then some attempts at automating the name placement process have been made [e.g. Hirsch, 1982; Ahn, 1984; Lacroix, 1984; Doerschler and Freeman, 1992; Johnson and Basoglu, 1989; Ebinger and Goulette, 1989; Jones, 1989]. Each of these efforts have been concentrated in one specific area of automated name placement.

Hirsch [1982] focused on the spatial search technique requirements of point name placement. He had studied the work of Imhof and Yoeli and attempted to place point names based on Yoeli's proposed point name positions. Hirsch used point-in-polygon methods for detecting three categories of overlap. The types of interference considered were names with names, names too close to another point, and names outside the map

border. The map features themselves were not considered by Hirsch; his work focused exclusively on the map names.

The problem of automated name placement is an interest of professor Herbert Freeman. He has supervised various theses in specific aspects of this area of research. Three students whose research is worth noting are Ahn [1984], Doerschler [1987], and Lacroix[1984].

Ahn's [1984] work resulted in the development of a Fortran program for automated name placement. Ahn's goal was to produce a non-interactive system which could place names based on general principles, general rules, and specific rules. Ahn also studied methods of placing area names within a feature. He considered using curve fitting along the area skeleton. An example of his contribution to area name placement is shown in Figure 3.

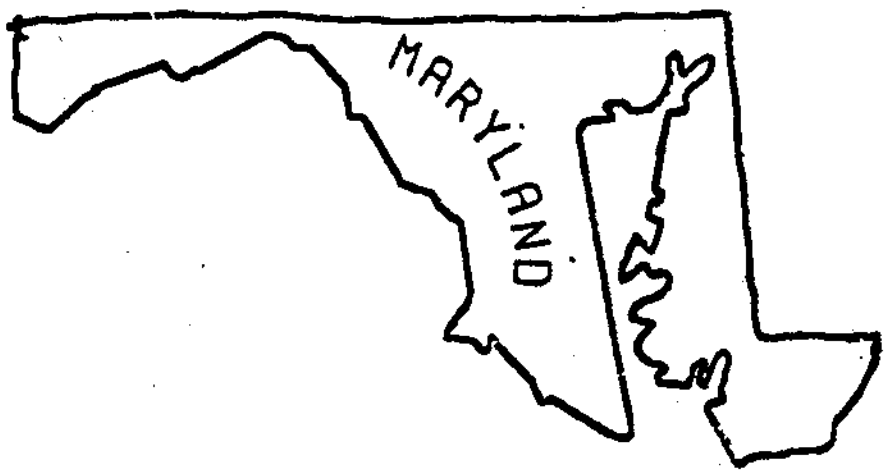


Figure 3. An example of area name placement [from Ahn, 1984].

Lacroix [1984] proposed an improved area name placement algorithm. This work attempts to improve previous efforts at area name placement by placing more emphasis on having names parallel to the baseline when possible. Lacroix subdivided the area rules into three categories of rules which include lettering, baseline, and centering rules. Lacroix also extended the research to include the placement of sea names.

Doerschler and Freeman [1992] gave consideration to the required spatial data structures for dense maps and produced a system for this purpose. The spatial data structures which they studied included a grid with the cell size set to one-fifth the smallest font size and k-d trees. Their results for dense street maps are impressive.

Ebinger and Goulette [1989] prototyped a non-interactive system for automated name placement. Their work was done for the United States Bureau of Census, and therefore, a recursive algorithm which required human intervention was not feasible because of the number of various maps required. In the system they developed, the strict rules for map labeling were relaxed. If necessary, the label sizes were altered, the map scale was changed, and names could be suppressed as required. They were looking for a practical solution which relaxed the aesthetic requirements of cartographic name placement.

Another person who worked with the problem of automated name placement was Jones [1989]. Jones's work is somewhat unique in that his development was done in Prolog. His algorithm for interference detection is raster based and works on the principle of overlap tolerance. If a group of names cannot be placed within the desired tolerance, then the tolerance is increased and backtracking is done.

Johnson and Basoglu [1989] used neural networks to study the problem of name placement. Their use of the neural network technology has, however, been restricted to the generation of the production rules to be used by the name placement system. Their tool is to be used by cartographers to generate the rules for labeling which they want implemented. Their concentration on the rule gathering process seems to be quite novel to this area of research.

Through the literature search it is evident that, although the main problem has been researched for quite some time, solving all of the component problems would be extremely difficult. It is also apparent that there is no definitive way of approaching the problem. Each person who has worked in the area has tried something different. It is also apparent that much of the research done has merit, but some have limitations which remain to be overcome.

1.3 Thesis Objectives

The objective of this thesis is to investigate the potential for using a knowledge base and data structures for automated name placement. This task was initially outlined in five objectives, as follows:

1 To discover methods for representing rules about name placement which do not fall in the IF... THEN... category of rules used in most expert systems.

2 To investigate the use of a rule-based expert system for automated name placement using the rule representation discovered by the first objective.

3 To discover a data structure suitable for the integration of the names and their geographical features.

4 To investigate ways in which names can be automatically matched with their associated geographical features.

5 To develop a prototype expert system for name placement using names and topographic data from the National Topographic Data Base for the Fredericton, New Brunswick area.

CHAPTER 2.

DESIGN CONSIDERATIONS

The entire name placement problem, consisting of area, linear and point name placement, was considered. A decision was made to focus this research effort on the placement of point feature names since considering all three types of name placement would be beyond the scope of a thesis. The design of a prototype was the goal, but there were many stages of design to be addressed. The design can be divided into the following phases: initial constraints, rule extraction, data correlation, name structure, spatial data structure, and prototype architecture. These aspects of the thesis were considered globally. The detailed design of each phase was then carried out with the global picture in mind.

2.1 Initial Constraints

When this thesis was started, the initial constraints were defined. Some of these parameters were inherent because of the association with Energy, Mines and Resources in Ottawa. The proposed research was to study EMR specifications in order to determine if automated name placement could assist cartographers working with Canadian topographic data. The initial constraints include the data to be used, the rules to be used, the quality of design to be done, and the computer platform to be used.

The data for this thesis consists of the names to be placed on a topographic map and the features which are on these topographic maps. This data is from two divisions of Energy, Mines and Resources, Canada. The names information is contained in the National Geographic Names Data Base (NGNDB) [EMR, 1989]. A portion of this database has been extracted for the 1:250,000 map sheet of Fredericton. This file of names for the Fredericton map 21G contains 3,108 names. This file is in alphabetical

order within region and each record of the file corresponds to one name. A sample from the names in New Brunswick showing five of the nine fields in the file is found in Figure 4. The fields not shown in the figure correspond to information such as the region code, gazetteers map and two other possible locations associated with the name [EMR, 1989]. The generic code is the classification used in the NGNDB which associates a name with a type of feature. For example, generic code 2300 indicates that the name belongs to an island. The latitude and longitude shown here are an approximate location of the name on the map.

Name	Generic Code	Unique Key	Latitude	Longitude
Barkers Island	2300	DBABS	455800	663600
Barkers Point	3	DBABT	455700	663700
Barlow Brook	601	DBABV	454600	663400
Barlows Bluff	1625	DBABW	452000	660500
Barnaby Head	1601	DBABZ	450700	663200
Barnes Island	2300	DBACF	450000	665400

Figure 4. Sample of the names file.

The EMR map features data is contained in the National Topographic Data Base (NTDB) [EMR, 1987]. A portion of this database can be extracted into a file for particular map sheets. Each line of this ASCII map features file may contain either a graphic characteristic command or graphic generation command in the first few bytes of the record. The graphic characteristic commands are descriptions which will be associated with all the features to follow until a new characteristic command is encountered. Examples of such commands are:

ASC/ graphic group number - the EMR code for the features

LAC/LC = color - the color of the features

The graphic generation commands are used to list the (x,y) co-ordinate pairs which make up the features. These points are specified in the UTM (Universal Transverse Mercator) coordinate system. Two of these command are:

$LST/OP,X_1,Y_1,...,X_n,Y_n$ - line string command

$CUR/X_1,Y_1,...,X_n,Y_n$ - curve command

When a graphic generation command is specified, the co-ordinate pairs are listed in order with a comma as a delimiter. When a series of co-ordinates require more than one record in the file the continuation line must have either blanks or the continuation command CON/ in the command area.

The file containing the features data for one 1:250 000 map sheet for the Fredericton area, map number 21G, is about 8 megabytes in size. An example listing of part of this file is shown in Figure 5. This example describes the centre-line coordinates of wooded island shorelines as indicated by the graphic group number 5790.

```
ASC/5790
LAC/LC=74
LST/OP,629829,4998263,629816,4998287,629766,4998331,629722,4998330,
    629722,4998305,629792,4998256,629829,4998263
LST/OP,629769,4998486,629662,4998529,629581,4998540,629537,4998565,
    629487,4998571,629468,4998539,629487,4998489,629525,4998452,
    629613,4998403,629695,4998385,629745,4998411,629769,4998486
```

Figure 5. Sample of the map features file.

Another initial constraint concerns the name placement rules of this thesis. The rules used are taken from the EMR Standards and Specifications [EMR, 1987]. This document contains the standards and specifications used in labeling topographic maps by the cartographers at the Canadian Centre for Mapping in Ottawa. Although the English

description of many rules is relatively simple, the conversion from English rules to expert system rules is not trivial.

The type of design proposed for the prototype was determined in the initial stages of this endeavor. The goal of the research is to develop a prototype based on a production quality design. This means that this design quality is an issue in all stages of the thesis.

When this thesis began, the question of computer platform was addressed. Since IBM and compatible microcomputers are prevalent in industry, the prototype to be developed was geared to this platform. The volume of data involved and the memory available on a microcomputer were analyzed and the decision was made to use a microcomputer platform. However, since some software was already available on the IBM mainframe, software related to the work already done would remain on the mainframe.

These constraints are firm and are a guide to the design and implementation of the name placement system. Working with this foundation gave the thesis direction from the start.

2.2 Rule Extraction

One of the underlying problems to be addressed is the representation of rules which do not fall in the IF... THEN... category of rules used in most expert systems. The rules in the EMR Canadian Standards and Specifications on name placement [EMR, 1988] were extracted and examined. We had to ensure that each rule could somehow be incorporated into the design. The rules were broken down into the classes specified in the standards document, with our emphasis being on point name placement.

The first set of rules in the EMR Canadian Standards and Specifications [EMR, 1988] are the general rules. These rules are "rules of thumb" or generalizations used by cartographers. Each of these rules were considered and the following was determined:

R1. "...the interior horizontally placed names are parallel to the base cord" [p.2/8]

This rule would indicate that the priority for name placement should always attempt to place the name horizontally and parallel to the base cord. On a topographic map, the base cord is the straight line from the lower left hand side to the lower right hand side of the neat line. The base cord is the straight with respect to the neat line and not the map sheet; therefore the names may have to be slightly slanted before they can be placed.

R2. "Labels and elevations should be read parallel to the map base." [p.2/8]

The map base is the bottom of the actual paper map sheet. Placing labels and elevations parallel to the map base will simply indicate that these labels are not curved at all.

R3. "Avoid placing a name where it will run parallel to, and overprint a grid line. Raise or lower the type to clear the grid line." [p.2/8]

This rule has two implications for the design of a prototype. The first concern is that collision of a name with a grid line must be detectable. This means that the spatial data structure adopted must maintain information indicating which pixels make up the grid lines. The ability to determine that a name is running parallel to a grid line must also be detectable. This has to be handled by writing a special search routine to determine if a name overlaps a grid line.

R4. "In **no case** will type be allowed to totally obscure a map feature." [p.2/8]

This rule is very straight-forward; if a name overlaps a map feature then it will have to be moved. This rule also indicates that the collision of a name and a map feature will have to be detectable and the information indicating where the map features are in the spatial data structures must be maintained.

R5. " ...the feature to which the type belongs should be easily recognized." [p.2/8]

Having the feature to which a type belongs being easily recognizable is not easy. It is a rule of thumb which cannot easily be obtained and cannot easily be measured. This rule requires consultation with a cartographer to explain more explicitly what this rule implies.

R6. "Names are most easily read when horizontal" [p.2/8]

This rule also indicates that the name placement priorities will always begin by trying to place a name horizontally, but they will be parallel to the base cord as indicated by R1.

R7. " In order of precedence, black names of cities and towns, park boundaries, Indian reserves, railroad identifications, various labels and border materials are affixed immediately after the grid." [p.2/8]

This rule can be handled in one of two ways. The first is to separate the different levels of precedence as a preprocessing step and then handle one level at a time in the placement process. The second approach is to allow the rule-based system to determine which names are in the specific levels of precedence and handle them accordingly.

R8. " ...care must be given not to separate specific and generic parts of names to the point where the eye does not readily relate them" [p.2/8]

This rule will have to be handled when the name is generated. This rule has not been considered in depth, but determining a measure of acceptable "separation" will be required. It would seem that the separation will depend on the font being used.

Black labeling on topographic maps is used for most names. All of the general rules for name placement apply to the black type. The black labels can be divided into the three classes; points, lines and areas. These classes have specific rules, but the only black rules considered here were the ones pertaining to point features.

Point rules are labeling rules associated with points on a map. These rules are divided into two sets; names of point features, and actual symbols. These two types are treated separately. Point features are placed in a similar manner to symbols because the name is placed with respect to a point on the map. However, these names have specific rules associated with their placement.

Two of the rules from the EMR Canadian Standards and Specifications on name placement of point features [EMR, 1988] are:

R9. " Names of point features such as towers, small islands, capes, heads, or bluffs shall be positioned as follows:
- to the right and 1 mm higher than the centre of the feature or
- to the left and 1 mm higher than the centre of the feature or
- to the right and 1 mm lower than the centre of the feature or
- to the left and 1 mm lower than the centre of the feature" [p.2/9]

This rule sets the order of precedence for point name placement. This rule indicates that the names of points should be placed in the above order. It also indicates that the distance of 1mm from a feature center must be calculated when placing a name.

R10. " Where a number of small islands and points are to be named, it is permissible to abbreviate Island to "I" and Point to "Pt", to avoid crowding, otherwise they will be spelled out in full" [p.2/9]

Determining that a region is crowded is not a simple task. The word crowded will have to be defined as a certain percentage of the neighboring pixels being occupied. The

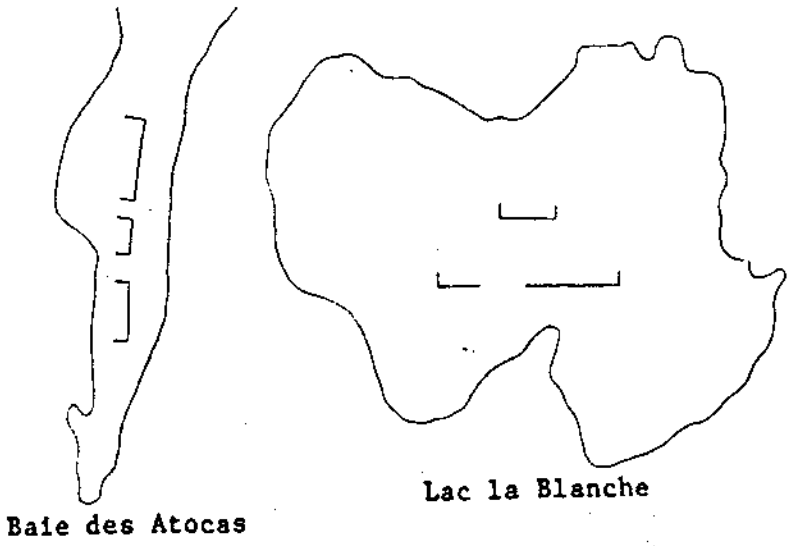
initial thought was to try to place an island or point name and then determine if the region is crowded.

Symbol rules correspond to the small symbols used on topographic maps to designate buildings or points. These symbols have their own set of rules found in the section CARTOGRAPHIC SYMBOLS in the EMR Canadian Standards and Specifications [EMR, 1988]. These rules are not considered in this thesis.

The current research is concentrated in the area of point feature name placement. Since many small areas are handled in the same manner as point features, area features of less than twenty five square millimeters are also being considered. Examples of these small areas with appropriate labeling are shown in Figure 6. From this figure it should be noted that only the first three examples have been considered since the fourth case is treated as an area name placement.

Figure 6. Examples of name positioning of small areas [from EMR, 1987].

Areas on the map have rules specifying where a name is to be placed based on the size and shape of the area. In order to demonstrate these rules and show how they differ from point and linear name placement the following examples are included in EMR Canadian Standards and Specifications [EMR, 1988].



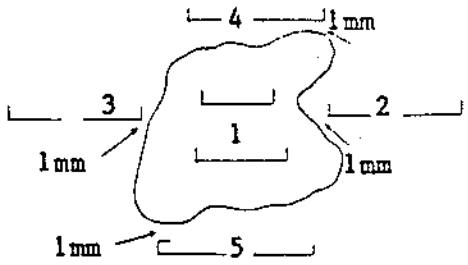


Figure 7. Examples of area name placement [from EMR, 1998].

Many black lines on a map are not labeled at all, but some are labeled with text. An example of a labeled line on a map is a boundary. These lines are labeled similar to streams, but they are generally handled separately.

Brown labeling on a topographic map corresponds to contours. The rules for contour labeling indicate that contour numbers should read uphill and index contours are to be given preference. The contours must also not interfere with other nearby lettering and not all numbers are shown; a pattern which shows the lowest, highest and some in-between elevations is used.

The blue water type rules are used to determine how bodies of water should be labeled. These bodies of water are divided into five classifications:

a. Large Bodies of Water, Oceans, Gulfs, Bays

b. Medium and Large Lakes (5 cm and larger)

c. Small Lakes (up to 5 cm)

d. Double Line Rivers, Streams

e. Single Line Streams

The rules for the blue type names are specific to one of the above classes. The size of the area or the characteristics of the body of water on the map dictate the name placement rules used. The only class of bodies of water that were considered in this thesis were ponds since they are labeled similarly to point features.

The rule extraction phase of the design was critical. Each rule was considered in the design to insure that it could be accommodated in the prototype. The rule extraction was done as one on the initial stages of the research and every subsequent stage of the design took the rules extracted into consideration.

2.3 Data Correlation

Matching the names in the National Geographic Names Data Base to their associated features in the National Topographic Data Base is an important step of the preprocessing required for automated name placement since the two databases do not have a close correspondence. The only way to correlate the names and data is by exhaustive search. The location of the name in the NGNDB is only an approximation and does not match the NTDB exactly. Unfortunately, the generic codes used to determine what type of feature the name is associated with do not correspond to the graphic group numbers representing the feature types since these two databases are from different divisions of EMR.

The first step to matching the names to features was to extract the necessary data. Two programs had already been written in PLI on the IBM mainframe. The first program extracts all of the names in a specified region and did not require any changes. The second program was designed to extract up to ten different types of features in a specified region. This program was changed so that all of the features in the region could be extracted. The details for using these programs are found in Paine and Mullin [1992].

The next step in matching the names from the EMR names database to EMR features was to generate a look-up table of generic codes for names which correspond to graphic group numbers of features. Since this data is intended to be used for all of Canada, many of the name codes are duplicated; one for the English name code and one for the French version. The codes for the features are also more specific than those of the names. These two considerations resulted in a table which is not a one-to-one mapping from the names to features. A portion of the names-to-features look-up table is shown in Table 1. From the look-up table we can see that the generic code 113 matches both

graphic group numbers 4120 and 4450. The completed table has 46 point feature name generic codes with their corresponding graphic group numbers (see Appendix A).

Table 1. Sample of the names-to-features look-up table.

Generic Code	Language	Meaning	Graphic Group Number	Meaning
113	E	Post Office	4120	Post Office to Scale
			4450	Post Office
119	E	Trading Post	4132	Trading Post to Scale
			4133	Trading Post
120	E	Fort	4310	Fort to Scale
			4311	Fort Symbol
125	F	Bureau de poste	4120	Post Office to Scale
			4450	Post Office

To aid the matching of EMR names to EMR features, a search algorithm has been designed and implemented. These results can be divided into three categories; single match, multiple matches, and no matches.

This algorithm first places all of the extracted features into a grid overlay structure. This overlay structure divides the map sheet into many smaller sections. The number of sections in the grid is a parameter for the user to determine. The data structure for the overlay consists of a two-dimensional array of linked lists. The nodes of the linked lists contain data for each feature which is in the grid cell. When the name search is carried out, only the necessary cells are searched. While this is being done, the offset of the position of the feature from the start of the file for each feature is recorded so that the feature can easily be accessed later if matched to a name. After all of the features for one map sheet are placed in the grid structure, the names are read sequentially. The name generic code is then searched for in the look-up table and the matching graphic group numbers are obtained. If no matching features are in the look-up table, then the feature is

not considered to be a point. If there are matching graphic group numbers for a generic code, then the grid cell in which the name is located is searched for matching features. The name and all of its matching features, i.e. features of appropriate graphic group numbers in the vicinity of the approximate name position, are written to a file. A diagram of this grid overlay data structure is depicted in Figure 8.

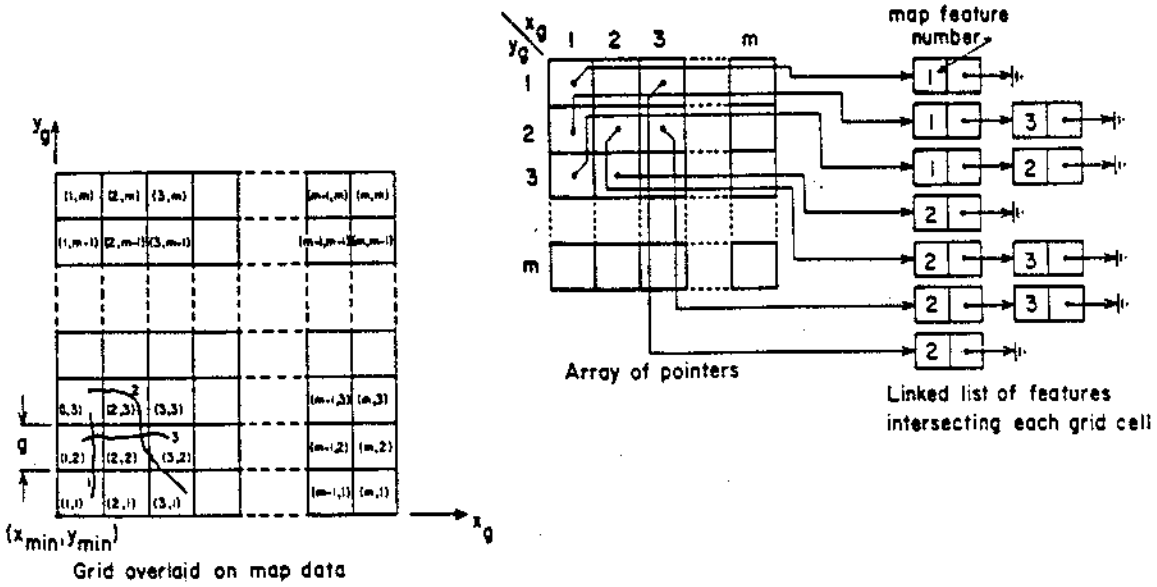


Figure 8. Grid Data Structure [from Nickerson, 1987].

When the name to feature correlation is done there are several cases to be consider. There may be one feature which matches a name, several possibilities for a name or no features which match a name. Each case was considered in the design of the correlation process.

The first case considered was one matching feature found in the appropriate grid cell of the name being processed. It was assumed that this feature is a match and the match is recorded in a file.

Finding no matching feature for a particular name can be divided into several cases based on the type of name in question. For the purposes of this thesis, only point names with no matching feature have been examined. Some names will not appear on small scale maps because the NGNDB contains all names with no consideration of their use on maps. For example, a small town which appears as an area on one map scale may appear as a point on another scale if present at all. However, if no matching feature is found for a point name in the initial grid cell , three additional grid cells are searched. These three grid cells are the three closest to the position of the name. A diagram showing an example of the three additional cells searched for when a match is not found in the initial cell id shown in Figure 9. The name position in the figure is indicated by the X. If these additional searches do not yield matching features the name is eliminated from further processing.

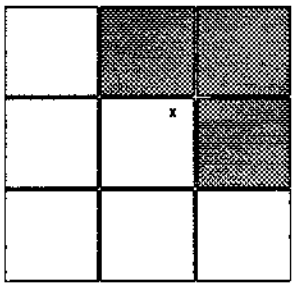


Figure 9. Example of extra cells searched when there is no match.

If multiple features are found as being possible matches for a name, the user is required to determine which feature is to be selected as a match. To help the user with the selection, the list of all matching features is sorted in increasing distance from the centroid

of the feature to the placement associated with the name. This list of features also includes the area of the feature on the map sheet in millimeters squared.

Once this stage of the correlation work was designed and implemented, it was apparent that more tools could be beneficial to aid the user in matching the names to features interactively. Two additional utilities were built which dealt with plotting the name and feature data. The first addition was the ability to generate HPGL (Hewlett Packard Graphic Language) which could be plotted. The other tool generated was software which would allow the user to browse a map sheet on the CRT and display the labels. This display tool does not have the names positioned as on a map but just displayed horizontally at the approximate position in the names database. The ability to zoom in and out on data for a map sheet seemed to be a tool which would be used extensively.

Since the research in this thesis has been limited to point features and small area features which can be handled as points, little consideration has been given to area or line names which do not have matching features. Most of the area and linear names are currently not matched because the look-up table has only been completed for points. The searching algorithm implemented will generally be feasible for areas and lines once the look-up table is extended.

Design and implementation of a data correlation tool consumed a large amount of time. The task of design was substantial because it assumed that a totally automated algorithm is not possible. An architecture for the complete matching process is shown in Figure 10.

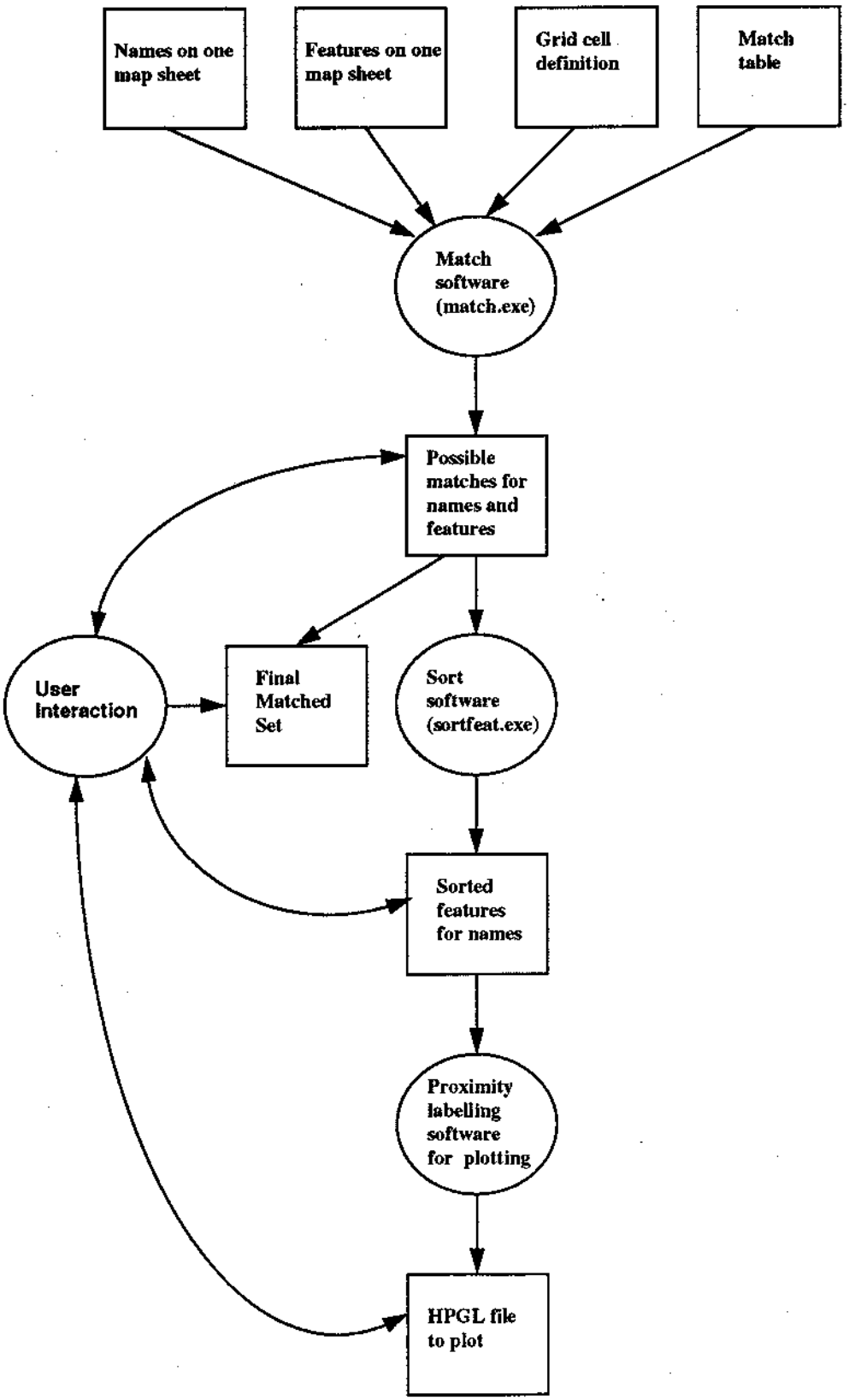


Figure 10. An architecture for data correlation.

As can be seen from the architecture for data correlation, the process is quite involved and requires substantial interaction by the user. The steps to follow in the complete matching process is further discussed by means of an example in chapter 3.

2.4 The Knowledge Base

The approach used for the design of the knowledge base is the use of a priority scheme for name placement which interacts with a quadtree-based spatial data structure. The three components of the knowledge base are the priority scheme, the names frame structure and the rules from section 2.2.

The priority of the next relative position to try for each point name is held in a structure in the knowledge base. There are eight possible positions which may be used; the first four are horizontal while the lower priority positions are curved. The sequence of priorities looks like:

(HRU HLU HRD HLD CRU CLU CRD CLD)

where:

H = Horizontal C = Curved

R = to the Right L = to the Left

U = Up D = Down

This structure indicates that the most favored position for a point name is horizontal, to the upper right of the feature being labeled. It should be noted that other placements are optimal for area and line features. If no horizontal placement is possible, one of the four curved placements can be used. A diagram of the possible curved positions is shown in Figure 11.

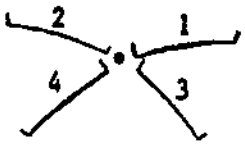


Figure 11. Possible curved point name placements [from EMR, 1988].

A frame structure for the names is included in the knowledge base design. The name, generic code and location slots of the frame structure will be taken from the input passed to the expert system while the remaining slots are used during the placement process. The placed-at slot contains the quadtree coordinates of the current placement. The other necessary information for the name are its font-type, font-size and orientation. During the placement process, some of the slots correspond to boolean flags will be determined. The "crowded" flag indicates if the region around a name is crowded with a value of either true (t) or false (f). The other slots determined during name placement are the current-placement-position code and three overlap detection flags. The final slot of the frame structure contains a linear quadtree representation for the name. In this representation, G corresponds to gray, B to black and W to white. This means of recording the position of each name is essential in the placement process since the ability to remove names and try other placements is required. The frame structure for each name being placed looks like the following:

```
(name-placement
   (name  Nashwaaksis)
   (generic-code  200)
   (location  455900 663900)
   (placed-at  34567 25001)
   (font-type  Century-Text-Roman)
   (font-size  8)
   (orientation  2.6)
   (crowded  f)
   (current-placement-position  HRU)
   (overlaps-grid-line  f)
   (overlaps-map-feature  f)
   (overlaps-existing-name  f)
   (qt-rep  GGGBWGB...B))
```

When a name is positioned, a call from the expert system will invoke a C function which determines the placement of the name based on the present configuration of the name in the frame structure. The characters of each name will be placed in the quadtree data structure which is used to detect overlap. Calls to the C functions are also required to return any new information to the frame structure, in particular overlapping and crowded conditions. The system rules can then be used to make decisions about whether or not the current name placement is acceptable, if the next position should be tried, or if some alternative action is required. The procedures will be data driven such that the names being placed cause rules to fire automatically. A forward chaining expert system shell called ART-IM [Inference, 1991] was considered for top level construction of the rules.

2.5 The Spatial Data Structure

Another major aspect of this thesis was determining an appropriate spatial data structure for the name placement process. This structure must be capable of holding the raster representation of the features, grid lines, and names while maintaining a list of objects which intersect each pixel. This structure is necessary so that detection of

collisions can be done to find appropriate name placements which do not have names overlapping other names, features or grid lines.

A quadtree is a spatial data structure that recursively divides the space into four quadrants. These subspaces are generally labeled northwest, northeast, southwest, and southeast. Each node of the structure contains pointers to the four children, an ancestor pointer and the color of the node. The three possibilities for colors of the nodes are white for unoccupied leaf nodes, black for occupied leaf nodes and gray for interior nodes. This structure seemed to be appropriate except for one problem. In a typical quadtree, the concept of origin of an occupied pixel is not a consideration. However, in order to determine which objects a name has collided with, the object information of the existing black pixels must be maintained. This information is also required when the a name is to be moved.

The quadtree implemented in this thesis maintains the object information for each black node. Pointers to the objects to which the occupied leaf nodes belong are maintained in the structure. This information is kept in small linked lists that are added to the occupied leaf nodes. These linked lists contain two pieces of information. The first component of the linked list node is one character describing the type of object in this node. The three possibilities are "f" for a feature, "g" for a grid line, and "n" for a name. The other component of the linked list node is the graphic group number for features, a code for grid lines indicating horizontal or vertical orientation, and a unique key for each name.

The process of placing a name requires interference detection of the name being placed with features, grid lines and other names which have already been placed. With the

object information being maintained in the quadtree structure, overlaps can easily be detected and the interfering objects can be identified. This is necessary since different rules apply depending on what object the name is in conflict with.

Before name placement can begin, the raster representation of the features must be generated, since name placement rules are based on having the name placed with minimal interference with the map features. This part of the problem has been solved with a series of steps. These steps are to (1) expand the points of a feature, (2) build run length code for features, (3) build quadtrees for the features and (4) form a union of the quadtrees into one large feature quadtree.

The first step of the process requires turning the center line coordinates of the features into connected strips of specific widths. The widths for feature types are found in a look-up table. Since the width specifications found in EMR [1988] are in millimeters, the same units appear in the look-up table file. A sample of this width table is shown in Table 2.

Table 2. Sample of feature width look-up table.

Code	Description	Width in mm
2020	Road A (Paved Divided)	0.89
2030	Road B (Paved Undivided - not elevated - 2 lanes)	0.5
2040	Road C (Paved Undivided - not elevated - 1 lane)	0.25
2050	Road D (Gravel Undivided - 2 lanes)	0.55
2060	Road E (Gravel Undivided - 1 lane)	0.4
2070	Road F (Unimproved)	0.25
2080	Road G (Track - Cart/Tractor)	0.17
2090	Road H (Trail)	0.15
5010	Shoreline Lake	0.12
5780	Shoreline Island	0.12
5790	Shoreline	0.12

The general method of generating lines of specific widths is by generating circles of this diameter in succession. Since this is not a feasible alternative for our processing, the decision was made to form the features by a series of rectangles with interconnecting triangles as shown in Figure 12. The three vertices of the triangle have been circled here to indicate the interconnection.

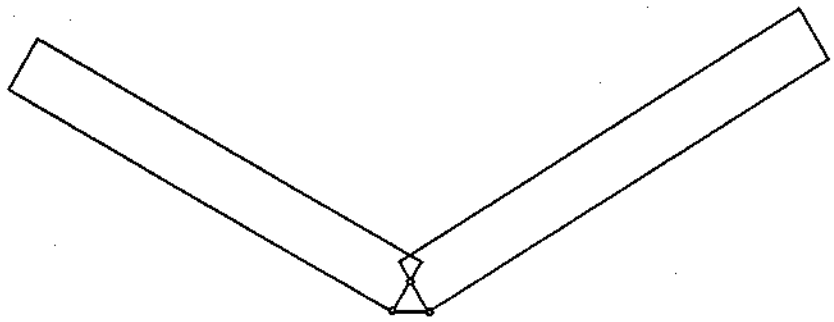


Figure 12. Method of interconnection to form a feature.

Figure 13 represents the centerline coordinates of a shoreline. Figure 14 illustrates the expanded version of this shoreline feature into connected rectangles, BOX, and

triangles, TRI. The BOX consists of the eight coordinates making up the four corners of the rectangle and TRI lists the three coordinate pairs for the small triangles. The diagram in Figure 15 shows how these coordinates would be expanded.

```
ASC/5790
LAC/LC=74
LST/OP,629829,4998263,629816,4998287,629766,4998331,629722,4998330,
    629722,4998305,629792,4998256,629829,4998263
```

Figure 13. A sample shoreline feature.

```
ASC/ 5790
BOX 629827 4998262 629814 4998286 629817 4998287 629830 4998263
BOX 629815 4998285 629765 4998329 629766 4998332 629816 4998288
TRI 629816 4998287 629816 4998288 629817 4998287
BOX 629766 4998329 629722 4998328 629721 4998331 629765 4998332
TRI 629766 4998331 629765 4998332 629766 4998332
BOX 629723 4998307 629793 4998258 629790 4998253 629720 4998302
TRI 629722 4998305 629720 4998302 629721 4998331
BOX 629791 4998258 629828 4998265 629829 4998260 629792 4998253
TRI 629792 4998256 629792 4998253 629790 4998253
TRI 629829 4998263 629830 4998263 629829 4998260
```

Figure 14. The expanded coordinates for the shoreline feature.

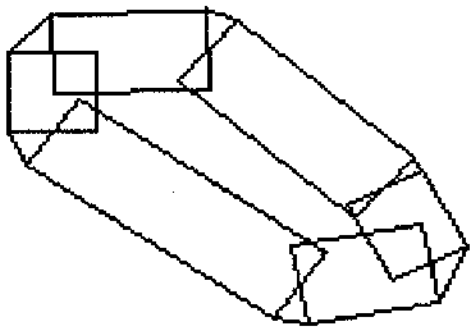


Figure 15. Plot of the expanded shoreline feature.

At this point the UTM coordinates are converted to quadtree coordinates. The conversion could have been done sooner in the processing, but this is a more appropriate point since the coordinate expansion in UTM coordinates is more accurate than expansion of converted points. The equation used for this transformation is :

$$\begin{bmatrix} x \\ y \end{bmatrix}_{qt} = \frac{c\left(\begin{bmatrix} x \\ y \end{bmatrix}_{UTM} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}_{UTM}\right)}{d}$$

where: (x_0,y_0) = the lower left hand corner of the map sheet,

c = number of pixels per meter for the quadtree (chosen as 20,000 here),

d = map scale number (e.g. 250,000).

For each rectangle and triangle of the feature a run length code is generated. Run length code is a representation for raster images which groups similar pixels into one-dimensional blocks. These blocks are usually ordered vertically, with the three elements being the vertical value, the initial colored pixel and the ending colored pixel. These smaller pieces of run length code are combined into run length code for the entire object. This step eliminates any overlaps in the generated rectangles as well as producing a more efficient representation to be used in the generation of a quadtree for the object. Figure 16 shows all of the run length code generated for the shoreline feature presented earlier in Figure 13. As you can see, there is some overlap in the run length code; for example, `RUN 1205 4079 4080` is duplicated. The run length code is sorted and condensed in Figure 17 by extending runs and eliminating duplicates. Here you can see that the runs: `RUN 1203 4080 4081`, `RUN 1203 4076 4078`, `RUN 1203 4077 4080`, and `RUN 1203 4080 4081` are combined to form: `RLC 1203 4076 4081`.

```
ASC/ 5790
RUN 1203 4080 4081
RUN 1204 4080 4080
RUN 1205 4079 4080
RUN 1205 4079 4079
RUN 1206 4077 4078
RUN 1207 4076 4077
RUN 1208 4075 4076
RUN 1209 4075 4075
RUN 1205 4079 4080
RUN 1208 4072 4075
RUN 1209 4074 4075
RUN 1208 4075 4075
RUN 1209 4075 4075
RUN 1202 4077 4077
RUN 1203 4076 4078
RUN 1204 4074 4077
RUN 1205 4073 4075
RUN 1206 4072 4074
RUN 1207 4072 4072
RUN 1206 4072 4072
RUN 1207 4072 4072
RUN 1208 4072 4072
RUN 1202 4078 4078
RUN 1203 4077 4080
RUN 1202 4077 4078
RUN 1203 4080 4081
```

Figure 16. The pieces of run length code for the sample shoreline feature.

```
ASC/ 5790
RLC 1202 4077 4078
RLC 1203 4076 4081
RLC 1204 4074 4077
RLC 1204 4080 4080
RLC 1205 4073 4075
RLC 1205 4079 4080
RLC 1206 4072 4074
RLC 1206 4077 4078
RLC 1207 4072 4072
RLC 1207 4076 4077
RLC 1208 4072 4076
RLC 1209 4074 4075
```

Figure 17. The combined run length code for the sample shoreline feature.

For each of these features a separate quadtree (S) is built. Each of these quadtrees will contain all of the traditional quadtree nodes as well as the linked list nodes. The linked list nodes are added to each black leaf node of the features so that the object information is maintained. A universal quadtree (U) will also be maintained as the union of the individual quadtrees for the features.

The algorithm for constructing a quadtree is a recursive algorithm which builds the tree from the bottom up [Samet, 1990]. All of the pixels in the space are given a node in the tree and the lower levels are pruned if the four children are of the same color. For a general bit map with random coloring this seems to be a good algorithm. For map features and names it is extremely inefficient. Most of the features on a topographic map cover only a small portion of the map sheet and therefore only a small portion of the quadtree.

The standard algorithm for constructing the quadtree of a feature has been modified to make use of the bounding rectangle of the features. At this stage, each feature is represented in run length code. Since this code is sorted, getting the bounding rectangle is done by taking the first and last values for y. The left and right limits are found by looking for either the lowest beginning x or the largest ending x in the feature. The quadtree with objects and the steps leading to it can handle area, linear, and point features since all of the features are required for any type of name placement.

2.6 Prototype Architecture

The design of a prototype for semi-automated name placement on EMR topographic maps was achieved. This prototype consists of all the steps required for name

placement of point features. An architecture, shown in Figure 18, for the problem of automated name placement has been developed.

The first aspect of the name placement architecture is data correlation. This phase begins by the extraction of the required names and features which appear on the map sheet. These are saved as two separate files. The names are then matched to features in an attempt to find the one feature corresponding to each name. At this point user interaction is required to determine which of the multiple matches are correct.

Once the data is correlated, the features can be placed in the spatial data structure. The grid lines are added to the spatial data structure at this time as well. At this point, the names can be placed. This is done by reading each name into the knowledge base and then placing them into the data structure. When all of the name placements are complete, the results are written to a file.

The following sequence of steps are from the architecture of the NAPLES prototype, and result in placement of the names:

1. Extract the features
2. Extract the names
3. Match the names to features
4. Insert the features into the quadtree
5. Insert the grid lines into the quadtree
6. Read the names into the knowledge base
7. Place the names
8. Save the resulting placed names as a disk file

A more detailed algorithmic summary of the specific steps proposed for automated point name placement are as follows:

1 Generate the run length code for the features

2 Generate the run length code for the grid lines

3 Enter the features into quadtree U

4 Enter the grid lines into quadtree U

5 For each name do

5.1 Determine the most appropriate position for the name

5.2 Generate run length code for this name at this position

5.3 Determine and remember adequacy of this placement

5.4 While placement is not adequate do

5.4.1 Choose next best position for the name

5.4.2 If no more available positions, then place at the best placement position encountered, and exit while loop

End While

End For

6 Save the complete name structure as a disk file

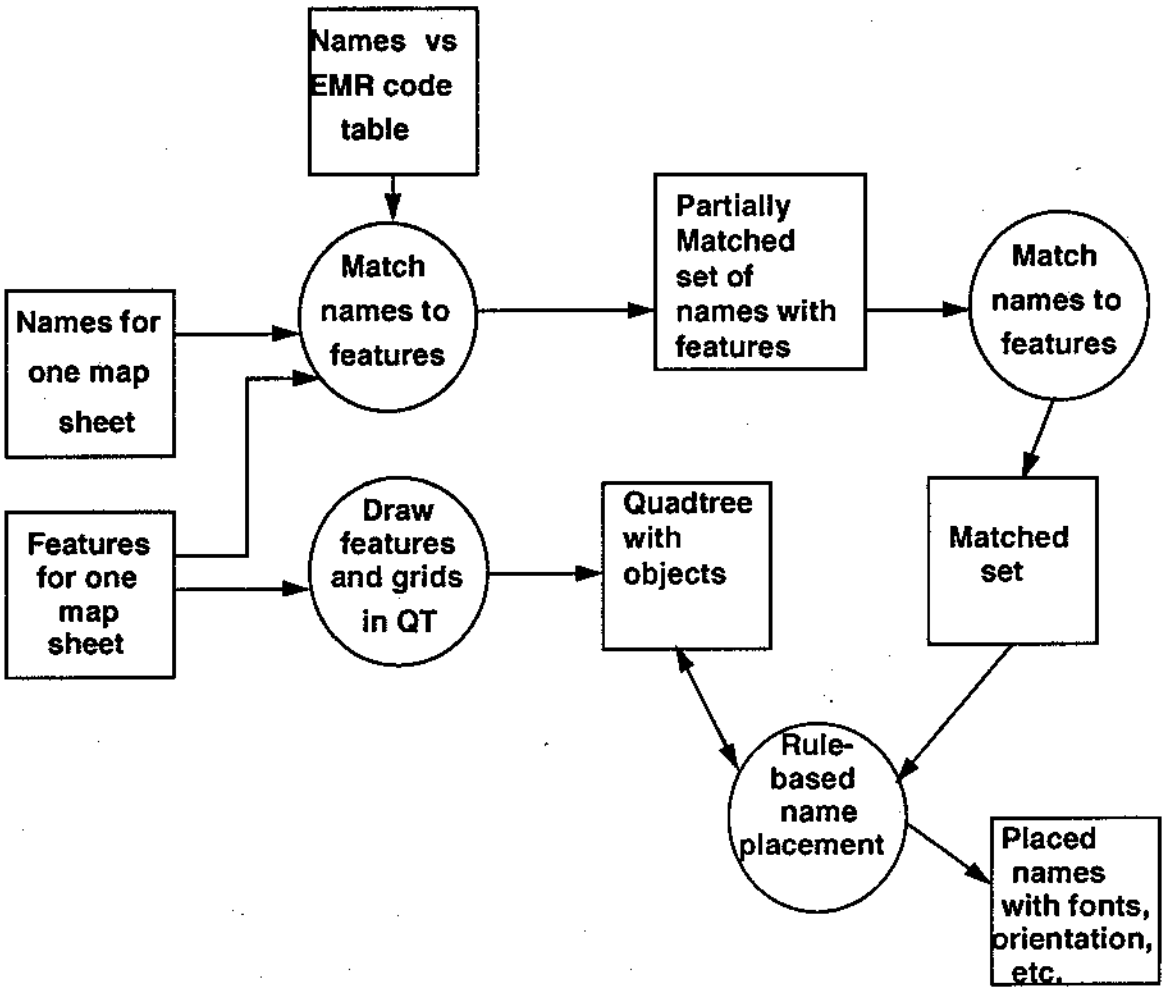


Figure 18. The system architecture for NAme PLacement by Expert System.

2.7 Font Issues

In order to do precise name placement, the font requirements are quite extensive. A package that can display a name in a specific font with a specific size is not necessarily suitable for use in an automated name placement system. In order to generate the raster run length code for a name, the actual font definitions must be accessible so that a bit map containing the name can be generated. Many systems for name placement [e.g. Jones, 89] use bounding boxes rather than a raster representation for the names interference detection.

A few of the 28 point feature fonts required for name placement on Canadian topographic maps are shown in Figure 19.

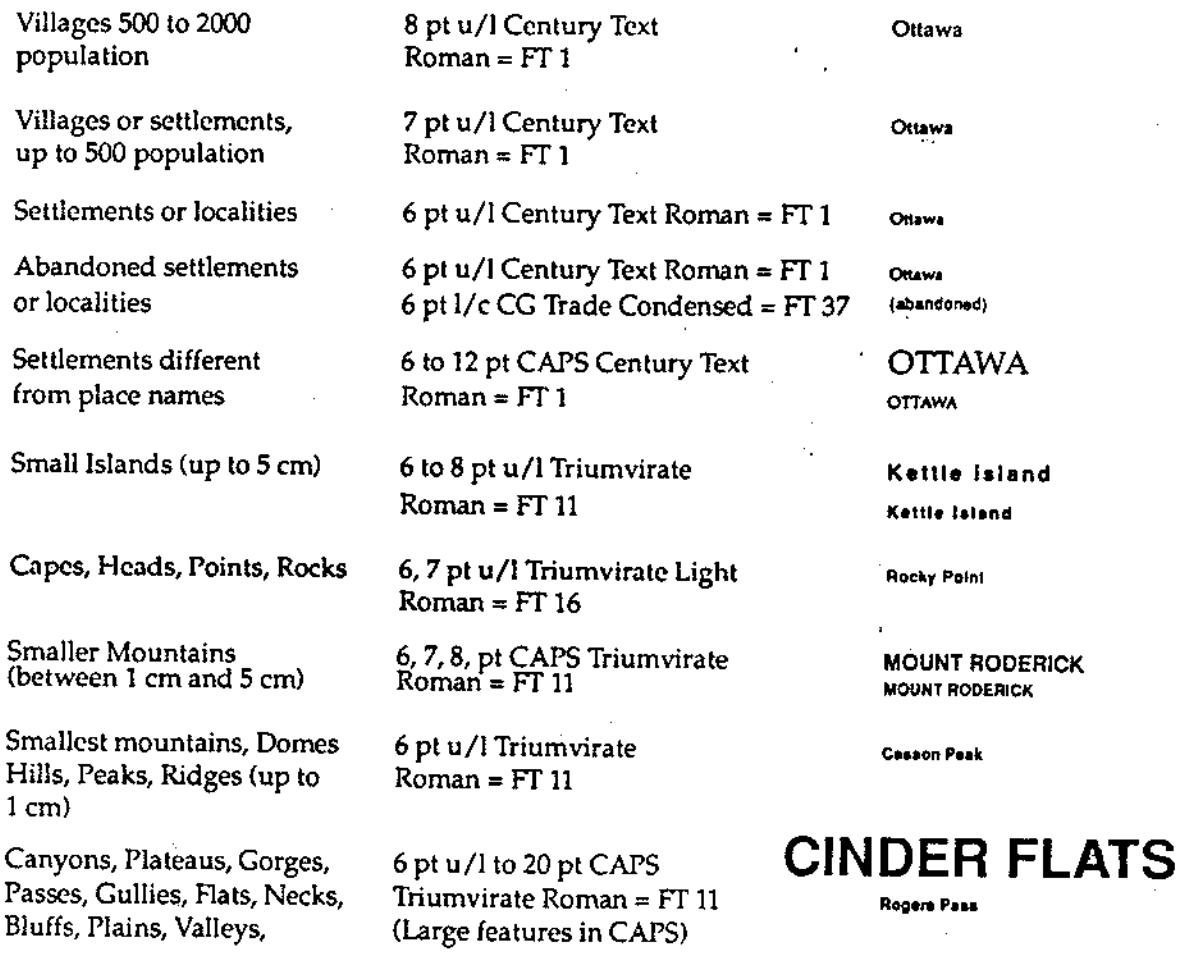


Figure 19. A sample of the required fonts for names [from EMR, 1988].

For NAPLES, software was written [Arendtsz, 1992] to return a bounding box for the string representing the place name. This software uses Windows 3.1 fonts which closely match one of the above required fonts. The look-up table used for the fonts is given below in Table 3.

Table 3. Sample font look-up table.

Index	Facename	Pitch	Family
0	Tms Rmn	VAR_PITCH	FF_ROMAN
1	Helv	VAR_PITCH	FF_SWISS
2	Courier	FIX_PITCH	FF_MODERN

The representation of names using their required fonts will require more work before names can be incorporated into the full implementation of the prototype. Without the raster representation of the names with the appropriate fonts, overlap detection is limited to determining the amount of overlap of the name's bounding box with other objects.

The design aspects of this thesis were substantial. The components which were given the most detailed design were the data correlation and the data structure. It would seem that the two components which require further detailed design before implementation can be completed are: the conversion of the rules to a computer representation, and the generation of names in the raster form of the appropriate fonts.

CHAPTER 3.

IMPLEMENTATION

The implementation for this thesis was divided into three main portions. The first part dealt with data correlation and the associated test with one map sheet. The second part of the implementation was constructing the knowledge base. The knowledge base was started but was not pursued since it was decided that this thesis would concentrate on the spatial data structures for name placement. This was a decision based on timing and priority of implementation was given to the data structure. The considerations for the spatial data structure included memory management as well as new algorithms for this particular implementation.

3.1 Data Correlation Example

The implementation of the data correlation phase is best described by considering a example of the entire process of section 2.3. Data correlation is required to match names with their appropriate features.

3.1.1 Data Extraction

The matching process begins in TSO on the IBM mainframe. There are two PLI programs; one to get the required names and one for the features. The name extraction program requires only the window coordinates of the area to be extracted in degrees. In this example, these programs were used to extract the NTS21G map sheet information, and the names parameters were 680000,460000,660000,450000, meaning to extract names within the bounding box of (46°N, 68°W) to (45°N, 66°W). The feature extraction program requires more parameters. The parameters for the NTS21G test are shown in Figure 20. The first line indicates how many different graphic group numbers are to be

extracted. When all features are to be extracted a special case is used, so this number is 1. The second part of the parameter lists the graphic group numbers to be extracted. If the graphic group number specified is a -1, then all features are extracted. The third part of the parameters is the window for extraction specified in UTM coordinates, the coordinate system of the features database. The final parameter is a flag. If this flag is a 0 then the features extracted must completely fit within the window bounds in order to be extracted; a 1 indicates that features partially within the window should also be extracted. The output for this example includes two files containing over 3000 names and over 8 megabytes of features. Once files were generated, they were transferred to an IBM PS/2 workstation for direct use.

```
1
-1
577434,5094316,736453,4987112
1
```

Figure 20. The parameters for extracting NTS21G features.

The NTDB data is lacking bounding boxes, and there is a need for extracting subsets for experimental work. Another PLI program was written in order to add a bounding box to each feature extracted. The bounding box lines in the file have the following format:

BOX/ smallest x, smallest y, largest x, largest y

The previous sample of the file shown in section 2.1 with the bounding boxes added might look like:

```
ASC/5790
LAC/LC=74
BOX/ 629722,4998256,629829,4988331
LST/OP,629829,4998263,629816,4998287,629766,4998331,629722,4998330,
    629722,4998305,629792,4998256,629829,4998263
BOX/ 629468,4498385,629769,4998571
LST/OP,629769,4998486,629662,4998529,629581,4998540,629537,4998565,
    629487,4998571,629468,4998539,629487,4998489,629525,4998452,
    629613,4998403,629695,4998385,629745,4998411,629769,4998486
```

Figure 21. Sample of the map features file after adding the bounding boxes.

3.1.2 Data Matching

Matching software has been implemented to aid in the matching of names to features and is documented in Mullin [1992]. This software consists of a series of eight Microsoft C programs. For the purposes of illustrating the algorithm of matching a name to its associated feature, I will use the example of the NTS21G map sheet data and the name "Bliss Island".

The names of two files are expected to have the same file name with the extension **NMS** for the names and **DAT** for the features data. The match program also requires a parameter file as shown in Figure 22. This file has the same file name with the extension **PAR**.

```
MATCH.TBL
25
45:00:00 -68:00:00 46:00:00 -66:00:00
```

Figure 22. Parameter file for NTS21G.PAR matching.

The first line contains the name of the look-up table for matching generic codes to graphic group numbers as described in section 2.3. The next line is the number m used in the matching algorithm; m indicates that an m by m grid structure is to be used as shown in Figure 8. The largest possible grid structure is 50 by 50. The last line in the parameter file contains the latitude and longitude ranges of the map sheet.

When the matching process is carried out, the files shown in Figure 23 will be created. The resulting matched file from the first process will have the extension **MAT.** The other file which is produced by the matching process has the extension **NAM.** It contains a truncated version of the names in the EMR names input file (NTS21G.NMS), which has a record length of less than 80 characters. This allows the file to be printed easily and is not actually necessary for the data correlation.

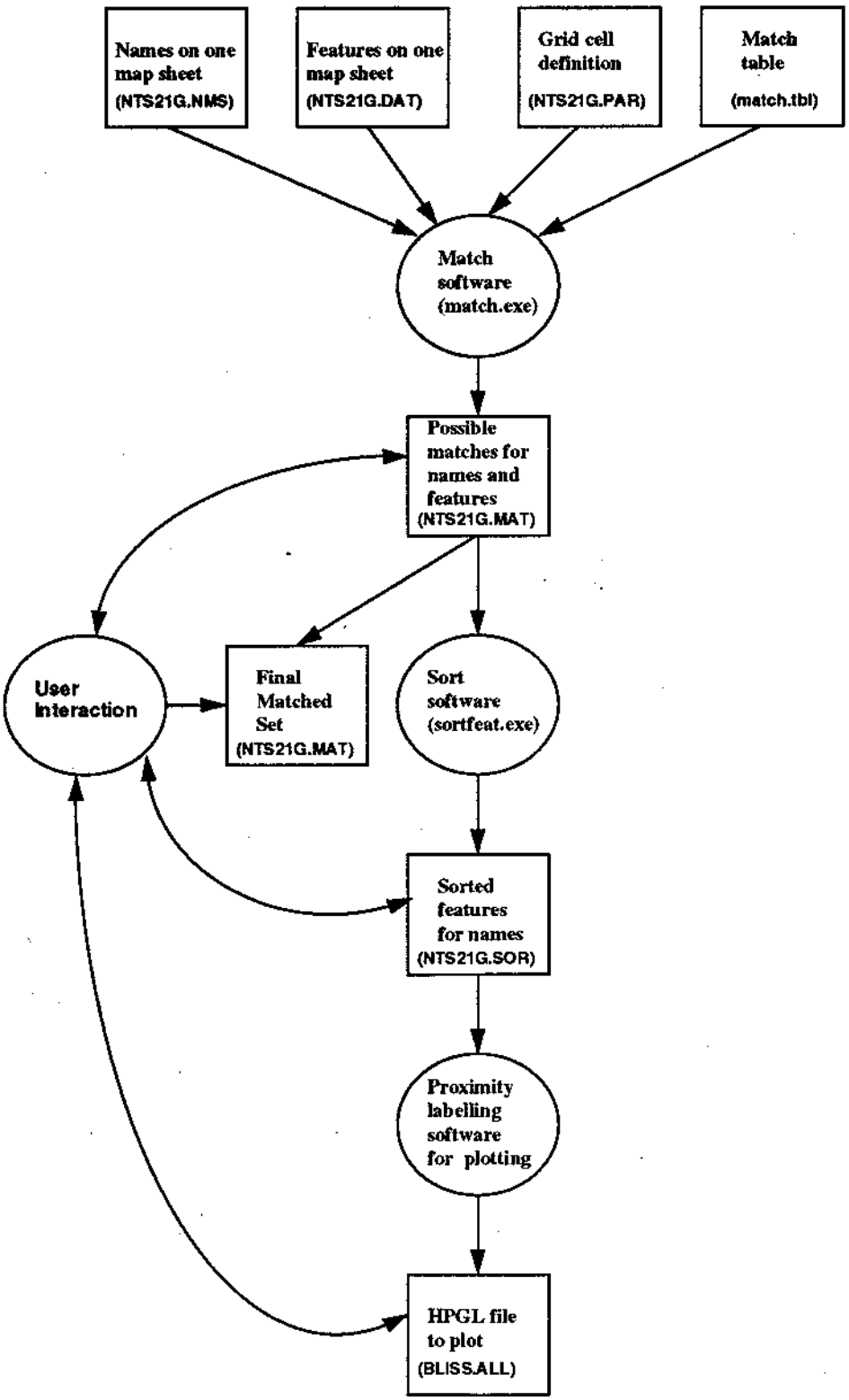


Figure 23. The matching example for map sheet NTS21G.

If there are multiple matches, as with the name "Bliss Island", then the resulting match file may not be sufficient. The portion of this file which is associated with Bliss Island is found in Appendix B. To help the user with multiple matches the **sortfeat.exe** program can be used. A sample of the NTS21G.SOR sorted features which correspond to the name can be found in Figure 24. A plot of them is given in Figure 25. The user can either sort the entire file of matches for NTS21G by typing **sortfeat nts21g.mat,** or the name in question can be isolated by editing the file and copying only the data for the name in question to a new file as in this example.

```
2300 Bliss Island          450100 665000 670722 4986868
  1 Centroid 670172 4987485 Dist   826.552m  Index   5  Area is    14.97491
  2 Centroid 671643 4988350 Dist  1744.868m  Index   2  Area is       0.07976
  3 Centroid 671068 4988595 Dist  1761.319m  Index   3  Area is   34366.99313
  4 Centroid 671154 4985055 Dist  1863.758m  Index  22  Area is   55590.57773
  5 Centroid 669522 4988549 Dist  2065.372m  Index   4  Area is       0.36794
  6 Centroid 668485 4987983 Dist  2499.479m  Index   9  Area is       0.34289
  7 Centroid 671557 4989263 Dist  2536.385m  Index  16  Area is       0.24862
  8 Centroid 668284 4987608 Dist  2547.831m  Index  12  Area is       1.75970
  9 Centroid 670682 4989417 Dist  2549.314m  Index  18  Area is       0.34429
 10 Centroid 668664 4988474 Dist  2610.479m  Index   8  Area is       0.13542
 11 Centroid 668461 4988289 Dist  2670.461m  Index  10  Area is       0.33284
 12 Centroid 668118 4987936 Dist  2814.505m  Index  13  Area is       0.25502
 13 Centroid 670588 4989708 Dist  2843.160m  Index  19  Area is       0.33023
 14 Centroid 668516 4988896 Dist  2996.535m  Index  11  Area is       0.32239
 15 Centroid 671273 4989995 Dist  3175.174m  Index  17  Area is   34361.79988
 16 Centroid 669335 4989750 Dist  3198.389m  Index  20  Area is       0.20752
 17 Centroid 669129 4989657 Dist  3211.880m  Index   6  Area is   27855.10225
 18 Centroid 670088 4991258 Dist  4435.545m  Index   7  Area is   27913.34115
 19 Centroid 666117 4989270 Dist  5193.807m  Index  21  Area is       0.48578
 20 Centroid 665435 4985364 Dist  5496.761m  Index   1  Area is       2.76578
 21 Centroid 665175 4986380 Dist  5568.425m  Index  15  Area is       3.54822
 22 Centroid 666312 4990391 Dist  5644.433m  Index  23  Area is  227176.47009
 23 Centroid 665138 4989092 Dist  6010.593m  Index  14  Area is       0.30898
```

Figure 24. A sample of the sorted possible features for "Bliss Island".

Once a user has used the matching and sorting software, manual matching is required to determine which of the multiple matches is correct. Software has been provided to aid in this task. In order to plot the names and features, labeling software is available.

Three labeling programs have been implemented. The first is called **labels.exe**. This program can be used to produce a file containing HPGL code to be displayed on the screen by the **select.exe** program. The second program, **labnames.exe,** will take part of the matched data and produce a file containing the labels for the names in HPGL which is sent to a printer. The final labeling tool is **labfeat.exe**. This program takes part of the sorted features data, for example the sorted features which are possible matches for Bliss Island, and produces a file containing the labels f1, f2, ...fn which correspond to the feature numbering scheme of the sorted list. These labels are then combined with the features a marker for the name, along with the centroid of the nearby features numbered in sorted order by increasing distance from the name. An example of this was done using the labfeat.exe and labnames.exe with the Bliss Island possible features and the result was saved in the file **BLISS.ALL**. It should be noted that the **+1** in Figure 25 corresponds to the position for the name "Bliss Island". From Figure 25, it can be determined that the name "Bliss Island" probably refers to feature 1 of the sorted list in Figure 24.

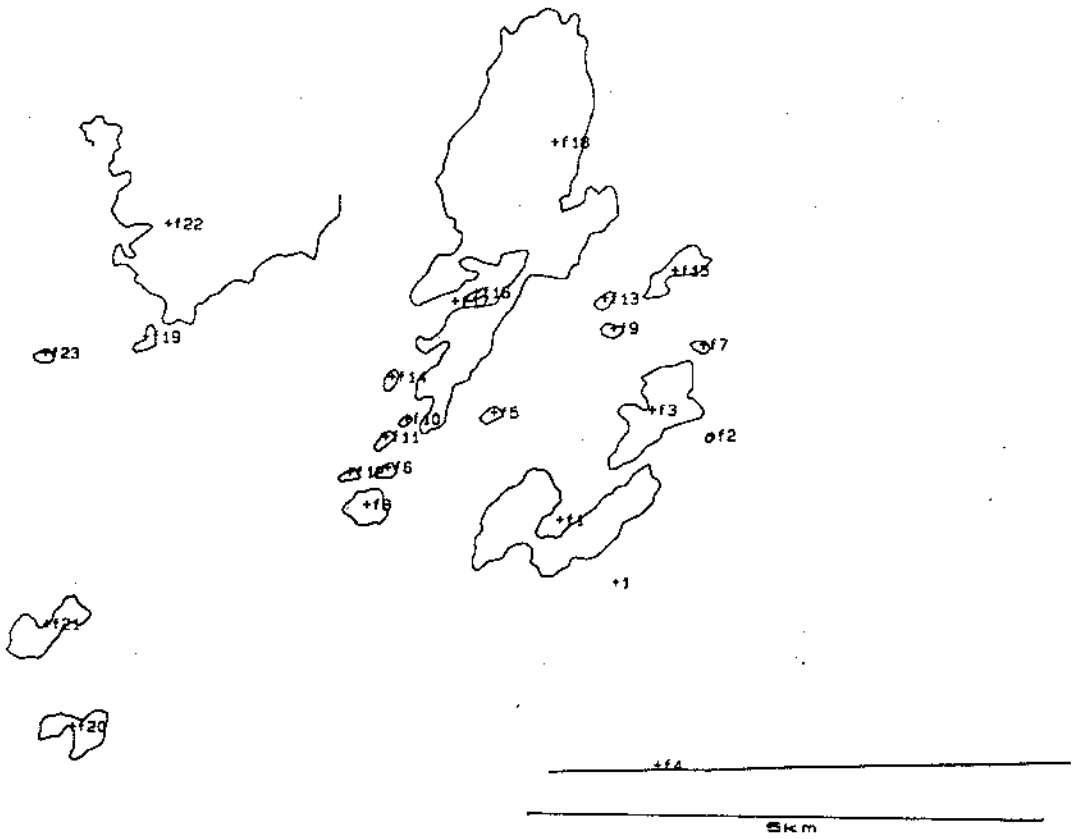


Figure 25. "Bliss Island" possibilities generated by labeling software.

There may be many matches to be inspected by the user and it may not be worthwhile to plot each one. The select software described next, allows the user to eliminate many of the possible matches by looking at the region on the CRT.

3.1.3 Interactive Matching

The ability to display a map with its associated names is provided by the select software. Since the feature file is very large for one map sheet, the map is divided into 16 smaller windows. This division means that as the user zooms in and out looking at the

map, an average of 0.5 MB of data is read by the program. These 16 smaller windows are generated by the extraction software. A script generator program written in PLI creates a script which calls the extract program once per window to generate the input files for the extract program. Zooming is done by splitting the extracted data and names into 16 smaller files. These subdivisions are numbered as depicted in Figure 26. The bounding boxes allow the program to avoid parsing lines of input which are not displayed on the screen. These addition of a bounding box and splitting the data into smaller portions greatly improved the performance of the select software on the IBM PS/2 workstation.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figure 26. The numbering scheme for the select software.

The **select.exe** program begins by displaying an overview of the entire map. The zooming can be done by use of the mouse. To zoom in on a grid the left mouse must be used. To back up a level the right mouse is used; however if the user wishes to leave the application, the escape (**Esc**) button will work when the features are being drawn on the CRT. In order to display the NTS21G map sheet the user will type **select nts21g**. The sequence of steps used to generate the input files is given in Paine and Mullin [1992]. Figure 27 shows that there are 16 plus 1 feature files and 16 name files. The one extra feature file is an overview of the entire map sheet, and it contains only a few feature types

and does not contain any names. There are also 17 parameter files giving a total of 50 input files.

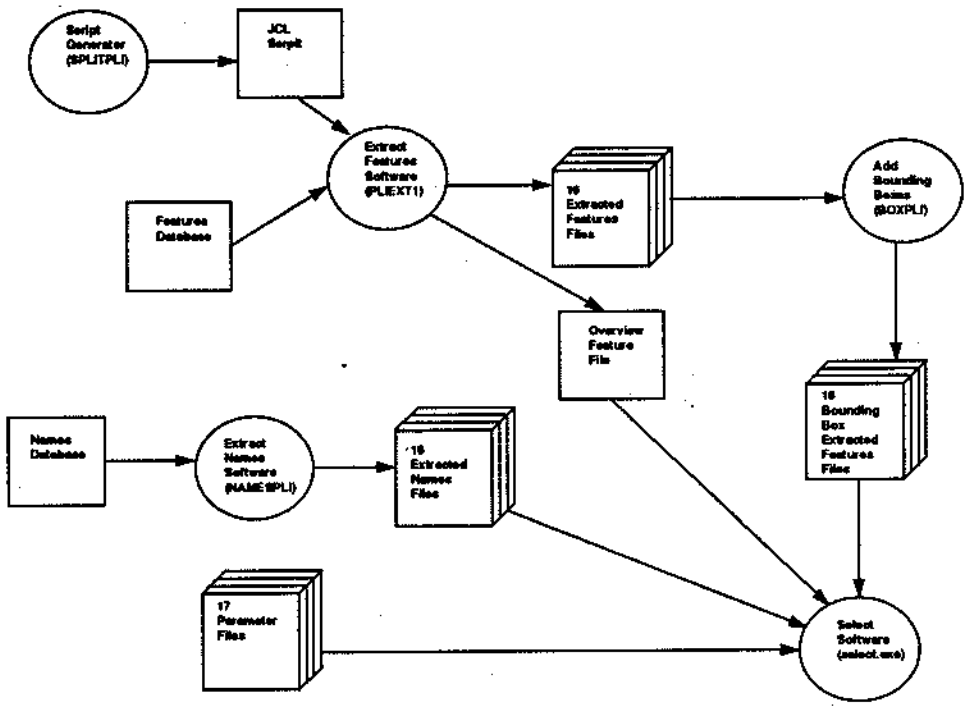


Figure 27. An architecture for extracting EMR data.

A summary of the steps which should be done in order to match the EMR names to the EMR features is as follows:

a) use the **select.exe** software to determine if the names extracted should be on the map sheet at this scale

b) use the match.exe software to find possible matches

c) sort the features with the sortfeat.exe software to determine as many matches as possible referring to the original map sheet.

d) for the remaining names having more than one possible match identified in step c), plot the features and names on the printer and compare them to the original map sheet to find the remaining matches.

A test for the points on the NTS21G map sheet was carried out. The results of this experiment, which took a few weeks to complete, resulted in the classification of 309 point names. The exercise involved matching the names to features, sorting, plotting, using the select software and consulting the NTS21G map sheet in order to determine correct matches. The findings of the matching experiment are found in Appendix C.

3.2 The Knowledge Base

The knowledge base was considered but was not developed. The names frame structure was implemented in ART-IM [Inference, 1991]. Two simple rules were tried; one to determine that a name had not been placed and another to detect that the name for an island had been placed in a crowded area. These rules, however, do not call the C function to act on the rules; they simple print a message indicating that the situation has been detected. This early test implementation can be found in Appendix D.

3.3 A Quadtree with Objects

The implementation of the design for a spatial data structure was carried out. This stage of the thesis can be divided into three major tasks. The first barrier to be overcome was the management of memory. Once the memory was accessible, new construction and union operations for quadtrees with objects were implemented and tested.

3.3.1 Memory Management

The actual generation of the quadtree for automated name placement has been done using Microsoft C version 7 and the Software Development Kit version 3.1 [Microsoft, 1992]. It was run on an IBM PS/2 model 90 workstation, which included an Intel 486DX, 33 MHz with 16 megabytes of RAM.

In order to generate quadtrees that can represent the entire map sheet, a very large structure must be built. Most of the conventional DOS C compilers are not designed to deal with such large structures. In order to implement the structure designed, the decision was made to use 32-bit pointers. This capability is not part of standard Microsoft C7, but a dynamic link library (DLL) has been provided. The functions in the WinMem32.DLL allow the programmer to allocate units of memory of up to 16 megabytes at once, create a pointer to the memory, release a pointer to the memory and release the memory. There is a limit however, on the number of times you can allocate space. By running a small program containing a loop to allocate nodes, I found the maximum number of possible handles to be limited to just over 2000 . For this reason, allocating one quadtree node per call is not feasible. The allocation is done by allocating a large number of nodes with one call and offsetting into the region to access specific nodes. When the nodes for one handle are full, another one is allocated.

Maintaining the allocated memory was also a consideration. Using the Win32 DLL and the Windows environment has made the memory management quite easy. In order to access allocated memory, a pointer to the desired quadtree node must be established. This causes that particular segment of memory to be moved into the current segment. When you are finished with the node, the pointer to the node is released so that the current page is "unlocked" and can be paged. The windows memory manager pages the allocated space to the virtual memory of the machine whenever necessary. This paging seems to work very well and the time to create a quadtree with 3049 nodes is only 35 seconds.

Appendix E contains a description of the code used to generate nodes for the object quadtree.

3.3.2 Modified Construction Operation

In the CONSTR algorithm of Figure 28, a type called pair has been used. Pair is a structure which contains two pieces of information: the color of the node and a pointer to a quadtree. The definitions of pair, qtree, and link are:

```
type
  pair: record
    color:(B,W,G);
    point: qtree
end;

type
  qtree: ->tree;
  tree: record
    color: (B,W,G);
     NE, NW, SE, SW: qtree;
     parent: qtree;
     llist: ->link
end;
```

```
type
  link: record
    code_letter: (f,g,n);
    code_number: integer;
    next: ->link
end;
```

```
function CONSTR(n, x, y, ASC_code:integer):pair;
{ Construct a quadtree from an image of 2**n by 2**n which has an
    origin of (0,0) being the SW corner }

   var
      p: array[1..4] of pair;
      par: pair;
      r, q: qtree;
      level, i: integer;

   begin
      if ( n = 0)   { process the pixel }
         begin
            if IMAGE(x, y) = 0
               par.color := 'W'
            else
               par.color := 'B';
               par.point := null;
         end
      else
         begin      { process the non-pixel }
            level := n  - 1;
                                                { NW corner }
            if ( (xmin <= x-2**level) and (ymax >= y-2**level)
               p[1] := CONSTR(level, x-2**level,y,ASC_code);
            else
               p[1].color = "W";
                                                { NE corner }
            if ( (xmax >= x-2**level) and (ymax >= y-2**level)
               p[2] := CONSTR(level, x,y,ACS_code);
            else
               p[2].color = "W";
                                                { SW corner }
            if ( (xmin <= x-2**level) and (ymin <= y-2**level)
               p[3] := CONSTR(level, x-2**level,y-2**level,ASC_code);
            else
               p[3].color = "W";
                                                { SE corner }
            if ( (xmax >= x-2**level) and (ymin <= y-2**level)
               p[4] := CONSTR(level, x, y-2**level);
            else
               p[4].color = "W";

            if (p[1].color  = p[2].color) and
               (p[1].color  = p[3].color) and
               (p[1].color  = p[4].color) )
               par := p[1]                { All children of same type }
            else
```

Figure 28. Modified construct algorithm. (pg 1 of 2)

```
                begin                    {create a non-terminal GRAY node }
                    new(q);
                   for i := 1 to 4 do
                       begin
                         if p[i].color = "G"    {link p[i] to its parent}
                             begin
                               case i of
                                   1: q->NW = p[i].point;
                                   2: q->NE = p[i].point;
                                   3: q->SW = p[i].point;
                                   4: q->SE = p[i].point;
                               end; {case}
                             end;
                          else
                            begin
                               new(r);
                               r->color := p[i].color;
                               if ( r->color = "B" )
                                   AddBlack(r,ASC_code);
                               r->NW := r->NE := r->SW := r->SE := null;
                               case i of
                                  1: q->NW = r;
                                  2: q->NE = r;
                                  3: q->SW = r;
                                  4: q->SE = r;
                               end; {case}
                               r->Parent := q;
                            end
                       end        {for i = 1 to 4 }
                   q->color = "G";
                   par.point := q;
                   par.color :="G"
               end { create a non-terminal gray node }
           end {for non-pixel }
    return(par);
end; { CONSTR }
```

Figure 28. Modified construct algorithm, continued. (pg 2 of 2)

As you can see from Figure 28, the bounding box is used to allow pruning of the branches from the top down rather than bottom up. In this implementation, the bounding box of a feature is stored in the global variables xmin, xmax, ymin, and ymax. The modified CONSTR algorithm also needs one additional function. The routine AddBlack simply allocates a linked list node containing the graphic group number and attaches it to the black quadtree leaf node r.

This modified quadtree construction takes substantially less time than the original algorithm. For a test case which has data from the NTS21G map sheet, 3049 quadtree nodes were generated using this algorithm for a quadtree with fourteen levels. If the same data were used with the original algorithm, then over 268 million nodes would be generated for the fourteenth level nodes, with millions of interior nodes as well. This is because the original quadtree construct algorithm visits each pixel to determine its color. Visiting vast regions of the map which are not in question to unnecessarily determine that they are white adds substantial time and space requirements to the problem. In the test case mentioned above, the whole tree was generated using the modified algorithm in 35 seconds. Using the original algorithm, after over an hour of processing the construct routine was still working on the first quarter of the quadtree.

3.3.3 Modified Union Operation

The standard union of two quadtrees is a simple algorithm. The resulting quadtree contains all of the black nodes in either of the original quadtrees. This union algorithm is shown in Figure 29. In this union, the resultant quadtree will be U.

```
QTUnion(S:qtree, var U: qtree);

   if S -> color = W then return;
   if U -> color = B then return;
   if S -> color = B and U -> color = W then U -> color = B
   else if S -> color = B and U -> color = G then U := S
   else if S -> color = G and U -> color = W then U := S
   else if S -> color = G and U -> color = G then
   begin
      QTUnion(S -> NW, U -> NW);
      QTUnion(S -> NE, U -> NE);
      QTUnion(S -> SW, U -> SW);
      QTUnion(S -> SE, U -> SE);
   end;
end;
```

Figure 29. Standard quadtree union algorithm [Samet, 1990].

In order to form a union two object quadtrees S and U, the quadtrees for each of the spaces must first be generated. The quadtrees are then traversed, blackening any node of U which is black in S. Since the quadtree must also maintain the object to which the nodes belong, this will also result in adding linked lists to existing black nodes of U and possibly adding new black nodes. Using this algorithm, the resultant quadtree may include interior gray nodes which have four black children if the children are from different features.

The quadtree for the entire space is initially null or empty. The features are added into the structure one at a time, by generating a quadtree representation for the feature and adding it to the existing quadtree. The addition of a new feature to the existing quadtree is accomplished by doing a modified union on the two quadtrees. The union has been implemented so that if the union of quadtrees S and U is formed, the resultant quadtree will be U, the universal quadtree. The pseudo-code for the algorithm used in order to form the union of two quadtrees is shown in Figure 30.

```
OQTUnion(S:qtree, var U: qtree);

   if S -> color = W then return
   else if S -> color = B and U -> color = B then
      Update(U, S-> link list)
   else if S -> color = B and U -> color = W then
      U -> color = B
      Update(U, S-> link list)
   else if S -> color = B and U -> color = G then
   begin
      Update(U -> NW, S-> link list);
      Update(U -> NE, S-> link list);
      Update(U -> SW, S-> link list);
      Update(U -> SE, S-> link list);
   end;
   else if S -> color = G and U -> color = W then
      U := S
   else if S -> color = G and U -> color = B then
   begin
      Update(S -> NW, U-> link list);
      Update(S -> NE, U-> link list);
      Update(S -> SW, U-> link list);
      Update(S -> SE, U-> link list);
      U := S;
   end;
   else if S -> color = G and U -> color = G then
   begin
      OQTUnion(S -> NW, U -> NW);
      OQTUnion(S -> NE, U -> NE);
      OQTUnion(S -> SW, U -> SW);
      OQTUnion(S -> SE, U -> SE);
   end;
end;
```

Figure 30. Modified quadtree union algorithm.

The modified union makes use of the routine Update which copies a linked list of one node to the end of the linked list of another node. In this routine the node which is to get the additional information has been labeled D and the node with the required information is E. The pseudo-code for this routine is shown in Figure 31.

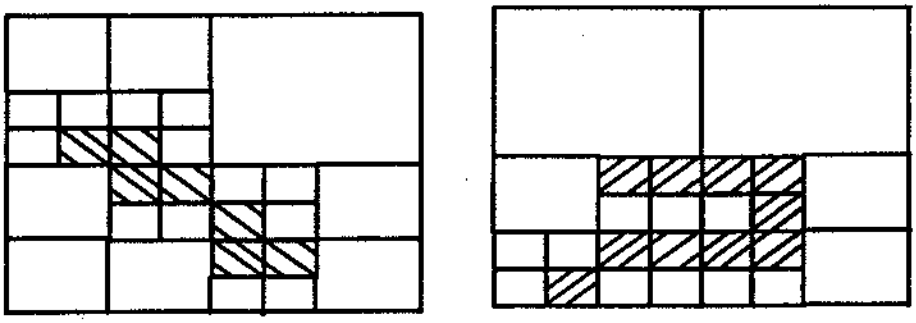
```
Update(var D_head: qtree, E_head: llist);

   if D_head -> color = 'B' then
      copy the black linked list of E to the linked list of D
   else if D_head -> color = 'W' then
   begin
      D_head -> color = 'B'
      copy the black linked list of E to the linked list of D
   end
   else
   begin
      Update(D_head -> NW, E_head);
      Update(D_head -> NE, E_head);
      Update(D_head -> SW, E_head);
      Update(D_head -> SE, E_head);
   end
end;
```

Figure 31. Update algorithm.

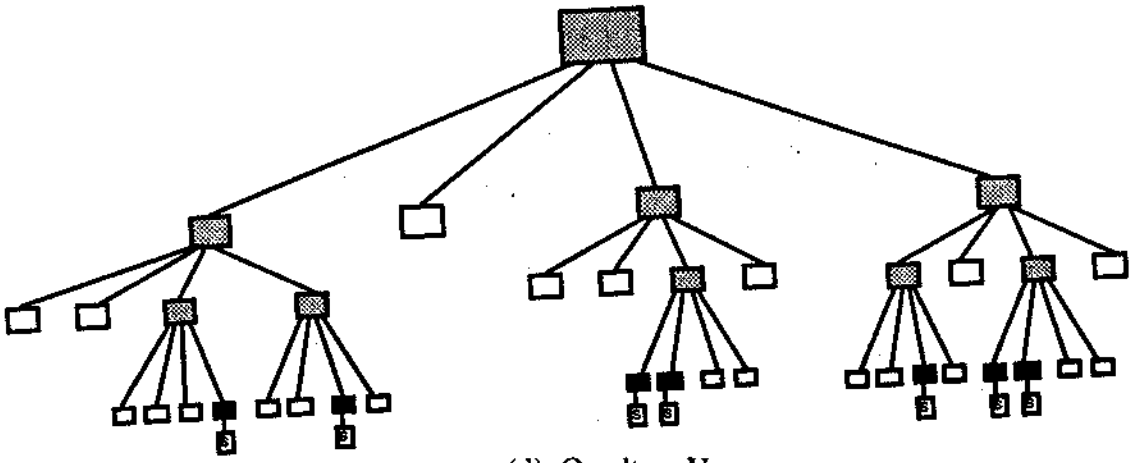
Figure 32 is the modified quadtree union. This diagram has two quadtrees, S and U, and the resulting quadtree U when the two are combined. For this example, it is assumed that S and U represent different types of features.

(a) Space of S

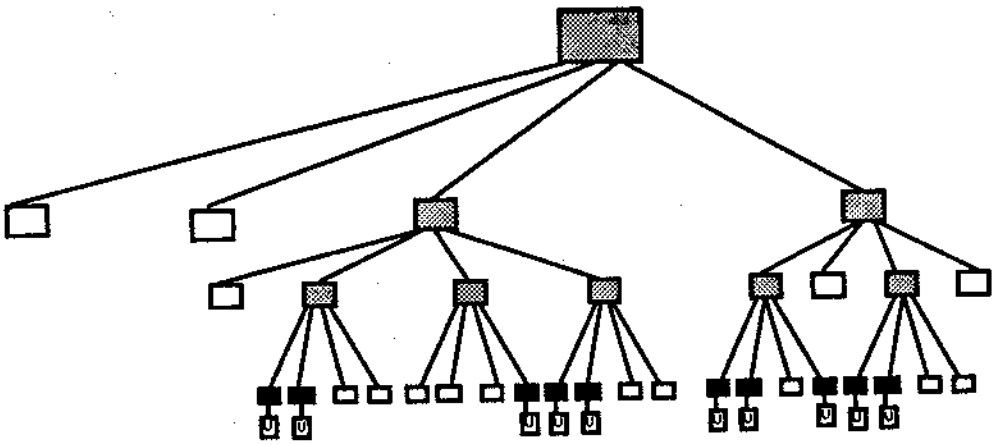
(b) Space of U



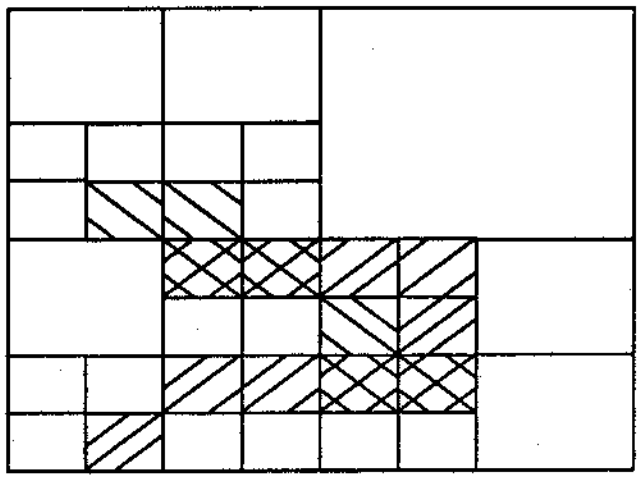
(c) Quadtree S



(d) Quadtree U



(e) Space of S union U



(f) Quadtree S union U

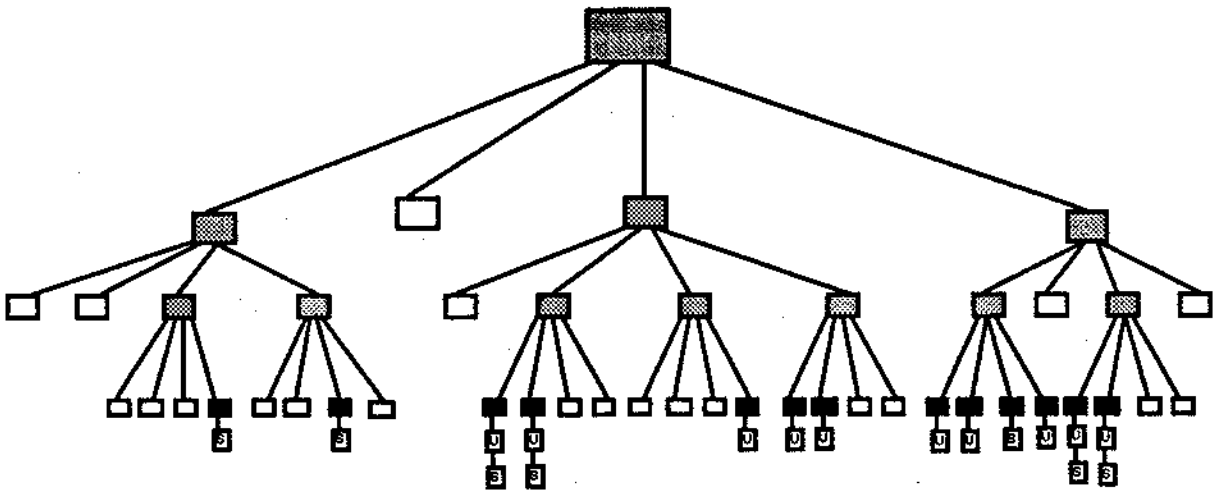


Figure 32. Sample union of two quadtrees.

Figures 32(a) and 32(b) are the spaces on which we want to form a union. The resulting combined space is shown in Figure 32(e). The slanted fill lines in the object indicates which space the pixels belong to. We can see that when the union of two spaces, some of the pixels will be black since they were black in S, some of the pixels will be black since they were black in U, and some of the pixels will be black since they were black in S and in U.

Before the union can be formed, the two quadtrees with objects must be generated. These quadtrees are shown in Figures 32(c) and 32(d). These quadtrees have the object information maintained in linked lists descending from the black leaf nodes. When the union of S and U is formed Figure 32(f) is generated. From this diagram we see that some of the black leaf nodes have one node in the linked list and some have two. The black quadtree nodes with two nodes in the linked list indicate that the pixel was black in both S and U. It should also be noted that one of the interior gray nodes in the last subdivision of Figure 32(f) has four black leaf nodes but has not been pruned. This is done because the objects to which the individual pixels belong must be maintained. This is important since it is essential that we can determine what objects are occupying specific pixels. In particular, the name placement rules dealing with interference resolution hinge on being able to determine what the name is in conflict with.

The names to be placed will be added to the spatial data structure. This will be done by first generating the appropriate run length code for the name in a preprocessing step. This run length code is based on a bit map for the name with the appropriate font and has a lower left hand corner anchored at an origin of (0,0). When the name is placed, this run length code is shifted into the appropriate quadtree coordinate frame. During the name

placement process a quadtree S for each name will be built and then the union of the name quadtree with the universal quadtree U will be formed.

Each name has a unique number associated with it. This number is simply assigned in sequence when the name is read into the knowledge base. This unique number or key is required in order to differentiate one name from another. When a name is placed, but the placement is found to be unacceptable, then the name will have to be removed from the quadtree. If there are two names occupying the same black pixel, removing one of the names from the U should in no way impact the representation of the other name in U. Without a unique key, there is no way of determining which of the black nodes belong to each name.

Detection of interference for a name being placed is easily handled with the spatial data structure designed in this thesis. Not only is interference detectable, but a means of determining what specifically is in conflict with the name being placed is also possible. When a name is placed, i.e. the union of its quadtree S with U is formed, an overlap may be detected. Overlapping is triggered when a black node of the name quadtree S is added to the linked list of a node in U which is already black. Since the key for the name is added to the end of the linked list, then traversing the list to find what is in conflict with the name can be easily done.

A name can overlap a feature, a grid line or another name which has already been placed. In order to determine if the name placement is satisfactory, the knowledge based system is required. The rules in the knowledge base could determine that the placement is

satisfactory, the name just placed should be moved, or that another name in conflict should be moved.

The implementation of the quadtree with objects for an entire map sheet on the platform chosen took substantial effort but the results look promising. The quadtree can maintain all of the required information and the time for generation is reasonable. The memory management problems encountered are limitations which were dealt with, but will not be a limitation of future microcomputers as it seems that 32 bit pointers will become the norm and the software will be geared to these pointers.

CHAPTER 4.

PARTIAL RESULTS FOR NTS21G

Although the names have not been placed, the EMR data for some features from the map sheet NTS21G has been placed into the object quadtree structure. The diagram in Figure 33 is a representation of the entire map sheet space for a few features. The quadtree generated for these features resulted in 3049 nodes, 1585 white, 765 black, and 711 gray. The quadtree diagrams of Figures 33, 34, and 35 have been generated using the quad2ps software [Balakrishnan, 1992]. This software generates a postscript file from a linear quadtree representation.

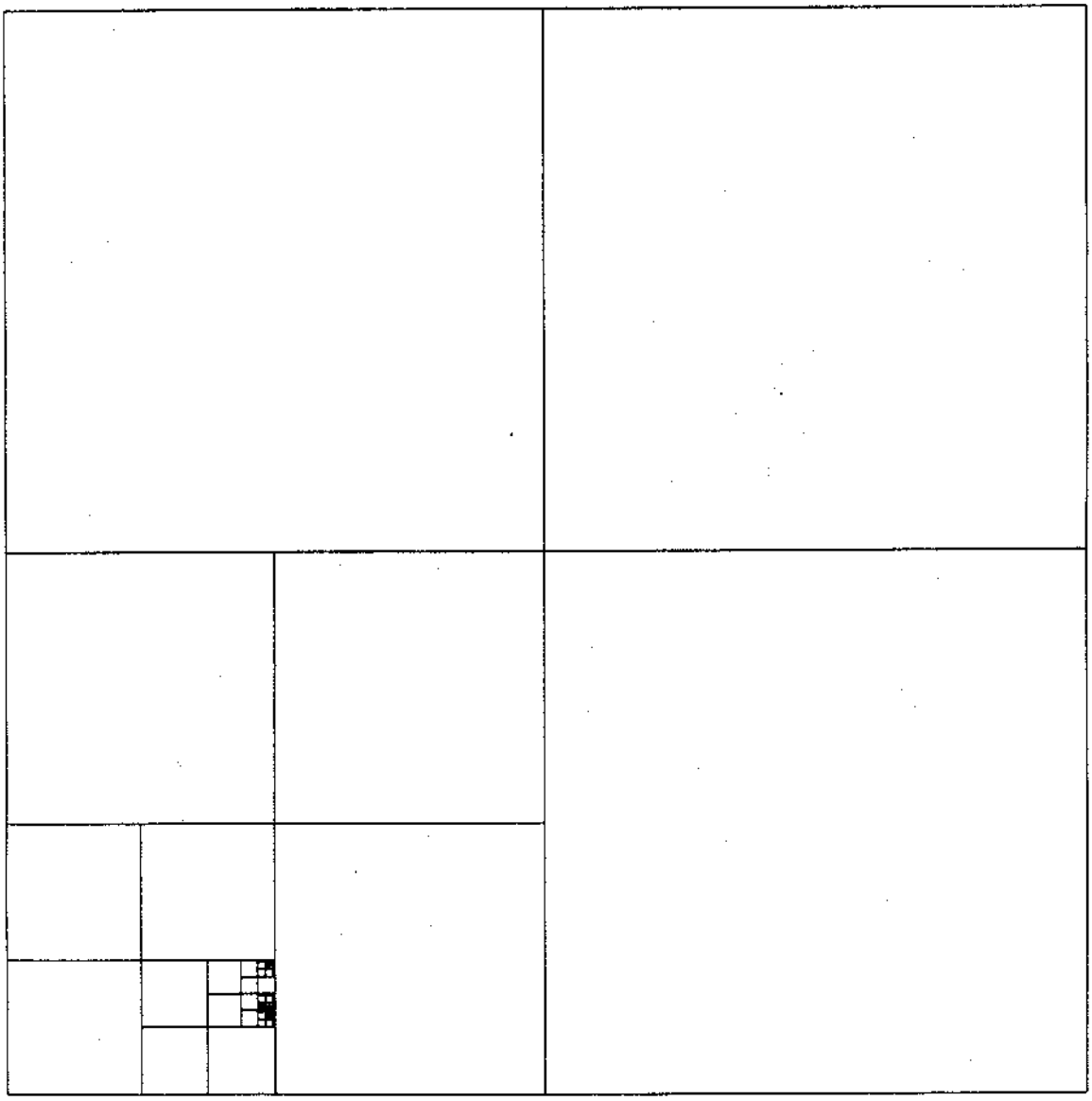


Figure 33. Quadtree generated for a few EMR features.

These two regions are then enlarged in Figures 34 and 35 by manually changing the linear quadtree to eliminate top levels of the quadtree. Figure 34 has been altered to have the first four levels of the quadtree not shown. This output is then changed to eliminate one more level to show only the south east features in Figure 35.

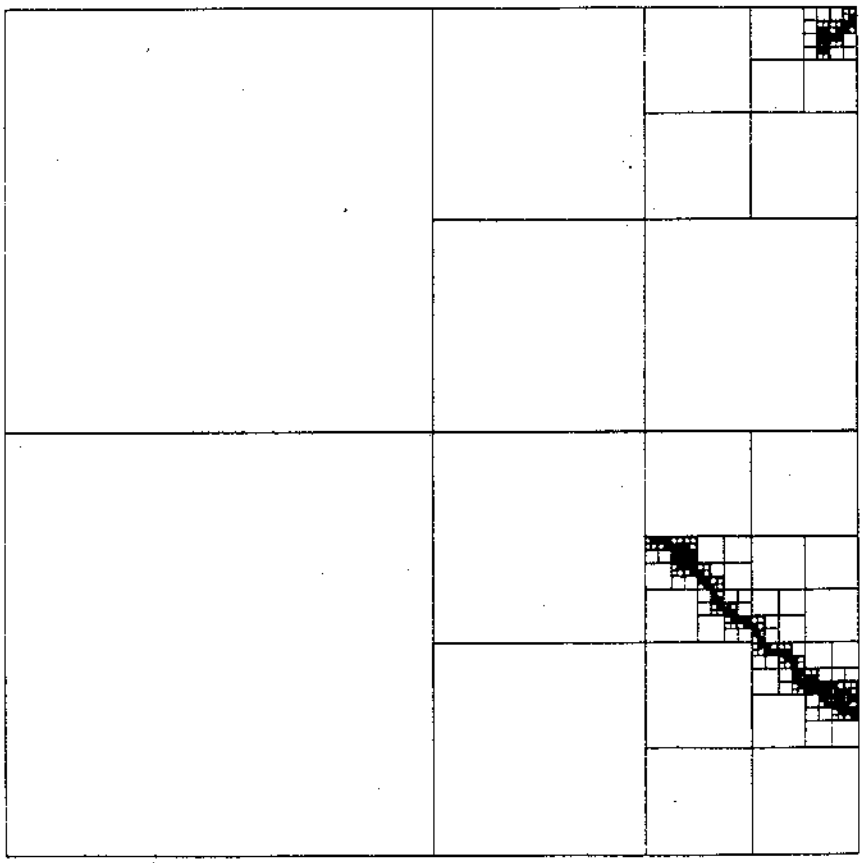


Figure 34. Zoom in four levels on the EMR test features object quadtree.

The shoreline feature introduced in Figure 13 section 2.5 as an example is present in the test case shown in Figure 35. It appears on the right side near the top of the long shoreline shown below.

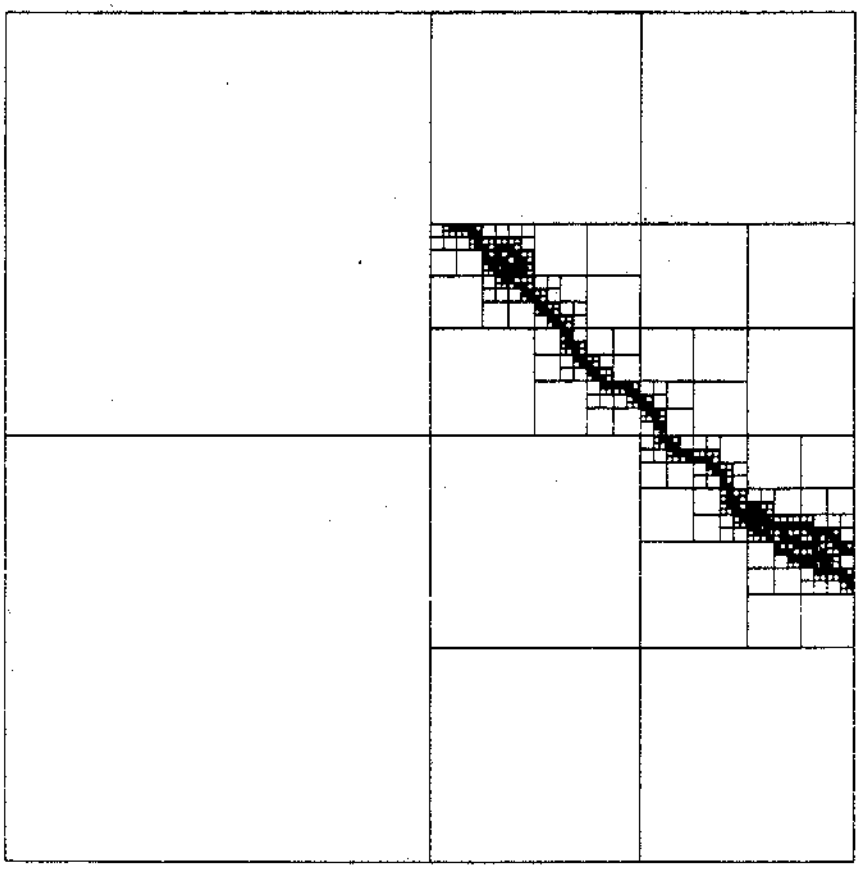


Figure 35. Zoom in five levels on the EMR test features object quadtree.

Figure 36 shows a portion of the paper map version of area around these features. This is shown as approximately 6 times larger than scale (i.e. 1 cm on the original map ≅ 6 cm here).

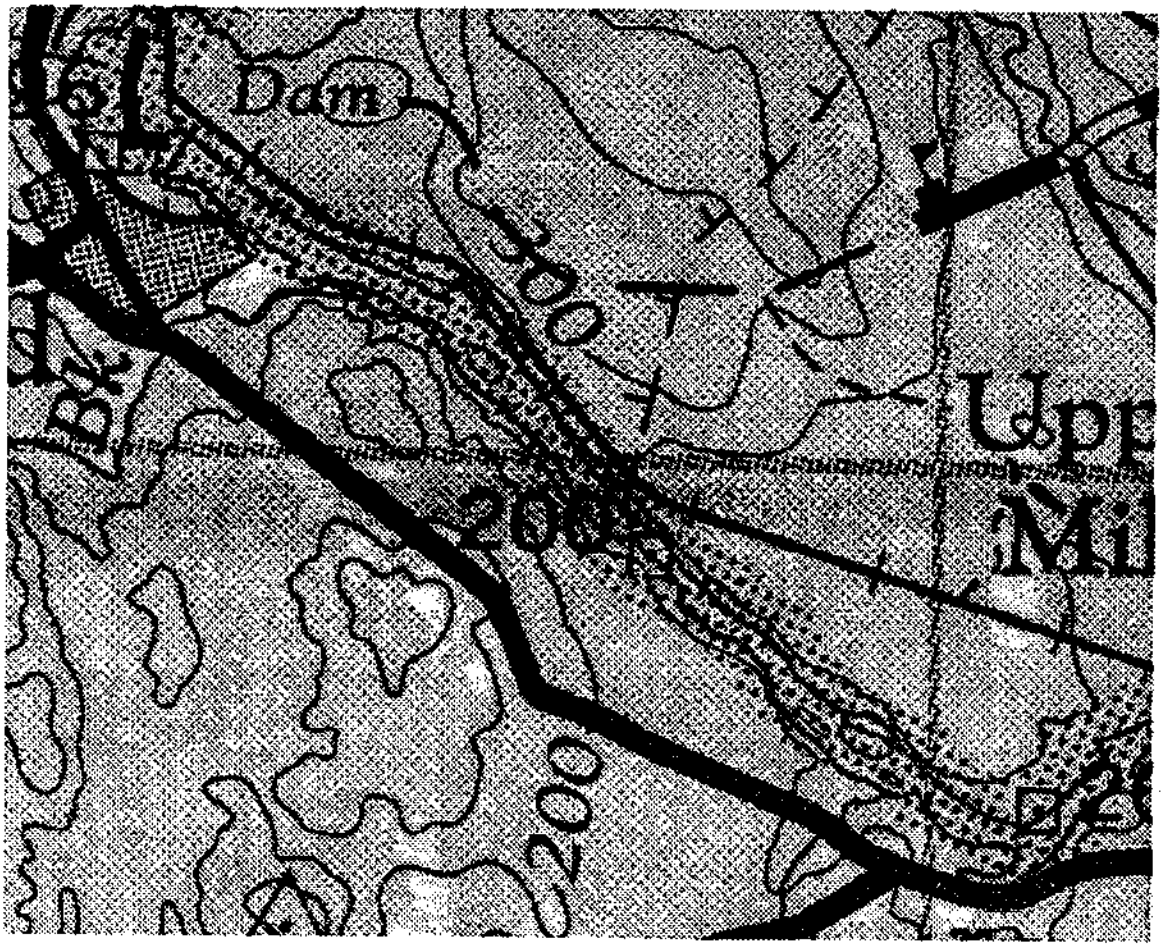


Figure 36. A portion of the paper map test area.

The features shown in this section comprise eight features from the actual test data. Only eight were shown because the testing of the quadtree for map features was concentrated on this sample. These features were chosen because they were relatively close together, allowing the quadtree to be enlarged for closer inspection. In order to test

actual data, each step of the quadtree generation must be inspected. Working with large volumes of data was not done for these results, as validation is very time consuming.

CHAPTER 5.

CONCLUSIONS

The investigation into the design and implementation of data structures and an expert system to deal with name placement on EMR Canadian topographic maps was carried out. The goal of the research was to determine how an expert system can best interact with the spatial data requirements of the name placement problem. A detailed design for a Name Placement by Expert System (NAPLES) prototype was done. A series of programs were written to perform many of the preliminary processing steps necessary for the two databases involved. An appropriate spatial data structure was developed which allows object subtraction and overlap identification to be done.

5.1 Summary

The five objectives for the thesis will now be examined to determine what exactly was accomplished by this research effort and what the contribution of this work is to the problem of automated name placement.

The first objective was: "To discover methods for representing rules about name placement which do not fall in the IF... THEN... category of rules used in most expert systems." This goal was considered in great detail. The rules from EMR were identified and written down in a form which can be incorporated into NAPLES. Some of the rules of thumb can be implemented quite easily, but some of them will require further expert knowledge into exactly what to do in specific situations rather than in the general case. These rules, however were not implemented in an expert system.

"To investigate the use of a rule based expert system for automated name placement using the rule representation discovered by the first objective" was the second goal. This

rule based expert system was designed and its use does seem encouraging. This work, however was limited to point names.

The third goal was: "To discover a data structure suitable for the integration of the names and their geographical features." This objective became one of the two major focal points of the thesis. A novel object-oriented quadtree structure which supports the subtraction or set difference operation has been designed, built, and tested. Tests were done using real EMR data to show insertion into an object quadtree of depth 14. The structure currently has features inserted into it, but all that is needed in order to add names is the raster representation of the name in the proper font. This object quadtree forms the basis for implementing the full NAPLES system.

The fourth objective was: "To investigate ways in which names can be automatically matched with their associated geographical features." This was a large undertaking in the research effort. The two different databases (i.e. the names database and the features database) did not have a close correlation. Initial design of an appropriate algorithm for matching names to features did not include sorting, plotting and displaying the data. A complete suite of preprocessing software to correlate the names and features has been produced. This software is essential for any production name placement to insure that names which are supposed to appear on a particular map are not missed.

The final objective of this thesis was:" To develop a prototype expert system for name placement using names and topographic data from the National Topographic Data Base for the Fredericton, New Brunswick area." A prototype for complete production quality, semi-automated name placement was designed, and a good portion of it was implemented. The rule-based system and name fonts still have to be addressed in order for

the prototype to be complete. The data from Energy, Mines and Resources was used in the prototype, and the prototype does demonstrate that at least semi-automated name placement is possible for Canadian topographic maps.

Besides the initial objectives of the thesis, the difficult memory management problems for producing very large quadtrees were overcome. This memory management solution can be incorporated in various other applications in which large data structures are inherent.

This research also produced two additional findings. The first finding was that there are a substantial numbers of names which cannot be matched to a feature, do not appear on particular map scales, or which appear without the associated feature being displayed. Initially the expectation was that most point names would match a feature; however, only 20 of 309 point names were matched, as shown in the test results in Appendix C. The second realization was how big the problem of automated name placement really is. The work completed thus far indicates that the task is substantial and will continue to be a research area for a number of years.

There are various merits of this research which constitute a contribution to the name placement area of research. The first contribution is that the data used in this thesis is genuine cartographic data for Canadian topographic maps. Many systems for automated name placement do not use real data because of the added complexity it imposes. Another achievement of this thesis is that the work was all based on the genuine cartographic rules used in production. Very few research efforts have been geared to the Canadian data and rules. The final contribution of this thesis is the design and implementation of a spatial data structure which is appropriate to the name placement problem. This data structure is

an object quadtree which allows the names to be inserted and subtracted while detecting interference. Although not all of the objectives were meet, all indications are that computers can, and will be used to aid in the time consuming task of name placement in the future.

5.2 Future Work

The above accomplishment represents a significant first step in accomplishing the original objectives. To make NAPLES into a workable name placement prototype, the following work needs to be done:

1. Extend the current rule base to include area and linear rules.
2. Design and implement methods for the interaction of the names frame structure with the object quadtree.
3. Build a tool for CRT and printer display of the results.
4. Consider using the full raster representation of the name instead of a bounding box for determining overlap and feature intersection.
5. Add integration with the Statistics Canada database of populations for determining font size and to assist the matching of names to features.

REFERENCES

Ahn, J.K., "Automatic Map Name Placement System", report IPL-TR- 063, Electrical, Computer and Systems Engineering Dept., Rensselaer Polytechnic Institute, Troy, NY, 1984.

Arendtz,J., "GenFont Developer's Guide", User Bulletin 130, UNB AI Laboratory, Faculty of Computer Science, April, 1992.

Balakrishnan, R., "Quad2ps User's Guide", User Bulletin 132, UNB AI Laboratory, Faculty of Computer Science, November, 1992.

Doerschler, J.S., "A Ruled-Based System for Dense-Map Name Placement", report SR-005, Center for Computer Aids for Industrial Productivity, Ruters - The State University of New Jersey, New Brunswick, NJ, 1987.

Doerschler, J.S. and Freeman, H., "A Ruled-Based System for Dense Map Name Placement", Communications of the ACM, vol. 35, no. 1, 1992, pp. 68-79.

Ebinger, L.R., and Goulette, A.M., "Automated Names Placement in a Non-Interactive Environment", AUTO CARTO 9, Ninth International Symposium on Computer-Assisted Cartography, Baltimore, Maryland, April 2-7, 1989, pp. 205-214.

EMR, "Approved Names of Canada", Peter Revie, Topographic Mapping Division, Canada Centre for Mapping, Ottawa, Ontario, October, 1989.

EMR, "Digital Topographic Data Distribution Information", Software Section, Systems Engineering Group, Topographic Surveys Division, Energy, Mines & Resources, Surveys & Mappings Branch, 162-615 Booth Street, Ottawa, Ontario, May, 1987.

EMR, "Standards and Specifications", Topographic Mapping Division, Canada Centre for Mapping, Ottawa, Ontario, First Edition, 1988.

Hirsch, S.A., "An Algorithm for Automatic Name Placement Around Point Data", The American Cartographer, vol. 9, no. 1, 1982, pp. 5-17.

Imhof, E., "Positioning Names on Maps", The American Cartographer, vol. 2, no. 2, 1975, pp. 128-144.

Inference Corporation, "ART-IM Programming Language Reference", Inference Corporation, 550 N Continental Blvd., El Segundo, California, 1991.

Johnson, D.S., and Basoglu, U., "The Use of Artificial Intelligence in the Automated Placement of Cartographic Names", AUTO CARTO 9, Ninth International Symposium on Computer-Assisted Cartography, Baltimore, Maryland, April 2-7, 1989, pp. 225-230.

Jones, C.B., "Cartographic Name Placement with Prolog", IEEE Computer Graphics & Applications, Sept. 1989, pp. 36-47.

Lacroix, V., "An Improved Area-Feature Name Placement", report IPL-TR- 064, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY, 1984.

Mullin, "Matching EMR Names to Features", User Bulletin 124, AI Laboratory, Faculty of Computer Science, March, 1992.

Nickerson, B.G., "Automated Cartographic Generalization for Linear Map Features", Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, NY, May, 1987.

Paine, C. and Mullin, L., "Extracting Data Subsets from Spatial Map Datasets at UNB", User Bulletin 112, AI Laboratory, Faculty of Computer Science, January, 1992.

Samet, H. "Applications of Spatial Data Structures", Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

Yoeli, P., "The Logic of Automated Map Lettering", The Cartographic Journal, vol. 9, no. 2, 1972, pp. 99-108.

Appendix A. The Complete Match Table

The listing below is the complete match table used in this thesis. The lines which begin with ;;; are comment lines. These comments lines were added so that the table could be structured in a similar format as the original list of generic codes. These generic codes have been separated into various types and a comment appears before each indication the new type. The fields in the table are the generic code, a one letter language code (E = English and F = French), the NGNDB description, the ASC code and the NTDB description. If the ASC code is a * then the generic code was not considered in this thesis. A -1 was used to indicate that no corresponding feature was found for a point name. One reason this occurred in the data is because towns and villages are classified based on population. The population statistics data for Canada was obtained from Statistics Canada; however the data has not been incorporated into this thesis. As can be seen from the descriptions, there is no one-to-one correspondence between names and features.

```
;;;1 Populated                    *
;;;Incorporated                   *
1     E City                      *
2     E Town                      1351  Boundary Town
                                  1352  Boundary Town Tint
                                  1353  Boundary Town Unsurveyed
3     E Village                   1354  Boundary Village
                                  1355  Boundary Village Tint
                                  1356  Boundary Village Unsurveyed
4     E Metro Municipality        1309  Boundary Metropolitan Area
                                  1310  Boundary Metropolitan Area Tint
                                  1322  Bndry Metrop Area Unsurveyed
5     E Inc Village Municipality  *
6     E Inc Community             *
7     E Undesignated Municipality *
8     E Borough                   *
9     E Separate Town             *
10    E New Town                  *
11    E Hamlet                    *
12    E Summer Village            *
13    E PopPlace Outside Canada   *
14    F Municipalite              *
```

15	F	Cite	*	
16	E	Resort Municipality	*	
17	E	Organized Hamlet	*	
18	E	Resort Village	*	
19	E	Northern Hamlet	*	
20	E	Northern Settlement	*	
21	E	Northern Hamlet	*	
22	E	Municipality	*	
23	F	Ville (2)	*	
24	F	Cite-jardin	*	
;; Unincorporated			*	
100	E	Hamlet	*	
101	E	Village	*	
102	E	Police Village	*	
103	E	Rural Village	*	
104	E	Settlement	*	
105	E	Compact Rural Community	*	
106	F	Depot	*	
107	E	Resort	*	
108	F	Locality	*	
109	F	Lieu-dit	*	
110	E	Railway Junction	*	
111	E	Railway Point	*	
112	E	Station	*	
113	E	Post Office	4120	Post Office to Scale
			4450	Post Office
114	E	Landing	*	
115	E	Siding	*	
116	F	Depot	*	
117	E	Abandoned Locality	*	
118	E	Railway Siding	2420	Railway Siding
			2421	Railway Siding Abandoned
119	E	Trading Post	4132	Trading Post to Scale
			4133	Trading Post
120	E	Fort	4310	Fort to Scale
			4311	Fort Symbol
121	E	Community	-1	A Point with no Feature
122	E	Dispersed Rural Community	-1	A Point with no Feature
123	F	Hameau	*	
124	F	Localite	-1	A Point with no Feature
125	F	Bureau de poste	4120	Post Office to Scale
			4450	Post Office
126	F	Ville (1)	*	
127	F	Jonction	*	
128	F	Arret	*	
129	F	Gare	*	
130	F	Bureau de poste militaire	*	
131	F	Centre de villegiataire	*	
132	E	Weather Station	*	
133	E	Hutterite Colony	*	
134	E	Railway Stop	2500	Railway Stop
			2501	Railway Stop to Scale
135	F	Ville miniere	*	

```
136  F Voie d'evitement          *
137  E Colony                    -1    A Point with no Feature
138  E Junction                  *
139  E Trailer Park              7550  Trailer Park
140  E Abandoned Community       *
141  E Customs Point             4364  Customs Symbol - Flag
142  E Townsite                  *
143  E Summer Post Office        *
144  E Flag Stop                 *
145  E Tent-Trailer Park         *
146  E Indian Community          1306  Boundary Indian Reserve
                                 1307  Boundary Indian Reserve Tint
                                 1308  Bndry Indian Reserve Unsurveyed
147  E Indian Settlement         1306  Boundary Indian Reserve
                                 1307  Boundary Indian Reserve Tint
                                 1308  Bndry Indian Reserve Unsurveyed
148  F Poste                     *
148  E Mobile Home Community     *
150  F Village nordique          *
151  F Centre de ski             *
152  E Ranch                     *
153  E Mining Settlement         *
154  E Vac or Seasonal Settlemt  *
155  E Cannery                   4392  Cannery to Scale
                                 4393  Cannery
156  E Indian Village            1306  Boundary Indian Reserve
                                 1307  Boundary Indian Reserve Tint
                                 1308  Bndry Indian Reserve Unsurveyed
157  E Summer Report             *
158  E Former Post Office        *
160  F Village forestier         *
161  F Village naskapi           *
162  F Station de chemin de fer  *
163  F <tablissement amerindien  *
164  F <tablissement noteilier   *
165  F <tablissement saisonnier  *
166  F Faubourg                  *
167  F Parc de maison mobiles    *
168  F Poste de traite           *
169  F Ville miniere             *
170  F Gare ferroviaire          *
171  F Bureau de douane          4364  Customs Symbol - Flag
172  F Arret ferroviaire         *
173  F <tablissement             *
174  F Village                   -1    A Point with no Feature
175  F Village cri               *
;;;Unincorporated Urban          *
200  E Urban Community           -1    A Point with no Feature
201  E Neighbourhood             *
202  E Suburban Community        -1    A Point with no Feature
203  E Indusrtial Park           *
204  E Industrial Area           *
205  F Secteur residentiel       -1    A Point with no Feature
```

```
206  E Subdivision                *
207  F Parc industriel            *
208  E Urban Renewal Area         *
209  E Industrial Siding          *
210  F Quartier urban             *
211  F Faubourg                   *
212  F Ensemble residentiel       *
213  F Quartier                   *
;;;3 Water Features               *
;;;Standing Water                 *
;;;      Surrounded by Land      *
953  E Pond                       5280  Tundra Ponds
                                  5620  Pond
                                  5621  Pond Key
                                  5626  Pond Alkali
954  E Ponds                      5280  Tundra Ponds
                                  5620  Pond
                                  5621  Pond Key
                                  5626  Pond Alkali
982  F Pond                       5280  Tundra Ponds
                                  5620  Pond
                                  5621  Pond Key
                                  5626  Pond Alkali
983  F Ponds                      5280  Tundra Ponds
                                  5620  Pond
                                  5621  Pond Key
                                  5626  Pond Alkali
;;;4 Terrain Features             *
;;;Elevated Shoreline Features    *
1600  E Cape                      *
1601  E Head                      -1    A Point with no Feature
1602  E Isthmus                   *
1603  E Peninsula                 *
1604  E Point                     -1    A Point with no Feature
1605  E Promontory                *
1606  E Headland                  *
1607  E Foreland                  *
1608  E Spit                      *
1609  E Points                    *
1610  E Heights                   *
1611  E Land                      *
1612  E Coast                     *
1613  E Arch                      *
1614  E Bank                      *
1615  E Cliff                     6190  Cliff
1616  F Cote                      *
1617  E Dune                      *
1618  E End                       *
1619  E Hummock                   *
1620  E Neck                      *
1621  E Nose                      *
1622  E Sand Point                *
1623  E Seaside                   *
```

```
1624  F Tete                        *
1625  E Bluff                       *
1626  E Bill                        *
1627  F Pointe                      -1    A Point with no Feature
1628  F Presqu'ile                  *
1629  E Cap                         *
1630  E Caps                        *
1631  F Peninsule                   *
1632  E Bend                        *
1633  F Falaise                     *
1634  F Nez                         *
1634  F Rive                        *
1636  E Coastal Plain               *
1637  E Corner                      *
1638  E Ness                        *
1639  E Delta                       *
1640  F Promontoire                 *
1641  E Cliffs                      *
1642  F Capes                       *
1643  E Bluffs                      *
1644  E Heads                       *
1645  E Hummocks                    *
1646  F Falaises                    *
1647  F Pointes                     *
1648  F Langue de terre             *
1649  F Arches                      *
1650  F Cotes                       *
1651  E Blow Me Down                *
1652  F Berge                       *
1653  F Segment de cote (2)         *
1654  F Hauteurs                    *
1655  F Cap                         *
1656  F Caps                        *
;;;Terrain Surrounded by Water     *
2300  E Island                      5780  Shoreline Island
                                    5783  Shoreline Island in Intermit Lake
                                    5790  Shoreline Island Wooded
                                    5971  Shoreline Island Swampy
                                    5980  Shoreline Island Marshy
2301  E Islands                     5780  Shoreline Island
                                    5783  Shoreline Island in Intermit Lake
                                    5790  Shoreline Island Wooded
                                    5971  Shoreline Island Swampy
                                    5980  Shoreline Island Marshy
2302  F Isle                        5780  Shoreline Island
                                    5783  Shoreline Island in Intermit Lake
                                    5790  Shoreline Island Wooded
                                    5971  Shoreline Island Swampy
                                    5980  Shoreline Island Marshy
2303  F Isles                       5780  Shoreline Island
                                    5783  Shoreline Island in Intermit Lake
                                    5790  Shoreline Island Wooded
                                    5971  Shoreline Island Swampy
```

		5980	Shoreline Island Marshy
2304	E Rock	5460	Rock
2305	E Rocks	*	
2306	F Archipelago	*	
2307	E Islet	*	
2308	E Islets	*	
2309	F Pinnacle	*	
2310	F Pinnacles	*	
2311	F Atoll	*	
2312	F Cap	*	
2313	F Cay	*	
2314	E Key	*	
2315	E Nubble	*	
2316	E Thrum	*	
2317	E Thrumcap	*	
2318	E Monument	1410	Boundary Monument
		1415	Astronomic Monument
		1790	Set Up Monument
		7489	Historic Monument to Scale
		7490	Historic Monument
2319	E Group	*	
2310	F >lette	*	
2321	F >le	*	
2322	F >les	*	
2323	F >lot	*	
2324	F Archipel	*	
2325	F Monument	1410	Boundary Monument
		1415	Astronomic Monument
		1790	Set Up Monument
		7489	Historic Monument to Scale
		7490	Historic Monument
2326	F Carre	*	
2327	F Caye	*	
2328	F Rocher	*	
2329	E Rock Pile	*	
2330	F Roches	*	
2331	E Crab	*	
2332	E Thrums	*	
2333	F Rochers	*	
2334	F >lots	*	
2335	F Cayes	*	
;;;6 Man Made Features		*	
;;;Resource Related		*	
4100	E Forestry Camp	*	
4101	E Generation Station	*	
4102	E Oilwell	3689	Well Oil,gas (250,000)
		7284	Oil Well
4103	E Mine	6270	Mine
4104	E Mines	*	
4105	E Drill Site	*	
4106	E Quarry	6260	Quarry
4107	E Mining Camp	*	
4108	E Mining Property	*	

4109	E Pits	*	
4110	F Carriere	*	
4111	F Camp forestier	*	
4112	F Domaine forestier	*	
4113	F Pepiniere	*	
4114	E Pit	*	
4115	E Pipe Line	*	
4116	E Boom	*	
4117	F Centrale hydro	4398	Electric Facility
4118	E Mine Site	*	
4119	F Station forestier	*	
4120	F Carrieres	*	
4121	F Depot forestier	*	
4122	F Centre <ducatif forest	*	
4123	E Mine	*	
;;;Transportation Related		*	
4300	E Seaplane Base	3070	Seaplane Base
4301	E Portage	2091	Road H Portage
4302	E Road	*	
4303	E Trail	*	
4304	E Concession	1373	Boundary Concession Line
		1374	Boundary Concession Line Tint
		1375	Bndry Concession Line Unserveyed
4305	E Ferry	2220	Ferry
		2225	Ferry Symbol
4306	E Ford	2230	Ford
4307	E Bridge	2181	Bridge (Roadway, Railway)
		2182	Brg (Roadway, Railway) - One Side
		2183	Bridge (Roadway, Railway) - Arch
		2184	Brg (Roadway, Railway) - Covered
		2185	Bridge (Roadway, Railway) - Draw
		2186	Bridge (Roadway, Railway) - Lift
		2187	Bridge (Roadway, Railway) - Swing
		2190	Bridge (Roadway)
		2191	Bridge (Roadway) - Arch
		2192	Bridge (Roadway) - Covered
		2193	Bridge (Roadway) - Draw
		2194	Bridge (Roadway) - Floating
		2195	Bridge (Roadway) - Lift
		2196	Bridge (Roadway) - Swing
		2210	Footbridge
		2211	Bridge (Railway)
		2212	Bridge (Railway) - One Way
		2213	Bridge (Railway) - Arch
		2214	Bridge (Railway) - Covered
		2215	Bridge (Railway) - Draw
		2216	Bridge (Railway) - Lift
		2217	Bridge (Railway) - Swing
		2820	Bridge (Roadway) - One Side
4308	E Causeway	*	
4309	E Landing Strip	*	
4310	E River Crossing	*	
4311	E Lock	*	

4312	E	Locks	8350 Locks to Scale
			8360 Locks
4313	E	Harbour	*
4314	E	Airport	3020 Airport Runways
			3021 Airport Runways and Taxiways
			3022 Airport Taxiways
			3050 Airport
			3683 Airport (250,000)
4315	E	Breakwater	*
4316	E	Canal	*
4317	E	Landing	*
4318	E	Port	*
4319	E	Railway Yard	2425 Railway Yard
			2426 Railway Yard Abandoned
4320	E	Railway Spur	2505 Spur
			2506 Spur Abandoned
4321	E	Spur	2505 Spur
			2506 Spur Abandoned
4322	F	Quai	*
4323	F	Aeroport	3020 Airport Runways
			3021 Airport Runways and Taxiways
			3022 Airport Taxiways
			3050 Airport
			3683 Airport (250,000)
4324	F	Base d'hydravions	*
4325	E	Boat Landing	*
4326	E	Dock	8208 Dock
			8240 Dry Dock
4327	E	Wharf	8210 Wharf
			8490 Wharf to Scale
4328	F	Port de plaisance	*
4329	E	Highway	*
4330	E	Rang	*
4331	E	Steamer Landing	*
4332	E	Maintenance Camp	*
4333	F	Aerodrome	*
4334	F	Arret	*
4335	F	Bassin portuaire	*
4336	F	Pont	2181 Bridge (Roadway, Railway)
			2182 Brg (Roadway, Railway) - One Side
			2183 Bridge (Roadway, Railway) - Arch
			2184 Brg (Roadway, Railway) - Covered
			2185 Bridge (Roadway, Railway) - Draw
			2186 Bridge (Roadway, Railway) - Lift
			2187 Bridge (Roadway, Railway) - Swing
			2190 Bridge (Roadway)
			2191 Bridge (Roadway) - Arch
			2192 Bridge (Roadway) - Covered
			2193 Bridge (Roadway) - Draw
			2194 Bridge (Roadway) - Floating
			2195 Bridge (Roadway) - Lift
			2196 Bridge (Roadway) - Swing
			2210 Footbridge

		2211	Bridge (Railway)
		2212	Bridge (Railway) - One Way
		2213	Bridge (Railway) - Arch
		2214	Bridge (Railway) - Covered
		2215	Bridge (Railway) - Draw
		2216	Bridge (Railway) - Lift
		2217	Bridge (Railway) - Swing
		2820	Bridge (Roadway) - One Side
4337	F Debarcadere	*	
4338	F Route	*	
4339	E Airfield	*	
4340	E Winter Road	*	
4341	E Path	*	
4342	F Portage	2091	Road H Portage
4343	E Airstrip	*	
4344	E Road Intersection	2132	Intersection
		2132	Intersection Traffic Circle
4345	E Corner	*	
4346	E Lift Lock	*	
4347	E Marine	*	
4348	F Terminal	*	
4349	E Road Bend	*	
4350	E Crossing	*	
4351	E Deepwater Terminal	*	
4352	E Road Corner	*	
4353	E Road Cut	*	
4354	E Road Corners	*	
4355	E Road Cut	*	
4356	E Trestle	*	
4357	E Piers	*	
4358	E Railway Cut	*	
4359	E Railway Track	*	
4360	E Trails	*	
4361	F Chemin	*	
4362	E Crib	*	
4363	F Autoroute	*	
4364	F Pier	*	
4365	F Pistle	*	
4366	E Canal	*	
4367	F <changeur	*	
4368	F Embrachement ferroviaire	*	
4369	F Port	*	
4370	F Cote (1)	*	
4371	F Pont couvert	2184	Brg (Roadway, Railway) - Covered
		2214	Bridge (Railway) - Covered
4372	F Ponts	2181	Bridge (Roadway, Railway)
		2182	Brg (Roadway, Railway) - One Side
		2183	Bridge (Roadway, Railway) - Arch
		2184	Brg (Roadway, Railway) - Covered
		2185	Bridge (Roadway, Railway) - Draw
		2186	Bridge (Roadway, Railway) - Lift
		2187	Bridge (Roadway, Railway) - Swing
		2190	Bridge (Roadway)

		2191 Bridge (Roadway) - Arch
		2192 Bridge (Roadway) - Covered
		2193 Bridge (Roadway) - Draw
		2194 Bridge (Roadway) - Floating
		2195 Bridge (Roadway) - Lift
		2196 Bridge (Roadway) - Swing
		2210 Footbridge
		2211 Bridge (Railway)
		2212 Bridge (Railway) - One Way
		2213 Bridge (Railway) - Arch
		2214 Bridge (Railway) - Covered
		2215 Bridge (Railway) - Draw
		2216 Bridge (Railway) - Lift
		2217 Bridge (Railway) - Swing
		2820 Bridge (Roadway) - One Side
4373	E Tunnel	2178 Tunnel Railway
		2179 Tunnel Roadway
		2180 Tunnel
4374	F Hyrobase	*
4375	F Pont naturel	*
4376	E Marina	4537 Marina to Scale
		4538 Marina
4377	F Portage	2091 Road H Portage
4378	F Carrefour	2132 Intersection
		2133 Intersection Traffic Circle

The following abbreviations have been used:

Bndry = Boundary

Brg = Bridge

forest = forestier

hydro = hydroelectrique

Inc = Incorporated

Metro = Metropolitan

Pop = Populated

Settlemt = Settlement

Vac = Vacated

Appendix B. The Possible Matches for Bliss Island

```
2300 Bliss Island                          450100 665000 670722 4986868
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,665347,4985514,665334,4985520,665259,4985495,665191,4985488,
    665160,4985462,665129,4985412,665142,4985362,665124,4985306,665124,
    4985256,665149,4985250,665212,4985275,665324,4985295,665385,4985390,
    665423,4985396,665467,4985340,665469,4985184,665438,4985121,665432,
    4985084,665470,4985059,665532,4985085,665601,4985148,665725,4985205,
    665762,4985275,665773,4985412,665791,4985469,665784,4985531,665709,
    4985555,665671,4985555,665609,4985511,665566,4985466,665529,4985453,
    665453,4985490,665372,4985502,665347,4985514
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,671681,4988358,671669,4988390,671625,4988389,671594,4988351,
    671601,4988320,671626,4988308,671663,4988327,671681,4988358
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,670915,4988676,670815,4988675,670772,4988656,670741,4988605,
    670754,4988568,670836,4988506,670849,4988450,670837,4988400,670750,
    4988293,670720,4988211,670646,4988129,670660,4988023,670722,4988024,
    670790,4988075,670871,4988100,670921,4988132,671026,4988258,671101,
    4988296,671156,4988334,671187,4988429,671243,4988442,671293,4988467,
    671548,4988551,671579,4988607,671560,4988645,671515,4988701,671471,
    4988744,671421,4988750,671420,4988793,671464,4988844,671461,4989094,
    671429,4989125,671373,4989112,671280,4989073,671155,4989060,670962,
    4988951,670981,4988927,671019,4988902,670982,4988852,670964,4988808,
    670977,4988783,671015,4988739,671041,4988646,671016,4988621,670991,
    4988620,670915,4988676
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,669616,4988550,669590,4988600,669546,4988631,669496,4988624,
    669391,4988517,669392,4988492,669461,4988455,669591,4988525,669616,
    4988550
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,669331,4987123,669332,4987073,669345,4987010,669395,4987004,
    669488,4987093,669581,4987106,669619,4987119,669668,4987201,669693,
    4987226,669755,4987258,669805,4987271,669861,4987272,669912,4987222,
    669912,4987147,669900,4987084,669976,4987029,669989,4986985,670014,
```

```
    4986948,670115,4986949,670183,4987006,670270,4987051,670288,4987126,
    670344,4987145,670363,4987133,670444,4987127,670525,4987147,670593,
    4987198,670661,4987273,670748,4987349,670797,4987487,670971,4987576,
    671094,4987703,671100,4987771,671137,4987815,671161,4987866,671149,
    4987903,671117,4987934,671092,4987940,671079,4987952,671053,4988065,
    671040,4988077,670991,4988045,670891,4987969,670824,4987862,670743,
    4987849,670706,4987811,670687,4987780,670637,4987767,670582,4987716,
    670520,4987628,670414,4987558,670353,4987495,670278,4987463,670210,
    4987388,670192,4987350,670142,4987337,670066,4987349,670004,4987329,
    669966,4987348,669947,4987385,670014,4987536,670102,4987568,670126,
    4987637,670188,4987700,670174,4987800,670124,4987849,670079,4987955,
    670035,4987998,669916,4988016,669860,4987990,669811,4987890,669761,
    4987846,669680,4987820,669618,4987769,669563,4987700,669476,4987630,
    669415,4987505,669415,4987448,669428,4987411,669397,4987361,669354,
    4987317,669356,4987173,669331,4987123
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,669472,4989168,669447,4989143,669391,4989105,669348,4989017,
    669299,4988985,669281,4988903,669144,4988852,669125,4988827,669108,
    4988720,669071,4988639,669048,4988476,668998,4988419,668917,4988394,
    668861,4988355,668824,4988355,668805,4988392,668810,4988461,668847,
    4988512,668897,4988562,668933,4988675,668926,4988725,668851,4988724,
    668789,4988686,668758,4988723,668775,4988880,668830,4988943,668911,
    4989019,668973,4989050,669028,4989163,669083,4989233,669077,4989276,
    669033,4989289,668853,4989162,668784,4989199,668758,4989230,668758,
    4989298,668795,4989336,668833,4989305,668895,4989300,668982,4989369,
    669038,4989439,669049,4989501,669099,4989570,669186,4989621,669460,
    4989649,669542,4989669,669615,4989819,669683,4989864,669813,4990046,
    669818,4990090,669849,4990172,669830,4990196,669755,4990196,669699,
    4990176,669630,4990169,669586,4990144,669574,4990094,669556,4990056,
    669538,4990044,669412,4990074,669305,4990123,669224,4990115,669174,
    4990077,669187,4990040,669257,4989991,669344,4989985,669363,4989961,
    669333,4989910,669295,4989897,669245,4989897,669165,4989796,669003,
    4989744,668767,4989636,668729,4989654,668703,4989692,668703,4989760,
    668758,4989855,668869,4989974,668893,4990043,668955,4990138,669080,
    4990145,669117,4990158,669148,4990190,669197,4990278,669203,4990359,
    669146,4990390,669121,4990414,669139,4990477,669114,4990508,669076,
    4990514,669045,4990539,669019,4990632,668992,4990788,668941,4990869,
    668953,4990913,669009,4990976,669020,4991063,669057,4991158,669068,
    4991270,669123,4991346
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,669123,4991346,669290,4991522,669346,4991611,669363,4991711,
    669400,4991774,669443,4991868,669429,4991980,669516,4992087,669546,
    4992131,669577,4992194,669564,4992294,669562,4992400,669593,4992432,
    669625,4992395,669663,4992364,669712,4992395,669737,4992452,669768,
    4992502,669817,4992534,669905,4992554,669942,4992591,669998,4992630,
    670029,4992630,670111,4992593,670149,4992481,670181,4992475,670212,
    4992500,670236,4992551,670273,4992589,670305,4992564,670318,4992508,
```

```
    670300,4992426,670313,4992364,670376,4992346,670395,4992302,670402,
    4992246,670384,4992183,670385,4992108,670398,4991990,670437,4991921,
    670469,4991790,670471,4991634,670380,4991321,670369,4991227,670326,
    4991114,670315,4990983,670254,4990876,670243,4990751,670199,4990744,
    670149,4990718,670176,4990594,670338,4990658,670387,4990696,670399,
    4990802,670417,4990827,670454,4990840,670530,4990778,670611,4990848,
    670654,4990848,670680,4990824,670705,4990749,670720,4990555,670689,
    4990524,670639,4990492,670596,4990423,670559,4990391,670472,4990384,
    670441,4990359,670404,4990258,670349,4990177,670338,4990076,670281,
    4990057,670251,4990007,670239,4989951,670214,4989925,670145,4989925,
    670045,4989949,669876,4989959,669833,4989915,669790,4989827,669772,
    4989771,669686,4989614,669612,4989538,669569,4989456,669513,4989375,
    669502,4989243,669472,4989168
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,668729,4988492,668667,4988429,668629,4988416,668598,4988440,
    668604,4988465,668660,4988522,668697,4988535,668729,4988492
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,668521,4988058,668502,4988052,668384,4987976,668366,4987926,
    668385,4987907,668504,4987908,668553,4987921,668578,4987971,668540,
    4988052,668521,4988058
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,668517,4988365,668480,4988364,668436,4988345,668418,4988295,
    668369,4988238,668363,4988213,668376,4988176,668407,4988170,668519,
    4988265,668568,4988321,668561,4988353,668517,4988365
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,668511,4988977,668449,4988895,668450,4988801,668482,4988771,
    668519,4988790,668562,4988853,668587,4988922,668573,4988990,668517,
    4988983,668511,4988977
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,668125,4987667,668075,4987623,668063,4987579,668076,4987554,
    668164,4987499,668209,4987436,668309,4987437,668371,4987463,668421,
    4987451,668458,4987476,668501,4987546,668495,4987571,668457,4987614,
    668449,4987758,668405,4987795,668330,4987794,668280,4987768,668199,
    4987768,668162,4987699,668125,4987667
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
```

```
LST/OP,668121,4987973,668041,4987929,668010,4987897,668041,4987866,
    668229,4987899,668209,4987949,668171,4987999,668121,4987973
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,665202,4989126,665221,4989107,665227,4989057,665203,4989026,
    665097,4989024,665046,4989043,665027,4989068,665033,4989099,665095,
    4989124,665126,4989150,665176,4989157,665202,4989126
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,664936,4986479,664874,4986403,664825,4986309,664839,4986234,
    664827,4986197,664796,4986146,664809,4986128,664909,4986066,664934,
    4986060,665034,4986080,665103,4986068,665153,4986094,665239,4986219,
    665282,4986264,665350,4986371,665406,4986421,665449,4986428,665512,
    4986397,665580,4986448,665617,4986498,665611,4986536,665510,4986603,
    665459,4986690,665403,4986684,665359,4986652,665341,4986602,665310,
    4986570,665261,4986532,665237,4986438,665168,4986381,665112,4986368,
    665087,4986393,665080,4986437,664986,4986504,664948,4986492,664936,
    4986479
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,671628,4989252,671622,4989277,671577,4989314,671502,4989325,
    671440,4989294,671466,4989244,671554,4989201,671597,4989208,671628,
    4989252
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,671299,4990192,671219,4990085,671145,4990009,671076,4989984,
    671045,4989958,671046,4989871,671022,4989821,670979,4989758,670991,
    4989745,671023,4989740,671135,4989759,671210,4989785,671234,4989817,
    671190,4989885,671190,4989922,671245,4989967,671476,4989988,671520,
    4990007,671638,4990127,671612,4990152,671562,4990151,671537,4990176,
    671518,4990238,671473,4990269,671411,4990268,671299,4990192
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,670776,4989406,670769,4989449,670707,4989461,670650,4989486,
    670606,4989479,670563,4989441,670570,4989385,670633,4989354,670689,
    4989342,670758,4989381,670776,4989406
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,670504,4989703,670498,4989666,670536,4989628,670574,4989623,
    670630,4989642,670691,4989749,670685,4989786,670653,4989798,670603,
    4989779,670504,4989703
```

```
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,669303,4989766,669266,4989747,669235,4989703,669247,4989691,
    669304,4989691,669409,4989742,669440,4989768,669434,4989805,669409,
    4989817,669303,4989766
ASC/5790
LAC/LT=0
LAC/LS=0
LAC/LC=74
LST/OP,666107,4989260,666020,4989215,666002,4989171,666027,4989140,
    666139,4989166,666226,4989211,666212,4989379,666174,4989410,666143,
    4989404,666125,4989360,666119,4989266,666107,4989260
ASC/5790
LAC/LC=74
LST/OP,680000,4985309,679993,4985309,679568,4985293,679368,4985291,
    678286,4985249,677261,4985226,677024,4985211,676523,4985206,676242,
    4985191,676017,4985189,675792,4985174,675461,4985171,675067,4985154,
    674823,4985152,674498,4985136,673936,4985124,673611,4985109,673242,
    4985105,673036,4985090,672504,4985079,671648,4985039,671235,4985041,
    670885,4985025,670310,4985013,670085,4984999,669698,4984995,669360,
    4984979,668673,4984966,668479,4984951,668116,4984948,667585,4984924,
    667097,4984919,666022,4984871,665729,4984874,665360,4984858,665322,
    4984876,665272,4984920,665196,4984938,665122,4984918,665041,4984842,
    664635,4984845,664628,4984907,664677,4984995,664726,4985033,664758,
    4985046
ASC/5790
LAC/LC=74
LST/OP,668001,4990734,667998,4990660,668000,4990585,667994,4990522,
    667889,4990440,667796,4990314,667773,4990145,667755,4990082,667629,
    4990150,667586,4990143,667555,4990130,667505,4990123,667467,4990111,
    667411,4990129,667348,4990103,667237,4990033,667175,4989964,667163,
    4989883,667120,4989838,667020,4989869,666963,4989856,666889,4989805,
    666827,4989735,666734,4989703,666697,4989653,666659,4989628,666622,
    4989627,666591,4989596,666598,4989533,666580,4989439,666549,4989426,
    666436,4989469,666368,4989425,666293,4989430,666280,4989474,666316,
    4989593,666309,4989637,666271,4989686,666152,4989729,666089,4989778,
    666063,4989815,666001,4989840,666000,4989877,666025,4989909,665974,
    4989964,665918,4990008,665843,4990007,665811,4989994,665786,4990019,
    665785,4990094,665816,4990194,665853,4990207,665898,4990095,665942,
    4990077,665992,4990083,666004,4990109,665953,4990164,665941,4990202,
    666164,4990379,666157,4990391,666120,4990410,666001,4990409,665926,
    4990383,665889,4990389,665838,4990426,665775,4990525,665774,4990556,
    665823,4990663,665829,4990726,665878,4990770,665934,4990808,665952,
    4990852,665952,4990921,665914,4990933,665796,4990888,665758,4990925,
    665739,4990968,665751,4991006,665782,4991038,665787,4991119,665812,
    4991144,665849,4991163,665823,4991251,665835,4991294,665866,4991345,
    665853,4991382,665815,4991401,665715,4991406,665696,4991424,665683,
    4991468,665671,4991480,665614,4991480,665565,4991411,665477,4991410,
    665465,4991385,665472,4991353,665510,4991322,665517,4991241,665586,
    4991229,665586,4991186
```

Appendix C. The Match Results for NTS21G

A few weeks of this thesis were spent doing a test matching the point names to their associated features for the Fredericton, New Brunswick area map sheet NTS21G. A summary of the results is shown below.

The results for the NTS21G map sheet were :

Names not matched / not placed:	124
Names omitted from the map:	149
Names placed on map but no feature:	16
Names placed on the map with a feature:	20

The total number of names:	309

The summary has the point names split into four categories. The first category names were discarded without a matching feature. Many of these names correspond to small islands. The next category are names with matching features, but which did not appear on the 1:250 000 map sheet. Again some of these were small islands and some were pond names. The third category of names were displayed on the map even though no feature was shown. This seems to be the names associated with small villages. The final category of names was the name on the map with matching features on the map; only 20 point names fell in this category. The file produced while doing this experiment is shown below.

```
;;;   Points on the NTS21G 1:250 000 map
;;;
;;;   O - name has been omitted on the map
;;;   N - no matching feature found on map
;;;   M - match found and name is to be placed on the map
;;;   P - no matching feature but name is on the map
;;;
O     1 Adam Island              450100 665400
```

N	2	Alder Island	452100	665300
O	3	Anderson Island	451300	662000
N	4	Andys Pond	450400	664300
O	5	Apple Island	455500	661600
N	6	Ballantyne Pond	455500	660500
O	7	Bar Island	450300	665000
M	8	Barkers Island	455800	663600
P	9	Barkers Point	455700	663700
O	10	Barnes Island	450000	665400
O	11	Basley Island	453900	663400
O	12	Beams Island	452200	660800
O	13	Beans Island	450000	665600
N	14	Bears Bath Tub	454000	673200
N	15	Becketts Pond	453600	661200
N	16	Bells Island	453600	672200
N	17	Ben Beachs Island	453600	673000
M	18	Big Island	451000	664700
O	19	Big Island	453700	672900
N	20	Big Island	454400	671200
N	21	Billy Island	454100	674800
N	22	Birch Island	453300	670200
O	23	Bird Island	450900	665800
N	24	Black Island	453800	673400
N	25	Blacks Harbour	450300	664700
M	26	Bliss Island	450100	665000
N	27	Bodkin Island	454300	671200
N	28	Bradt Island	452000	664700
O	29	Breakwater Island	455800	663600
M	30	Brothers, The	451800	660700
O	31	Burnt Island	451900	660600
N	32	Burnt Island	454900	673200
O	33	Burpee Island	455800	664800
O	34	Butler Islands	450800	672100
N	35	Butterfly Island	454400	671200
N	36	Camp Jersey Island	453100	670100
N	36	Camp Jersey Island	453100	670100
N	37	Cannonball Island	450900	664700
N	38	Canterbury	455300	672900
N	39	Carr Island	454900	673100
M	40	Catons Island	452800	660700
N	41	Cedar Islands	453000	672900
N	42	Cedar Islands	454100	670900
N	43	Clarks Pond	455600	660800
O	44	Clements Island	455800	664200
N	45	Club Island	454900	673200
O	46	Cochranes Island	450400	664700
N	47	Collicott Pond	455200	672100
N	48	Colwells Wharf	454300	660500
O	49	Cooks Island	450300	665400
O	50	Coreys Island	454800	660600
N	51	Cork Station	454100	665500
N	52	Coxs Islands	451800	665000
O	53	Crocker Island	451100	671600

O	54	Crow Island	450100	665600
O	55	Crow Island	450700	662200
O	56	Crow Island	451600	660500
O	57	Crow Island	450300	665300
N	58	Cummings Island	453700	672900
O	59	Currie Island	455900	664500
N	60	Curries Island	453900	663700
N	61	Curries Pond	454000	663700
N	62	Daley Ponds	451400	663300
M	63	Deer Island	450000	665800
N	64	Denton Pond	455500	660900
N	65	Dibblees Island	454900	673300
O	66	Dicks Island	450800	670000
O	67	Dog Islands	452200	672600
N	68	Doherty Pond	455500	672100
O	69	Douglas Island	450300	665200
N	70	Doyle Pond	455200	672000
N	71	Duck Island	451200	664800
N	72	Duck Pond	453800	672400
O	73	Dunphy Island	455800	664300
O	74	Eagle Island	450100	665200
N	75	East Riverside-Kingshurst	452200	660000
O	76	English Island	450100	665700
N	77	Estabrooks Pond	455400	661000
N	78	Estys Island	453600	672200
N	79	Fairvale	452500	660000
O	80	False Island	451600	660500
O	81	Fish Island	450100	665600
O	82	Flea Island	450200	665000
N	83	Flume Islands	452600	670100
N	84	Foss Pond	454800	663200
O	85	Fox Island	450300	665000
P	86	Fredericton Junction	454000	663700
O	87	French Island	455500	661800
N	88	Frog Island	455100	674000
M	89	Frye Island	450300	665100
O	90	Fulton Island	455400	661400
P	91	Gagetown	454700	660900
O	92	Gagetown Island	454700	660800
N	93	Gardner Pond	451700	665900
O	94	Gilbert Island	455300	661900
O	95	Goat Island	451600	660500
O	96	Goat Island	451900	660700
O	97	Gooseberry Island	450800	661600
N	98	Grand Bay	451800	661200
O	99	Grassy Island	451000	664800
O	100	Grassy Island	453100	660400
N	101	Grassy Islands	452200	665300
N	102	Grassy Islands	452500	672800
N	103	Gravel Island	452800	672900
O	104	Green Island	450200	665000
N	105	Green Island	452500	672700
O	106	Grimross Island	454900	661000

O	107	Gull Island	455700	661700
N	108	Gunters Wharf	455000	661200
N	109	Halfmoon Pond	454100	663500
N	110	Halfway Island	454900	673200
O	111	Harbour Island	453700	665800
N	112	Hardwood Island	450100	665600
N	113	Hardwood Island	453600	673300
M	114	Hardwood Island	450700	670000
O	115	Hardy Island	451200	660900
O	116	Harrison Island	455400	661600
N	117	Hartts Island	453700	665100
O	118	Hartts Island	455800	664500
P	119	Harvey (Harvey Station P.	454400	670000
P	120	Harvey Station (Harvey Vi	454400	670000
N	121	Hatch Island	452700	665300
O	122	Head Island	450800	662800
O	123	Hills Island	450300	664900
O	124	Hog Island	450200	665200
O	125	Hog Island	454200	660300
N	126	Hog Island	452700	672900
O	127	Hog Island	453300	660100
O	128	Hog Island	450800	665800
N	129	Holden Pond	454900	663100
N	130	Horse Island	452200	672600
N	131	Horse Island	452500	664000
O	132	Hospital Island	450700	670100
O	133	Howards Island	450200	665000
O	134	Hoyt Island	450300	665500
O	135	Hoyt Nub	450300	665500
O	136	Huestis Island	454700	660600
O	137	Hunters Island	455600	661000
O	138	Indian Island	451800	660700
O	139	Indian Island	455600	661800
N	140	Indian Pond	452100	671600
N	141	Jackknife Islands	453400	670600
O	142	Jail Island	450400	665000
N	143	Jakes Pond	454400	663400
O	144	Jameson Island	450200	665600
N	145	Jerry Pond	450900	664200
O	146	Jewett Island	455800	664200
N	147	Kelly Island	453600	665900
M	148	Kennebecasis Island	451900	660800
O	149	Keswick Island	455900	664900
O	150	Killaboy Island	454300	660500
O	151	King Brook Islands	452100	672600
N	152	Knight Pond	450900	664200
N	153	Layden Pond	460000	660200
N	154	Lindsay Island	453700	672900
N	155	Little Indian Island	453600	672700
O	156	Little Island	450100	665500
O	157	Long Island	450900	665800
M	158	Long Island	454000	660500
N	159	Long Island	453700	672800

M	160	Long Island	452300	660200
N	161	Long Island	451200	664800
N	162	Long Island	454400	671200
N	163	Long Pond	451800	661800
N	164	Long Pond	455400	672100
O	165	Lower Musquash Island	454200	660500
O	166	Lower Shores Island	455800	664900
N	167	Luffs Island	453700	672300
O	168	MacDougalls Island	450800	665500
N	169	MacGougans Island	454000	663800
N	170	MacKenzies Island	453800	672900
O	171	Macs Island	450300	665600
O	172	Man of War Island	450200	665100
M	173	Manawagonish Island	451200	660600
M	174	Marshalls Island	455200	661100
O	175	Mather Island	452500	660100
N	176	Matts Island	453900	664900
P	177	McAdam	453600	672000
N	178	McAdam Pond	453500	672000
O	179	McAllisters Island	454800	661100
N	180	McCulloughs Pond	451300	670300
O	181	McGibbon Island	455800	664500
O	182	McGraws Island	450300	665500
N	183	McLaughlin Pond	454400	663200
O	184	McVicar Island	451200	671100
P	185	Meductic	460000	672900
O	186	Merrithews Island	455700	665000
O	187	Middle Island	451600	660500
O	188	Middle Island	455300	662500
N	189	Mill Island	454900	673100
N	190	Mill Pond	452700	660000
N	191	Ministers Face	452400	660200
P	192	Ministers Island	450600	670200
O	193	Mink Island	450000	665600
O	194	Mink Island	450200	664900
O	195	Mink Island	450300	665200
O	196	Mitchells Island	455800	664900
O	197	Mohawk Island	450200	665400
N	198	Moon Pond	455300	672100
O	199	Moose Island	450300	664600
O	200	Morans Island	450200	665200
N	201	Morrow Pond	454500	663300
O	202	Mosquito Island	451200	672600
O	203	Motts Wharf	454400	660100
P	204	Mouth of Keswick (Keswick	460000	665000
O	205	Mowat Island	450000	665400
N	206	Munson Island	451200	664000
O	207	Murray Island	455800	664300
O	208	Musquash Island	451000	661400
N	209	Nackawic	460000	671500
N	210	Nan Island	454100	674800
M	211	Navy Island	450400	670300
O	212	Nevers Island	454700	660600

O	213	Nevers Island	455800	664400
O	214	New Ireland	450200	665600
O	215	New River Island	450700	663300
N	216	Northcott Island	454900	673200
N	217	Nova Scotia Island	454300	671200
O	218	Nub Island	450300	665500
O	219	Nubble Island	450000	665400
N	220	Nubbles, The	453600	673200
N	221	O'Malleys Island	453600	672700
O	222	Oak Island	454900	661000
O	223	Odell Island	450800	670500
P	224	Oromocto	455100	662900
M	225	Oromocto Island	455200	662900
O	226	Oven Head Island	450900	665700
O	227	Ox Island	455200	661700
O	228	Park Island	450500	664800
O	229	Park Islands	450500	664800
M	230	Parker Island	450200	665500
N	231	Parker Island	453600	673200
O	232	Parsnip Island	455800	664500
M	233	Partridge Island	450100	665600
O	234	Partridge Island	451400	660300
O	235	Peat Island	450500	664700
M	236	Pendleton Island	450200	665700
O	237	Penn Island	450600	663700
O	238	Pig Island	453400	660100
N	239	Pine Island	453200	670200
N	240	Pine Island	454600	674800
O	241	Pines, Isle of	453000	660500
O	242	Pitt Island	455800	664500
N	243	Pleasant Pond	453900	672200
O	244	Pocologan Island	450600	663400
O	245	Princes Island	455700	661600
O	246	Ram Island	452100	660800
O	247	Ram Island	455500	661700
O	248	Ram Island	455200	661600
O	249	Ram Island	455700	663700
O	250	Reach Island	455500	662100
N	251	Ready Pond	451800	661600
N	252	Red Ledges	451200	664700
P	253	Renforth	452200	660100
O	254	Rickets Island	451200	670900
O	255	Ring Island	455400	661200
N	256	Robinsons Pond	452300	664000
O	257	Rocky Island	452900	660600
O	258	Ross Island	455800	664200
N	259	Rothesay	452300	660000
N	260	Round Pond	451700	671800
N	261	Round Pond	451100	670100
O	262	Rowans Island	451600	660500
O	263	Rush Island	453000	660500
P	264	Saint Andrews	450500	670300
O	265	Salkeld Islands	450600	663000

N	266	Sam Orr Pond	451000	670300
O	267	Savage Island	455900	664600
N	268	Scovils Island	454400	671200
N	269	Seeley Pond	453900	663400
N	270	Ship Island	453700	670000
O	271	Silver Island	450700	665100
O	272	Simpsons Island	450000	665500
N	273	Slugundy Ponds	455200	671100
O	274	Spectacle Island	450000	670000
N	275	Splan Pond	451500	672000
O	276	Spoon Island	451300	671100
O	277	Spoon Island	453700	660400
O	278	Spruce Island	451000	664700
N	279	Spruce Island	453200	670200
O	280	Spruce Island	450100	665200
P	281	St. George	450800	665000
P	282	St. Stephen	451200	671700
O	283	Star Island	454100	672900
O	284	Stillwater Island	451300	662000
M	285	Sugar Island	455900	664800
O	286	Taylors Island	451300	660800
O	287	Thatch Island	455000	660900
O	288	Thatch Island	455100	662900
N	289	Thorne Pond	455800	660500
N	290	Three Tree Creek Pond	454100	663500
O	291	Thumb Island	450300	665500
N	292	Todds Island	453600	673000
P	293	Tracy	454100	664100
O	294	Tub Island	450500	664800
N	295	Turnover Island	451900	665100
M	296	Upper Musquash Island	454400	660600
O	297	Upper Shores Island	455800	664900
N	298	Varney Island	453400	672500
O	299	Vernon Island	451000	665000
N	300	Waasis Pond	454900	663200
O	301	Walkers Island	455500	662000
P	302	Westfield	452000	661400
O	303	White Head Island	450100	665200
N	304	White Pond	454700	663300
N	305	Whites Island	453700	672200
N	306	Williams Island	453700	672200
N	307	Wilson Island	454900	673100
N	308	Wingdam Island	453200	672600
N	309	Works Island	453100	672500

Appendix D. The Initial Knowledge-Base Attempt

These following code is the initial attempt at the knowledge-base for automated name placement. This code is has not been completed; it is just in its infancy.

```
; Lisa Mullin
;
; Rule-based system for name placement
;  Initial Attempt
;
; August 24, 1992.
;
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Definition of global variables
;

(defglobal ?*file* = (open "x:\\art\\some.nms" "r"))

; Definition of the slots to be used
;

(defschema name "name being placed"
        (instance-of slot)
        (cardinality multiple))

(defschema type "type of name being placed"
        (instance-of slot))

(defschema location "location of name being placed"
        (instance-of slot)
        (cardinality multiple))

(defschema overlaps-grid-line "boolean for overlap with grid line"
        (instance-of slot))

(defschema crowded "boolean for crowded condition"
        (instance-of slot))

(defschema overlaps-map-feature "boolean for overlap with a map feature"
        (instance-of slot))

(defschema overlap-existing-name "boolean for overlap with an name"
        (instance-of slot))

(defschema current-placement-position "the current position"
        (instance-of slot))
```

```
(defschema path-to-qt-node "the path to the quadtree node"
       (instance-of slot))

(defschema positions "the 8 possible positions"
       (instance-of slot))

; The Name Placement Schema ;

(defschema name-placement "the general name placement structure"
       (name)
       (type)
       (location)
       (overlaps-grip-line)
       (crowded)
       (overlaps-map-feature)
       (overlaps -existing -name)
       (current-placement-position)
       (path-to-qt-node))

;
; Read the names into the schemas
;

;;;(defrule read-input-names
;;;     (declare(salience 1))
;;;=>
;;;     (while (not (feof ?*file*)) do
;;;          (bind ?line (read-line ?*file*))
;;;          (if(not (equal ?line *EOF* )) then
;;;                (bind ?junk (string-substring ?line 1 1)
;;;                (bind ?junk (string-substring ?line 2 4)
;;;                (bind ?junk (string-to-integer ?junk)
;;;
;;;     (assert
;;;
;;;(close ?*file*))
;;;

;;;  test a new names


(defschema Fredericton
       (instance-of name-placement
       (name Fredericton)
       (type 135)
       (location 100 20))

(defschema Small-Island
       (instance-of name-placement)
       (name Small Island)
       (type 1221)
       (crowded t)
```

```
        (location 93 275))

(defschema Gaspe
        (instance-of name-placement
        (name Gaspe)
        (type 111)
        (location 100 2000))

; Rule for initial name placement ;

(defrule place-name
        (schema ?name
             (instance-of name-placement)
             (current-placement-position 0)

=>
        (printout t "Names to be placed "?name t)

;
; Rule for crowded islands
;

(defrule crowded-Island
        (schema ?name
             (instance-of name-placement)
             (name $? Island)
             (crowded t))
=>

      (printout t "A crowded Island" ?name t))
```

Appendix E. The Construct Algorithm Source Code

The following pages of this appendix contain the source code used to construct a quadtree with objects. The modules included are :

Of these modules, the ones which are used for memory management are LSpace, QSpace, PAlloc, and PFree. The first two routines , LinkSpaceAllocate and QuadSpaceAllocate are used to allocate a large section of 32 bit memory. Although these two are similar, they were done separately for simplicity. The routine PointerAlloc is used to convert a 32 bit handle into a 16 bit address in the current segment of memory. This implies that the memory will be paged at this point in order to have the desired memory locations in the current segment. PointerFree id used to release the 16 bit pointer to the current segment, thus allowing the memory manager to page the memory when necessary.

Only the main portion of the code is given here. The functions for interacting with windows, opening files and reading data are not included for the sake of brevity.

```
/* ------------------------------------------------------------------------
 *
 *   Module: CONSTR.c
 *   System: PS/2 Model 70
 *           MicroSoft QuickC
 *
 *   Purpose: Module to build a quadtree with objects.
 *
 *   Programmer: Lisa Mullin
 *   Date : Sept 26, 1992.
 *   Detail:  This module will build a quadtree with objects.  The algorithm
 *            used is a modified one which prunes the tree from the top
 *            by using a bounding box for the feature.  This routine
 *            makes use of 32 bit pointers.
 *
 *
 * ------------------------------------------------------------------------ */
/* ---------------------------------------- Include Files ----------- */
#include "windows.h"
#include <winmem32.h>
#include <stdio.h>
#include "quad.h"
#include "quadtree.h"
/* ---------------------------------------- Module Definitions ------ */

          /* none */

/* ---------------------------------------- Module Macros ----------- */

          /* none */

/* ---------------------------------------- Imported Variables ------ */

extern  int   debug, STOP;

extern  long int  nodes, test, wn, bn, gn, apow[maxpow];
extern  long int  norlc;
extern  long int  CurrentIndex, CurrentIndexBlack;
extern  long int  xmin, xmax, ymin, ymax;

extern  rlength rlc[maxrlc];

extern  FILE  *outfile;
extern  WORD  wSel, wold, CurrentwSel;
extern  WORD  wSelBlack, CurrnetwSelBlack;
extern  DWORD dwBuff, OldOffset, QuadSize;

/* ---------------------------------------- Imported Functions ------ */

extern void  AddBlack(pair blk, int ASC_code);          /* in AddBlack.c  */
extern void  FreeSpace(WORD wSel);                      /* in Free.c      */
extern int   IMAGE(long int x, long int y);             /* in Image.c     */
extern DWORD PointerAlloc(WORD wSel, DWORD dwOffset, DWORD Size);
```

```
                                        /* in PAlloc.c    */
extern void  PointerFree(WORD wSel,DWORD dwOffset);    /* in PFree.c     */
extern WORD  QuadSpaceAllocate();                      /* in QSpace.c    */

/* ---------------------------------------- Exported Variables ------ */

          /* none */

/* ---------------------------------------- Local Typedefs ---------- */

          /* none */

/* ---------------------------------------- Local Global Variables -- */

          /* none */

/* ---------------------------------------- Local Functions --------- */

          /* none */

/* ---------------------------------------- Exported Functions ------ */

          /* none */
/* ------------------------------------------------------------------- */

pair CONSTRUCT(n,x,y, ASC_code)
int        n, ASC_code;
long int   x,y;
{

  pair     par,p[4], q,blk;
  qtree    *quadtree, *r, *child;

  int      imag,level,i;

  char     szString2[95];
  char     szString3[45];

  WORD     wSelChild;
  DWORD    dwOffset, dwOffsetChild;

  wsprintf(szString3,"Construct");

  test++;

  if (n == 0)                                  /* process the node */
  {
    par.wSel = 0;
    par.dwOffset = 0;
    imag = IMAGE(x-1,y-1);
    if ( imag == 0)
       {
       par.colour = 'W';
```

```
    }
  else
    {
    par.colour = 'B';
    }
}
else
{       /* n != 0 */
    level = n-1;
                          /* initialize the children to null */
     p[0].wSel = 0;
     p[0].dwOffset = 0;
     p[1].wSel = 0;
     p[1].dwOffset = 0;
     p[2].wSel = 0;
     p[2].dwOffset = 0;
     p[3].wSel = 0;
     p[3].dwOffset = 0;


    if ( (xmin <= x-apow[level]) && (ymax >= y-apow[level]) )
     p[0] = CONSTRUCT(level,x-apow[level],y,ASC_code);
    else
     p[0].colour = 'W';

    if ( (xmax >= x-apow[level]) && (ymax >= y-apow[level]) )
     p[1] = CONSTRUCT(level,x,y,ASC_code);
    else
     p[1].colour = 'W';

    if ( (xmin <= x-apow[level]) && (ymin <= y-apow[level]) )
     p[2] = CONSTRUCT(level,x-apow[level],y-apow[level],ASC_code);
    else
     p[2].colour = 'W';

    if ( (xmax >= x-apow[level]) && (ymin <= y-apow[level]) )
     p[3] = CONSTRUCT(level,x,y-apow[level],ASC_code);
    else
     p[3].colour = 'W';

    if ( debug == 1)
    {
     fprintf(outfile,"The NW  %d %ld \n",p[0].wSel,p[0].dwOffset);
     fprintf(outfile,"The NE  %d %ld \n",p[1].wSel,p[1].dwOffset);
     fprintf(outfile,"The SW  %d %ld \n",p[2].wSel,p[2].dwOffset);
     fprintf(outfile,"The SE  %d %ld \n",p[3].wSel,p[3].dwOffset);
    }
    if ( (p[0].colour != '(') &&
       (p[0].colour == p[1].colour) &&
       (p[1].colour == p[2].colour) &&
       (p[2].colour == p[3].colour) )
    {
    par.wSel =  p[0].wSel;           /* all children are the same */
```

```
par.dwOffset = p[0].dwOffset;
par.colour = p[0].colour;
}
else                                /* create a non-terminal GRAY node */
{
wold = wSel;
OldOffset = dwOffset;
CurrentIndex = CurrentIndex + 1;
if (CurrentIndex < NODES_PER_ALLOC)
  dwOffset = CurrentIndex * QuadSize;
else
{
             /* allocate another  */
   wsprintf(szString2,"Allocate another now\n %ld",CurrentIndex);
   MessageBox(NULL, szString2, szString3, MB_OK);
   wold = wSel;
   OldOffset = dwOffset;
   wSel = QuadSpaceAllocate();
   CurrentwSel = wSel;
   CurrentIndex = 0;
   dwOffset = 0;
}
dwBuff = PointerAlloc(wSel,dwOffset,QuadSize);
if (STOP == 1)
{
  wsprintf(szString2,"Allocate problem 18\n %d %ld",wSel,dwOffset);
  MessageBox(NULL, szString2, szString3, MB_OK);
}
quadtree = (QTREE) dwBuff;
q.wSel = wSel;
q.dwOffset = dwOffset;
nodes += 4;
for (i = 0; i < 4; i++)
{
   if (p[i].colour == '(')          /* link p[i] to its parent */
   {
     gn++;
     if ( debug == 1 )
     {
      wsprintf(szString2,"Another gray %d\n ",gn);
      MessageBox(NULL, szString2, szString3, MB_OK);
     }

     switch (i)
     {
      case 0 :
        quadtree->wSelNW = p[0].wSel;
        quadtree->dwOffsetNW = p[0].dwOffset;
        quadtree -> colour = p[0].colour;
        wSelChild = p[0].wSel;
        dwOffsetChild = p[0].dwOffset;

        dwBuff = PointerAlloc(wSelChild,dwOffsetChild, QuadSize);
```

```
 if (STOP == 1)
 {
   wsprintf(szString2,"Allocate problem 19\n %d %ld",
                wSelChild,dwOffsetChild);
   MessageBox(NULL, szString2, szString3, MB_OK);
 }
 child = (QTREE) dwBuff;
 child->wSelFather = wSel;
 child->dwOffsetFather = dwOffset;
 PointerFree(wSelChild,dwOffsetChild);
 break;

case 1 :
  quadtree->wSelNE = p[1].wSel;
  quadtree->dwOffsetNE = p[1].dwOffset;
  wSelChild = p[1].wSel;
  dwOffsetChild = p[1].dwOffset;
  quadtree -> colour = p[1].colour;

  dwBuff = PointerAlloc(wSelChild,dwOffsetChild,QuadSize);
  if (STOP == 1)
  {
    wsprintf(szString2,"Allocate problem 20\n %d %ld",
                wSel,dwOffset);
    MessageBox(NULL, szString2, szString3, MB_OK);
  }
  child = (QTREE) dwBuff;
  child->wSelFather = wSel;
  child->dwOffsetFather = dwOffset;
  PointerFree(wSelChild,dwOffsetChild);
  break;

case 2 :
  quadtree->wSelSW = p[2].wSel;
  quadtree->dwOffsetSW = p[2].dwOffset;
  quadtree -> colour = p[2].colour;
  wSelChild = p[2].wSel;
  dwOffsetChild = p[2].dwOffset;

  dwBuff = PointerAlloc(wSelChild,dwOffsetChild,QuadSize);
  if (STOP == 1)
  {
    wsprintf(szString2,"Allocate problem 16\n %d %ld",
                wSel,dwOffset);
    MessageBox(NULL, szString2, szString3, MB_OK);
  }
  child = (QTREE) dwBuff;
  child->wSelFather = wSel;
  child->dwOffsetFather = dwOffset;
  PointerFree(wSelChild,dwOffsetChild);
  break;

case 3 :
```

```
        quadtree->wSelSE = p[3].wSel;
        quadtree->dwOffsetSE = p[3].dwOffset;
        quadtree -> colour = p[3].colour;
        wSelChild = p[3].wSel;
        dwOffsetChild = p[3].dwOffset;

        dwBuff = PointerAlloc(wSelChild,dwOffsetChild,QuadSize);
        if (STOP == 1)
        {
          wsprintf(szString2,"Allocate problem 16\n %d %ld",
                      wSel,dwOffset);
          MessageBox(NULL, szString2, szString3, MB_OK);
        }
        child = (QTREE) dwBuff;
        child->wSelFather = wSel;
        child->dwOffsetFather = dwOffset;
        PointerFree(wSelChild,dwOffsetChild);
        break;
     }
}
else                    /* create a maximal node for p[i] */
{
  wold = wSel;
  OldOffset = dwOffset;
  CurrentIndex = CurrentIndex + 1;
  if (CurrentIndex < NODES_PER_ALLOC)
   dwOffset = CurrentIndex * QuadSize;
  else
  {
              /* allocate another  */
   wsprintf(szString2,"Allocate another now\n %ld",CurrentIndex);
   MessageBox(NULL, szString2, szString3, MB_OK);
   wold = wSel;
   OldOffset = dwOffset;
   wSel = QuadSpaceAllocate();
   CurrentwSel = wSel;
   CurrentIndex = 0;
   dwOffset = 0;
   }
   dwBuff = PointerAlloc(wSel,dwOffset,QuadSize);
   if (STOP == 1)
    {
    wsprintf(szString2,"Allocate problem 17\n %d %ld",wSel,dwOffset);
    MessageBox(NULL, szString2, szString3, MB_OK);

    wsprintf(szString2,"Current Index and QuadSize \n %ld %ld",
              CurrentIndex, QuadSize);
    MessageBox(NULL, szString2, szString3, MB_OK);
    }

    r = (QTREE) dwBuff;
    r->colour = p[i].colour;
    r -> wSelBlack = 0;
```

```
r -> dwOffsetBlack = 0;
if ( (r->colour) == 'B')
 {
   bn++;
   if ( debug == 1 )
   {
     wsprintf(szString2,"Another black %d\n ",bn);
     MessageBox(NULL, szString2, szString3, MB_OK);
   }
   blk.wSel = wSel;
   blk.dwOffset = dwOffset;
   PointerFree(wSel,dwOffset);
   AddBlack(blk, ASC_code);    /* add the link list node */
   dwBuff = PointerAlloc(wSel,dwOffset,QuadSize);
   if (STOP == 1)
     {
      wsprintf(szString2,"Allocate problem 17\n %d %ld",
                  wSel,dwOffset);
      MessageBox(NULL, szString2, szString3, MB_OK);

      MessageBox(NULL, szString2, szString3, MB_OK);
     }
  else
     r = (QTREE) dwBuff;
     if ( debug == 1 )
     {
        fprintf(outfile,"Add node at  %d %ld %d \n",
           wSel, dwOffset, ASC_code);
        fprintf(outfile,"Add new Black %d %ld\n",
         r -> wSelBlack,r -> dwOffsetBlack);
         fprintf(outfile,"BLACK  \n");
     }
 }
else
 {
   wn++;
   if ( debug == 1 )
   {
     wsprintf(szString2,"Another white %d\n ",wn);
     MessageBox(NULL, szString2, szString3, MB_OK);
     fprintf(outfile,"WHITE  \n");
   }
 }
switch (i)
{
case 0:
  r->wSelNW = 0;
  r->dwOffsetNW = 0;
  quadtree -> wSelNW =  wSel;
  quadtree -> dwOffsetNW =  dwOffset;
  break;

case 1:
```

```
                r->wSelNE = 0;
                r->dwOffsetNE = 0;
                quadtree -> wSelNE =  wSel;
                quadtree -> dwOffsetNE =  dwOffset;
                break;

              case 2:
                r->wSelSW = 0;
                r->dwOffsetSW = 0;
                quadtree -> wSelSW =  wSel;
                quadtree -> dwOffsetSW =  dwOffset;
                break;

              case 3:
                r->wSelSE = 0;
                r->dwOffsetSE = 0;
                quadtree -> wSelSE =  wSel;
                quadtree -> dwOffsetSE = dwOffset;
                break;
            }
              r->wSelFather = q.wSel;
              r->dwOffsetFather =  q.dwOffset;

            }
        }  /* for i */
        quadtree->colour = '(';
        if ( debug ==  1 )
           fprintf(outfile,"GRAY  \n");
        par.wSel = q.wSel;
        par.dwOffset = q.dwOffset;
        par.colour = '(';
        }  /* create a non-terminal GRAY node */
    }  /* for non-pixel */
    return(par);
} /* Construct */

/*------------------------ End of Constr.c ------------------------------*/
```

```
/*****************************************************************************
*   Module: AddBlack.c
*   System: PS/2 Model 70
*           MicroSoft C7.0
*
*   Purpose: Module to add a linked list node to a black leaf node.
*
*   Programmer: Lisa Mullin
*   Date : November 1, 1992.
*   Detail:
*
*
* ------------------------------------------------------------------------- */
/* ---------------------------------------- Include Files ----------- */
#include <windows.h>
#include <winmem32.h>
#include <stdio.h>
#include "quad.h"
#include "quadtree.h"
/* ---------------------------------------- Module Definitions ------ */

            /* none */

/* ---------------------------------------- Module Macros ----------- */

            /* none */

/* ---------------------------------------- Imported Variables ------ */

extern  int   STOP, debug;

extern  long  int CurrentIndexBlack;

extern  FILE  *outfile;

extern  WORD  CurrentwSelBlack;
extern  DWORD LinkSize, QuadSize;

/* ---------------------------------------- Imported Functions ------ */
extern void  FreeSpace(WORD wSel);                    /* in Free.c      */
extern WORD  LinkSpaceAllocate();                     /* in LSpace.c    */
extern DWORD PointerAlloc(WORD wSel, DWORD dwOffset, DWORD LinkSize);
                                              /* in PAlloc.c    */
extern void  PointerFree(WORD wSel,DWORD dwOffset);    /* in PFree.c     */

/* ---------------------------------------- Exported Variables ------ */

            /* none */

/* ---------------------------------------- Local Typedefs ---------- */

            /* none */
```

```
/* ---------------------------------------- Local Global Variables -- */

            /* none */

/* ---------------------------------------- Local Functions --------- */

            /* none */

/* ---------------------------------------- Exported Functions ------ */

            /* none */

/* ------------------------------------------------------------------- */

void AddBlack(head, Code)
pair head;
int Code;
{
 qtree    *node;
 black    *blacknode, *r;

 char     szString2[95];
 char     szString3[45];

 int      found;

 WORD     wSel, oldwSel;
 DWORD    dwOffset,  dwBuff, olddwOffset, dwOffsetBlack;


 wsprintf(szString3,"Add Black");
 found = 0;
 oldwSel = 0;

 dwBuff = PointerAlloc(head.wSel,head.dwOffset,QuadSize);
 if (STOP == 1)
    {
      wsprintf(szString2,"Allocate problem 1\n %d %ld",wSel,dwOffset);
      MessageBox(NULL, szString2, szString3, MB_OK);
    }
  else
    {
      node = (QTREE) dwBuff;           /* get the quadtree node  */
      wSel = node -> wSelBlack;
      dwOffset = node -> dwOffsetBlack;
      PointerFree(head.wSel,head.dwOffset);
     }
                  /* traverse the linked list looking for the save code */
 while ( (wSel != 0)  && (found == 0) && ( STOP == 0) )
   {
   dwBuff = PointerAlloc(wSel,dwOffset,LinkSize);
   if (STOP == 1)
      {
```

```
            wsprintf(szString2,"Allocate problem 2\n %d %ld",wSel,dwOffset);
            MessageBox(NULL, szString2, szString3, MB_OK);
        }
        else
        {
            blacknode = (BNODE) dwBuff;
            if (( blacknode->type == 'f') && (blacknode->EMR_code == Code) )
               {
                  found = 1;
               }
             if ( debug == 1 )
                fprintf(outfile,"In AddBlack passed %d and found = %d\n",
                          blacknode->EMR_code,found);
            oldwSel = wSel;
            olddwOffset = dwOffset;
            wSel = blacknode -> wSelNext;
            dwOffset = blacknode -> dwOffsetNext;
            PointerFree(oldwSel,olddwOffset);
        }
    }
                        /* if the code is not in the linked list add it */
    if ( (found == 0) && (STOP == 0) )
       {
          CurrentIndexBlack = CurrentIndexBlack + 1;
          if (CurrentIndexBlack < NODES_PER_ALLOC_BLACK)
              dwOffsetBlack = CurrentIndexBlack * LinkSize;
          else
            {
                                 /* allocate another  */
              wsprintf(szString2,"Allocate another now Black list\n %ld",
                           CurrentIndexBlack);
              MessageBox(NULL, szString2, szString3, MB_OK);
              CurrentwSelBlack = LinkSpaceAllocate();
              CurrentIndexBlack = 0;
              dwOffsetBlack = 0;
            }
            dwBuff = PointerAlloc(CurrentwSelBlack,dwOffsetBlack,LinkSize);
            if (STOP == 1)
             {
               wsprintf(szString2,"Allocate problem 3\n %d %ld",
                           CurrentwSelBlack,dwOffsetBlack, LinkSize);
               MessageBox(NULL, szString2, szString3, MB_OK);
            }
            else
            {
              blacknode = (BNODE) dwBuff;
              blacknode -> type = 'f';
              blacknode -> EMR_code = Code;
              blacknode -> wSelNext = 0;
              blacknode -> dwOffsetNext = 0;
              if ( debug == 1 )
                  fprintf(outfile,"Add BLACK to end of list \n");
              PointerFree(CurrentwSelBlack,dwOffsetBlack);
```

```
                if ( oldwSel == 0)
                {
                   dwBuff = PointerAlloc(head.wSel,head.dwOffset,QuadSize);
                   if (STOP == 1)
                     {
                      wsprintf(szString2,"Allocate problem 1\n %d %ld",wSel,
                               dwOffset);
                      MessageBox(NULL, szString2, szString3, MB_OK);
                      }
                  else
                     {
                      node = (QTREE) dwBuff;
                      node -> wSelBlack = CurrentwSelBlack;
                      node -> dwOffsetBlack = dwOffsetBlack;
                      PointerFree(head.wSel,head.dwOffset);
                     }
                }
                else
                   {
                     dwBuff = PointerAlloc(oldwSel,olddwOffset,LinkSize);
                     if (STOP == 1)
                       {
                          wsprintf(szString2,"Allocate problem 1\n %d %ld",
                                 oldwSel,olddwOffset);
                          MessageBox(NULL, szString2, szString3, MB_OK);
                         }
                       else
                       {
                         r = (BNODE) dwBuff;
                         r -> wSelNext = CurrentwSelBlack;
                         r -> dwOffsetNext = dwOffsetBlack;
                         PointerFree(oldwSel,olddwOffset);
                       }
                   }
               }
    }
}
```

```
/* ------------------------------------------------------------------------
 *
 *   Module: IMAGE.c
 *   System: PS/2 Model 70
 *           MicroSoft QuickC
 *
 *   Purpose: This module return the value of 0 for a while pixel and a
 *            one for a black pixel at x,y.
 *
 *   Programmer: Lisa Mullin
 *   Date :
 *   Detail:
 *
 *
 * ------------------------------------------------------------------------ */
/* ---------------------------------------- Include Files ----------- */
#include <windows.h>
#include <winmem32.h>
#include "quad.h"
#include "quadtree.h"
/* ---------------------------------------- Exported Variables ------ */

            /* none */

/* ---------------------------------------- Local Typedefs ---------- */

            /* none */

/* ---------------------------------------- Local Global Variables -- */

            /* none */

/* ---------------------------------------- Local Functions --------- */

            /* none */

/* ---------------------------------------- Exported Functions ------ */

            /* none */
/* ------------------------------------------------------------------------ */

int IMAGE (x,y)
long int     x,y;

{
     char    szString2[95];
     char    szString3[45];
     int j;
     extern rlength rlc[maxrlc];
     extern long int norlc;
     extern int debug, first_call;
```

```
int     i;

   if ( debug == 1 )
   {
     wsprintf(szString3,"Image\n ");
     wsprintf(szString2,"Process begins \n ");
     MessageBox(NULL, szString2, szString3, MB_OK);
   }
   i = 0;

   if ( y > rlc[(norlc-1)].r )
     {
       return(0);
     }
   else
   {
       while ((rlc[i].r < y) && (i < norlc))  i++;

       if ((rlc[i].r != y) || (i >= norlc))
       {
           return(0);
       }
       else
       {
          while ((rlc[i].r == y) && (rlc[i].xe < x)) i++;
          if ((rlc[i].r == y) && (rlc[i].xb <= x) &&
            (rlc[i].xe >= x))
           {
            return(1);
           }
           else
           {
              return(0);
           }
        }
     }
 }
```

```
/* ------------------------------------------------------------------------
 *
 *   Module: LSpace.c
 *   System: PS/2 Model 70
 *           MicroSoft C7.0
 *
 *   Purpose: Module to allocate 32 bit memory for the linked lists.
 *
 *   Programmer: Lisa Mullin
 *   Date : Sept 6, 1992.
 *   Detail:
 *
 *
 * ------------------------------------------------------------------------ */
/* ---------------------------------------- Include Files ----------- */
#include <windows.h>
#include <winmem32.h>
#include "quad.h"
#include "quadtree.h"

/* ---------------------------------------- Module Definitions ------ */

            /* none */

/* ---------------------------------------- Module Macros ----------- */

            /* none */

/* ---------------------------------------- Imported Variables ------ */

extern int STOP;
extern DWORD LinkSize;

/* ---------------------------------------- Imported Functions ------ */

            /* none */

/* ---------------------------------------- Exported Variables ------ */

            /* none */

/* ---------------------------------------- Local Typedefs ---------- */

            /* none */

/* ---------------------------------------- Local Global Variables -- */

            /* none */

/* ---------------------------------------- Local Functions --------- */

            /* none */
```

```
/* ---------------------------------------- Exported Functions ------ */

            /* none */

/* ------------------------------------------------------------------- */


WORD LinkSpaceAllocate()

{
   char    szString2[95];
   char    szString3[45];


   LPWORD lpSel;
   WORD flag, Retvalue;
   static DWORD ver;
   WORD wSel;


  wsprintf(szString3,"Link Space Allocation");

  wsprintf(szString2,"Link Memory allocation \n %d \n %ld"
                        ,Retvalue, (NODES_PER_ALLOC * LinkSize));

  ver = GetWinMem32Version();

  if ( ver < 0x0101)
      flag =  (WORD) 0;
  else
      flag =  (WORD) GMEM_DDESHARE;

  lpSel = &wSel;

  Retvalue = Global32Alloc( (DWORD)(NODES_PER_ALLOC * LinkSize), lpSel,
                        (DWORD)(NODES_PER_ALLOC * LinkSize), flag);
  if ( Retvalue != 0)
      {
        wsprintf(szString2,"Link Memory allocation problem\n %d \n %ld"
                        ,Retvalue, (NODES_PER_ALLOC * LinkSize));
        MessageBox(NULL, szString2, szString3, MB_ICONSTOP);
        STOP = 1;
      }

  return(wSel);

}

 /* ----------------------- End of LSpace.c ------------------------------ */
```

```
/* ------------------------------------------------------------------------
 *
 *   Module: PAlloc.c
 *   System: PS/2 Model 70
 *           MicroSoft C7.0
 *
 *   Purpose: Module to allocate 16:32 bit pointer.
 *
 *   Programmer: Lisa Mullin
 *   Date : Sept 6, 1992.
 *   Detail:
 *
 *
 * ------------------------------------------------------------------------ */
/* ---------------------------------------- Include Files ---------- */
#include <windows.h>
#include <winmem32.h>
#include "quad.h"
#include "quadtree.h"

/* ---------------------------------------- Module Definitions ------ */

            /* none */

/* ---------------------------------------- Module Macros ---------- */

            /* none */

/* ---------------------------------------- Imported Variables ------ */

extern int STOP;

/* ---------------------------------------- Imported Functions ------ */

            /* none */

/* ---------------------------------------- Exported Variables ------ */

            /* none */

/* ---------------------------------------- Local Typedefs ---------- */

            /* none */

/* ---------------------------------------- Local Global Variables -- */

            /* none */

/* ---------------------------------------- Local Functions --------- */

            /* none */

/* ---------------------------------------- Exported Functions ------ */
```

```
            /* none */

/* ------------------------------------------------------------------------ */


DWORD PointerAlloc(WORD wSel,DWORD dwOffset, DWORD Size)

{
   char    szString2[95];
   char    szString3[45];
   WORD    Retvalue;
   DWORD   dwBuff;
   LPDWORD lpBuffer;

   wsprintf(szString3,"Pointer Allocation ");

   lpBuffer = &dwBuff;

   Retvalue = Global16PointerAlloc(wSel,dwOffset,lpBuffer,Size,0);
   if ( Retvalue != 0)
    {
       wsprintf(szString2,
            "return value %d \nwSel %d \noffset %ld \nsize %d"
            ,Retvalue,wSel,dwOffset,Size);
       MessageBox(NULL, szString2, szString3, MB_ICONSTOP);
       STOP = 1;
    }

   return(dwBuff);

}

 /* ----------------------- End of PAlloc.c ---------------------------- */
```

```
/* ------------------------------------------------------------------------
 *
 *   Module: PFree.c
 *   System: PS/2 Model 70
 *           MicroSoft C7.0
 *
 *   Purpose: Module to free 16 bit pointer.
 *
 *   Programmer: Lisa Mullin
 *   Date : Sept 6, 1992.
 *   Detail:
 *
 *
 * ------------------------------------------------------------------------ */
/* ---------------------------------------- Include Files ---------- */
#include <windows.h>
#include <winmem32.h>
#include "quad.h"
#include "quadtree.h"

/* ---------------------------------------- Module Definitions ------ */

            /* none */

/* ---------------------------------------- Module Macros ---------- */

            /* none */

/* ---------------------------------------- Imported Variables ------ */

extern int STOP;

/* ---------------------------------------- Imported Functions ------ */

            /* none */

/* ---------------------------------------- Exported Variables ------ */

            /* none */

/* ---------------------------------------- Local Typedefs ---------- */

            /* none */

/* ---------------------------------------- Local Global Variables -- */

            /* none */

/* ---------------------------------------- Local Functions --------- */

            /* none */

/* ---------------------------------------- Exported Functions ------ */
```

```
            /* none */

/* ------------------------------------------------------------------------ */


void PointerFree(WORD wSel, DWORD dwOffset)

{
   char    szString2[95];
   char    szString3[45];
   WORD    Retvalue;

   wsprintf(szString3,"Free Space");

   Retvalue = Global16PointerFree(wSel,dwOffset,0);
   if ( Retvalue != 0)
      {
        wsprintf(szString2,"Pointer free problem\n %d",Retvalue);
        MessageBox(NULL, szString2, szString3, MB_ICONSTOP);
        STOP = 1;

      }

}

 /* ----------------------- End of PFree.c ------------------------------ */
```

```
/* ------------------------------------------------------------------------
 *
 *   Module: QSpace.c
 *   System: PS/2 Model 70
 *           MicroSoft C7.0
 *
 *   Purpose: Module to allocate 32 bit memory for the quadtree nodes.
 *
 *   Programmer: Lisa Mullin
 *   Date : Sept 6, 1992.
 *   Detail:
 *
 *
 * ------------------------------------------------------------------------ */
/* ---------------------------------------- Include Files ---------- */
#include <windows.h>
#include <winmem32.h>
#include "quad.h"
#include "quadtree.h"

/* ---------------------------------------- Module Definitions ------ */

            /* none */

/* ---------------------------------------- Module Macros ---------- */

            /* none */

/* ---------------------------------------- Imported Variables ------ */

extern int STOP;
extern DWORD QuadSize;

/* ---------------------------------------- Imported Functions ------ */

            /* none */

/* ---------------------------------------- Exported Variables ------ */

            /* none */

/* ---------------------------------------- Local Typedefs ---------- */

            /* none */

/* ---------------------------------------- Local Global Variables -- */

            /* none */

/* ---------------------------------------- Local Functions --------- */

            /* none */
```

```
/* ---------------------------------------- Exported Functions ------ */

            /* none */

/* ------------------------------------------------------------------- */


WORD QuadSpaceAllocate()

{
   char    szString2[95];
   char    szString3[45];


   LPWORD lpSel;
   WORD flag, Retvalue;
   static DWORD ver;
   WORD wSel;


   wsprintf(szString3,"Quad Space Allocation");

   ver = GetWinMem32Version();

   if ( ver < 0x0101)
       flag =  (WORD) 0;
   else
       flag =  (WORD) GMEM_DDESHARE;

   lpSel = &wSel;

   Retvalue = Global32Alloc( (DWORD)(NODES_PER_ALLOC * QuadSize), lpSel,
                        (DWORD)(NODES_PER_ALLOC * QuadSize), flag);
   if ( Retvalue != 0)
      {
        wsprintf(szString2,"Quad Memory allocation problem\n %d \n %ld"
                        ,Retvalue, (NODES_PER_ALLOC * QuadSize));
        MessageBox(NULL, szString2, szString3, MB_ICONSTOP);
        STOP = 1;
      }
  return(wSel);

}

/* ----------------------- End of QSpace.c ------------------------------ */
```

Vita

Candidate's Full Name:	Lisa Mullin
Place and Date of Birth:	Gaspe, Quebec, October 24, 1966.
Permanent Address:	647 York Blvd West Gaspe, Quebec Canada
Schools Attended:	1973 - 1979 Sacred Heart School 1979 - 1984 C.E. Pouliot Polyvalent 1984 - 1985 CEGEP de la Gaspesie 1985 - present University of New Brunswick
Publications:	Mullin, L. and Nickerson, B.G. [1991], "A Knowledge-Based System for Automated Name Placement", Proceedings of the 4th UNB AI Symposium, Fredericton, N.B., Canada, Sept. 20-21, 1991.