RASTERIZING THE CIA WORLD DATA BANK II

by

Mahesh T. Ursekar

Project submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

APPROVED:

[illegible]

Dr. Clifford A. Shaffer, Chairman

James B Campbell

Dr. James Campbell

[illegible]

Dr. Edward A. Fox

Sept., 1991

Blacksburg, Virginia

c.2

LD
5655
V851
1991
H773
c.2

RASTERIZING THE CIA WORLD DATA BANK II

by

Mahesh T. Ursekar

Committee Chairman: Dr. Clifford A. Shaffer

Department of Computer Science

(ABSTRACT)

The geographic database is central to a Geographical Information System. The database may commonly be either in vector or raster format. The CIA World Data Bank II is a public domain vector database of the entire world to one second resolution. However, the database, in its standard distribution format, contains numerous unmarked line segment intersections and 'orphan' line segments that represent unclosed polygons. This makes it nearly impossible to do many GIS operations such as point-in-polygon location and boundary following. This project makes use of the PMR (Polygonal Map Random) quadtree data structure as a spatial index to the vector database to support editing and rasterizing operations. Using the PMR quadtree, the entire database for internal boundaries and coastlines is searched to remove unmarked intersections and orphans. Then various additional boundaries for major seas, gulfs and bays are added. Finally, a novel approach for converting the database from its current form to a raster is developed. This technique first builds the database into a line quadtree structure and then uses a flood-fill algorithm to color the different polygons (countries, islands and lakes). In the final stage, the line quadtree is converted to a region quadtree. All editing and display programs were developed using the X Window System.

ACKNOWLEDGEMENTS

I wish to extend my sincere thanks to Dr. Clifford A. Shaffer, my committee chairman and advisor for his immense help and guidance throughout this project. The project could not have been completed without him. Thanks are also due to Dr. James Campbell and Dr. Edward A. Fox for serving on my committee. I would also like to take this opportunity to thank my parents and brothers for the support and encouragement they have given me throughout my education, both here and at home. For all the enjoyable Friday evenings, I am grateful to Vasant, Vish, and Sam. Kalyan and Arun have been the best pair of roommates I've had the opportunity of staying with and I thank them for all the fun, good food and movies we've enjoyed together. My colleagues Vincent, Sheryl, Tim and Pat have been most helpful and fun to work with and I thank them for the same.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Geographical Information Systems (GIS) are gaining in importance every day. A GIS is a set of tools for collecting, storing, retrieving, transforming, and displaying spatial data from the real world for a particular application [Burr86]. Examples of applications are urban planning, soil surveying, utility mapping etc. A GIS is supported by computer hardware and software. The hardware could typically include a central processing unit (CPU), a digitizer, a plotter and tape and disk storage. Software modules exist for data input and verification, database management, data output, data transformation and user interfaces.

Central to software in a GIS is the geographic database. There are two traditional ways in which the database can be organized: vector and raster. These two forms of storage are not mutually exclusive; techniques exist for creating a database that combines the two schemes [Burr86]. The relative merits and demerits of the two approaches is summarized in Table 1.1 [Burr86].

A vector database is composed of 3 main entities - points, line segments and areas or polygons. Points may be stored as simple xy-coordinates, line segments as a pair of xy-coordinates and polygons as either series of line segments or as series of points. In addition, all entities may have attribute information associated with them that further describe the particular entity.

Vector Methods

Advantages

• Good representation of phenomenological data structure
• Compact data structure
• Accurate graphics
• Retrieval, updating and generalization of graphics and attributes are possible

Disadvantages

• Complex data structures and algorithms
• Combination of several vector polygon maps through overlay creates difficulty
• Display and plotting can be expensive, particularly for high quality, color and cross-hatching
• The technology is expensive, particularly for the more sophisticated software and hardware

Raster Methods

Advantages

• Simple data structures
• The overlay and combination of mapped data with remotely sensed data is easy
• Various kinds of spatial analysis are easy
• The technology is cheap and being energetically developed

Disadvantages

• Volumes of graphic data
• The use of large cells to reduce data volumes means that phenomenologically recognizable structures can be lost and there can be a serious loss of information
• Crude raster maps are considerably less beautiful than maps drawn with fine lines.

Table 1.1: Comparison of Vector and Raster Methods. Adapted from [Burr86]

Raster data, on the other hand, is an array of grid cells or picture elements (pixels) with attribute data associated with each element. A point may be a single pixel, a line segment, a string of neighboring pixels or an area a group of adjoining pixels sharing the same value.

The CIA World Data Bank II (WDBII) [1] is a vector database of the entire world. The data describing the international boundaries and coastlines, islands and lakes of rank 1 (the meaning of this term is described in Section 2.1) are about 2 million line segments. The segments are arranged in the form of chains where a single chain typically contains a few hundred line segments. Unfortunately, the data are not well organized. The chains do not form complete polygons and in addition, there are intersections and orphans. By intersections we mean crossing of two line segments in the database where the intersection is not a vertex in the database. An orphan is a line segment that is connected to another segment at only one endpoint while the other end-point is free, i.e., it is not part of a closed polygon.

Since WDBII segments are not associated with any attribute identifying the countries which they surround, point-in-polygon searches are particularly difficult. A point-in-polygon search determines whether a query point lies inside, outside or on the border of a given polygon. Further, because of the intersections and orphans contained in this database, point-in-polygon algorithms become nearly impossible. Moreover, even if these searches could be performed, the lack of organization for the chains combined with the shear size of the database would make them prohibitively expensive.

Another useful primitive operation is boundary following. In boundary following, given a starting segment, it is required to follow it in a clockwise or counter-clockwise direction along the boundary of the polygon of which it forms a border. However, border following algorithms applied to polygons with unmarked intersections will cause the border following to end at the intersection since the next segment to be visited cannot be determined, and segments ending in an orphan do not form a polygon at all. Hence, this operation is not

1The database can be obtained from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161 feasible with the WDBII in its standard format.

The objective of this project is to make the database more useful than it is in its current form in order to support future work by our research group and others. To accomplish this, errors in the database must be corrected, and then each of the countries in the database must be assigned a label. This can be done more easily if the database is in a raster format than in the vector form originally provided. The methodology used to rasterize the database and label the polygons is divided into five phases as follows:

a) Remove all intersections and orphans from the vector database.

b) Build a line quadtree [Same84], with all the segments in the vector database.

c) 'Color' each of the polygons formed by the above process.

d) Build a region quadtree [Same89a] using the 'colored' line quadtree. The region quadtree is equivalent to a raster but has a more compact representation.

e) Assign a label to each of the different colored polygons.

The above steps are described below as follows. Chapter 2 gives detailed information on the CIA WDBII. Chapter 3 talks about the cleanup procedure for removing intersections and orphans. Chapter 4 covers the rasterizing process and polygon labeling and Chapter 5 offers conclusions and future research.

Chapter 2

The CIA World Data Bank II

2.1 Organization of the WDBII

The CIA World Data Bank II (WDBII) is a vector database of the entire world. It divides the world into five geographic areas namely North America, South America (which includes Antarctica), Europe, Africa and Asia (which includes Australia). Figure 2.1 shows the division of the five areas graphically [CIA77].

The CIA WDBII is commonly available in two formats. One format is the CIA's original uncompressed version in which each of the five areas is contained on a single magnetic tape (9 track, 1600 BPI, odd parity, EBCDIC). A sample of the way the data is organized on tape is shown in Figure 2.2 with a description of what the different fields represent. As seen in the figure, the database is composed of a series of 20 character records. Each chain is classified by a chain number (columns 1-7), a rank (described later in this section) and by the number of points that make up the chain (columns 16-20). This is followed by a series of points. Each point is described by a latitude and a longitude and the number of the corresponding point.

The second format is the compressed version which is available via Internet. It contains the same information as the uncompressed version but instead of having each point individually listed as in the uncompressed version, only the coordinates of the first point of a

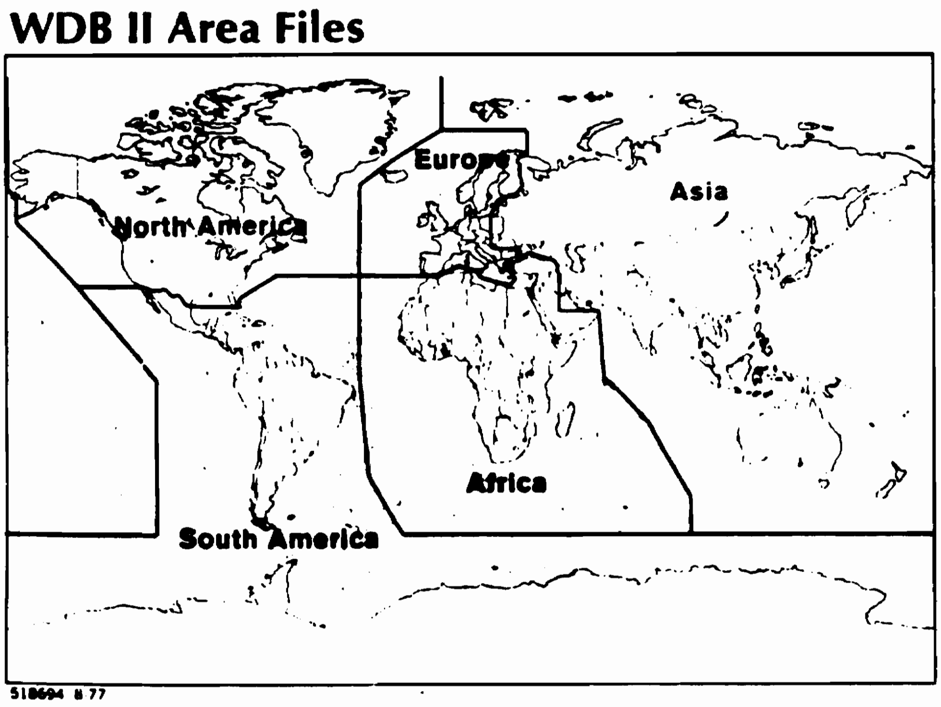


Figure 2.1: Graphical Breakup of the CIA WDBII

Columns:

```
1234567890123456789012345678901234567890123456789012345678901234567890
```

Data Sample:

```
3015010  1      66      0364559N  12  319E      1364542N 12   315E      2364525N .......
364450N  12  218E      6364441N  12  159E      7364432N 12   140E      8364425N .......
364450N  12  0  2E    12364419N  115935E     13364426N  115914E     14364439N .......
364517N  115816E     18364525N  115755E     19364531N  115731E     20364541N .......
```

Close-up of above data:

Columns:

```
1234567890123456789012345678901234567890

3015010  1      66      0364559N  12  319E     1
```

Columns description for header of chain:

1 - 7	Chain Number
8 - 9	Rank
10 - 15	Number of Points in Chain
16 - 20	Zeros

Columns description for rest of chain:

1 - 7	Latitude (DDMMSSI)
8 - 15	Longitude (DDDMMSSI)
16 - 20	Point Number

Figure 2.2: Data Organization of the WDBII

chain are listed after which are listed the x and y offsets of each point from the previous one. Compression results since the offsets require less storage space than actual coordinates. In addition, all coordinates are represented as integer values rather than as ASCII strings. North and east coordinates are positive numbers while south and west are negative numbers, all measured in seconds.

Both versions of the database are stored in five directories which we will refer to as:

a) africa.Map
b) asia.Map
c) europe.Map
d) namer.Map (North America)
e) samer.Map (South America)

Each directory contains 3 files except 'namer.Map' which contains 4. The files common to the directories are:

bdy.cbd : international boundaries
cil.cbd : coastlines, islands and lakes
riv.cbd : rivers

namer.Map contains the additional file:

pby.cbd : internal boundaries

Associated with the different chains in each file is a number known as the rank. The rank identifies the geographical feature represented by the chain. This is defined in Table 2.1 [CIA77].

This project used all the three ranks associated with the files bdy.cbd. Only the chains with rank 01 in cil.cbd were used. The files riv.cbd and pby.cbd were not used at all.

The compressed data occupies about 13.5 Mbytes on disk. The number of segments used by our project from each file is shown in Table 2.2. The resolution of the database is

I. International boundaries or limits of sovereignty (bdy.cbd)

01	Demarcated or delimited
02	Indefinite or in dispute
03	Other line of separation or sovereignty on land

II. Coast, islands, and lakes (cil.cbd)

01	Coast, islands, and lakes that appear on all maps
02	Additional major islands and lakes
03	Intermediate islands and lakes
04	Minor islands and lakes
06	Intermittent major lakes
07	Intermittent minor lakes
08	Reefs
09	Salt pans-major
10	Salt pans-minor
13	Ice shelves-major
14	Ice shelves-minor
15	Glaciers

III. Rivers (riv.cbd)

01	Permanent major rivers
02	Additional major rivers
03	Additional rivers
04	Minor rivers
05	Double-lined rivers
06	Intermittent rivers - major
07	Intermittent rivers - additional
08	Intermittent rivers - minor
10	Major canals
11	Canals of lesser importance
12	Canals-irrigation type

IV. Internal boundaries (pby.cbd)

01	First order administration

Table 2.1: Definitions for Ranks

Directory	File	Number of Segments
africa.Map	bdy.cbd	114,433
	cil.cbd (rank 1 only)	171,205
asia.Map	bdy.cbd	105,963
	cil.cbd (rank 1 only)	553,003
europe.Map	bdy.cbd	48,320
	cil.cbd (rank 1 only)	204,734
namer.Map	bdy.cbd	8,433
	cil.cbd (rank 1 only)	475,514
samer.Map	bdy.cbd	63,188
	cil.cbd (rank 1 only)	281,223
		Total: 2,026,016

Table 2.2: Number of Segments

one second, i.e., the minimum line segment is one second in length (line segments of this length do appear in the database). The longest segment is 3118.9 seconds long and the average length of the segments is about 35 seconds.

2.2 Quirks in the Database

There is an error in the file cil.cbd in the directory asia.Map. As can be seen from Figure 2.1, the boundary of Asia crosses the 180 degree longitude line. Therefore, there are segments which begin in the eastern hemisphere and end in the western hemisphere, i.e., a segment may have an eastern end longitude value such as $+179\ \ 59^{0}10^{\prime\prime}$ and a western hemisphere value such as $-179\ \ 59^{0}5^{\prime\prime}$. However, the database stores a group of segments with values less than −180 degrees. These segments are around the island of Fiji in the southern hemisphere, around the island of Wrangell in the northern hemisphere and on the northern border of the USSR that crosses between hemispheres. These are the only landmarks that cross the 180 line.

It was initially assumed that these coordinates referred to the eastern hemisphere i.e., −184 would translate to 180 − (184 − 180) = +176 . However, when interpreted this way, and then viewed on screen, the portion of Wrangell Island in the eastern hemisphere was stretched out on the northern border by almost 4 (14400 seconds). This caused the southern border to be incomplete. The same phenomenon was observed on one of the islands of Fiji. On analyzing the segments, it was found that the maximum value of a coordinate less than −180 was −184 5^{0} $49^{\prime\prime}$. This is an offset of 14749 seconds from −180 , the apparent offset observed on the display. Therefore, this value was added as the offset to all coordinates less than −180 ; i.e., −184 would translate to the usual 176 as before but 14749 seconds would be added to it. This approach worked and both islands and the northern border of the USSR were restored to their correct form.

Chapter 3

Database Clean Up

3.1 Data Structures for Phase I

3.1.1 Criteria for the Data Structure

The first phase of the project was to remove all intersections and orphans from the database. In order to do this efficiently an appropriate data structure was required. The unorganized list of chains in the original database would require that every line segment be compared against every other, virtually an impossible task for a database of this size. Thus a spatial index was required to allow intersections and orphans to be detected locally. There are a variety of spatial indexes available for storing vector data. The basis for choosing one for this phase of the project was that it at least meet the following criteria [Same85]:

a) It should store polygonal maps without information loss (i.e., it should not suffer a loss of accuracy resulting from digital approximations of line segments).

b) It should not be overly sensitive to the positioning of the map (i.e., modifying the data should not drastically increase the storage requirements). This is necessary since segments would have to be moved from their original locations to remove intersections.

c) It should be easy to manipulate (i.e., it should allow easy insertion and deletion of data).

d) It should facilitate more complex operations such as edge following and point-in-polygon operations.

Samet [Same89a] lists various spatial indexes for storing line data. These include the MX quadtree, the line quadtree, the edge quadtree, PM quadtrees, and Edge EXCELL. Of these, a variant of the PM quadtree called the PMR quadtree met the above requirements and was chosen as the data structure to be used. To explain the PMR quadtree, the region quadtree, which is one of the most studied quadtree structures, is introduced first. The region quadtree is also used in the final phase of the project.

3.1.2 The Region Quadtree

The quadtree data structure is most amenable to storing spatial data information. It is a hierarchical, variable resolution data structure which recursively subdivides the plane into four quadrants based on some decomposition rule. The decomposition rule is what differentiates one quadtree from another.

Figure 3.1 shows how a sample region would be represented using a region quadtree. Figure 3.1a is the region. Figure 3.1b is the region represented in an 8×8 bit grid. 1's correspond to picture elements (pixels) that are in the region and 0's to those that are outside the region. Figure 3.1c is the region as it would be decomposed by the region quadtree. It is seen that the image is successively divided into four equal sized quadrants. The subdivision continues until a quadrant is formed that contains all 1's or all 0's, i.e., the quadrant is entirely contained in the region or disjoint from it. The above process is represented by a tree as shown in Figure 3.1d. The root node corresponds to the entire grid. Each non-leaf or internal node has four children. Each child of a node represents a quadrant, labeled either NW (North-West), NE (North-East), SW (South-West) or SE (South-East). The leaf nodes of the tree correspond to those quadrants for which no further subdivision is necessary. A leaf node is termed BLACK if its corresponding quadrant is entirely contained within the region and WHITE if the quadrant is outside the region. All non-leaf or internal nodes are termed GRAY.

3.1.3 The PMR Quadtree

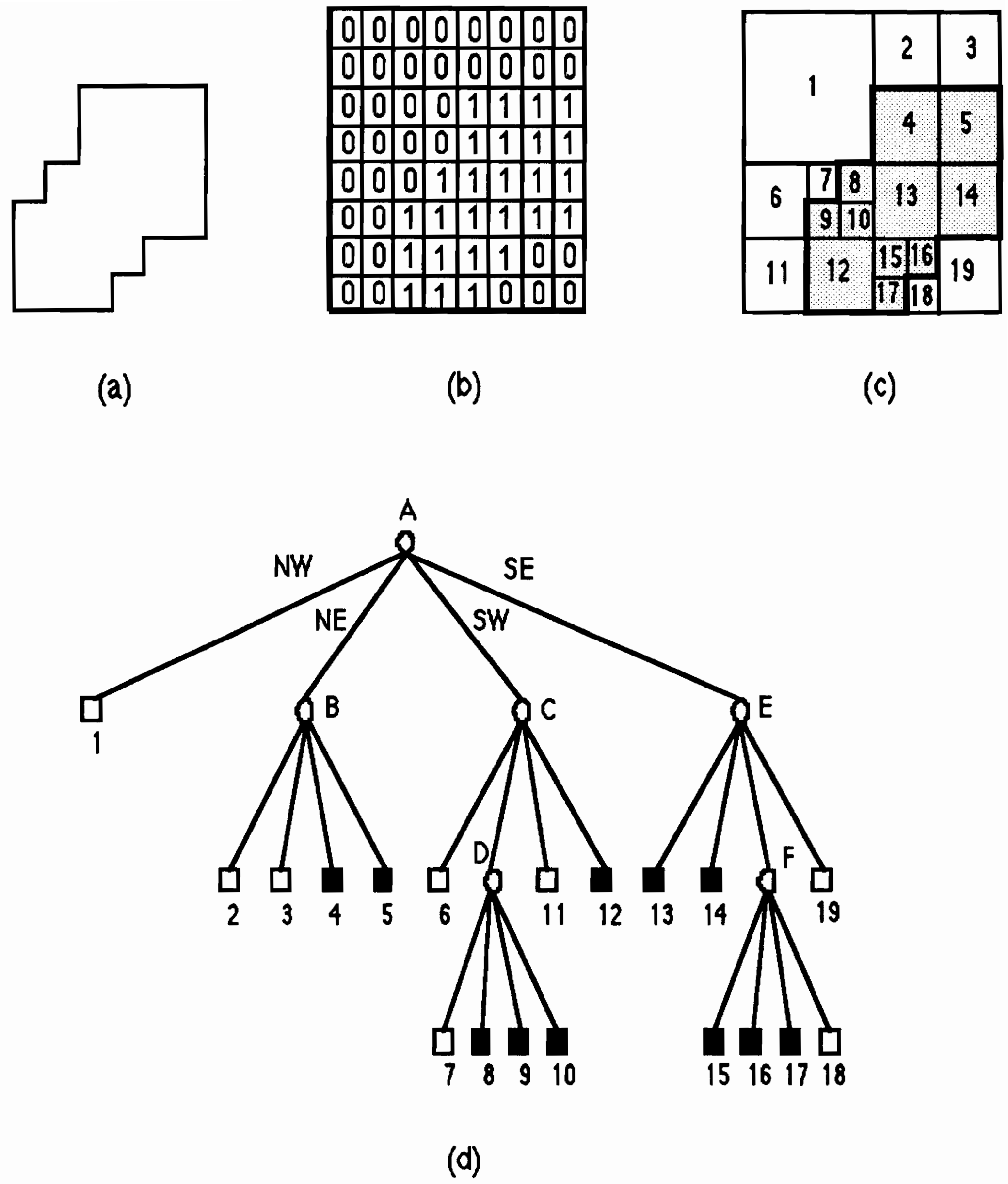
The PMR (Polygonal Map Random) quadtree, unlike the region quadtree, is used to store vector or line data. First the original structure as given in [Nels86] is described and then the modifications made to suit the requirements of this project are explained.

Figure 3.2 shows the building of the PMR quadtree as segments are added. The splitting rule states that if the number of segments (or portions of segments) in a node (i.e., quadrant) exceeds a given threshold, say n, then the node is split *once* into four quadrants. Figure 3.2 assumes a threshold of two. Adding segments 1 and 2 as shown in 3.2a does not cause any splitting since the threshold has not been exceeded. When segment 3 is added, the segment count in the southeast child of the root exceeds two and this causes one split as shown in 3.2b. Segments 4,5, and 6 do not cause any change. Note that in Figure 3.2d, insertion of segment 7 causes two blocks to be split (the NW and NE quadrants) since the capacity of each of these blocks is exceeded. The splits caused by insertion of segments 8 and 9 are shown in Figure 3.2d. An important characteristic of this splitting rule is that since a node is split *only once* when the insertion of a segment causes the threshold n to be exceeded, a node may contain more than n segments. This is seen in Figure 3.2d.

Since the PMR quadtree also accommodates deletion of segments, there is a corresponding merging rule that states that merge while the number of distinct line segments contained in the node and its siblings is less than or equal to n. This implies that if segment 7 was deleted from the tree in Figure 3.2c, the NW and NE quadrants of the root to each merge into one quadrant.

3.1.4 Modifications to the PMR Quadtree

Since the PMR quadtree allows a single node to hold more than n segments, this implies



A region, its binary array, its maximal blocks, and the corresponding quadtree. (a) Region (b) Binary array (c) Block decomposition of the blocks region in (a). Blocks in the region are shaded. (d) Quadtree representation of the blocks in (c).

Figure 3.1: The Region Quadtree

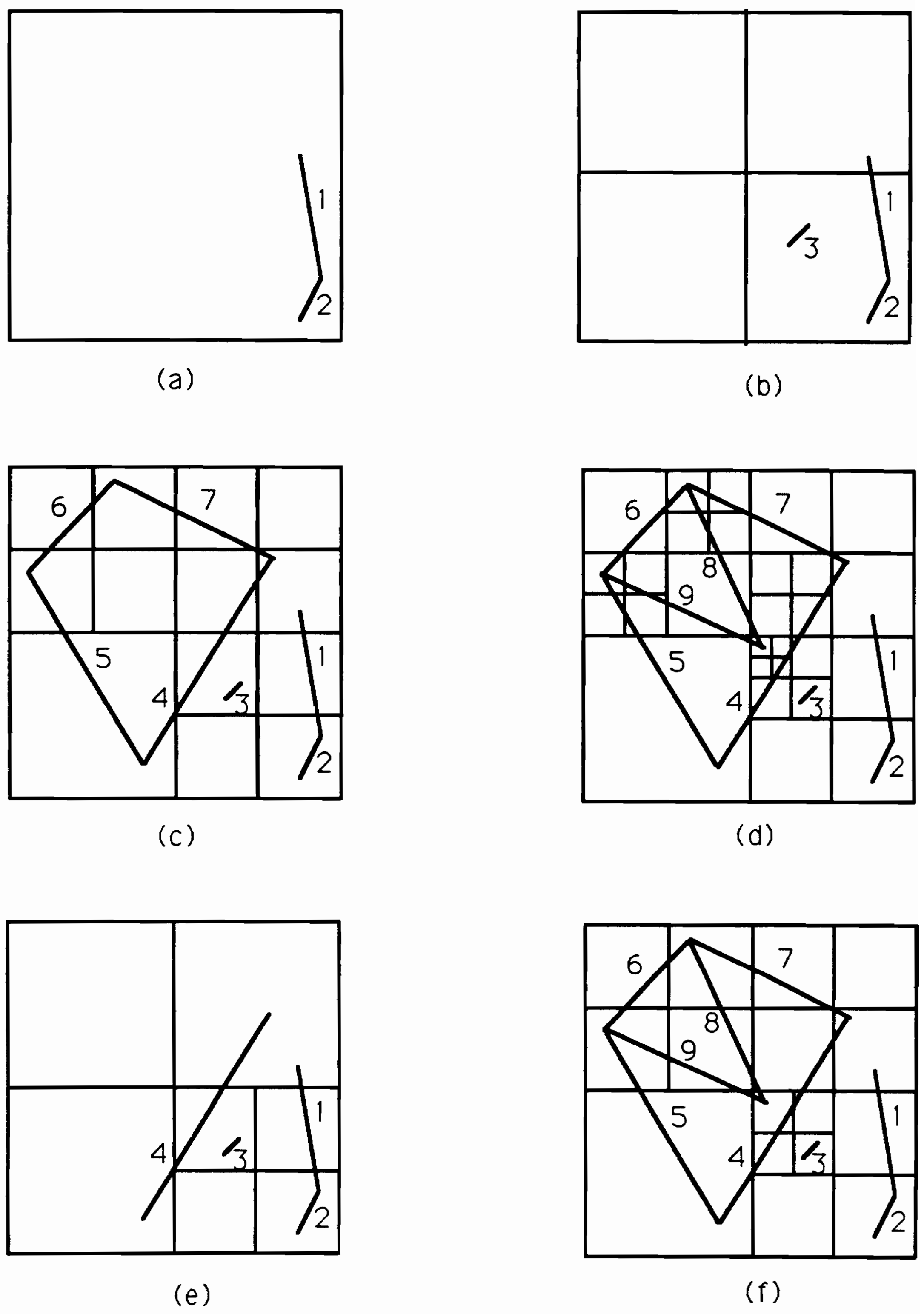


Figure 3.2: Building the PMR quadtree

that in terms of implementation, the data structure has to contain variable sized nodes. If the entire tree is to be built in main memory, dynamic memory allocation could be used to accommodate this. However, the size of our database requires that it reside on disk with only small portions stored in main memory at any given instant.

Variable sized nodes are much more complicated to implement than fixed sized nodes. Hence, the splitting rule was changed so that if the number of segments in the node exceeds the threshold, n, keep splitting the node into quadrants until all the new nodes contain fewer than n segments. With a threshold of two, it is not possible to build the modified PMR quadtree for the segments in Figure 3.2d. This is because there are points with 3 segments incident on them and no amount of splitting will reduce this number. The tree with a threshold of 3 is shown being built in Figure 3.2e-f. Note that is Figure 3.2e, only when segment 4 is added that the first split occurs. The complete tree is shown in Figure 3.2f.

We chose a threshold size of 5 for our project since in a typical map, it was unlikely that more than 5 line segments would ever meet at a point. The merging rule was not implemented and if a node did become empty, it was left untouched.

The quadtree structures shown have been based on the assumption that all quadrants have the same width and height. However, the aspect ratio of the earth is 2:1, i.e., when the globe is flattened and indexed by longitude and latitude, it is twice as wide as it is high. Hence, the PMR quadtree was constructed with quadrants having an aspect ratio of 2:1. The 'projection' used was strictly longitude and latitude corresponding to x and y coordinates respectively.

3.1.5 Storage of Line Segments

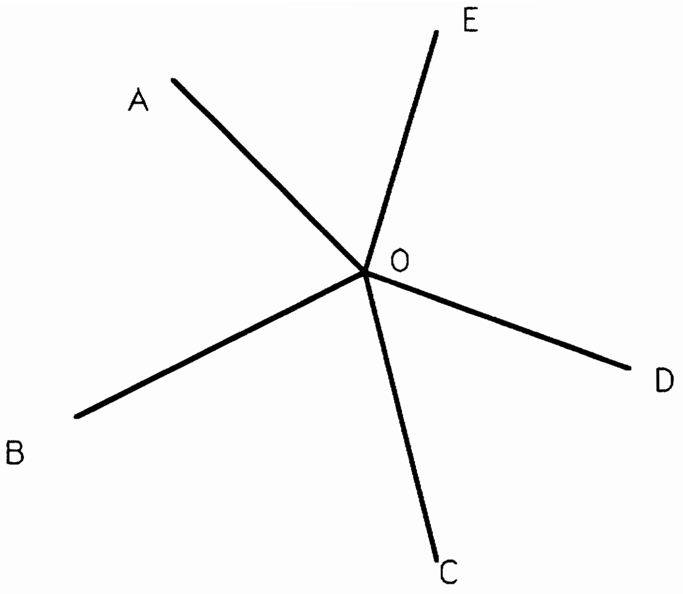
The next decision was how to store the line segments at each node of the quadtree. Initially, each node that a segment crossed stored both end-points of the segment. Since the threshold size chosen was 5, this meant that a node could store up to 10 points (2 points per node). With this structure, the tree became unacceptably large, and after the segments constituting Europe, Africa and Asia were inserted into the tree, it occupied 67 Mbytes of disk space. This was only 60% of the data, and the expected space requirements for the entire database exceeded available storage.

In order to reduce storage requirements, the fact that the database was arranged in the form of chains was exploited and the following novel modification was made. Say there was a chain formed by 3 points A, B and C and all three points fell within one quadtree node. In the earlier scheme, the node would store segments A-B and B-C with point B being stored twice. This can be eliminated if a check is made to see if the new segment being added forms a chain with the existing segment. This means that when segment B-C is inserted, since it forms a chain with A-B, only the point C is added and the chain A-B-C is stored at that node. It was decided that up to eight points would be stored per node which is equivalent to having a maximum of 5 segments meeting at a point. This is seen in Figure 3.3. The points forming the chains are A, B, C, D, E and O. In this simple example, our representation would store chains AOC, EOD and BO requiring a total of 8 points. This simple scheme offered considerable savings and Europe, Africa and Asia now occupied 30 Mbytes of storage, a saving of over 50%.

The program to generate the entire PMR quadtree for the selected portions of the WDBII took about 37 hours to complete on a MacII. Other statistics on the tree are given in Table 3.1

3.1.6 Cross-Over Segments

Given two points on the earth, there are always two ways to connect them along the great circle. Consider points A and B, in Figure 3.4, that lie on either side of the 180 degree longitude line. They can either be connected as shown, by a segment that crosses the 180 degree line, or else by a segment that goes around the earth along the great circle. The same applies for segments crossing over at 0 degrees longitude. To remove this ambiguity, the following convention is used. Points lying on either side of the 0 degree longitude or 180 degree longitude are connected with a segment that crosses the 0 degree or 180 degree



One method to store chains in quadrant:
A-O-C, E-O-D, B-O

Figure 3.3: Five Segments meeting at a point

Total space occupied : 50,762,016 bytes.
Number of gray nodes : 397,645
Number of black nodes: 1,040,695

Length of Chain	Number
1	43,363
2	103,427
3	127,742
4	129,386
5	137,535
6	81,659
7	52,372

Total Number of Chains: 675,484
Average Chain Length: 3.99
Average Number of Segments per Node: 4.19
Average Number of Points per Node: 5.24

Number of Points per Node	Number of Nodes
2	29,139
3	79,326
4	110,602
5	126,799
6	144,010
7	91,407
8	61,767

Table 3.1: Statistics on the PMR quadtree

longitude and not by a line that goes around the earth.

The width of the quadtree is assumed to extend from 0 to 360 degrees and the height from 0 to 180 degrees. This means that +180 degrees longitude on the earth translates to 360 degrees on the quadtree and −180 degrees longitude on the earth translates to 0 degrees on the quadtree. Correspondingly, +90 degrees latitude on the earth corresponds to 0 degrees on the quadtree and −90 latitude on the earth corresponds to +180 degrees on the quadtree. Using the above convention, it implies that points A and B lie on different ends of the quadtree; A lying close the 360 degrees and B lying close to 0 degrees. Hence, the segment connecting them has to be clipped into two before insertion; one portion of the segment extending from A's longitude to 360 degrees and the other extending from 0 degrees to B's longitude. This does not happen in the case of segments crossing over at 0 degrees longitude. Consider two points lying on either side of 0 degrees, say −1 and +1 degrees. They translate to +179 degrees and +181 degrees respectively. Since, the convention is to connect the two points so that the segment crosses over the 0 degree longitude, there is no need to split the segment.

A very simple approach is taken to compute the latitude of the cross-over segment at 180 degrees longitude as shown in Figure 3.4. AB is the cross-over segment. A has latitude θ_A, B has latitude θ_B and the cross-over point has latitude θ_E. For small lengths of AB, AA'E and BB'E can be considered to be two-dimensional triangles. Then, angle A'EA is equal to angle B'EB and if the tangents of the two angles are equated, we get:

$$r(\theta_A - \theta_E)/x = r(\theta_E - \theta_B)/y \text{ (where r is the earth's radius)}$$

$$\theta_E = x\theta_B + y\theta_A/(x+y)$$

3.2 The Graphical Editor

An editor was implemented for removing intersections and orphans. It was built using the X11 Window System, Release 4 under A/UX (Apple Unix) on a Macintosh II computer equipped with a mouse. A view of the editor is shown in Figure 3.5 with a typical pair of

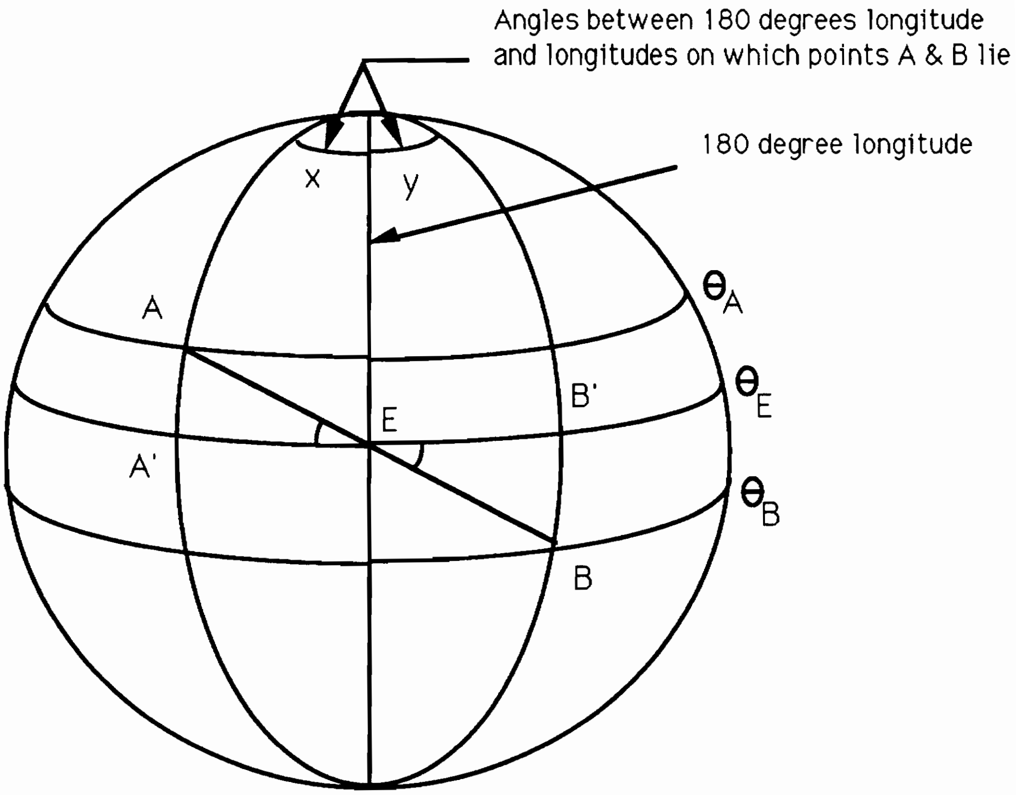

Angles between 180 degrees longitude
and longitudes on which points A & B lie
180 degree longitude
x
y
A
E
B'
A'
B
θ_A
θ_E
θ_B

Figure 3.4: Cross-Over Segments

intersecting lines. The buttons have the following functionality:

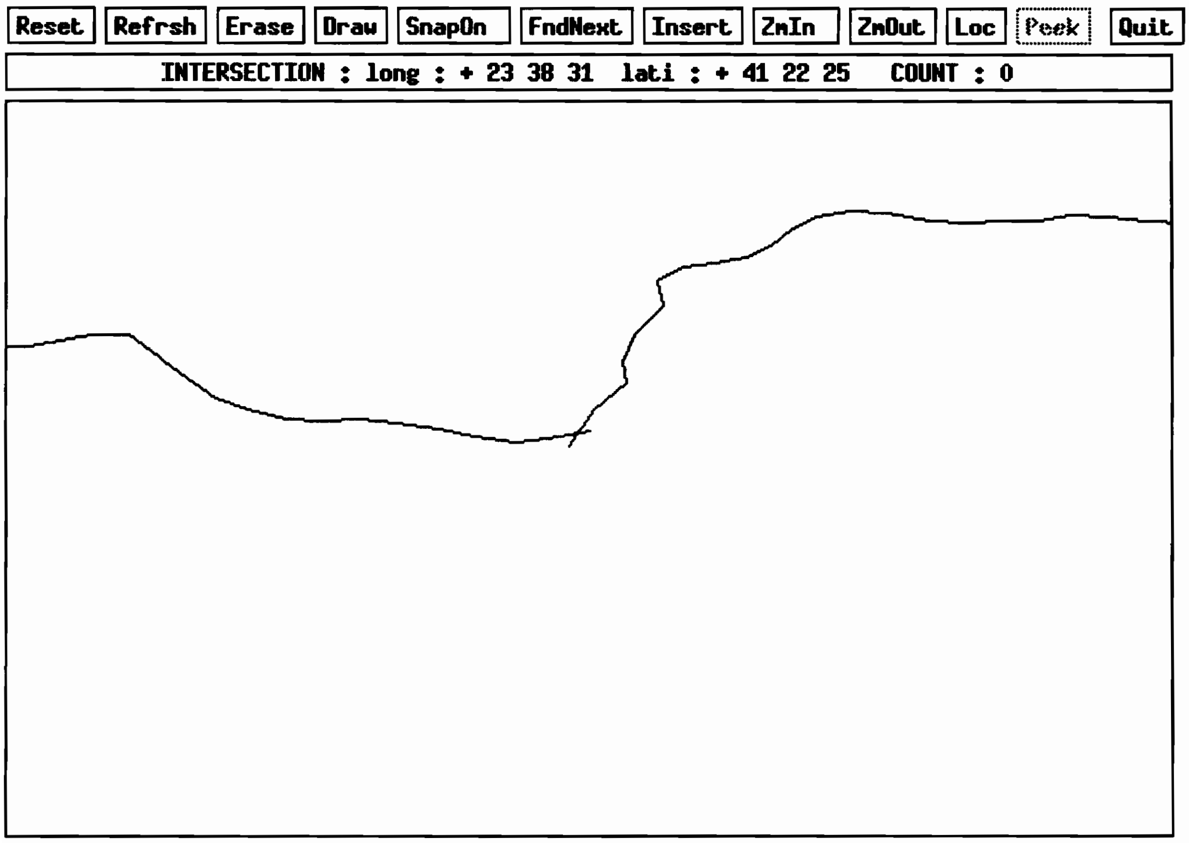
FndNext: When initialized, the editor opens a file called Intersect in which are consecutively stored, coordinates of pairs of intersecting lines. When this button is clicked, all lines in a 600×380 second window on the earth's surface (centered at the intersection point of the current pair of lines from the file Intersect) are displayed. The coordinates of the point of intersection are also displayed in the information window just below the buttons. The coordinates are in terms of latitude and longitude. If there are no more intersecting lines a message 'NO MORE INTERSECTIONS' is displayed. The program is also used to remove orphans from the database. In this case, each orphan line is stored twice in the file Intersect to make the format similar to a pair of intersecting lines.

Reset: Resets the current screen to the original state.

Refresh: Redraws the current screen but retains all modifications.

Erase: Every alternate segment in a chain is displayed on the screen is either green or red. This is done to make the editing task easy so that collinear segments and partially overlapping segments can be distinguished. When the mouse is clicked, the line closest to the cursor changes color to blue to indicate that it has been selected. Selecting a line and then pressing this button causes the selected line to be deleted from the screen *but not from the database.*

SnapOn: This button toggles between two states SnapOn and SnapOff. When the mouse is clicked within a 5×5 pixel square surrounding the endpoint of a selected line, it causes the endpoint to move along with the cursor dragging the line behind it. This is called rubberbanding. The cursor can then be used to modify the selected line by moving, stretching or shrinking it. When the mouse is clicked a second time, if the button is in SnapOn state, the endpoint snaps to the endpoint of the nearest line. If in SnapOff state, the endpoint is released at the current cursor position.



INTERSECTION : long : + 23 38 31 lati : + 41 22 25 COUNT : 0

Figure 3.5: Editor with segments

Draw: Clicking this button initiates drawing of a new line. The first click of the mouse marks the beginning of the line. The second click marks the end. Rubberbanding has been implemented so that after the first point is clicked, moving the cursor to position the second point causes a line to dynamically follow the cursor (until the second point is clicked).

Insert: This causes all lines that have been modified on the current screen to be inserted or deleted from the PMR quadtree. It is clicked once the user is satisfied that the current screen has been cleared of all orphans or intersections.

ZmIn: Zoom In. This allows the user to magnify the current set of lines on the display. Each click on the button increments the scale factor by one. The scale factor is used as follows. If it is required to scale a line by a factor of Z, the line is first translated to the origin, each coordinate is multiplied by Z, and then the line is retranslated to the original position. The translation is necessary in order to ensure that the line does not change its position, as would happen if the line were directly scaled.

ZmOut: Zoom Out. This allows the user to zoom out the current display. Each click decrements the scale factor by one until the scale factor is unity i.e., one pixel on the screen is one second on the earth's surface. If clicked once more, the display shows all lines in window of 2×1.25 degrees on the earth's surface centered at the current intersection point. This is done because at times it is not clear how best to remove an intersection on the screen and a wider context is required.

Loc: Locate. When clicked, a menu is displayed as shown in Figure 3.6. Using the scroll bars, the user selects a point on the earth's surface and a window in which to view it. The window size can be a maximum of 20×20 degrees and a minimum of 1×1 degree. Figure 3.6 shows the point selected to be 39.6 degrees latitude and 20.6 degrees longitude with a window of 20×10 . On clicking the OK button, the chosen portion of the earth's surface is displayed. This is shown in Figure 3.7. Due to the large window size, individual segments cannot be identified and so no modification of segments is allowed on this screen. However, the user is allowed to draw lines. This facility is used for drawing water body boundaries as explained in the next section. If the user selects a window that crosses +180 degrees (-180 degree) longitude, the display wraps around and the area around -180 degrees (+180 degrees) longitude is displayed.

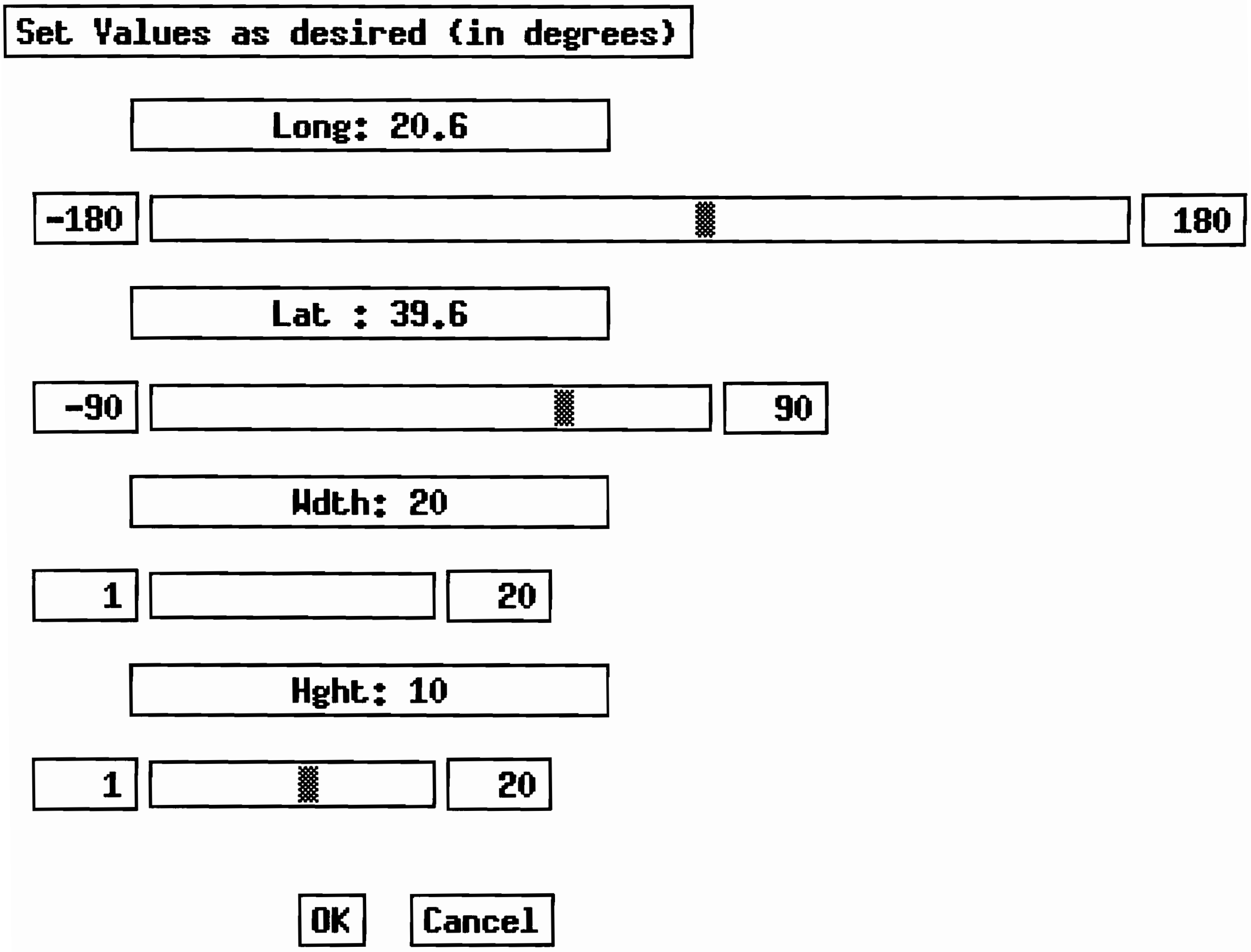
Peek: This button is activated only after Loc locates the area of interest on the screen. If the user wishes to peek at any particular area, he/she clicks the mouse near the area of interest and then clicks the Peek button. A 600 × 380 second window (as in FndNext above) around the clicked point is chosen and the segments in that window are displayed. These segments can be modified and inserted into the database. This is very useful for adding water body boundaries as explained in the next section. Also, there are country boundaries that extend beyond a land mass and into the sea (possibly to demarcate islands). These boundaries end as 'orphans' in the sea and hence the portion extending out from the land has to be removed. However, sometimes, this extended chain is too long to fit on one screen. *Loc* and *Peek* can be used in succession to remove the long chain from the database.

Quit: Quits the editor.

Though attempts have been made to close all polygons, the graphical editor will not catch a situation shown in 3.8. The line joining the two triangles does not form a legal demarcation.

3.3 Drawing Sea Boundaries

One of the features that a GIS provides is to answer the question, given a point where is its location on the map? Once the land area in the CIA WDBII is demarcated, it may be possible at some later stage to identify the country or water body where the chosen point is


Set Values as desired (in degrees)

Long: 20.6

-180 180

Lat : 39.6

-90 90

Wdth: 20

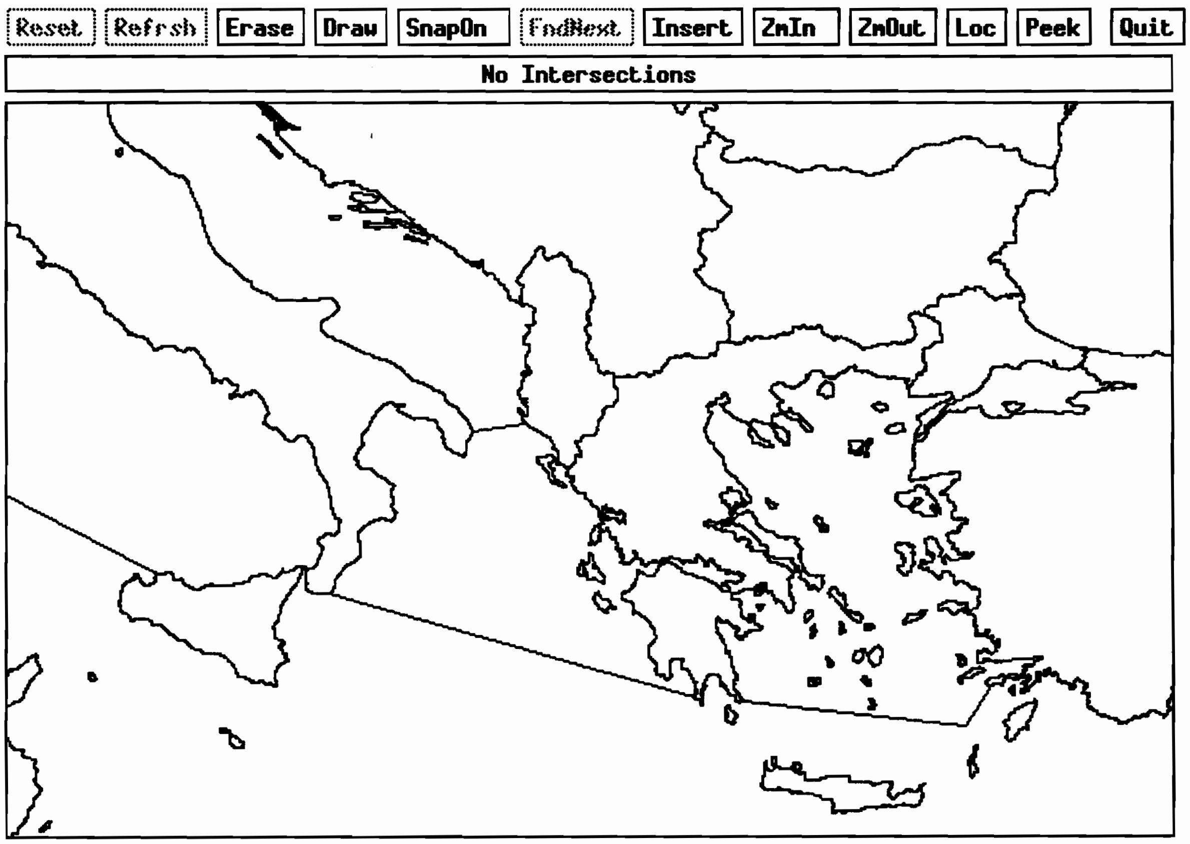
1 20

Hght: 10

1 20

OK Cancel

Figure 3.6: Menu for displaying window



No Intersections

Figure 3.7: View Around Mediterranean Obtained using 'Loc'

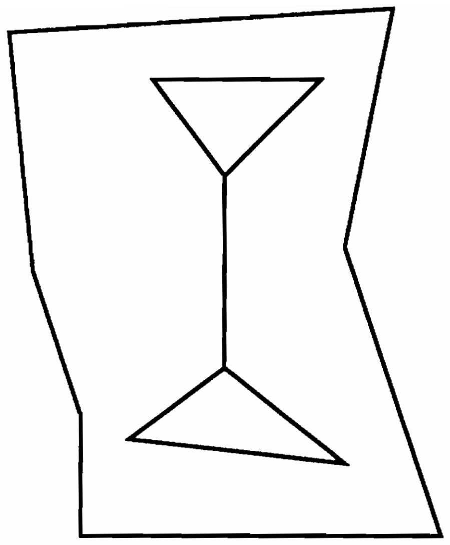


Figure 3.8: Undetectable Polygon Configuration

located. To allow the same to be done for the water bodies, some kind of demarcation needs to be made to differentiate one water body from another. Lines were inserted in the PMR quadtree to separate the water bodies as shown in Figure 3.7. The seas distinguishable here are the Adriatic Sea, Tyrrhenian Sea, Aegean Sea, Ionian Sea and Mediterranean Sea. The lines were chosen completely at the discretion of the author and are clearly approximate. The water bodies chosen are shown in Table 3.2 and are a broad separation of the earth's major water bodies. The approximate boundaries were obtained from [Coup89].

To allow more seas to be added the procedure to be followed is outlined. Using *Loc* (see Section 3.3), the general area where the water body boundary is to be inserted is located. A window of 12 $\times$ 7 degrees seems to work best. Using *Draw*, the appropriate boundary is drawn. It is important to ensure that the line is drawn as close as possible to the land mass that it is to be connected to. *Insert* is then clicked on to insert the line in the database. The next step is to connect the line to the land mass. To do this, the endpoint of the line close to the land mass is clicked on and then *Peek* is used to zoom in on the area. With *SnapOn* active, the line is connected to the appropriate segment on the land mass. It is recommended that the sea boundary not be too long (about 6-7 degrees is appropriate).

This is because when *Peek* is used to zoom in, a very long line tends to show up disoriented i.e. if, for example, the line orignally ran from east to west, after *Peek* was used, it might be seen running from southwest to northeast. To correct this problem, the zoom factor can be reduced to its minimum. This tends to reduce the difference in the scale between figures displayed by *Loc* and *Peek*.

3.4 Advantages of Using a Quadtree

Using the PMR quadtree, it is very easy to find all intersections and orphans in the database. As can be seen from Figure 3.2, to find out if a given segment touches another, it has to be checked against only those segments that lie in the quadrants that it crosses. The same rule applies for intersections. Therefore, we need only to traverse the PMR quadtree and, at each black node, compare all segments within the node against each other. As seen in Table 3.1, there are 643,050 black nodes and 4.19 segments on average per node. This implies there are about 6 intersection checks on average in each node; a total of $643,050 \times 6 = 3,858,300$ checks in all. Though 3 million checks is large, it is a small fraction of the number of checks that would have to be made if each segment was checked against every other (approximately 4 trillion).

Locating orphans requires the same traversal as before, but in each node, a count is kept of the number of segments incident on each vertex within that node. All vertices with a count of one form the endpoints of orphan line segments. From Table 3.1, the total number of vertices is calculated as 3,370,704 (due to storing end segments in every node that contains it), a number comparable to the number of intersection checks.

Another great advantage is in the speed of retrieval of segments in a given window. Given a specific window, it is a simple task to find all black nodes (and hence segments) that fall in that window. Compare this with the data structure used by the Cartographic Automatic Mapping (CAM) program developed by the CIA [Ande77]. In addition to the data set shown in Figure 2.2, CAM uses a second data set that serves as a look-up table for retrieving data

Oceans: Atlantic, Pacific, Arctic, Indian

North America

Gulfs/ Bays etc.: Gulf of Alaska, Gulf of California, Gulf of Mexico, Hudson Bay, Baffin Bay
Seas: Bering Sea, Caribbean Sea, Beufort Sea, Greenland Sea.

South America/Antartica

Seas: Bellingshausen Sea, Weddell Sea, Lazarev Sea, Riiser-Larsen Sea, Cooperation Sea, Dumont D'urville Sea, Ross Sea, Amundsen Sea

Africa

Gulfs/Bays etc.: Gulf of Guinea, Mozambique Channel, Gulf of Aden
Seas: Red Sea

Europe

Gulfs/Bays etc.: Gulf of Bothnia, English Channel, Gulf of Finland, Gulf of Riga, Bay of Biscay
Seas: Mediterranean Sea, Ligurian Sea, Tyrrhenian Sea, Adtriatic Sea, Ionian Sea, Aegean Sea, Black Sea, Baltic Sea, North Sea, Irish Sea, Sea of Azov, Norwegian Sea, Sea of Marmara

Asia

Gulfs/Bays etc.: Gulf of Siam, Bay of Bengal, Persian Gulf, Gulf of Oman, Gulf of Tonkin, Gulf of Papua
Seas: Barents Sea, White Sea, Kara Sea, Laptev Sea, East Siberian Sea, Bering Sea, Sea of Okhotsk, Sea of Japan, Yellow Sea, East China Sea, South China Sea, Sulu Sea, Celebes Sea, Banda Sea, Arafura Sea, Java Sea, Flores Sea, Molucca Sea, Seram Sea, Arabian Sea, Caspian Sea, Andaman, Solomon Sea, Bismarck Sea

Australia

Gulfs/Bays etc.: Gulf of Carpentaria
Seas: Coral Sea, Tasman Sea, Timor Sea

Table 3.2: Seas for Which Boundaries are Demarcated

from the first [Ande77]. Each record in this data set has a chain identifier, the rank of the chain, the number of points making up the chain, the minimum and maximum latitude in that chain, the minimum and maximum longitude in that chain and a pointer to the record of the first data set where the values of latitude and longitude for the given chain are stored. To plot lines in a given window, the program sequentially runs through the look-up table and compares the window coordinates against those formed by the minimum and maximum latitude and longitude. If they overlap, the program uses the pointer information to locate the chain which is to be plotted. The sequential search is clearly less efficient and only made possible at all by the second index file. There are over 30,000 chains in the portion of the database used and even more when all ranks are included. A linear search through these many records each time would be very slow compared to data retrieval through the PMR quadtree.

Chapter 4

Large Database Rasterization

4.1 Rasterization Algorithms

Most known rasterization algorithms can be divided into two categories [Peuq81]. In one, the vector file is read in sequential order and raster locations are assigned to each vector as it is encountered. In the second method, the raster file is generated on a raster by raster basis by repeatedly searching the vector file for lines that intersect the current raster. Both these techniques require extensive sorting.

If it is required to rasterize the lines in Figure 4.1, an algorithm using the first method works as follows. It is assumed that the full raster is too big to fit in main memory. Hence, the raster is divided into strips that can be read into main memory and rasterized. In Figure 4.1, the raster is divided into four such strips S_1, S_2, S_3, S_4. Lines are rasterized one strip at a time. Initially, the lines are sorted by their lower y-coordinates within the strips. Therefore, for the first strip, S_1, L_1, L_2, L_3 and L_4 are chosen. Lines L_1 and L_2 are completely rasterized. Only the portions of L_2 and L_3 within S_1 are rasterized. This process is then repeated for all the strips till all the lines are converted. Many optimizations can be made to the above process as described in [Fran79].

In the second method, the lines are first sorted by their lower y-coordinates. A linked list of the maximum number of lines that can be stored in main memory is read from the

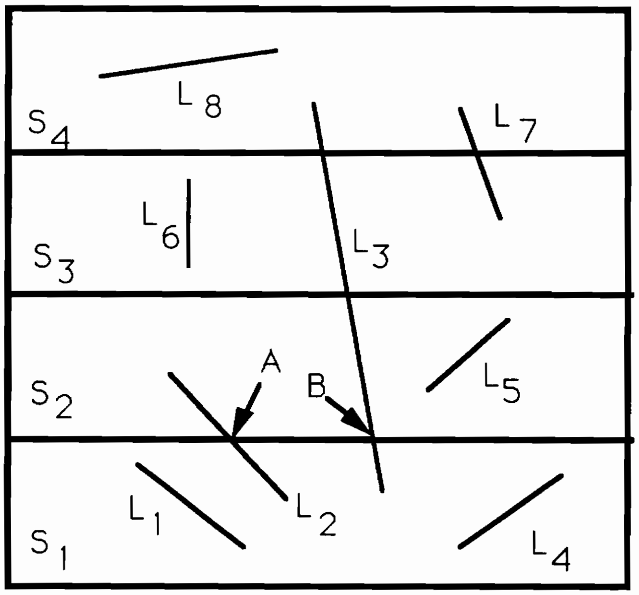


Figure 4.1: Scan Conversion Algorithm

sorted list. A raster line is moved across the lines from the bottom to the top and at each stage the points that intersect the raster line are rasterized. Therefore, for the raster line between strips S_1 and S_2, the points A and B on L_2 and L_3 are rasterized. The next batch of lines is then read until all lines have been covered. The sorted list is needed so that, depending on the position of the scan line, a decision can be made as to which vector lines are to be considered for rasterization and which are to be discarded. Optimizations to this technique can be found in [Fran79].

The next few sections outlines a new technique for rasterization using a line quadtree. The final product is not exactly a raster but a region quadtree.

4.2 The Line Quadtree

4.2.1 Original Definition

The approach used in this project to rasterize the database has not been documented earlier although it is based on the standard method of floodfilling [Fole90]. The first phase in this process involves building a line quadtree of the cleaned up database. As done for the PMR quadtree, the original definition of the line quadtree is provided and then the adaptations made for this project are described.

The line quadtree is used to store information about the boundaries of a polygonal map [Same84]. However, the boundaries cannot be arbitrary lines but have to follow along edges of pixels, the smallest resolution unit of the map. An example polygonal map is shown in Figure 4.2a. The line quadtree partitions the map via a recursive decomposition technique that successively subdivides the map until obtaining blocks (leaf nodes) with no line segments passing through their interior. A code is associated with each leaf node to indicate which of its four sides (north, south, east, west) forms a boundary of a region. Note that a quadrant can have more than one side marked. This boundary cannot be partial but has to cover the entire width of the node.

Figure 4.2b shows a complete 4-ary tree of the map in Figure 4.2a. Figure 4.2c is the line quadtree formed from the complete 4-ary tree. The leaf nodes in the line quadtree are formed by merging the nodes of the 4-ary tree. The criterion for merging is that the four siblings to be merged should have no internal borders. The merged node then has borders formed by the logical AND of the corresponding borders of its children, i.e., the north border of the merged node is formed by the logical AND of the north borders of the NW and NE children. It is also seen that the border information is hierarchical in that it is also associated with the internal nodes. Details can be found in [Same84].

4.2.2 Modification to the Line Quadtree

The information associated with the internal nodes is used for edge-following algorithms [Same84]. Since this particular application does not use this feature, the line quadtree built did not have this information built into it.

When building the line quadtree in Figure 4.2c, the method of [Same84] assumes that the complete polygonal map is available. Hence, it is possible to do the AND operation described previously. However, in the present situation, the complete polygonal map is not

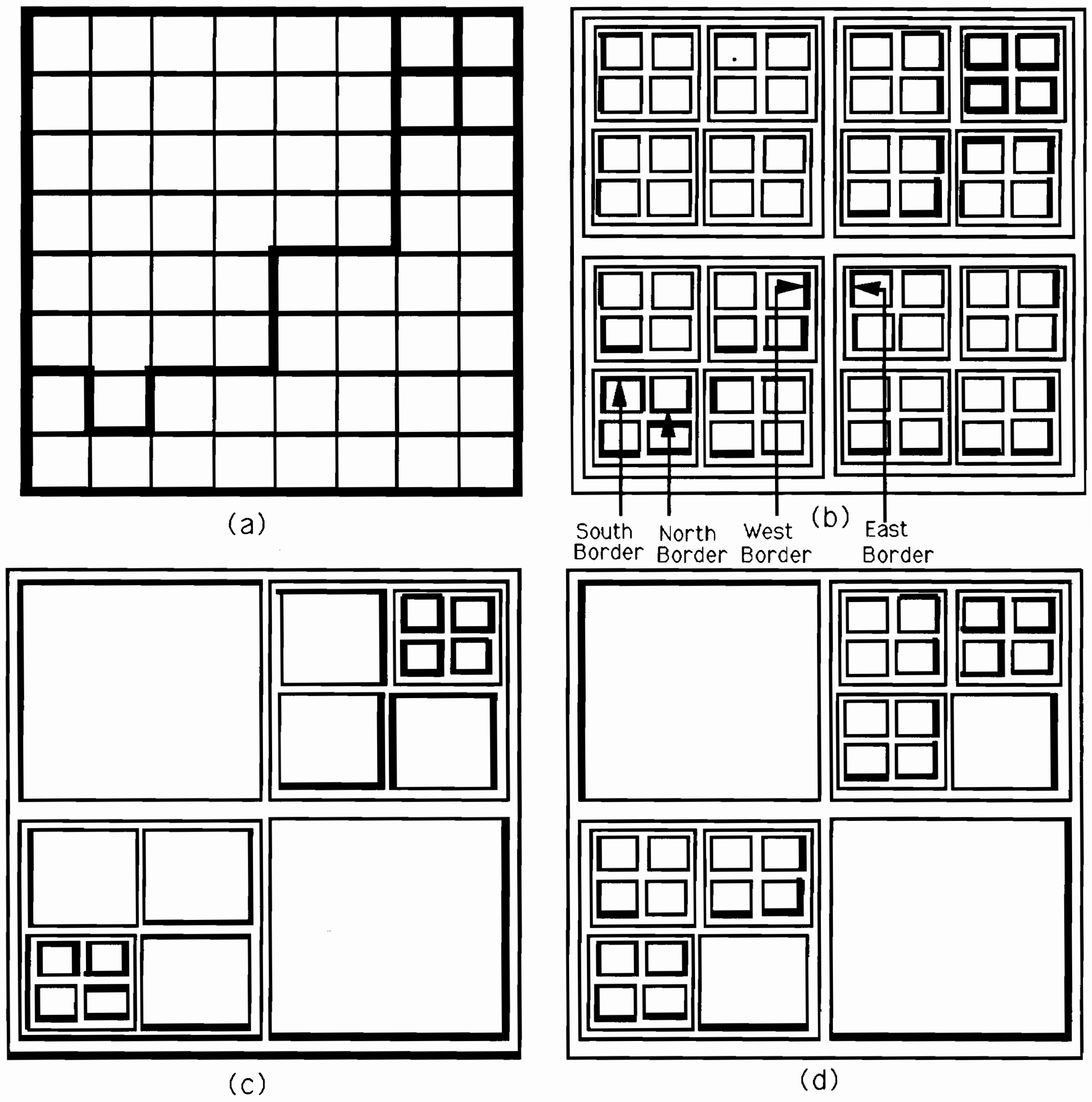


Figure 4.2: (a) Polygonal Map (b) 4-ary Tree (c) Line Quadtree (d) Modified Line Quadtree

available since the line quadtree is being built with individual lines extracted from the nodes of the PMR quadtree built earlier. The merging rule is therefore changed so that merging takes place only if the children of a parent node have complete borders with respect to the parent node.

It is seen from Figure 4.2b that for every horizontal segment of the polygonal map, a north and a south boundary is marked on the line quadtree and for every vertical segment, an east and west boundary is marked. Note that the north, south, east and west boundaries are with respect to the line segment and not the quadrant in which they lie i.e. the north border of a segment may be a south border with respect to a quadrant. If only one boundary is inserted for each kind of segment the number of nodes in the line quadtree can be considerably reduced. At the same time, there is no loss of information since the boundary of every polygon can yet be traced. It was decided to insert the east boundary for vertical segments and the south boundary for horizontal segments. Choosing to insert the other pair of boundaries is equivalent. The line quadtree formed thus is shown in Figure 4.2d. Note that the outer boundary of the line quadtree does not follow this rule since the west and the south outer boundaries do not have any corresponding east or north boundaries.

4.2.3 Scan Converting Segments

As stated earlier, segments to be inserted into the line quadtree have to follow the edges of pixels. This implies that any arbitrary line segment has to be "transformed" so that it follows the edges of pixels. This is termed as scan conversion. Bresenham's algorithm [Fole90] is used and its working is graphically illustrated in Figure 4.3. The new line can be seen to approximate the original line as a series of 'steps'.

4.2.4 Results

In order to estimate the size of the final line quadtree, the PMR quadtree was traversed in preorder and a log file was maintained to compare the growth of the line quadtree with respect to number of segments inserted from the PMR quadtree. After allowing the program

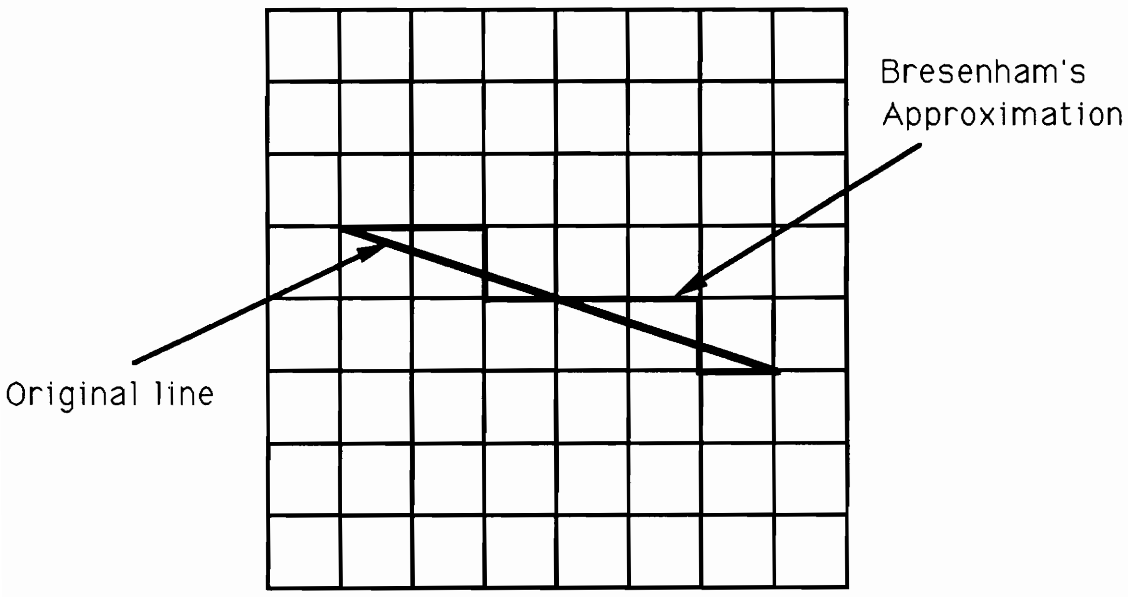


Figure 4.3: Bresenham's Algorithm

to run for about an hour, it was seen that the line quadtree was growing to be about 45 times the size of the PMR quadtree. This meant it would occupy over 1 Gbyte of memory – storage space that was not available.

To rectify this situation, it was decided to create the line quadtree at a coarser resolution. Every segment, before scan converting, was scaled down (i.e. each coordinate value was divided by 150) by a factor of 150 in both the x and y coordinates. The factor 150 was chosen because the growth of the log file was such that it indicated that the final line quadtree would occupy a managable amount of disk space. Segments which scaled to a single point were discarded. This scheme worked effectively and the final line quadtree occupied 4.8 Mbytes. Our program is designed to build the line quadtree at any desired scale factor; this scale was selected simply as a convenient test of the conversion software.

4.3 Flood Fill

In order to identify different countries, each polygon formed by the line quadtree is assigned a number associated with the appropriate country. The flood fill algorithm is a standard graphics algorithm used for this purpose. Given a region of one color bounded by a set of pixels of another color, if it is desired to flood-fill the region with a new color, the following procedure is adopted. Start at any point in the interior of the region and use the following algorithm:

```
FloodFill4(
   x, y : integer; {starting point in region}
   oldValue, {value of interior of region}
   newValue : color); {new color of region}
begin
   if ReadPixel(x,y) = oldValue then
      begin
         WritePixel(x,y, newValue);
         FloodFill4(x, y-1, oldValue, newValue);
         FloodFill4(x, y+1, oldValue, newValue);
         FloodFill4(x-1, y, oldValue, newValue);
         FloodFill4(x+1, y, oldValue, newValue);
```

```
        end
      end;
```

This algorithm was modified to fill a line quadtree polygon. The line quadtree was traversed in preorder until an uncolored node was reached. The modified flood fill was used on this node and the polygon defined by that node was colored with the current color (number). Once the line tree was completely traversed, all the polygons were colored. Figure 4.4 shows the raster version of the area in the Mediterranean seen in Figure 3.7. The actual picture uses up to 20 colors but due to the black and white printer, this cannot be seen.

When the flood fill algorithm had finished running, the color count was 12,839. This does not reflect the number of actual polygons in the vector database because the line quadtree algorithm created many small 'artifact' polygons when it rasterized segments very close together. One such case is illustrated in Figure 4.5. This problem occurs because the resolution of the database is 1 sec but the resolution of the raster is 150 seconds. This can be looked upon as an aliasing problem and needs an anti-aliasing solution. At the time of this writing, no solution to this problem was obtained.

4.4 The Region Quadtree

The final stage was to create a region quadtree (as described in Section 3.2.2) from the line quadtree. This procedure involved removing all boundary marks and merging all nodes of one color as far as possible. For example in Figure 4.2d, both the northwest and northeast children of the southwest child of the root can be merged into one node (as seen in Figure 4.2c) because they are 'colored' the same. When this was done the tree occupied 3.6 Mbytes on disk, compared to the raster which occupies 4.6 Mbytes. This saving of 21% is obtained with the data set scaled down by a factor of 150. For larger datasets this saving will correspondingly improve.

The advantage of having the data in a region quadtree rather than a raster is that it

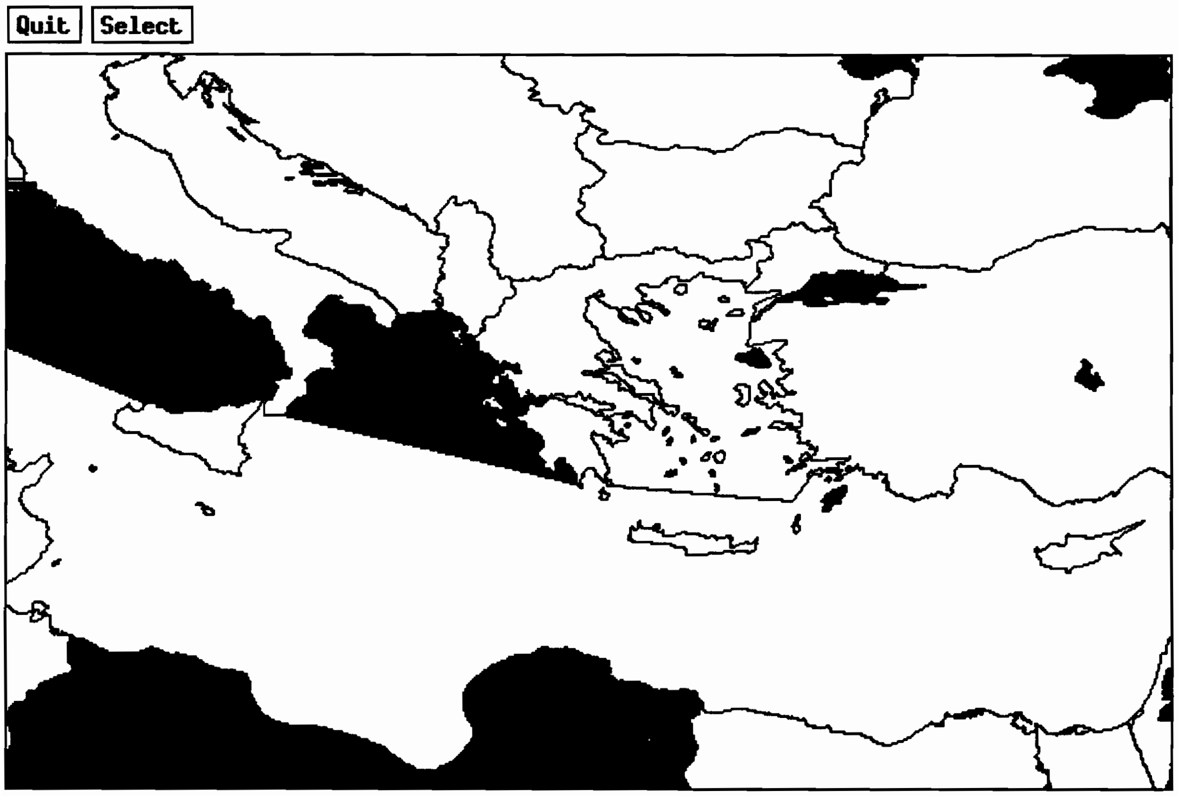


Figure 4.4: View Around Mediterranean Obtained by Line quadtree

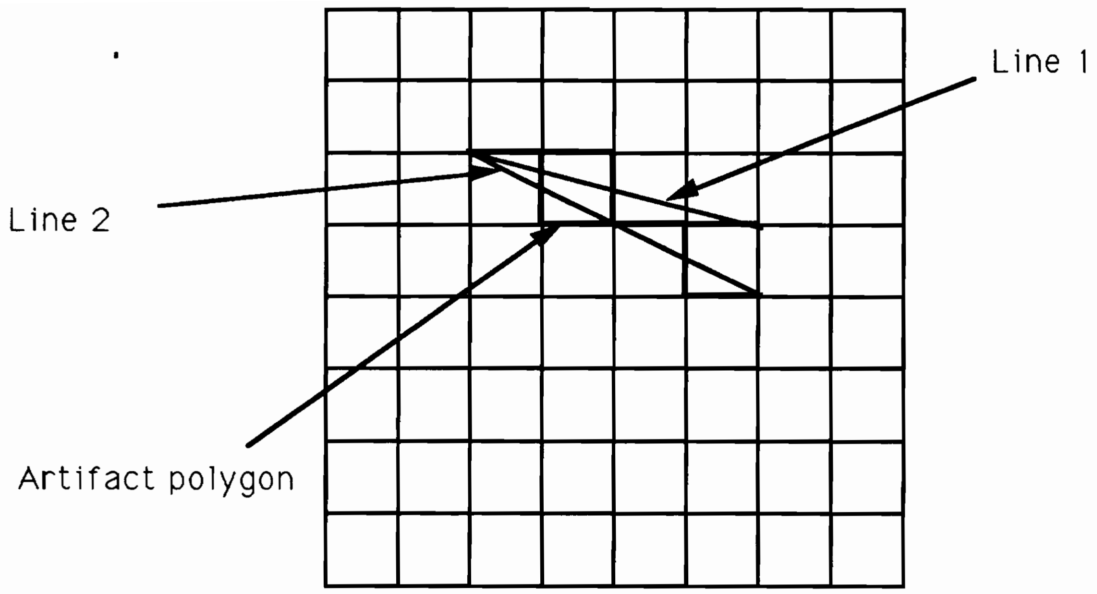


Figure 4.5: Formation of Undesireable Polygons

is easy to assign a label to each 'color'. The label identifies the corresponding land or sea body in the real world. The program to do this has not been written but its functionality is as follows. The user would display a certain portion of the earth's surface. He/she would then click the mouse on each of the polygons displayed in turn. Each mouse click would call up a box which prompts the user for a name for the polygon clicked on. The mouse click is also used to locate the 'color' in region quadtree. The 'color' and the name are then associated together. This process is continued until all the polygons have been assigned a label.

Chapter 5

Conclusions and Future Work

5.1 Summary

An effective technique to convert a large volume of vector data to raster data has been established. The technique is outlined in Figure 5.1. This technique is superior to the standard edge filling algorithm since it does not require extensive computation or storage even for datasets as large as 2 million line segments.

The first step in the work flow is to insert the vector data into a PMR quadtree. After the program to do this had completed, the tree occupied over 50 Mbytes of disk space. The program ran for 37 hours on a Mac II under A/UX. The tree had 1,438,340 nodes of which 397,645 were internal nodes. There were, on average, 4.19 segments per node and 5.24 points per node.

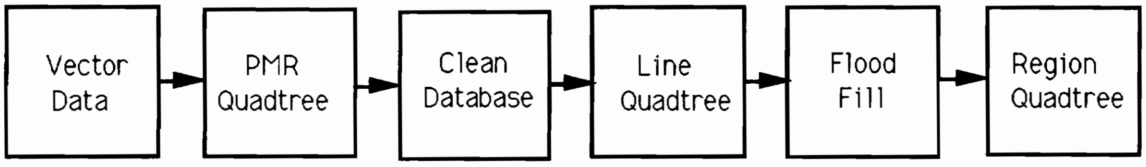


Figure 5.1: Workflow for Rasterization

A graphical editor was then built to remove all intersections and orphans from the PMR quadtree. In addition, approximate boundaries were inserted to demarcate the major water bodies on the earth's surface. The clean linework was then inserted into a line quadtree. Before insertion, each line segment was scaled down by a factor of 150 (i.e. the resolution of the map was made 150 seconds instead of 1 sec). The scaling factor was chosen so that the line quadtree would not occupy too much space on disk. However, if finer resolution is required and storage space is available, this factor can be reduced. The line quadtree occupied 4.8 Mbytes of disk space. The time complexity for building the line quadtree is O(sum of lengths of lines)$\times$ depth. The sum of lengths is the perimeter of the map and by Hunter's theorem [Same89b], O(perimeter) = number of nodes in the quadtree. The depth of the quadtree is the resolution of the map. Therefore, the complexity can be said to be the (number of nodes) $\times$ resolution. A flood fill was then performed on the line quadtree. The time complexity for flood is $4 \times$ depth $\times$ number of nodes (the factor 4 is because 4 adjacent neighbours are checked each time during the flood-fill operation) which implies that the complexity is the same as that for building the line quadtree. The flood-filled tree was then made into a region quadtree. The region quadtree occupied 3.6 Mbytes of disk space. This is 21% less space that an equivlent raster which would occupy 4.6 Mbytes.

Labeling of the polygons on the earth's surface has not been done as yet but the methodology to be followed is outlined in Section 4.4.

A disadvantage with this method is that the rasterization tends to be slightly erroneous if the segments in the vector data are very close together. The errors might be corrected by using some heuristics but this is left as future work.

Future work in this effort is to create a geographical database at different resolutions ranging from the scale of the world to that of individual cities and arranging it in the form of a hierarchy. The user of the GIS will have the capability to zoom in on any desired portion of the earth's surface and query the database.

5.2 An Alternative Approach

Another objective of creating a database free from gaps, slivers, orphans and intersections is to make possible point-in-polygon searches and boundary following. In order to do this we need to identify the segments belonging to each country and store them separately. This can be done as follows. A small section of the map is displayed using the graphics editor. The mouse is clicked on the displayed area. The coordinates of the clicked point are translated to the corresponding point on the earth's surface and the segment that lies closest to the point is identified. This segment is marked as visited. The right-hand most segment in the clockwise direction is then visited and marked. This process is repeated till the original segment is reached again indicating the polygon is complete. A check is also kept of the cumulative turning angle as the boundary is followed. If, when the polygon is complete, this is not 360 degrees, an error is flagged. A number is then associated with each completed polygon which is later translated to a country name.

The problem with this approach is that it is difficult to identify when all polygons have been completed. In order to do this a raster could be used to keep track of the progress. Each time a polygon boundary is traversed, the corresponding portion on the raster is filled in. To identify if an area is not yet traversed, the raster can be searched. The problem with this is that all polygons contained within other polygons (e.g., lakes in countries) need to be traversed first otherwise when the outer polygon is traversed, the raster would fill in the inner polygon as well thereby causing information loss. Since there is no specific information on all lakes in the database (all lake information has islands or coastlines included in it), it would be a difficult task to ensure that they have all been identified first.

Burrough [Burr86] suggests the following approach. First follow the outer-most boundary of each independent polygon unit. This is done as before but the left-hand most segment in the clockwise direction is chosen during the boundary traversal. After this, all the remaining polygons are followed as done before. A tally is kept of the number of times a segment is visited. When this reaches two, that segment is no longer considered. This approach however assumes that the polygons are not in their proper topological hierarchy, i.e., if there is a polygon embeded within another polygon, the inner polygon will be available as a separate unit to be followed independently. The hierarchy in Burrough's approach is established after all polygons have been traversed.

Due to the above reasons, this may be a difficult task to perform.

REFERENCES

[Ande77] D. E. Anderson, J. L. Angel, A. J. Gorney, World Data Bank II: Content, structure, and application, *Advanced Study Symposium on Topological Data Structures for Geographic Information Systems*, (October 1977), 16-21, Harvard University Laboratory for Computer Graphics and Spatial Analysis.

[Burr86] P.A. Burrough, *Principles of Geographical Information Systems for Land Resources Assessment*, Clarendon Press, Oxford, 1986.

[CIA77] Central Intelligence Agency, *World Data Bank 11 General Users Guide*, PB-271 869, July, 1977.

[Coup89] A. Couper (Editor), *Atlas and Encyclopaedia of the Sea*, Times Books, London, 1989.

[Espe88] E.B. Espenshade, Jr.(Editor), J.L. Morrison (Consultant), *Goode's World Atlas*, Rand McNally and Company, Chicago IL, 1988.

[Fole90] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*, Addison Wesley Publishing Company, Reading MA, 1990.

[Fran79] W. R. Franklin, Evaluation of Algorithms to Display Vector Plots on Raster Devices, *Computer Graphics and Image Processing*, 11,4 (December 1979), 377-397.

[John89] E.F. Johnson, and K. Reichard, *X Windows Applications Programming*, MIS Press, Portland OR, 1989.

[Nels86] R.C. Nelson and H. Samet, A consistent hierarchical representation for vector data, *Computer Graphics 20*, 4(August 1986), 197-206.

[Nye90] A. Nye, and T. O'Reilly, *The Definitive Guides to the X Window System - Volumes 1-5*, O'Reilly and Associates Inc., Sebastopol CA, 1990.

[Peuq81] D. J. Peuquet, An examination of techniques for reformatting digital cartographic data/part 2: The vector to raster process, *Cartographica*, 18,3, 21-33, 1981.

[Same89a] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison Wesley, Reading MA, 1990.

[Same89b] H. Samet, *Applications of Spatial Data Structures*, Addison Wesley, Reading MA, 1990.

[Same85] H. Samet and R.E. Webber, Storing a collection of polygons using quadtrees, *ACM Transactions on Graphics 4*, 3(July 1985), 182-222.

[Same84] H. Samet, R.E. Webber, On Encoding Boundaries with Quadtrees, *Pattern Analysis and Machine Intelligence*, 6,3 (May 1984), 365-369.

[Will81] B.M. Willett (Editor), *Philips' International Atlas*, George Philip and Son, London, 1981.

[Youn90] D.A. Young, *The X Window System Programming and Applications with Xt*, Prentice-Hall Inc., Englewood Cliffs NJ, 1990.