REPORT DOCUMENTATION PAGE

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
23-JAN-2003	Conference Proceedings, (refereed)	

4. TITLE AND SUBTITLE

Designing And Building A Vector Feature Database

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62435N

6. AUTHOR(S)

MARLIN GENDRON   STEPHANIE S EDWARDS   GEARY LAYNE

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
74-6636-C3

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Marine Geoscience Division
Stennis Space Center, MS 39529-5004

8. REPORTING ORGANIZATION REPORT NUMBER

NRL/PP/7440--03-1005

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

10. SPONSOR/MONITOR'S ACRONYM(S)

ONR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release,distribution is unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

High-resolution imagery can be stored on the computer in digital form as a picture, for example, a digital raster map image file. These images then can be geo-registered by computing coefficients from points with known latitude and longitude locations. Features such as roads and airports can be extracted by applying image-processing techniques to the geo-registered raster image. Attributes describing these features and their geographical locations are stored in a "vector feature database". The vector feature database contains many feature types and is considered accurate to a given map scale. In a real-time processing system there is a need to input attributes and their locations and subsequently retrieve such feature attributes from the database with minimum processing time. The overall size of the database is also a consideration. This paper explores the design and construction of a vector feature database to 1) optimize the size of the database by reducing the number of attributes while still maintaining an adequate and unique description of the feature, and 2) enable high-speed input and retrieval of features. Several data structures that might be used to construct the database are discussed, including hash tables, binary-trees, quad-trees, and R-trees. Ultimately a quad-tree structure modified to use geographic bitmaps is implemented and evaluated.

15. SUBJECT TERMS

satellite imagery, geo-registered, image processing, raster image, vector map

16. SECURITY CLASSIFICATION OF:

a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a NAME OF RESPONSIBLE PERSON
unclassified	unclassified	unclassified	Unlimited	9	Marlin Gendron

19a NAME OF RESPONSIBLE PERSON
Marlin Gendron

19b. TELEPHONE NUMBER (Include area code)
228-688-4773

Standard Form 298 (Rev. 8/98)

20030416 336

Designing and Building a Vector Feature Database

Marlin Gendron, Naval Research Laboratory
Stephanie Edwards, Naval Research Laboratory
Geary Layne, Naval Research Laboratory

Abstract

High-resolution imagery can be stored on the computer in digital form as a picture, for example, a digital raster map image file. These images then can be geo-registered by computing coefficients from points with known latitude and longitude locations. Features such as roads and airports can be extracted by applying image-processing techniques to the geo-registered raster image. Attributes describing these features and their geographical locations are stored in a "vector feature database". The vector feature database contains many feature types and is considered accurate to a given map scale. In a real-time processing system there is a need to input attributes and their locations and subsequently retrieve such feature attributes from the database with minimum processing time. The overall size of the database is also a consideration. This paper explores the design and construction of a vector feature database to 1) optimize the size of the database by reducing the number of attributes while still maintaining an adequate and unique description of the feature, and 2) enable high-speed input and retrieval of features. Several data structures that might be used to construct the database are discussed, including hash tables, binary-trees, quad-trees, and R-trees. Ultimately a quad-tree structure modified to use geographic bitmaps is implemented and evaluated.

Acknowledgments

Raster and Vector Databases

Maps come in many forms, such as nautical, aerial, and topographic. Map can be derived from satellite and underwater imagery. These maps are usually stored digitally as raster image files or vector databases. Raster maps are simply stored in the computer as pixels and usually originate from paper charts scanned into the computer or directly from satellite or sonar imagery. Vector feature databases (VFDb) exist as independent feature attributes such as roads, airports, and descriptive text that can be overlaid and combined to form a readable map. Features in a VFDb exist separately from each other and can be added to or subtracted from the map. Features in raster map images can only be pulled out with special feature detection algorithms. These algorithms are often complex and CPU intensive.

When pixels in the raster image or points on a vector feature correspond to points on a surface, such as the Earth, the raster image or vector feature is geo-rectified or geo-registered. Several map files stored together in a structured way form a map database. Digital raster map databases exist at many different resolutions, or map scales, and define how individual map files can be pieced together to form complete maps (Lohrenz, 1991).

The geo-rectifying process occurs by taking control points, at known latitude and longitude points on the Earth, to generate coefficients. These coefficients are used to compute new x and y positions in the image. The number of control points and their precision determines the accuracy of the raster map. The geographic extent of the image and its size in pixel space determines the map scale of the image (Hager, et al., 1990; Landrum, 1989).

Figures 1 and 2 show grayscale raster map image files that have been geo-rectified. Figure 1 shows a satellite image file depicting a highway interchange. Figure 2 shows an geo-rectified file of sidescan sonar imagery with a bright spot and shadow of a bottom object marked by a detection algorithm.

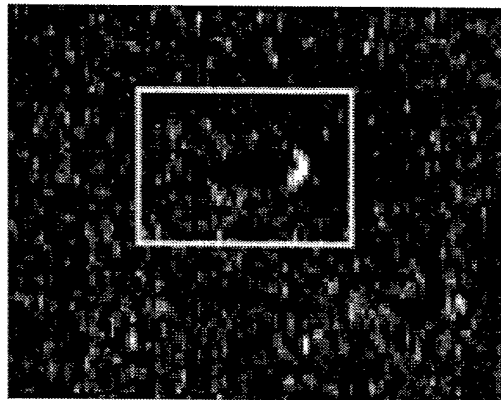Figure 1 - Geo-Registered Satellite Image (Raster)

Figure 2 - Geo-Registered Sonar Image (Raster)

Figure 3 shows the highway interchange extracted from the satellite imagery and the shadow and bright spot extracted from the sonar imagery. Two different feature detection algorithms were used to accomplish the extraction and produce the figure.
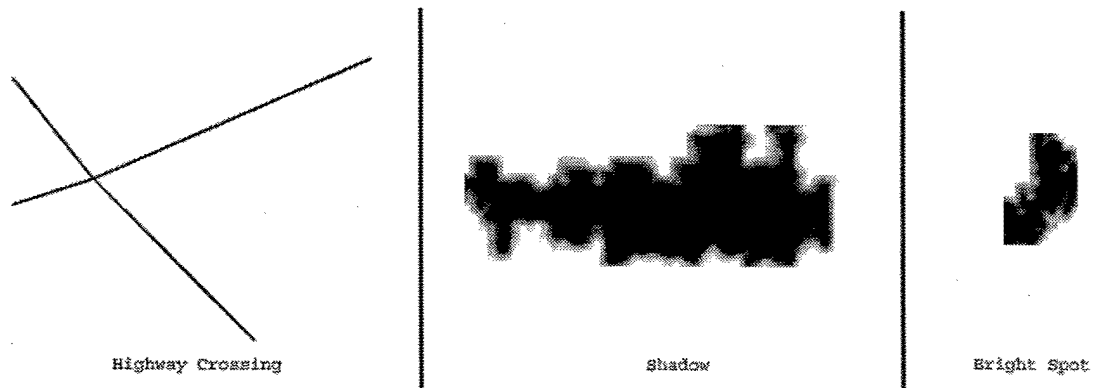
Highway Crossing   Shadow   Bright Spot

Figure 3 – Examples of Features Extracted From Raster Images

High-resolution imagery (e.g., one-meter) can cover a large area and contain many image files. The features extracted from the imagery can be abundant and difficult to manage. One solution is to store these features in a VFDb and provide a fast and efficient means to retrieve them. This paper presents a method for creating a vector database of features and populating it with one and two-meter features. The features are stored as a compact two-dimensional representation or picture that reduces the amount of attribute information required to describe the object. A method is discussed in detail to query the database and retrieve the features that fall within a geographic area of interest (AOI) in the form of a polygon similar to that shown in Figure 4.
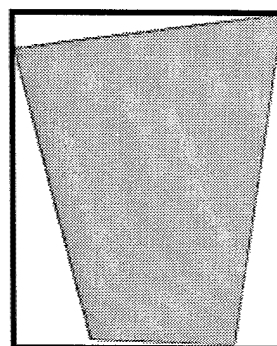
■ Minimum Bounding
  Rectangle (MBR)

▨ Area of Interest (AOI)
  Polygon

**Figure 4 – Geographic Area of Interest
around 50m by 30m**

Data Structures

Various data structures can be considered when building a VFDb, such as hashtables and binary-trees. The hashtable maps a hash key to a value in memory without searching through every element to retrieve a desired element. A hashing function generates an index number into the element array from a hash key. Collisions occur when two different hash keys generate the same index. The first element hashed is stored at the index location, and subsequent

elements hashed to the same location are nearby (often just by adding one to the index) and retrieved later in the same fashion.

Trees data structures are naturally efficient at representing hierarchical data. Rooted trees are comprised of a set of nodes (vertices) and a set of arcs (edges) that link a parent node to one of its child nodes. Each root node has one parent. Any node can be reached by following a unique path of arcs from the root. When vertices are interconnected, the tree is considered bi-directional.

Binary-trees are the simplest kind of trees and each parent node has at most two children. Binary-trees and hashtable data structures are only useful for one-dimensional data. Two-dimensional data, like maps and feature data, are better managed with data structures like quad-trees and R-trees. A quad-tree is similar to a binary tree, but parent nodes have at most four vertices.

R-trees combine the best features of binary-trees and quad-trees to efficiently store and retrieve two-dimensional data (Roussopoulous, Kelly and Vincent, 1995). A R-tree structure is an excellent choice to store the map feature data and provide an efficient method to retrieve features inside an AOI, but this paper will present a unique method where a quad-tree structure is modified to use geographic bitmaps (GB) to store features on a storage device. This structure will allow quick retrieval of features within the AOI and, at the same time, allow for the creation of a binary feature map.

Geographic Bitmaps

Bitmaps are two-dimensional binary structures in which bits are turned on (set) or off (cleared) and the row and column of each bit gives it a unique position. This concept is extended to construct geographic bitmaps (GB), where every bit represents a unique location on the Earth at a given map scale. Set bits denote that data exists (each bit represents a specific coordinate on the Earth). Although the GB is defined for the entire world at a given map scale, memory is only allocated dynamically when groups of spatially close bits are set. This makes the GB a fast and compact data structure (Gendron, et al., 1997).

For this paper, a VFDb is designed so features are stored in tiled GBs without the need to store the actual map images. This greatly reduces the overall size of the database. A variable tiling scheme based on feature resolution is defined for feature GBs at each level. The feature GBs can contain more than one feature and some features can span more than one GB. Only bits that are part of a specific feature are set. Each feature representation is distinguished by its unique central latitude/longitude value. A binary map of the features that fall within an AOI is created by logically "ANDING" GBs within the AOI. From the binary map, vectors can later be formed from the features and related to feature attributes stored separately in the VFDb.

Vector Feature Database Using Geographical Bitmaps

The VFDb presented in this paper is created by first modifying a quad-tree data structure to use GBs, thereby reducing the search time. The database stores the feature data below one-degree cell directories (in subdirectories) on a storage device in a quad-tree hierarchy based on the feature's central latitude and longitude location. For example, a feature with the center latitude/longitude value of 10.34… degrees latitude and 65.09… degrees longitude would be stored below the 10N065E directory on disk.

To quickly determine which degree cell directories exist, a GB of the entire world is created and stored on disk at the same level as the cell directories. A set bit in the GB indicates the corresponding one-degree cell directory exists on disk. For this paper, only features below 50 degrees latitude and above –50 degrees latitude are stored in the database. The size of the world GB is 100 rows by 360 columns:

Rows = 50 – (-50); Total Latitude
Columns = 180 – (-180); Total Longitude
Resolution = 1 degree of latitude and longitude per bit

The subdirectory structure below a one-degree cell directory is a quad-tree structure with five levels (Figure 5). The VFDb testing for this paper contains two-meter feature data on level four and one-meter feature data on level five.
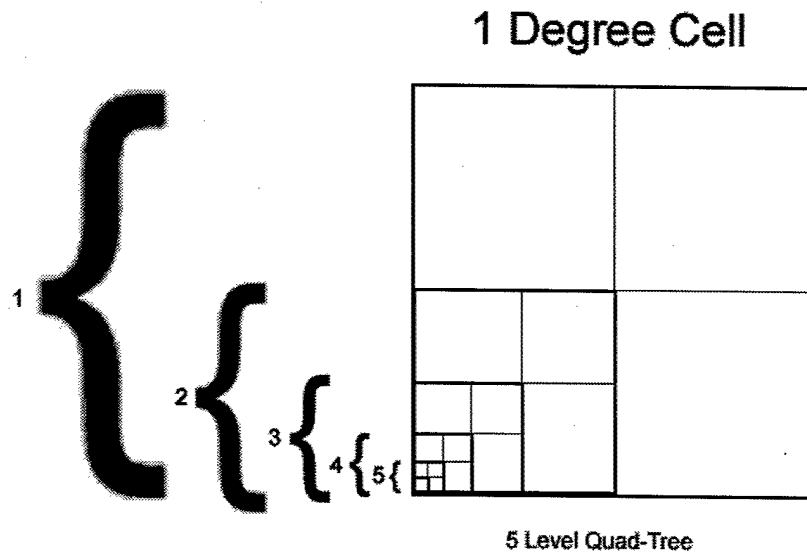
1 Degree Cell

5 Level Quad-Tree

Figure 5 – Quad-tree

To determine which of the four nodes (subdirectories) a latitude/longitude point (Y,X) falls within on any given level is found by comparing it with the central latitude/longitude point (CY,CX) of each node at that level, as follows:

Node One:	$X \geq CX$ and $Y \geq CY$
Node Two:	$X \geq CX$ and $Y < CY$
Node Three:	$X < CX$ and $Y \geq CY$
Node Four:	$X < CX$ and $Y < CY$

To avoid searching down the entire quad-tree subdirectory structure from node to node to determine if features are available for an AOI, GBs are stored on level one to indicate which one-meter and two-meter features exist inside level four and level five subdirectories, respectively. The one-meter GBs are 16 rows by 16 columns found by:

$$\text{Rows} = 2^{(\text{level-1})}$$
$$\text{Columns} = 2^{(\text{level-1})}$$

with a maximum of 4 GBs (one for each subdirectory node on level 1).

In terms of latitude and longitude:

Rows = delta latitude / latitude resolution
Columns = delta latitude / latitude resolution

where

delta latitude = 1.0 / $2^{(\text{top level})}$; top level = 1
delta longitude = 1.0 / $2^{(\text{top level})}$; top level = 1

and

$$\text{Latitude resolution} = 1.0 / 2^{(\text{level-1})}$$
$$\text{Longitude resolution} = 1.0 / 2^{(\text{level-1})}$$

Each one-meter feature is stored at level five in a GB. The size of the GB is variable, depending on its location on its latitude/longitude location. GBs closer to the equator are wider than those further away. The total height and width of a node at level five is:

Total Height = height of level 5 (in meters) / latitude resolution; latitude resolution = 1 m
Total Width = width of level 5 (in meters) / longitude resolution; longitude resolution = 1 m

The size of the one-meter feature GB on level 5 is found by:

Bitmap height = Total Height / Number of desired tiles
Bitmap Width = Total Width / Number of desired tiles

The height and width in meters is calculated as the great circle distance (WGS-84 datum) of the node. For example, a level five directory at the lower left corner of the 00N000E one-degree cell is 3455.45m high and 3478.73m wide. A feature GB in that directory would be 431 bits high by 434 bits wide, if the number of desired tiles is 8.

Similarly, two-meter resolution feature coverage is indicted by a maximum of 4 (8 by 8 bitmaps) stored at level one for each of the four nodes of level one. The total height and width of a node at level four would be 6910.89m by 6910.89m, and would contain two-meter feature GB's of the size 431 bits high by 434 bits wide. Note that this is the same size as the one-meter feature GBs, even though the area is twice as large. This is because the resolution of a feature at two-meters is exactly half that at one-meter. Figure 6 shows an example of the VFDb.
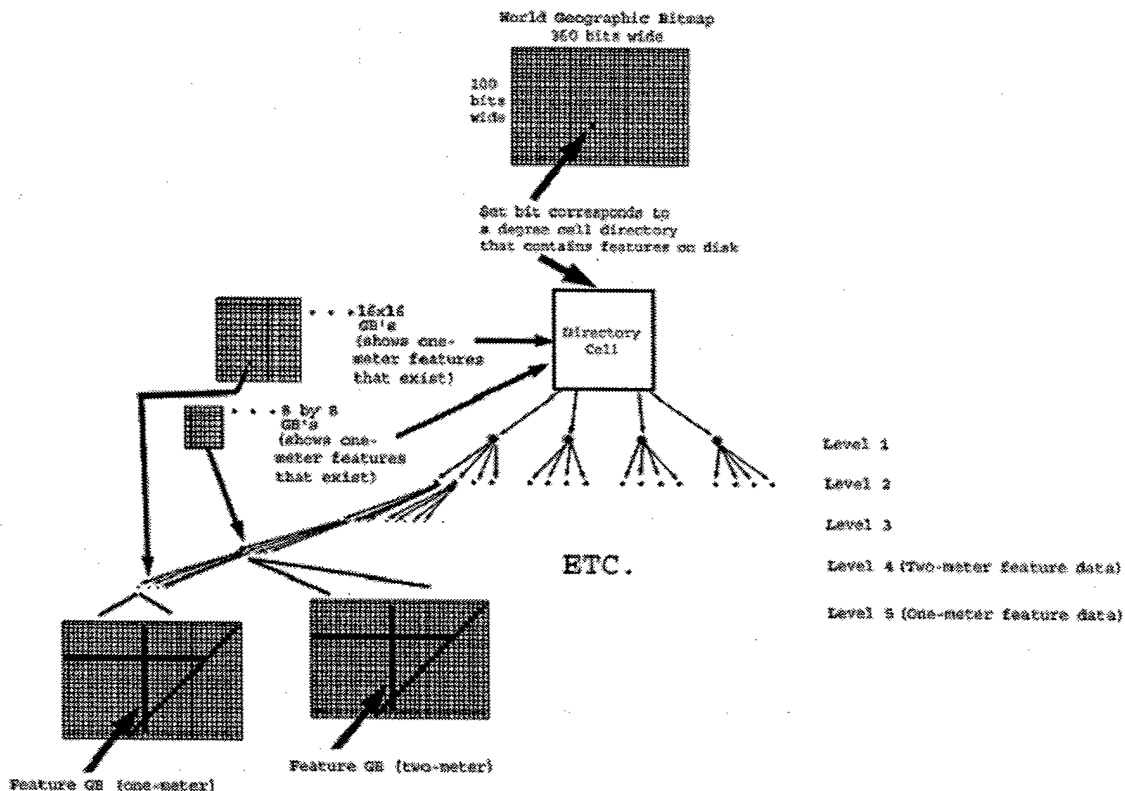

Figure 6 – VFDb Structure

Querying the Database

To query the VFDd and find all one-meter features that fall within the defined AOI (Figure 4), a GB of the MBR size at one-meter resolution is created and bits are set that geographically correspond to the AOI polygon (Figure 4). A software routine capable of draw and filling a polygon inside a GB is used to set the appropriate bits. These routines accept the latitude and longitude of the polygon vertices as arguments.

The degree cells that the MBR falls within are then computed and each of these degree cells is processed one at a time. The degree cell GB is read to determine if the degree cells to be processed exist. The one-meter AOI GB is logically ANDED with all the one-meter 16 by 16 GBs on level one. For each bit match, the corresponding level five directories are read and all feature GB stored there are ANDED with the AOI GB. By repeating this process for each set bit and each GB at level one, a feature binary map of one-meter features is formed that span the AOI. Note that features may or may not be available for all or some of the AOI. The whole process is repeated to find two-meter features on level four by first creating a AOI GB with two-meter resolution and ANDING it with the 8 by 8 two-meter GB's on level one.

The following is pseudo code of the query process to create a binary map for one-meter data:

```
AOI_bitmap = Geo_Bitmap_Create (minlat, maxlat, minlon, maxlon, one-meter resolution...)
World_bitmap = Read_World_Bitmap ()
WHILE (degree_cell = Determine_Degree_Cells(AOI_bitmap))
        If (Bit_Is_Set (World_bitmap, degree_cell))
                For (All one-meter GBs at level 1 of the degree_cell)
                        available_data_bitmap = And_Bitmap (AOI_bitmap, one-meterGB)
                        If (Bit_is_Set (available_data_bitmap))
                            Node = Determine_Node ()
                            Read_Node_Directory (Node)   /* At level 5 */
                            Binary_map = AND_Bitmap (Node_GB, AOI_bitmap)
                        End
                End
        End
End
```

Attributes

Each feature in the VFDb has a unique central latitude and longitude value, or simply the center of the MBR. The central latitude/longitude value is tied to a separate file that contains attributes similar to those listed below and is dependent on the source map data.

Examples of feature attributes are as follows:

Attributes:
- **Feature Number (1, 2, 3...)**
- **Feature Type (Highway, Airport, Sand Ripple, Rock)**
- **Bounding Polygon (Vertex Number, Latitude/Longitude...)**
- **Heading**
- **Points within the Feature (#, latitude/longitude location...)**

Conclusion

This paper presents a method of building a vector database that allows quick retrieval of features at many different resolutions. A binary feature map is easily constructed from retrieved features within a geographic AOI. Using powerful feature detection algorithms to build GBs of features and extract attributes is one primary means to

populate the VFDb. The same algorithms can be modified to detect similar features over an AOI and match these features with those stored in the database.

Further implementation, evaluation and testing of the GB VFDb is currently being done. This includes extending the database above and below +/-50 degrees latitude. This extension is possible because the current GB data structure supports polar projections. Work also continues on algorithms that will produce feature vectors from the resulting binary map of retrieved features. The combination of robust feature detection algorithms, both for post and real-time processing, the VFDb, and feature matching algorithms could have many applications, such as supporting navigation systems for autonomous vehicles.

References

M. L. Gendron, P. B. Wischow, M.E. Trenchard, M.C. Lohrenz, L.M. Riedlinger, M.J. Mehaffey (1997). "Moving Map Composer," Naval Research Laboratory, US Patent No. 6,218,965 B1, 1-29.

J. Hager, L. Fry, S. Jacks, and D. Hill (1990). "Datums, Ellipsoids, Grids, and Grid Reference Systems," Defense Mapping Agency, TM8358.1.

J. Landrum (1989). "Illustration of the Relationships Among Database Resolution, Map Scale, and Display Technique," Naval Ocean Research and Development Activity, NORDA Technical Note 439, 1-7.

M. Lohrenz (1991). "The Navy Standard Compressed Aeronautical Chart (CAC) Database," Naval Research Laboratory, SP 024:351:91, 1-93.

N. Roussopoulos, S. Kelley and F. Vincent (1995). "Nearest Neighbor Queries," SIGMOD 1995, San Jose, CA.