11-30-1993

An interface between the GRASS geographic information system and ORACLE relational detabase management system

David Gordon Buker
Florida International University

DOI: 10.25148/etd.FI14051876

FLORIDA INTERNATIONAL UNIVERSITY
Miami, Florida

An Interface Between the GRASS Geographic Information System and ORACLE Relational Database Management System

A thesis submitted in partial satisfaction of the requirements for the degree of Master of Science in Computer Science

by

David Gordon Buker

1993

To Professors Nagarajan Prabu, Wei Sun, and Naphtali Rishe:

This thesis, having been approved in respect to form and mechanical execution, is referred to you for judgement upon its substantial merit.

Dean Arthur W. Herriott
College of Arts and Sciences

The thesis of David Gordon Buker is approved.

Nagarajan Prabu

Wei Sun

Naphtali Rishe, Major Professor

Date of Examination: November 30, 1993

Dean Richard Campbell
Division of Graduate Studies

Florida International University, 1993

DEDICATION PAGE

This thesis is dedicated to my wife Regina, whose love and support made this possible, and to the memory of my parents Norman and Carolyn, who always encouraged me to pursue my goals with dedication and perseverance.

ACKNOWLEDGEMENTS

The author would like to acknowledge Mr. Kurt Buehler of the United States Army Construction Engineering Research Laboratories (USA-CERL) for developing an enhanced version of Xgen and for providing an early release of the software for use in developing this GRASS to ORACLE interface. The graphical user interface versions of this interface would have taken many months to develop, rather than weeks, without Xgen. The author would also like to acknowledge the many programmers involved in developing the GRASS programming libraries. This project would have been much more difficult without these libraries.

ABSTRACT OF THE THESIS

An Interface Between the GRASS Geographic Information System and ORACLE Relational Database Management System

by

David Gordon Buker

Florida International University, 1993

Miami, Florida

Professor Naphtali Rishe, Major Professor

A query and display interface has been developed between the GRASS geographic information system and the SQL-based ORACLE relational database management system (DBMS). This interface enables multiple non-spatial attributes of GRASS map features to be maintained with the DBMS. GRASS alone is capable of storing only one attribute per feature. The interface allows the user to provide both spatial (GRASS) and non-spatial (SQL) selection criteria for any query. Spatial selection methods include picking items from the GRASS map with a mouse, and specifying areas of interest with user-drawn (via a mouse) polygons and transects. The results of the combined query are displayed both graphically (the selected GRASS map features are highlighted in a graphics window) and textually

(the DBMS attribute data are shown in a text display window). Options include creating reclassified maps based on the DBMS output, and updating the attributes retrieved by a query.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

This thesis presents a software system that integrates GRASS geographic information system spatial data with ORACLE relational database management system (DBMS) attribute data for the GRASS maps. It is not a single program, but rather an integrated system that consists of several separate programs. These programs work either together or alone to perform a number of different functions. Although the system has been implemented using ORACLE, it is capable of supporting other structured query language (SQL) based DBMS products with the development of additional DBMS drivers. No program customization is needed to use the interface with different GRASS maps or DBMS tables.

Project Background

Geographic Information Systems (GIS) have become an important tool for many businesses and governments at all levels. Although GISs have been in use since the 1960s, the combination of readily available spatial data and improved software and hardware, along with a drastic reduction in the costs of the systems, has resulted in a great increase in use over the last few years [Smith *et al*-87].

Although there is much disagreement over exactly what constitutes a GIS ([Clarke-86], [Cowen-88]), we will define a GIS as a computer automated system for the input, editing, management, display, and analysis of geographically referenced spatial data, along with associated non-spatial attribute data. It is the geographically oriented analytical ability, in particular, which differentiates a GIS from computer aided design (CAD) and automated cartography systems. GISs are used for a variety of purposes by different organizations. Nagy and Wagle [Nagy/Wagle-79] survey ten specific systems which represent examples of a variety of applications and systems that existed at that time. Dangermond and Freedman [Dangermond/Freedman-86] list a number of examples of GIS applications for municipal governments, such as performing vehicle routing, traffic analysis, facility siting, land use planning, and facilities management. The U. S. Federal government agencies use GISs for a variety of reasons including tracking the geographical distribution of census information and disease statistics ([GISWorld-89A], [GISWorld-89B]). Locally, Everglades National Park has used a GIS for analyzing the potential impact of water management alterations on the plants and animals of Everglades National Park [SFRC-90].

GRASS (Geographic Resources Analysis Support System) is a public domain GIS that was originally developed by the United

States Army Corps of Engineers at the Construction Engineering Research Laboratory (CERL) in the mid 1980s. Since then, the use of GRASS has expanded to include equal representation in government, education, and private sectors [Goran-92]. The expanded use of GRASS has necessitated the formation of two organizations to coordinate the development, distribution, and use of GRASS. The Office of GRASS Integration (OGI), located at CERL, manages the U. S. federal government's involvement with GRASS, while The Open GRASS Foundation (OGF) coordinates the GRASS activities of academia, the private sector, and state and local government organizations ([Goran-92], [Schell-92]). Although the OGI at CERL continues to be the primary development site for GRASS, enhancements and additions to the GRASS software are now being developed by all segments of the user community, both public and private.

GRASS is used by a wide variety of federal government agencies, and is the official standard GIS for the National Park Service and the Soil Conservation Service ([NPS-93], [SCS-90]). As an employee of Everglades National Park, the author uses and manages the park's GRASS GIS. One of the important features lacking from GRASS has been the ability to associate multiple attributes with the features of maps, only a single attribute could be used. For example, a road could be identified with its name, but not also with its road-type,

date of construction, and width. This need to associate multiple attributes with the GRASS map features led to the development of the software described here. The software will be used not only in Everglades National Park, but will be made available to the entire GRASS user community. The software will be submitted to the OGI for evaluation and incorporation in the official GRASS software release.

GIS Basics

The general characteristics and capabilities of relational database management systems, such as ORACLE, are well known to most readers. Geographic information systems, however, have not been common until relatively recently. For this reason, the general features of GIS are discussed below. This information is necessary to fully understand the interface which has been developed.

Spatial Data Representation

Geographic data can be represented by three basic primitives: points, lines and polygons (areas). Associated with each of these objects are one or more attributes which describe something about the object. For example, on a small scale

map a point object may represent the location of a building. The attributes for this building may include the address, the owner's name, and the number of rooms. A river would be represented by a line feature. Attributes for the river might be the name, width and depth. A farmer's field would be represented by a polygon. The farmer's name, the type of crop grown, and the yield per acre are examples of attributes which might be stored.

The geographic data can be represented in the computer in two different ways: vector or raster (cell-based). The decision as to which representation is used is frequently a function of how the data were captured for use in the GIS. A vector representation stores the information in an explicit point, line, area format, with each object "tagged" with it's attributes (Figure 1). This format is used in GIS applications which require visually appealing output and/or precise location information. Data in this format are usually captured by manual digitizing or scanning of existing map data. The raster format divides an area into a set of grid cells (Figure 2). These grid cells are generally of uniform size and shape, with the most common shape being a square. The attributes are associated with the individual grid cells. In general, the individual objects no longer have a distinct identity, rather an object is implicitly defined in that all the cells of the object have the same

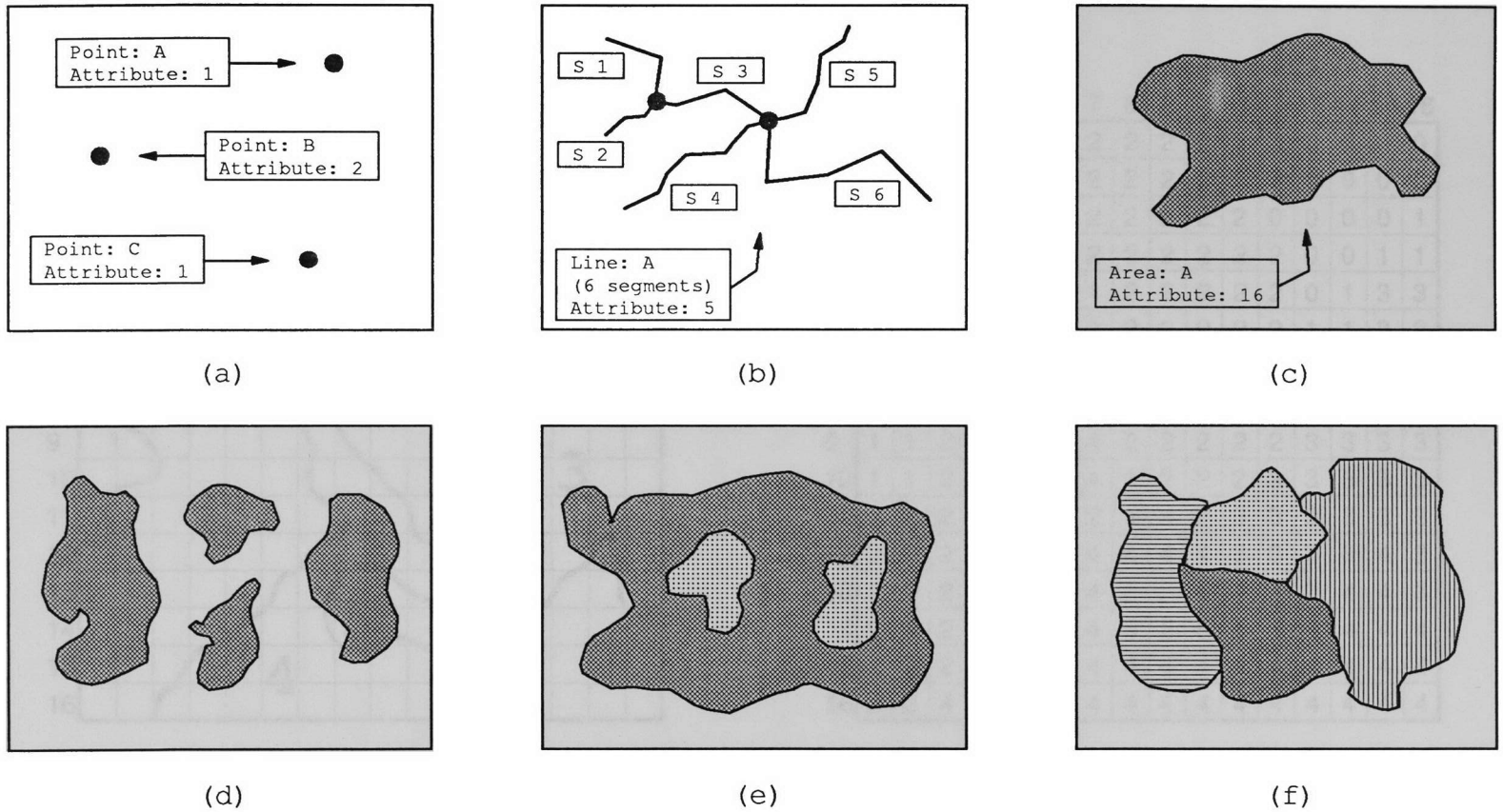
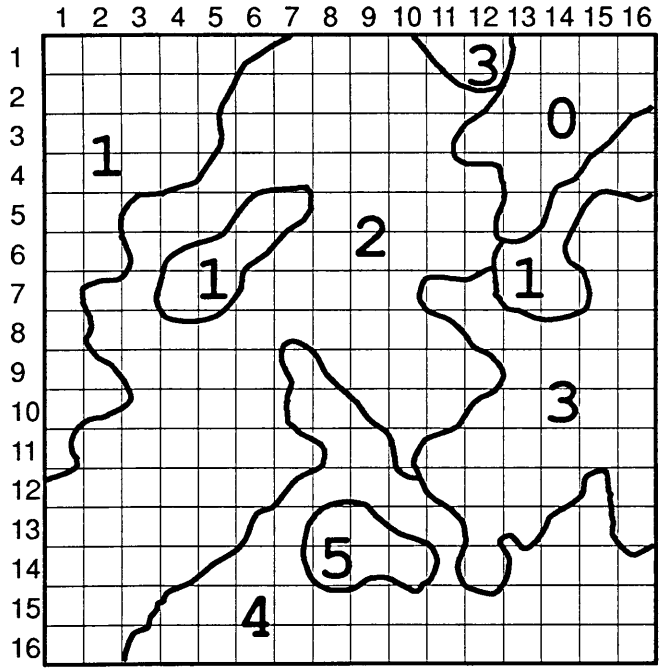


Figure 1. Vector representations of map information. A vector map consists of points, lines, and areas, each with its own locations and non-spatial attributes. (a) shows three points, two of which have the same attribute value. (b) represents a line feature made up of six segments. (c) shows a single area along with its attribute value. (d) shows four separate areas of the same type. (e) illustrates an area which has two other areas, called "islands", within it. (f) shows four adjacent areas.



(a)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	1	1	1	2	2	2	2	3	3	0	0	0	0
2	1	1	1	1	1	2	2	2	2	2	2	2	0	0	0	0
3	1	1	1	1	1	2	2	2	2	2	2	0	0	0	0	1
4	1	1	1	1	2	2	2	2	2	2	2	2	0	0	1	1
5	1	1	2	2	2	1	1	2	2	2	2	2	0	1	3	3
6	1	1	2	1	1	1	2	2	2	2	2	2	1	1	3	3
7	1	2	2	1	1	2	2	2	2	2	3	3	1	1	3	3
8	1	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3
9	1	1	2	2	2	2	4	2	2	2	2	2	3	3	3	3
10	1	1	2	2	2	2	4	4	2	2	2	3	3	3	3	3
11	1	2	2	2	2	2	2	4	4	2	3	3	3	3	3	3
12	2	2	2	2	2	2	4	4	4	4	3	3	3	3	4	3
13	2	2	2	2	2	4	4	5	5	4	4	3	3	4	4	3
14	2	2	2	2	4	4	4	5	5	5	4	3	4	4	4	4
15	2	2	2	4	4	4	4	4	4	4	4	4	4	4	4	4
16	2	2	4	4	4	4	4	4	4	4	4	4	4	4	4	4

(b)

Figure 2. Raster representation of map information. (a) shows a vector area map which has been superimposed on a grid. The numbers are attribute values for the areas. (b) shows how the map would be stored in the raster format. The attribute values are stored as the values for the grid cells.

attribute value. Satellite imagery is the most common type of data which is in a raster format. At times, data which were originally captured in a vector form are converted to raster form. This is done because many of the algorithms for raster map analysis operations are more efficient than the corresponding vector operations ([Burrough-86], [Maffini-87], [Monmonier-82]). The relative advantages and disadvantages of the two formats are discussed by a number of authors ([Peuquet-86], [Maffini-87], [Smith *et al*-87]).

Because of the distinct differences between vector and raster data, most operational GISs have been built around one or the other of the two types, but not both ([Abel/Smith-86], [Keating *et al*-87], [Lorie/Meier-84]). In some cases, a system may handle both types, but data have to be explicitly transferred or converted from one type to the other ([Clarke-86], [Jackson *et al*-88]). Maffini [Maffini-87] points out, however, that the conversion process may result in data quality problems due to the fact that the raster to vector conversion is an ambiguous process. There are a number of efforts underway to combine both raster and vector data into a single unified system, with the form of the data being transparent to the user ([Anthony/Corr-88], [Haralick-80], [Jackson/Mason-86], [Jackson *et al*-88], [Peuquet-84] [Shapiro-80], [Waugh/Healey-87]).

GRASS supports both raster and vector maps but it is primarily a raster-based system ([Shapiro *et al*-89], [USArmy-91]). Although GRASS also has "site lists" maps, these can be considered a special case of vector data. Most of the GRASS GIS analysis functions only support raster data. The type of map data is not transparent to the user. Most commands explicitly include the map type as part of the command name. For example, the three basic commands to display maps are *d.rast*, *d.vect*, and *d.sites* for raster, vector, and site maps, respectively.

Another basic feature of geographic information handling concerns the concept of topology. Topological considerations have an impact not only on how data are entered into the system but also on how the data are stored and analyzed. Topology refers to the scale-independent spatial relationships between different map objects, or as Burrough [Burrough-86] defines it, the way in which geographical elements are linked together. Issues relating to topology are discussed in [Keating *et al*-87], [Muller/O'Connor-82], and [Peuquet-86]. Relationships such as "connected", "next to", and "enclosed by" are examples of topological properties. These types of relationships are some of the most common that users are interested in determining from a GIS database, yet they are computationally difficult to

determine unless this information is explicitly recorded in the database [Smith *et al*-87].

To achieve this topological structure, vector data must be captured, usually by manual digitizing, in a particular way. Data are frequently digitized off of map sheets which have been specially prepared for this purpose. These map sheets usually are just line drawings of the features of interest (Figure 3 (a)). The features are digitized one at a time, with each distinct line segment being digitized as a separate item (Figure 3 (b)). Line segments are also commonly referred to as "arcs." Wherever two (or more) lines cross, a "node" item is created. Associated with a node is its geographical location. A line segment starts at a node and ends at a node. A line segment may be either part of a linear feature (e.g., a power line) or may be the edge (boundary) of an area feature (e.g., a lake). The representation of linear features in this manner is frequently called "arc-node" format. Associated with each line segment is a list of geographical coordinates which describe the segment. Area edge segments also start and end at nodes but, in addition, have an area to the right and an area to the left (edges by definition separate two areas). Another important topological property is that of containment. Areas which are completely contained within another area are called "islands" (Figure 1 (e)). Islands

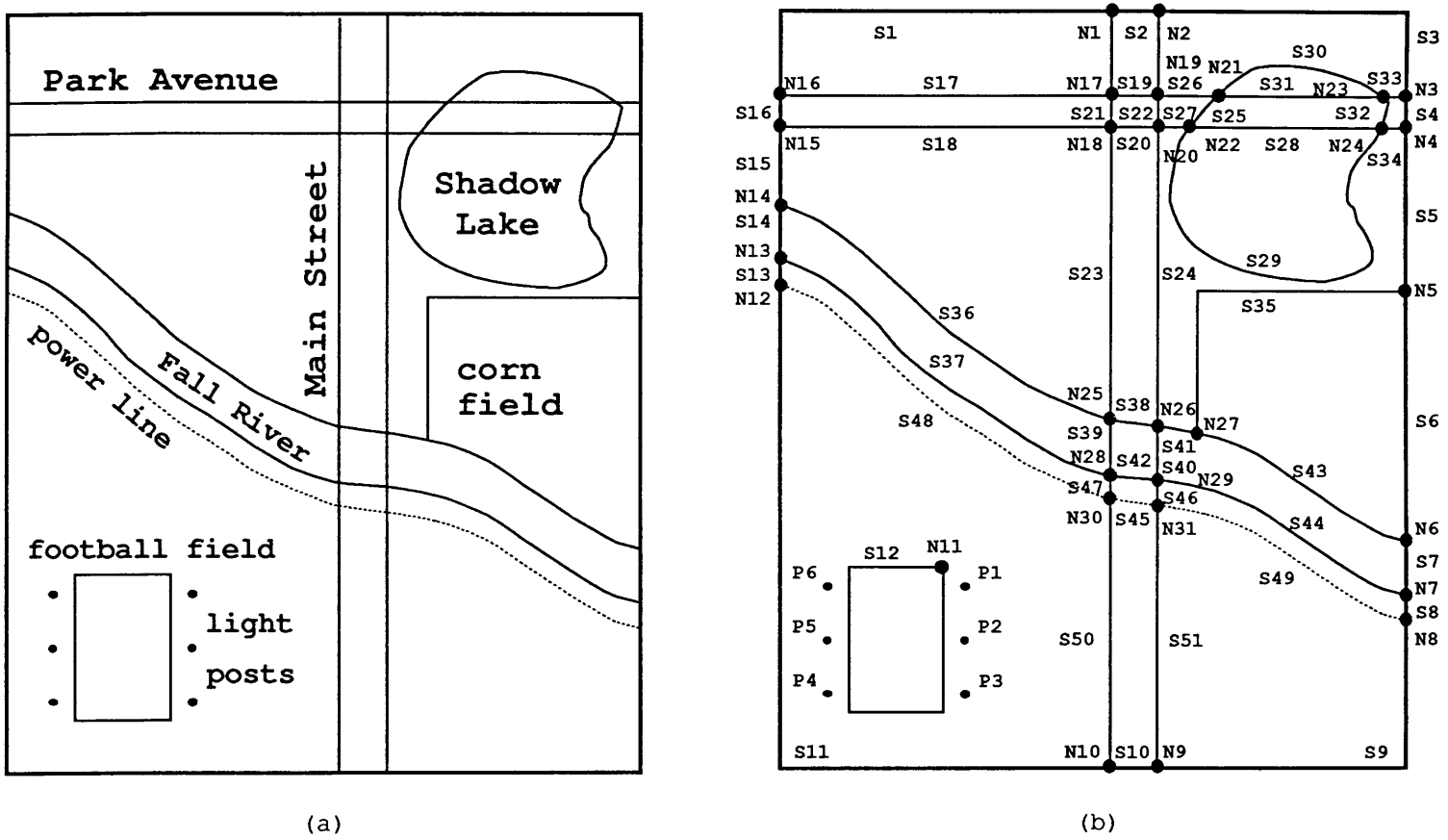


Figure 3. Vector digitizing. (a) shows a simple line drawing of geographic features. Data entry into a GIS via digitizing is usually performed from simple maps such as these. (b) represents the digitized elements of the map shown in (a). Each element has a unique identity. Nodes are labeled with "N", segments (the individual sections of lines and area edges) are labeled with "S", and points are labeled with "P."

are themselves "normal" areas, and therefore have all the properties and relations of other areas. Hardware and software systems which scan vector maps and automatically develop a fully topological data set are making vector map input a much easier process than the hand digitizing method traditionally used.

The topological relationships for raster data are not explicit as they are for vector data. As noted above, vector data items are typically thought of as distinct objects; each point, line, or area being an individual, identifiable, object. Each object can be assigned one or more attributes at the users convenience. Raster data, on the other hand, traditionally has been viewed very differently. The entire raster matrix is seen as a single "layer", "coverage" or "theme." Within the layer, each cell of the matrix has, usually, only a single value (Figure 2 (b)). Multiple attributes are not supported. Discrete areas of contiguous, same-value cells do not have a distinct identity in the way that vector areas have an identity. A raster area is implicit (a group of contiguous, same-value cells) whereas a vector area is explicit. This imposes on the user, and the software, very different ways of working with the data.

GIS Data Structures

A variety of different data structures have been used for GISs, most based upon hierarchical, network, or relational models ([Armstrong/Densham-90], [Abel/Smith-86], [Clarke-86], [Burrough-86], [Haralick-80], [Jackson/Mason-86], [Lorie/Meier-84], [Monmonier-82], [Peuquet-84], [Shapiro-80], [VanRoessel-87], [Waugh/Healey-87]). Most of these systems have stored the spatial data separately from the attribute data ([Clarke-86], [Waugh/Healey-87]). Many of the existing systems have also been designed as stand-alone, special purpose systems with little attention paid to traditional database management concerns such as data protection, security and integrity [Frank-88]. Several recent GISs have been proposed which address this concern by building the spatial database around existing commercially available database management systems ([Abel-89], [Abel/Smith-86], [Lorie/Meier-84], [Waugh/Healey-87]).

Existing commercial DBMSs, however, are not well suited to the requirements of GIS processing ([Abel/Smith-86], [Frank-88], [Keating *et al*-87], [Lorie/Meier-84]). One of the major problems is the performance of these systems in retrieving spatial data. Frank points out the necessity for GISs to have very fast response times for drawing maps on screen, and estimates that an average map contains between 2000 and 5000

different items. If each of these items must be retrieved separately from disk, as is the case with most DBMSs, he estimates that it will take one to three minutes to draw the map on screen. He states that this time is unacceptable. The problem is that items which are located close together geographically, are not physically clustered on the disk. In order to achieve adequate performance, this physical clustering is necessary. This performance problem is being addressed by some of the recent work involving commercial database systems [Abel-89].

Another drawback of commercial DBMSs is that they contain no spatial data query or manipulation language capabilities. A number of authors have discussed the types of capabilities which are required ([Nagy/Wagle-79], [Peuquet-86], [Shapiro-80]) and some have developed, or proposed, extensions to existing DBMS query languages which include spatial operations ([Abel/Smith-86], [Frank-88], [Goh-89], [Roussopoulos *et al*-88]), others are researching more "natural" language interfaces to GISs ([Robinson *et al*-86], [Samet *et al*-84]), knowledge-based languages [Wu *et al*-89], and tabular based Query By Example (QBE) approaches [Joseph/Cardenas-88].

The GRASS database structure is described in [Shapiro *et al*-89], with some changes noted in [Gerdes-91] and

[Shapiro/Westervelt-91]. The basic characteristics are noted here in relation to the discussion above. Some of the information discussed below is not documented, but was determined by direct examination of the source code.

GRASS databases are structured hierarchically using the UNIX directory structure. An individual map layer consists of a number of separate files, located in different directories, which contain various types of information concerning the map. Examples of these files include the spatial data itself, a header file describing the map, a feature attribute (category value) list, category labels, and color tables. The files for the different map types (i.e., raster, vector, sites) are stored in separate, type-specific, directories. Some of the files apply only to certain map types. For example, the color tables only apply to raster data. In other cases, the information contained in two or more files for one map type may be combined into one file for another map type. For example, a single site list file contains the header information, spatial data, and category or label information. The structure of the spatial-data files is different for all three map types.

A site list file is a flat ASCII file. An individual site record is one line in the file. Each record consists of the Universal Transverse Mercator (UTM) coordinates of the site

and an integer category value or textual label for the site. UTM is an equal-area geographic coordinate system, unlike latitude/longitude where the distance between longitude lines, and hence area, varies with latitude. There are no indexes maintained for the data in the file, so processing this file requires sequential record retrieval.

Vector data records are stored in a binary format using the arc-node representation described previously. Nodes are not stored separately from the arcs, they are simply the beginning and ending points of each arc. The arc-node records are stored in the same order in which they were digitized or imported into GRASS. Attributes of the vector features are stored separately. Unlike site lists files, however, indexes are created and stored to provide rapid random access to the vector data (the actual coordinates of the area edges, lines, or points) and attributes. The indexes for the vector data are just pointers from a list of the vectors into the actual vector data, they are not spatial indexes.

Raster data are stored in one of two different binary formats, uncompressed and compressed, or a third ASCII format for reclassified maps. Conceptually, a GRASS raster file can be thought of as a two dimensional matrix where each cell of the matrix contains an integer value, as shown in

Figure 2 (b). This integer is the "category value" for the cell. Note that what is stored are the category values for cells, not explicit spatial information. The integer values are stored using from one to four bytes of data. Integer values greater than 255 are stored using *big-endian* format (i.e., a base 256 number with the most significant digit first). Negative values are stored as signed values (i.e., with the highest bit set to 1). This results in all negative values always requiring the number of bytes used by the machine to store integers.

The matrix is spatially referenced using information contained in the raster "header" file. It is the header file that contains the UTM coordinates and cell size (e.g., 2 miles X 2 miles) for the map, along with other descriptive information. The spatial location of any given cell within the matrix is determined based on the row and column numbers of the cell, the origin of the matrix, and the cell size.

The uncompressed file format is essentially a binary representation of this matrix format. Within the GRASS system, the category value for each cell of a matrix is stored using from one to four bytes, as described above. For any given map, however, all cell category values occupy the same amount of computer storage. The number of bytes used for any particular map is the number of bytes needed to store

the maximum category value contained in the map. The physical storage of records is by row, with all the cells (columns) of the row stored as part of the record. All rows have the same number of cells, so the physical sizes of all records are identical. Rows can be retrieved randomly based on this known record size. Once a row is retrieved, the value of any cell is quickly retrieved based on the cell column number and the known category-value storage size.

The compressed files use a run-length encoding scheme to reduce the disk storage requirements. As with uncompressed files, the records are stored by rows of the matrix, but in this case each row is compressed using the encoding scheme. The encoding is done by storing a single byte repeat-count followed by a category value. The repeat-count represents the number of sequential cells in the row that have the particular category value. Whenever a category value sequence ends within a row, and another sequence begins, another repeat-count/category-value pair is generated. The number of bytes used to store the category value is consistent within any given row, but may vary from one row to another within a single map. This is unlike the uncompressed files, where the number of bytes used does not vary within a map. If an encoded row is determined to be longer than the same un-encoded row, the un-encoded row is used. Due to this encoding scheme, the physical size of row records varies

within a map file. In order to provide rapid row record retrieval, an index to the start of each record is generated and stored. Once a row record is retrieved, the record must be uncompressed in order to find the value of any desired cell within the row.

The third raster format is used for "reclassed" raster maps. A reclassed map does not actually store any category data. Instead, it stores category-reclassification rules which reference an existing raster map. These rules are contained in an ASCII flat file. For example, consider that a cell in the original map contains category five (5). The reclass map, for whatever reason, may include the rule that category five should be reclassified to category nine (9). Whenever the reclassed map is displayed or used for analysis, the original map is first retrieved and then the reclass rules are applied. In our example, therefore, the cell value used for the given cell (or any other original-map cells with a category value of five), would be nine. Note that this scheme means that the original map must never be deleted, since then the reclassed map loses its reference.

Data Access Requirements of a GIS

The capabilities required for a GIS database have been discussed by a number of authors ([Dangermond/Freedman-86], [Dangermond-86], [Frank-88], [Lorie/Meier-84], [Monmonier-82], [Peuquet-86], [Smith *et al*-87], [Webster-88]). The requirement for access to data based on spatial characteristics is the primary feature which differentiates GIS database systems from most other DBMSs. Smith et al. [Smith *et al*-87] point out that there are two basic types of queries needed in accessing a GIS database:

1) find the **locations** of some specified objects (e.g., "display all rivers")

2) find the **objects** within a given location (e.g., "display everything in Miami")

These two queries illustrate the two different access paths which are needed by a GIS database system. One path must allow access to the data **based on spatial location**, the other path must provide access **based on an object's non-spatial attributes**. The objective of this project was to develop an interface that would provide both of these access paths for GRASS maps and their associated attributes. The next section discusses this in relation to GRASS's current limitations.

Why develop this interface?

GRASS is very capable of storing, managing, displaying and analyzing complex spatial data. Any given map contains many features of interest to users. Every cell of a raster map and every vector item of a vector file can be given an integer value (category number) to identify it. In turn, each unique category number can be given a text label containing some meaningful non-spatial information (attribute) about the category. Every site location in a site list file can be given a comment. This comment can be just text or, optionally, can consist of a pound sign (#) followed by a number which can be followed by a space and a text string (e.g., #7 light pole). Basically, this means that all three of these map types are limited to a **single** attribute for each feature. However, real map features frequently have **multiple** attributes associated with them.

To illustrate the need for multiple attributes, consider a database of Florida panther observations (this is a real application which is being implemented at the South Florida Research Center in Everglades National Park). This example will be used throughout the paper. Florida panthers are an endangered sub-species of panthers, also known as mountain lions and cougars, living in Florida. Only 30 to 50 Florida

panthers are believed to exist in the wild. A research program was begun a number of years ago to monitor the status of the known panthers in southern Florida. One of the tasks of this research project was to radio collar the panthers and determine their locations once a day. The information collected for each observation includes the geographic coordinates of the observation, the panther identification (ID) number, and the date and time of the observation. In GRASS, however, a panther observation could be labeled only with the panther's ID number, for example. It is true that all of the information that we record for each observation could be "packed" into the single label string (e.g., Feb-3-1992, 10:25 am, Panther # 14), but GRASS does not allow the user to selectively pick observations out of the sites file based on the information packed into the string. A user could not, for example, select only the observations of panther number 14. The same situation applies for raster and vector maps. Using a spatial mask, a user can selectively display only certain raster categories, but there is no convenient, batch oriented, way to select the categories for display or analysis based on the information contained in the label for the category. This inability to handle multiple descriptive attributes in a way that makes it easy for a user to select particular features is a major limitation of GRASS. GRASS users need to be able to query, display, and analyze

the information based on both the spatial and non-spatial characteristics of the data.

Database management systems, on the other hand, are excellent systems for storing, managing, displaying and analyzing non-spatial data. This is what DBMS software was designed to do. This makes a DBMS an excellent way to handle the multiple attribute data associated with GRASS maps. Many GRASS users have in fact been doing this for years. However, performing selections of the GRASS features based on the DBMS data, or vice versa, is a complicated, time consuming, multi-step process involving exporting information from one system and importing it into the other system.

The ability to easily and quickly relate GRASS map features to associated attribute data stored in a DBMS would therefore increase both the power and flexibility of the overall system. This is exactly what this interface does.

Although it would be possible to build DBMS capabilities directly into GRASS, as opposed to linking GRASS to an existing DBMS, there are at least two disadvantages of doing this. One disadvantage is that in order to have as complete and flexible functionality as available in existing DBMS packages, software development time would be much greater than for an interface to an existing DBMS. To keep

development time down, just a subset of capabilities could be provided, but this would decrease the usefulness of the system. A second disadvantage is that users are already using certain DBMS software and wish, or are required, to continue using it for their non-spatial data. For example, some agencies or companies mandate the use of particular DBMS software. In other situations, users just do not want to have to work with two different DBMS packages, a specialized one for GRASS related data and another for the rest of their data. For these reasons, an interface was developed to an existing DBMS. As mentioned previously, the interface was developed so that it could be used with other DBMS packages. This requires the development of a DBMS driver for each DBMS package, but this is a reasonable task for an experienced programmer.

This interface is just a short-term solution to the problem of handling multiple non-spatial attributes related to GRASS maps. The author believes that, ultimately, GRASS will have to be modified to more closely integrate with DBMS software. This is needed so that direct joint GRASS/DBMS analysis and printing are supported, not just joint query and screen display. There are efforts already underway by other groups to do this. The need to link to multiple commercial DBMS packages will still be critical, however, even if DBMS capability is built-in to GRASS.

Project Objectives

There were a number of major objectives which influenced the design and implementation of this interface. These objectives fall into two basic classes, functional objectives (i.e., **what** the software should do) and design/implementation objectives (i.e., **how** the software should do it). The objectives of each type are discussed below.

Functional Objectives

- Provide **direct** support for combined GRASS and DBMS data query and display capabilities.

  The software should support concurrent "two-way" queries that apply both spatial and non-spatial selection criteria to the same query. The importance of this capability are discussed further in the System Overview section.

- Provide **indirect** support for map analysis and printing.

Allow the user to use the results of combined GRASS-DBMS queries to create new, reclassified, GRASS maps which can then be used for map analysis and printing.

- Provide the ability to save and print the DBMS results of the query.

- Provide three different levels of user interfaces.

  - Command Level

    Allow the user to enter most user input arguments on the command line. User input that is selection-method specific will not be supported as command line arguments, but will be entered interactively using the keyboard or mouse. The entry of these method-specific arguments is the same for all three levels.

  - Command Interactive Level

    Allow the program to be executed without any command line arguments. In this case, all of the arguments will be entered by the user in response to prompting by the program. The prompting and

corresponding user entry are done in a simplistic line-based method.

- Full Interactive Level

  Provide a graphical user interface that allows the user to enter all command line arguments using a combination of keyboard and mouse input. Keyboard entry is done in editable text-entry fields. Mouse-based entry involves selecting from displayed lists of valid values for the arguments.

- Include the ability to save and retrieve interface sessions.

  Allow the user to save the arguments specified for any query. This option should save all arguments, including the DBMS table name, GRASS map name, SQL clauses, spatial selection method, and map colors.

- The interface should be easy to use for beginning users.

  The interface should require only a few arguments. This will enable inexperienced users to get useful

results immediately without needing to know much about the software.

- The interface should be both powerful and flexible for advanced users.

  The interface should support many optional arguments. Experienced users should be able to specify a variety of optional arguments which enable advanced operations to be performed or which allow the output to be customized. These optional arguments will be pre-set with defaults which will be used if the arguments are not specified.

Design and Implementation Objectives

- The interface should have the same "look and feel" as other GRASS programs.

  This objective involves providing a consistent user interface and using common naming conventions for input arguments. This does restrict the design of the user interfaces for the programs of the system.

- The interface must be **strictly** an add-on capability, with no changes to basic GRASS data structures or commands.

  This restriction puts significant limitations on the design and implementation of the system. This objective is required, however, to limit the scope of the project to a realistic level. Dozens of other GRASS programs would be affected by changes to basic data structures. Modifying all of these other programs to use any modified data structures is a task beyond the capabilities of a single individual. Even if the modification of the existing versions of the programs could be accomplished, the future maintenance of all of the programs would be impossible for one person.

- The interface software development should conform to GRASS programming standards.

  These programming standards include requirements such as developing within UNIX, using the C programming language (K&R C) and using standard GRASS program libraries for device independence. Any software which is intended for wide distribution must use the native C compilers on the computer systems. This requirement

is specified to both reduce costs and to ensure that the software can be installed and compiled at any site without the need to buy additional compilers. This would be needed if the source code used C language extensions or library functions which were only available in a particular proprietary compiler.

- Develop programs with portability in mind.

  GRASS is used on a wide variety of computer systems. For this reason, non-standard C language or library extensions should be avoided.

- The interface should be easy to adapt for use with other SQL-based database management systems.

  This objective was originally specified for the basic interface only (*db.interface*) due to the more complex nature of the screen-form version (*db.forms.interface*). A design was developed, however, which may allow DBMSs other than ORACLE to be used with the screen-form version of the interface.

- The interface should be developed so that it is easy to add or modify GRASS spatial selection methods.

- Eliminate the command-level distinction among the map types (raster, vector, site).

  This is an objective that deviates from the objective of retaining the standard GRASS "look and feel." For most existing GRASS operations, there are separate programs for each map type. For example, there are three different programs for displaying GRASS maps: *d.rast* for raster maps, *d.vect* for vector maps, and *d.sites* for site maps.

  This interface has been developed with the map type as an argument to the program; separate versions of the program have not been created based on map type. This objective has been specified for two reasons: (1) a single program results in less duplicate code, which decreases software maintenance requirements, and (2) philosophically, the author believes that the distinction between types should, as much as possible, be transparent to the user.

- Use a well defined, robust, conservative link between the GRASS map features and the DBMS table rows.

- The system should support reasonably simple GRASS-map/DBMS-table **development**.

The **developer** of a linked map/table must be experienced with both GRASS and relational DBMS table design and implementation. Given this level of ability, development should not be complex and only a few new concepts should need to be learned.

INTERFACE DESCRIPTION

System Overview

The interface provides integrated GRASS and DBMS query and display capabilities. The user can specify both spatial (GRASS) and non-spatial (DBMS) selection criteria for any query of the joint GRASS/DBMS database. Spatial selection criteria are specified using one of the six spatial selection methods provided, such as picking GRASS map features from the display screen with a mouse and overlaying a vector map containing areas of interest. Non-spatial criteria are specified using standard SQL. Both the spatial and non-spatial criteria are simultaneously applied to the joint database by the interface. Based on the results of the query, the selected GRASS features are drawn or highlighted in the graphics display window while the DBMS attribute data for those features are displayed on the text display screen or within a text window.

There are actually two versions of the interface, the difference being in the format of the DBMS output. One version (*db.interface*) produces simple output which consists of one output line per DBMS record (the "line" may actually consist of one or more screen lines if the data line is longer than the width of the screen), with many lines (records) being displayed on the screen at one time. The other version (*db.forms.interface*) produces output using ORACLE SQL*Forms screen forms. The structure of the form (i.e., the placement and order of fields on the screen) is defined by the designer when the form is created. In most cases, only one record is displayed at a time. This forms-based version of the interface uses these previously created forms for its DBMS output, and it allows the user to step through each output record retrieved by the query. The DBMS records retrieved by a query can also be updated by the user and saved back to the DBMS database.

To illustrate this query and display capability, consider the example, introduced earlier, of a database of Florida panther observations. In this case, however, it is now a joint database using both GRASS and the ORACLE DBMS. The GRASS database consists of a site list of the location observations. The DBMS database consists of two tables. One table (panther_obs) records information about each observation (e.g., the panther identification (ID) number,

and the date and time of the observation). The second table (panther) records general information about each panther being monitored (e.g., panther ID number, gender, name).

Given this Florida panther database, a possible application of the interface might be as follows. A major cause of death of panthers is being struck by cars. A road is being planned for a certain area. Will this have any potential impact on the panthers? Consider also that the interface user is particularly interested in the use of this area since January 1991. Using the transect option of the AREA selection method, the user draws a line along the proposed route of the road and specifies a 500 meter buffer on each side. The user also specifies that only observations since January 1991 are of interest (i.e., the SQL WHERE clause would be: WHERE Date >= '01-Jan-91'). The joint query would be applied to the database. In the GRASS display window, all panther sightings falling within the transect area since January 1991 would be highlighted. In the text display window the DBMS information for these selected panther observations would be displayed (whichever DBMS columns the user specified, or all columns of the panther_obs table if none were specified). In addition to displaying the DBMS data, the data could be printed, saved to a file, or used to produce a new reclassed map.

The ability to apply both spatial and non-spatial selection criteria to the same query is an important feature of this interface. It allows queries which return the intersection of the spatial and non-spatial data sets (Figure 4 (a)).

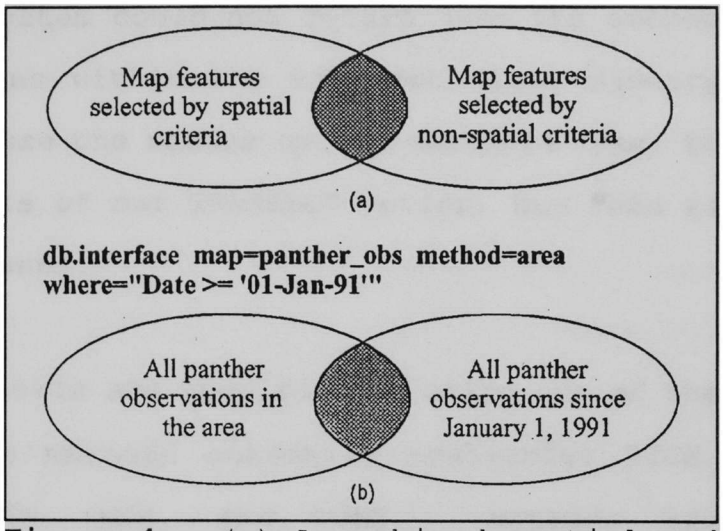


Figure 4. (a) A combined spatial and non-spatial query returns the intersection of the sets. (b) An example using the panther database. The query returns only those panther observations within the area since January 1991.

Figure 4 (b) uses our example to illustrate this capability. The example query returns the subset of panther observations within the transect since January 1991. Another recently developed GRASS to DBMS (Informix) interface will display map features based on non-spatial queries, but spatial selection is very limited [Farley-93]. It only supports a single map-feature PICK operation for vector and raster maps, and a single RADIUS operation for site data. The RADIUS option allows the user to specify a radius on the command line and then a map location with the mouse. All sites falling within the radius are selected. In this system, using our example, the user could display all of the panther observations since January 1991 but could not specify just those panthers within

the transect. This system could not return just the subset of panther observations within the transect since January 1991. The user could use the radius option multiple times to **approximate** the results of our TRANSECT option, but this is impractical in most cases.

Spatial selection criteria are specified by using one of the six spatial selection methods currently available: PICK, AREA, OVERLAY, REGION, MAP, and NONE. Methods are implemented as drivers, so additional methods can be developed easily without the need to modify the interface itself. In all cases, the spatial selection method selects an initial, tentative, set of map features which is then further refined by the SQL SELECT statement clauses provided by the user.

Non-spatial criteria for a query are specified using ANSI standard SQL. Although ORACLE provides a number of extensions or enhancements to SQL, standard SQL was chosen to allow use with other database management systems. The method of specifying the SQL SELECT statement clauses depends on which version of the interface is being used. For *db.interface*, the basic line-based version of the interface, the clauses are specified on the command line or interactively via the standard GRASS parser. For *db.menu*, the Xgen graphical user interface (GUI) version of

db.interface, the clauses are entered in text entry fields of the GUI. For *db.forms.interface* and *db.forms.menu*, the forms-based version of the interface and the Xgen GUI version of *db.forms.interface*, respectively, the non-spatial selection criteria are entered in the form-based method used by ORACLE SQL*Forms (a combination of explicitly entering clauses and entering selection constraints in fields of the form). In all cases, the sub-clauses needed by the interface to actually perform the link between a GRASS map and its linked DBMS table are automatically and transparently added by the interface to the SELECT statement. The user does not have to enter any detailed information about the link. At most, the user enters the map name and type and/or the table or form name; the interface determines the rest of the information needed and adds it to the SELECT statement.

The beginning user does not need to know any SQL to use this interface. By default, the user can just use the spatial selection methods and display all corresponding DBMS information. In order to apply any non-spatial selection criteria or join tables, however, the user must know SQL. With the Xgen version of the basic interface (*db.menu*), designers can develop and save complex queries which can then be used by other users who do not know SQL.

Although this interface was developed primarily to support integrated query and display capabilities, it was recognized that GIS analysis of information resulting from these joint queries is important. In order to support this capability, programs were developed to allow the creation of new GRASS maps based on the results of interface queries. These new maps can then be used, outside of the interface, to perform any display or analysis functions provided by GRASS. Since the actual analysis can not be done from within the interface, we refer to this as "indirect" support for map analysis, as opposed to the "direct" query and display support provided by the interface. The creation of the new maps can be done directly from within the interface, if the default choices used by the interface are acceptable for the user's application. In some cases, the user may need to create an output file within the interface and then use the map reclass programs outside of the interface. This allows different options to be used in creating the new map. In most cases, the DBMS output results are used to create a new **reclassified** map. For our example, a user might choose to create a new map which contains only the selected panthers within the transect since January 1991, and automatically reclassify the observations by panther number (i.e., add the panther number as the comment in the new sites file). The new map does not actually have to be a reclassified version of the original, however, it can simply be a subset of the

original features that retains the original classification scheme. For example, a user might create a new sites file with just the panthers within the transect since January 1991, but retain the observation-number classification.

All three GRASS map types (raster, vector, and site) are supported by the interface. At this time, only Universal Transverse Mercator (UTM) GRASS databases are supported, although latitude/longitude support is planned for the future. The DBMS interface itself is independent of the spatial reference scheme, but many of the spatial selection methods only work with UTM data.

The GRASS to DBMS link

The information contained in GRASS and a DBMS can be described as existing in two separate data **domains**. GRASS is the domain of the spatial data, and the DBMS is the domain of the non-spatial data. However, we want to treat all of the data as if it were in one virtual domain (Figure 5). We do not want to have to know, or care, where the data are stored.

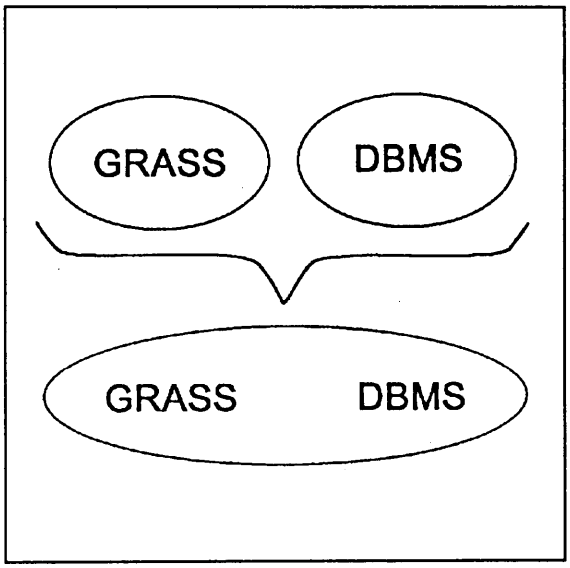


Figure 5. We want to treat the separate GRASS and DBMS information domains as if they were one combined domain.

We want to be able to query, display, and analyze the information in one standardized way independent of how or where it is stored. The question is, since we **know** that the data are managed by two different systems, what can we do to make it **appear** as though there is only one system? How do we link the two domains to form one combined virtual domain (Figure 6)?

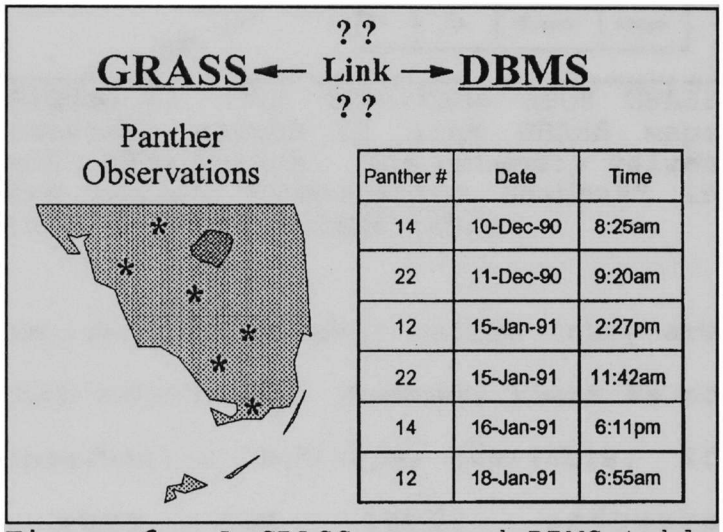


Figure 6. A GRASS map and DBMS table showing Florida panther observations in southern Florida. What can be used to link the GRASS map with the DBMS table?

This interface uses the GRASS category value to create the link (Figure 7). **For this interface, the category value is the one and only way to distinguish, spatially or non-spatially, the information stored in the joint data domain.** The interface does not use spatial criteria to differentiate features, other than when performing the initial spatial selection. For this reason, it is important that a one-to-one link be created between a map feature (category value) and its related DBMS attribute data.

The category value is actually not an optimum identifier to use as a link. For example, category values do not usually uniquely identify different raster **clumps** (areas of contiguous cell-category values), and site lists do not really have strictly defined category values (they are defined for use with this interface). However, there is no other well defined, conservative identifier available. If category values were not used, software implementation-dependent "tricks" would have to be used to form the link. For example, for vector area features one could use the area index, which is a unique identifier for each area stored in the file, but this is an implementation characteristic that is not strictly defined in terms of how the index varies as areas are added, deleted, or modified; it could change in future releases. This would be worse than making the interface crash, because the interface might still **function**, but the **results would be incorrect**. Category values are well defined and are under the complete control of

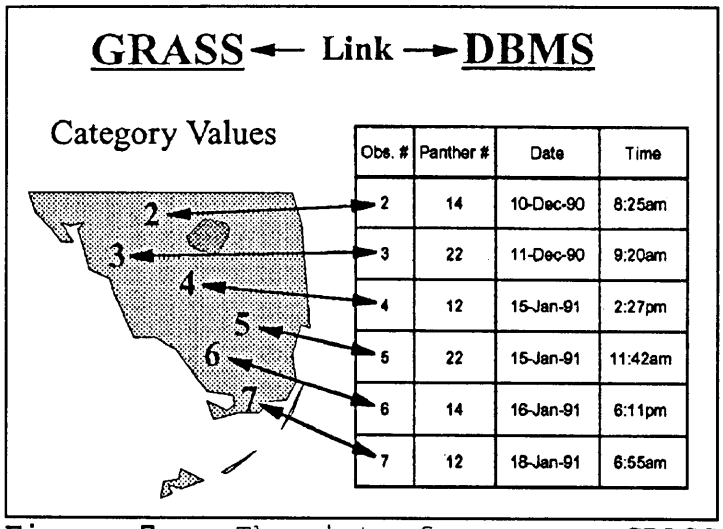


Figure 7. The interface uses GRASS category values to link GRASS maps with DBMS tables. The category values are called "observation numbers" in this particular DBMS table.

the map/table developer; nothing "hidden" can happen to these values and are, hence, conservative identifiers. Note, however, that as far as the DBMS is concerned, it does not care **what** identifier is used, it is completely independent of this; it just requires an integer value of some sort. Therefore, in the future, a different GRASS feature identifier could be used with no programming changes being required on the DBMS side of the interface. The actual identifier **values** stored in the DBMS tables would, of course, have to be changed.

The basic concept of how the category values are used to form the links between a GRASS map and its corresponding DBMS table and between an SQL*Forms screen form and a GRASS map are outlined below.

- GRASS maps

  - Raster: The GRASS category value is used to uniquely identify each distinct clump of cells. The GRASS command *r.clump* can be used to create the unique category values for a map. In some cases it may be impractical to conform strictly to this rule. Any deviation from this rule should be done with caution, however, and with a clear understanding of the consequences.

- Vector: The GRASS category value is used to uniquely identify each distinct vector feature (point, line, or area).

- Sites: For the purposes of this interface, we "define" a category value (essentially just a "site number"). Each site in the file is given a unique integer value. The integer is stored in the comment field of the site record, immediately following the pipe symbol (|).

- DBMS tables

  - Every DBMS table that is linked to a GRASS map must have a column that contains the category values of the linked GRASS map. This table is referred to as the **base** table for the link. For an example, see Figure 7. Other tables may exist that are related in some way to this base table, but it is the base table that contains the links to the GRASS map. The user can join these other tables to the base table when using the interface, if desired.

  - Every row of the base DBMS table corresponds to a specific GRASS map feature, which is uniquely defined

by its category value. This category value is stored in the column noted above. The relationship of table rows to GRASS features should be one-to-one (1:1).

The specific information describing each map-to-table link (i.e., table name, map name and type, etc.) is stored in a special DBMS table referred to as the map-to-table "link" table (*grass_to_dbms_link*). This information is specified by the GRASS-map/DBMS-table **developer**, and only has to be done once for each linked map/table. In most instances, once this is done the **user** does not need to know any of this information other than the map name and, in some cases, the table name. If the user does need to specify any of this information, utility programs are provided to display the needed information. In most cases, the interface programs access and use this "link" information transparently.

- ORACLE SQL*Forms screen forms

These instructions apply only to the SQL*Forms version of the interface (*db.forms.interface*). You must have the ORACLE SQL*Forms option to use this program.

- Every form that is to be linked to a GRASS map must include a field to contain the category value. The

form must also be implemented using four specific SQL*Forms user exits that have been developed as part of this interface ("user exit" is a term used by SQL*Forms). Once a developer is familiar with SQL*Forms design, adding in these four user exits is not difficult. These user exits actually link the form to a DBMS base table, not directly to a GRASS map. The DBMS table must in turn be linked to a GRASS map by the methods outlined above. In this way a single form may be linked to many maps, and vice versa, while still maintaining a normalized DBMS database. The implemented form can be used either alone or as part of the interface. When a linked form is executed, one of the user exits determines if the form was started by the interface. If the form was started by the interface, the appropriate links are automatically made to GRASS. If the form was not started by the interface, it can be used to access the DBMS data, but with no interaction with GRASS. In this way, only one form is needed, not one for when using GRASS and a separate one for when not using GRASS.

The specific information describing each form-to-table link (i.e., table name, form name, etc.) is stored in a special DBMS table referred to as the form-to-table "link" table

(*form_to_dbms_link*). This information is specified by the SQL*Forms-form/DBMS-table **developer**, and only has to be done once for each linked form/table. As with the map-to-table link, in most instances the **user** does not need to know any of this information other than the map name and, in some cases, the form name. If the user does need to specify any of this information, utility programs are provided to display the needed information. In most cases, the interface programs access and use this link information transparently. In order to conveniently view the indirect form-to-map links, a DBMS user view (*grass_to_form_link*) has also been created. The information in this user view is generated, automatically, from the tables *grass_to_dbms_link* and *form_to_dbms_link*.

General interface operation

This section discusses, in general terms, how the category values are used by the interface during processing. Most of this occurs automatically, and is totally transparent to the user. The sequential steps that occur when the interface is run are described below.

1. The user types in the interface command and includes any command line options desired. For example:

db.interface map=panther_obs method=area where="Date >= '01-Jan-91'"

2. The GRASS spatial selection method (just "method" in the following discussion) is started by the interface. Once started, the method controls subsequent events. It tells the interface what to do and when to do it (e.g., apply the SQL filter, display DBMS data, display GRASS features).

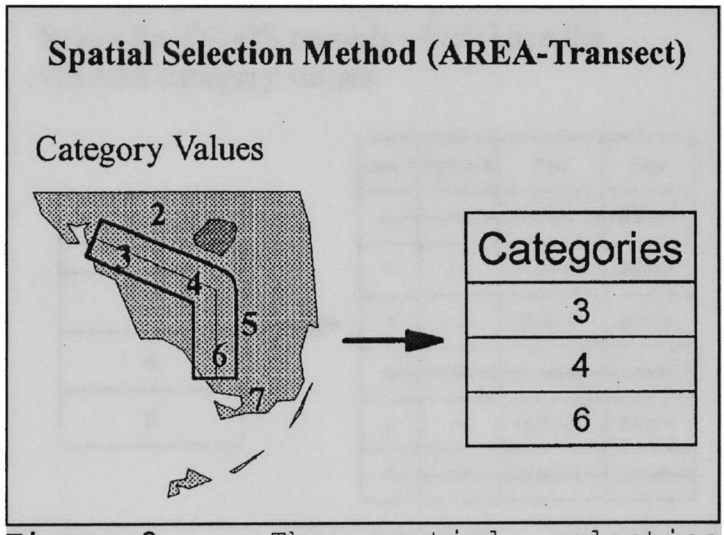


Figure 8. The spatial selection method is used to create a tentative set of category values.

The method is applied to the GRASS map to create the initial, tentative set of category values. For example, the AREA method (transect option) would return the category values of all features occurring in the specified transect area (Figure 8). The spatial selection method determines

the specific map **features** that are selected, and then finds the **category values** of these features. Once the category values have been determined, there are no more **spatial** distinctions made among features. For example, if a feature is selected using the AREA method (and it passes the non-spatial criteria also) and it has a category value of 22, then **any** GRASS feature in the map with a category value of 22 will be highlighted on the GRASS map, **whether it occurs in the selected area or not.** This is why it is important to assign unique category values to every feature in a map.

3. The tentative set of categories found above is passed to the DBMS selection function which applies the non-spatial SQL query selection criteria to the set of values. This results in the final set of category values (Figure 9 and Figure 10).

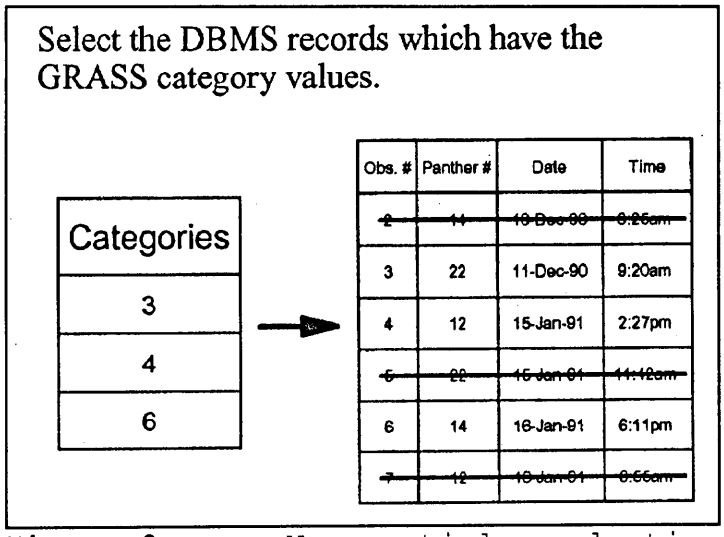


Figure 9. Non-spatial selection processing, part 1. Only those records having the selected category values are retained.

4. The final set of category values is sent to the appropriate program functions to be used to:

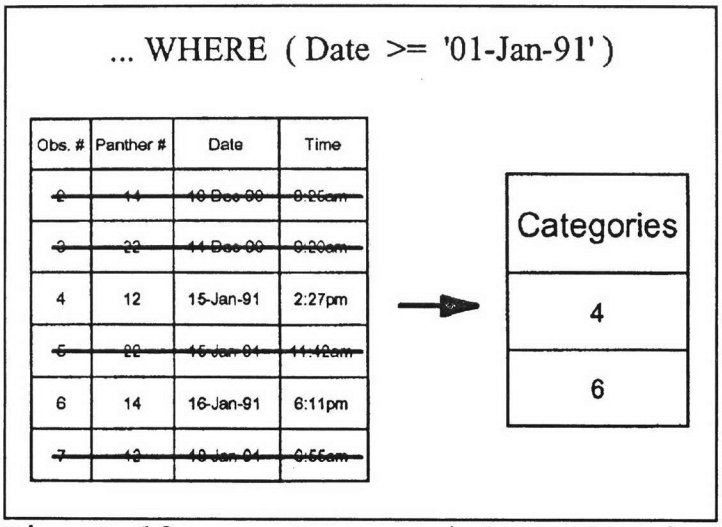


Figure 10. Non-spatial selection processing, part 2. The WHERE clause is applied to refine the selection. This creates the final set of category values.

a. Display/highlight the corresponding GRASS map features in the GRASS graphics display window (Figure 11).

b. Select and display the corresponding DBMS data in the text display window, either in the simple line-based output (Figure 11)

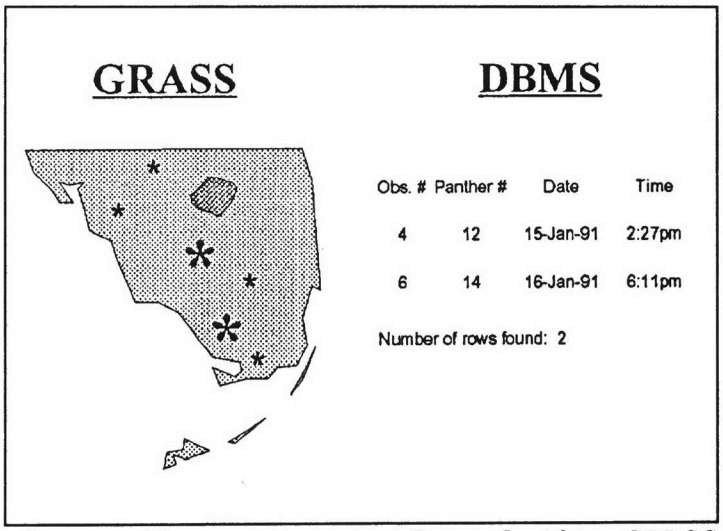


Figure 11. An example of the GRASS and DBMS data display. The GRASS map would be displayed in a graphics window, the DBMS data in a separate text window.

or the form-based output, depending on which program was used.

Query Formation

The general operation of the interface when performing a query was discussed above. This section describes specifically how the DBMS query is formulated and executed.

The spatial selection methods control what the interface does, and when. The interface, for this reason, can be considered a slave to the selection methods. The selection methods generate the tentative list of category values for a query. The procedure used to generate this list is, of course, a function of the particular method used. When a selection method selects a category value, the value is passed to the interface where it is initially stored as a member of a doubly linked list. Before a value is added to the list, however, the list is scanned to check if the value is already stored. If it is, the duplicate is not added to the list. The selection method will continue passing category values to the interface in this manner until it is finished its selection process. Note that a doubly linked list is not an optimal storage structure for this activity. The speed of "insert" and "member" operations are of O(N),

where N is the number of categories in the list. A tree-based implementation such as a 2-3 tree would provide $O(\log_2 N)$ performance. A switch to a more efficient implementation can be made relatively easily, since the data structure is not directly used by client functions. All needed operations on the list are provided by functions, so the implementation can be changed without affecting any functions which use the list. The programs would need to be re-compiled, however, if the implementation were changed.

At the completion of the spatial selection process, therefore, the doubly linked list will contain the set of category values selected by the method. The method then requests that the interface apply the "SQL filter." The SQL filter is an SQL SELECT statement that is generated by the interface based on the categories in the list and the user's non-spatial selection criteria (i.e., the FROM and WHERE clauses of the SELECT statement). To perform the filter, the categories in the list are first inserted into a DBMS table referred to as a "category table." The actual name of this table in the database can be set in the "settings" files described later. For our discussion, it will be called CAT_1. This table has a single column named "category", and is created automatically the first time a user uses the interface. A separate CAT_1 table is created for each user.

Once created, CAT_1 is used for all future interface sessions, as it is not dropped when the interface terminates.

The interface retrieves the link information (e.g., the base table name and category-column name) from the GRASS-to-DBMS link table and begins to form the SELECT statement. Any user-specified joins of the base table with other tables are built into the statement. The generated SELECT statement also includes a join with CAT_1 such that the only DBMS records retrieved are those which have category values that are also contained in CAT_1. The only column selected by this SQL filter is the category value column from the base DBMS table.

The generated SELECT statement is actually a subquery of an INSERT statement. The category values retrieved by the SELECT statement are inserted into a second category table which we will call CAT_2. The CAT_2 table is identical in structure to CAT_1, and is also automatically created for each user the first time he/she uses the interface. Upon completion of the query, therefore, CAT_2 contains only the category values that were selected by the spatial selection method and that passed the SQL filter. The category values in CAT_2 can be used in any way needed by the selection method. In most cases, at the request of the selection

method, the category values in CAT_2 are used to highlight the GRASS map features on the graphics screen.

At the request of the selection method, the CAT_1 table is next used in another query of the DBMS database in order to display the DBMS data. The generated SELECT statement is very similar to the SELECT statement generated for the SQL filter. In this case, however, the SELECT statement is not a subquery of an INSERT statement. The major difference in the SELECT statement itself is that instead of selecting only the category values, all of the user-specified table columns are selected. The only other difference is that this SELECT statement would include the ORDER-BY clause if one was specified by the user. The ORDER-BY clause is not allowed in subqueries of an INSERT statement. The FROM and WHERE clauses are identical for both queries. The query is formed, executed, and the results are displayed in the text window.

Note that the method used here to connect the categories of a GRASS map to the attributes in DBMS tables can be thought of as a relational join from the DBMS to the GRASS map itself. Although the mechanics of performing this operation are complex, the analogy of a join is useful and valid. From this perspective, the GRASS map is simply another table that can be queried in the combined GRASS/DBMS domain, and the spatial selection methods act as spatial "extensions" to SQL.

Interface Software Structure

The basic structure of the interface programs is shown in Figure 12. This figure shows the interface system at a high level of abstraction which includes the main software modules and the relationships among them. The ovals represent the two separate versions of the interface, *db.interface* and *db.forms.interface*. The boxes represent the major functional units (modules) which form the programs. In each box is a list of one or more C program files which are the major components of the module. Each file contains a number of related functions which perform certain types of tasks. There are many more C files used to create the programs but these are not shown. The files that are not shown are primarily library functions which perform various support tasks for the modules shown. The solid lines between boxes indicate that the modules are linked to form a single executable program. The "pipes" between boxes indicate that the modules on either side are separate processes that communicate via UNIX pipes. The arrows indicate that the module at the base of the arrow executes the process at the head of the arrow.

Both versions of the interface, *db.interface* and *db.forms.interface*, are shown in Figure 12. These two

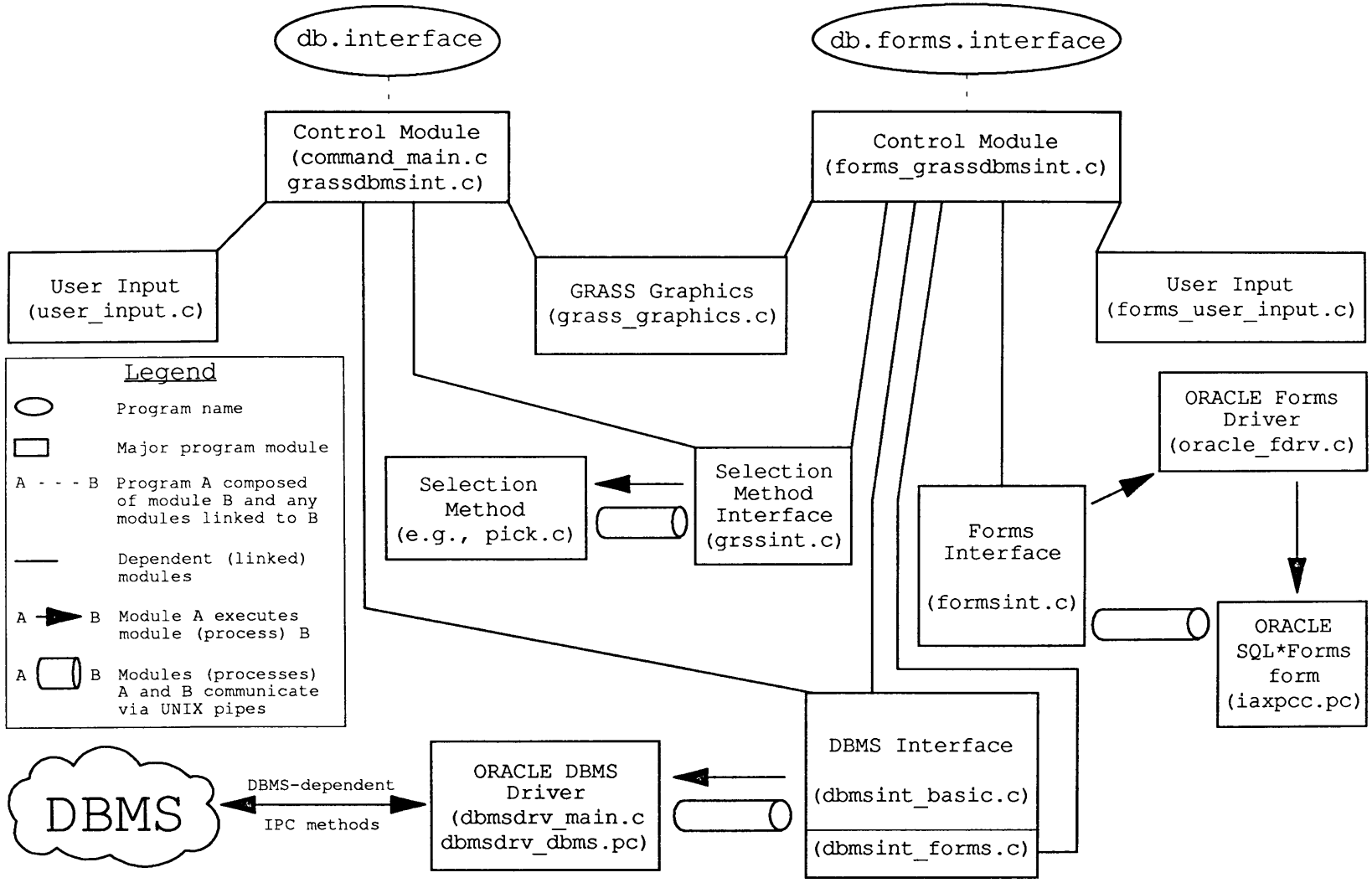


Figure 12. High-level chart showing the basic structure of the two interface programs.

programs are created separately but, as the diagram illustrates, they both use the same major modules. The *db.forms.interface* program requires some additional modules for interfacing with the screen forms, and requires slightly modified versions of the interface-control and user-input modules. The DBMS driver and all of the spatial selection method drivers are the same for both versions of the interface.

The interface has been implemented using ORACLE, but the system was designed to enable the use of other DBMS software. For this reason, all DBMS-specific information and functions were placed into a single DBMS driver module. This DBMS-specific driver accesses the DBMS in whatever way is required by the particular DBMS. The ORACLE DBMS driver was implemented using ORACLE's Pro*C software, which allows a programmer to embed SQL commands into a C program. The Pro*C Precompiler converts the SQL commands into C calls to the ORACLE DBMS. The precompiled version of the driver is then compiled with a standard C compiler. The file *dbmsdrv_main.c* contains driver functions containing only C code. The file *dbmsdrv_dbms.pc* contains those driver functions which make DBMS calls, and is the file which must be precompiled.

The DBMS driver is executed by the interface at runtime, and communicates with the interface via UNIX pipes. All

communication between the interface and the DBMS driver is performed by functions in *dbmsint_basic.c* and *dbmsint_forms.c* via calls to a set of high-level communication functions in the file *hilevel_pipe.c* (not shown). Note that the functions in *hilevel_pipe.c* are used for all interprocess communication for this system. By implementing the DBMS interaction functions in this way, any DBMS can be used as long as a driver is written for it that supports the database query requests from the interface (i.e., ANSI standard SQL SELECT statements). The particular DBMS driver to be used when running the interface is specified in system-wide and/or user-specific "settings" files.

When either version of the interface is run, the command line may include a spatial selection method name. If no method is specified on the command line, a system-specified, or user-specified default method is used. The appropriate method driver is executed by a function in the *grssint.c* file. All communication between the interface and the method is performed by other functions in *grssint.c* via calls to the set of communication functions in the file *hilevel_pipe.c*.

The SQL*Forms version of the interface, *db.forms.interface*, also uses a driver process to execute the DBMS screen forms. Although this project proposal did not originally include a generic screen-form capability (only an ORACLE-specific

screen-form capability was proposed), a generic forms-interface structure **was** implemented. A function in *formsint.c* executes a DBMS-specific forms driver process which executes the form and communicates with the interface via UNIX pipes using the *hilevel_pipe.c* functions.

In the ORACLE implementation, a two-step process is required. The forms-driver (*oracle_fdrv.c*) is executed and it, in turn, executes a modified version of the ORACLE SQL*Forms program. The SQL*Forms program was modified using standard SQL*Forms and Pro*C techniques that enable a programmer to link functions, called "user exits", into SQL*Forms. Four interface-to-forms communication functions were linked into the ORACLE SQL*Forms program. These functions are contained in the file *iaxpcc.pc*. The user exits implemented for this project are form-independent. All ORACLE screen forms that are linked to a GRASS map can use the same SQL*Forms program without modification. Each linked screen form itself, however, must be created or modified to make use of the user exits, as this is **not** an automatic process. Note that the same SQL*Forms program can be used by any ORACLE screen form, whether it is linked to a GRASS map or not. The ability to link with a GRASS map is strictly an add-on capability that does not affect the use of the SQL*Forms program in any way for non-linked forms.

The interaction of the interface with a screen form is complex and is constrained by the limitations and restrictions of the SQL*Forms program. It was much more difficult to create the interface to the ORACLE screen-form than it was to create the basic interface. Other DBMS programs may be either easier or harder than ORACLE to link with the screen-form interface. It may be impossible to link some programs to the forms interface. The structure of the interaction with different DBMS screen forms has been implemented, however, and the particular DBMS forms-driver executed is based on the DBMS specified in the "settings" files.

The implementation of the DBMS driver, screen-form driver, and the selection method drivers meets the system design objectives outlined previously. The implementation allows other DBMS drivers and selection methods to be developed without any need to modify the interface programs. Any new drivers that are developed in the future can make use of library functions which handle the driver-side communication and other common tasks. This allows the developer to concentrate on the driver-specific operations, and not the "overhead" tasks needed for all drivers.

Most of the graphics tasks, such as displaying the GRASS maps and highlighting the selected GRASS map features, are handled

by functions contained in *grass_graphics.c*. Many of these operations are performed at the request of the selection method drivers. For example, it is the selection method drivers that request that the selected map features be highlighted. The system was implemented in this way so that selection method programmers could implement new methods easily without having to deal with low-level graphics processing for common tasks. Method-specific graphics operations such as used in the AREA selection method (e.g., allowing the user to draw a complex polygon on the screen), however, are performed locally by the selection method, not by the functions in *grass_graphics.c*. Some of these method-specific graphics tasks are available as library functions which can be used by future programmers.

The command-line input is processed by two different functions. The basic interface uses the function in *user_input.c*, the forms interface uses the function in *forms_user_input.c*. Both of these functions use the standard GRASS command-line parser, a library function which is included as part of the GRASS distribution. These two files specify all of the valid command-line arguments for the programs. If a user chooses not to specify the command-line arguments, the parser automatically starts a simple interactive user input process. For each argument, the parser prompts the user for input. Included with the prompt

is a list of any specified options and/or the default for the argument. After the user enters each response, the parser displays the response and requests confirmation of the answer. In this way, all arguments are input to the program. The parser is very simplistic, but it provides a consistent "look and feel" to all GRASS programs. This is an important, but difficult, objective to achieve when programs are developed by dozens of different programmers across the country. The standard command-line parser helps to achieve this objective. Although consistency is a valid goal, the line-based input of the parser is rather primitive in the present, graphically oriented, computer environment. For this reason, a graphical user interface was also developed for the interface programs.

The graphical user interfaces were developed using Xgen. Xgen is a public domain Motif-based program used for the rapid creation of user interfaces. Creating the interface consists of developing an interface description script. This script defines the characteristics of the interface to be implemented. The script is then interpreted by Xgen at runtime. The scripts which were developed for this project simply collect the information needed for the command-line arguments. The information is then used to form a command-line for the standard interface programs, which are executed by Xgen. It is important to recognize that the Xgen scripts

are merely "on top of" the interface programs, they are not modified versions of the programs.

This interface system is intended for distribution to the entire GRASS user community. In order to make this practical, sufficient flexibility must be incorporated into the system to allow customization to meet the needs or preferences of different installations and users. Some of the main features which allow this flexibility, such as using DBMS and method drivers, have already been discussed. In addition to this, the system supports the use of four types of "settings" files which specify values for a wide variety of variables. The particular DBMS being used for a session is just one of these values that has been mentioned previously. Other variables include DBMS schema, table, and form names, map feature highlight color, and the default spatial selection method. The four types of settings files, in increasing priority order, are: (1) system-wide, DBMS-independent; (2) system-wide, DBMS-specific; (3) user, DBMS-independent; (4) user, DBMS-specific. Any of the variables can be set in any of the files, but a variable which is set in a lower priority file will be overridden by a value for the variable set in a higher priority file. This capability allows the system administrator to define global default settings for the installation and any settings that vary from one DBMS to another (e.g., ORACLE may have one set of table

names, Informix a different set). Users can override these system-wide settings by creating their own settings files. This would allow the user to use a private DBMS database rather than the system-wide database, for example. More commonly, a user might specify his/her preferred database-output delimiter and map feature colors.

In addition to the settings files, DBMS-specific terminal mapping files exist for the interface system. One file is needed for each DBMS used. Each file contains terminal-definition to DBMS-definition mappings needed for the forms version of the interface. Each entry in this file consist of a terminal definition (as stored in the TERM environment variable) followed by the corresponding value used by the DBMS for displaying screen forms. This setup may not be needed by some DBMS form programs, and it may not be adequate for others, but the format is general enough that it should be adequate for most systems. The systems administrator must modify these files to add the terminal mappings for the particular terminals used at his/her installation.

PROGRAM DESCRIPTIONS

As noted previously, this interface software system consists of a variety of programs that work together or alone to

provide a number of services to the user. The system consists not only of the interface programs, but also of support programs for managing the links between GRASS and the DBMS, and for creating new reclassified maps. Many programs have two versions, one that provides line-based DBMS output and one for SQL*Forms forms-based output. All programs were developed using the standard GRASS parser in order to retain the same "look and feel" as other GRASS programs. Therefore, program options can be entered either as command line arguments or as responses to prompts from the parser. Xgen graphical user interface scripts are also provided "on top of" most of the programs to provide a more convenient way of using the programs, particularly for novice users.

The programs were developed on a SUN SPARCstation 2 using SunOS 4.1.2, GRASS version 4.0, ORACLE version 6.0, ORACLE Pro*C version 1.3, and ORACLE SQL*Forms version 3.0. You must start GRASS prior to running any of these programs. The programs that form the software system are listed below.

- **Interface programs**

  These two programs are the core of the system. These are the programs which users use to query and display data from the joint GRASS/DBMS database.

- *db.interface*:

  This version of the interface produces line-based DBMS output to the screen. This version also allows the user to print or file the DBMS output data, and use the DBMS data to create a new reclassified GRASS map.

- *db.forms.interface*:

  This version of the interface produces ORACLE SQL*Forms-based DBMS output. This version does not allow the user to print or file the DBMS output data or use the DBMS data to create a new reclassified GRASS map. The user can, however, step through the DBMS output one record at a time and save any changes to the DBMS data. To use this version of the program with other DBMS packages, a screen-forms driver would need to be developed in addition to the new DBMS driver.

Both of these programs support the same spatial selection methods. The methods currently available are described below. Methods are implemented as drivers, so additional methods can be developed easily without the need to modify the interface itself. The spatial selection method selects an initial, tentative, set of map features which is then

further refined by the SQL SELECT statement clauses provided by the user, if any. All of these methods can be used with any map type (raster, vector, or site).

- PICK:

  The user uses a mouse to select one or more specific GRASS map features from the GRASS display screen.

- AREA:

  The user uses a combination of mouse and/or keyboard input to draw areas of interest on the GRASS display screen. The user has four ways to define the areas. Any combination of one or more areas is allowed for any given query (e.g., two circles, three polygons, and one transect).

  - Polygon: Using the mouse, the user can draw a complex polygon on the screen around an area of interest.

  - Transect: The user has the option of either square or rounded ends of the transect. Using the mouse, the user can draw a multi-segment line along a path of interest. A buffer width for the

transect is entered at the keyboard, the transect path is calculated using the buffer value, and the transect boundaries are displayed on the screen. The full transect area is then used by the interface to select the initial set of features.

- Subregion: Using the mouse, the user specifies two diagonal corners of a rectangular area. The edges of the rectangle are calculated and displayed on the screen.

- Circle: Using the mouse and/or keyboard, the user specifies the center point and radius of a circle. The circle is calculated and displayed on the screen.

- OVERLAY:

The user uses an existing vector map's area features to select GRASS features from the linked map of interest (raster, vector, or site map). The vector map is "overlaid" on the linked map. The user can specify the areas of interest from the overlaid vector map in three ways:

- Pick: Using the mouse, the user can select one or more specific areas from the overlay vector map. These specified areas will be used to select features from the underlying linked map.

- All: All areas in the vector map are used for the selection of features from the linked map.

- Category file: Prior to running the interface, the user can create a file containing a list of category values. These category values correspond to the category values of areas in the vector overlay map. When the user runs the interface and specifies the category file name, all areas in the overlay vector map that have a category value matching a value contained in the file will be used to select features from the underlying linked map. The overlay vector map must be labeled for this option to work.

- REGION:

All features of the linked map within the currently defined GRASS region (called a window prior to GRASS 4.0) are selected. The GRASS region is the "active"

geographic area being used by GRASS. Anything outside of the region is ignored.

- MAP:

  All features within the linked GRASS map are selected. In this case, DBMS information may be displayed for GRASS features which fall outside of the current region and, therefore, are not highlighted in the GRASS display window.

- NONE:

  No spatial selection is made. All features within the linked DBMS table are used as the initial set of features prior to applying the SQL SELECT statement. In this case, DBMS information may be displayed for GRASS features which do not occur in the current map or current region and, therefore, are not highlighted in the GRASS display window.

- **Link support programs**

There a number of support programs needed for this system. These programs are used for creating, modifying, deleting,

and viewing the links between GRASS maps and DBMS tables and between SQL*Forms screen forms and DBMS tables. This link information is needed by the interface programs to perform the joint GRASS/DBMS queries. These programs can also be used by users to determine what information is available in the linked GRASS/DBMS database.

- Line-based input/output support programs for the basic interface:

  - *db.makelink*: Allows creating and deleting GRASS-map to DBMS-table link records.

  - *db.viewlinks*: Allows viewing GRASS-map to DBMS-table link records.

- ORACLE SQL*Forms related programs for managing links. You must have the ORACLE SQL*Forms option to use these programs. Two methods of managing links are provided.

  - Forms -- Standard ORACLE forms have been created for accessing this information. These forms can be used by running the program *db.links*. This program allows the user to use any one of the three link table/view forms (described below) that are provided with this system. The

particular form is specified by a command-line argument.

The *grass_to_dbms_link* form supports creating, modifying, deleting, and viewing the GRASS-map to DBMS-table link records. This form provides all of the capabilities of *db.makelink* and *db.viewlinks*, but uses a screen form instead of line-based input and output.

The *form_to_dbms_link* form supports creating, modifying, deleting, and viewing the ORACLE SQL*Form to DBMS-table link records.

The *grass_to_form_link* form supports viewing (only) the indirect GRASS-map to ORACLE SQL*Form link records. These links are actually just a DBMS "user view" created from the real links in *grass_to_dbms_link* and *form_to_dbms_link*.

- Line-based input/output programs -- This program provides an alternative method of viewing GRASS-map to SQL*Form link records.

- *db.view_gf_link*: Displays GRASS-map to ORACLE SQL*Form link records from the ***g****rass_to_****f****orm_link* user view.

- **Map reclass programs**

These can be used either as stand-alone programs or from within *db.interface* (but NOT from within *db.forms.interface*). These programs were developed primarily to support direct creation of reclassified maps from within the GRASS to DBMS interface. The reclassification is based on the selected features' DBMS information.

These new maps can then be used with any appropriate GRASS program for display or analysis. In this way, the interface provides "indirect" support for GRASS map analysis.

- *db.v.reclass*:

  Reclasses GRASS vector maps based on DBMS output information.

- *db.s.reclass*:

  Reclasses GRASS site maps based on DBMS output information.

- *db.reclass*:

  An interface to *db.v.reclass*, *db.s.reclass*, and *r.reclass* (the standard GRASS raster map reclass program). Using this program, reclassifications can be performed based on integer, floating point, or text (alphanumeric) fields of the database. For floating point and text data, the program automatically creates an integer category value, which is required by GRASS, for each unique floating point value or text string (case sensitive or insensitive); the floating point values or text strings can optionally be retained as labels for the categories.

- **Xgen graphical user interface programs**

  - *db.menu*:

    This is an Xgen interface to the line-based programs. In addition, *db.menu* provides the capability of saving

db.interface "sessions" for later use. All of the information used for a query (map and table names, spatial selection method, SQL clauses, etc.) is saved in a session file. This session file can be retrieved and used at a later time. This is particularly useful for designers, who can create standard sessions for use by other, inexperienced, users. The line-based programs accessed by *db.menu* are *db.interface*, *db.makelink*, and *db.viewlinks*.

- *db.forms.menu*:

This is an Xgen interface to the ORACLE SQL*Forms programs and forms. In addition, *db.forms.menu* provides the capability of saving *db.forms.interface* "sessions" for later use. All of the information used for a query (map and form names, spatial selection method, etc.) is saved in a session file (SQL clauses ARE NOT saved). This session file can be retrieved and used at a later time. This is particularly useful for designers, who can create standard sessions for use by other, inexperienced, users. The line-based programs accessed by *db.forms.menu* are *db.forms.interface*, *db.links*. You must have the ORACLE SQL*Forms option to use this program.

FUTURE WORK

There are a number of possible improvements and enhancements to this software which may be done in the future, some of these are listed below.

- Enhance/improve existing spatial selection methods. Some of the algorithms currently used need to be made more efficient. For example, plane-sweep methods for detecting rectangle intersections, as outlined in [Samet-90], may be used to improve the speed of the AREA and OVERLAY selection methods.

- Add support for latitude/longitude GRASS databases. This requires modifications to the spatial selection methods.

- Add support for site location coordinates (UTM and latitude/longitude) contained within the DBMS tables (i.e., no GRASS map will be needed).

- Add automatic vector buffer generation around existing vector map features (e.g., automatically generate a 500 meter buffer on either side of an interactively user-specified road).

- Add support for a file of coordinates to be used to generate polygons, subregions, transects, and circles in the AREA spatial selection method.

- Add full computer network support (some support is built-in already).

- For the SQL*Forms version of the interface, highlight the individual map features corresponding to each form record as the user steps through the records (at this time, all of the selected features are highlighted on the GRASS display, the user does not know which specific selected feature(s) correspond to which specific DBMS record).

- Allow interactive selection of DBMS tables, columns and SQL operators for query formation and display when using *db.menu*.

SUMMARY

This software system provides an easy to use interface between the GRASS GIS and ORACLE DBMS. Other DBMS software can be used if appropriate DBMS drivers are written. The interface allows the user to apply both spatial and non-

spatial criteria for any query of the database. Spatial criteria are specified by using one of the spatial selection methods. There are currently six spatial selection methods provided: PICK, AREA, OVERLAY, REGION, MAP, and NONE. The AREA and OVERLAY methods have a number of sub-methods available for selecting the features of interest. Non-spatial criteria are specified using ANSI standard SQL. The results of the query are displayed both graphically and textually. Using the basic interface, *db.interface*, the user can view, print, or file the DBMS output, or use it to directly create a new reclassed GRASS map. Using the ORACLE SQL*Forms interface, *db.forms.interface*, the user can view and update the DBMS attribute data for the records retrieved during a query.

REFERENCES

[Abel-89] D.J. Abel. "SIRO-DBMS: A Database Tool-Kit for Geographical Information Systems". Int. J. Geographical Information Systems, vol. 3, no. 2, 1989. (pp. 103-116)

[Abel/Smith-86] D.J. Abel and J.L. Smith. "A Relational GIS Database Accommodating Independent Partitionings of the Region". Proceedings: Second International Symposium On Spatial Data Handling, 1986. (pp. 213-224)

[Anthony/Corr-88] S.J. Anthony and D.G. Corr. "Data Structures in an Integrated Geographical Information System". ESA Journal, vol. 12, no. 1, 1988. (pp. 69-72)

[Armstrong/Densham-90] M.P. Armstrong and P.J. Densham. "Database Organization Strategies for Spatial Decision Support Systems". Int. J. Geographical Information Systems, vol. 4, no. 1, 1990. (pp. 3-20)

[Burrough-86] P.A. Burrough. *Principles of Geographical Information Systems for Land Resources Assessment.* Monographs on Soil and Resources Survey No. 12. Oxford University Press, New York, NY, 1986.

[Clarke-86] K.C. Clarke. "Recent Trends in Geographic Information System Research". Geo-Processing, vol. 3, 1986. (pp. 1-15)

[Cowen-88] D.J. Cowen. "GIS Versus CAD Versus DBMS: What Are the Differences?". Photogrammetric Engineering and Remote Sensing, vol. 54, no. 11, 1988. (pp. 1551-1555)

[Dangermond-86] J. Dangermond. "Geographic Database Systems". Geo-Processing, vol. 3, 1986. (pp. 17-29)

[Dangermond/Freedman-86] J. Dangermond and C. Freedman. "Findings Regarding a Conceptual Model of a Municipal Database and Implications for Software Design". Geo-Processing, vol. 3, 1986. (pp. 31-49)

[Farley-93] J. Farley. *Workshop V: Data Base Management and GRASS.* Unpublished workbook for a workshop at the 8th Annual GRASS GIS User's Conference and Exhibition, Reston, Virginia, 1993.

[Frank-88] A.U. Frank. "Requirements for a Database Management System for a GIS". Photogrammetric Engineering and Remote Sensing, vol. 54, no. 11, 1988. (pp. 1557-1564)

[Gerdes-91] D. Gerdes. *GRASS Vector Library Changes (Beta Version)*. Unpublished report. Construction Engineering Research Laboratory, Champaign, Illinois, 1991.

[GISWorld-89A] Anonymous. "TIGER Promises Roar". GIS World, vol. 2, no. 1, 1989.

[GISWorld-89B] Anonymous. "The U.S. Federal Agencies". GIS World, vol. 2, no. 2, 1989.

[Goh-89] P.C. Goh. "A Graphic Query Language for Cartographic and Land Information Systems." Int. J. Geographical Information Systems, vol. 3, no. 3, 1989. (pp. 245-255)

[Goran-92] B. Goran. "New GRASS Seeds". GRASSClippings, J. Open Geographic Information Systems, vol. 6, no. 3, 1992. (p. 6)

[Haralick-80] R.M. Haralick. "A Spatial Data Structure for Geographic Information Systems". In H. Freeman and G.G. Pieroni (eds.), *Map Data Processing*, Academic Press, New York, 1980.

[Jackson/Mason-86] M.J. Jackson and D.C. Mason. "The Development of Integrated Geo-Information Systems". Int. J. Remote Sensing, vol. 7, no. 6, 1986. (pp. 723-740)

[Jackson *et al*-88] M.J. Jackson, W.J. James, and A. Stevens. "The Design of Environmental Geographic Information Systems". Philosophical Transactions of the Royal Society of London, series A, vol. 324, 1988. (pp. 373-380)

[SFRC-90] South Florida Research Center. *An Assessment of Hydrological Improvements and Wildlife Benefits from Proposed Alternatives for the U.S. Army Corps of Engineers' General Design Memorandum for Modified Water Deliveries to Everglades National Park.* U. S. National Park Service, Everglades National Park, Homestead, Florida, 1990.

[Joseph/Cardenas-88] T. Joseph and A.F. Cardenas. "PICQUERY: A High Level Query Language for Pictorial Database Management". IEEE Transactions on Software Engineering, vol. 14, no. 5, 1988. (pp. 630-638)

[Keating *et al*-87] T. Keating, W. Phillips, and K. Ingram. "An Integrated Topologic Database Design for Geographic Information Systems". Photogrammetric Engineering and Remote Sensing, vol. 53, no. 10, 1987. (pp. 1399-1402)

[Lorie/Meier-84] R.A. Lorie and A. Meier. "Using a Relational DBMS for Geographical Databases". Geo-Processing, vol. 2, 1984. (pp. 243-257)

[Maffini-87] G. Maffini. "Raster Versus Vector Data Encoding and Handling: A Commentary". Photogrammetric Engineering and Remote Sensing, vol. 53, no. 10, 1987. (pp. 1397-1398)

[Monmonier-82] M.S. Monmonier. *Computer-Assisted Cartography: Principles and Prospects.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.

[Muller/O'Connor-82] J.R. Muller and R.P. O'Connor. "Representing Topological Properties in Raster Data Structures". In J. Foreman (ed.) , *ISPRS IV - Proceedings*, International Society for Photogrammetry and Remote Sensing and American Congress on Surveying and Mapping, Falls Church, Virginia, 1982.

[Nagy/Wagle-79] G. Nagy and S. Wagle. "Geographic Data Processing". Computing Surveys, vol. 11, no. 2, 1979. (pp. 139-181)

[NPS-93] National Park Service. *National Park Service GIS Sourcebook.* NPS Geographic Information Systems Division, Denver, 1993.

[Peuquet-84] D.J. Peuquet. "Data Structures for a Knowledge-Based Geographic Information System". Proceedings of the International Symposium On Spatial Data Handling, Volume II, 1984. (pp. 372-391)

[Peuquet-86] D.J. Peuquet. "The Use of Spatial Relationships to Aid Spatial Database Retrieval". Proceedings: Second International Symposium On Spatial Data Handling, 1986. (pp. 459-471)

[Robinson *et al*-86] V.B. Robinson, M. Blaze, and D. Thongs. "Representation and Acquisition of a Natural Language Relation for Spatial Information Retrieval". Proceedings: Second International Symposium On Spatial Data Handling, 1986. (pp. 472-487)

[Roussopoulos *et al*-88] N. Roussopoulos, C. Faloutsos, and T. Sellis. "An Efficient Pictorial Database System for PSQL". IEEE Transactions on Software Engineering, vol. 14, no. 5, 1988. (pp. 639-650)

[Samet *et al*-84] H. Samet, A. Rosenfeld, C.A. Shaffer, and R.E. Webber. "Use of Hierarchical Data Structures in Geographical Information Systems". Proceedings of the International Symposium On Spatial Data Handling, Volume II, 1984. (pp. 392-411)

[Samet-90] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[Schell-92] D. Schell. "Greetings from The Open GRASS Foundation". GRASSClippings, J. Open Geographic Information Systems, vol. 6, no. 3, 1992. (p. 3)

[SCS-90] Soil Conservation Service. *GRASS Geographic Information System.* SCS Cartography and GIS Division, 1990.

[Shapiro-80] L.G. Shapiro. "Design of a Spatial Information System". In H. Freeman and G.G. Pieroni (eds.), *Map Data Processing*, Academic Press, New York, 1980.

[Shapiro *et al*-89] M. Shapiro, J. Westervelt, D. Gerdes, M. Higgins, and M. Larson. *GRASS 3.0 Programmer's Manual.* USACERL ADP Report N-89/14. Construction Engineering Research Laboratory, Champaign, Illinois, 1989.

[Shapiro/Westervelt-91] M. Shapiro and J. Westervelt. *GRASS Programming Changes.* Unpublished report. Construction Engineering Research Laboratory, Champaign, Illinois, 1991.

[Smith *et al*-87] T.R. Smith, S. Menon, J.L. Star, and J.E. Estes. "Requirements and Principles for the Implementation and Construction of Large-Scale Geographic Information Systems". Int. J. Geographical Information Systems, vol. 1, no. 1, 1987. (pp. 13-31)

[USArmy-91] U. S. Army Corps of Engineers. *GRASS Version 4.0: Geographic Resources Analysis Support System User's Reference Manual.* Construction Engineering Research Laboratory, Champaign, Illinois, 1991.

[VanRoessel-87] J.W. Van Roessel. "Design of a Spatial Data Structure Using the Relational Normal Forms". Int. J. Geographical Information Systems, vol. 1, no. 1, 1987. (pp. 33-50)

[Waugh/Healey-87] T.C. Waugh and R.G. Healey. "The GEOVIEW Design: A Relational Data Base Approach to Geographical Data Handling". Int. J. Geographical Information Systems, vol. 1, no. 2, 1987. (pp. 101-118)

[Webster-88] C. Webster. "Disaggregated GIS Architecture: Lessons From Recent Developments in Multi-Site Database Management Systems". Int. J. Geographical Information Systems, vol. 2, no. 1, 1988. (pp. 67-79)

[Wu *et al*-89] J.-K. Wu, T. Chen, and L. Yang. "A Versatile Query Language for a Knowledged-Based Geographical Information System." Int. J. Geographical Information Systems, vol. 3, no. 1, 1989. (pp. 51-57)