Florida International University
FIU Digital Commons

FIU Electronic Theses and Dissertations | University Graduate School

11-20-2000

Efficient storage and retrieval of georeferenced objects in a semantic database for web-based applications

Debra Lee Davis
Florida International University

DOI: 10.25148/etd.FI14062210
Follow this and additional works at: https://digitalcommons.fiu.edu/etd

 Part of the Databases and Information Systems Commons

Recommended Citation
Davis, Debra Lee, "Efficient storage and retrieval of georeferenced objects in a semantic database for web-based applications" (2000). *FIU Electronic Theses and Dissertations*. 2744.
https://digitalcommons.fiu.edu/etd/2744

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

EFFICIENT STORAGE AND RETRIEVAL OF GEOREFERENCED OBJECTS IN A SEMANTIC DATABASE FOR WEB-BASED APPLICATIONS

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Debra Lee Davis

2000

To: Dean Arthur W. Herriott
College of Arts and Sciences

This thesis, written by Debra Lee Davis, and entitled Efficient Storage and Retrieval of Georeferenced Objects in a Semantic Database for Web-Based Applications, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Naphtali Rishe

Shu-Ching Chen

Nagarajan Prabakar, Major Professor

Date of Defense: November 20, 2000

The thesis of Debra Lee Davis is approved.

Dean Arthur W. Herriott
College of Arts and Sciences

Interim Dean Samuel S. Shapiro
Division of Graduate Studies

Florida International University, 2000

© Copyright 2000 by Florida International University

High Performance Database Research Center

All rights reserved.

DEDICATION

I dedicate this thesis to my husband, David Chu, Jr., and my grandmother, Maria Gil. They have supported me and encouraged me in all my endeavors, and have helped me reach for the stars and fulfill my dreams. I am eternally grateful to them and love them to the very depth of my soul.

ACKNOWLEDGMENTS*

I would like to thank everyone who has played a role in helping me accomplish this important work. If it wasn't for the support of those around me, I would not be where I am today.

I particularly would like to thank Dr. Nagarajan Prabakar. He has spent many hours guiding me and helping me accomplish this work, as well as providing much needed advice and mentoring for my education and career. I also extend a big thank you to Dr. Naphtali Rishe, Dr. Maxim Chekmasov and Dr. Shu-Ching Chen for their advice and guidance in this work.

There are many others to whom I am grateful. My biggest thanks goes to my husband, David Chu, Jr., and my grandmother, Maria Gil. Without their patience, nurturance and understanding, I would never have made it this far. Many thanks go to Dmitre Raposo, Dmitry Vasilevsky and Dr. Andriy Selivonenko for their technical support and encouragement, and to my friends Martha Gutierrez, Jorge Vidal, Tin Ho and many others, for their encouragement and emotional support.

*This research was supported in part by NASA (under grants NAG5-9478, NAGW-4080, NAG5-5095, NAS5-97222, and NAG5-6830) and NSF (CDA-9711582, IRI-9409661, HRD-9707076, and ANI-9876409).

ABSTRACT OF THE THESIS

EFFICIENT STORAGE AND RETRIEVAL OF GEOREFERENCED OBJECTS IN A SEMANTIC DATABASE FOR WEB-BASED APPLICATIONS

by

Debra Lee Davis

Florida International University, 2000

Miami, Florida

Professor Nagarajan Prabakar, Major Professor

The use and dissemination of remotely-sensed data is an important resource that can be used for environmental, commercial and educational purposes. Because of this, the use and availability of remotely-sensed data has increased dramatically in recent years. This usefulness, however, is often overshadowed by the difficulty encountered with trying to deal with this type of data. The amount of data available is immense. Storing, searching and retrieving the data of interest is often difficult, time consuming and inefficient. This is particularly true when these types of data need to be rapidly and continually accessed via the Internet, or combined with other types of remotely-sensed data, such as combining Aerial Photography with US Census vector data. This thesis addresses some of these difficulties, a two-fold approach has been taken. First, a database schema which can store various types of remotely-sensed data in one database has been designed for use in a Semantic Object-Oriented Database System (Sem-ODB). This database schema includes in its design a linear addressing scheme for remotely-sensed objects which maps

an object's 2-dimentional (latitude/longitude) location information to a 1-dimensional integrated integer value. The advantages of using this Semantic schema with remotely-sensed data is discussed and the use of this addressing scheme to rapidly search for and retrieve point-based vector data is investigated. In conjunction with this, an algorithm for transforming a remotely-sensed range search into a number of linear segments of objects in the 1-dimensional array is investigated. The main issues and the combination of solutions involved are discussed.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

The demand for remotely-sensed data and the technology used to access this data is increasingly moving from solely the realm of the GIS expert to include the average computer user's desktop. The information that can be gleaned from spatial data is vast and the number of domains to which this information can be applied is increasing on a daily basis. Remotely-sensed data has traditionally been used for applications such as Cartography, Cadastral Mapping, hydrogeological surveys, natural resource exploration and other geological/environmental applications. In recent years, the availability and demand for remotely-sensed data has increased, and, consequently, the cost of acquiring it has decreased. This has made spatial data more affordable for a wider variety of applications. It has begun to be used for applications as diverse as real estate endeavors and sales, vacation planning, education and product marketing.

Spatial data provides a great deal of information for numerous applications. This wealth of information, however, comes at a cost. A large part of this cost is the complexity involved with dealing with this type of data. Remotely-sensed data is inherently very large and difficult to deal with. It is not unusual for one file to be larger than 20 Megabyes or for remotely-sensed data libraries to contain over 1 Terabyte of data. This data's use often requires powerful hardware and software systems such as ArcInfo, ENVI or ERDAS Imagine. These systems are rather expensive, difficult to use and require that the user have substantial understanding of the data prior to use.

Use of remotely-sensed data is further complicated by the various file formats that the data can come in. For example, there are numerous satellites (e.g., LANDSAT5, LANDSAT7, AVHRR, GOES-8, SPOT, SPIN2, etc.) which collect data while orbiting the earth, each of which provides data in a different format. As technology advances and more uses for remotely-sensed data are discovered, new ways of acquiring remotely-sensed data are also found. Satellites with more precise data collection instruments, such as the IKONOS satellite, have recently been launched and new types of data, such as AltM data, is being collected. With the proliferation of various sources of data, there is an increase in the number of different remotely-sensed data file formats, and thus, an increase in the complexity involved in dealing with this data.

Spatial data is not limited to imagery-type data such as aerial photography and satellite imagery. Vector data (e.g., point, line and polygon data), often stored in the form of textual data, is also increasingly being used and combined with other types of remotely-sensed data, such as integrating remotely-sensed raster data (e.g., aerial photography, satellite imagery, etc.) [MUFF87] with textual-based digital data sets. The scope and importance of the information that can be gleaned from this data is immense. This data, however, brings with it additional problems. For wide area networks, such as the Internet, remotely-sensed data transmission is based on the raster data model. With this type of data, file size tends to remain predictable and constant regardless of its complexity. For other types of georeferenced data, such as textual-based digital data sets, this is not the case [BUT99]. This is particularly problematic as this type of data not only

provides information on its own, it is also an invaluable tool for better understanding of raster data types.

Because of these issues, storage and retrieval of remotely-sensed data is often cumbersome at best. These complexities are further aggravated when more difficult data handling techniques are required such as when remotely-sensed data needs to be rapidly and continually accessed via the Internet, or combined with other types of remotely-sensed data. Nevertheless, the increased usefulness, along with an increase in the availability of remotely-sensed data, has greatly increased the demand for the ability to access the data quickly and easily.

TerraFly

To address some of these issues, the High Performance Database Research Center at Florida International University has been developing TerraFly, an interactive vehicle for 'flying' over and manipulating remotely-sensed data using any standard, Java-enabled Web browser. Through the use of a friendly graphical user interface, TerraFly allows even novice users to work with large amounts of spatial data without special hardware or software on their machines. All data is stored on the TerraFly server and retrieved for transmission over the Internet only as needed.

A screen shot depicting TerraFly's user interface can be seen in Figure 1. Some of the features currently available in TerraFly are:

- *Internet-enabled*: Users can access TerraFly through the use of any standard, Java-enabled Web browser such as Internet Explorer or Netscape.

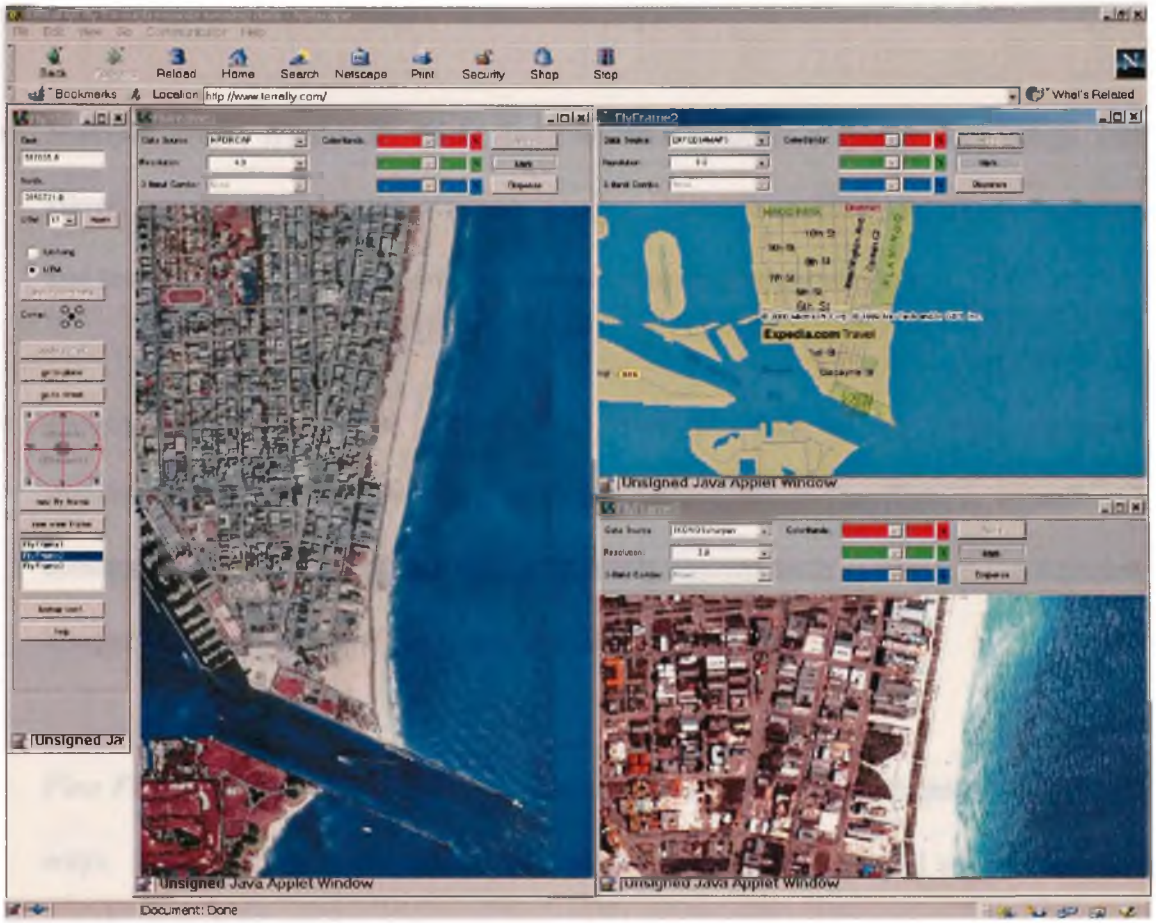

Figure 1. Web TerraFly User Interface

- *Multiple Data Types*: TerraFly is designed to support virtually any standard remotely-sensed data type. Multiple data types can be loaded simulaneously.

- *Multiple Resolution*: Users can view the available data at numerous resolutions, limited only by the resolution of the data itself (i.e., TerraFly does not interpolate images. The finest resolution available is determined by the data itself.).

- *Multiple Flight Windows*: Each flight window displays a spatial data image and allows users to fly over the available images. Users may open additional windows and load different data types and/or resolutions in each flight window.

- *Smooth, Continuous Flight*: Users 'fly' over the data in all flight windows simultaneously. All retrieved data has been preprocessed so that flight is smooth and continuous. The user need not deal with manipulations such as mosaicking adjacent images as this is automatically done for them by the system.

- *Fine Flight Direction and Compass Control*: Users can control flight in one of two ways. First, a compass control tool is provided on the Control Panel where users can use the mouse to click on the direction they desire to 'fly' on the compass image. This provides very fine direction control. Alternately, users may click on the image itself to determine flight direction.

- *Varied Flight Speed*: The user may vary the speed of the flight by either increasing or decreasing the speed using the Compass Control tool (which provides finer control), or by

positioning the cursor closer to the edge of the Main Flight Window to fly faster and positioning the cursor closer to the center of the window to fly slower.

- *Varied Refresh Rate*: Users may adjust the refresh rate of the images while flying using the Compass Control tool. For a smoother flight on fast Internet connections, user should increase the refresh rate. For slower Internet connections, users should decrease the refresh rate.

- *Informational and Drop-down textboxes:* These are textboxes and drop-down menus from which the user may select the desired information or data. These include:
    - Data set
    - Data resolution
    - RGB Intensity
    - Spectral Band Information
    - Image Coordinate Information (either Latitude/Longitude or UTM)

- *Go-To Coordinate Function*: This function allows the user to specify the coordinate (either latitude/longitude or UTM) to which he or she wishes to travel by entering the information in the text boxes on the Control Panel. This loads the desired location directly.

- *Go-To Place and Address Functions*: These functions allow the user to retrieve and select a place of interest, or enter a street address, and have the location of interest loaded in all flight windows simultaneously.

- *Look-up Feature*: Using TerraFly's Ctrl+click option, users can retrieve information regarding any point on the screen. If using during flight, the point's coordinates are displayed. If used while not moving, information regarding the closest populated place and point of interest is displayed in addition to the point's coordinates.
    - *Look-up Feature Configuration*: Users may select the specific type of place information retrieved for any given point.

- *Sensor Band Controls*: These controls allow the user to manipulate the sensor band combinations of spectral data (i.e., Landsat TM, Hyperspectral, etc.) to view false color images. This provides greater flexibility and availability of information. For example, with the Landsat data, users are able to select from a list of seven possible sensors for each color band. TerraFly provides two ways of doing this:
    - *Pre-defined Three-Band Combinations*: This is a drop-down menu that provides commonly used, predefined sensor band combinations.
    - *Advanced Three-Band Combinations*: This is a series of three drop down menus (*Red*, *Green* and *Blue*) in which the more scientific user may choose any desired three-band combination they wish to study or analyze.

- *RGB Intensity Control*: This control allows the user to increase or decrease the intensity of each flight window's color intensity.

- *Data Dispensing Capability*: TerraFly allows users to graphically select an area of interest and retrieve the desired type of data for that area in the user's preferred file format and resolution.

- *Animated Help Tutorial*: This gives the user detailed, step-by-step, animated instructions on how to use the features of the TerraFly system.

TerraFly has been designed as a thin client, with most of the work taking place on the server side of the system. This allows for faster processing time and the use of fewer resources on the client side, which has limited resources due to browser and related limitations. To further enhance performance, both static and dynamic processes are used by TerraFly to process and display the data [DAVI99]. During the static display process, data for each band is retrieved on the server side, decompressed and used to create a false color composite image. The data is then sent to the client to be displayed on the screen. Once on the client side, the dynamic process builds on top of the static process by use of a double buffering system. The images that are currently being displayed on the screen are in the main buffer. A secondary buffer contains additional images that correspond to areas that are contiguous to the screen image. This double buffering system provides TerraFly with the ability to pre-fetch images, thus guaranteeing faster processing and a smooth flight.

CURRENT WORK

In order for TerraFly to fly smoothly and continuously over remotely-sensed data, the system must be able to quickly retrieve, process and display the data. At the same time, the security and integrity of the data must be maintained. As was stated previously, because spatial data is inherently large, storage, retrieval and real-time processing of these large data sets can be slow and cumbersome. This thesis addresses this issue. Specifically, this thesis discusses a database schema which can effectively store and retrieve diverse types of remotely-sensed data in one database. It has been designed for use in a Semantic Object-Oriented Database System (Sem-ODB). This database schema includes in its design a linear addressing scheme for remotely-sensed objects which maps an object's 2-dimentional (latitude/longitude) location information to a 1-dimensional integrated integer value. The advantages of using this Semantic schema with remotely-sensed data is discussed and the use of this addressing scheme to rapidly search for and retrieve point-based vector data is investigated. In conjunction with this, an algorithm for transforming a remotely-sensed range search into a number of linear segments of objects in the 1-dimensional array is investigated. The main issues and the combination of solutions involved are discussed.

The remainder of this thesis is organized as follows. Chapter 2 provides basic information regarding the different types of data involved in this work. Chapter 3 provides a general survey of database storage technology, including Relation, Object-oriented and Semantic database technologies. Chapter 4 discusses the overall Semantic database schema in this thesis. Chapters 5 discusses spatial data structures relevant to the

spatial addressing scheme investigated in this thesis and chapter 6 provides detail and algorithms for the spatial addressing scheme itself. Chapters 7 and 8 present conclusions and suggestions for future work, respectively.

II. REMOTELY SENSED DATA

There is a vast number of different types of remotely-sensed data. Each of these different types can be used by scientists and engineers to better understand what is occurring on the Earth's surface, as well as by other professionals for anything from planning to information gathering to the marketing of commercial products. This chapter describes the main types of data currently used in the TerraFly system. It is not an exhaustive list of the data types that the TerraFly system supports, or of the types of spatial data that are available, but rather the most commonly used data by TerraFly's users.

AERIAL PHOTOGRAPHY

Aerial photography is typically high resolution imagery that comes in black and white, natural color or color-infrared. With resolutions as fine as 3 inches per pixel, its potential applications are endless. Aerial photographs are primarily obtained by aircraft equipped with instruments that can record the visible and invisible portions of the electromagnetic spectrum. The portion of the spectrum that is visible to the human eye is the colors that we see every day, such as when we look at a rainbow. Other parts of the spectrum which are not visible to the human eye, such as near-infrared wavelengths, can provide important information. Near-infrared and visible wavelengths can be simultaneously recorded using standard color and color-infrared films, and later combined to provide unique and valuable views of the earth and its features. So, aerial

photographs are recorded on film, and can then be scanned to create image files or printed onto photographic paper.

Color infrared aerial photography is particularly useful to scientists. Healthy green vegetation reflects high levels of near-infrared wavelengths. Thus, when images are processed as color-infrared composites, anything that is green visibly looks red [USINT].

Aerial photography can be obtained from numerous sources and the number of square miles covered by one aerial photograph depends upon the data source. There are many private corporations which can be hired to provide custom services and fly over specific areas of interest. The largest source of aerial photography, particularly color-infrared photography, is from the National Aerial Photography Program (NAPP) by the USGS. They fly over most areas of the U.S. on regular intervals, typically every five years.

MULTISPECTRAL SATELLITE DATA

When most people think of imagery of the earth, satellite imagery is what comes to mind. Satellite data, however, is not merely a visual picture of the earth. Satellite data typically contains much more information than what can be found in the visible field, providing invaluable information that would otherwise be difficult or impossible to collect. This gathering of 'more than the eye can see' is possible through the use of

spectral bands. When these satellites are launched, they have on board multispectral sensors which record information at different wavelength ranges for different spectral bands. These ranges often differ for different instruments. Once downlinked and processed, information from these bands can be combined to provide not only a natural looking image of the earth, but also information such as vegetation cover, ground temperature readings, urban development, coastal characteristics and other geographical information. Two commonly used types of multispectral satellite data come from the Landsat and IKONOS satellites. Brief descriptions of each are found below.

LANDSAT

The first Landsat satellite was launched in 1972 to provide global coverage of the Earth. Since then, other Landsat satellites have been launched, with the most recent being Landsat 7, launched in 1999. The Landsat satellites carry electronic sensors which record the visible and near-infrared light levels of Earth's reflected energy. This data is transmitted to back to Earth in 4 to 7 spectral wavelength bands [USINT]. Landsats 1-5 each carried a multispectal scanner (MSS) which simultaneously collected data for four bands at an overall resolution of approximately 80 meters. Landsats 4 and 5 carry the thematic mapper (TM) sensor and Landsat 7 carries the enhanced TM (ETM+) sensor (at this time, only Landsat 5 and 7 are still in orbit collecting data). The TM sensor collects data for seven bands and the ETM+ collects data for eight bands. They both have an overall resolution of approximately 30 meters. Having a larger number of bands yields more detailed spectral information [AERIAL].

BAND NUMBER	*Spectral Range(microns)*	*Ground Resolution(m)*
1	.45 to .515	30
2	.525 to .605	30
3	.63 to .690	30
4	.75 to .90	30
5	1.55 to 1.75	30
6	10.40 to 12.5	60
7	2.09 to 2.35	30
Pan	.52 to .90	15

Table 1. Landsat 7 and ETM+ Characteristics

Because of the number of spectral bands and the resolution of the TM sensor, the color composite images created when the various spectral bands are combined provide a wealth of detailed information. Landsat 7, for example, has a total of eight bands. Three bands are visible light, three bands are infrared, one band is panchromatic and one band is thermal infrared. Different combinations of these bands can provide information such as discriminating vegetative, crop and timber types, monitoring urban growth, investigating volcanic surface deposits, estimating snow melt runoff, tracking beach erosion, and assessing grass and forest fires. Information on the specific wavelengths and bands for Landsat 7 can be found in Table 1 [LAND7].

BAND NUMBER	Spectral Range (microns)
1	.45 - .52 (blue)
2	.53 - .61 (green)
3	.64 - .72 (red)
4	.77 - .88 (NIR)
Panchromatic	.45 - .90

Table 2. IKONOS Data Characteristics

IKONOS

The IKONOS satellite is the first satellite launched which provides a commercial source for one meter resolution satellite data. Launched in the Fall of 1999, IKONOS provides data with spatial resolutions of one meter panchromatic (black and white), four meter multispectral and one meter pan-sharpened (panchromatic data colorized with the four meter data for a natural view). Although IKONOS data does not provide as many bands as Landsat imagery, it provides much higher resolution and, thus, greater detail of the area of interest. With a revisit time of five days (i.e., IKONOS can provide new imagery for any area of the world every five days), IKONOS is particularly useful for local governments, utilities and the telecommunications industry. Information on the specific wavelengths and bands for Landsat 7 can be found in Table 2 [IKONOS].

DIGITAL VECTOR DATA SETS

There are many types of georeferenced digital data (e.g. point, line, polygon and textual data) available on the market. Two of the most popular are GNIS (Geographic Names Information System) [GNIS] and US Census Tiger/Line files [TIGER]. GNIS data primarily consists of names and types of places along with associated coordinate point information. This data is compiled by the United States Geological Survey in conjunction with the U.S. Board on Geographic Names (BGN). This data contains information about nearly 2 million physical and cultural geographic features in the United States, its territories as well as Antarctica. Each feature described in the database is identified by its Federally recognized name, and a feature's location is referenced by state, county and geographic coordinates. The GNIS data currently used in this thesis includes data for the entire US and its territories.

US Census Tiger/Line data consists of point, line and polygon data which provides information such as feature types, address ranges and ZIP Codes, codes for legal and statistical entities, landmark point features, area landmarks, latitude/longitude coordinates of linear and point features, key geographic features and area boundaries. These files are not comprised of graphical data or images similar to raster-type data, but rather digital data sets describing identified geographic features. The Tiger/Line data currently used in this thesis includes information regarding US highways, major roads, streets/addresses and populated-area/county-subdivision shape coordinate points data. The US highways, major roads and street/address data is line data, with each line segment

consisting of a series of ordered points with separate beginning and ending points. The populated-area/county-subdivision data is polygon data consisting of a series of ordered points that form a polygon shape.

III. SPATIAL DATA STRUCTURES

Managing spatial data has been a topic of research for a number of years, and the topic of this thesis. Because relational database systems are notoriously inefficient at handling spatial data, a great deal of research has gone into creating efficient spatial data structures. In relational database systems, the data structures used for indexing favors searches based on comparisons. Most queries to spatial data, however, are proximity based (see Chapter 4 for a more detailed discussion). With spatial data, a user is much more likely to be interested in what is near a particular place of interest. For example, a real estate agent might be interested in finding the nearest shopping mall to a particular house that they are trying to sell.

According to Knuth [KNUTH73], there are three main types of spatial data queries:

- *Point query* – determines whether a given data point is in the database and retrieves the record
- *Range query* – retrieves a set of data whose keys have specific values or a range of values
- *Boolean query* – combination queries of the above two types of queries along with Boolean operators such as AND, NOT, OR, etc.

A common type of query which falls under the range query category is the nearest neighbor query [ROUS95]. In this type of query, the n^{th} nearest neighbors of any given

point are retrieved. In the example above, a real estate agent could query the database for all of the shopping malls within a five mile radius of a particular home.

The data structures specifically designed for spatial data can effectively deal with searches comparable to this. These data structures are typically proximity-based, storing objects near each other. There are numerous spatial data structures that can be discussed, and a number of surveys have been written discussing the major structures and their applications [SAMET90a][SAMET90b][GUT94]. In this thesis, the most relevant of these data structures and their variants will be discussed, namely, quad-trees [SAMET90a][SAMET90b], k-d trees [BENT75][SAMET90a] and R-trees [GUTT84] [CHEN00].

QUAD-TREES AND K-D TREES

Quad-trees are hierarchical data structures which are based on the principle of a recursive decomposition of space. They are commonly used to store data such as points, areas, curves, surfaces and volumes. In the tree representation, the root node represents an entire array. Each node of the tree, except for the leaf nodes, have up to four child nodes. These child nodes represent the four quadrants of the parent node, namely the northwest, northeast, southwest and southeast quadrants. The leaf nodes represent the quadrants for which further subdivision is not needed. The resulting rectangles in quad-trees are non-overlapping.

There are many different variants of quad-trees. The most common are point quad-trees and region quad-trees. Closely related to these are k-d trees. Brief descriptions of each of these variants can be found below.

POINT QUAD-TREES

The point quad-tree was formally presented in 1974 by Finkel and Bentley [FINK74]. This tree structure contains non-uniform sized blocks, each of which contains one data element. In two-dimensional space where each data element is unique, each node (and thus each data element) is represented as a record of type node with seven attributes. The first four attributes are pointers to the four child nodes, each corresponding to the different quadrants (NW, NE, SW and SE). The next two attributes contain the X and Y location coordinates (e.g., latitude and longitude) of the specific data element. The last attribute contains the name or description of the data the node represents (e.g., U.S. Capital Building).

Record insertion into a point quad tree is similar to that of a binary search tree. A given node X is used to divide an area into quads by drawing two lines, one horizontal and one vertical, through the element's point coordinates. This creates the four quadrants, each of which correspond to the child nodes. An example of a point quad-tree is illustrated in Figure 3.

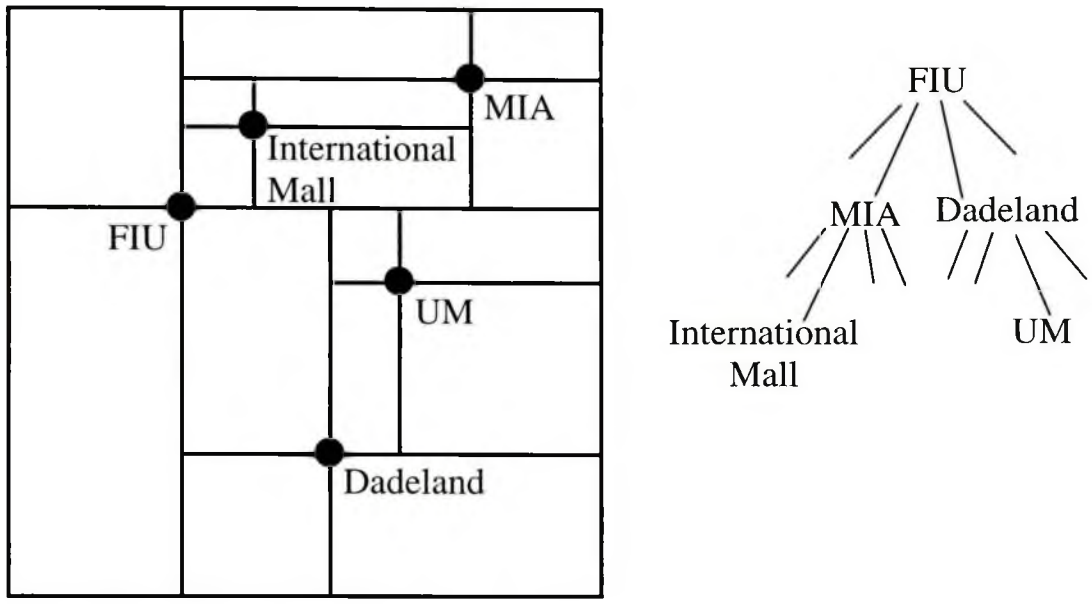

Figure 2. A Point Quad-Tree and its Visualization

Point quad-trees are well suited for applications which require proximity searches, as with the above real estate example. Its efficiency lies in its ability to reduce the amount of searching required. It does not search through many records that do not need to be examined. Further, data points can be searched for in any connected figure. For example, there are a number of algorithms available for searching within an arbitrarily sized rectangular window [FINK74]. The cost of searching through has been studied by a number of researchers [LEE77]. They have shown that, in the worst case, range searching a compete two-dimensional point quad-tree takes $O(2 * N^{1/2})$ time.

Point quad-trees do have several disadvantages:

- all *k* keys for a *k*-dimensional quad-tree must be tested for each node

- leaf nodes are costly in terms of space due to the four null pointers
- nodes for a *k*-dimensional tree are large because it takes $k + 2^k + 1$ words for each

Some of these deficiencies are alleviated and improved upon by the k-d tree.

K-D TREES

The *k* in k-d tree signifies the representative state's dimensionality. A k-d tree is basically a binary search tree where the direction a branch is to be created is determined by testing a different key at each level. This tree structure contains non-uniform sized blocks, and each data element is represented by one node. In two-dimensional space, x-coordinate values are compared at the root and at even depths. Y-coordinate values are compared at odd depths. Each node (and thus each data element) is represented as a record of type node with six attributes. The first two attributes are pointers to the two child nodes, each corresponding to the different directions, left and right. The next two attributes contain the X and Y location coordinates (e.g., latitude and longitude) of the specific data element. The next attribute contains the name or description of the data the node represents (e.g., U.S. Capital Building). The last attribute indicates the name of the coordinate to be tested via its level.

Record insertion into a k-d tree is analogous to that of a binary search tree. Records are searched for based on x and y coordinate values. X coordinate values are compared at even depths and y coordinate values are compared at odd depths. If a node's level is even, then nodes whose x values are less than the current node's x value will be

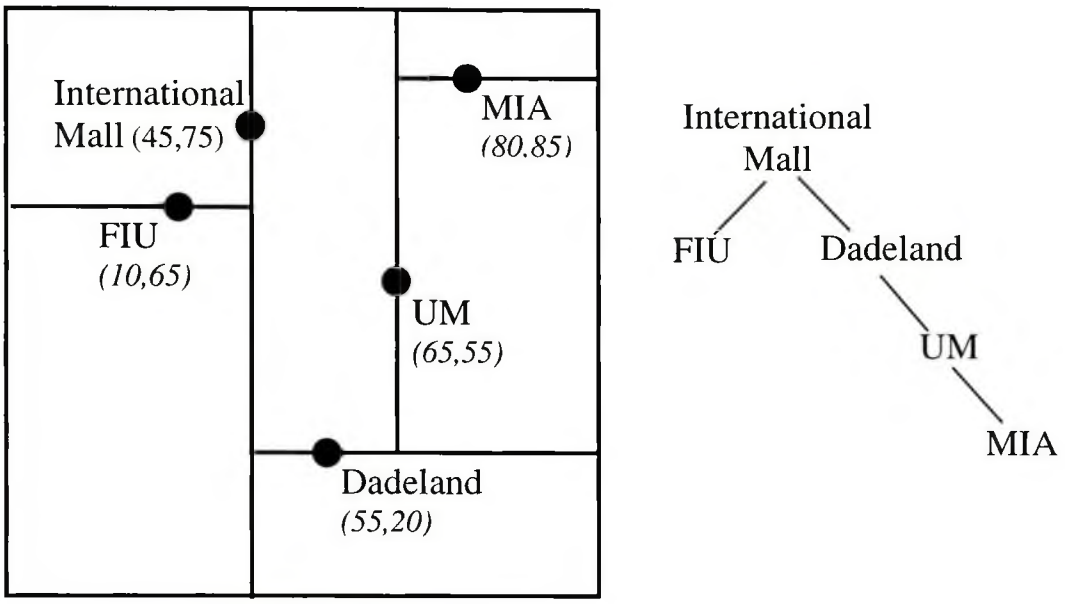

Figure 3. A K-D Tree and Its Visualization

placed in the left subtree (e.g., node.x > leftchild.x). Even level nodes whose x values are greater than the current node's x value are place in the right subtree (e.g., node.x < rightchild.x). If a node's level is odd, then nodes whose y values are less than the current node's y value will be placed in the left subtree (e.g., node.y > leftchild.y). Odd level nodes whose y values are greater than the current node's y value are place in the right subtree (e.g., node.y < rightchild.y). Once the bottom of the tree is reached, the node is inserted. An example of a point quad-tree is illustrated in Figure 4.

K-d trees are also well suited for applications which require proximity searches, as with the previous real estate example. As with point quad-trees, its efficiency lies in its ability to reduce the amount of searching required. It does not search through many records that do not need to be examined. The cost of searching depends upon the type of

query. In the worst case, given N points, the cost of a range query for a complete k-d tree is $O(k * n^{1-1/k})$ [LEE80].

The k-d tree alleviates the many of the problems with point quad-trees noted earlier. Point quad-trees, however do have one main advantage over k-d trees. Point quad trees are inherently a parallel data structure, whereas k-d trees can be thought of more as serial data structures. In a point quad-tree, parallel key comparison operations can be performed for the *k* key values. This can not be done with the k-d tree.

REGION QUAD-TREES

For point quad-trees and k-d trees, space is decomposed based on the points themselves. This leads to regions of unequal size. Region quad-trees add the requirement that subdivisions should be of equal size. There are two main types of region quad-trees, namely MX and PR quad-trees.

MX QUAD-TREES

An MX quad-tree is very similar to a point quad-tree. The main difference is that the tree structure contains uniform sized blocks, and the root node represents the entire area of interest. The node structure of MS quad-trees is the same as that of point quad-trees, containing the same seven attributes.

To create the tree, the regions in the tree are split evenly, producing four quadrants (e.g., the child nodes NW, NE, SE, SW). Splitting of each subsequent node

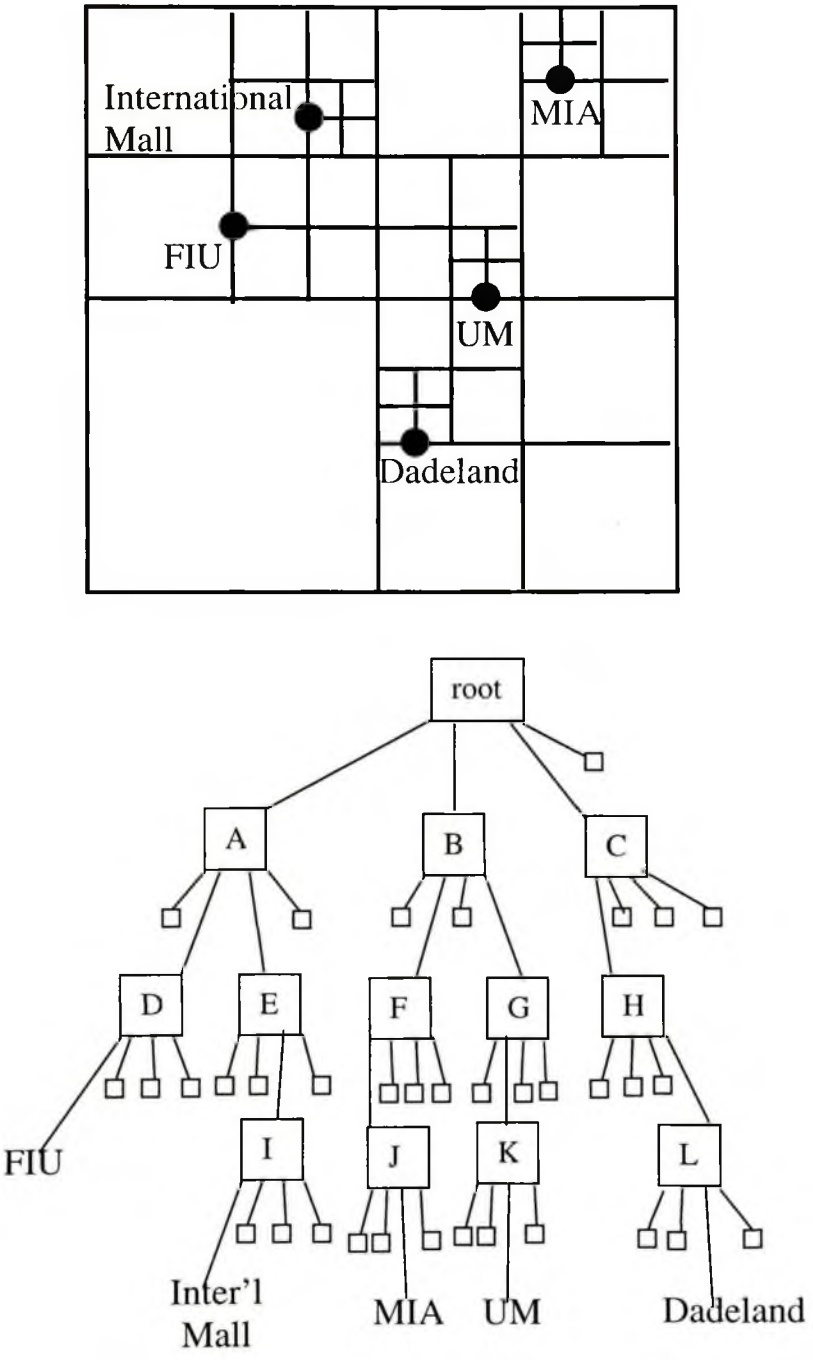

Figure 4. A MX Quad-Tree and Its Visualization

continues until each data point in the quad-tree corresponds to a 1 x 1 square. Data points are inserted into the tree by searching for them and then coloring the corresponding leaf node black. Data points are always contained in the leaf nodes. An example of an MX quad-tree is illustrated in Figure 5.

Unlike point quad-trees and k-d trees, the shape of the MX quad-tree is independent of the order that data is inserted into the tree. Range searches are performed in MX quad-trees as they are in point quad-trees. The cost of searching for all points in a rectangle whose sides are parallel to the quadrant lines is, in the worst case, $O(f + 2^n)$ where f is the number of points found and n is the maximum depth.

As long as the domain of the points finite and discrete, the MX quad-tree is feasible. Otherwise, it is not. Therefore, an alternative has been created, the PR quad-tree.

PR QUAD-TREES

PR quad-trees associate point data which does not need to be discrete to quadrants. The tree structure contains uniform sized blocks, and the root node represents the entire area of interest. Each leaf node is either empty or contains one data point. The node structure of PR quad-trees contains eight attributes. The first four attributes are pointers to the four child nodes, each corresponding to the different quadrants (NW, NE, SW and SE). The fifth attribute indicates whether the node contains a data point, or if it's a non-leaf node. The next attribute contains the name or description of the data the node

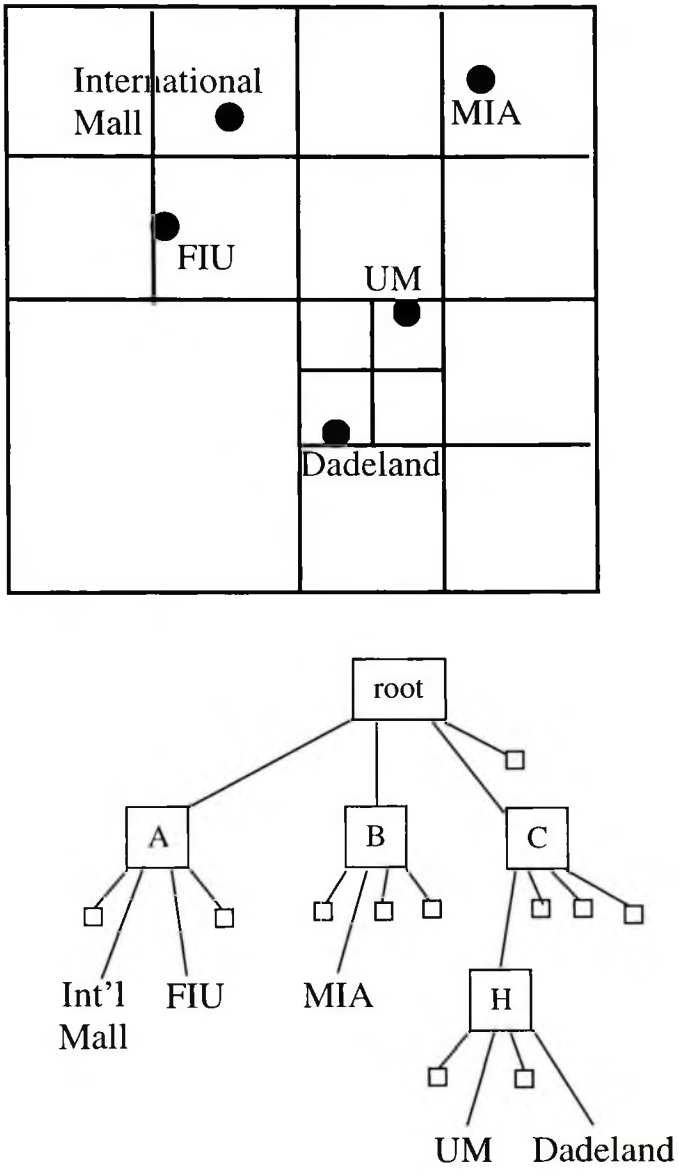

Figure 5. A PR Quad-Tree and Its Visualization

represents (e.g., U.S. Capital Building). The last two attributes contain the X and Y location coordinates (e.g., latitude and longitude) of the specific data element.

Tree creation is similar to the MX tree. The difference is that, although one point corresponds to one quadrant, the point need not lie on a quadrant boundary. Data points are inserted into the tree in a manner analogous to point quad-tree. Data points are always in the leaves. An example of a PR quad-tree is illustrated in Figure 6.

Similar to the MX quad-tree, the shape of the tree is independent of the order that the data is inserted. However, the order does matter for the shape of the intermediate trees. Range searches are performed in PR quad-tree as they are in MX and point quad-trees. The cost of searching for all points in a rectangle whose nodes are parallel to the quadrant lines is, in the worst case, , $O(f + 2^n)$ where f is the number of points found and n is the maximum depth.

R-TREES

R-trees are hierarchical data structures which are based on B+-trees. They are commonly used to store d-dimensional objects through the use of minimum bounding d-dimensional boxes. In the tree representation, each node represents the rectangular area which encompasses its child nodes. The leaf nodes represent the objects themselves (i.e., if contained in a database, the leaf nodes contain pointers to the objects). The resulting rectangles in R-trees may be overlapping.

R-trees must follow a number of rules:

- all leaves reside on the same level

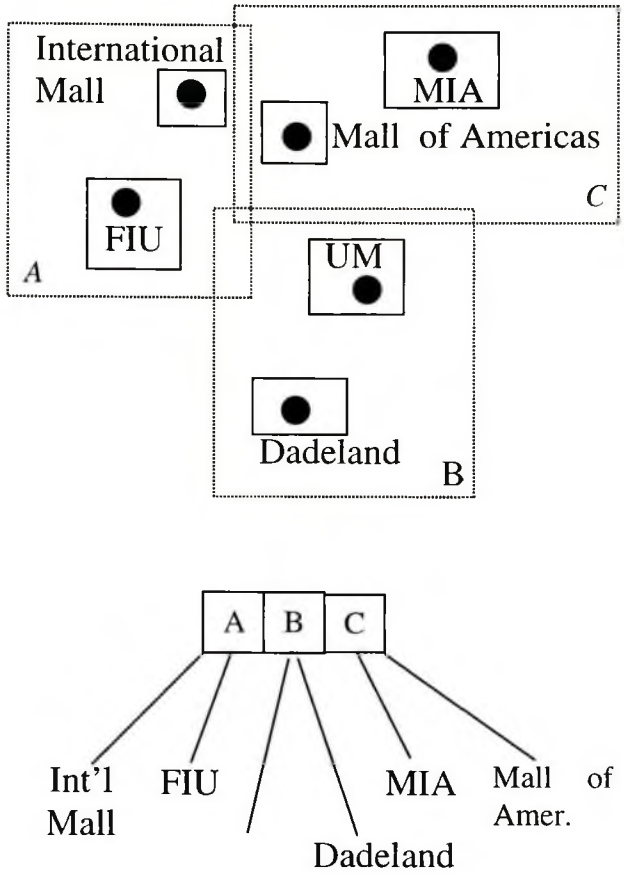

Figure 6. An R-Tree and Its Visualization

- every leaf pair is of the form (MR, O) such that MR is the minimal rectangle containing the object O
- other nodes are of the form (MR,P) such that MR is the minimal rectangle containing the rectangles of its child nodes and P is a pointer to a child node
- an R-tree is classified as (m,M) where, with the exception of the root node which has at least two pairs if it is not a leaf, all nodes contain between m and M pairs with $m \leq \lceil M/2 \rceil$.

R-tree construction is dynamic, and inserts and deletions can take place in-between searches. An example of an R-tree is illustrated in Figure 7.

IV. DATABASES

WHY USE A DATABASE?

It has been estimated that over sixty percent of the cost of installing a GIS system constitutes cost related to the development of an appropriate database. These costs increase over the life of the system as the system needs to be maintained. So why use a database at all? Why not just keep the data in a file system or in a data structure in main memory?

Using a file system to store data is a simple, easy to understand and accomplish solution. File systems, however, do suffer from a number of drawbacks. These drawbacks include:

- Data Redundancy: The same data might be stored in different locations.

- Poor Data Control: Redundant data is not always appropriately updated, so all the data may not be up to date at the different locations.

- Inability to Easily Manipulate Data: Much of the data modification must be done manually (i.e., there is no standard way of querying all types of data files). This is a tedious and error prone activity.

- Cryptic Work Flows: Accessing the data can take excessive programming effort and is too difficult for real-users (as opposed to programmers) [Sol98].

- Lack of Security: File systems do not provide sufficient security for sensitive or important applications.

Although these limitations of file systems are substantial for GIS applications, one paramount reason the use of a database is needed for large-scale GIS applications involves the amount of data that needs to be stored. As was stated previously, remotely-sensed data in inherently very large. A file system simply would not be able to handle searching and retrieving spatial data in an efficiently, particularly when a great deal of data needs to be stored.

Database systems are designed to handle and efficiently organize large amounts of data. A database is a computerized record keeping system, yet it is more than that. A database is a system which effectively stores the data of interest, provides a standardized method for searching, retrieving and updating the data, and allows storage and retrieval of the data easily and efficiently regardless of the amount of data being manipulated [SOL98]. In short, some of the requirements of a database for a large GIS system, such as TerraFly, are:

- Support multiple types of data – As was stated previously, there are many different types of spatial data available in the market today. A database for a GIS system should be able to store various types of data in the same database

- Provide Scalability – As more data is acquired, the size of the database will increase. It is also likely that the number of users will grow. This will require the system to be able to efficiently distribute data over a growing number of servers while maintaining efficient access to the data.

- Provide Good Performance – The database must be able to provide data fast enough to allow for real-time, interactive visualization of the data.

- Provide Data Security – The data must be secure against simultaneous updates or data corruption due to human error.

RELATIONAL DATABASE SYSTEMS

The relational database was first formally introduced in 1970 by Edgar F. Codd [CODD70]. Since then, the use of relational databases has grown and they are now the most widely used type of database system. Relational databases are table-based database systems where records that relate to each other are stored in a table, similar to a spreadsheet. . For example, records for a company's employees would be stored in one table, the employee table.

Each table consists of a number of records where the field names in the table are the same, but the field values differ. For example, in the employee table mentioned above, field names may include employee name, position and salary, whereas the field values would be each employee's actual name (e.g., Bill Press, Joann Smith, etc.), position (e.g., Vice-President, CEO, etc.) and salary (500K, 750K, etc.). Every table has a unique identifier called a primary key. This is a field or combination of fields that provide a means to individually distinguish each record. Thus, no two records may have the same value in their primary key.

Most applications require more than one table. In relation databases, relationships between tables can be defined. These relationships connect one table to another based on fields common to both tables. Relational databases work well for certain applications, typically any type of data which works well as tabular data. This is often, but not always, the case with textual-based data such as financial data. Information on relational databases and its advantages is widely available [cite]. Some of the most popular commercial relation databases are Microsoft Access, Oracle, Sybase and IBM's DB2.

Relational databases, however, have been found to be lacking when dealing with other types of more complicated data [RISHE94]. This is particularly true when one is dealing with data such as spatial data. Remotely-sensed data tends to contain complex data structures such as line and polygon data, arbitrary data types, large unstructured and irregular data such as surfaces (e.g., aerial photography, satellite imagery, etc), and null

values when specific data is missing. This complex type of data is best dealt with using hierarchical modeling [SAMET90]. The table-based approach of relation databases does not have the mechanisms needed to deal with this well. Most relational databases do have the capacity to store binary objects such as an aerial photograph, but many must store different kinds of data in different physical databases [WAUGH].

The deficiencies of relational database systems when dealing with remotely-sensed data have been considered in detail in various publications [KEAT87], [LORIE84], [EGEN94]. Due to these deficiencies, many researchers have been looking for alternatives which are better designed to deal with spatial data. This is one of the motivations for the development of object-oriented database systems. These systems are designed to continue to provide most of the advantages of relational database systems while offering better handling of hierarchical data models and data similar to remotely-sensed data.

OBJECT-ORIENTED DATABASE SYSTEMS

Research into object-oriented databases began in the late 1970's [LOCK79] and by the early 1980's, was well established as a significant area of research. Object-oriented database technology can be described as a system in which object-oriented programming technologies are combined with traditional database technologies to support functionality such as query, transaction and security along with the ability to

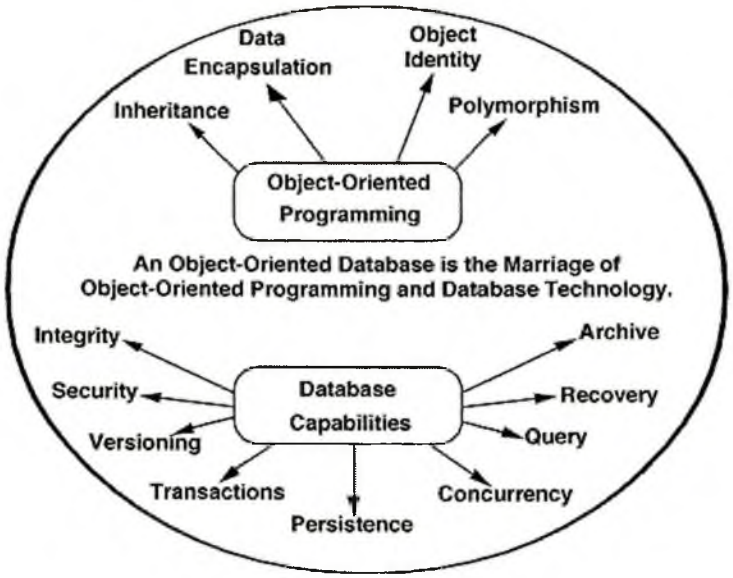

Figure 7. Structure of an Object-Oriented Database System

persistently handle objects and object identifiers. Figure 2 [MCFAR99] provides an illustration demonstrating object-oriented databases.

There are a number of differences between relational and object-oriented database systems. As quoted from Mnushkin [MNUSH], the main differences are:

- "A RDB stores simple, fixed length data in tables. If your data fits naturally in tables, this will work fine.

- An OODB supports arbitrary structures, nested structures, dynamically varying structures, arbitrary many-to-many relationships and most others you can think of.

- An RDB supports simple operations such as select, project, and join over localized amounts of data through the use of SQL.

- An OODB allows arbitrary operations, defined by users, with arbitrary complexity. These operations might traverse inter-object relationships, affect many objects in different databases, or do any number of user-defined tasks.

One of the most notable characteristics of object-oriented database technology is its combination of database technology with object-oriented programming to furnish an integrated system for application development. McFarland et. al. [MCFAR99] notes that including the operational definitions with definition of the data has a number of advantages. "First, the defined operations apply ubiquitously and are not dependent on the particular database application running at the moment. Second, the data types can be extended to support complex data such as multi-media by defining new object classes that have operations to support the new kinds of information."

There are other well known strengths of object-oriented modeling which, in combination, provide productivity advantages to developers. Inheritance, encapsulation, object identity, polymorphism and dynamic binding are a few examples. Inheritance permits the incremental development of solutions to complex problems by defining new objects as subcategories of existing objects. With encapsulation, an object contains both

behavior and data specifications. Unlike most relational databases, which are value-based, most object-oriented databases are identity-based where each data representation has its own identifiers. Polymorphism and dynamic binding allow for the composition of objects that provide solutions without having to write code that is object specific. That is, these provide the ability to define operations for one object and then share its specifications with others. These objects can then, in turn, extend these operations to provide unique behaviors to those objects [MNUSH][MCFAR99].

Object-oriented databases represent relationships between objects explicitly. This is a significant difference from relational databases and allows both navigational and associative access to the data. This is particularly beneficial if the relationships between objects are rather complex. The more complex the relationships, the greater the advantages of representing the relations explicitly. This explicit representation also provides improved performance over relational value-based relationships when accessing data [MCFAR99].

With these advantages, it is clear that object-oriented databases can be of great use to a number of applications that relational databases traditionally do not manage well. This is particularly true when the problem scope contains [RISHE98B][MCFAR99] [MNUSH]:

- many different data types
- numerous relationships between the objects
- objects with complex behaviors

Although the use object-oriented database systems provides improvement in the management and manipulation of remotely-sensed data, it still has its drawbacks. For example, most object-oriented database systems do not support standard SQL. Because of these deficiencies, Semantic database systems have come into existence.

SEMANTIC OBJECT-ORIENTED DATABASE SYSTEM (SEM-ODB)

Due to the inherently large size of spatial data, the storage and retrieval method used in this thesis is central to its success. The Semantic Object-oriented Database (Sem-ODB) technology under development at HPDRC has been designed to be efficient at dealing with spatial data and related products. It is a general-purpose database management system (DBMS) that supports a wide spectrum of applications ranging from transaction-oriented to decision-support systems. A Multi-user Semantic Database Engine is currently operational and a main goal at HPDRC has been to achieve the quality that would make the Sem-ODB server viable as a commercial product.

Sem-ODB is designed to store varied types of data in an efficient and logical manner, and it easily deals with non-conventional data such as spatial data. It has a number of key advantages over current database technology. A few of Sem-ODB's advantages are:

- Sem-ODB gives the user control via an intuitive structure of information.

- The end-user is empowered to pose complex ad hoc queries.

- A conceptual data model of the enterprise is directly supported.

- Queries are made simple and very short. Queries can be up to ten times shorter (and so easier to pose) than in relational databases. For example, the user need not bother about "joins" (cross-references between relational tables), many-to-many relations and inheritance.

- User programs for a semantic view are substantially shorter than for a relational view, achieving major improvements in the application software development cycle, maintenance and reliability.

- SQL, the standard relational database language, has been adapted to work with semantic databases. This runs counter to most object-oriented database systems. Further, programs in SQL for Sem-ODB tend to be an order of magnitude simpler and shorter than for relational database systems.

- Sem-ODB's ODBC driver allows SQL querying of a semantic database and interoperability with relational database tools, e.g. end-user systems like MS Access Query-By-Example or Crystal Reports. In these tools the number of user keystrokes

required is proportional to the size of the generated SQL program. So again, savings are realized and simplicity is attained through the use of the semantic view.

- Varied types of data can be stored in one database. Most DBMS are unable to do this and must store different types of data in separate databases. This is inefficient and can make them difficult to use.

- Data types are unlimited — strings can be of any length and techniques have been developed to represent numbers of unlimited length and precision.

- Algorithms have been developed to provide very efficient full indexing, allowing fast access to every single fact in the database. Further, a proprietary algorithm guarantees optimality of the basic queries defined in our Semantic Algebra; this includes optimality of range queries.

- Objects can belong to several different categories at the same time. The operation to categorize/de-categorize objects can be performed efficiently and on-line.

- There is no need for NULL attributes. Sparse tables in relational databases may waste space and processing time.

- There is no need for tables and indices. This reduces the space allocation required, an aspect of particular importance when dealing with spatial data.

- No keys are needed. Referential integrity constraints are supported automatically by the semantic database.

Sem-ODB can easily handle Terabytes of data. To further improve performance and flexibility, Sem-ODB can be used as a distributed database. Data for TerraFly can be stored in a distributed Sem-ODB database. In this way, data could be stored on multiple servers and in multiple locations, and retrieved simultaneously from the various locations, greatly expanding application capabilities. This organization, of course, would be invisible to the user; access to the various locations would be automatic.

Thus, Sem-ODB can be said to provide the best of both worlds. It has many of the advantages of both relational and object-oriented databases, and is specifically designed to work well with spatial data. The use of Sem-ODB as the database for this project is one component responsible for the efficiency of the storage and retrieval of the data.

V. SEMANTIC DATABASE SCHEMA

Because of the inherently large size of the data and the complexity of the project, the database used in this thesis is the Sem-ODB system. Sem-ODB is designed to efficiently store and handle spatial and related data in one database. This chapter will discuss the semantic analysis of the application and its data, as well as the semantic database schema design based on this analysis.

SEMANTIC ANALYSIS OF THE APPLICATION

The database design for this thesis is intended for use with the TerraFly system. At the same time, the design should be reasonable for use with other systems. Thus, the data and information on its metadata must be preserved while, at the same time, additional information needed by TerraFly, such as information associated with preprocessing, is preserved.

Recall that TerraFly is an interactive vehicle for flying over and manipulating remotely-sensed and related data via any standard Web-browser. TerraFly's database must be designed in such a way that the storage and retrieval of the data is fast, efficient, logical and secure. At the same time, it must be flexible enough to deal with the demands of other potential applications, as well as TerraFly. In this section, the types of data used by the TerraFly system will be discussed and analyzed for the creation of TerraFly's integrated database schema.

DATA STORAGE REQUIREMENTS

Because we need to store many different data sets, their metadata and preprocessing information, there are a number of storage requirements that must be met. The main requirements are:

- *Storage of varied types of data.* Although the use of Sem-ODB provides this ability, the database schema must still be structured in such a way that different types of spatial data can be stored logically and efficiently in the database.

- *Storage of data in a uniform format.* When dealing with the different types of spatial data, data needs to be stored in a uniform manner for easier and more efficient retrieval. We would not, for example want to have a different database structure for every different type of spatial data. This would prove inefficient and waste resources as every time a new type of data is encountered, a new database structure (and related tools) would need to be created. It would be prudent to minimize the new coding needed when a new data type is used in the system.

- *Storage of the same type and area of data with varied acquisition dates.* The database schema must be able to efficiently store historical data. Historical data is an extremely useful and important tool for many users. A true GIS database would need to have the ability to support this type of information.

- *Provide structures for fast and efficient data retrieval.* The schema should be optimized for faster data retrieval when possible, particularly when dealing with the vector data. The logic, however, should still be maintained.

- *Provide a secure and logical database design.* The database design must be easy to understand and appropriately represent the data being stored. The design should include constraints which help control the data and reduce the possibility of data corruption.

- *Storage of metadata and preprocessing information.* Storage of the metadata is particularly important for understanding the data in the database. In TerraFly's previous databases, this information was not stored in the database and was, instead, hard coded into TerraFly's code or stored in scripts. This severely limits the usefulness and flexibility of the database, and can be quite problematic should information about the data be required by someone who does not have access to TerraFly's code or scripts. Further, the inclusion of preprocessing information can help TerraFly provide a more interesting, easier to use and appealing interface for the user.

In order to deal effectively with different types of spatial data, similarly structured data can be stored in the same framework (i.e., using the same categories and relations). As can be seen below, spatial data can generally be categorized as either raster (e.g.,

Aerial Photography, Satellite imagery, etc.) or vector data (e.g., points, lines and polygons). As such, the same type of database query can request either Landsat or Ikonos imagery by indicating the type of data needed as opposed to having to search for a separate category.

DATA FORMAT DESCRIPTION

To further understand the needs of the database and appropriately represent the data, the components of each data type and the information that each component provides must be defined. The specific data format and components of the data used in this thesis are discussed below.

RASTER/IMAGE DATA

The raster data typically used by TerraFly, and thus for this thesis, includes both imagery and multispectral data. Images are commonly in TIFF, World-TIFF and JPEG image formats. Multispectral data is often in PPM format and contains different numbers of spectral bands that can be combined to create false color composite images. These composite images provide different types of information depending on the data source and spectral band combination used (see Chapter 2 for a more detailed discussion). Nevertheless, image and multispectral data is similar enough that they can be categorized together as raster data, and can be stored in the database using the same catogories and relationships.

Because Sem-ODB can store both textual and binary data in the same database, the data itself can be stored in the database along with its metadata. For raster type data, this metadata includes:

- Data type and name
- Data resolution (pixel size)
- Compression method/data format
- Origin coordinates
- Geographic projection
- Acquisition date
- Spectral band number (for multispectral data only)

Additional information related to preprocessing of the data also needs to be stored in the database. As was stated previously, this is data which TerraFly can specifically use to aid in dealing with the data. This information includes:

- Zoom level
- Tile height and width
- Coverage information such as County, State and Country
- Default color saturation
- Predefined band combinations
- Image color range
- Default blue, green and red band values

One of the primary challenges of storing different types of raster data in the same type of framework involves dealing with multispectral and non-multispectral (image) data together. Ideally, the database schema should be able to store these two different types of data using the same categories and relations. This will likely mean that when image data is stored, a number of attributes, relations and categories will be null.

Because we are using Sem-ODB, however, this is not a problem as null information does not take up additional space.

VECTOR/TEXTUAL DIGITAL DATA

The storage requirements and data format of the different types of raster data is fairly straightforward. Vector data, however, consists of points, lines and polygons, all of which are often found in the same data set. With this type of structure, it can be a more difficult to generalize data organization than it is for raster data. Thus, a closer look at the specific formats of the data of interest is needed. As was stated previously, the two types of data specifically used by TerraFly is Geographic Names Information System (GNIS) and U.S. Census Tiger/Line data. Detailed descriptions of the data formats of these two types of data are discussed in this subsection.

GEOGRAPHIC NAMES INFORMATION SYSTEMS (GNIS)

As was stated previously, GNIS data consists of names and types of places along with associated coordinate information. Thus, GNIS data contains point data which is categorized by the type of information each point represents. Each record contains the following fields:

Place_Name: String

Place_Type: String

Latitude: Integer or Float

Longitude: Integer or Float

All four of these fields are required. Each place has a Place_Name, but each name is not necessarily unique. Each place also has a Place_Type. Naturally, these place types are shared among the place names. There is also associated Latitude and Longitude coordinates. GNIS data provides these coordinate points in both degrees/minutes/seconds (as an integer) and decimal formats so that the desired format may be used without the need to convert between them. Each coordinate pair is not unique. One particular point may be associated with more than one place. For example, one GNIS record is:

Place_Name: Florida International University

Place_Type: school

Latitude: 25.667

Longitude: -80.57487

U.S. CENSUS TIGER/LINE

US Census Tiger/Line data consists of point, line and polygon data which provides information such as feature types, address ranges and ZIP Codes, codes for legal and statistical entities, landmark point features, area landmarks, latitude/longitude coordinates of linear and point features, key geographic features and area boundaries. This data is considerably more complicated than GNIS data because of the inclusion of this additional information. Tiger/Line data provides a wealth of information. We are going to focus on place information, street information and area feature information when dealing with the point, line and polygon data respectively. This data is most relevant for TerraFly and quite representative of other similar data provided by the U.S. Census.

Place Data

The format of the Tiger/Line place data is similar to that of GNIS data. Each record includes a place name, place type and associated latitude/longitude coordinates. As with GNIS data, a particular place name may be associated with more than one place record and location. Similarly, each latitude/longitude coordinate point may be associated with more than one place.

Street and Address Range Data

U.S. Census Tiger/Line street and address range data is line data and its associated information. It provides information on street names, street types, associated address ranges, beginning and ending latitude/longitude coordinate points as well as zip code information. The original data files are grouped by state and county.

Although the street information provided by Tiger/Line files seems straightforward, the actual structure of the data is not. The main reason for this is that most streets are not straight, continuous lines from beginning to end. Streets often curve, sometimes breaks or obstructions occur in a street and streets sometimes have different names in different areas. Because of this, streets are broken down into street segments and street sections. Together, these create what is commonly considered one street. The structure of the street information is discussed below.

Tiger/Line street data is organized by county and state. A street, as defined in the Tiger/Line data, is unique to each county. If an actual street crosses a county line, then it is considered a different street. An actual street is made up of street sections. Each street section has a unique street ID, as well as street name and street type. Each street section can have multiple names, and each street name can be associated with many street sections that are logically related. Each street section is classified by a street type. For example, one street section may have the following format:

Street_Section_ID: F1298395739589

Street_Name: Bird Road, SW 40th Street, etc.

Street_Type: Major Road

Each street section is made up of a number of street segments, and can not exist without at least one street segment. A street segment must be part of only one street section. Each street segment has associated with it coordinate point information, zip code information, and left and right side address ranges for the starting and ending points. The latitude/longitude points of each line segment are not unique. One street section's ending point is often the starting point of another street section (or, if it is a multiple-way intersection, it can be the starting point of another street section). Address ranges are also not unique. Two parallel street segments may have the same starting address ranges and/or ending address ranges. Naturally, zip codes associated with the start and end of each street segment are not unique, including in the same street segment. Finally, not all street segments include information on address ranges or zip codes. An example of a street section would be as follows:

Associated_Street_Section: F7385729783

To_Latitude: 25.78837

To_Longitude: -80.783297

From_Latitude: 25.79023

From_Longitude: -80.784837

Left_Starting_Address: 1500

Right_Starting_Address: 1501

Left_Ending_Address: 1598

Right_Ending_Address: 1599

Left_Zip_Code: 33173

Right_Zip_Code: 33183

Each street segment can be visualized as a straight line between two points. Hence, a street section can be visualized as a series of connected line segments.

Area Feature Data

U.S. Census Tiger/Line area features is polygon data and its related information. This type of data represents objects such as municipal and geographic boundaries. As with other Tiger/Line data, area features are organized by state and county.

To represent a polygon area, an ordered list of point coordinates is used. The first coordinate point in the list is always both the starting and ending point. By following the points in order, the boundary of the area of interest is established. Points in this boundary

are not unique. It is often the case that two or more area features share at least part of their boundary coordinates with each other and other area features. The number of points used to delimit different area boundaries is not uniform. Some areas may require ten points to delineate their boundaries whereas other areas may require significantly more. An example of an area feature would be as follows:

Area_Name: Kendall

Coordinate_List: 25.8798, -80.83789

25.8978, -80.84987

. . .

25.7878, -80.82739

SEMANTIC SCHEMA DESIGN

To deal effectively with this data, a Semantic database schema has been designed for use with TerraFly. In creating the database schema for this thesis, a number of issues must be taken into consideration. In short, these issues include relevant applications, data storage requirements, the characteristics of the data and possible future expansion of the project. These issues have been discussed in detail in previous chapters. This section will present and discuss the schema design in detail as well as how these issues effected its design. The subschema which deals with raster data is first discussed, followed by a discussion of the subschema which stores the vector digital data.

TERRAFLY DATABASE SUBSCHEMA 1: RASTER DATA STORAGE

There are two main goals that were focused upon when designing this part of the database schema. First, we wanted to be able to effectively store varied types of raster data, as well as its metadata and preprocessing information, in a uniform format. Second, we needed to ensure that structures for fast and efficient data retrieval and a logical design are supported. To accomplish this, the category DATASET was created and can be considered the central category of this subschema. A discussion of each category, starting with the DATASET category, can be found below. Subschema 1 is illustrated in Figure 8.

DATASET — category (A record of data set objects. This is, for example, 1995 color infra-red aerial photography for Florida.)

The DATASET category represents each data set object. It contains attributes and relations relevant to each data set, including relations to relevant counties and states. Each data set can be identified by its name. For example, one data set currently used by TerraFly is HPDRCAP, color infra-red aerial photography over Miami-Dade county. Another example is LANDSAT, which currently includes mosaicked Landsat 5 data over the state of Florida.

METADATA ATTRIBUTES

data-type-name — attribute of *DATASET*, range: *String* (*key*) (The unique name of each specific data set (i.e., Landsat-Miami).)

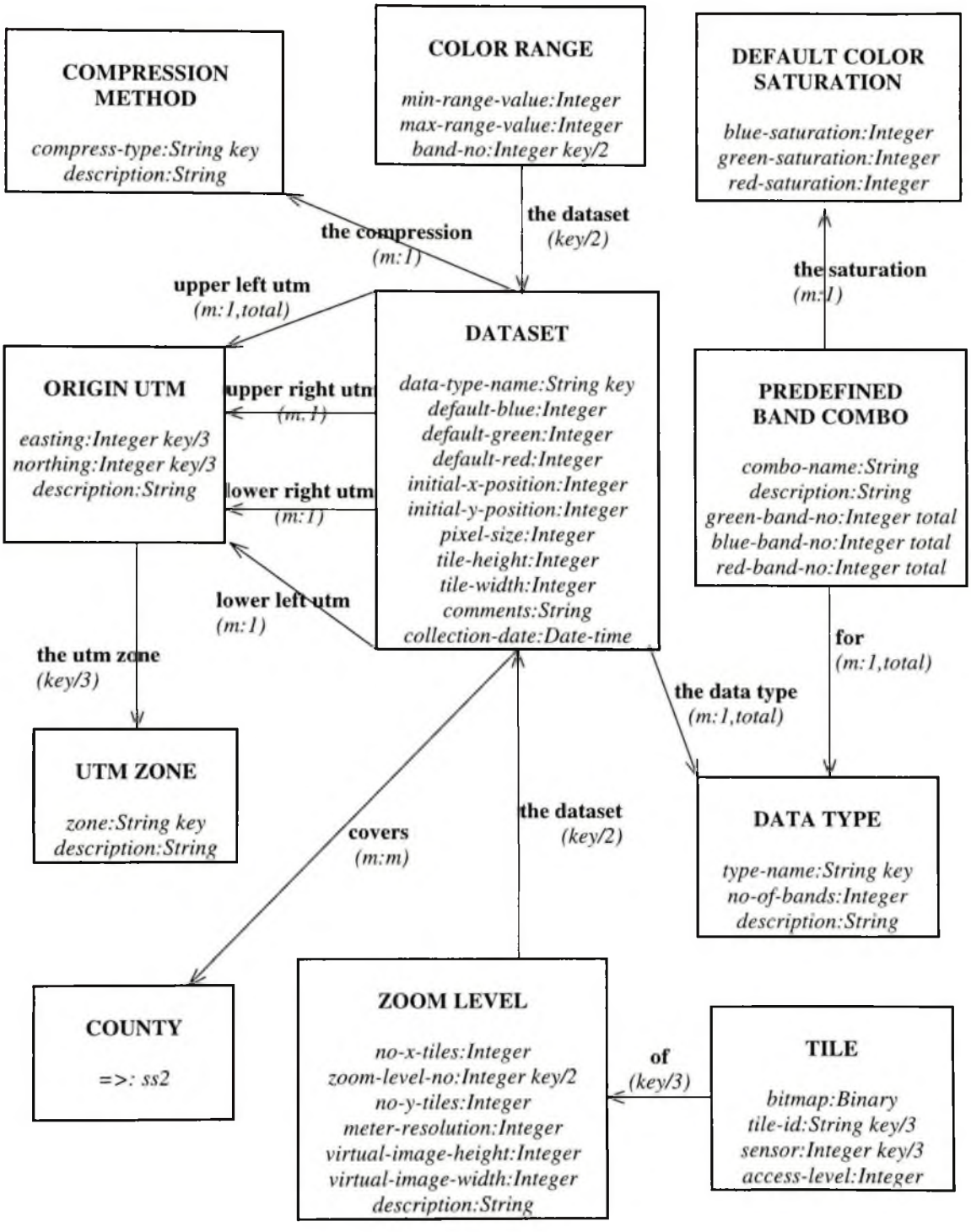

Figure 8. TerraFly Database Subschema 1: Raster Data Storage

pixel-size — attribute of *DATASET*, range: *Integer* (*m:1*) (The pixel-size in bits of the data-type.)

collection-date — attribute of *DATASET*, range: *Date-time* (*m:1*) (The acquisition date of the data set (e.g., the date the data set was collected).)

PREPROCESSING INFORMATION ATTRIBUTES

The tile height and width attributes are used to indicate the size of the tiles used for each data set. Including these two attributes allows for an arbitrary tile size which can be set based on the requirements of the user or application, thus providing greater flexibility. For example, previous versions of TerraFly used a tile height of 256 pixels. The current version of TerraFly uses a tile height of 512 pixels.

tile-height — attribute of *DATASET*, range: *Integer* (*m:1*) (The tile-height in pixels of the data set. This attribute allows for an arbitrary tile size, providing greater flexibility for various applications.)

tile-width — attribute of *DATASET*, range: *Integer* (*m:1*) (The tile-width in pixels of the data set. This attribute allows for an arbitrary tile size, providing greater flexibility for various applications.)

The initial x and y position attributes are used to indicate the default center point coordinates of the data to be retrieved from the data set should the requesting application not provide this information in its request. Including these two attributes allows a default position to be indicated without having to hard code this information into either application code or scripts.

initial-x-position — attribute of *DATASET*, range: *Integer* (*m:1*) (The initial UTM x/northing coordinate position of the data set. This attribute can be used to indicate what the default center coordinate of the data set should specific coordinates not be requested by the application retrieving the data.)

initial-y-position — attribute of *DATASET*, range: *Integer* (*m:1*) (The initial UTM y/easting coordinate position of the data set. This attribute can be used to indicate what the default center coordinate of the data set should specific coordinates not be requested by the application retrieving the data.)

The default blue, green and red attributes are used to indicate the default spatial band values for each data set when dealing with multispectral data. Including these three attributes allows for the creation of a default composite image without having to hard code this information into either application code or scripts. For example, for a Landsat 5 data set, the default can be set at band 3, band 2 and band 1 (for red, green, blue respectively) for a natural view.

default-blue — attribute of *DATASET*, range: *Integer* (*m:1*) (The default blue spatial band value of the data set when dealing with multispectral data should this information not be included in a data query by the application retrieving the data)

default-green — attribute of *DATASET*, range: *Integer* (*m:1*) (The default green spatial band value of the data set when dealing with multispectral data should this information not be included in a data query by the application retrieving the data)

default-red — attribute of *DATASET*, range: *Integer* (*m:1*) (The default red spatial band value of the data set when dealing with multispectral data should this information not be included in a data query by the application retrieving the data)

comments — attribute of *DATASET*, range: *String* (*m:1*) (This attribute provides an area where comments or a description of the data may be entered.)

RELATIONS

the-data-type — relation from *DATASET* to *DATA-TYPE* (*m:1,total*) (The data set's data type.. A data set can not exist without a data type.)

The following four relations can be used to determine the rectangular boundary of each data set. Because all data used with TerraFly must be georeferenced, the upper left UTM coordinate is required for each data set. The other origin coordinates are recommended, but not required.

upper-left-utm — relation from *DATASET* to *ORIGIN-UTM* (*m:1,total*) (The upper left UTM coordinates of the area encompassed by the data set. This coordinate is required for any given data set.)

upper-right-utm — relation from *DATASET* to *ORIGIN-UTM* (*m:1*) (The upper right UTM coordinates of the area encompassed by the data set. This coordinate is recommended but not required for any given data set.)

lower-left-utm — relation from *DATASET* to *ORIGIN-UTM* (*m:1*) (The lower

left UTM coordinates of the area encompassed by the data set. This coordinate is recommended but not required for any given data set.)

lower-right-utm — relation from *DATASET* to *ORIGIN-UTM* (*m:1*) (The lower right UTM coordinates of the area encompassed by the data set. This coordinate is recommended but not required for any given data set.)

the-compression — relation from *DATASET* to *COMPRESSION-METHOD* (*m:1*) (The type of compression or data format the data set is stored in. Some examples are JPEG, TIFF, PPM, G-ZIP, etc.)

covers — relation from *DATASET* to *COUNTY* (*m:m*) (This relation indicates which counties are at least partially covered by each particular data set. Because a data set may potentially include data for multiple counties or even states, this is a many-to-many relation.)

DATA-TYPE — category (A record of data types. This is, for example, aerial photography, Landsat 5, Landsat 7, IKONOS, etc.)

The DATA-TYPE category represents each data type object. Each data type is its own object which has attributes and relations relevant specifically to data types (as opposed to information specific to a particular data set). Each data type can be identified by its name. For example, one data type currently is Landsat 7. Another example is IKONOS imagery.

ATTRIBUTES

type-name — attribute of *DATA-TYPE*, range: *String* (*key*) (The unique name of each specific data type (i.e., Aerial Photography).)

no-of-bands — attribute of *DATA-TYPE*, range: *Integer* (*m:1*) (This attribute indicates the number of spectral bands for the data type. If this attribute is NULL, then there can not be a PREDEFINED-BAND-COMBO associated with this object (i.e., there can not be a for relation between them).)

description — attribute of *DATA-TYPE*, range: *String* (*m:1*) (This attribute provides an area where a description of the data type may be entered.)

PREDEFINED-BAND-COMBO — category (A record of the predefined-band-combo objects.)

The PREDEFINED-BAND-COMBO category represents a specific predefined band combination associated with a specific data type object. Each predefined band combination is its own object which has attributes and relations relevant specifically to these combinations. There can be numerous predefined band combinations for each data type, but a predefined band combination can not exist without a relation to a data type.

ATTRIBUTES

combo-name — attribute of *PREDEFINED-BAND-COMBO*, range: *String* (*m:1*) (The name of the predefined-name-combo. The name is commonly the type of information that the band combination provides. Some examples are natural

view, subsea vegetation, urban development, etc.)

description — attribute of *PREDEFINED-BAND-COMBO*, range: *String* (*m:1*) (This attribute provides an area where a description of the predefined band combination may be entered.)

The red, blue and green band numbers are used to indicate which bands should be combined to create the false color composite image for the predefined band combination. All three attributes must contain values for the predefined band combination to exist.

green-band-no — attribute of *PREDEFINED-BAND-COMBO*, range: *Integer* (*m:1,total*) (The spectral sensor number used in the green band to create the predefined band combination.)

blue-band-no — attribute of *PREDEFINED-BAND-COMBO*, range: *Integer* (*m:1,total*) (The spectral sensor number used in the blue band to create the predefined band combination.)

red-band-no — attribute of *PREDEFINED-BAND-COMBO*, range: *Integer* (*m:1,total*) (The spectral sensor number used in the red band to create the predefined band combination.)

RELATIONS

for — relation from *PREDEFINED-BAND-COMBO* to *DATA-TYPE* (*m:1,total*) (The data type the predefined band belongs to. The band combination used to create different views of the data will be different for different data types. For example, the band combination used to create a natural view for Landsat7 will

be different from the combination used for Hyperspectral data.)

the-saturation — relation from *PREDEFINED-BAND-COMBO* to *DEFAULT-COLOR-SATURATION* (*m:1*) (The color saturation that should be used for a particular predefined band combination.)

DEFAULT-COLOR-SATURATION — category (A record of default-color-saturation.)

The DEFAULT-COLOR-SATURATION category represents an image color saturation associated with one or more predefined band combination objects. Each color saturation is its own object which has three attributes which defines it. The information used to create this object is created during preprocessing. Its main purpose is to provide a visually more appealing false color composite image.

ATTRIBUTES

blue-saturation — attribute of *DEFAULT-COLOR-SATURATION*, range: *Integer* (*m:1*) (The percentage of blue color saturation in the default color saturation object.)

green-saturation — attribute of *DEFAULT-COLOR-SATURATION*, range: *Integer* (*m:1*) (The percentage of blue color saturation in the default color saturation object.)

red-saturation — attribute of *DEFAULT-COLOR-SATURATION*, range: *Integer* (*m:1*) (The percentage of blue color saturation in the default color saturation

object.)

ORIGIN-UTM — category (A record of OriginUTM types.)

The ORIGIN-UTM category represents a UTM coordinate which acts as a geographic point of reference for the origin of a data set. A set of two or four origin UTM objects (i.e., upper left and lower right, lower left and upper right, or upper left, upper right, lower left and lower right) associated with one data set is used to delimit the rectangular area the data set encompasses. Each origin UTM is uniquely identified by its northing and easting attribute values, and its UTM zone.

ATTRIBUTES

easting — attribute of *ORIGIN-UTM*, range: *Integer* (*key/3*) (The easting of the origin UTM coordinate.)

northing — attribute of *ORIGIN-UTM*, range: *Integer* (*key/3*) (The northing of the origin UTM coordinate.)

description — attribute of *ORIGIN-UTM*, range: *String* (*m:1*) (This attribute provides an area where a description of the origin UTM object may be entered.)

RELATIONS

the-utm-zone — relation from *ORIGIN-UTM* to *UTM-ZONE* (*key/3*) (The UTM zone for a given UTM origin coordinates.)

UTM-ZONE — category (A record of UTM-Zone types.)

The UTM-ZONE category represents a specific UTM zone object. This information is used to determine where on the earth the origin UTM coordinates associated with it are found. An origin UTM object can not exist without an associated UTM zone object.

ATTRIBUTES

zone — attribute of *UTM-ZONE*, range: *String* (*key*) (The zone of the UTM-zone object.)

description — attribute of *UTM-ZONE*, range: *String* (*m:1*) (This attribute provides an area where a description of the UTM zone object may be entered.)

ZOOM-LEVEL — category (A record of original and derived zoom (resolution) levels.)

The ZOOM-LEVEL category represents an original or derived zoom level. Each zoom level is uniquely identified by its zoom level number and the data set associated with it. A zoom level of 0 indicates that the tiles associated with the zoom level are at the original resolution of the data. All other levels are derived from this. This category includes as its attributes both data resolution and zoom level number. It could be argued that including both of these is redundant and unnecessary. However, there may be cases where the original data set is not loaded into the database, yet a way of calculating the

resolution of the original data set is still desired. For example, there may occur a case where a given data set is provided on the terms that the data not be placed in the database at its original resolution for viewing in TerraFly. Yet it is still available for purchase through TerraFly. By including information for both of these attributes, the original resolution can easily be calculated without human intervention.

ATTRIBUTES

zoom-level-no — attribute of *ZOOM-LEVEL*, range: *Integer* (*key/2*) (The level number of the zoom level. The original resolution of the data set is always set at level 0. Additional levels are derived from there.)

meter-resolution — attribute of *ZOOM-LEVEL*, range: *Integer* (*m:1,total*) (The resolution in meters of the zoom level.)

The following four attributes hold preprocessed information. Although this information can be derived from other related data, the inclusion of this information without having to calculate it can improve image processing speed and database retrieval.

no-x-tiles — attribute of *ZOOM-LEVEL*, range: *Integer* (*m:1*) (The number of x tiles that makes up this zoom level.)

no-y-tiles — attribute of *ZOOM-LEVEL*, range: *Integer* (*m:1*) (The number of y tiles that makes up this zoom level.)

virtual-image-height — attribute of *ZOOM-LEVEL*, range: *Integer* (*m:1*) (The number of pixels in the height of this zoom level of the data set.)

virtual-image-width — attribute of *ZOOM-LEVEL*, range: *Integer* (*m:1*) (The

number of pixels in the width of this zoom level of the data set.)

description — attribute of *ZOOM-LEVEL*, range: *String* (*m:1*) (This attribute provides an area where a description of the zoom level object may be entered.)

RELATIONS

the-dataset — relation from *ZOOM-LEVEL* to *DATASET* (*key/2*) (The data set that this zoom level belongs to. A zoom level can not exist with out an associated data set.)

TILE — category (A record of tile objects.)

The TILE category represents each individual tile of the data set, categorized by zoom level. In order to be able to retrieve raster data from the data base in a fast and efficient manner, the original data is mosaicked together (if more than one data file is involved) and cut into tiles prior to being put in the database. In this way, data can be retrieved only as needed tile by tile, instead of having to retrieve the entire data set. Each tile is uniquely identified by its tile ID, spatial sensor number and the zoom level associated with it. If the data set this tile belongs to is not multispectral data, then the sensor number defaults to zero.

ATTRIBUTES

bitmap — attribute of *TILE*, range: *Binary* (*m:1*) (The bitmap data of the tile.)

tile-id — attribute of *TILE*, range: *String* (*key/3*) (The tile ID created during the static tile grid mosaicking process.)

sensor — attribute of *TILE*, range: *Integer* (*key/3*) (The spatial sensor number of the tile. If the data set the tile belongs to is not mulitspectral, the sensor number defaults to zero.)

access-level — attribute of *TILE*, range: *Integer* (*m:1*) (The access level that a user must have to be able to view this tile; the default is 0, which is HPDRC access)

RELATIONS

of — relation from *TILE* to *ZOOM-LEVEL* (*key/3*) (The zoom level the tile belongs to.)

COMPRESSION-METHOD — category (A record of compression method/data format objects.)

The COMPRESSION-METHOD category represents the compression method/data format used for the data set. Each compression method is uniquely identified by its type.

ATTRIBUTES

compress-type — attribute of *COMPRESSION-METHOD*, range: *String* (*key*)

(The compression type of the compression method. Some examples of this are

JPEG, TIFF, GZIP, etc.)

description — attribute of *COMPRESSION-METHOD*, range: *String* (*m:1*) (This attribute provides an area where a description of the compression method object may be entered.)

COLOR-RANGE — category

The COLOR-RANGE category represents the optimal color range for the data set. This range is determined during preprocessing of the data and is used to make the data set's images more visually appealing. Each color range is uniquely identified by its band number and the data set associated with it. The range is typically between 0 and 256. A limitation on this range has not been put in place, however, to allow for any future image processing changes that may occur to the standard range.

ATTRIBUTES

min-range-value — attribute of *COLOR-RANGE*, range: *Integer* (*m:1*) (The minimum color range value for this color range object.)

max-range-value — attribute of *COLOR-RANGE*, range: *Integer* (*m:1*) (The maximum color range value for this color range object.)

band-no — attribute of *COLOR-RANGE*, range: *Integer* (*key/2*) (The spectral sensor number associated with this color range. If the data is not multispectral data, this attribute defaults to zero.)

RELATIONS

the-dataset — relation from *COLOR-RANGE* to *DATASET* (*key/2*) (The data set that the color range belongs to.)

COUNTY — category (See subschema ss2.)

TERRAFLY DATABASE SUBSCHEMA 2: VECTOR DIGITAL DATA

This part of the database schema deals with the efficient storage and retrieval of the vector digital data, namely the GNIS and U.S. Census Tiger;/Line data, previously discussed. As with the previous subschema, we want to be able to effectively store varied types of similar data in a uniform format. A logical design with fast data access capabilities is also required. To help accomplish this, a combination of a good schema design and data preprocessing is used.

One thing to note when looking at this subschema is that the names given to the categories are based on U.S. data and information specific to the TerraFly application. This was primarily done for better understanding of the schema since this is the data that is primarily dealt with at HPDRC. The file format of the Tiger/Line files, in particular, are rather complex and difficult to understand. Having similarly named categories will help with understanding how the data maps to the schema. However, information regarding similar types of non-U.S. data (e.g., Canadian Geographic Names Database, Canadian Street Files, etc.) have been analyzed and incorporated into the design of the

subschema. The effects of this are noted in the discussions of the individual categories and relations.

Unlike subschema 1, this part of the database schema does not center upon one category, but its organization can be subcategorized into three areas of interest. These areas involve dealing with place data (i.e., point-based data), street and address range data (i.e., line data) and area data (i.e., polygon data). Although these three areas can be viewed as quasi-independent of each other, they do have shared categories in common. A discussion of each category, starting with the shared categories, can be found below. Subschema 2 is illustrated in Figure 9.

SHARED CATEGORIES AND RELATIONS

COUNTY — category (A record of counties. This is, for example, Miami-Dade County. For countries other than the U.S., municipal areas equivalent to a County would be placed in this category.)

The COUNTY category represents each County level municipal object. It contains two attributes and one relation. Both GNIS and U.S. Census Tiger/Line files organize data by county and state. Each county can be identified by its name and which state it is a county of. For example, Miami-Dade county is identified by Miami-Dade and Florida.

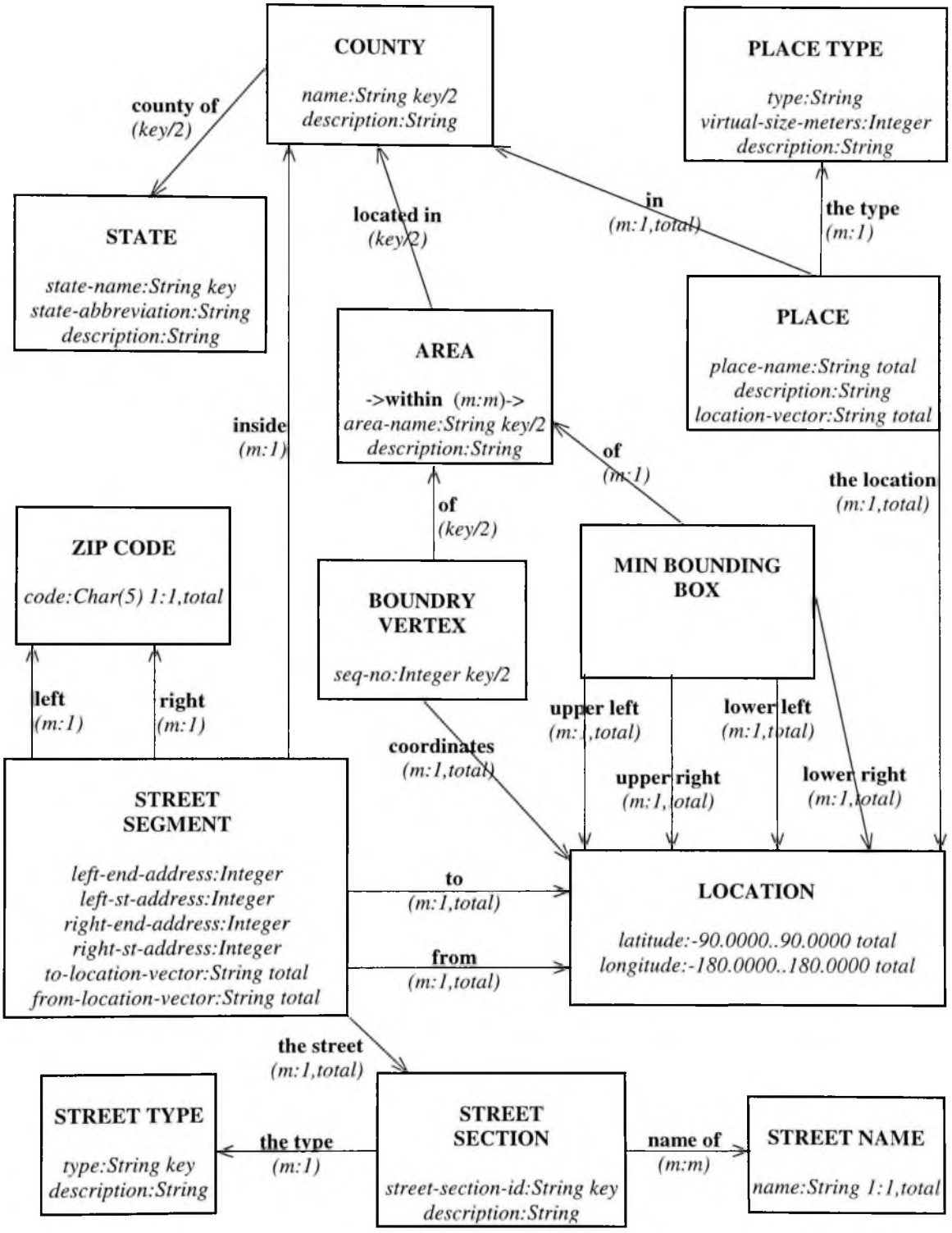

Figure 9. TerraFly Database Subschema 2: Vector Digital Data Storage.

ATTRIBUTES

name — attribute of *COUNTY*, range: *String* (*key/2*) (The name of the county.)

description — attribute of *COUNTY*, range: *String* (*m:1*) (This attribute provides an area where a description of the data may be entered.)

RELATIONS

county-of — relation from *COUNTY* to *STATE* (*key/2*) (The state or province the county belongs to. A county can not exist without a state.)

STATE — category (A record of states, territories and provinces.)

The STATE category represents each State, Territory or Province level object. Although not represented in this subschema (due to space limitations), in addition to its three attributes, it should also have a relation to a country category, and perhaps in the far future, the country should have a relation to a planetoid category. With the inclusion of a country category, each state can be identified by its name and country.

ATTRIBUTES

state-name — attribute of *STATE*, range: *String* (*key*) (The name of the state, territory or province.)

state-abbreviation — attribute of *STATE*, range: *String* (*m:1*) (A short abbreviation used to represent the state, usually containing two letters.)

description — attribute of *STATE*, range: *String* (*m:1*) (This attribute provides an area where a description of the data may be entered.)

LOCATION — category (A record of latitude/longitude point coordinates.)

The LOCATION category represents a latitude/longitude point location. Each location must have a latitude and longitude coordinate. Each location point can be associated with all three main types of vector digital data.

ATTRIBUTES

latitude — attribute of *LOCATION*, range: *-90.0000..90.0000* (*m:1,total*) (The latitude of the location in decimal format. If the original coordinate is in degrees, minutes and seconds, it is converted prior to being put in the database.)

longitude — attribute of *LOCATION*, range: *-180.0000..180.0000* (*m:1,total*) (The longitude of the location in decimal format. If the original coordinate is in degrees, minutes and seconds, it is converted prior to being put in the database.)

PLACE CATEGORIES AND RELATIONS

PLACE — category (A record of places of interest.)

The PLACE category represents point of interest objects. A place object must have a name, location vector and a relation to country it is in. Neither place names nor location vectors are unique. A place's location vector is created using the place's latitude/longitude coordinate during preprocessing of the data and exists to increase range search efficiency. It is part of the addressing scheme and is discussed in detail in Chapter 6.

ATTRIBUTES

place-name — attribute of *PLACE*, range: *String* (*m:1,total*) (The name of the place.)

location-vector — attribute of *PLACE*, range: *String* (*m:1,total*) (This attributes represents the place's latitude/longitude coordinates using bit interleaving. This attribute is included to increase search and retrieval speed. The location object associated with the place object can not be changed unless the location vector is also changed. Similarly, the location vector can not be changed unless the location object is changed.)

description — attribute of *PLACE*, range: *String* (*m:1*) (This attribute provides an area where a description of the data may be entered.)

RELATIONS

in — relation from *PLACE* to *COUNTY* (*m:1,total*) (The county in which the place is located.)

the-location — relation from *PLACE* to *LOCATION* (*m:1,total*) (The

georeferenced location of the place.)

the-type — relation from *PLACE* to *PLACE-TYPE* (*m:1*) (The type of the place.)

PLACE-TYPE — category (A list of types of places of interest.)

The PLACE-TYPE category represents type of place objects. A place type object has a type name and virtual size. The value for the virtual size attribute is created at the time the data is entered in the database. It is an estimate of the average size of the type of place and can be used to limit the type of places retrieved from the database.

ATTRIBUTES

type — attribute of *PLACE-TYPE*, range: *String* (*m:1*) (The type of the place.)

virtual-size-meters — attribute of *PLACE-TYPE*, range: *Integer* (*m:1*) (The virtual size of the place type.)

description — attribute of *PLACE-TYPE*, range: *String* (*m:1*) (This attribute provides an area where a description of the data may be entered.)

STREET AND ADDRESS RANGE CATEGORIES AND RELATIONS

STREET-SECTION — category (A record of streets.)

The STREET-SECTION category represents a street object. In U.S. Census Tiger/Line data, a street section represents a street within a county. This is not

necessarily the case with other types of data such as Canadian Street Files. Each street section is uniquely identified by its ID, and usually has a street name and street type. It can have more than one name, but only one street type.

ATTRIBUTES

street-section-id — attribute of *STREET-SECTION*, range: *String* (*key*) (The street ID uniquely identifies the street.)

description — attribute of *STREET-SECTION*, range: *String* (*m:1*) (This attribute provides an area where a description of the data may be entered.)

RELATIONS

name-of — relation from *STREET-SECTION* to *STREET-NAME* (*m:m*) (The name of the street. Each street section can have more than one name and each name can be associated with more than one street section.)

the-type — relation from *STREET-SECTION* to *STREET-TYPE* (*m:1*) (The street's type.)

STREET-NAME — category (A record of street names.)

The STREET-NAME category represents the names of streets. A street name must have a name and no other street name object can have the same name.

ATTRIBUTES

name — attribute of *STREET-NAME*, range: *String* (*1:1,total*) (The street name.)

STREET-TYPE — category (A record of street types.)

The STREET-TYPE category represents a type of street. A street type is uniquely identified by its type name. Each street type can be associated with many street sections, but each street section can only be of one street type.

ATTRIBUTES

type — attribute of *STREET-TYPE*, range: *String* (*key*) (Contains the name of the type of street.)

description — attribute of *STREET-TYPE*, range: *String* (*m:1*) (This attribute provides an area where a description of the data may be entered.)

STREET-SEGMENT — category (A record of segments.)

The STREET-SEGMENT category represents part of a street which can be represented by a single line segment. A street section is made up of one or more street segments. A street segment can not exist which is not part of a street section. Each street segment has starting and ending address range attributes, left and right side zip codes, and to and from location vectors. As with the place location vector, the to and from location vectors are created using the street segment's latitude/longitude coordinates during

preprocessing of the data. They exist to increase range search efficiency and are part of the addressing scheme discussed in detail in Chapter 6.

The street segment also has a relation with the county it is located in. For U.S. Census Tiger/Line data, street sections are more directly associated with counties. As such, it would be more appropriate to have the 'inside' relation from street section to county. However, in gathering information on the street files of other countries (e.g., Canadian Street Network Files), having the relation from the smaller street segments to the county objects would be more flexible and representative of these other types of data sets.

ATTRIBUTES

right-st-address — attribute of *STREET-SEGMENT*, range: *Integer* (*m:1*) (The right starting address of the street segment.)

left-st-address — attribute of *STREET-SEGMENT*, range: *Integer* (*m:1*) (The left starting address of the street segment.)

right-end-address — attribute of *STREET-SEGMENT*, range: *Integer* (*m:1*) (The right ending address of the street segment.)

left-end-address — attribute of *STREET-SEGMENT*, range: *Integer* (*m:1*) (The left ending address of the street segment.)

to-location-vector — attribute of *STREET-SEGMENT*, range: *String* (*m:1,total*) (This attributes represents each of the street segment's latitude/longitude coordinates using bit interleaving. This attribute is included to increase search

and retrieval speed. The location object associated with the street segment's to relation can not be changed unless the to location vector is also changed. Similarly, the to location vector can not be changed unless the corresponding location object is changed.)

from-location-vector — attribute of *STREET-SEGMENT*, range: *String* (*m:1,total*) (This attributes represents each of the street segment's latitude/longitude coordinates using bit interleaving. This attribute is included to increase search and retrieval speed. The location object associated with the street segment's to relation can not be changed unless the location vector is also changed. Similarly, the to location vector can not be changed unless the corresponding location object is changed.)

RELATIONS

the-street — relation from *STREET-SEGMENT* to *STREET-SECTION* (*m:1,total*) (The street which the street segment is part of.)

from — relation from *STREET-SEGMENT* to *LOCATION* (*m:1,total*) (The starting georeferenced location of the street segment.)

to — relation from *STREET-SEGMENT* to *LOCATION* (*m:1,total*) (The ending georeferenced location of the street segment.)

inside — relation from *STREET-SEGMENT* to *COUNTY* (*m:1*) (The county in which the street segment is located.)

left — relation from *STREET-SEGMENT* to *ZIP-CODE* (*m:1*) (The zip code of the left side of the segment.)

right — relation from *STREET-SEGMENT* to *ZIP-CODE* (*m:1*) (The zip code of the right side of the segment.)

ZIP-CODE — category (A record of zip-codes.)

The ZIP-CODE category represents a zip code. A zip code object can not exist without its code.

ATTRIBUTES

code — attribute of *ZIP-CODE*, range: *Char(5)* (*1:1,total*) (The zip code.)

AREA CATEGORIES AND RELATIONS

AREA — category (A record of areas.)

The AREA category represents a two-dimensional polygon area, often delimiting a municipal boundary. An area is uniquely identified by its name and the county it is located in. An area can be a city, suburb or any place of interest that occupies a region as opposed to a point (although it is not unusual for an area's center point to be included in the place data as a point). An area also has a relation with other areas. The relation indicates which areas lie within each other. This data is not currently included in the database as it would require substantial preprocessing, but the relation is included for possible future use.

ATTRIBUTES

area-name — attribute of *AREA*, range: *String* (*key/2*) (The name of the area.)

description — attribute of *AREA*, range: *String* (*m:1*) (This attribute provides an area where a description of the data may be entered.)

RELATIONS

within — relation from *AREA* to *AREA* (*m:m*) (The area this area is encompassed by.)

located-in — relation from *AREA* to *COUNTY* (*key/2*) (The county the area is located in.)

BOUNDRY-VERTEX — category (A record of area boundary vertices.)

The BOUNDRY-VERTEX category represents one element of an area's ordered point boundary. Taken together, the set of an area's boundary vertices creates the polygon shape of the area. Each boundary vertex is uniquely identified by its sequence number and the area it is part of. There is no standard number of boundary vertexes per area.

ATTRIBUTES

seq-no — attribute of *BOUNDRY-VERTEX*, range: *Integer* (*key/2*) (The sequential number of the vertex of an area.)

RELATIONS

of — relation from *BOUNDRY-VERTEX* to *AREA* (*key/2*) (The area the vertex is part of.)

coordinates — relation from *BOUNDRY-VERTEX* to *LOCATION* (*m:1,total*) (The geographic location of the vertex.)

MIN-BOUNDING-BOX — category (The minimal bounding box which contains the related area.)

The MIN-BOUNDING-BOX category represents the minimum bounding box which encompasses the area. It must have four location coordinates associated with it, one for each corner of the box.

RELATIONS

upper-left — relation from *MIN-BOUNDING-BOX* to *LOCATION* (*m:1,total*) (The upper left coordinate of the bounding box)

upper-right — relation from *MIN-BOUNDING-BOX* to *LOCATION* (*m:1,total*) (The upper right coordinate of the bounding box.)

lower-left — relation from *MIN-BOUNDING-BOX* to *LOCATION* (*m:1,total*) (The lower left coordinate of the bounding box.)

lower-right — relation from *MIN-BOUNDING-BOX* to *LOCATION* (*m:1,total*) (The lower right coordinates of the bounding box.)

of — relation from *MIN-BOUNDING-BOX* to *AREA* (*m:1*) (The area to which this bounding box belongs.)

VI. ADDRESSING SCHEME

When dealing with vector-based digital data, it is most often the case that a much larger number of objects must be retrieved at once from the database than with raster data. This typically involves the use of range searches which are based on positional values rather than searches based on comparisons.

Relational databases search based on comparison and are inefficient when it comes to proximity searchers. Sem-ODB, like the spatial data structures discussed in Chapter 3, is quite efficient at these types of searches. The reason for this is because the Sem-ODB engine uses a B-tree structure to help organize the data. Spatial objects are stored both physically and logically close to each other in the database.

Although this structure does provide efficient proximity-based range searches, the efficiency of these searches when dealing with two-dimensional data can be enhanced by preprocessing the data and using a one-dimensional addressing scheme. In short, this linear addressing scheme maps an object's 2-dimentional (latitude/longitude) location information to a 1-dimensional integrated integer value. Range searches are then performed based on these 1-dimensional integrated integer values. This chapter discusses the implementation of this linear addressing scheme in TerraFly's database schema, as well as an efficient search algorithm and possible applications for which this addressing scheme is potentially valuable.

ADDRESSING SCHEME OVERVIEW

As was discussed in Chapter 3, there are a number of data structures which are quite efficient at handling spatial data. These techniques are often employed in relational database systems to provide more appropriate means of storing and retrieving spatial data. Implementing these data structures in a database, however, can be quite complex. Because of the way Sem-ODB handles spatial data, a simpler yet still highly effective manner of dealing with spatial data, specifically range searches of point data, can be implemented. We are calling this technique our linear addressing scheme.

This addressing scheme involves converting latitude and longitude coordinates into 32-bit binary representations and interleaving the bits of these two representations to create a 64-bit integer representation of the point's location (i.e., its linear address). Then, when a place object is entered into the database, its corresponding 64-bit linear address is entered with it. This will allow users to query the database via this linear address.

To query the database, a decomposition of the spatial area is accomplished using the linear address. When data for an area of interest is requested, it is in the form of a bounding box containing the area of interest. The coordinates of this box (e.g., lower left and upper right) are used in a recursive search algorithm to narrow down the range search and query the database via the linear addresses of these coordinates. Doing this, in essence, provides a decomposition of the spatial area. The use of this structure emulates

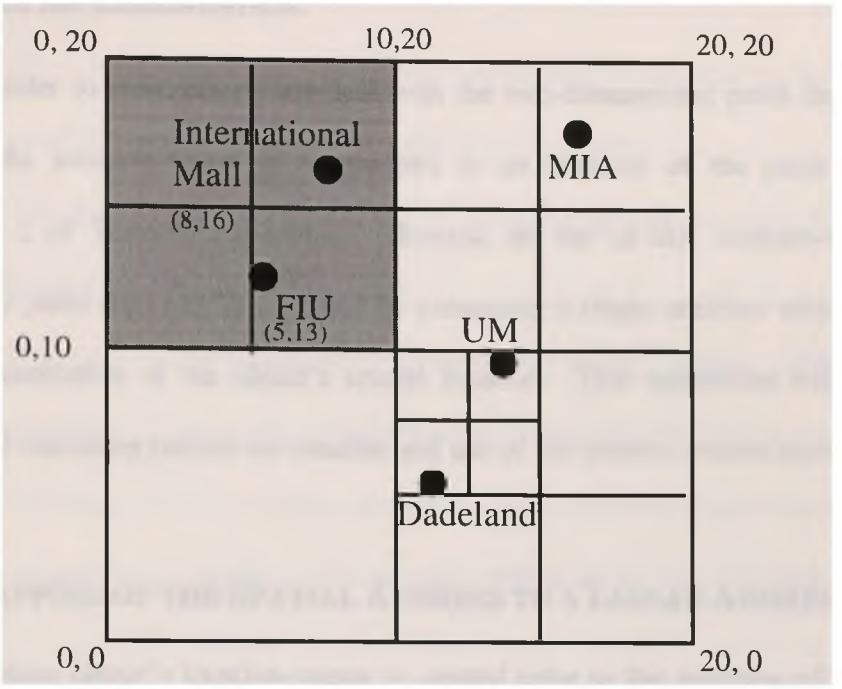

Figure 10. Quad-tree Representation of the Addressing Scheme

the use of a quad tree in Sem-ODB, where the searching algorithm recursively calculates how to conduct the range query.

A simplified example is illustrated in Figure 10. If the range query is for all points in the shaded area, then the lower left (0,10) and upper right (10,20) coordinates would be used to create the query. Only the points within that shaded area would be retrieved. Note how this structure is similar to the PR quad-tree discussed in Chapter 3. Of course, although the structure looks similar, this is more of an emulation than an actual implementation of a PR quad-tree. The actual process is discussed in detail in the next two subsections of this chapter.

DATABASE IMPLEMENTATION

In order to more effectively deal with the two-dimensional point data within the database, the location-vector is represented as an attribute of the place category in subschema 2 of TerraFly's database. Through the use of this location-vector, range searches for point data can be executed by examining a single attribute which is a sorted linear representation of the object's spatial location. This subsection will discuss the process and reasoning behind the creation and use of the location-vector attribute.

MAPPING OF THE SPATIAL ADDRESS TO A LINEAR ADDRESS

A place object's location-vector is created prior to the insertion of the data into the database by combining the place's latitude/longitude location coordinates into a one-dimensional linear address via bit interleaving. The process for doing this is as follows.

Step 1: Convert the point's latitude coordinates (in decimal format) to a 32-bit binary representation format. This is accomplished as follows:

- The most significant bit (MSB) indicates the sign where 1 is negative and 0 is positive.
- The next 8 bits represent the degrees in binary format. Because we are dealing with latitude, this number will range between 00 and 90. When needed, leading zeros will be used to fill out the 8 bits.
- The next 23 bits represent the binary representation of the decimal portion of the latitude coordinate. This number will range between 000000 and 999999.

Step 2: Convert the point's longitude coordinates (in decimal format) to a 32-bit binary representation format. This is accomplished as follows:

- The most significant bit (MSB) indicates the sign where 1 is negative and 0 is positive.
- The next 8 bits represent the degrees in binary format. Because we are dealing with longitude, this number will range between 000 and 180.
- The next 23 bits represent the binary representation of the decimal portion of the latitude coordinate. This number will range between 000000 and 999999.

Step 3: Use bit interleaving to create a 64-bit vector representing the point's location. This vector is entered as the data value for location-vector and is used to more effectively retrieve the point data. This is accomplished by:

- Take the MSB of the 32-bit latitude representation as the MSB of the 64 bit vector.
- The MSB bit of the 32-bit longitude representation is then taken as the next bit of the 64-bit vector.
- This interleaving of the 32-bit latitude and 32-bit longitude representations would continue until the 64-bit vector is completed.

Thus, the most significant two bits of the 64-bit vector represent the sign values of the latitude and longitude coordinates respectively. If the original 32-bit representations

were numbered from 31 to 0, then the resulting 64-bit vector would look something like this:

$Lat_{31}Long_{31}Lat_{30}Long_{30}...Lat_{1}Long_{1}Lat_{0}Long_{0}$

As a result of this, each static GIS place object is stored in the database with its 64-bit vector. Because we are using Sem-ODB, objects that have the same latitude and longitude coordinates will have similar location-vectors and will be clustered in adjacent positions. This would necessitate an additional constraint on the database. The original latitude and longitude values of any stored object which includes a location-vector attribute should not be modified without the associated modification of the location-vector attribute. Further, the location vector cannot be modified unless the original latitude and longitude values are first modified

SEARCH ALGORITHMS

This subsection discusses the algorithm used to perform the range query for point-based data. As was stated previously, the range query is in the form of a bounding box where the lower left (LL) and upper right (UR) coordinates are used as the search parameters. When a small spatial range includes a major quad segment boundary (e.g. LL_latitude: 25.9888 and UR_latitude: 26.12345), the linear range becomes too large to encompass both adjacent major quad regions. This, in turn, increases the number of objects retrieved and deteriorates the performance. Thus, a solution is to segment the spatial range into multiple sub-spatial ranges.

A recursive algorithm to accomplish this solution takes the boundary coordinates of a bounding box as its input. Processing of the bounding box coordinates determines whether the bounding box needs to be further reduced and the algorithm called recursively with the smaller bounding boxes. The algorithms is as follows:

Step 1. Convert these two sets of coordinates (LL and UR) into 32-bit and then 64-bit representations. The process for this is the same as for the creation of the location-vector as described above.

Step 2. Perform an exclusive OR (XOR) between the 64-bit representations of LL and UR. The index of the *leading bit* on the resulting vector is the position of the left most bit that has a value of 1 in the 64-bit vector (with index range 63...0). The actual_variation_length of the resulting vector is the index of the leading bit plus one.

Step 3. Calculate the absolute numerical difference between the LL and UR latitude, and convert this difference into the 32-bit representation as described above. The same should be done with the LL and UR longitude coordinates. These two 32-bit representations should then be interleaved to create a 64-bit representation. The length of resulting value is assigned as the minimum_variation_length value. Note that the minimum_variation_length value will always be less than or equal to the actual_variation_length value.

Step 4. Determine whether the bounding box should be further reduced or if all of the points encompassed by the bounding box should be retrieved. This is done as follows:

```
If (actual_variation_length - minimum_variation_length) > variation_length_threshold
    {
        find (subdividing-latitude-boundary and subdividing-longitude-boundary)
        split the bounding box using these boundaries
        recursively process these new bounding boxes
    }
    else  return data from the bounding box
```

Where the subdividing-latitude-boundary and subdividing-longitude-boundary are determined by the leading bit of the actual_variation_length XOR bits, and the variation_length_threshold is a user-defined acceptable length variant.

This process may seem complex, but in order to appropriately limit the number of objects retrieved, it is essential.

POSSIBLE APPLICATIONS

There are numerous applications for which the addressing scheme can be particularly useful. It can be used to enhance some of TerraFly's current features such as retrieving objects for the Go-To Place feature. More interestingly, new features which would require fast database retrieval of point-based data can be added to TerraFly.

One new possible feature would involve querying for the nearest street intersections. In discussing the usefulness of street information with TerraFly's current users, many indicated that information regarding the closest intersection would be more useful than merely information regarding the closest street. Although the street and address range data is line-based data, to-location-vector and from-location-vector attributes were added to the database schema allow this data to be queried as if one were dealing with point-based data. The algorithm for data retrieval would be similar to that of places. This algorithm would be as follows:

Step 1. From the current position, create a bounding box with a radial distance r between the current position and the bounding box's boundaries.

Step 2. Apply a recursive function to determine whether the bounding box should be further reduced or if all of the points encompassed by the bounding box should be retrieved.

```
For each bounding box
{
        Find the set of location vectors
        For each vector
        {
                find the set of associated segments
                If two or more segments are related to the same location
```

Then it is an intersection

}

}

Another new feature could involve constant searching and display of the retrieved point-based data on the client side. More specifically, the place closest to the center point of the image would be displayed while the user is 'flying' over the data. Because the greatest lag when retrieving data involves transmittal over the Internet, this would involve constant searching on the client side. This would work as follows:

Step 1. A range search as described above would be performed on the server side to retrieve the data of interest.

Step 2. Once the data is retrieved, it is placed into a data structure which can quickly and easily be searched. The data structure recommended for this task is an R-tree. R-trees have been found to be very efficient at dealing with two-dimensional point data, particularly when the R-tree is stored in memory as it would be on the client side. A description of the R-tree data structure can be found in Chapter 3.

Step 3. Once the data is in the appropriate data structure, it can be sent to the client as one package (e.g., XML can be used to accomplish this)

Step 4. Once the client receives the data, searching for the place closest to the center point of the image can begin. This will continue until a preset boundary is crossed and a new set of data is requested and received from the server.

VII. CONCLUSION

The ability to efficiently store and retrieve spatial data is an area which is steadily growing in importance. This is primarily due to the increased demand, availability and usefulness of this data. The information that can be gleaned from this data is enormous, and its increased availability and new technology has reduced the cost of acquiring this data. As a result, it has begun to be used for applications from real estate endeavors to cadastral mapping.

Remotely-sensed data, however, is complex and difficult to deal with. This data is inherently very large and storage of the data often involves large amounts of data. Use of the data often requires powerful and expensive hardware and software systems, greatly limiting potential users. TerraFly, a prototype interactive vehicle for flying over and manipulating remotely-sensed data via any standard Java-enabled Web browser, is currently under development to address some of these issues. In order for TerraFly to work effectively, it must have an efficient and secure way of storing and retrieving its data. This thesis has addressed this issue by researching various storage requirements and options, and providing a combination of solutions.

A substantial amount of research has been done on various data structures that can be used to efficiently store spatial data. These structures include, but are not limited to point quad-trees, K-D trees, region quad-trees and R-trees. Each of these has its own advantages and disadvantages when dealing with different types of data. A main

disadvantage of these types of structures is the complexity involved in implementing them, particularly when trying to implement them within a database. The general implementation of these structures in a database also does not take into account any specialized capabilities a particular database system may have.

Many GIS systems employ the use of a database to store and retrieve spatial data. Because of the large size and value of the data, efficient and secure databases are needed for data storage. Relational database systems have been found to be very inefficient at dealing with spatial data. Research into other types of databases, such as Object-oriented database systems, has shown that they are more effective at handling spatial data. The Semantic Object-oriented Database Management System (Sem-ODB) was specifically designed to deal effectively with spatial data. As a result, Sem-ODB was chosen as the database for this thesis.

To effectively deal with TerraFly's data storage requirements, a semantic schema has been designed to efficiently store different types of spatial data in the same database. This schema provides a uniform manner of storing data from various sources. Specifically, different types of raster data are uniformly stored using the same database categories and relations, as are the point, line and polygon types of data. Structures for fast and efficient data retrieval were included in the database schema, and metadata and preprocessing information regarding the data has been preserved. This was all accomplished while providing a logical and secure database schema design.

To provide more effective retrieval of large amounts of point-based vector data, this database schema includes in its design a linear addressing scheme for remotely-sensed objects which maps an object's 2-dimentional (latitude/longitude) location information to a 1-dimensional integrated integer value. In conjunction with this, an algorithm for transforming a remotely-sensed range search into a number of linear segments of objects in the 1-dimensional array was investigated. Through the use of a binary representation and bit interleaving, this addressing scheme can be used to quickly and easily perform a range search to retrieve point-based vector data from Sem-ODB.

The research involved with this thesis focused on the effective storage and retrieval of remotely-sensed data and its relation information. There is still a great deal of research that can be done on various levels to further improve the storage and retrieval of spatial data, as well as the dissemination of this data to the public. Some of the suggested areas of future research includes:

- Using the data storage and retrieval capabilities discussed in this thesis to create new features for the TerraFly system. For example, a number of possible extensions of the TerraFly system were discussed in Chapter 6 which could take advantage of fast retrieval of places and street intersections.
- Creating more efficient structures for the storage and retrieval of line-based street data and polygon-based area information. The current system provides the capability to store preprocessing information related to these two types of data, such as a minimum bounding box for the area data. These two areas, however, were not focused upon as the point-based data was. Future work

could focus more on finding more efficient algorithms for dealing with these types of data

- Investigating efficient storage and retrieval of higher dimensional spatial data from a semantic database. The current research focuses on two-dimensional data. It is likely that in the future, TerraFly will support higher dimensional data, such as three-dimensional data. More efficient data storage and retrieval algorithms for these types of data should be investigated.

IX. REFERENCES

[AERIAL] Aerial Photographs and Satellite Images, United States Geological Survey, URL: http://mapping.usgs.gov/mac/isb/pubs/booklets/aerial/aerial.html

[BENT75] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associated Searching", *Communications of the ACM*, pp. 509-517, 1975.

[CHEN00] Chen, S., Rishe, N., Wang, X. and Weiss, M.A. "A User-Friendly Multimeida System for Querying and Visualizing of Geographic Data." Unpublished Manuscript, Florida International University, 2000.

[CODD70] E. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* Vol. 13, No. 6, June 1970.

[DAVI99] Debra Davis-Chu, Elma Alvarez, and Naphtali Rishe, "The Creation of a System for 3D Satellite and Terrain Imagery." Proc. *Thirteenth International Conference on Applied Geological Remote Sensing*, Vancouver, BC, Canada, 1-3 March 1999, vol. 2, pp. 329-336.

[EGEN92] M. Egenhofer, "Why not SQL!" In *International Journal on Geographical Information Systems*, 6(2), p. 71-85, 1992.

[FINK74] R. Finkey, J. Bentley, "Quadtrees: A data structure for retrieval on Composite Keys", pp. 57-97, 1974.

[GNIS] USGS mapping Information: Geographic Names Information System (GNIS), URL: http://mapping.usgs.gov/www/gnis/

[GUT94] R.H. Guting, "An introduction to spatial database systems", *VLDB Journal*, 3(4), p. 357-400, 1994.

[GUTT84] A. Guttman, "R-trees: A dynamic index structure for spatial searching", In *Proceedings ACM SIGACT-SIGMOD Conference on the Principles of Database Systems*, p. 569-592, 1984.

[IKONOS] Space Imaging Catalog of Products and Services, September, 1999.

[KEAT87] 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

[KNUTH73] D.E. Knuth, "The art of computer programming, volume 3, Sorting and Searching", Addison-Wesley, Reading, MA, 1973.

[LAND7] Landsat Project Policy and History: Landsat 7 Mission Specifications, NASA Goddard Space Flight Center, URL: http://ltpwww.gsfc.nasa.gov/LANDSAT/CAMPAIGN_DOCS/PROJECT/L7_Specifications.html

[LEE77] D.T. Lee and F.P. Preparata, "An improved algorithms for the rectangle enclosure problem", *SIAM Journal on Computing*, 6(3), pp. 594-606,1977.

[LEE80] D.T. Lee and B.J. Shacter, "Two algorithms for constructing a Delaunay Triangulation, *International Journal of Computer Information Sciences*, 3, pp. 219-242,1980.

[LOCK79] P.C. Lockemann, H.C. W.H. Weil, W H. Wohlleber, "Data Abstraction for Database Systems." *ACM Transactions on Database Systems*, 4(1), 1979.

[LORIE84] R.A. Lorie and A. Meier. Using a Relational DBMS for Geographical Databases. Geo-Processing, vol. 2, 1984, pp. 243- 257.

[MCFAR99] Gregory McFarland, Andres Rudmik, and David Lange, "Object-Oriented Database Management Systems Revisited", Technical Report for Air Force Research Laboratory - Information Directorate, Modus Operandi, Indialantic, FL, 1999.

[MNUSH] D. Mnushkin, "Object Oriented Databases", URL: http://spuds.cpsc.ucalgary.ca/courses/547-96/tamw/547/oot/#OODB.

[MUFF87] G. Muffin, "Raster versus Vector Data Encoding and Handling: A Commentary", *Photogrammetric Engineering and Remote Sensing*, Vol. 53, No. 10, pp.1397-1398, 1987

[RISHE92] N. Rishe, "A Database Design: The Semantic Modeling Approach", McGraw-Hill, 1992.

[RISHE94] N. Rishe, and Q. Li, "Storage of Spatial Data in Semantic Databases." In *Proceedings of the 1994 ASME International Computer in Engineering Conference*, Minneapolis, MN, pp. 793-800, Sept 11-14, 1994.

[RISHE98A] N. Rishe, D. Barton, M. Chekmasov, K. Madhyanapu, S. Graham, and M. Chekmasova, "Everglades Data Integration using a Semantic Database System." *International Conference Geospatial Information in Agriculture and Forestry*, Lake Buena Vista, FL, pp. I-567 - I-573, June 1-3, 1998a.

[RISHE98B] N. Rishe, D. Barton, F. Urban, M. Chekmasov, M. Martinez, E. Alvarez, M. Gutierrez, and P. Pardo, "High Performance Database Management for

Earth Sciences." *NASA University Research Center Technical Conference*, Huntsville, AL., pp. 539-544, Feb 22-25, 1998b.

[ROUS95] N. Roussopoulos, C. Faloutsos, and T. Sellis, "Nearest Neighbor Queries", In *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 71-79, 1995.

[SOL98] Selena Sol, "Introduction to Databases for the Web", 1998, URL: http://wdvl.com/Authoring/DB/Intro/index3.html.

[SAM90a] H. Samet, "The design and analysis of spatial data structures", Addison-Wesley, Reading, MA,1990.

[SAM90b] H. Samet, "Applications of spatial data structures", Addison-Wesley, Reading, MA, 1990.

[Tiger] Tiger Overview, United States Census Bureau, Tiger/Line data, URL: http://www.census.gov/geo/www/tiger/overview.html

[USINT] Understanding Color-Infrared Photographs. U.S. Department of Interior, U.S. Geological Survey.

[WAUGH] T.C. Waugh, and R.G. Healey, "The GEOVIEW Design: A Relational Database Approach to Geographical Data Handling", International Journal of Geographical Informal Systems, Vol. 1, No. 2, pp. 101-118, 1987.