A High-Performance Web-Based System Design for Spatial Data Accesses[*]

Shu-Ching Chen
School of Computer Science
Florida International University
chens@cs.fiu.edu

Xinran Wang
School of Computer Science
Florida International University
wang01@cs.fiu.edu

Naphtali Rishe
School of Computer Science
Florida International University
rishen@cs.fiu.edu

Mark Allen Weiss[1]
School of Computer Science
Florida International University
weiss@cs.fiu.edu

Abstract

With the increasing use of geographical data in real-world applications, Geographic Information Systems (GISs) have recently emerged as a fruitful area for research. Nowadays, a GIS can be combined with World Wide Web (WWW) techniques to provide information to a multitude of users. A high-performance web-based GIS, called TerraFly, has been developed in order to provide web-based GIS accesses to the general public. The design of TerraFly considers three major aspects including system architecture, data structures, and networking. The system architecture utilizes the existing resources to achieve maximum performance by using the "Internally Distributed Multithreading (IDMT)" technique. A spatial access method, semantic R-trees, is used to search an object based on both spatial and semantic information. System performance results are presented and analyzed. Reducing network traffic to achieve faster response to users is also discussed.

Keywords: GIS, Internally Distributed Multithreading (IDMT), Semantic R-tree.

[*]This research was supported in part by NASA (under grants NAGW-4080, NAG-5095, NAS5-97222, and NAG5-6830), NSF (CDA-9711582, IRI-9409661, HRD-9707076, and ANI-9876409), ARO (DAAH04-96-1-0049 and DAAH04-96-1-0278), AFRL (F30602-98-C-0037), BMDO (F49620-98-1-0130 and DAAH04-0024), DoI (CA-5280-4-9044), and State of Florida.

[1] Mark Allen Weiss was supported in part by NSF (EIA-9906600).

1. INTRODUCTION

The use of Geographic Information Systems with spatial data is becoming more and more popular nowadays. Dramatically increased availability and usage of remotely sensed data require advanced technologies in computer science and related areas. With the exponential growth of the World Wide Web (WWW), there are more domains open to GIS applications. A major consideration is to make a GIS system accessible to the general public, who have little knowledge of the spatial data, and allow them to interact with the system to manipulate and retrieve information they need. In order to address these issues, we developed a user-friendly web-based multimedia system, called TerraFly [2] The TerraFly system lets remote users using different platforms interact with the system, exploit spatial data of their interest, and issue queries to retrieve information they need, without the need for extensive training with the system, thus avoiding a painful experience.

Numerous research works of GIS applications, both academic and commercial, have been performed for decades [4][6][7][8] and [9]. The existing research is extensive and advanced and focuses on the following areas:

1) Database systems to efficiently store and retrieve heterogeneous spatial data;
2) Spatial data analysis capabilities;
3) Spatial data indexing methods.

Even though there have been great achievements over the past decades on these areas, a web-based GIS application still suffers from the following drawbacks:

1) Inability of databases to efficiently handle large and different spatial data sets;
2) Tendency for complicated WWW technologies and distributed computing to add complexity to the system;
3) Significant degradation of system performance in many misconfigured GIS systems.

In this paper, detailed research on the methodology and techniques used to improve performance for a web-based GIS application are discussed. Overall performance of a web-based GIS system depends on the design of the system architecture. For a distributed GIS system, a desired design is to best utilize existing resources to obtain maximum performance. To achieve this goal, we designed a model called "Internally Distributed Multithreading Model (IDMT)." In this model, componentization and distribution of threads are based on the different functionality each thread may have, enabling the system to better utilize server CPU and other resources.

In a GIS system, users usually request specific information, such as "*Find the nearest airport.*" This kind of information is very important for a GIS system, but is not used at all in constructing a spatial data structure such as an R-tree. A data structure called a Semantic R-tree that contains semantic information is proposed in this paper. This data structure provides significant savings in response time. Our experimental results show that the Semantic R-tree outperforms the well-known R-trees in answering specific information that users often request.

This paper is organized as follows. In Section 2, we present our system architecture as a whole. The information server system structure is discussed in Section 3, and the Semantic R-tree is discussed in Section 4. Section 5 concludes the paper.

2. SYSTEM ARCHITECTURE

The system's design is based on a three-tier client/server architecture as shown in Fig-1. The second tier handles all application logic: namely, it retrieves data requested by the clients and answer queries for the clients. Java is used in developing the client side to generate byte codes running across different platforms. Each client is a data-less graphical user interface (GUI). The database server stores and retrieves multi-dimensional spatial data such as image data as well as alphanumeric data. Together, the client/server architecture forms a complete computing system with a distinct division of responsibility.

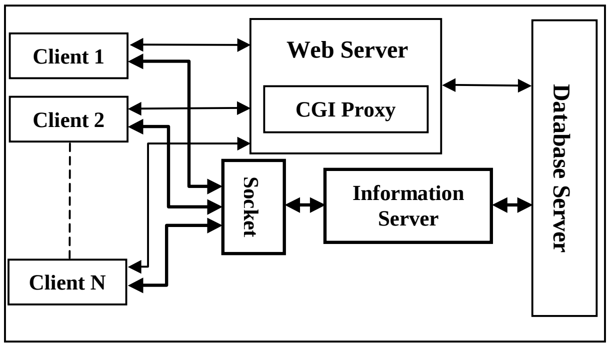

Fig-1: TerraFly system architecture

Java clients use a connection wrapper to synchronize data transmission. The clients send requests to either a web server or an information server, and receive data (image data and textual data) from different servers. When the web server receives a request from a client, the proxy server will parse the request, retrieve image and textual data related to the fly-over request from the database server, and send the data back to the client. The information server will search a spatial data structure (a semantic R-tree), to retrieve information related to range queries and nearest neighbor queries [15] from the database server, and send the answers to the client.

2.1 Java client

The client is written using Java to achieve platform independence. A snapshot of the client GUI is shown in Fig-2. The main features of the client include:

1. Capability to fly over the Landsat TM data, Digital Orthophoto Quad (DOQ) at different directions by positioning the mouse within the image.
2. Customized three-band (sensor) combination: users can select some predefined and useful three-sensor combinations to view false color images from a drop down menu.
3. Advanced three-band color composite: this application allows scientific users to enter any three-band combinations (RGB) that the user is interested in studying or analyzing.
4. RGB intensity control: this option allows the users to increase or reduce the intensity of any of the bands that represent the colors.
5. Capability to issue range queries and nearest-neighbor queries.
6. Capability to obtain feature extraction of the image.
7. Capability to display online information (latitude, longitude, regions, etc.) of the images that users are viewing.

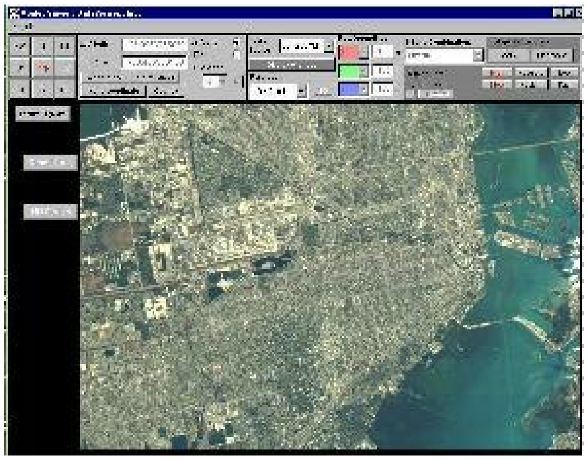

Fig-2: TerraFly Client

2.2 Database server

The database server contains a multimedia spatial database system built by our group using the Semantic Object-Oriented Database Management System (Sem-ODB) [13] [14] based on Semantic Binary Object-Oriented Model [12]. In the Semantic Binary Object-Oriented Model, information is represented by logical association (*relations*) between pairs of objects and by the classification of objects into *categories*. Unlike the traditional database systems that consist only of alphanumeric data, the Sem-ODB not only has alphanumeric data, but also has data that cover multi-dimensional spaces such as image data (maps). Currently, the database contains semantic/textual, spatial/remote sensed (Landsat) and digital data including Digital Orthophoto Quad (DOQ) (Arial photograph) data. The Sem-ODB provides an efficient data storage and manipulation methodology.

2.3 Proxy server

A proxy server is used to relay data requested by the clients to the database server, and to transmit the data retrieved by Sem-ODB back to the clients. This proxy needs to use two different protocols, one to interface with the clients, and another to interface with the Sem-ODB server. We use Common Gateway Interface (CGI) as the first protocol and Sem-ODB's APIs as the second protocol. Upon receiving a request from a client, the proxy server must decode the request and retrieve approximate data from the database server. The proxy process accomplishes this by using Sem-ODB APIs. What the proxy server needs to do is to find where the database server is located. After the database connection is established, all queries can be performed using APIs provided by the Sem-ODB.

2.4 Information server

The system as a whole provides an integrated view of spatial and associated data along with the capability to display and manipulate spatial images. Also, being a GIS system, it provides the capability to issue range queries and nearest neighbor queries to satisfy particular interests of scientists and public users. Range queries are used to find spatial objects in a specific area around a location specified by the user. A sample query is "*Find all rental car companies around Miami International Airport within six miles*." Nearest neighbor queries are used to find the nearest spatial object to the object that the user specifies. For instance, "*Find the nearest car rental company*." The information server is designed for answering these two types of queries. The information server is a multithreaded application. It receives requests from the clients through a UNIX socket. The configuration of the information server is shown in Fig-3, which is discussed in detail in the following section.

3. INFORMANTION SERVER

In the TerraFly Information Server, we use POSIX threads [11] to enhance performance. The server is based on the Internally Distributed Multithreading Model (IDMT) that will be described next.

3.1 Internally distributed multithreading (IDMT) model

In order to improve system performance, we propose an internally distributed multithreading (IDMT) model. Fig-3 shows the proposed server structure. The backbone of this structure is a thread pool containing a number of threads that do computation in the back end, and we call these working threads the back-end threads. Their counterparts are the front-end threads, which are dedicated to communicate with the clients. We will see that this structure not only achieves better performance than the general model, but that it also has better scalability property.

3.2 Analysis of internally distributed multithreading

As mentioned earlier, the front-end communication threads are dedicated to communicating with the clients, and do first-phase computations, such as converting data types and getting query values. All requests from the client-side are sent to the front-end threads through certain communication channels (we use sockets in this GIS system). When these threads receive the requests, they analyze the requests based on the agreed-to protocols between the server and the client, and post the analyzed requests to the job queue, where the back-end threads take over the job.

We now describe how this distribution improves the server in three ways: throughput, internal load balancing, and scalability.

Throughput: If a client sends multiple requests, the response time in the IDMT model is much less because the server distributes the requests to back-end threads that run concurrently to solve the requests from the same client. The best case is that these threads compute almost simultaneously to get the results. Heavy load is distributed to several threads, making the system more robust and achieving a better response time.

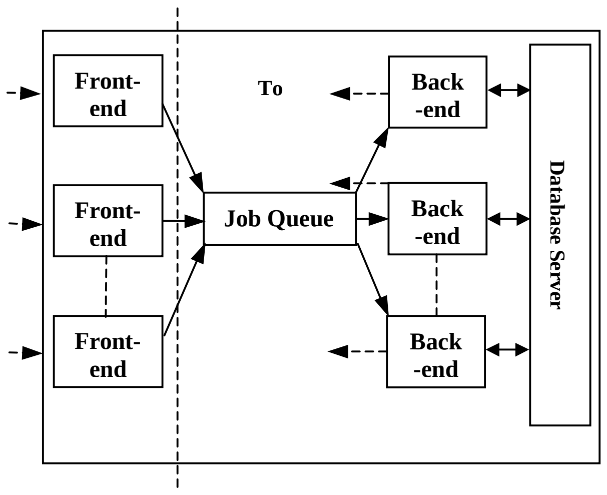

Fig-3: The Internally Distributed Multithreading Design

Internal load balancing: The back-end threads in the server thread pool perform the major computations. When the server detects a heavy load, such as too many requests from the clients, it determines that there are not enough back-end threads to handle the situation. The server can then generate some extra back-end threads on demand to relieve the heavy load. This dynamic thread generation can balance the server performance with a minimal cost of resources needed.

Scalability: When we separate a large process into several upgradeable components, we actually gain a benefit of scalability. Each component is independent from others and has its own functionality. If one component is upgraded, there is little impact on the other components. In our design, the communication module is separated from other modules. If another communication method is used, there is no effect on the server computation at all. The computation module consists of a number of back-end threads, which can actually be distributed to different powerful workstations. Thus, updating will have the minimal impact on the whole system.

4. SEMANTIC R-TREE FOR SEMANTIC QUERIES

An R-tree is an extension of B-trees for multidimensional objects. An object in an R-tree is represented by its minimum bounding rectangle (MBR). Details of R-trees are discussed in [3]. There are various R-tree variants, such as the R+ tree [17], R* tree [1], and the packed R-tree [10][16]. An R-tree is based on heuristic optimization to minimize the area of each enclosing rectangle in the inner nodes. Most of the research on the R-trees focuses on minimizing overlapping MBRs and optimizing storage utilization. However, for a GIS system, more specific research on data structures is needed to answer all kinds of range queries, which are one significant feature of a GIS system. The R-tree and its variants are very efficient methods that support range queries. We can further improve the R-trees (or other spatial data structure) for a GIS system, especially a GIS with static data.

4.1 Semantic query

Semantic queries are used to find specified objects according to their relations to the base object. An example semantic query is "*Find a nearest rental car company around Miami International Airport (MIA).*" In this query, the base object is "*MIA*," the specified object (i.e., the semantic object) is "*a rental car company,*" and the relationship is "*nearest to.*" On the other hand, a query such as "*Find an object nearest to MIA,*" where no attribute of this object is specified, is a general query. In comparing these two queries, a semantic object is defined as an object with some information that a user asks for. In the above example, a user wants to find a rental car company. This "*rental car company*" is one type of semantic information a user may specify. If a spatial data structure can be built with this type of information, query performance can be greatly enhanced.

The difference between the semantic queries and the general queries is subtle but significant. A general query formula:

$$Q = \underset{i=1}{\overset{K}{G}} (\mathrm{O}_i \ \mathrm{R}_i \ \mathrm{OB}_i) \qquad \text{where}$$

- G is a set operator that is either Union ($\cup$) or Intersect ($\cap$),
- K is the number of union and/or intersect operations,
- O_i is a set of objects to be found,
- OB_i is a base object,
- R_i is a relation.

A semantic object is O^s=O(s), where s is the specified information (s $\in$ S_j). For each GIS system, there are a number of semantic subsets. Let S_j be one semantic subset in a GIS semantic set S ($\mathrm{S}_j \subseteq S$) and H be the number of semantic subsets in S, where $S = \bigcup_{j=1}^{H} (\mathrm{S}_j)$. A semantic query is

$$Q_s = \underset{i=1}{\overset{K}{G}} (\mathrm{O}_i(\mathrm{s}) \ \mathrm{R}_i \ \mathrm{OB}_i)$$

A user uses a semantic query Q_s to ask for information he/she wants. For each semantic subset S_j, there are a limited (fixed) number of elements. For example, in a map GIS system, S = {S_1, S_2, S_3}, where

- S_1 ={Dade county, Broward County, Orange county…} by county;
- S_2 ={river, bridge, route…} by type;
- S_3 ={school, building, shopping center…} by category.

The semantic subset information can be used to construct a spatial data structure to answer the semantic queries more efficiently. If a user specifies that he wants to find a rental car company nearest to him, the server will search the data structure not only by the spatial specifications, but also by this semantic information to find a car rental company. In order to optimize (at least partially) a data structure based on the semantic subsets, these subsets first need to be found. In actuality, this is not a problem at all. Finding semantic subsets is a practical issue of searching the data sets to identify different properties, and then categorizing these properties into different subsets. Some unnecessary subsets can be further pruned according to the features of each GIS system so that only a few subsets need to be taken into consideration.

4.2 Semantic R-trees

A spatial data structure with built-in semantic information is better able to answer semantic queries. For this purpose, a spatial data structure with built-in semantic information, called a semantic R-tree, is proposed. Without such built-in semantic information, a spatial data structure has difficulty in answering a query such as "*Find all schools within 20 miles*" efficiently. Searching the semantic R-tree will get all the objects within 20 miles, and further processing is then required to get the desired objects (i.e., the schools) in answer to this query. With built-in semantic information, some sub-trees containing unrelated information can be pruned, which makes semantic searching quite efficient. In our design, the "*category*" subset

```
Semantic-packing Algorithm:

Step 1: Select one semantic subset that is best for one GIS
        system.
Step 2: Categorize data according to the selected semantic
        subset.
Step 3: Sort data in each category based on the x or y
        coordinate.
Step 4: /* create the child node at level $l$ (leaf nodes are at
        level 0) */

        While (there are data in the sorted list)
            Generate a new R-tree node.
            If (the remaining $P$ data in the same category
                with $P >= M$)
                    Assign the next $M$ data into this node.
            Else
                    Assign next $P$ data into this node.
Step 5: /* create the parent node, at level ($l$ + 1) */

        While (there are nodes at level $l$ not sorted yet)
             Sort the nodes at level $l$ based on their
             generation times.
        Go to Step 4.
```

Fig-4: Pseudo-code of Semantic-packing algorithm

is used to build a semantic R-tree and the algorithm used is based on R-tree's packing technique. For each node, its semantic information is categorized. Then, in each category, the MBRs are sorted on the x or y coordinate of one of the corners of the rectangle. Sorting MBRs is similar to the method proposed by Roussopoulos and Leifker [16]. In each category, the sorted list of rectangles is scanned and assigned to one R-tree leaf node until this leaf node is full or there is no data left for this category. The algorithm is shown in Fig-4. Let *M* be the maximum number of entries of one node.

Because the semantic information is packed into an R-tree, there might be some underflow nodes (less than *M*/2 children). However, since only a fixed number of elements exist in one semantic subset (usually this number is small), there might be only a few underflow nodes. A semantic R-tree is also packed in the first place, so it may have a better space utilization than a non-packed R-tree. Therefore, better performance for the semantic query can be achieved.

- **Observation 1:** Even if a semantic R-tree is partially optimized by semantic information, there are still many cases where a semantic R-tree outperforms an R-tree in answering a general query. If in some cases, the performance of a semantic R-tree degenerates in answering a general query, the degeneration is minimal.
- **Observation 2:** A semantic R-tree data structure has better performance in answering semantic queries.

4.3 Experimental results

The semantic R-tree was implemented and several experiments were conducted to demonstrate the capability and usefulness of the proposed semantic approach, applied to GIS semantic queries. Real data from the GNIS [5] was used. The tested data represents a map of Florida consisting of 32,108 segments. The page size for data and the directory storage pages was chosen to be 1,024 bytes. From the chosen page size, the maximum number of entries in the directory pages was 40.

We compare a semantic R-tree with an R-tree that uses quadratic split algorithm. The quadratic split algorithm is chosen because there is no essential performance gain resulting from the linear split algorithm [3]. The performance metric used is the response time of both general queries and semantic queries. In the general queries, the degree of degeneration of a semantic R-tree in comparison to an R-tree, if any, can be obtained. For the semantic queries, experiments were conducted to show how well the system is able to answer semantic queries using the semantic R-tree.

The sample range sizes of the rectangles were large size (200 miles) and small size (20 miles) in the GNIS file. The sample areas chosen were northwest (NW) region, northeast (NE) region, southwest (SW) region, southeast (SE) region, and center (CEN) region in the data set. The number of nodes tested ranged from 5,000 to 30,000.

Fig-5 and Fig-6 show the experimental results of the general queries. The Y-axis represents the ratio value, which is (response time using an R-tree)/(response time using a semantic R-tree). The critical value is one, and if this ratio is greater than one, it means that it takes more time searching the R-tree to find results than searching the semantic R-tree. The X-axis is the number of nodes. For the queries with a large rectangle size, the semantic R-tree outperforms an R-tree for some of the queries. When an R-tree performs better than a semantic R-tree, most of results show that the relative response time is very close to one (as shown in Fig-5), which supports our first observation in Section 4.2. Similar experimental results are obtained for small rectangle size (as shown in Fig-6). Experimental results for semantic queries are shown in Fig-7, where the X-axis is the number of nodes and the Y-axis represents the ratio value, which is (response time using an R-tree)/(response time using a semantic R-tree). The critical value is one. The results support the second observation in Section 4.2: the semantic R-tree performs much better than the R-tree. If no objects are found for the queries, the relative response time may be higher than one.

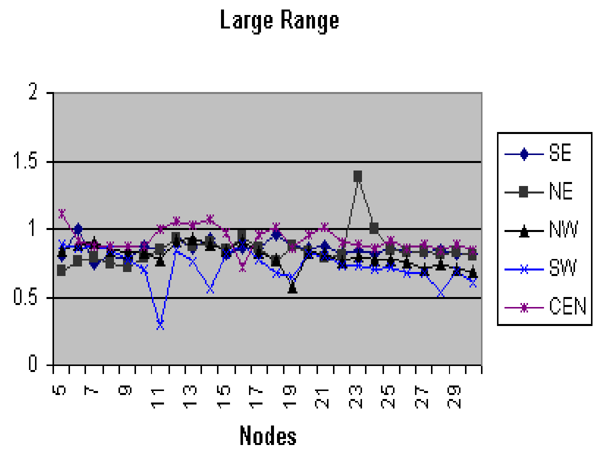

Fig-5: Test results for Large range.

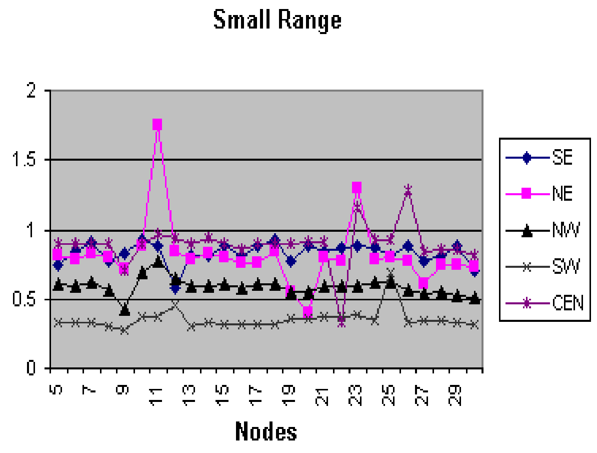

Fig-6: Range query result with small size range

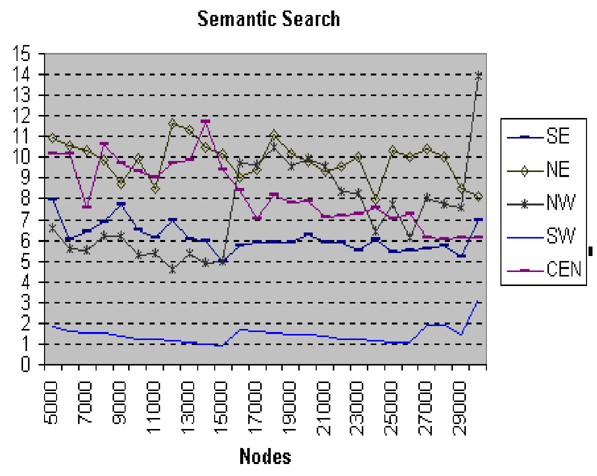

Fig-7: Results of finding all objects with category "Bank".

All these experimental results show that the semantic R-tree is a very efficient method for answering semantic queries. As expected, the performance gain of the semantic R-tree over an R-tree in answering semantic queries is considerable. In many GIS applications, the support for semantic queries is more desirable since this kind of query can provide information that users often request. Hence, a semantic R-tree is more suitable in such cases.

5. Conclusion

In this paper, a GIS system called TerraFly is introduced. The TerraFly system is a multimedia application that allows users to view images, manipulate the retrieved data, and issue range queries and nearest neighbor queries. A spatial access method, the semantic R-tree, is used to search the objects based on both the spatial and semantic information. The TerraFly information server uses a technique called "Internally Distributed Multithreading Model (IDMT)" to achieve better performance. A semantic object-oriented database management system is developed to meet the database requirements. Spatial data such as the maps can be stored and retrieved from this database. Several experiments were conducted to compare the semantic R-tree with the R-tree based on general queries and semantic queries. The experimental results show that the semantic R-trees perform better than the R-trees for semantic queries, and have similar performance for general queries.

6. References

[1] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 322-331, 1990.

[2] Shu-Ching Chen, Naphtali Rishe, Xinran Wang, and Mark Allen Weiss, "A User-Friendly Multimedia System for Querying and Visualizing of Geographic Data," In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI 2000)*, 2000, Vol II, Information Systems Development: pp.689-694, July 23-26, 2000, Orlando, Florida.

[3] Guttman, "R-tree: A Dynamic Index Structure for Spatial Search," in *Proc. ACM SIGMOD*, pp. 47-57, June 1984.

[4] http://elib.cs.berkeley.edu.

[5] http://mapping. usgs.gov/www/gnis/gnisftp.html.

[6] http://www.envi-sw.com.

[7] http://www.erdas.com.

[8] http://www.esri.com.

[9] http://www.gvu.gatech.edu/gvu/virtual/VGIS/.

[10] Ibrahim Kamel, Christos Faloutsos, "On Packing R-trees," *CIKM*, pp. 490-499, 1993.

[11] Bill Lewis, Daniel J. Berg, *Multithreaded Programming With Pthreads*, Sun Microsystems Press, 1998.

[12] Naphtali Rishe, *A Semantic Approach to Database Design: the Semantic Modeling Approach*, McGraw-Hill, 1992.

[13] N. Rishe, A. Vaschillo, D. Vasilevsky, A. Shaposhnikov, and S.-C. Chen, "A Benchmarking Technique for DBMS`s with Advanced Data Models," to appear in *ACM SIGMOD ADBIS-DASFAA Symposium on Advances in Databases and Information Systems*, September 2000.

[14] N. Rishe, J. Yuan, R. Athauda, S.-C. Chen, X. Lu, X. Ma, A. Vaschillo, A. Shaposhnikov, and D. Vasilevsky, "SemanticAccess: Semantic Interface for Querying Databases," to appear in *the International Conference on Very Large Databases (VLDB)*, September 2000.

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

[16] N. Roussopoulos and D. Leifker, "Direct Spatial Search on Pictorial Database Using Packed R-trees," *Proc. ACM SIGMOD*, May 1985.

[17] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos, "The R+-tree: A Dynamic Index for Multi-Dimentional Objects," *Proc. 13th Int'l Conf. on Very Large Data Bases*, pp. 507-518, 1987.